Part IV ECHONET Basic API Specification

Release information (as of August 7th 2001)

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

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

Contents

Chapter 1 Overview

1.1 Basic Concept

To implement the ease of development and implantation of Application Software in ECHONET, the API (Application Programming Interface) is specified as the Basic API (API for using the ECHONET Communication Middleware functions described in Part 2) and service API (API for using the service middleware described in Part 8). Part 4 describes the specifications of the former or Basic API .

The Basic API is intended 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 manipulating the 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 permits requesting an object service to an object of another device, receiving a response to it, receiving an object service requested from another device, and transmitting a response to it after processing on the self side.

The Basic API specification is provided in consideration 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 where a concrete language has been specified. Regarding the input/output data items, the detailed specification is indicated as “Level 1 ECHONET Basic API Specification”. Regarding functions, the detailed specification is indicated as “Level 2 ECHONET Basic API Specification”.

1.2 Positioning on Communication Layers

Fig. 1.1 shows the positioning of the Basic API on the communication layer. The ECHONET communications processing block performs communication protocol processing, information maintenance for communication protocol processing, and various information control 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 for the Application Software to use this ECHONET communications processing block.

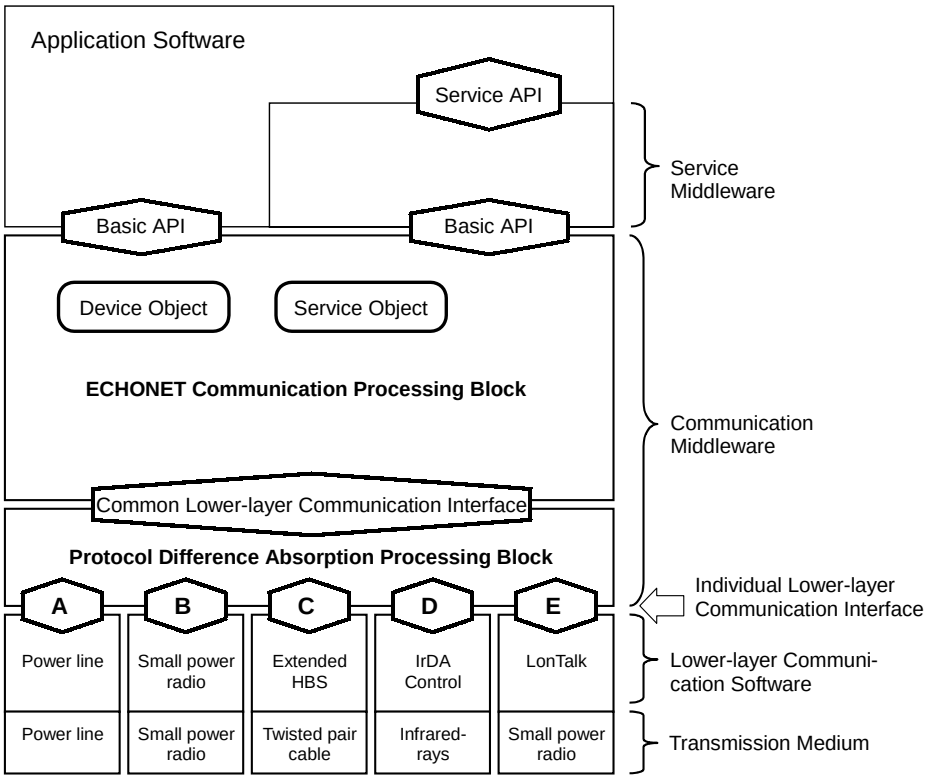

Fig. 1.1 Positioning of Basic API on the Communication Layer Configuration

Chapter 2 ECHONET Basic API Function Specification

2.1 List of ECHONET Basic API Functions

In the standard for ECHONET Communication Middleware (see detailed specification 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 (API). The ECHONET objects to be operated can be classified into the following eight types from the standpoint of the application developer. The following classification is based on the viewpoint that the Application Software operates objects:

(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)	Another 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)	Another 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)	Another 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 communications middleware and to obtain status or information through ECHONET.
(8)	Another 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 setting. This function is based on middleware operations.

These ECHONET objects to be operated 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 certain knowledge of 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 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 block needed to support this 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 specification of each API is 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 resetting	Requests reset processing for communications processing block in ECHONET Communication Middleware.	
5	Request for suspension	Requests suspend operation for communications processing block in ECHONET Communication Middleware.	
6	Request for operation restart	Requests restart operation for communications processing block in ECHONET Communication Middleware.	
7	Self-node profile object operation	Sets and gets property values of the profile object of the self-node, and notifies other nodes.	
8	Another node profile object operation	Gets property values of profile object of another node.	
9	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.	
10	Another node device object operation	Sets property values of self-node device object and gets property values.	
11	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.	
12	Another node communication definition object operation	Sets and gets property values of communication definition object of another node.	
13	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.	
14	Another node service object operation	Sets and gets property values of service object of another node.	
15	Addition or deletion of control object	Adds or deletes objects under control of ECHONET communication block in units of property.	

2.2 ECHONET Basic API Function Specification

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 state transition is mainly described. For the status underlined in the description, see Part 2, “8.2 ECHONET Communication 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 communications software using the specified information. After execution of initialization, the ECHONET communications processing block is put into a start waiting 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 waiting 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. (No special operation stop takes place.)

(4) Request for resetting

Requests resetting of software related to communications under ECHONET Communication Middleware. Upon receiving this request, the ECHONET communications processing block executes reset processing for the protocol difference absorption processing block and all lower-layer communications software if the request relates to the resetting of the ECHONET communications processing block proper, and then waits in start waiting status. If the request relates to resetting of the discrete lower-layer communications software, said block executes reset processing for the software of the specified portion.

(5) 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 communications software, said block executes suspend processing only for the software of the specified portion.

(6) 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 the self-block.

(7) 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.

(8) Another node profile object operation

Sets property values of profile object of another node and obtains set values for ECHONET communications processing block. The ECHONET communications processing block accepts this API processing only in normal operation status.

(9) 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.

(10) Another 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.

(11) 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 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.

(12) Another 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 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.

(13) 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.

(14) Another 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 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.

(15) 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.

Chapter 3 Level 1 ECHONET Basic API Specification

3.1 List of Level 1 ECHONET Basic APIs

Table 3.1 shows a list of level 1 ECHONET Basic API s that the ECHONET Communication Middleware supports. In the ECHONET Basic API s of level 1, the API items shown in Table 3.1 include some of those shown Table 2.1, which are further classified. The mounted APIs conforming to this 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 divided into multiple data items further. 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 resetting	Requests reset processing for the communications processing block under ECHONET Communication Middleware.	Optional
5	Request for suspension	Requests to suspend the operation for the communications processing block under ECHONET Communication Middleware.	Optional
6	Request for operation restart	Requests to restart the operation for the communications processing block under ECHONET Communication Middleware.	Optional
7	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
8	Self-node profile object property value getting	Gets the information set as property values of the profile objects of the self-node.	Required
9	Another node profile object property value getting	Gets the information on property values of the profile object of another node.	Optional
10	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
11	Self-node device object property value getting	Gets the information set as property values of the device object of the self-node.	Optional
12	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

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

No.	API name	Function outline	Mounting specification
13	Another node device object property value getting	Gets the information on property values of the device object of another node.	Optional
14	Another 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
15	Another 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
16	Self-node communication definition object property value setting and notification	Sets or notifies the information on property values of the communication definition object of the self-node.	Optional
17	Self-node communication definition object property value getting	Gets the information set as property values of the communication definition object of the self-node.	Optional
18	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
19	Another node communication definition object property value getting	Gets the information on property values of the communication definition object of another node.	Optional
20	Another 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
21	Another node communication definition object property value request	Requests to set (a request for control) the information on property values of the communication definition object of another node.	Optional
22	Self-node service object property value setting and notification	Sets or notifies the information on property values of the service object of the self-node.	Optional
23	Self-node service object property value getting	Gets the information set as property values of the self-node service object of the self-node.	Optional
24	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
25	Another node service object property value getting	Gets the information of the property values of the service object of another node.	Optional
26	Another 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
27	Another node service object property value request	Requests to set (a request for control) the information on property values of the service object of another node.	Optional
28	Addition of control object	Adds an object under the control of the ECHONET communications processing block in units of property.	Optional
29	Deletion of control object	Deletes an object under the control of the ECHONET communications processing block in units of property.	Optional
30	Control object acquisition	Gets an object under the control of the ECHONET communications processing block in units of property.	Optional

3.2 Level 1 ECHONET Basic API Detail Specification

For each API shown in Table 3.1 in the previous section, data input and output 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 communications 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 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 communications 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 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. Identifying the ECHONET communications processing block, protocol difference absorption processing block, and individual lower-layer communications 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 input/output specifications.

Table 3.4 List of Fault Notice API Input/Output Data

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

(4) Request for resetting

Requests reset of software related to communications under ECHONET Communication Middleware. Upon receiving this request, the ECHONET communications processing block executes reset processing for the protocol difference absorption processing block and all lower-layer communications software if the request relates to the resetting of the ECHONET communications processing block proper, and waits in the initial status. If the request relates to the resetting of the discrete lower-layer communications software, said block executes reset processing only for the software of the specified portion. Table 3.5 shows input/output specifications.

Table 3.5 List of Resetting Request API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	device_id	Indicates a target to be reset. Identifying the ECHONET communications processing block, protocol difference absorption processing block, and individual lower-layer communications software shall be enabled.	Optional
Output	Return Value	TRUE: Resetting acceptable, FALSE: Resetting not acceptable	Optional

(5) 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” or “Request for resetting”, the ECHONET communications processing block shall not accept it from the Application Software or the protocol difference absorption processing block (and lower-layer communications software). When the request relates to the suspension of the protocol difference absorption processing block and the discrete lower-layer communications software, the ECHONET communications processing block executes only suspend processing for the software of the specified portion. Table 3.6 shows input/output specifications.

Table 3.6 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 communications software shall be enabled.	Optional
Output	Return Value	TRUE: Suspension acceptable, FALSE: Suspension not acceptable	Optional

(6) 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 3.7 shows input/output specifications.

Table 3.7 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 communications software shall be enabled.	Optional
Output	Return Value	TRUE: Success in restart, FLASE: Restart disabled (including failure)	Optional

(7) 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 communications 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 to be performed to set a property value (write a value) of the profile object on the ECHONET communications processing block. Notification is an operation to be performed to notify a property value of the profile object as data on ECHONET. Fig. 3.1 shows the relationship between this API and the ECHONET communications processing block, and Table 3.8 shows input/output specifications.

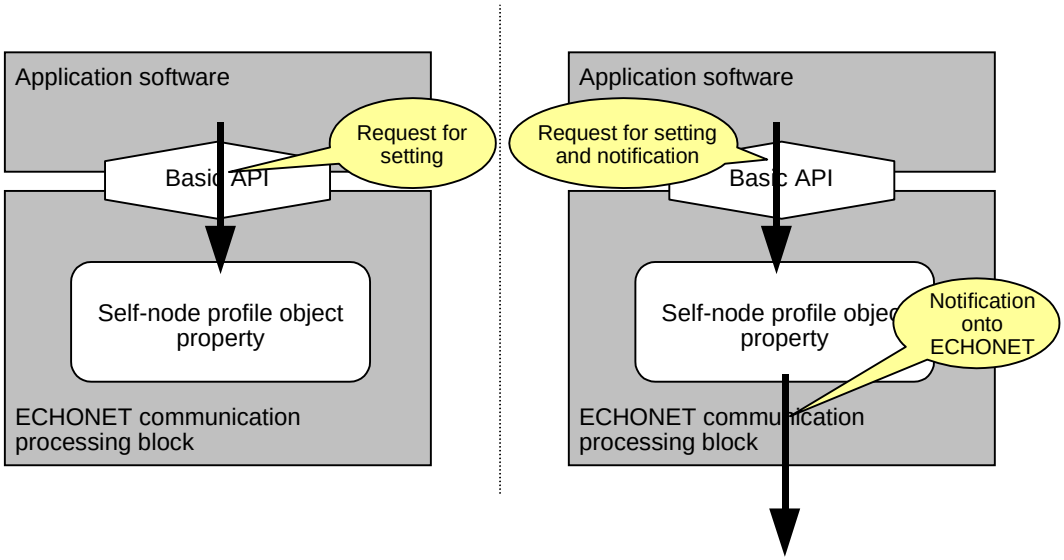

Fig. 3.1

Table 3.8 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 communications software are targets.	Required
Input	prop_id	Specifies a target property.	Required
Input	announce_info	Specifies whether 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

(8) 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 communications software profile object instance of the self-node.

Fig. 3.2 shows the relationship between this API and the ECHONET communications processing block, and Table 3.9 shows input/output specifications.

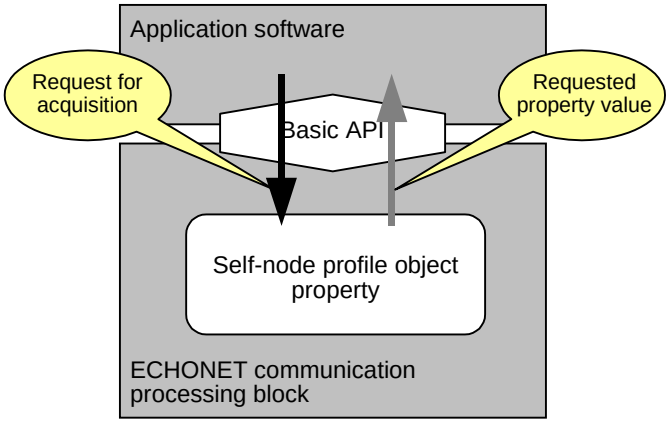

Fig. 3.2

Table 3.9 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 communications 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

(9) Another node profile object property value getting

Reads (gets) property values of the node profile class, router profile class, and individual lower-layer communications software profile class of another node for ECHONET communications processing block. This is classified into two cases, one in which getting of the values monitor-controlled on the communications middleware is requested (CASE 1 in the following figure) and another 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. The profile information is the information of the profile object property, such as initial setting information on the self-node EA and lower-layer communications software (see Part 2). Fig. 3.3 shows the relationship between this API and the ECHONET communications processing block, and Table 3.10 shows input/output specifications.

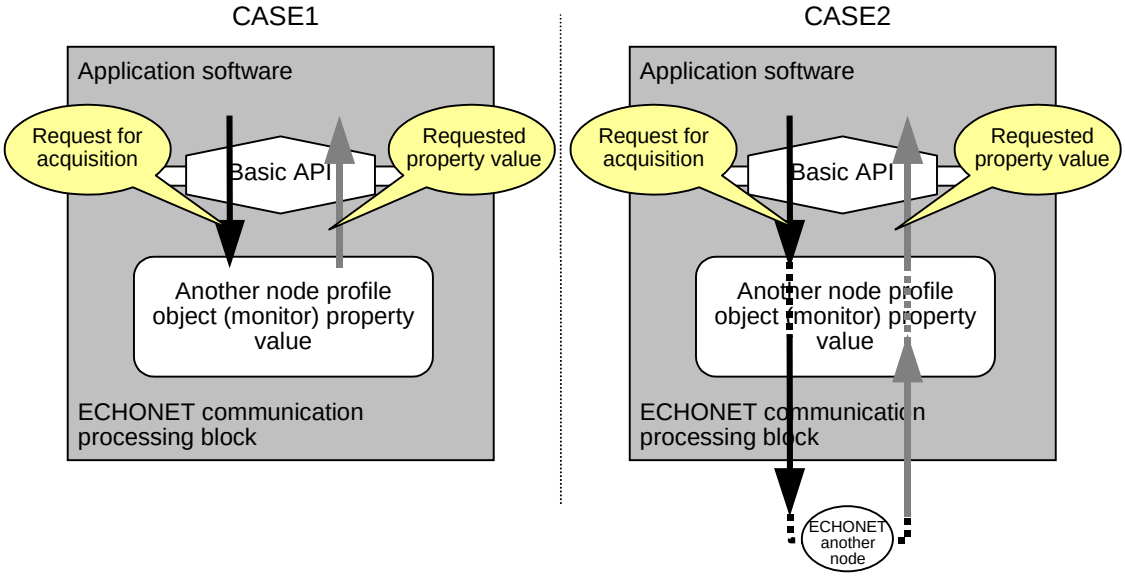

Fig. 3.3

Table 3.10 List of Another 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

(10) 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). Fig. 3.4 shows the relationship between this API and the ECHONET communications processing block. Table 3.11 shows input/output specifications.

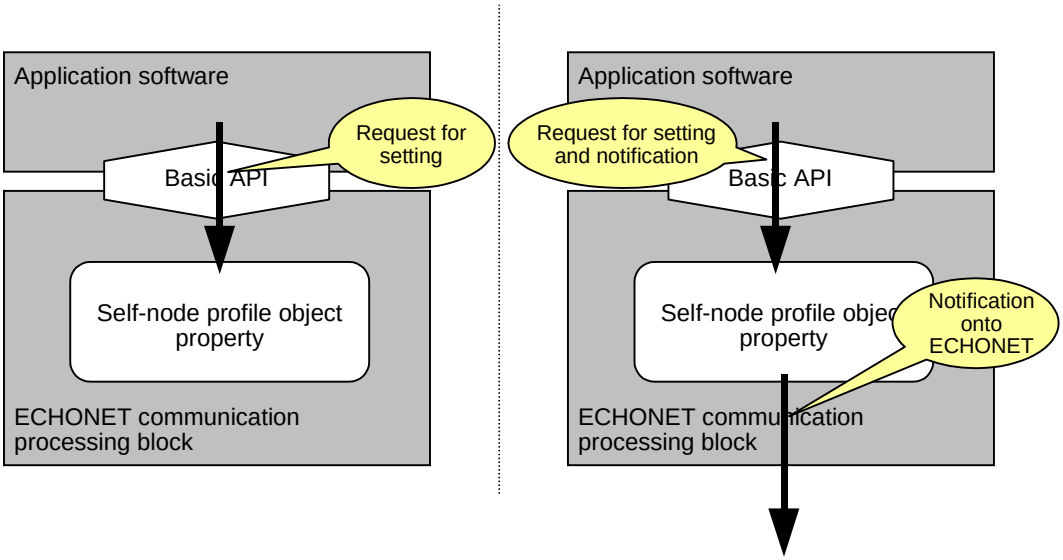

Fig. 3.4

Table 3.11 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 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

(11) Self-node device object property value getting

Reads (gets) property value of each device object instance of the self-node for ECHONET communications processing block. The target property items, contents, etc. to be read differ with the individual device object instance (see Part 2). Fig. 3.5 shows the relationship between this API and the ECHONET communications processing block. Table 3.12 shows input/output specifications.

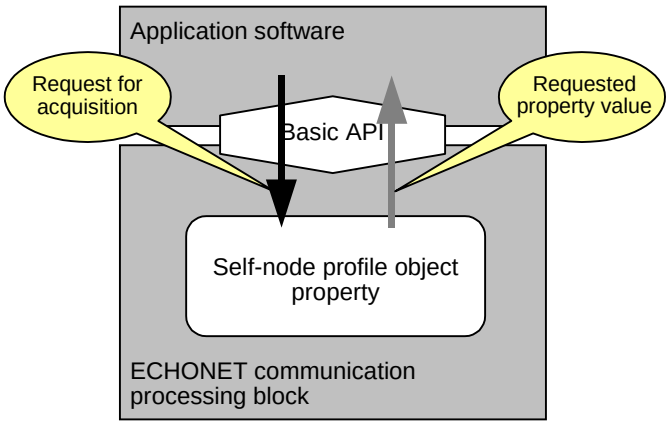

Fig. 3.5

Table 3.12 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

(12) 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 ECHONET communications processing block. The target property items, contents, etc. to be accepted from another 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 communications 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 communications middleware does not change property values without a request from the Application Software.) Fig. 3.6 shows the relationship between this API and the ECHONET communications processing block, and Table 3.13 shows input/output specifications.

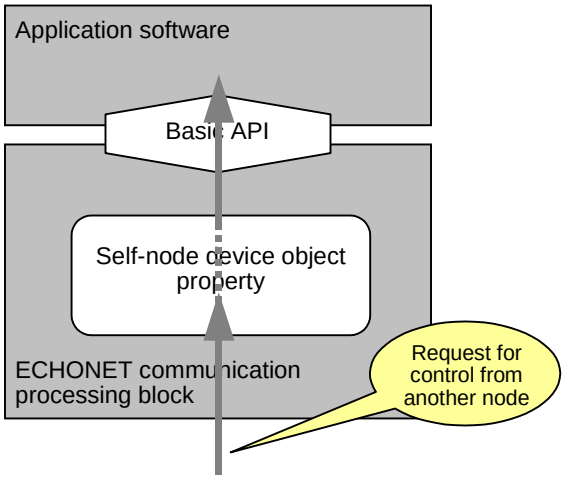

Fig. 3.6

Table 3.13 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

(13) Another node device object property value getting

Reads (gets) property value of each device object instance of another node for ECHONET communications processing block. This is classified into two cases, the first in which getting of the values monitor-controlled on the communications 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). Fig. 3.7 shows the relationship between this API and the ECHONET communications processing block, and Table 3.14 shows input/output specifications.

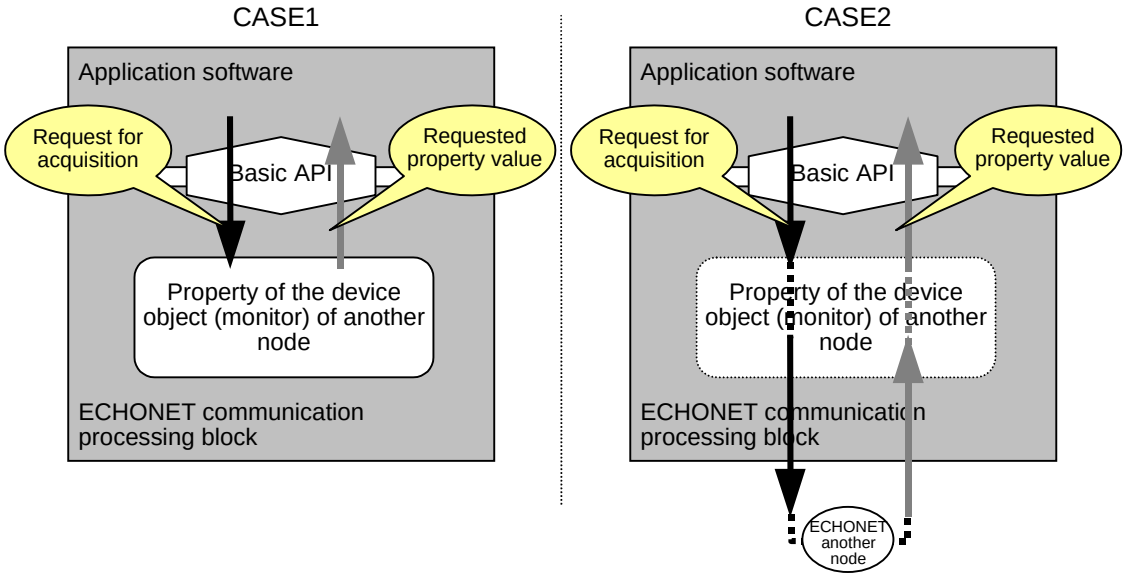

Fig. 3.7

Table 3.14 List of Another 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

(14) Another node device object property value notice acquisition

Reads (gets) the property value of each device object instance notified by another node for 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. Fig. 3.8 shows the relationship between this API and the ECHONET communications processing block, and Table 3.15 shows input/output specifications.

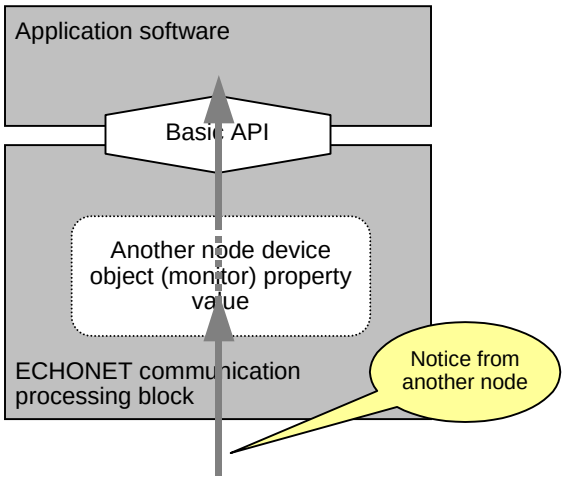

Fig. 3.8

Table 3.15 List of Another 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

(15) Another node device object property value setting request

Requests to set property value of each device object instance of another node for ECHONET communications processing block. Another node device object property value setting request is classified into two cases, the first in which a response of a setting request result is not required (CASE 1 in the following figure) and the second in which a 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). Fig. 3.9 shows the relationship between this API and the ECHONET communications processing block, and Table 3.16 shows input/output specifications.

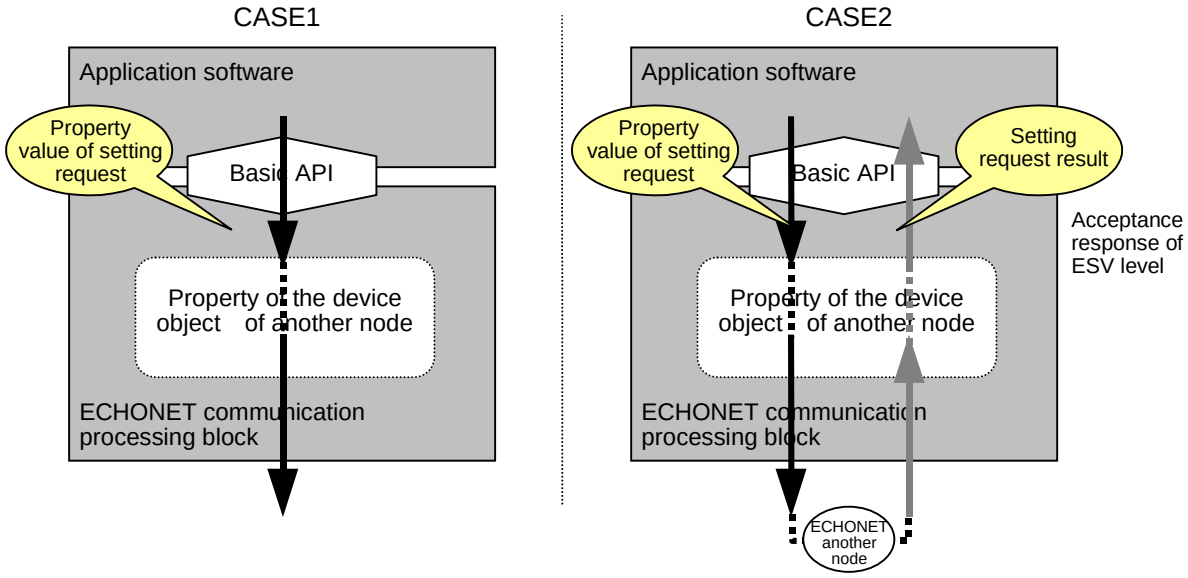

Fig. 3.9

Table 3.16 List of Another 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

(16) 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 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). Fig. 3.10 shows the relationship between this API and the ECHONET communications processing block, and Table 3.17 shows input/output specifications.

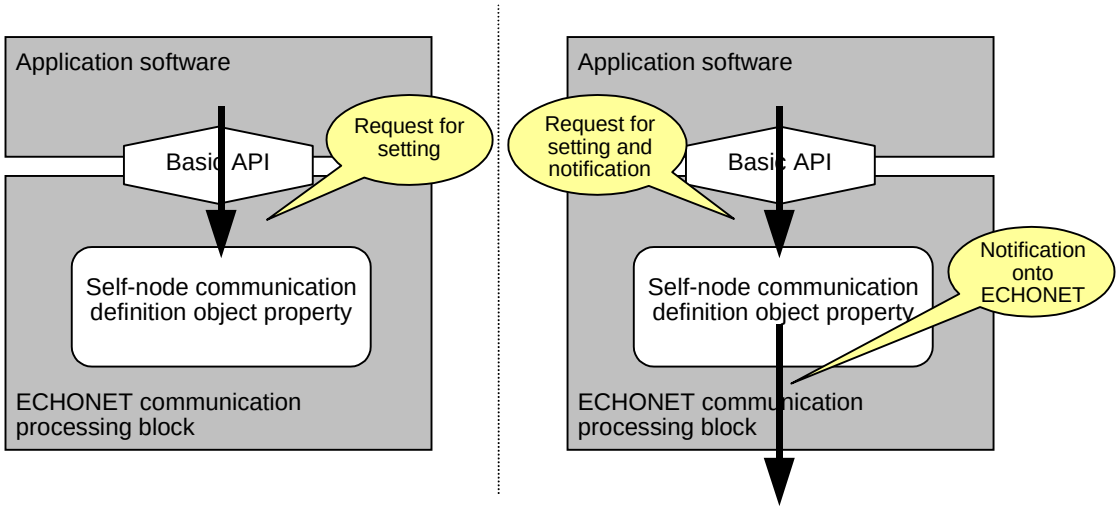

Fig. 3.10

Table 3.17 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 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

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

Reads (gets) property value of each communication definition object instance of self-node for ECHONET communications processing block. The target property items, contents, etc. for reading differ depending on the communication definition object instance (see Part 2). Fig. 3.11 shows the relationship between this API and the ECHONET communications processing block, and Table 3.18 shows input/output specifications.

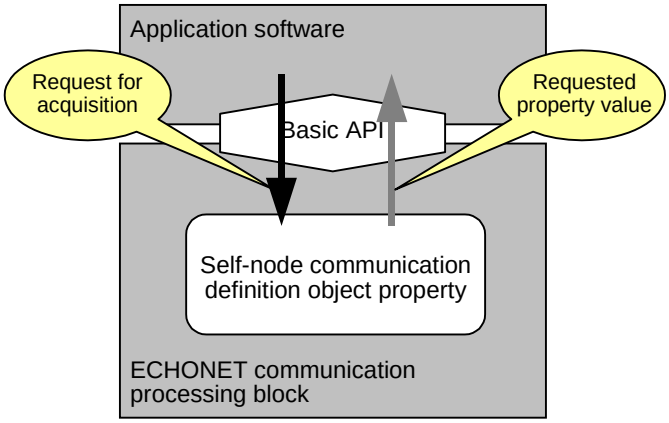

Fig. 3.11

Table 3.18 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

(18) 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 ECHONET communications processing block. (A request for setting a property value from another node shall be processed in the communications middleware but specially not put up to the Application Software.) The property items, contents, etc. that accept the setting from another 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 communications 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 communications middleware does not change a property value without a request from the Application Software.) Fig. 3.12 shows the relationship between this API and the ECHONET communications processing block and Table 3.19 shows input/output specifications.

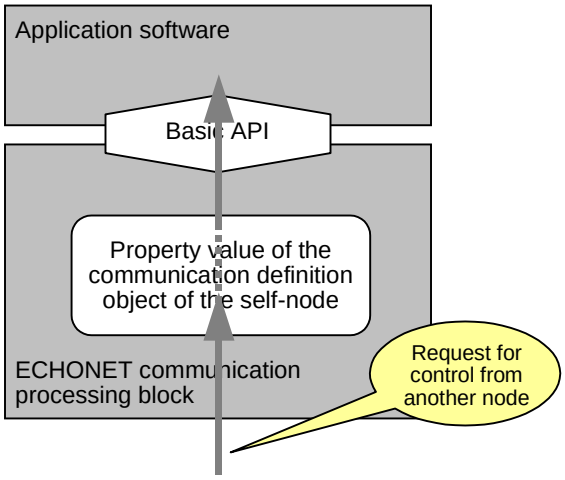

Fig. 3.12

Table 3.19 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

(19) Another node communication definition object property value getting

Reads (gets) property value of each communication definition object instance of another node for 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 communications 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). Fig. 3.13 shows the relationship between this API and the ECHONET communications processing block, and Table 3.20 shows input/output specifications.

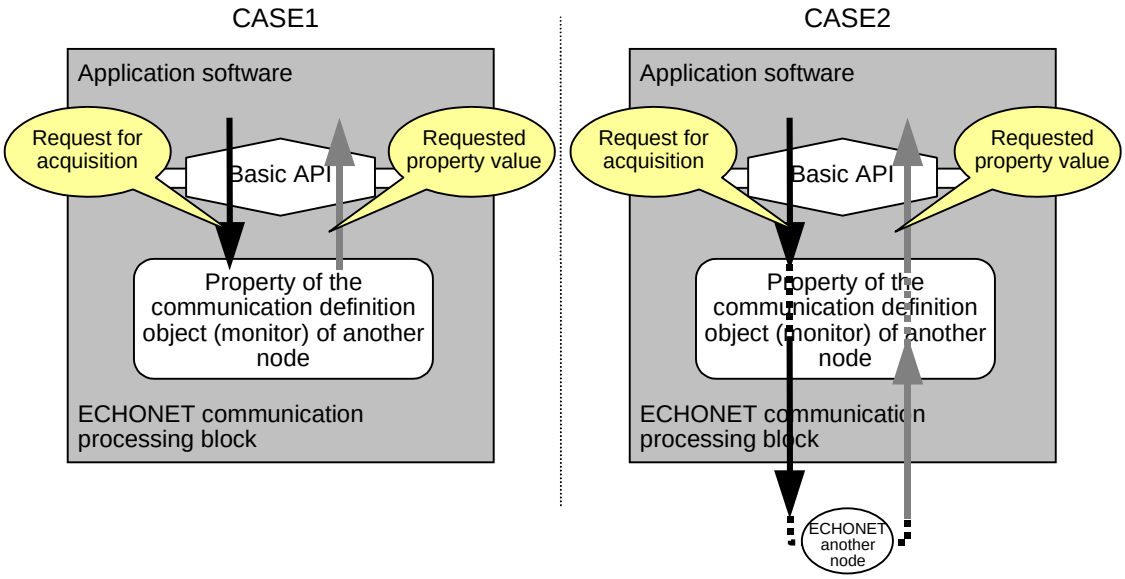

Fig. 3.13

Table 3.20 List of Another 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

(20) Another node communication definition object property value notice acquisition

Reads (gets) property value of each communication definition object instance notified by another node for ECHONET communications processing block. Synchronization between read time and notice time from another node shall not be specified (non-synchronization is allowed). Fig. 3.14 shows the relationship between this API and the ECHONET communications processing block, and Table 3.21 shows input/output specifications.

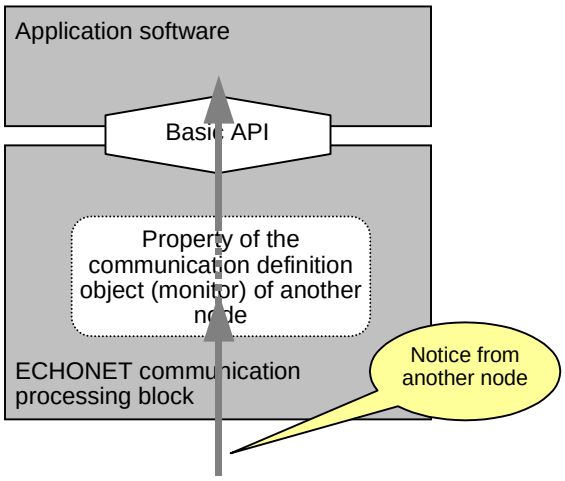

Fig. 3.14

Table 3.21 List of Another 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 from 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 from another node.	Optional
Input	prop_id	Specifies a property to check if a property value setting notice of the communication definition object was given from 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

(21) Another node communication definition object property value setting request

Requests to set the property value of each communication definition object instance of another node for ECHONET communications processing block. Another node communication definition object instance property value setting request is classified into two cases, the first in which a response of a setting request result is not required (CASE 1 in the following figure) and the second in which a 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). Fig. 3.15 shows the relationship between this API and the ECHONET communications processing block, and Table 3.22 shows input/output specifications.

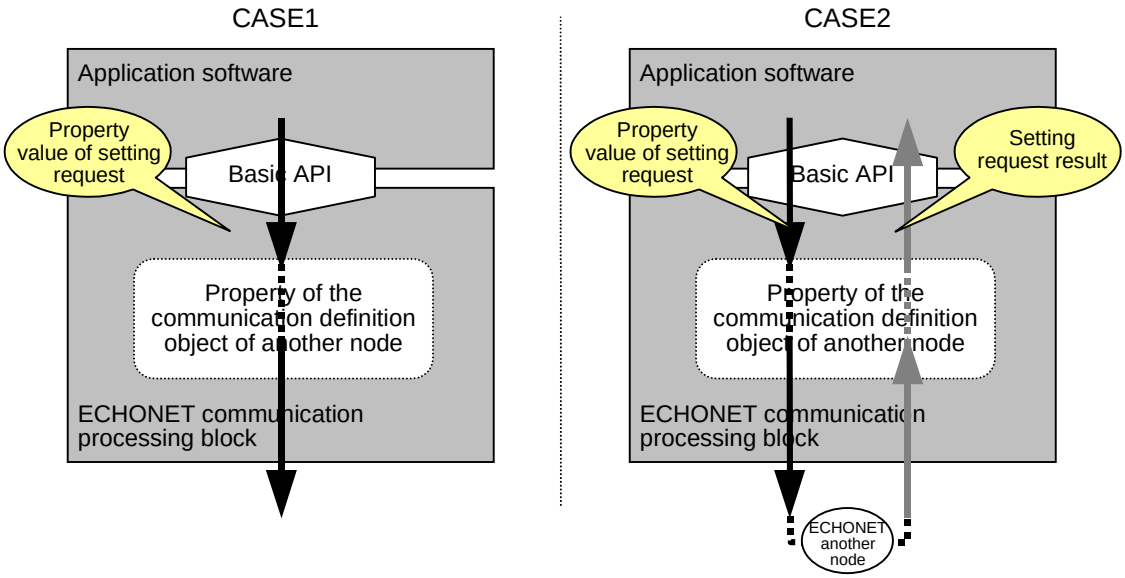

Fig. 3.15

Table 3.22 List of Another 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

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

Sets property value of each service object instance of self-node for 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). Fig. 3.16 shows the relationship between this API and the ECHONET communications processing block, and Table 2.23 shows input/output specifications.

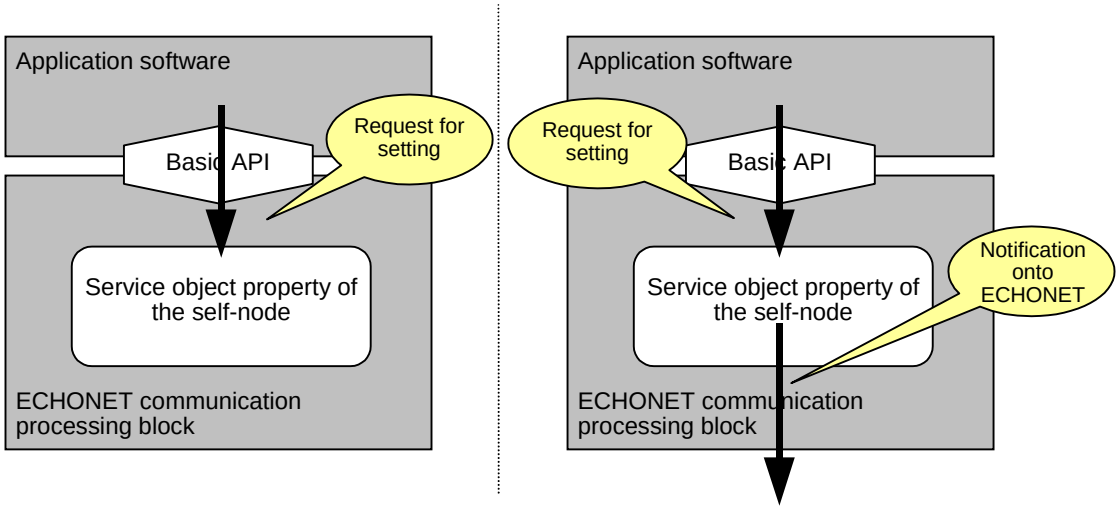

Fig. 3.16

Table 2.23 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 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

(23) Self-node service object property value getting

Reads (gets) property value of each service object instance of self-node for ECHONET communications processing block. The target property items, contents, etc. for reading differ for each service object instance (see Parts 2, 8, and 9). Fig. 3.17 shows the relationship between this API and the ECHONET communications processing block, and Table 3.24 shows input/output specifications.

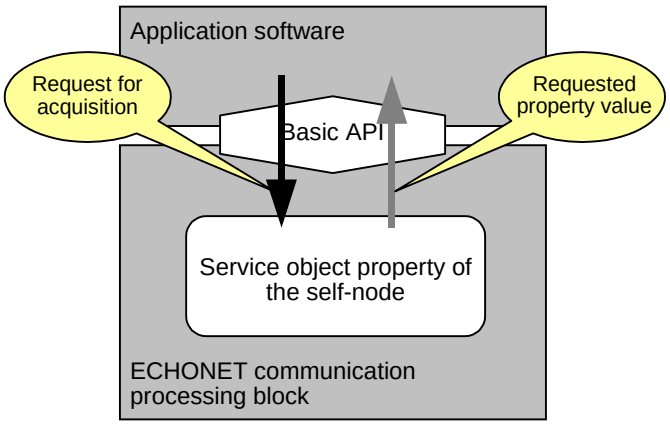

Fig. 3.17

Table 3.24 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

(24) 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 ECHONET communications processing block. (Requests for reading a property value from another node shall be processed in the communications 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 communications 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 communications middleware will not change a property value without a request from the Application Software.) Fig. 3.18 shows the relationship between this API and the ECHONET communications processing block, and Table 3.25 shows input/output specifications.

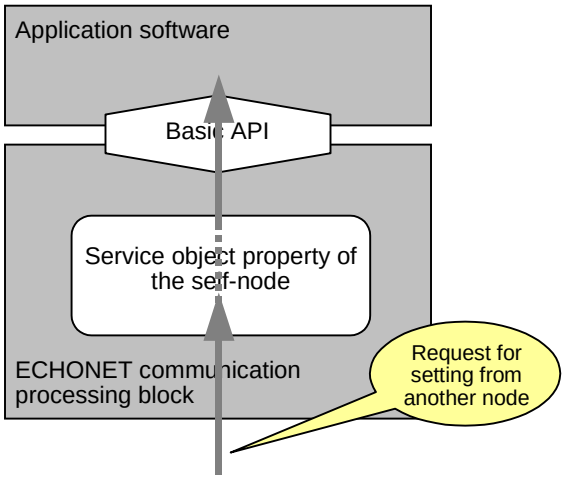

Fig. 3.18

Table 3.25 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

(25) Another node service object property value getting [MidGetOutAplData]

Reads (gets) property value of each service object instance of another node for 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 communications 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). Fig. 3.19 shows the relationship between this API and the ECHONET communications processing block, and Table 3.26 shows input/output specifications.

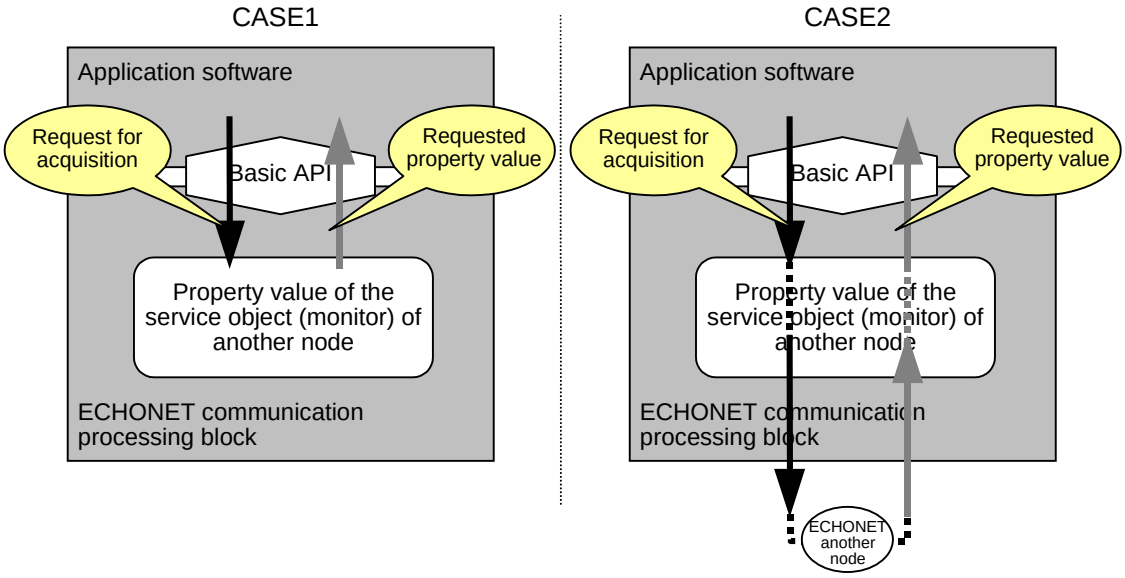

Fig. 3.19

Table 3.26 List of Another 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

(26) Another node service object property value notice acquisition

Reads (gets) property value of each service object instance notified by another node for ECHONET communications processing block. Synchronization between read time and notice time from another node shall not be specified (non-synchronization is allowed). Fig. 3.20 shows the relationship between this API and the ECHONET communications processing block, and Table 3.27 shows input/output specifications.

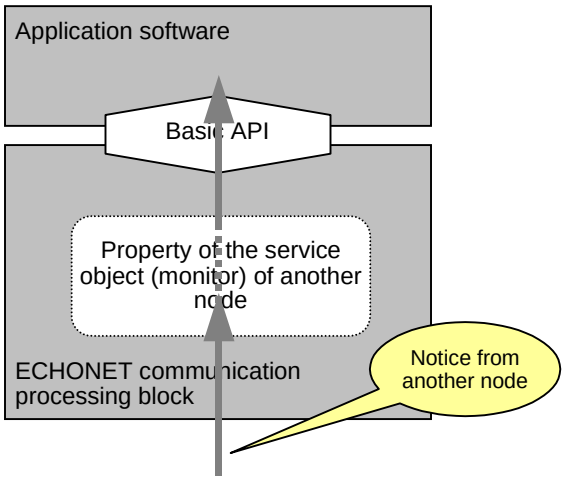

Fig. 3.20

Table 3.27 List of Another 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 from another node.	Optional
Input	objclass_id	Specifies an object instance to check if a property value setting notice of the service object was given from another node.	Optional
Input	prop_id	Specifies a property to check if a property value setting notice of the service object was given from 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

(27) Another node service object instance property value setting request

Requests to set the property value of each service object instance of another node for ECHONET communications processing block. Another node service object instance property value setting request is classified into two cases, the first in which a response of a setting request result is not required (CASE 1 in the following figure) and the second in which a 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). Fig. 3.21 shows the relationship between this API and the ECHONET communications processing block, and Table 3.28 shows input/output specifications.

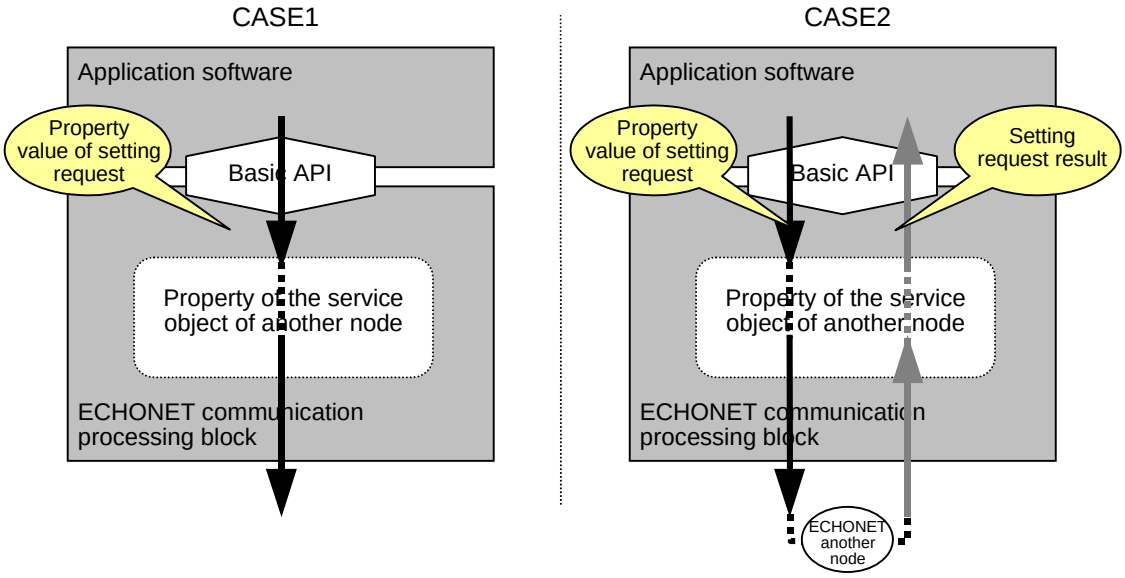

Fig. 3.21

Table 3.28 List of Another 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

(28) 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 3.29 shows input/output specifications.

Table 3.29 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

(29) 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 3.30 shows input/output specifications.

Table 3.30 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

(30) 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 3.31 shows input/output specifications.

Table 3.31 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
Input	objclass_id	Instance of the target control object for acquisition.	Required
Input	prop_id	Specifies a target control property for acquisition.	Required
Output	Return Value	TRUE: Normal, FALSE: error	Optional

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

The Level 2 ECHONET Basic API Specification specifies functions for the following languages in consideration of the reusability of applications to be developed using the Basic API . When necessity arises in the future, functions shall be specified for other languages.

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

This Section describes the level 2 ECHONET Basic API specification only for language (1) above. Because the specification for level 2 is intended to secure interchangeability of the communications middleware from the viewpoint of the Application Software developer, details of the functions are specified. The Basic API functions to be specified for C language are based on the following assumptions. This does not mean that setting and using functions other than those specified in this Section are prohibited.

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

The ECHONET standard targets mainly home devices (white-colored home electrical devices). Even when Basic API functions are mounted on a device to implement a single function, mounting must be attained 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 communications 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 function (required)

  A function of the type that permits the function operations specified in Chapter 3 using the most basic object operations.
- High-level Basic API function

  A function of the type that permits 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:

(0) Function return value
(1) ID type
(2) ESV code
(3) Data type
(4) Access rule
(5) Communication middleware status
(6) 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_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_SetIM	: 0x64 (Request for writing a property value of element specification not requiring a response)
ESV_SetCM	: 0x65 (Request for writing a property value of element specification requiring a response)
ESV_GetM	: 0x66 (Request for reading a property value of element specification)
ESV_INFMReq	: 0x67 (Request for notifying a property value of element specification)
ESV_AddMI	: 0x68 (Request for adding a property value of element specification not requiring a response)
ESV_AddMC	: 0x69 (Request for adding a property value of element specification requiring a response)
ESV_DelMI	: 0x6A (Request for deleting a property value of element specification not requiring a response)
ESV_DelMC	: 0x6B (Request for deleting a property value of element specification requiring a response)
ESV_CheckM	: 0x6C (Request for checking a property of element specification)
ESV_AddMI	: 0x6D (Request for adding an element not requiring a response)
ESV_AddMC	: 0x6E (Request for adding an element requiring a response)
ESV_INF	: 0x73 (Notice of a property value)
ESV_INFM	: 0x77 (Notice of a property value of element specification)

(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_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_ALLM	: 0x1F00 (All element specification services)

(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 indicating target control objects explicitly. An expert well-versed in operations of the ECHONET Communication Middleware and familiar with ECHONET objects can exert every necessary control using only the functions in this function group. These functions require expert operation but enable the control 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	ECHONET object non-array property data write request function (1)	Required
9	MidSetEpc	ECHONET object non-array property data write request function (2)	Required
10	MidGetReceiveEpc	ECHONET object non-array property read request function (1)	Required
11	MidGetEpc	ECHONET object non-array property read request function (2)	Required
12	MidSetSendCheckEpc	ECHONET object non-array property data write check function	Required
13	MidSetSendEpcM	ECHONET object array property data write request function (1)	Optional
14	MidSetEpcM	ECHONET object array property data write request function (2)	Optional
15	MidGetReceiveEpcM	ECHONET object array property data read request function (1)	Optional
16	MidGetEpcM	ECHONET object array property data read request function (2)	Optional
17	MidSetSendCheckEpcM	ECHONET object array property data write check function	Optional
18	MidGetReceiveCheckEpc	ECHONET property data read check function	Optional
19	MidGetEpcSize	ECHONET property size getting function	Optional
20	MidGetEpcAttrib	ECHONET object property attribute getting	Optional
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

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

No.	Function name	Name	Supplement
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	MidInit	ECHONET communications processing block initialization function	Required
38	MidRequestRun	ECHONET communications processing block operation start function	Required
39	MidReset	ECHONET communications processing block resetting function	Optional
40	MidSuspend	ECHONET communications processing block suspension request function	Optional
41	MidWakeUp	ECHONET communications processing block operation restart request function	Optional

4.3 Low-level Basic API Function Detail Specification

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

(1) Name
Indicates a function name.

(2) Function
Explains a function.

(3) Syntax
Indicates the syntax of the function.

(4) Explanation
Explains detailed specifications of arguments and variables.

(5) Return value
Indicates a return value.

(6) Structure
Indicates the specifications of a structure if it exists.

(7) Notes/restrictions
Indicates precautions or restrictions if they are provided.

4.3.1 MidOpenSession

(1) Name

MidOpenSession ECHONET communications processing block operation start request function

(2) Function

Opens a session of the communications middleware.

(3) Syntax

long MidOpenSession(short MidNo)

(4) Explanation

Starts a session with the communications middleware specified in midNo. When there is only one communications middleware on the computer, always specify 0 in midNo. When there is multiple communications middleware on the computer, use midNo to specify the communications middleware to open the session. Use the MidInit function or start communications middleware in any 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 opened session of the communications middleware.

(3) Syntax

long MidCloseSession(void)

(4) Explanation

Terminates all the currently opened sessions and releases all communication resources with communications 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

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 an arbitrary time during setting 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

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 an arbitrary 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 communications software ID is also obtained. The function can be called at an arbitrary time during device ID or lower-layer communications 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 communications middleware.

(3) Syntax

long MidSetControlVal(MidControl *m_data)

(4) Explanation

Sets the operation information of the communications middleware being started. The function can be called at an arbitrary 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, a 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 communications middleware operation information.

(3) Syntax

long MidGetControlVal(MidSetup *midset)

(4) Explanation

Obtains operation information of the communications middleware being started. The function can be called at an arbitrary time during information acquisition.

Midset : [out] Operation information acquisition area

(5) Return value

EPAI_NO_ERROR : Success in acquisition

EAPI_NOTOPEN : Non-start (Session not opened)

(6) Structure

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

(7) Notes/restrictions

None

4.3.8 MidSetSendEpc

(1) Name

MidSetSendEpc ECHONET object non-array property data write request function (1)

(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)

(4) Explanation [Optional function]

Writes data to the ECHONET property specified in id,eoj_code and epc_code and transmits the service of esv_code.

The function can be called at an arbitrary time during data writing.

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 one 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)

data : [in] Data contents

size : [in] Data size

(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)

(6) Structure

None

(7) Notes

Array elements cannot be handled.

4.3.9 MidSetEpc

(1) Name

MidSetEpc ECHONET object non-array property data write request function (2)

(2) Function

Writes data in non-array ECHONET property.

(3) Syntax

long MidSetEpc (short id_kind, short id, long eoj_code, short epc_code, LPCSTR data, short size)

(4) Explanation

Writes data to the ECHONET property specified in id, eoj_code and spc_code. This function can be called at an arbitrary time during data writing. If data to be written to the self-device differs from previous data, the state notice service is performed only when the state change notice processing is set.

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 bytes are used.)
data : [in] Data setting
size : [in] Area size

(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

(6) Structure

None

(7) Notes

Array elements cannot be handled.

4.3.10 MidGetReceiveEpc

(1) Name

MidGetReceiveEpc 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 )
```

(4) Explanation

Reads the received data of the ECHONET property specified in id, eoj_code, and epc_code. This function can be called at an arbitrary time during reading received data. If the received data is a write request (requiring a response), a response is returned upon completion of 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] SEOJ code (Only 3 low-order bytes are used.)
epc_code : [in] EPC code (Only 1 low-order bytes are used.)
buff_size : [in] Area size
esv_code : [in] ESV code save area
data : [out] Data contents save area
data_size : [out] Data read size
eoj_code2 : [in] SEOJ code or DEOJ code on communication
Only 3 high-order bytes are used. If the code is not found, 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 data received
EAPI_MEMBER_EPC	: Array element property
EAPI_DATASIZE_EROR	: Illegal data size

(6) Structure

None

(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 an arbitrary 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 bytes are 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

(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)

(4) Explanation

Checks whether data can be written into the ECHONET property specified in id, eoj_code, and epc_code. This function can be called at an arbitrary 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 bytes are used.)

(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 resource
EAPI_NOCONDITION : Write disable property

(6) Structure

None

(7) Notes

The array element specification cannot be read.

4.3.13 MidSetSendEpcM

(1) Name

MidSetSendEpcM ECHONET object array property data write request function (1)

(2) Function

Data is written in 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 member_no, LPCSTR data, short size)

(4) Explanation

Writes data in the element of member_no of the ECHONET property specified in id, eoj_code, epc_code and transmits the esv_code service. This function can be called at an arbitrary time during data writing. The element is validated upon completion of writing.

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 bytes are 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)

member_no : [in] Element No. (0 to 0xFFFE)
data : [in] Data contents
size : [in] Data size

(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_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)

(6) Structure

None

(7) Notes

Write is disabled except for array element specification.

4.3.14 MidSetEpcM

(1) Name

MidSetEpcM 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)

(4) Explanation

Writes data to the element of member_no of the ECHONET property specified in id, eoj_code, and epc_code. This function can be called at an arbitrary time during data writing.

If data written to the self-device differs from previous data, a status notice service is performed only when the status change notice processing is set.

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 bytes are used.)
member_no : [in] Element No. (0 to 0xFFFE)
data : [out] Data contents save area
size : [in] Area size

(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

(6) Structure

None

(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

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 arbitrary 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 bytes are 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

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 arbitrary 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 bytes are 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

(1) Name

MidSetSendCheckEpcM ECHONET object array property data read check function

(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)

(4) Explanation

Checks whether 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 bytes are used.)
member_no : [in] Element No. (0 to 0xFFFE)

(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 resource
EAPI_NOCONDITION	: Write disable property

(6) Structure

None

(7) Notes

None

4.3.18 MidGetReceiveCheckEpc

(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)

(4) Explanation

The objects of all devices are searched, and the IDs and EPC in which reception occurred are listed in receiving order. This function can be called at any arbitrary time during receipt check.

buff_num : [in] Maximum number of listed elements
id : [out] Device ID (-1: No ID control)
EA : [out] ECHONET address
eoj_code : [out] EOJ code (Only 3 low-order bytes are used.)
epc_code : [out] Received object EPC code save area (Only one low-order byte is used.)
esv_code : [out] ESV code save area
member_no : [out] Array element No. save area
For other than the array element object, -1 is saved.
out_num : [out] Listed number save area

(5) Return value

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

(7) Notes

In the case where buff_num < out_num, receive 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

Obtains data size of the ECHONET property specified in id, eoj_code, and epc_code. This function can be called at any arbitrary 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 bytes are 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

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 bytes are 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 MidGetEpcMemberData

(1) Name

MidGetEpcMemberData ECHONET object array property array element acquisition function

(2) Function

Gets array element object information.

(3) Syntax

long MidGetEpcMemberData (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

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 an arbitrary 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 bytes are 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

In the case where 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

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_NORESORCE : Insufficient resource

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

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 an arbitrary 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_NORESORCE	: Insufficient resource
EAPI_EXIST_OBJ	: Specified object exists
EAPI_NOTFOUND_NODE	: Specified control device not found

(6) Structure

None

(7) Notes

None

4.3.24 MidCreateEpc

(1) Name

MidCreateEpc 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)

(4) Explanation

Creates an ECHONET property specified in node_id, eoj_code, and epc_code in the specified device and the specified ECHONET object. The specified device and the specified object must already exist. This function can be called at an arbitrary time during ECHONET property creation.

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 bytes are 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)

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)

(5) Return value

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

None

(7) Notes

Addition of array ECHONET properties is not possible.

4.3.25 MidCreateEpcM

(1) Name

MidCreateEpcM 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)

(4) Explanation

Creates an ECHONET property with one array element specified in node_id, eoj_code, and epc_code in the specified device and the specified object. The specified device and the specified object must exist. This function can be called at an arbitrary time during array element property creation.

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 bytes are 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_ALLM : 0x1F00 (All element specification services)

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_NORESORCE	: Insufficient resource
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

None

(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

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 bytes are 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_NORESORCE	: Insufficient resource or total number of elements exceeding 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

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 bytes are 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_NORESORCE	: Insufficient resource or total number of elements exceeding 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

Deletes another control device specified in node_id. This function can be called at any arbitrary 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

Deletes an ECHONET object specified in node_id and eoj_code.

This function can be called at an arbitrary 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

Deletes the ECHONET property specified in node_id, eoj_code, and epc_code. This function can be called at an arbitrary 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 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_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

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 an arbitrary 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 bytes are 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 do not exist as a result of this function, this 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 communications middleware.

(3) Syntax

long MidGetState (short *state)

(4) Explanation

Obtains current status of communications 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

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

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 bytes are 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

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 MidObjRead and MidObjReadCheck.

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 bytes are 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

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 one 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

The case of buffnum < data_num signifies that there is a receipt target Epc that is not listed.

4.3.37 MidInit

(1) Name

Midlnit	ECHONET communications processing block initialization function

(2) Function

Starts and initializes communications middleware.

(3) Syntax

long MidInit(short mid_no, const char* mid_name, short mid_name_len, Dll_Init *p_init, short dev_num, Dll _Init *l_init)

(4) Explanation

Starts the ECHONET communications processing block of the communications middleware specified in mid_no. The protocol difference absorption processing block and the low-order communication module are loaded and initialized. No session is opened.

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 communications software initialization
EAPI_ILLEGAL_PARAM	: Illegal number of low-order communication modules mounted

(6) Structure

None

(7) Notes

None

4.3.38 MidRequestRun

(1) Name

MidRequestRun ECHONET Communication Middleware operation start function

(2) Function

Requests operation start of communications 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 communications 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

All operations under the ECHONET communications processing block (protocol difference absorption processing block and discrete lower-layer communications software) of the communications middleware are started.

4.3.39 MidRequest

(1) Name

MidRequest ECHONET Communication Middleware reset request function

(2) Function

Requests to reset the ECHONET Communication Middleware.

(3) Syntax

long MidReset(void)

(4) Explanation

Resets all protocol difference absorption processing blocks and discrete lower-layer communications software under the ECHONET communications processing block. Clears all data waiting for transmission and data waiting for reception that are held at that time.

(5) Return value

EAPI_NO_ERROR	: Success in resetting
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

After completion of resetting, status proceeds to wait status or a wait for RequestRun.

4.3.40 MidReset

(1) Name

MidReset ECHONET Communication Middleware suspension request function

(2) Function

Request to suspend communications middleware.

(3) Syntax

long MidSuspend(void)

(4) Explanation

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.

4.3.41 MidWakeUp

(1) Name

MidWakeUp ECHONET Communication Middleware operation restart request function

(2) Function

Request to restart operation of communications middleware.

(3) Syntax

```
long MidWakeUp( void )
```

(4) Explanation

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 shall be valid only when operation is stopped by MidSuspend.

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

(Details provided in V 1.0 or later.)