第４部　ECHONET 基本 API 仕様

改定履歴

・Version1.0　２０００年3月１８日　制定，コンソーシアム会員内公開。
２０００年7月　一般公開。
・Version1.01　２００１年5月２３日　コンソーシアム会員内公開。
Version1.0 の追補&正誤反映版。
・Version2.00　２００１年8月０７日　コンソーシアム会員内公開。
JAVA 言語版 API 規定を中心とした追加改訂。尚、第５章は、別冊の形でファイルを分けての公開としています。
・Version2.01　２００１年１２月１９日　コンソーシアム会員内公開。
Version2.00 から誤記修正
・Version2.10Preview　２００１年１２月２８日　コンソーシアム会員内公開。
・Version2.10Draft　２００２年2月１５日　コンソーシアム会員内公開。
・Version2.10　２００２年3月7日　コンソーシアム会員内公開。

変更のある目次項目は、以下の通り。

	変更部位（目次項目）	追加・変更概要
1	3.1	第 2 部状態遷移の改定にともない、リセット要求を初期化要求に集約し削除。
2	3.2	第 2 部状態遷移の改定しにともない、リセット要求を初期化要求に集約し削除。
3	4.2	第 2 部状態遷移の改定にともない、MidStart, MidInitAll を追加。MidReset の記載を修正。
4	4.3	第 2 部状態遷移の改定にともない、MidStart, MidInitAll を追加。MidReset の記載を修正。

・Version2.11　２００２年4月26日　コンソーシアム会員内公開。
変更のある目次項目は以下の通り

	変更部位（目次項目）	追加・変更概要
1	2.1	表2.1 に通信停止要求と完全停止要求を追加
2	2.2	通信停止要求と完全停止要求についての説明を追加。
3	3.1	表3.1 に通信停止要求と完全停止要求を追加
4	3.2	通信停止要求と完全停止要求についての説明を追加。
5	3.2	一時停止要求の記述内容の修正
6	4.2	表4.1 に MidStop と MidHalt を追加
7	4.3.47	MidStop API の説明を追加
8	4.3.48	MidHalt API の説明を追加

・Version3.00Draft　２００２年6月 １２日　コンソーシアム会員内公開。
変更のある目次項目は以下の通り

	変更部位（目次項目）	追加・変更概要
1	1.2	図1.1 新規伝送メディア追加
2	3.1	表 3.1 に下位通信ソフトウェアアドレスデータテーブル要求、マスタルータ通知要求、ハードウェアアドレスデータ要求を追加
3	3.2	下位通信ソフトウェアアドレスデータテーブル要求についての説明を追加。
4	3.2	マスタルータ通知要求についての説明を追加。
5	3.2	ハードウェアアドレスデータ要求についての説明を追加。
6	4.2	表4.1 に MidGetAddressTableData, MidSetMasterRouterFlag, MidGetHardwareAddress を追加
7	4.3.18	引数説明一部追加
7	4.3.47	項目番号修正
8	4.3.48	項目番号修正
9	4.3.49	MidGetAddressTableData の説明追加
10	4.3.50	MidSetMasterRouterFlag の説明追加
11	4.3.51	MidGetHardwareAddress の説明追加

・Version3.00　２００２年8月２９日　コンソーシアム会員内公開。
変更のある目次項目は以下の通り

	変更部位（目次項目）	追加・変更概要
1	5	Java 言語版 API に関する表記を修正。

・Version3.10Draft　２００２年１１月8日　コンソーシアム会員内公開。
変更のある目次項目は以下の通り

	変更部位（目次項目）	追加・変更概要
1	4.3.45	MidGetReceiveEPCMulti を修正。

・Version3.10　２００２年１２月１８日 コンソーシアム会員内公開。
・Version3.11　２００３年3月7日　コンソーシアム会員内公開。
・Version3.12　２００３年5月２２日　コンソーシアム会員内公開。
変更のある目次項目は以下の通り

	変更部位（目次項目）	追加・変更概要
1	3.1　表3．１ 3.2 (32)	下位通信ソフトウェアアドレステーブルデータサイズ取得の追記
2	3.2	表番号の付与 表3．３３～3．３６
3	4.2	表4.1No.31,No.45 関数名修正

・Version3.20Draft　　２００３年10月17日　コンソーシアム会員内公開。

	変更部位（目次項目）	追加・変更概要
1	4.3.8 4.3.9 4.3.12 4.3.13	・ MidSetEPC 系の関数を配列でも処理できるように修正
2	4.3.10 4.3.11	・ MidGetEPC 系の関数を配列でも処理できるように修正
3	第4部	・ 要素番号の範囲を[0～0xFFFE]を[0～0xFFFF]に変更 ・ 暗号方式をDESからAES-CBCに変更
4	4.3.41	・ MidRequestRunの引数・説明修正
5	4.3.53	・ 関数MidGetReceiveCheckEpcMultiを追加
6	4.3.54	・ 関数MidGetDevIDを追加

・Version3.20　２００４年　1月　8日　　コンソーシアム会員内公開。

	変更部位（目次項目）	追加・変更概要
1	3.1	・ 表3.1に「復号電文に対するデータ読出し確認」、「下位通信ソフトウェア搭載情報要求」、「最新送信エラー情報取得」を追加
2	4.3.8 4.3.9 4.3.10 4.3.12 4.3.13 4.3.17 4.3.24 4.3.25 4.3.44	・ Service Provider Level アクセス制限レベルを複数設定できるように値を修正
3	4.2	・ 表4.1に関数MidGetLastSendError追加
4	4.3.5 4.3.6 4.3.7	・ （5）戻り値の誤記修正
5	4.3.8	・ 名称の説明修正 ・ esv_codeにESV_INF_AREQ追加
6	4.3.10	・ esv_codeを[in]から[out]に変更
7	4.3.13	・ 名称の説明修正 ・ セキュア対応用の関数名を修正 ・ esv_codeの0x6D、0x6E修正、0x78追加
8	4.3.25	・ 構造体をEXT_EPC_Mに修正
9	4.3.45	・ 構文にesv_codeを追加 ・ 説明において esv_code を[in]から[out]に修正
10	4.3.54	・ 関数MidGetDevIDのデータ型をlongに変更
11	4.3.55	・ MidGetLastSendError追加
12	5.3.1.4	・ 例外にタイムアウト追加
13	5.3.1.20	・ thisが同報アドレスの場合、リターンコードは「－1」に修正
14	5.3.1.21	・ 構文５～８を追加 ・ 引数にタイムアウト時間を追加

		・ 例外にタイムアウトを追加

・Version3.21　２００４年　5月２６日　　コンソーシアム会員内公開。
・Version3.21　２００５年１０月１３日　　一般公開。

・ エコーネットコンソーシアムが発行している規格類は、工業所有権(特許,実用新案など)に関する抵触の有無に関係なく制定されています。
エコーネットコンソーシアムは、この規格類の内容に関する工業所有権に対して、一切の責任を負いません。
・ 本規格発行者は有償･無償を問わず、いかなる第三者に対しても JAVA、IrDA、Bluetooth、HBS のライセンスを許諾する権限や免責を与える権限を有していません。JAVA、IrDA、Bluetooth、HBS を使用する場合、当該使用者は自己の責任と判断に基づき、上記規格について使用許可を得るなどの措置が必要です。
・ この書面の使用による、いかなる損害も責任を負うものではありません。

目次

第1章 概要

１．１ 基本的考え方

ECHONET では、アプリケーションソフトウェアの開発の容易性、移植の容易性を実現するため、API（Application Programming Interface）を基本 API（第２部で述べた ECHONET 通信ミドルウェアの機能を使用するための API）およびサービス API（第８部で述べるサービスミドルウェアの機能を利用するための API）としてその仕様を規定する。本部では、前者の基本 API の規定を示す。

基本 API は、ECHONET 通信ミドルウェアの機能を使用するための API である。アプリケーションソフトウェアの開発者に、通信の手順や処理を意識させないインタフェースとすることに留意し、他ノード上の機能の操作は、ECHONET 通信ミドルウェアにある ECHONET オブジェクトを操作する形で実現するしくみとした。基本 API は、第２部で定義された通信プロトコルを用いて他の機器の持つオブジェクトに対してアクセスするインタフェースとして提供される。すなわち、本 API を使用することにより、他の機器のオブジェクトに対してオブジェクトサービスを要求したりその応答を受信したりすることができる。また、本 API を使用することにより、他の機器から要求されたオブジェクトサービスを受け取り自身の処理を行いその応答を送信することができる。

基本 API は、特定のアプリケーションソフトウェアを指向しない汎用的なインタフェースとして提供されるものであることに留意してその仕様を規定していく。本章では、ECHONET 基本 API のインタフェースの機能を規定し、機能の実現時に基本 API として用いる入出力データ項目の規定、及び具体的な言語が指定された場合の関数の規定を行う。入出力データ項目は、「レベル１ECHONET 基本 API 仕様」として詳細仕様を示し、関数は、「レベル２ECHONET 基本 API 仕様」として詳細仕様を示す。

１．２ 通信レイヤ上の位置づけ

基本 API の通信レイヤ構成上の位置付けを図１．１に示す。ECHONET 通信処理部では、アプリケーションソフトウェアが設備系システムの機器を遠隔制御したり機器の状態をモニタしたりする際の処理を簡単に行えるようにするための通信プロトコル処理や、通信プロトコル処理のための情報の保持や、自機器あるいは他機器の状態などの様々な情報の管理を行う。ECHONET 基本 API は、このような ECHONET 通信処理部をアプリケーションソフトウェアが利用するためのインタフェースである。

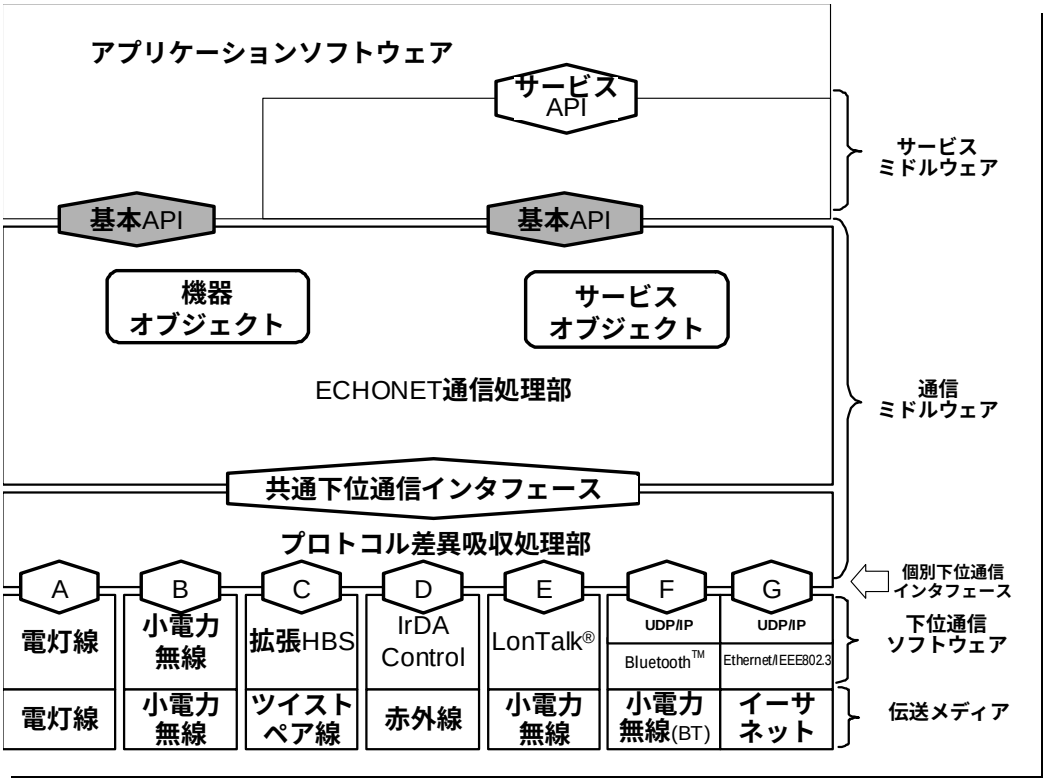


図1．1　基本 API の通信レイヤ構成上の位置付け

第２章 ECHONET 基本 API 機能仕様

２．１ ECHONET 基本 API 機能一覧

ECHONET 通信ミドルウェアの規格（第２部で詳細規定）においては、ECHONET ノード間での制御や設定のやり取りは、ECHONET オブジェクトを操作することで実現することとした。また、アプリケーションソフトウェアと ECHONET 通信ミドルウェア間の通信に係わる制御や設定のやり取りも、ECHONET オブジェクトを操作することで実現できるようにした。この為、ECHONET 通信ミドルウェアの操作（API）の基本は、ECHONET オブジェクト操作といえる。アプリケーション開発者から見ると、ECHONET オブジェクトの操作機能は、以下の８種類に分類される。

(1)	自ノードの機器オブジェクト操作機能
	自ノードの機器としての機能・情報を他ノードに開示、或いは、自ノードの機器としての機能・情報を他ノードから制御・設定させるための自ノード上の通信ミドルウェア操作機能。
(2)	自ノードのプロファイルオブジェクト操作機能
	自ノードのノードとしての機能・情報を他ノードに開示するための自ノード上の通信ミドルウェア操作機能。
(3)	自ノードの通信定義オブジェクト操作機能
	自ノードの機器オブジェクトやプロファイルオブジェクトの各プロパティの通信上の動作を通信ミドルウェアに設定したり、それらの情報を他ノードに開示（設定によっては、他ノードからの設定も受けつける）するための自ノード上の通信ミドルウェア操作機能。
(4)	自ノードのサービスオブジェクト操作機能
	自ノードのサービスミドルウェアの機能・情報を他ノードに開示、或いは、自ノードのサービスミドルウェアの機能・情報を他ノードから制御・設定させるための自ノード上の通信ミドルウェア操作機能。サービスミドルウェアによる操作を基本とする。
(5)	他ノードの機器オブジェクト操作機能
	ECHONET を介して、他ノードで開示されている機器としての機能（機器オブジェクト）に対して、設定制御や、状態・情報の取得を行うための自ノード上の通信ミドルウェア操作機能。
(6)	他ノードのプロファイルオブジェクト操作機能
	ECHONET を介して、他ノードで開示されるサービスミドルウェアの機能（サービスオブジェクト）に対して、設定制御や、状態・情報の取得を行うための自ノード上の通信ミドルウェアの操作機能。
(7)	他ノードの通信定義オブジェクト操作機能
	ECHONET を介して、他ノードの通信ミドルウェアでの機器オブジェクトやプロファイルオブジェクトの各プロパティの通信上の動作の設定制御や、状態・情報の取得を行う為の自ノード上の通信ミドルウェアの操作機能。
(8)	他ノードのサービスオブジェクト操作機能
	他ノードのサービスミドルウェアの機能・情報を参照したり、制御・設定させるための自ノード上の通信ミドルウェア操作機能。サービスミドルウェアによる操作を基本とする。

これらの操作対象の ECHONET オブジェクトは、第２部でも示したように、全て同じ構造（複数のプロパティを持ち、それぞれにサービスが規定される等）を持つものであり、

個々のオブジェクトの操作は、統一的に実現でき、比較的簡単な操作仕様とできる。しかし、そうした操作仕様の場合には、アプリケーションソフトウェアの開発者は、ECHONET 通信ミドルウェアの通信制御動作に対して、ある程度詳細の知識を持つ必要があり、知識が不足している場合には、ECHONET 通信ミドルウェアを操作するのが難しいといった状態になる。本部で示す EHCONET 基本 API 仕様を規格として規定する目的の一つは、できる限りアプリケーションソフトウェアの開発者に、通信動作を意識させることなく、ECHONET に接続された機器相互間での情報や制御のやり取りを実現できるようすることにある。しかしながら、あまりにも API が細分化してしまったのでは、逆に使いにくいということや、その API をサポートする為の ECHONET 通信ミドルウェア部のプログラムサイズの増加といったことが起こり得る。

そうした状況を考慮し、表２．１に一覧を示すインタフェース（API）の機能を、規定する。表２．１の機能概要説明は、アプリケーションソフトウェア開発者（基本 API の利用者）の立場での記述となっている。各 API の機能詳細仕様も同様の立場で、次項にて示す。

表２．１　ECHONET 基本 API 機能一覧表

No.	API 名称	機能概要	補足
1	初期化要求	ECHONET 通信ミドルウェア以下の通信処理部の初期化を要求する。	
2	動作開始要求	ECHONET 通信ミドルウェア以下の通信処理部の動作開始を要求する。	
3	障害通知	アプリケーションソフトウェアの障害（異常）状況を ECHONET 通信ミドルウェアに通知する。	
4	一時停止要求	ECHONET 通信ミドルウェア以下の通信処理部に対する、動作の一時停止を要求する。	
5	動作再開要求	ECHONET 通信ミドルウェア以下の通信処理部に対する、動作の再開を要求する。	
6	自ノードのプロファイルオブジェクト操作	自ノードのプロファイルオブジェクトのプロパティ値の設定や取得、他ノードへの通知を行う。	
7	他ノードのプロファイルオブジェクト操作	他ノードのプロファイルオブジェクトのプロパティ値の設定や取得を行う。	
8	自ノードの機器オブジェクト操作	自ノードの機器オブジェクトのプロパティ値の設定や取得、他ノードからのプロパティ値制御要求の取得や他ノードへのプロパティ値の通知を行う。	
9	他ノードの機器オブジェクト操作	他ノードの機器オブジェクトのプロパティ値の設定やプロパティ値の取得を行う。	
10	自ノードの通信定義オブジェクト操作	自ノードの通信定義オブジェクトのプロパティ値の設定や取得、他ノードからのプロパティ値制御要求や他ノードへのプロパティ値の通知を行う。	
11	他ノードの通信定義オブジェクト操作	他ノードの通信定義オブジェクトのプロパティ値の設定や取得を行う。	
12	自ノードのサービスオブジェクト操作	自ノードのサービスオブジェクトのプロパティ値の設定や取得、他ノードからの制御要求の取得や他ノードへのプロパティ値の通知を行う。	
13	他ノードのサービスオブジェクト操作	他ノードのサービスオブジェクトのプロパティ値の設定や取得を行う。	
14	管理オブジェクト追加・削除	プロパティ単位での、ECHONET 通信処理部で管理するオブジェクトの追加・削除を行う。	
15	通信停止要求	ECHONET 通信ミドルウェア以下の通信処理部に通信停止状態への遷移を要求する。	
16	完全停止要求	ECHONET 通信ミドルウェア以下の通信処理部に停止状態への遷移を要求する。	

２．２ ECHONET 基本 API 機能仕様

以下、前節表２．１で示した基本 API 毎に機能詳細仕様を、基本 API の利用者（アプリケーションソフトウェア開発者）の立場での記述として示す。また、ECHONET 通信処理部の動作については、状態遷移との関連を中心に示す。文中下線を引いた状態については、第２部「８．２ECHONET 通信処理部状態遷移」参照。

（1） 初期化要求
ECHONET 通信ミドルウェア以下の通信に関わる初期化を、ECHONET 通信処理部に対して要求する。この要求を受けた ECHONET 通信処理部は、指定された情報にて ECHONET 通信処理部、プロトコル差異吸収処理部、下位通信ソフトウェアの初期化を実行する。初期化実行後は、起動停止状態となるものとする。

（2） 動作開始要求
ECHONET 通信ミドルウェア以下の通信に関わるソフトウェアの動作開始を要求する。起動停止状態でこの要求を受けた ECHONET 通信処理部は、通常動作状態となる（動作を開始する）。

（3） 障害通知
アプリケーションソフトウェアの障害の状態を ECHONET 通信処理部へ通知する。本通知を受けた ECHONET 通信処理部は、アプリケーションソフトウェア異常を保持するが、状態としては、通常動作状態のままとする。（特に動作停止は行わない。）

（4） 一時停止要求
ECHONET 通信ミドルウェア以下の通信に関わるソフトウェアに対して一時停止を要求する。この要求を受けた ECHONET 通信処理部は、ECHONET 通信処理部自体の一時停止要求の場合には、一時停止状態となり待機する。プロトコル差異吸収処理部及び、個別の下位通信ソフトウェアの一時停止要求の場合には、指定された部位のソフトウェアの一時停止処理のみ実施する。

（5） 動作再開要求
ECHONET 通信ミドルウェア以下の通信に関わるソフトウェアに対して一時停止状態を解除し、動作を再開することを要求する。この要求を受けた ECHONET 通信処理部は、自身も含め指定されたソフトウェアの動作の再開を行う。

（6） 自ノードプロファイルオブジェクト操作
ECHONET 通信処理部に対して、自ノードのプロファイルオブジェクトのプロパティ値の設定や設定値の取得、他ノードへのプロパティ値の通知を行う。ECHONET 通信処理部では、通常動作状態の時のみ本 API の処理を受けつける。

（7） 他ノードプロファイルオブジェクト操作
ECHONET 通信処理部に対して、他ノードのプロファイルオブジェクトのプロパティ値の設定や設定値の取得を行う。ECHONET 通信処理部では、通常動作状態の時のみ本 API の処理を受けつける。

（8） 自ノード機器オブジェクト操作
ECHONET 通信処理部に対して、自ノードの機器オブジェクトのプロパティ値の設定、設定値の取得、他ノードからのプロパティ値の操作要求、他ノードへのプロパティ値の通知を行う。ECHONET 通信処理部では、通常動作状態の時のみ本 API の処理を受けつける。

（9） 他ノード機器オブジェクト操作
ECHONET 通信処理部に対して、他ノードの機器オブジェクトのプロパティ値の制御要求、設定値の取得を行う。ECHONET 通信処理部では、通常動作状態の時のみ本 API の処理を受けつける。

（１０）自ノード通信定義オブジェクト操作
ECHONET 通信処理部に対して、自ノードの通信定義オブジェクトのプロパティ値の設定や取得、他ノードからのプロパティ値の制御要求の取得や他ノードへのプロパティ値の通知を行う。ECHONET 通信処理部が保持する自ノード内の機器オブジェクトのプロパティの通信上の動作（定時通知設定や、状態変化時の相手先指定等）の制御が対象。ECHONET 通信処理部では、通常動作状態の時のみ本 API の処理を受けつける。

（１１）他ノード通信定義オブジェクト操作
ECHONET 通信処理部に対して、他ノードの通信定義オブジェクトのプロパティ値の設定や取得、他ノードからのプロパティ値の制御要求の取得を行う。他ノードの ECHONET 通信処理部が保持する機器オブジェクトのプロパティの通信上の動作（定時通知設定や、状態変化時の相手先指定等）の制御が対象。ECHONET 通信処理部では、通常動作状態の時のみ本 API の処理を受けつける。

（１２）自ノードサービスオブジェクト操作
ECHONET 通信処理部に対して、自ノードのサービスオブジェクトのプロパティ値の設定や取得、他ノードからのプロパティ値の制御要求の取得や他ノードへの情報の通知を行う。本 API の操作は、対象となるサービスオブジェクトを利用するサービスミドルウェアにて行うことを基本とする。ECHONET 通信処理部では、通常動作状態の時のみ本 API の処理を受けつける。

（１３）他ノードサービスオブジェクト操作

ECHONET 通信処理部に対して、他ノードのサービスオブジェクトのプロパティ値の設定や取得、他ノードからのプロパティ値の制御要求の取得を行う。本 API の操作は、対象となるサービスオブジェクトを利用するサービスミドルウェアが行うことを基本とする。ECHONET 通信処理部では、通常動作状態の時のみ本 API の処理を受けつける。

（１４）管理オブジェクト追加・削除

ECHONET 通信処理部に対して、管理している自ノード及び他ノードの各種オブジェクトの追加・削除を、プロパティの単位で行う。ECHONET 通信処理部では、通常動作状態の時のみ本 API の処理を受けつける。

（１５）通信停止要求

ECHONET 通信ミドルウェア以下の通信処理部に通信停止状態への遷移を要求する。

（１６）完全停止要求

ECHONET 通信ミドルウェア以下の通信処理部に停止状態への遷移を要求する。

第３章　レベル１ECHONET 基本 API 仕様

３．１ レベル１ECHONET 基本 API 一覧

表３．１に、ECHONET 通信ミドルウェアのサポートする ECHONET 基本 API レベル１の一覧を示す。レベル１の ECHONET 基本 API においては、表３．１の API の項目は表２．１の API のいくつかを分割した形となっている。本レベル１に準拠する実装上の API では、次項で規定する入出力データ項目を備えていれば十分であり、各データ項目の詳細，複数のデータ項目を一つのデータ項目として実現する、或いは一つのデータ項目を更に複数のデータ項目に分割する、などとなっていてもよいものとする。引数名は参考提示とする。各 API の機能説明及び入出力データ項目は、次項にて規定する。以下、説明は、基本 API の利用者（アプリケーションソフトウェア開発者）の立場での記述として示す。

表３．１ レベル１ECHONET 基本 API 一覧表（１／３）

No	API 名称	機能概要	実装規定
1	初期化要求	ECHONET 通信ミドルウェア以下の通信処理部の初期化を要求する。	Required
2	動作開始要求	ECHONET 通信ミドルウェア以下の通信処理部の動作開始を要求する。	Required
3	障害通知	アプリケーションソフトウェアの障害（異常）状況を ECHONET 通信ミドルウェアに通知する。	Optional
4	一時停止要求	ECHONET 通信ミドルウェア以下の通信処理部に対する、動作の一時停止を要求する。	Optional
5	動作再開要求	ECHONET 通信ミドルウェア以下の通信処理部に対する、動作の再開を要求する。	Optional
6	自ノード内プロファイルオブジェクトプロパティ値設定・通知	自ノードのプロファイルオブジェクトのプロパティ値の情報設定や通知を行う。	Required
7	自ノード内プロファイルオブジェクトプロパティ値取得	自ノードのプロファイルオブジェクトのプロパティ値として設定済み情報の取得を行う。	Required
8	他ノード内プロファイルオブジェクトプロパティ値取得	他ノードのプロファイルオブジェクトのプロパティ値の情報の取得を行う。	Optional
9	自ノード内機器オブジェクトプロパティ値設定・通知	自ノードの機器オブジェクトのプロパティ値の情報設定や通知を行う。	Required
10	自ノード内機器オブジェクトプロパティ値取得	自ノードの機器オブジェクトのプロパティ値として設定済み情報の取得を行う。	Optional
11	自ノード内機器オブジェクトプロパティ値設定要求取得	自ノードの機器オブジェクトのプロパティ値の他ノードからの設定・制御要求の取得を行う。	Optional
12	他ノード内機器オブジェクトプロパティ値取得	他ノードの機器オブジェクトのプロパティ値の情報の取得を行う。	Optional

表3．1 レベル1ECHONET 基本API 一覧表（2／3）

No	API 名称	機能概要	実装規定
13	他ノード内機器オブジェクトプロパティ値通知取得	他ノードが通知した他ノード内の機器オブジェクトのプロパティ値の取得を行う。	Optional
14	他ノード内機器オブジェクトプロパティ値設定要求	他ノードの機器オブジェクトのプロパティ値の情報設定要求（制御要求）を行う。	Optional
15	自ノード内通信定義オブジェクトプロパティ値設定・通知	自ノードの通信定義オブジェクトのプロパティ値の情報設定や通知を行う。	Optional
16	自ノード内通信定義オブジェクトプロパティ値取得	自ノードの通信定義オブジェクトのプロパティ値として設定済み情報の取得を行う。	Optional
17	自ノード内通信定義オブジェクトプロパティ値設定要求取得	自ノードの通信定義オブジェクトのプロパティ値の他ノードからの設定・制御要求の取得を行う。	Optional
18	他ノード内通信定義オブジェクトプロパティ値取得	他ノードの通信定義オブジェクトのプロパティ値の情報の取得を行う。	Optional
19	他ノード内通信定義オブジェクトプロパティ値通知取得	他ノードが通知した他ノード内の通信定義オブジェクトのプロパティ値の取得を行う。	Optional
20	他ノード内通信定義オブジェクトプロパティ値要求	他ノードの通信定義オブジェクトのプロパティ値の情報設定要求（制御要求）を行う。	Optional
21	自ノード内サービスオブジェクトプロパティ値設定・通知	自ノードのサービスオブジェクトのプロパティ値の情報設定や通知を行う。	Optional
22	自ノード内サービスオブジェクトプロパティ値取得	自ノードのサービスオブジェクトのプロパティ値として設定済み情報の取得を行う。	Optional
23	自ノード内サービスオブジェクトプロパティ値設定要求取得	自ノードのサービスオブジェクトのプロパティ値の他ノードからの設定・制御要求の取得を行う。	Optional
24	他ノード内サービスオブジェクトプロパティ値取得	他ノードのサービスオブジェクトのプロパティ値の情報の取得を行う。	Optional
25	他ノード内サービスオブジェクトプロパティ値通知取得	他ノードが通知した他ノード内のサービスオブジェクトのプロパティ値の取得を行う。	Optional
26	他ノード内サービスオブジェクトプロパティ値設定要求	他ノードのサービスオブジェクトのプロパティ値の情報設定要求（制御要求）を行う。	Optional
27	管理オブジェクト追加	ECHONET 通信処理部で管理するオブジェクトの追加を行う。	Optional
28	管理オブジェクト削除	ECHONET 通信処理部で管理するオブジェクトの削除を行う。	Optional
29	管理オブジェクト取得	ECHONET 通信処理部で管理しているオブジェクトの取得を行う。	Optional
30	通信停止要求	ECHONET 通信ミドルウェア以下の通信処理部に通信停止状態への遷移を要求する。	
31	完全停止要求	ECHONET 通信ミドルウェア以下の通信処理部に停止状態への遷移を要求する。	
32	下位通信ソフトウェアアドレステーブルデータサイズ取得	下位通信ソフトウェアで保持している下位アドレステーブルデータの組数の取得を行う。	Optional
33	下位通信ソフトウェアアドレステーブルデータ取得	下位通信ソフトウェアで保持している下位アドレステーブルデータを取得する。出力データはデータ組数、ハードウェアアドレス、NodeID、及びマスタルータであることを示すフラグからなる配列データ組からなる。	Optional

表3．1 レベル1ECHONET 基本API 一覧表（3／3）

No	API 名称	機能概要	実装規定
34	マスタルータ通知	通信ミドルウェアに対して、自ノードがマスタルータであるか否かを下位通信ソフトウェアに通知するように要求する。	Optional
35	ハードウェアアドレスデータ取得	下位通信ソフトウェアに対して保持しているハードウェアアドレスデータを要求する。出力データはハードウェアアドレスである。	Optional
36	複合電文に対するデータ読出確認	受信した複合電文を確認する。	Optional
37	下位通信ソフトウェア搭載情報要求	操作が可能な下位通信ソフトウェアの数、および種類の識別を示す下位通信ソフトウェア ID 情報を要求する。	Optional
38	最新送信エラー情報取得	ECHONET 通信ミドルウェアが保持する最新の ECHONET 電文送信エラー情報を取得する。	Optional

３．２ レベル１ECHONET 基本 API 詳細仕様

前項の表３．１で示した API 毎に、以下データの入出力を示す。以下の表中、「入力（Input)」とは、アプリケーションソフトウェアから ECHONET 通信処理部に対して渡される（ECHONET 通信処理部からみた入力である）ことを示し、「出力（Output)」とは、アプリケーションソフトウェアへ ECHONET 通信処理部が渡す（ECHONET 通信処理部からみた出力である）ことを示す。実装においては、これらのデータの内容が入出力として備わっていればよいものとし、受け渡し方法（構造体を用いるとか、受け渡し用のバッファのポインタ情報を渡す等）については、レベル１では特に規定しない。データ名については、参考提示とする。

（1）　初期化要求（搭載必須機能）

ECHONET 通信ミドルウェア以下の通信に関わる初期化 (動作状態の設定) を要求する。この要求を受けた ECHONET 通信処理部（ECHONET 通信ミドルウェア）は、指定された情報にて ECHONET 通信処理部、プロトコル差異吸収処理部、下位通信ソフトウェアの初期化を実行する。但し、通常動作開始は、「動作開始要求」を受けたタイミングで実施する。表３．２に入出力仕様を示す。

表3．２ 初期化要求 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	device_id	初期化対象の指示。 ECHONET 通信処理部、プロトコル差異吸収処理部、個々の下位通信ソフトウェアの識別が可能であること。	Optional
Input	p_init	初期化パラメータ。 各種タイムアウト時間、EA の設定方法指定等含むが初期化対象により具体的な内容は異なる。	Required
Output	Return Value	TRUE：初期化成功、FALSE：初期化失敗。	Optional

（2）　動作開始要求（搭載必須機能）

ECHONET 通信ミドルウェア以下の通信に関わるソフトウェアの動作開始を要求する。表３．３に入出力仕様を示す。

表３．３ 動作開始要求 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	device_id	動作開始対象の指示。 ECHONET 通信処理部、プロトコル差異吸収処理部、個々の下位通信ソフトウェアの識別が可能であること。	Optional
Output	Return Value	TRUE：動作開始成功、FALSE：動作開始失敗。	Optional

（3）　障害通知

アプリケーションソフトウェアの障害の状態を ECHONET 通信処理部へ通知する。本 API により ECHONET 通信処理部が取得した値は、ノードプロファイルの障害内容に設定する。表３．４に入出力仕様を示す。

表３．４ 障害通知 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	trouble_no	トラブル番号の通知。	Required
Output	Return Value	TRUE：障害通知受理、FALSE：障害通知受理不可。	Optional

（4）　一時停止要求

ECHONET 通信ミドルウェア以下の通信に関わるソフトウェアに対して一時停止を要求する。この要求を受けた ECHONET 通信処理部は、ECHONET 通信処理部自体の一時停止要求の場合には、アプリケーションソフトウェアからの「動作再開要求」、「初期化要求」及び「完全停止要求」以外の要求は、アプリケーションソフトウェアからもプロトコル差異吸収処理部（及び下位通信ソフトウェア）からも受けつけ無いものとする。プロトコル差異吸収処理部及び、個別の下位通信ソフトウェアの一時停止要求の場合には、指定された部位のソフトウェアの一時停止処理のみ実施する。表３．５に入出力仕様を示す。

表３．５ 一時停止要求 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	device_id	一時停止対象の指示。 ECHONET 通信処理部、プロトコル差異吸収処理部、個々の下位通信ソフトウェアの識別が可能であること。	Optional
Output	Return Value	TRUE：一時停止受理、FALSE：受理不可。	Optional

（5）　動作再開要求

ECHONET 通信ミドルウェア以下の通信に関わるソフトウェアに対して一時停止状態を解除し、動作を再開することを要求する。この要求を受けた ECHONET 通信処理部は、自身も含め指定されたソフトウェアの動作の再開を行う。表３．６に入出力仕様を示す。

表3．6 動作再開要求 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	device_id	動作再開対象の指示。 ECHONET 通信処理部、プロトコル差異吸収処理部、個々の下位通信ソフトウェアの識別が可能であること。	Optional
Output	Return Value	TRUE：再開成功、FALSE：再開不可(失敗含)。	Optional

（6）　自ノード内プロファイルオブジェクトプロパティ値設定･通知 (搭載必須機能)

ECHONET 通信処理部に対して、自ノードのノードプロファイルクラス，ルータプロファイルクラス及び個々の下位通信ソフトウェアプロファイルクラスのプロパティ値の設定及び、設定値の ECHONET 上ノードへの通知(Optional 機能)を行う。プロファイル情報は、プロファイルオブジェクト内プロパティの情報（第２部参照）である。設定は、ECHONET 通信処理部上にあるプロファイルオブジェクトのプロパティ値の設定（値の書き込み）を行う操作であり、通知は、プロファイルオブジェクトのプロパティ値の ECHONET 上への電文としての通知を行う操作である。図３．１に、本 API と ECHONET 通信処理部の関連を示し、表３．７に入出力仕様を示す。

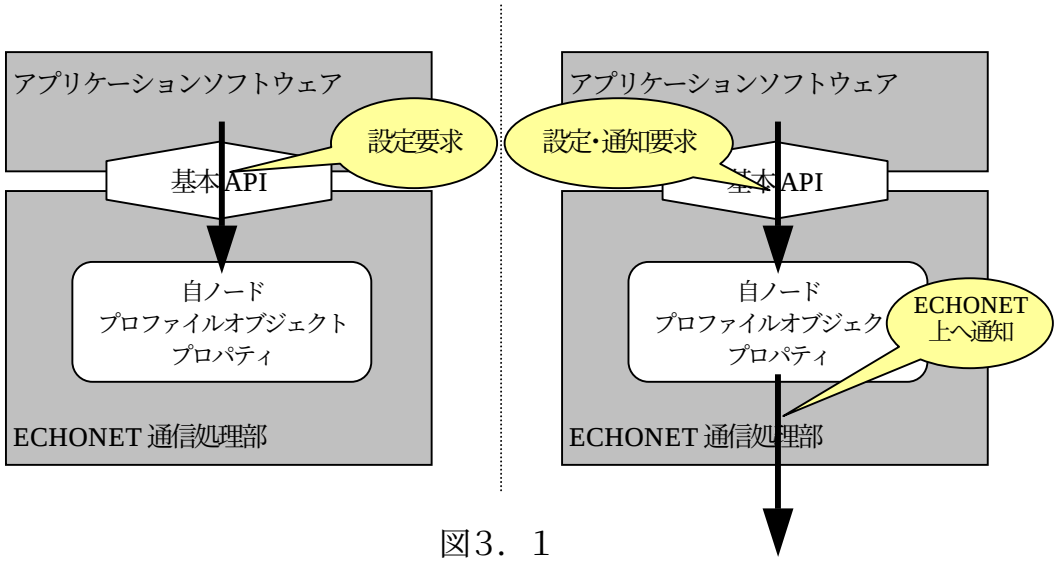


図３．１

表３．７ 自ノード内プロファイルオブジェクトプロパティ値設定・通知 API
入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	objclass_id	プロファイル情報の設定・通知対象のプロファイルクラスインスタンス指定。ノード，ルータ，個々の下位通信ソフトウェアそれぞれのプロファイルオブジェクトのインスタンスが対象となる。	Required
Input	prop_id	対象となるプロパティの指定。	Required
Input	announce_info	設定情報を ECHONET 上へ通知するかどうかの指定。通知する場合には通知先情報も含む。	Optional
Input	prop_info	objclass_id と prop_id で指定されたプロパティへの設定や変更の値の設定。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（7）　自ノード内プロファイルオブジェクトプロパティ値取得（搭載必須機能）

ECHONET 通信処理部に対して、自ノードのノードプロファイルオブジェクトのインスタンス，ルータプロファイルオブジェクトのインスタンス及び個々の下位通信ソフトウェアプロファイルオブジェクトのインスタンスのプロパティ値の読み出し（取得）を行う。図３．２に、本 API と ECHONET 通信処理部の関連を示し、表３．８に入出力仕様を示す。

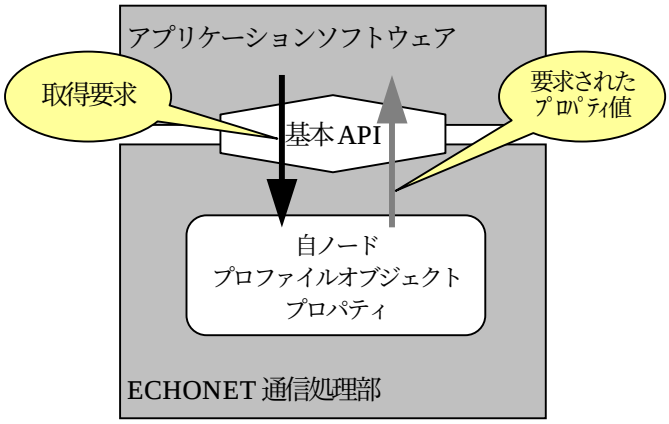


図３．２

表３．８ 自ノード内プロファイルオブジェクトプロパティ値取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	objclass_id	プロファイル情報の取得対象の指定。 全プロファイルオブジェクトクラス（ノード，ルータ，ECHOET 通信処理部，プロトコル差異吸収処理部，及び個々の下位通信ソフトウェアプロファイルクラス）インスタンスが対象となる。	Required
Input	prop_id	対象となるプロパティの指定。	Required
Output	profile_info or prop_info	objclass_id と prop_id で指定されたプロパティ値。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（8）　他ノード内プロファイルオブジェクトプロパティ値取得

ECHONET 通信処理部に対して、他ノードのノードプロファイルクラス，ルータプロファイルクラス及び個々の下位通信ソフトウェアプロファイルクラスのプロパティ値の読み出し（取得）を行う。通信ミドルウェア上でモニタ管理している値を取得要求するケース（下図 CASE1）と、ECHONET を介して、現状の値を取得要求するケース（下図 CASE2）がある。後者のケース（CASE2）の場合、値の要求と実際の取得値の受け取りの同期は、特に規定しないが、並列処理できないマシン（CPU）上で動作するソフトの場合には非同期とすることが望ましい。プロファイル情報としては、自ノード EA や下位通信ソフトウェアの初期設定情報等プロファイルオブジェクト内プロパティの情報（第２部参照）である。図３．３に、本 API と ECHONET 通信処理部の関連を示し、表３．９に入出力仕様を示す。

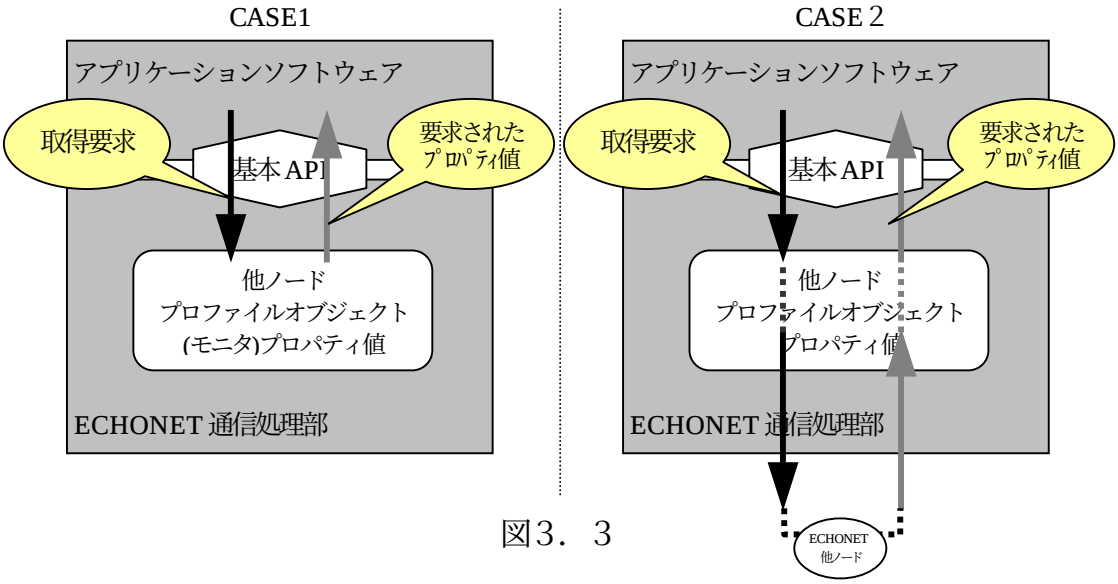


図3．3

表3．9 他ノード内プロファイルオブジェクトプロパティ値取得
API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	プロファイル情報の取得対象のノードの指定。	Required
Input	objclass_id	プロファイル情報の取得対象の指定。 全プロファイルクラスが対象となる。	Required
Input	prop_id	対象となるプロパティの指定。	Required
Output	prop_info	objclass_id と prop_id で指定されたプロパティ値。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（9）　自ノード内機器オブジェクトプロパティ値設定・通知（搭載必須機能）

ECHONET 通信処理部に対して、自ノードの個々の機器オブジェクトのインスタンスのプロパティ値の設定及び、設定値の ECHONET 上ノードへの通知(Optional 機能)を行う。設定・通知対象となるプロパティの項目、内容等、個々の機器オブジェクトのインスタンス毎に異なる（第２部参照)。図３．４に、本 API と ECHONET 通信処理部の関連を示し、表３．１０に入出力仕様を示す。

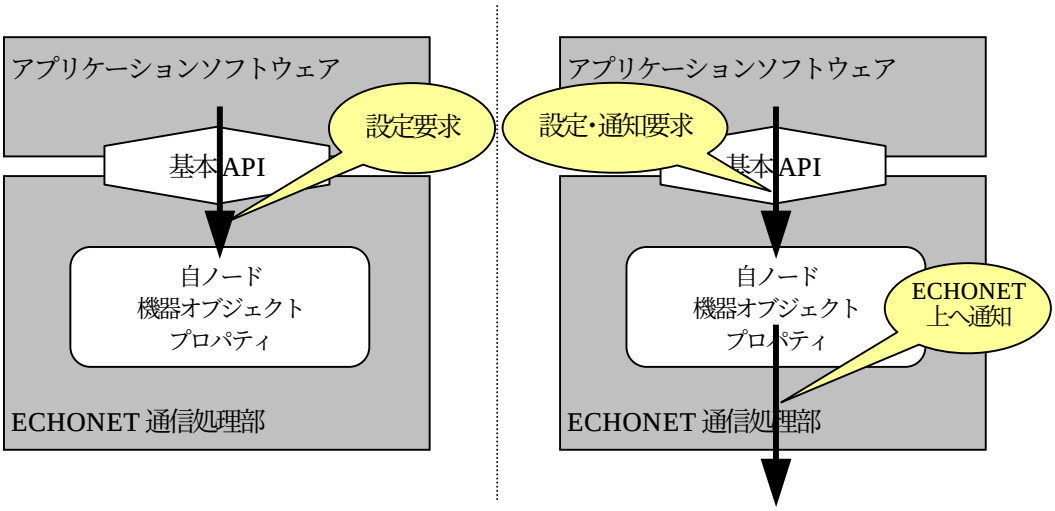


図３．４

表３．１０ 自ノード内機器オブジェクトプロパティ値設定・通知 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	objclass_id	設定・通知対象のプロパティ値の機器オブジェクトのインスタンスの指定。	Required
Input	prop_id	対象となるプロパティの指定。	Required
Input	announce_info	設定値を ECHONET 上へ通知するかどうかの指定。通知する場合には通知先情報も含む。	Optional
Input	prop_info	設定・通知するプロパティ値。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（１０）自ノード内機器オブジェクトプロパティ値取得

ECHONET 通信処理部に対して、自ノードの個々の機器オブジェクトのインスタンスのプロパティ値の読み出し（取得）を行う。読み出し対象となるプロパティの項目、内容等、個々の機器オブジェクトのインスタンス毎に異なる（第２部参照）。図３．５に、本 API と ECHONET 通信処理部の関連を示し、表３．１１に入出力仕様を示す。

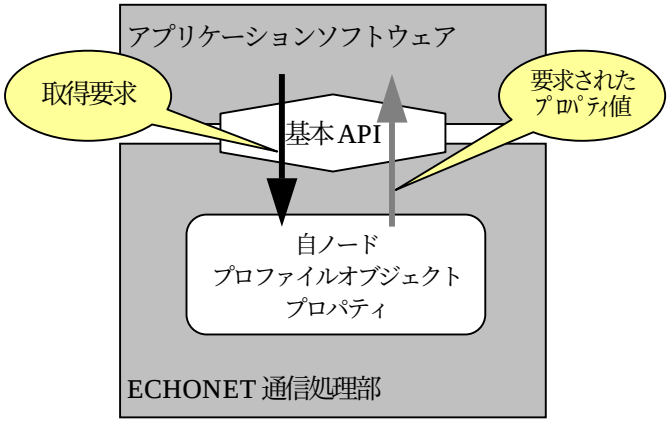


図３．５

表３．１１ 自ノード内機器オブジェクトプロパティ値取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	objclass_id	プロパティ値取得対象の機器オブジェクトの指定。	Required
Input	prop_id	取得対象のプロパティの指定。	Required
Output	prop_info	指定されたプロパティに設定されている値の情報。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（１１）自ノード内機器オブジェクトプロパティ値設定要求取得（搭載必須機能）

ECHONET 通信処理部に対して、他ノードからの自ノードの個々の機器オブジェクトのインスタンスのプロパティ値の設定（読出し以外）要求の取得を行う。他からの制御を受けつけるプロパティの項目、内容等、個々の機器オブジェクトのインスタンス毎に異なる（第２部参照)。他ノードからのプロパティ値の設定要求のアプリケーションソフトウェアでの取得は、アプリケーションソフトウェアから要求のあったタイミングで取得可能とするが、自動的に通知される（イベント）形式となっても構わないものとする。また、他ノードからのプロパティ値へ書き込み要求された値は、アプリケーションソフトウェアにてそのプロパティの実体の変化と同期して通信ミドルウェアに設定するものとし、別途設定されるまでは、要求受信前の値を保持しておく（通信ミドルウェアではアプリケーションソフトウェアからの要求無しにプロパティ値の変更は行わない)。図３．６に、本 API と ECHONET 通信処理部の関連を示し、表３．１２に入出力仕様を示す。

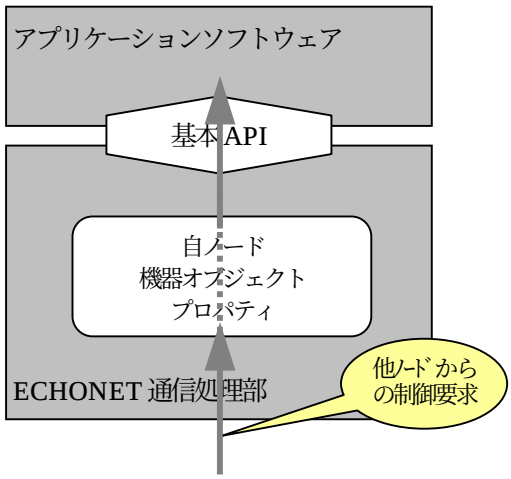


図３．６

表３．１２ 自ノード内機器オブジェクトプロパティ値設定要求取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	objclass_id	他ノードからの機器オブジェクトプロパティ値の設定要求があったかどうかを確認する対象の、オブジェクトのインスタンスの指定。	Optional
Input	prop_id	他ノードからの設定要求内容を確認するプロパティの指定。	Optional
Output	demobj_info	制御要求対象の機器オブジェクト情報（プロパティ情報含む)。機器オブジェクト（プロパティ含む）指定情報、制御サービス情報、及び具体的な設定制御値情報含む。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（１２）他ノード機器オブジェクトプロパティ値取得

ECHONET 通信処理部に対して、他ノードの個々の機器オブジェクトのインスタンスのプロパティ値の読み出し（取得）を行う。通信ミドルウェア上でモニタ管理している値を取得要求するケース（下図 CASE1）と、ECHONET を介して、現状の値を取得要求するケース（下図 CASE2）がある。後者のケース（CASE2）の場合、値の要求と実際の取得値の受け取りの同期は、特に規定しない。読み出し対象となるプロパティの項目、内容等、個々の機器オブジェクトのインスタンス毎に異なる（第２部参照）。図３．７に、本 API と ECHONET 通信処理部の関連を示し、表３．１３に入出力仕様を示す。

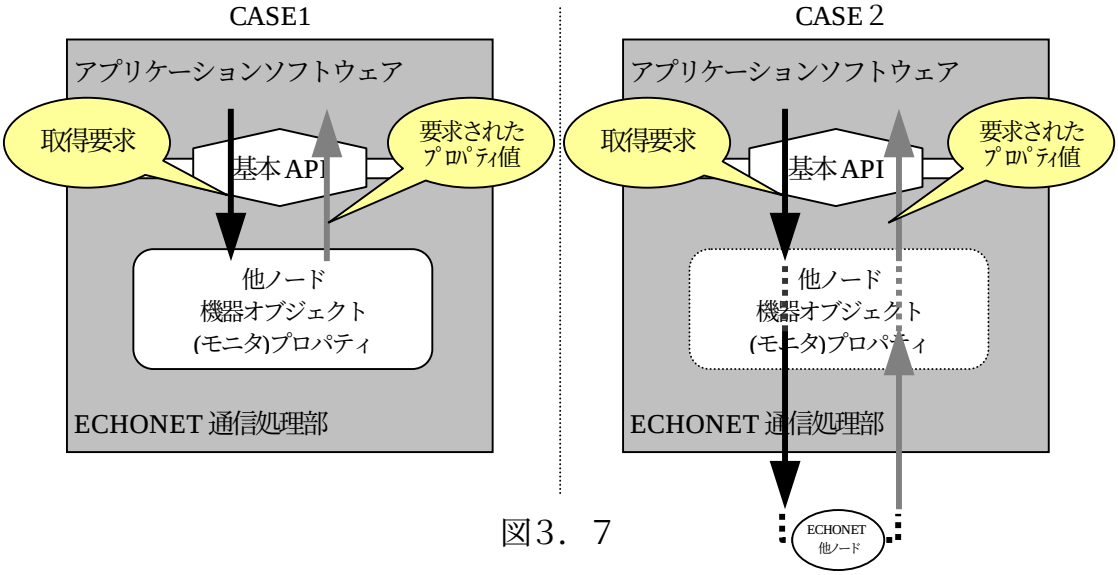


図３．７

表３．１３ 他ノード機器オブジェクト情報取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	機器オブジェクトプロパティ値の取得対象のノードの指定。	Required
Input	objclass_id	機器オブジェクトプロパティ値の取得対象の、クラスインスタンスの指定。	Required
Input	prop_id	プロパティ値取得対象のプロパティの指定。	Required
Input	place_info	情報取得対象の場所（現在自ノード上で保持している情報か、他ノード上の情報か）情報の指定。	Optional
Output	prop_info	指定されたプロパティに設定されている値の情報。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（１３）他ノード機器オブジェクトプロパティ値通知取得

ECHONET 通信処理部に対して、他ノードが通知した個々の機器オブジェクトのインスタンスのプロパティ値の読み出し（取得）を行う。読出しのタイミングと、他ノードからの通知のタイミングの同期は、特に規定しない（非同期でも構わない）ものとする。また、他ノードの機器オブジェクトのインスタンスのプロパティ値のアプリケーションソフトウェアでの取得は、取得要求のあったタイミングで取得可能とする。自動的に通知される（イベント）形式となっても構わないものとする。図３．８に、本 API と ECHONET 通信処理部の関連を示し、表３．１４に入出力仕様を示す。

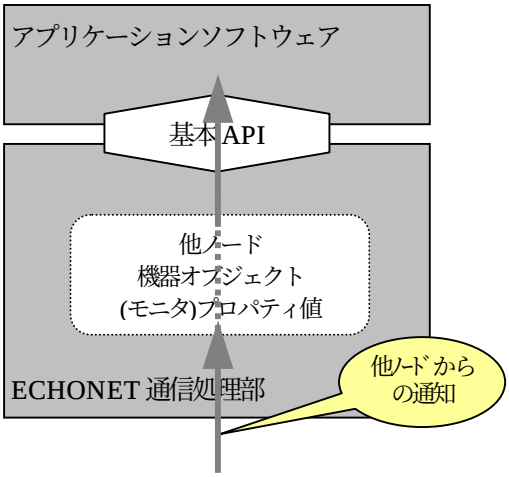


図3．８

表３．１４ 他ノード機器オブジェクトプロパティ値通知取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	他ノードからの機器オブジェクトプロパティ値の設定通知があったかどうかを確認するノードの指定。	Optional
Input	objclass_id	他ノードからの機器オブジェクトプロパティ値の設定通知があったかどうかを確認するオブジェクトのインスタンスの指定。	Optional
Input	prop_id	他ノードからの機器オブジェクトプロパティ値の設定通知があったかどうかを確認するプロパティの指定。	Optional
Output	obj_info	通知された機器オブジェクト情報（プロパティ情報含む）。機器オブジェクト（プロパティ含む）指定情報、制御サービス情報、及び具体的な設定制御値情報含む。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（１４）他ノード機器オブジェクトプロパティ値設定要求

ECHONET 通信処理部に対して、他ノードの個々の機器オブジェクトのインスタンスのプロパティ値の設定要求を行う。他ノードの機器オブジェクトのインスタンスのプロパティ値の設定要求としては、設定要求結果の応答を要求しないケース（下図 CASE1）と、設定要求結果の応答を要求するケース（下図 CASE2）がある。設定要求と実際の設定結果の取得の同期は、特に規定しない。設定・制御対象となるプロパティの項目、内容等は、個々の機器オブジェクトのインスタンス毎に異なる（第２部参照）。図３．９に、本 API と ECHONET 通信処理部の関連を示し、表３．１５に入出力仕様を示す。

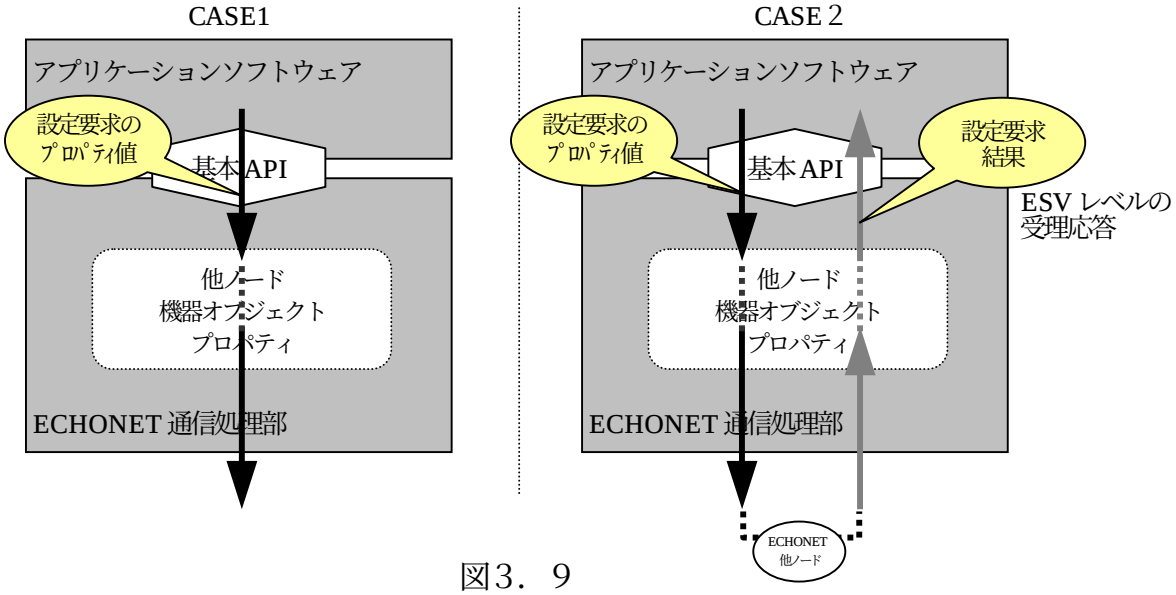


図３．９

表３．１５ 他ノード内機器オブジェクトプロパティ値設定要求 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	設定対象の機器オブジェクトのノードの指定。	Required
Input	objclass_id	設定対象の機器オブジェクトのオブジェクトのインスタンスの指定。	Required
Input	prop_id	設定対象のプロパティの指定。	Required
Input	prop_info	指定されたプロパティに設定する値の情報。サービスの指定を含む。	Required
Output	res_info	設定結果の情報。	Optional
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（１５）自ノード内通信定義オブジェクトプロパティ値設定・通知

ECHONET 通信処理部に対して、自ノードの個々の機器オブジェクトの通信定義オブジェクトインスタンスのプロパティ値の設定及び、設定値の ECHONET 上ノードへの通知(Optional 機能)を行う。設定・通知対象となるプロパティの項目、内容等、個々の通信定義オブジェクトクラス毎に異なる（第２部参照)。図３．１０に、本 API と ECHONET 通信処理部の関連を示し、表３．１６に入出力仕様を示す。

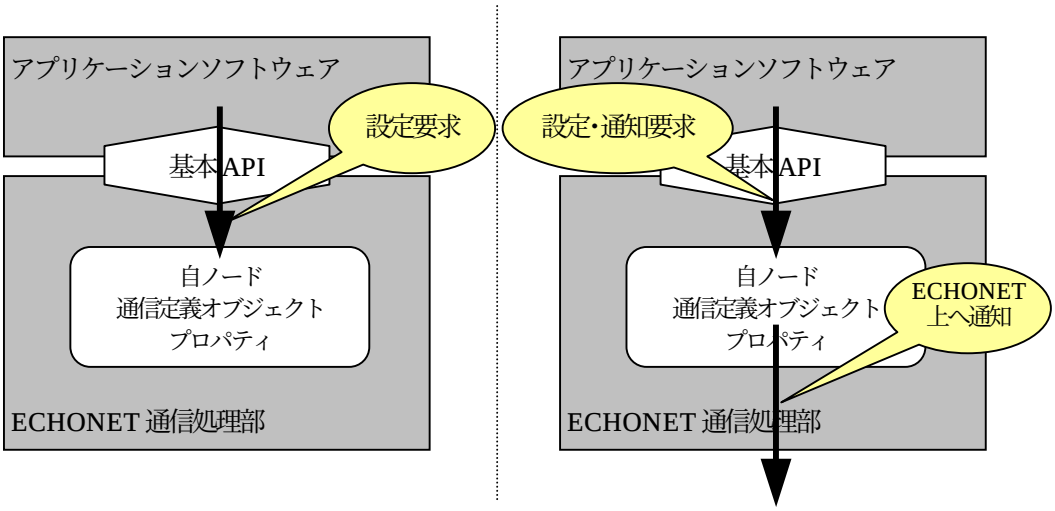


図３．１０

表３．１６ 自ノード内通信定義オブジェクトプロパティ値設定・通知 API
入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	objclass_id	設定対象の通信定義オブジェクトのオブジェクトのインスタンスの指定。	Required
Input	prop_id	設定・通知対象のプロパティの指定。	Required
Input	announce_info	設定情報を ECHONET 上へ通知するかどうかの指定。通知する場合には通知先情報も含む。	Optional
Input	prop_info	通信定義オブジェクトのプロパティ値。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（１６）自ノード内通信定義オブジェクトプロパティ値取得

ECHONET 通信処理部に対して、自ノードの個々の通信定義オブジェクトのインスタンスのプロパティ値の読み出し（取得）を行う。読み出し対象となるプロパティの項目、内容等、個々の通信定義オブジェクトのインスタンス毎に異なる（第２部参照)。図３．１１に、本API と ECHONET 通信処理部の関連を示し、表３．１７に入出力仕様を示す。

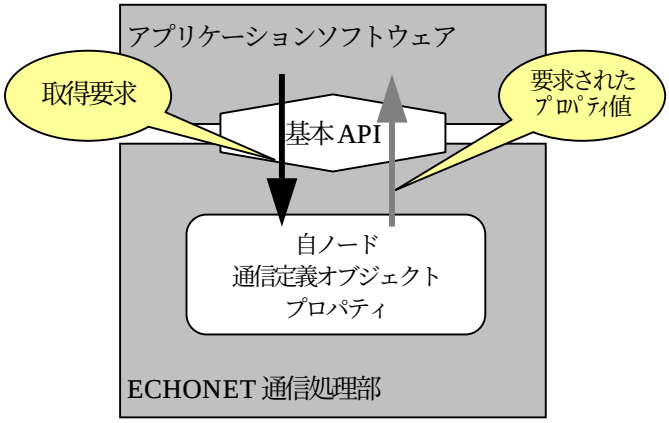


図３．１１

表３．１７ 自ノード内通信定義オブジェクトプロパティ値取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	objclass_id	通信定義オブジェクトプロパティの取得対象のオブジェクトのインスタンスの指定。	Required
Input	prop_id	プロパティ値取得対象のプロパティの指定。	Required
Output	comprop_info	指定されたプロパティに設定されている値の情報。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（１７）自ノード内通信定義オブジェクトプロパティ値設定要求取得

ECHONET 通信処理部に対して、他ノードから行われる自ノードの個々の機器オブジェクトの通信定義オブジェクトクラスのプロパティ値の設定要求の取得を行う。（他ノードからのプロパティ値の読出し要求は、通信ミドルウェアにて処理し、特にアプリケーションソフトウェアまでは上げないものとする。）他ノードからの設定を受けつけるプロパティの項目、内容等は、個々の通信定義オブジェクトのインスタンス毎に異なる（第２部参照）。他ノードからのプロパティ値の設定要求のアプリケーションソフトウェアでの取得は、アプリケーションソフトウェアからの通信ミドルウェアに対する呼び出しとして取得可能とするが、自動的に通知（イベントとして通知）される形式となっても構わないものとする。また、他ノードから、プロパティへ設定要求された値は、アプリケーションソフトウェアにてそのプロパティの実体が変化し、その旨が別途設定されるまでは、要求前の値を保持しておく（通信ミドルウェアではアプリケーションソフトウェアからの要求無しにプロパティ値の変更は行わない）。図３．１２に、本 API と ECHONET 通信処理部の関連を示し、表３．１８に入出力仕様を示す。

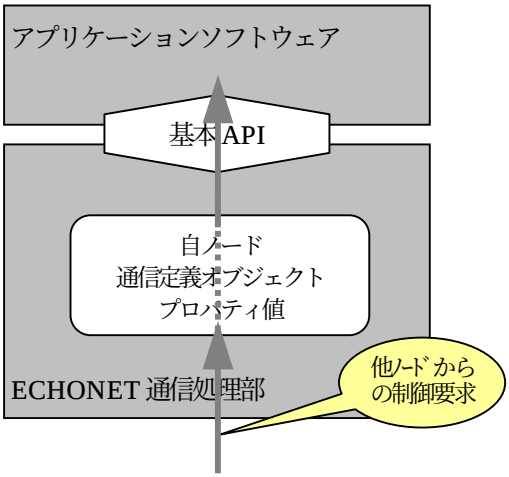


図３．１２

表３．１８ 自ノード内通信定義オブジェクトプロパティ値設定要求取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	objclass_id	他ノードからの通信定義オブジェクトプロパティ値の設定要求があったかどうかを確認する対象の、オブジェクトのインスタンスの指定。	Optional
Input	prop_id	他ノードからの通信定義オブジェクトプロパティ値の設定要求内容を確認するプロパティの指定。	Optional
Output	prop_info	制御要求対象の通信定義オブジェクトの情報。通信定義オブジェクト（プロパティ含む）指定情報、制御サービス情報、及び具体的な設定制御値情報含む。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（１８）他ノード内通信定義オブジェクトプロパティ値取得

ECHONET 通信処理部に対して、他ノードの個々の通信定義オブジェクトのインスタンスのプロパティ値の読み出し（取得）を行う。通信ミドルウェア上でモニタ管理している値を取得要求するケース（下図 CASE1）と、ECHONET を介して、現状の値を取得要求するケース（下図 CASE2）がある。後者のケース（CASE2）の場合、値の要求と実際の取得値の受け取りの同期は、特に規定しない。読み出し対象となるプロパティの項目、内容等は、個々の通信定義オブジェクトのインスタンス毎に異なる（第２部参照)。図３．１３に、本 API と ECHONET 通信処理部の関連を示し、表３．１９に入出力仕様を示す。

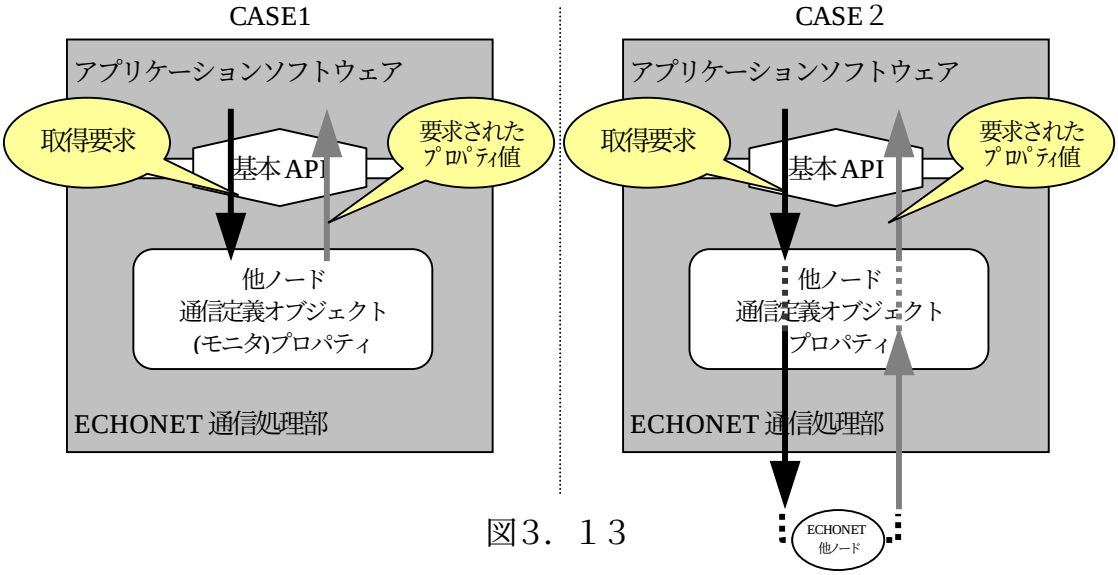


図３．１３

表３．１９ 他ノード内通信定義オブジェクトプロパティ値取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	プロパティ値の取得対象の通信定義オブジェクトのインスタンスの存在するノードの指定。	Required
Input	objclass_id	プロパティ値の取得対象のオブジェクトのインスタンスの指定。	Required
Input	prop_id	プロパティ値取得対象のプロパティの指定。	Required
Input	place_info	情報取得対象の場所（現在自ノード上で保持している情報か、他ノード上の情報かの指定）情報の指定。	Optional
Output	prop_info	指定されたプロパティに設定されている値の情報。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（１９）他ノード内通信定義オブジェクトプロパティ値通知取得

ECHONET 通信処理部に対して、他ノードが通知した個々の通信定義オブジェクトのインスタンスのプロパティ値の読み出し（取得）を行う。読出しのタイミングと、他ノードからの通知のタイミングの同期は、特に規定しない（非同期でも構わない）ものとする。図3．１4に、本API と ECHONET 通信処理部の関連を示し、表3．２0に入出力仕様を示す。

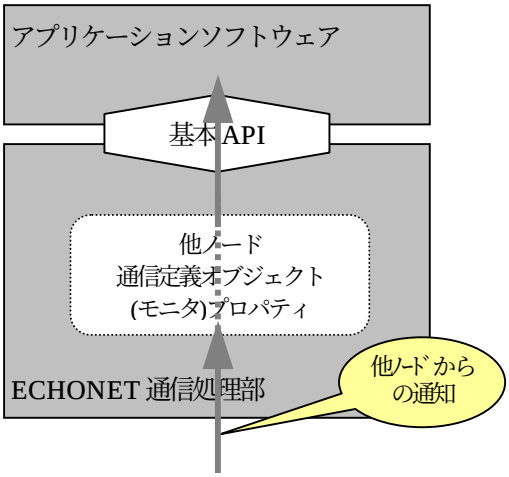


図3．１4

表3．２0 他ノード内通信定義オブジェクトプロパティ値取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	他ノードからの通信定義オブジェクトのプロパティ値の設定通知があったかどうかを確認するノードの指定。	Optional
Input	objclass_id	他ノードからの通信定義オブジェクトプロパティ値の設定通知があったかどうかを確認するオブジェクトのインスタンスの指定。	Optional
Input	prop_id	他ノードからの通信定義オブジェクトプロパティ値の設定通知があったかどうかを確認するプロパティの指定。	Optional
Output	prop_info	通知されたの通信定義オブジェクトのプロパティ値の情報（。通信定義オブジェクト（プロパティ含む）指定情報、制御サービス情報、及び具体的な設定制御値情報含む。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（２０）他ノード内通信定義オブジェクトプロパティ値設定要求

ECHONET 通信処理部に対して、他ノードの個々の通信定義オブジェクトのインスタンスのプロパティ値の設定要求を行う。他ノードの通信定義オブジェクトのインスタンスのプロパティ値の設定要求としては、設定要求結果の応答を要求しないケース（下図 CASE1）と、設定要求結果の応答を要求するケース（下図 CASE2）がある。設定要求と実際の設定結果の取得の同期は、特に規定しないが、並列処理できないマシン（CPU）上で動作するソフトの場合には非同期とすることが望ましい。設定・制御対象となるプロパティの項目、内容等、個々の機器オブジェクトのインスタンス毎に異なる（第２部参照）。図３．１５に、本 API と ECHONET 通信処理部の関連を示し、表３．２１に入出力仕様を示す。

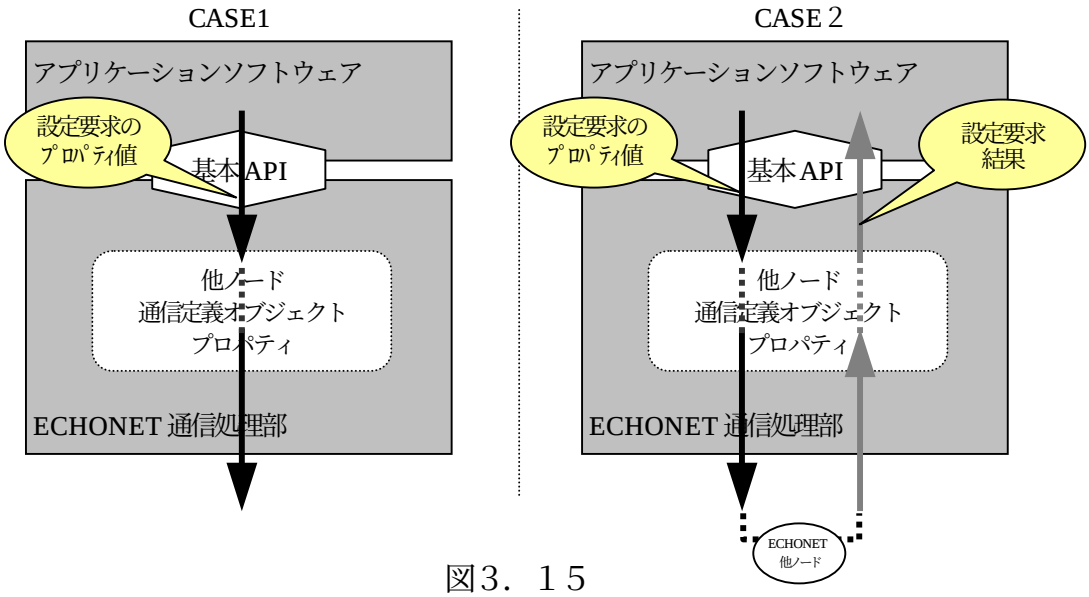


図３．１５

表３．２１ 他ノード内通信定義オブジェクトプロパティ値設定要求 API
入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	設定対象の通信定義オブジェクトのインスタンスのノードの指定。	Required
Input	objclass_id	設定対象の通信定義オブジェクトのインスタンスの指定を行う。	Required
Input	prop_id	設定対象のプロパティの指定を行う。	Required
Input	prop_info	指定されたプロパティに設定する値の情報。サービスの指定を含む。	Required
Output	res_info	設定結果の情報。	Optional
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（２１）自ノード内サービスオブジェクトプロパティ値設定・通知

ECHONET 通信処理部に対して、自ノードの個々のサービスオブジェクトのインスタンスのプロパティ値の設定及び、設定値の ECHONET 上ノードへの通知(Optional 機能)を行う。設定・通知対象となるプロパティの項目、内容等、個々のサービスオブジェクトインスタンス毎に異なる（第２部、第８部、第９部参照）。図３．１６に、本 API と ECHONET 通信処理部の関連を示し、表３．２２に入出力仕様を示す。

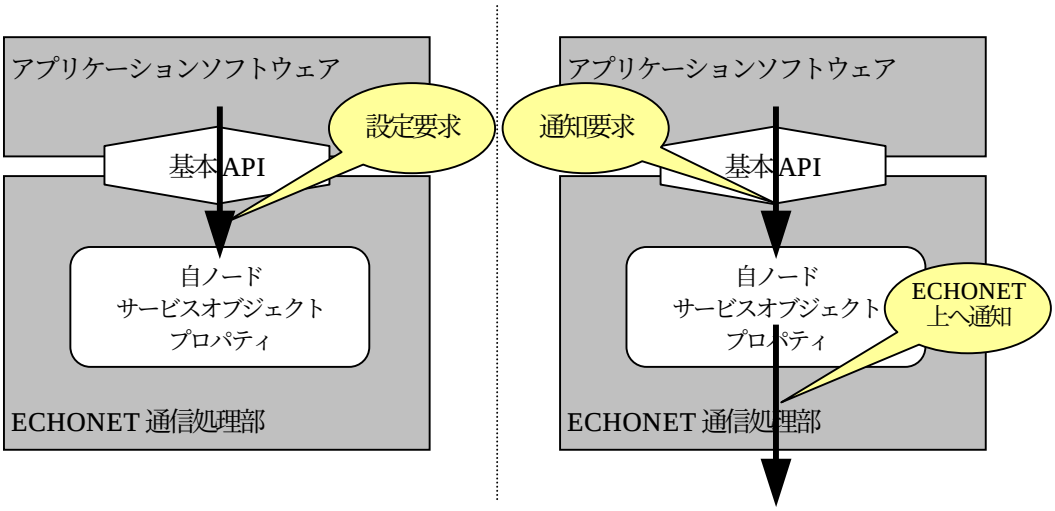


図３．１６

表３．２２ 自ノード内サービスオブジェクトプロパティ値設定・通知 API
入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	objclass_id	プロパティ値の設定・通知を行うサービスオブジェクトのインスタンスの指定。	Required
Input	prop_id	設定・通知対象のプロパティの指定。	Required
Input	announce_info	設定情報を ECHONET 上へ通知するかどうかの指定。通知する場合には通知先情報も含む。	Optional
Input	prop_info	プロパティ値の設定・通知情報。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（２２）自ノード内サービスオブジェクトプロパティ値取得

ECHONET 通信処理部に対して、自ノードの個々のサービスオブジェクトのインスタンスのプロパティ値の読み出し（取得）を行う。読み出し対象となるプロパティの項目、内容等、個々のサービスオブジェクトインスタンス毎に異なる（第2部、第8部、第9部参照)。図3．１7に、本 API と ECHONET 通信処理部の関連を示し、表3．２３に入出力仕様を示す。

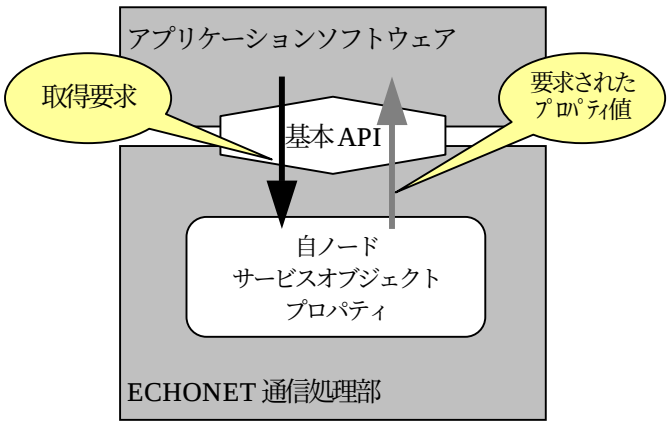


図3．１７

表3．２３ 自ノード内サービスオブジェクトプロパティ値取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	objclass_id	取得対象のプロパティ値のサービスオブジェクトのインスタンス指定。	Required
Input	prop_id	設定・通知対象のプロパティの指定。	Required
Output	prop_info	指定されたプロパティに設定されている値の情報。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（２３）自ノード内サービスオブジェクトプロパティ値設定要求取得

ECHONET 通信処理部に対して、他ノードからの自ノードの個々のサービスオブジェクトのインスタンスのプロパティ値の設定要求の取得を行う。(他ノードからのプロパティ値の読出し要求は、通信ミドルウェアにて処理し、特にアプリケーションソフトウェアまでは上げないものとする。)他ノードからの設定を受けつけるプロパティの項目、内容等は、個々のサービスオブジェクトのインスタンス毎に異なる（第２部、第８部、第９部参照)。他ノードからのプロパティ値の設定要求のアプリケーションソフトウェアでの取得は、アプリケーションソフトウェアからの通信ミドルウェアに対する呼び出しとして取得可能とするが、自動的に通知（イベントとして通知）される形式となっても構わないものとする。また、他ノードから、プロパティへ設定要求された値は、アプリケーションソフトウェアにてそのプロパティの実体が変化し、その旨が別途設定されるまでは、要求前の値を保持しておく（通信ミドルウェアではアプリケーションソフトウェアからの要求無しにプロパティ値の変更は行わない)。図３．１８に、本 API と ECHONET 通信処理部の関連を示し、表３．２４に入出力仕様を示す。

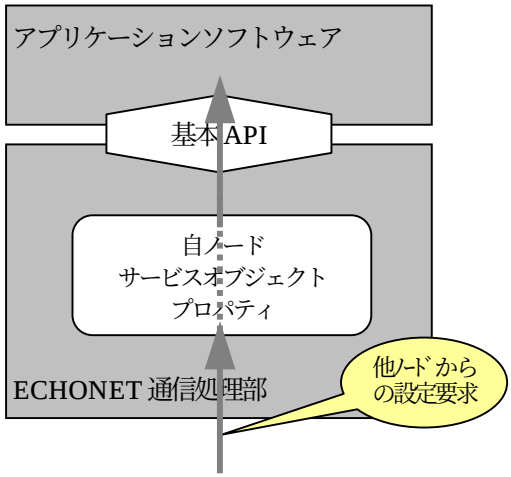


図３．１８

表３．２４ 自ノード内サービスオブジェクトプロパティ値設定要求取得
API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	objclass_id	他ノードからのプロパティ値の設定要求内容を確認する対象となるサービスオブジェクトの指定。	Optional
Input	prop_id	他ノードからのプロパティ値の設定要求内容を確認するプロパティの指定。	Optional
Output	prop_info	設定要求対象のサービスオブジェクトのプロパティ値。サービスオブジェクト（プロパティ含む）指定情報、制御サービス情報、及び具体的な設定制御値情報含む。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（２４）他ノード内サービスオブジェクトプロパティ値取得［MidGetOutAplData］

ECHONET 通信処理部に対して、他ノードの個々のサービスオブジェクトのインスタンスのプロパティ値の読み出し（取得）を行う。通信ミドルウェア上でモニタ管理している値を取得要求するケース（下図 CASE1）と、ECHONET を介して、現状の値を取得要求するケース（下図 CASE2）がある。後者のケース（CASE2）の場合、値の要求と実際の取得値の受け取りの同期は、特に規定しない。読み出し対象となるプロパティの項目、内容等、個々のサービスオブジェクトのインスタンス毎に異なる（第２部、第８部、第９部参照)。図３．１９に、本 API と ECHONET 通信処理部の関連を示し、表３．２５に入出力仕様を示す。

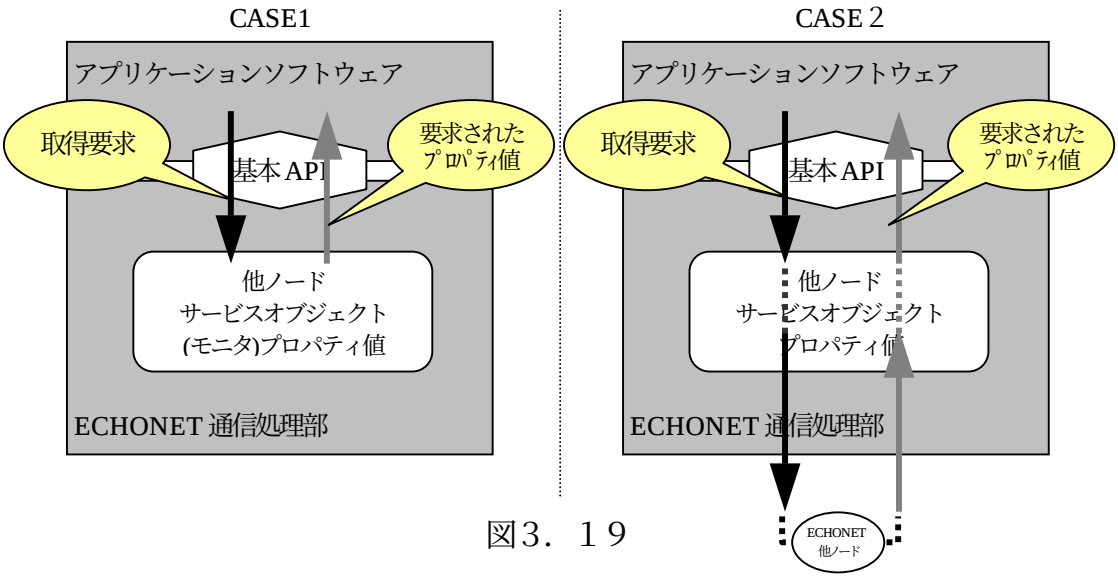


図３．１９

表３．２５ 他ノード内サービスオブジェクトプロパティ値取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	サービスオブジェクトプロパティ値の取得対象のノードの指定。	Required
Input	objclass_id	サービスオブジェクトプロパティ値の取得対象のオブジェクトのの指定。	Required
Input	prop_id	プロパティ値取得対象のプロパティの指定。	Required
Input	place_info	情報取得対象の場所（現在自ノード上で保持している情報か、他ノード上の情報かの指定）情報の指定。	Optional
Output	prop_info	指定されたプロパティに設定されている値の情報。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（２５）他ノード内サービスオブジェクトプロパティ値通知取得

ECHONET 通信処理部に対して、他ノードが通知した個々のサービスオブジェクトのインスタンスのプロパティ値の読み出し（取得）を行う。読出しのタイミングと、他ノードからの通知のタイミングの同期は、特に規定しない（非同期でも構わない）ものとする。図３．２０に、本 API と ECHONET 通信処理部の関連を示し、表３．２６に入出力仕様を示す。

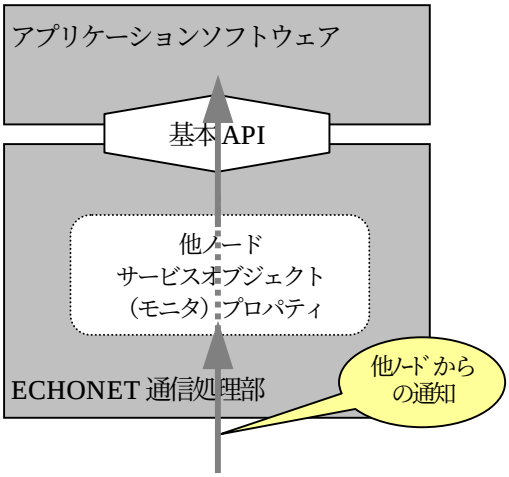


図３．２０

表３．２６ 他ノード内サービスオブジェクトプロパティ値取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	他ノードからのサービスオブジェクトのプロパティ値の設定通知があったかどうかを確認するノードの指定。	Optional
Input	objclass_id	他ノードからのサービスオブジェクトのプロパティ値の設定通知があったかどうかを確認するオブジェクトのインスタンスの指定。	Optional
Input	prop_id	他ノードからのサービスオブジェクトのプロパティ値の設定通知があったかどうかを確認するプロパティの指定。	Optional
Output	prop_info	通知されたのサービスオブジェクトのプロパティ値。サービスオブジェクト（プロパティ含む）指定情報、制御サービス情報、及び具体的な設定制御値情報含む。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（２６）他ノード内サービスオブジェクトプロパティ値設定要求

ECHONET 通信処理部に対して、他ノードの個々のサービスオブジェクトのインスタンスのプロパティ値の設定要求を行う。他ノードのサービスオブジェクトのインスタンスのプロパティ値の設定要求としては、設定要求結果の応答を要求しないケース（下図 CASE1）と、設定要求結果の応答を要求するケース（下図 CASE2）がある。設定要求と実際の設定結果の取得の同期は、特に規定しない。設定・制御対象となるプロパティの項目、内容等、個々のサービスオブジェクトのインスタンス毎に異なる（第２部、第８部、第９部参照)。図３．２１に、本 API と ECHONET 通信処理部の関連を示し、表３．２７に入出力仕様を示す。

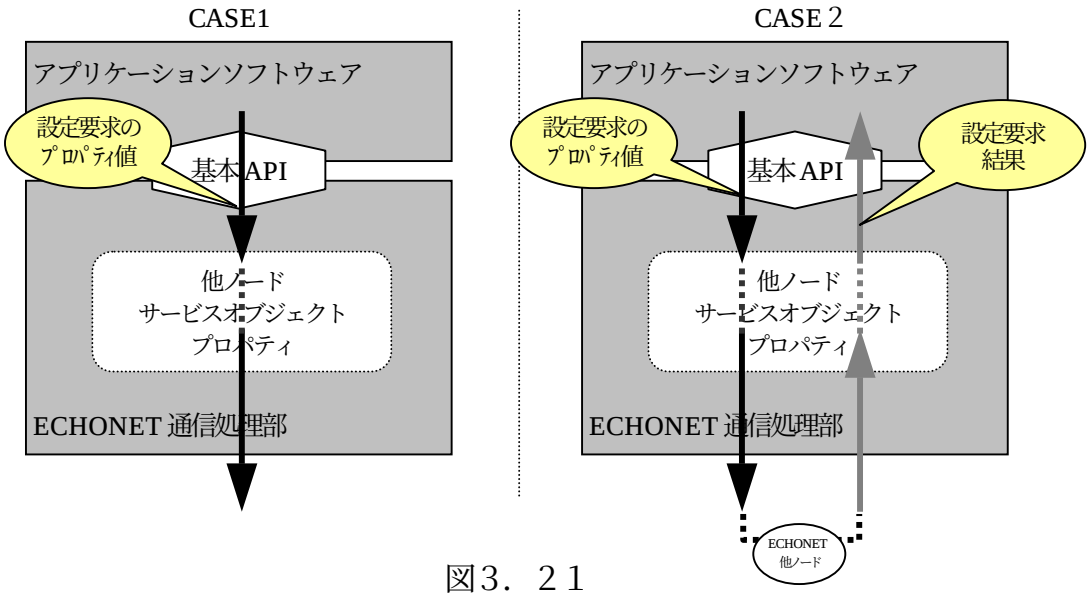


図３．２１

表３．２７ 他ノード内サービスオブジェクトプロパティ値設定要求 API
入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	設定対象のサービスオブジェクトのノードの指定。	Required
Input	objclass_id	設定対象のサービスオブジェクトのオブジェクトのインスタンスの指定。	Required
Input	prop_id	設定対象のプロパティの指定。	Required
Input	prop_info	指定されたプロパティに設定する値の情報。サービスの指定を含む。	Required
Output	res_info	設定結果の情報。	Optional
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（２７）管理オブジェクト追加

ECHONET 通信処理部に対して、管理している自ノード及び他ノードの各種オブジェクトのインスタンスの追加を、プロパティの単位で行う。表３．２８に入出力仕様を示す。

表３．２８ 管理オブジェクト追加 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	追加する管理対象のオブジェクトのインスタンスの存在するノードの指定。	Required
Input	objclass_id	追加する管理対象のオブジェクトのオブジェクトのインスタンスの指定。	Required
Input	prop_id	追加する管理対象のプロパティの指定。	Required
Input	prop_info	追加する管理対象のプロパティ値の指定。	Optional
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（２８）管理オブジェクト削除

ECHONET 通信処理部に対して、管理している自ノード及び他ノードの各種オブジェクトのインスタンスの削除を、プロパティの単位で行う。表３．２９に入出力仕様を示す。

表３．２９ 管理オブジェクト削除 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	削除する管理対象のオブジェクトのインスタンスの存在するノードの指定。	Required
Input	objclass_id	削除する管理対象のオブジェクトのオブジェクトのインスタンスの指定。	Required
Input	prop_id	削除する管理対象のプロパティの指定。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（２９）管理オブジェクト取得

ECHONET 通信処理部に対して、管理している自ノード及び他ノードの各種オブジェクトのインスタンスの取得を、プロパティの単位で行う。表３．３０に入出力仕様を示す。

表３．３０ 管理オブジェクト取得 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	enode_id	取得する管理対象のオブジェクトのインスタンスの存在するノードの指定。	Required
Output	objclass_id	取得する管理対象のオブジェクトのインスタンス。	Required
Output	prop_id	取得する管理対象のプロパティの指定。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時。	Optional

（３０）通信停止要求

ECHONET 通信ミドルウェア以下の通信処理部に通信停止状態への遷移を要求する。表３．３１に入出力仕様を示す。

表3．３１ 通信停止要求 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	device_id	通信停止対象の指示。 ECHONET 通信処理部、プロトコル差異吸収処理部、個々の下位通信ソフトウェアの識別が可能であること。	Optional
Output	Return Value	TRUE：通信停止受理、FALSE：受理不可。	Optional

（３１）完全停止要求

ECHONET 通信ミドルウェア以下の通信処理部に完全停止状態への遷移を要求する。表３．３２に入出力仕様を示す。

表3．３２ 完全停止要求 API 入出力データ一覧

方向	データ名	内容・条件	実装規定
Input	device_id	完全停止対象の指示。 ECHONET 通信処理部、プロトコル差異吸収処理部、個々の下位通信ソフトウェアの識別が可能であること。	Optional
Output	Return Value	TRUE：完全停止受理、FALSE：受理不可。	Optional

（３２）下位通信ソフトウェアアドレステーブルデータサイズ取得(Optional)

下位通信ソフトウェアで保持している下位アドレステーブルデータの組数の取得を行う。表3．３３に入出力仕様を示す。

表3．３３　下位通信ソフトウェアアドレステーブルデータサイズ取得 API 入出力データ一覧

方向	データ名	内容・条件	備考
Input	device_id	・ 下位通信ソフトウェア種別要求サービスにより取得した下位通信ソフトウェア ID を指定する。	Optional
Output	data_number	・下位アドレステーブルデータで保持しているアドレステーブルのデータの組数を示す。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時	Optional

（３２）下位通信ソフトウェアアドレステーブルデータ取得（Optional）

下位通信ソフトウェアで保持している下位アドレステーブルデータの取得を行う。データはデータ組数、各データ組はハードウェアアドレス、NodeID、及びマスタルータであることを示すフラグよりなるリストデータである。

表３．３４　下位通信ソフトウェアアドレステーブルデータ取得 API 入出力データ一覧

方向	データ名	内容・条件	備考
Input	device_id	・下位通信ソフトウェア種別要求サービスにより取得した下位通信ソフトウェア ID を指定する。	Optional
Output	data_number	・下位アドレステーブルデータで保持しているアドレステーブルのデータの組数を示す。	Required
Output	ListOfHardwareaddress	・下位アドレステーブルデータで保持しているアドレステーブルのハードウェアアドレスのリストを示す。	Required
Output	ListOfNode_id	・下位アドレステーブルデータで保持しているアドレステーブルの NodeID のリストを示す。	Required
Output	ListOfMasterRouter_Flag	・下位宛先アドレステーブルデータで保持している宛先アドレスに対応するノードがマスタルータであるか否かを示す識別子のリスト。マスタルータであれば 1、そうでなければ 0 となる。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時	Optional

（３３）マスタルータ通知（Optional）

通信ミドルウェアに対して、自ノードがマスタルータであるか否かを下位通信ソフトウェアに通知するように要求する。

表３．３５　マスタルータ通知 API 入出力データ一覧

方向	データ名	内容・条件	備考
Input	device_id	・下位通信ソフトウェア種別要求サービスにより取得した下位通信ソフトウェア ID を指定する。	Optional
Output	Return Value	TRUE：正常時、FALSE：異常時	Optional

（３５）ハードウェアアドレスデータ取得（Optional）

下位通信ソフトウェアに対して保持しているハードウェアアドレスデータの取得を行う。出力データはハードウェアアドレスである。

表3．３６　ハードウェアアドレスデータ取得 API 入出力データ一覧

方向	データ名	内容・条件	備考
Input	device_id	・ 下位通信ソフトウェア種別要求サービスにより取得した下位通信ソフトウェア ID を指定する。	Optional
Output	hardwareaddress	・ 下位アドレステーブルデータで保持しているハードウェアアドレスを示す。	Required
Output	Return Value	TRUE：正常時、FALSE：異常時	Optional

第４章 レベル２ECHONET 基本 API 仕様（C 言語用）

レベル２ECHONET 基本 API 仕様としては、周辺状況に鑑み、基本 API を利用して開発するアプリケーションの再利用性を考慮し、以下の言語を対象として、関数を規定する。今後、必要が出てきた段階で、他の言語についての規定も行うこととする。

（1）ANSI 規格の C 言語
（2）JAVA 言語

本章では、上記二つの言語の内、（1）用のレベル２ECHONET 基本 API 仕様を示す。レベル２の規定では、アプリケーションソフトウェア開発者からみて通信ミドルウェアの互換性を目的とすることから、関数の詳細まで規定する。C 言語用として規定する基本 API 関数は、以下を前提とする。また、本章に規定する以外の関数を設定して用いてはいけないということを規定するものではない。

・C 言語対応の８ビット～３２ビットのマイコン上での利用。
・Windows 及びμITRON といった OS 上での利用。

ECHONET 規格は、主として家庭における設備系機器（白物家電）をターゲットとしている規格であり、単純な機能を実現する機器上に搭載する場合でも、負荷が大きくなることなく搭載を実現できる必要がある。C 言語用レベル２ECHONET 基本 API 関数においては、以下の低レベル API 関数と高レベル API 関数が考えられる。現バージョンでは、アプリケーションソフトウェアの通信ミドルウェアに対する互換性を実現することを主眼とし、低レベル関数について詳細を規定する。高レベル関数については、今後、必要に応じて規定していくものとする。

・低レベル基本 API 関数（Required）
　第３章で規定の機能操作を、最も基本となるオブジェクト操作で実施する形式の関数。
・高レベル基本 API 関数
　（1）　第３章で規定の機能操作を、実際の操作対象を明示的に認識できる形での操作を実現する形式の関数。

以下、関数で共通に用いる定数規定を示し、低レベル基本 API 関数について関数一覧と詳細仕様を示す。

４．１ 各種定数仕様

本章で示す関数で、戻り値やデータ型などのラベルとして用いる定数の仕様を示す。以降の項では、本項で示したラベル名を関数詳細仕様の説明の中で用いる。ここで示す定数は、以下の７種類である。

（1）関数戻り値
（2）ID 種別
（3）ESV コード
（4）データ型
（5）アクセスルール
（6）通信ミドルウェア状態
（7）状変時アナウンス指定

尚、ラベル名称自体は参考提示とし、対応が明確であれば、別のラベル名としても構わないものとする。以下、それぞれの詳細を示す。

（1）関数戻り値

EAPI_NO_ERROR	：0(処理成功)
EAPI_SYSCALL	：1(システムコールエラー)
EAPI_NOMOREOPEN	：2(セッション数オーバー)
EAPI_NOTOPEN	：3(セッションが未 open 或いは、非起動中)
EAPI_ILLEGAL_PARAM	：4(パラメータ不正)
EAPI_NOTFOUND	：5(指定の対象が存在しない)
EAPI_NOTFOUND_NODE	：５0(管理機器が存在しない)
EAPI_NOTFOUND_OBJ	：５1(管理オブジェクトが存在しない)
EAPI_NOTFOUND_EPC	：５2(管理プロパティが存在しない)
EAPI_EXIST	：6(指定の対象が既存)
EAPI_EXIST_NODE	：６0(管理機器が既存)
EAPI_EXIST_OBJ	：６1(管理オブジェクトが既存)
EAPI_EXIST_EPC	：６2(管理プロパティが既存)
EAPI_EXIST_MEMBER	：６3(管理要素が既存)
EAPI_NORESOURCE	：7(リソース不足)
EAPI_NOCONDITION	：8(制御不可能)
EAPI_NODELETE	：9(削除不可能)
EAPI_TIMEOUT	：１0(通信タイムアウト)
EAPI_DATASIZE_EROR	：１1(データサイズエラー)
EAPI_NOTSEND	：１2(未送信データあり)
EAPI_MEMBER_EPC	：１3(配列要素プロパティ)
EAPI_NOTMEMBER_EPC	：１4(配列要素プロパティではない)
EAPI_NOTFOUND_MNO	：１5(配列要素が存在しない)
EAPI_MID_ERROR	：１6(ECHONET 通信処理部エラー)
EAPI_PRO_ERROR	：１7(プロトコル差異吸収処理部エラー)
EAPI_LOW_ ERROR	：１8(下位通信ソフトウェアエラー)
EAPI_NORECEIVE	：１9(受信データなし)
EAPI_ETC_ERROR	：２0(その他のエラー)

（2）ID 種別

APIVAL_NODE_KIND	：0(機器 ID)
APIVAL_EA_KIND	：1(ECHONET アドレス)
APIVAL_BROAD_KIND	：2(同報)

（3）ESV コード

ESV_SetI	：0x60(応答不要プロパティ値書き込み要求)
ESV_SetC	：0x61(応答要プロパティ値書き込み要求)
ESV_Get	：0x62(プロパティ値読み出し要求)
ESV_INF_REQ	：0x63(プロパティ値通知要求)

ESV_SetMI	：0x64(応答不要要素指定プロパティ値書込要求)
ESV_SetMC	：0x65(応答要要素指定プロパティ値書込要求)
ESV_GetM	：0x66(プロパティ値要素指定読み出し要求)
ESV_INFM_REQ	：0x67(プロパティ値要素指定通知要求)
ESV_AddMI	：0x68(応答不要プロパティ値要素指定追加要求)
ESV_AddMC	：0x69(応答要プロパティ値要素指定追加要求)
ESV_DelMI	：0x6A(応答不要プロパティ値要素指定削除要求)
ESV_DelMC	：0x6B(応答要プロパティ値要素指定削除要求)
ESV_CheckM	：0x6C(プロパティ値要素指定存在確認要求)
ESV_AddMSI	：0x6D(応答不要プロパティ値要素追加要求)
ESV_AddMSC	：0x6E(応答要プロパティ値要素追加要求)
ESV_Set_Res	：0x71(プロパティ値書き込み応答)
ESV_Get_Res	：0x72(プロパティ値読み出し応答)
ESV_INF	：0x73(プロパティ値通知)
ESV_INF_AREQ	：0x74(プロパティ値通知確認要求)
ESV_SetM_Res	：0x75(プロパティ値の要素指定書き込み応答)
ESV_GetM_Res	：0x76(プロパティ値の要素指定読み出し応答)
ESV_INFM	：0x77(プロパティ値の要素指定通知)
ESV_INFM_AREQ	：0x78(プロパティ値の要素指定通知確認要求)
ESV_AddM_Res	：0x79(プロパティ値の要素指定追加応答)
ESV_INF_Ares	：0x7A(プロパティ値通知確認応答)
ESV_DelM_Res	：0x7B(プロパティ値の要素指定削除応答)
ESV_CheckM_Res	：0x7C(プロパティ値の要素指定存在確認応答)
ESV_INFM_Ares	：0x7D(プロパティ値の配列指定通知確認応答)
ESV_AddMS_Res	：0x7E(プロパティ値の要素追加応答)
ESV_SetI_SNA	：0x50(プロパティ値書き込み要求不可応答)
ESV_SetC_SNA	：0x51(プロパティ値書き込み要求不可応答)
ESV_Get_SNA	：0x52(プロパティ値読み出し不可応答)
ESV_INF_SNA	：0x53(プロパティ値通知不可応答)
ESV_SetMI_SNA	：0x54(プロパティ値要素指定書き込み不可応答)
ESV_SetMC_SNA	：0x55(プロパティ値要素指定書き込み不可応答)
ESV_GetM_SNA	：0x56(プロパティ値要素指定読み出し不可応答)
ESV_INFM_SNA	：0x57 (プロパティ値要素指定通知不可応答)
ESV_AddMI_SNA	：0x58(プロパティ値要素指定追加不可応答)
ESV_AddMC_SNA	：0x59(プロパティ値要素指定追加不可応答)
ESV_DelMI_SNA	：0x5A(プロパティ値要素指定削除不可応答)
ESV_DelMC_SNA	：0x5B(プロパティ値要素指定削除不可応答)
ESV_CheckM_SNA	：0x5C(プロパティ値要素指定存在確認不可応答)
ESV_AddMSI_SNA	：0x5D(プロパティ値要素追加不可応答)
ESV_AddMSC_SNA	：0x5E(プロパティ値要素追加不可応答)

（4）データ型

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（データ型なし）

（5）アクセスルール

APIVAL_RULE_SET	：0x0001（Set）
APIVAL_RULE_GET	：0x0002（Get）
APIVAL_RULE_ANNO	：0x0040（Anno）
APIVAL_RULE_SETM	：0x0100（要素指定 Set）
APIVAL_RULE_GETM	：0x0200（要素指定 Get）
APIVAL_RULE_ADDM	：0x0400（要素指定追加要求）
APIVAL_RULE_DELM	：0x0800（要素指定削除要求）
APIVAL_RULE_CHECKM	：0x1000（要素指定存在確認要求）
APIVAL_RULE_ADDMS	：0x2000（要素追加要求）
APIVAL_RULE_ANNOM	：0x4000（要素指定通知要求）

（6）通信ミドルウェア状態

MID_STS_STOP	：0(停止中)
MID_STS_INIT	：1(初期化中、初期化処理終了)
MID_STS_RUN	：2(通常処理中)
MID_STS_APL_ERR	：3(アプリ異常)
MID_STS_PRO_ERR	：4(プロトコル差異吸収処理部異常)
MID_STS_LOW_ERR	：5(下位通信ソフトウェア異常)

（7）状態変化時アナウンス指定

APIVAL_ANNO_ON	：1（アナウンスあり）
APIVAL_ANNO_OFF	：0（アナウンスなし）

４．２ 低レベル基本 API 関数一覧

本稿で示す関数は、他の関数グループと異なり、制御対象オブジェクトを明示的には示すことなく全て同レベルで制御を実現するものである。ECHONET 通信ミドルウェアの動作を熟知し、ECHONET オブジェクトに精通した専門家であれば、本関数グループの関数だけで、必要な制御は全て実現可能である。
本グループの関数は、操作性は専門的になるが、少ない関数で ECHONET オブジェクトを全て制御できる。

表４．１ C 言語用レベル２　低レベル基本 API 関数一覧（１／２）

No	関数名	名称	補足
1	MidOpenSession	ECHONET 通信処理部動作開始要求関数	Optional
2	MidCloseSession	ECHONET 通信処理部動作終了要求関数	Optional
3	MidSetEA	ECHONET アドレス設定関数	Optional
4	MidGetEA	ECHONET アドレス設定値取得関数	Optional
5	MidGetNodeID	機器 ID の値の取得関数	Optional
6	MidSetControlVal	ECHONET 通信ミドル動作情報設定	Optional
7	MidGetControlVal	ECHONET 通信ミドル動作情報取得	Optional
8	MidSetSendEpc	ECHONET オブジェクトの配列でないプロパティに対応した送信要求関数	Required
	MidExtSetSendEpc		Optional
9	MidSetEpc	ECHONET オブジェクトの配列でないプロパティへのデータ書込要求関数	Required
	MidExtSetEpc		Optional
10	MidGetReceiveEpc	ECHONET オブジェクトの配列でないプロパティからのデータ読出要求関数(1)	Required
	MidExtGetReceiveEpc		Optional
11	MidGetEpc	ECHONET オブジェクトの配列でないプロパティからのデータ読出要求関数(2)	Required
12	MidSetSendCheckEpc	ECHONET オブジェクトの配列でないプロパティへのデータ書込確認関数	Required
	MidExtSetSendCheckEpc		Optional
13	MidSetSendEpcM, MidExtSetSendEpcM	ECHONET オブジェクトの配列のプロパティに対応した送信要求関数	Optional
14	MidSetEpcM, MidExtSetEpcM	ECHONET オブジェクトの配列のプロパティへのデータ書込要求関数	Optional
15	MidGetReceiveEpcM, MidExtGetReceiveEpcM	ECHONET オブジェクトの配列のプロパティからのデータ読出要求関数(1)	Optional
16	MidGetEpcM	ECHONET オブジェクトの配列のプロパティからのデータ読出要求関数(2)	Optional

表４．１ Ｃ言語用レベル２　低レベル基本 API 関数一覧（２／２）

No	関数名	名称	補足
17	MidSetSendCheckEpcM MidExtSetSendCheckEpcM	ECHONET オブジェクトの配列のプロパティへのデータ書込確認関数	Optional
18	MidGetReceiveCheckEpc MidExtGetReceiveCheckEpc	ECHONET プロパティのデータ読出確認関数	Optional
19	MidGetEpcSize	ECHONET プロパティのサイズ取得関数	Optional
20	MidGetEpcAttrib	ECHONET オブジェクトのプロパティの属性を取得する。	Optional
21	MidGetEpcMember	ECHONET オブジェクトの配列のプロパティの配列要素情報取得関数	Optional
22	MidCreateNode	管理機器追加作成関数	Optional
23	MidCreateObj	ECHONET オブジェクトの追加作成関数	Optional
24	MidCreateEpc	配列でないECHONETプロパティの追加作成関数	Optional
25	MidCreateEpcM	配列の ECHONET プロパティの追加作成関数	Optional
26	MidAddEpcMember	配列の ECHONET プロパティの要素追加（要素番号指定有）関数	Optional
27	MidAddEpcMemberS	配列の ECHONET プロパティの要素追加（要素番号指定無）関数	Optional
28	MidDeleteNode	管理機器削除関数	Optional
29	MidDeleteObj	ECHONET オブジェクトの削除関数	Optional
30	MidDeleteEpc	ECHONET プロパティ削除関数	Optional
31	MidDeleteEpcM	配列のECHONETプロパティの指定要素削除関数	Optional
32	MidGetState	ECHONET 通信処理部状態取得関数	Optional
33	MidSetRecvTargetList	データ受信通知対象リストの有効無効設定関数	Optional
34	MidAddRecvTargetList	データ受信通知対象リストの追加関数	Optional
35	MidDeleteRecvTargetList	データ受信通知対象リストの削除関数	Optional
36	MidGetRecvTargetList	データ受信通知対象リストの取得関数。	Optional
37	MidStart	ECHONET 通信処理部初期化関数	Optional
38	MidReset	ECHONET 通信処理部初期化関数	Optional
39	MidInit	ECHONET 通信処理部初期化関数	Required
40	MidInitAll	ECHONET 通信処理部初期化関数	Optional
41	MidRequestRun	ECHONET 通信処理部動作開始関数	Required
42	MidSuspend	ECHONET 通信処理部一時停止要求関数	Optional
43	MidWakeUp	ECHONET 通信処理部動作再開要求関数	Optional
44	MidSetSendMulti, MidExtSetSendMulti	ECHONET オブジェクトの配列でないプロパティに対応した送信要求関数　(複数プロパティ制御に対応)	Optional
45	MidGetReceiveEpcMulti	ECHONET オブジェクトの配列でないプロパティからのデータ読出要求関数(3)　(複数プロパティ制御に対応)	Optional
46	MidSetSecureContVal	セキュア通信用データ設定関数	Optional
47	MidStop	ECHONET 通信停止要求関数	Optional
48	MidHalt	ECHONET 完全停止要求関数	Optional
49	MidGetAddressTableDataSize	下位通信ソフトウェアアドレステーブルデータサイズ取得関数	Optional
50	MidGetAddressTableData	下位通信ソフトウェアアドレステーブルデータ取得関数	Optional

51	MidSetMasterRouterFlag	マスタルータ通知関数	Optional
52	MidGetHardwareAddress	ハードウェアアドレスデータ取得関数	Optional
53	MidGetReceiveCheckEpcMulti	複数のECHONETプロパティのデータ読出確認関数	Optional
54	MidGetDevID	下位通信ソフトウェア搭載情報要求関数	Optional
55	MidGetLastSendError	最新送信エラー情報取得関数	Optional

４．３ 低レベル基本 API 関数詳細仕様

本節では、表４．１で示した各関数の詳細仕様を、以下の７つの項目について示す。

（1）名称
　　関数名称を示す。
（2）機能
　　機能を説明する。
（3）構文
　　関数の構文を示す。
（4）説明
　　引数や、変数の詳細仕様について説明する。
（5）戻り値
　　戻り値を示す。
（6）使用する構造体
　　構造体があれば、構造体の仕様を示す。
（7）注意事項・制限事項
　　注意事項や制限事項があれば、示す。

注）詳細仕様中の「node_id」は、第２部の ECHONET アドレスで示した「NodeID」とは異なる。本部で記述している「node_id」は、ミドルウェア内部で管理するノード（機器）を識別するための ID（「機器 ID」）である。

４．３．１ MidOpenSession

（1）名称

MidOpenSession　－　ECHONET 通信処理部動作開始要求関数

（2）機能

通信ミドルのセッションを open する。

（3）構文

long MidOpenSession(short MidNo)

（4）説明［Optional 関数］

midNo で指定された通信ミドルウェアとのセッションを開始する。
通信ミドルがコンピュータ上に１つしか存在しないときは、midNo は常に０を指定する。コンピュータ上に通信ミドルが複数存在するときは、midNo でセッションを開く通信ミドルを指定する。
MidInit 関数を使用するか、または、なんらかの方法で通信ミドルを起動しておく形で使用する。MidInit 関数以外の API 関数を使用する前に必ずこの関数を呼び出す形で使用する。

MidNo　：[in]通信ミドルウェアＮｏ

（5）戻り値

EAPI_NO_ERROR	：open 成功
EAPI_SYSCALL	：ECHONET 通信処理部未起動
EAPI_NOMOREOPEN	：セッション数オーバー

（6）使用する構造体

特に無し。

（7）注意事項・制限事項

セッション open 済みのときもう一度このコールを実行すると、前のセッションは自動的に close される。

４．３．２ MidCloseSession

（1）名称

MidCloseSession　－　ECHONET 通信処理部動作終了要求関数

（2）機能

open 済みの通信ミドルのセッションを close

（3）構文

long MidCloseSession(void)

（4）説明［Optional 関数］

現在 open 中のセッションを終了し、通信ミドルとの通信用資源を全て解放する。通常、DLL がプロセスからデータッチされる時にこの処理が行われる為、特にこの関数を呼び出す必要はないが、ならかの理由でセッションを明示的に終了したい時は、この関数が呼び出される。

（5）戻り値

EAPI_NO_ERROR	：クローズ成功
EAPI_NOTOPEN	：非起動（セッションが未 open）

（6）使用する構造体

特に無し。

（7）注意事項・制限事項

特に無し。

４．３．３ MidSetEA

（1）名称

MidSetEA　－　ECHONET アドレス設定関数

（2）機能

自ノードの ECHONET アドレス及び、自ノード上で管理している他機器の ECHONET アドレスを設定する。

（3）構文

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

（4）説明［Optional 関数］

自ノードの node_id の値は 0 とする。それ以外は、ECHONET 通信処理部で管理している他機器を示す。本関数は、あくまで自ノード上のデータ操作。ECHONET アドレスを設定したい任意のタイミングで呼び出し可能。

node_id　：[in]機器 ID

dev_id　：[in]下位通信ソフトウェア ID
（自ノードの時のみ有効。下位媒体が一種類の場合は、０とする。）

ea　：[in]設定 ECHONET アドレス

（5）戻り値

EAPI_NO_ERROR	：設定成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_ILLEGAL_PARAM	：node_id 又は、dev_id が不正

（6）使用する構造体

特に無し。

（7）注意事項・制限事項

特に無し。

４．３．４ MidGetEA

（1）名称

MidGetEA　－　ECHONET アドレス設定値取得関数

（2）機能

設定されている ECHONET アドレスを取得する。

（3）構文

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

（4）説明［Optional 関数］

ECHONET 通信処理部で管理している自機器／他機器の ECHONET アドレスの設定値の取得を行う。（あくまで自ノード上のデータ操作。）

ECHONET アドレスを取得したい任意のタイミングで呼び出し可能。

node_id　：[in]機器 ID

dev_id　：[in]下位通信ソフトウェア ID

（自機器の場合のみ有効、下位媒体が一種類の場合は、０とする）

ea　：[out]取得 ECHONET アドレス

（5）戻り値

EAPI_NO_ERROR　：設定成功

EAPI_NOTOPEN　：非起動（セッションが未 open）

EAPI_ILLEGAL_PARAM　：node_id が不正

（6）使用する構造体

特に無し。

（7）注意事項・制限事項

特に無し。

４．３．５ MidGetNodeID

（1）名称

MidGetNodeID　－　機器 ID 値取得関数

（2）機能

機器 ID を取得する。

（3）構文

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

（4）説明［Optional 関数］

指定した ECHONET アドレスが設定されている機器 ID の取得を行う。
自機器で、下位媒体が複数搭載されている場合は下位通信ソフトウェア ID も取得する。機器 ID、下位通信ソフトウェア ID を取得したい任意のタイミングで呼び出し可能。

ea　　：[in]ECHONET アドレス
node_id　　：[out]機器 ID 格納エリア
dev_id　　：[out]下位通信ソフトウェア ID 格納エリア
（自機器の場合のみ有効、下位媒体が一種類の場合は、0が格納される）

（5）戻り値

EAPI_NO_ERROR　　：取得成功
EAPI_NOTOPEN　　：非起動（セッションが未 open）
EAPI_ILLEGAL_PARAM ：ea が不正

（6）使用する構造体

特に無し。

（7）注意事項・制限事項

特に無し。

４．３．６ MidSetControlVal

（1）名称

SetMidControlVal　－　ECHONET 通信処理部動作情報設定関数

（2）機能

通信ミドルウェアの動作情報を設定する。

（3）構文

long MidSetControlVal(MidControl *m_data)

（4）説明［Optional 関数］

起動中の通信ミドルウェアの動作情報の設定を行う。情報を設定したい任意のタイミングで呼び出し可能。

m_data　　：[in]通信ミドルウェア動作情報取得エリア

（5）戻り値

EAPI_NO_ERROR　　：設定成功
EAPI_NOTOPEN　　：非起動（セッションが未 open）
EAPI_ILLEGAL_PARAM　　：データ内容が不正

（6）使用する構造体

```
typedef   struct {
            short    sync;          /* 各サービス送信関数同期モード
                                0:非同期(通信の終了を待たずに関数からリターンする。
                                  実際の送信処理の完了は ObjWriteCheck() または
                                  ObjWriteCheckM()で、送信可能になることで知る。）
                                1：同期（送信の完了を待って関数からリターンする）
                                2：同期２（応答要のサービスの場合、応答の終了を待って関数からリターンする）*/
            short    sync_timer;    /* 同期タイムアウト値
                                （sync が０以外の場合有効・100ms 単位）
                                  ０の場合は、非同期とする。 */
} MidControl;
```

（7）注意事項・制限事項

未設定の場合の初期値は以下の通りとなる。

sync　　：0(非同期)
sync_timer　　：0

４．３．７ MidGetControlVal

（1）名称

MidGetControlVal　－　ECHONET 通信処理部動作情報取得関数

（2）機能

通信ミドルウェアウェアの動作情報を取得する。

（3）構文

long MidGetControlVal(MidSetup *midset)

（4）説明［Optional 関数］

起動中の通信ミドルウェアの動作情報の取得を行う。情報を取得したい任意のタイミングで呼び出し可能。

midset　：[out]動作情報取得エリア

（5）戻り値

EAPI_NO_ERROR　：取得成功
EAPI_NOTOPEN　：非起動（セッションが未 open）

（6）使用する構造体

```
typedef   struct {
            short    sync;          /* 各サービス送信関数同期モード */
            short    sync_timer;    /* 通信同期タイムアウト値 */
} MidControl;
```

（7）注意事項・制限事項

特に無し。

４．３．８ MidSetSendEpc，MidExtSetSendEpc

（1）名称

MidSetSendEpc, MidExtSetSendEpc　－　ECHONET オブジェクトのプロパティに対応した送信要求関数

（2）機能

ECHONET プロパティにデータを書き込み、サービスを送信する。

（3）構文

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

（4）説明［MidExtSetSendEpc は、Optional 関数］

MidSetSendEpc は、id,eoj_code,epc_code によって指定された ECHONET プロパティにデータを書き込み、esv_code のサービスを送信する。
データを書き込みたい任意のタイミングで呼び出し可。
MidExtSetSendEpc も、基本的には MidSetSendEpc と同じ機能を持つが、書き込みデータに関するセキュア通信等拡張された設定が行える関数である。

id_kind　：[in]ID 種別
- APIVAL_NODE_KIND　:０(機器 ID)
- APIVAL_EA_KIND　:１(ECHONET アドレス)
- APIVAL_BROAD_KIND　:２(同報)

id　：[in]機器 ID または ECHONET アドレスまたは同報種別

seoj_code　：[in]SEOJ コード (下位 3byte のみ使用）
- SEOJ なしの場合は、-1 を設定

deoj_code　：[in]DEOJ コード（下位 3byte のみ使用）
- DEOJ なしの場合は、-1 を設定

epc_code　：[in]EPC コード(下位 1byte のみ使用）

esv_code　：[in]ESV コード
- ESV_SetI　：0x60(応答不要プロパティ値書き込み要求)
- ESV_SetC　：0x61(応答要プロパティ値書き込み要求)
- ESV_Get　：0x62(プロパティ値読み出し要求)
- ESV_Inf_Req　：0x63(プロパティ値通知要求)
- ESV_INF　：0x73(プロパティ値通知)
- ESV_INF_AREQ　：0x74(プロパティ値通知確認要求)

data　：[in]データ内容へのポインタ

size　：[in]データサイズ

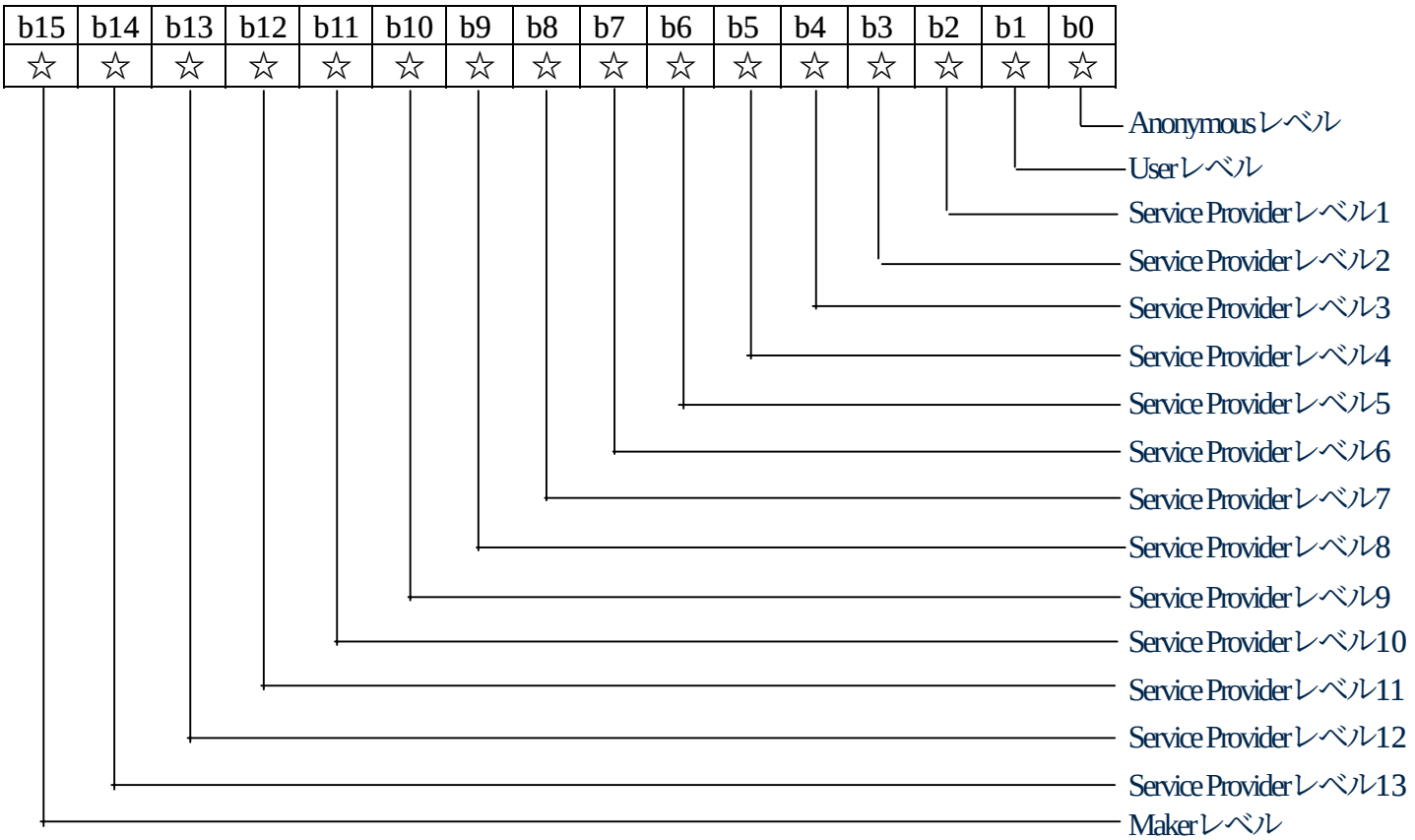
extcont　：[in]セキュア通信オプション

（5）戻り値

EAPI_NO_ERROR　　：設定成功
EAPI_NOTOPEN　　：非起動（セッションが未 open）
EAPI_ILLEGAL_PARAM　　：id_kind もしくは esv_code が不正
EAPI_NOTFOUND_EPC　　：プロパティが存在しない
EAPI_DATASIZE_EROR　　：書き込みデータサイズが不正
EAPI_NORESOURCE　　：リソース不足、
id_kind が EA_KIND・BROAD_KIND の時のみ
EAPI_NOCONDITION　　：制御不可能なプロパティ
EAPI_MEMBER_EPC　　：配列要素プロパティ
EAPI_NOTSEND　　：未送信データあり
EAPI_TIMEPOUT　　：通信タイムアウト（通信モードが同期の時）
EAPI_ETC_ERROR　　：指定された拡張通信機能の処理不可

（6）使用する構造体

```
typedef struct {
        short   ext_hed;    /* この構造体の種類を示すコード。
                            0x0001：セキュア通信指定*/
        short   cipher;         /* 暗号化指定（方式指定含）。
                            0x0000：暗号化無し
                            0x0001：AES-CBC
                            0x0002～0xFFFF：for future reserved */
        short   authent;        /* アクセス制限レベル指定
```



```
                                その他：for future reserved */
        short   authentication          /* 認証有無指定 */
        long    makerKeyIndex           /* メーカ KeyIndex */
        short   makerKeysize            /* メーカ Key サイズ*/
        char    makerKey                /*  メーカ Key  格納エリア*/
    } EXT_CONT
```

（7）注意事項

特に無し。

４．３．９ MidSetEpc，MidExtSetEpc

（1）名称

MidSetEpc, MidExtSetEpc　－　ECHONET オブジェクトのプロパティへのデータの書込要求関数

（2）機能

ECHONET プロパティにデータを書き込む。

（3）構文

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

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

（4）説明［MidExtSetEpc は、Optional 関数］

MidSetEpc は、id,eoj_code,epc_code によって指定された ECHONET プロパティにデータを書き込む。データを書き込みたい任意のタイミングで呼び出し可。
自機器への書き込みデータが前回のデータと変わった時で、状態変化通知処理設定のある場合のみ状態通知サービスを行なう。
MidExtSetEpc も、基本的には MidSetEpc と同じ機能を持つが、状態変化通知処理設定のある場合のみ状態通知サービスを行なう際に、外部への通知データに関するセキュア通信等拡張された設定が行える関数である。

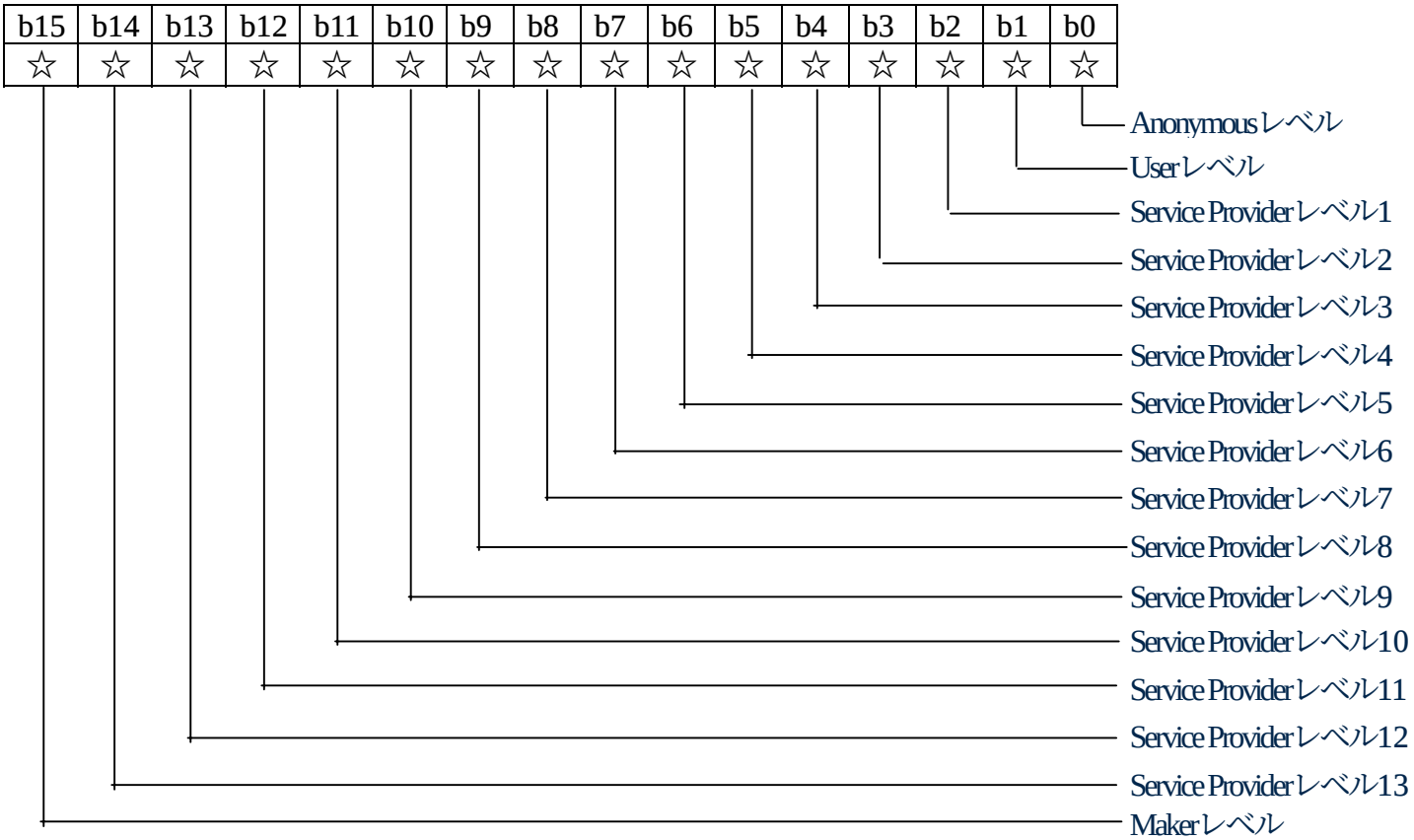
id_kind	：[in]ID 種別
APIVAL_NODE_KIND	:０(機器 ID)
APIVAL_EA_KIND	:１(ECHONET アドレス)
id	：[in]機器 ID または ECHONET アドレス
eoj_code	：[in]EOJ コード(下位 3byte のみ使用)
epc_code	：[in]EPC コード(下位 1byte のみ使用)
data	：[in]設定データ
size	：[in]データサイズ
extcont	：[in]セキュア通信オプション

（5）戻り値

EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NOTFOUND_EPC	：プロパティが存在しない
EAPI_MEMBER_EPC	：配列要素プロパティ
EAPI_DATASIZE_EROR	：データサイズが不正
EAPI_ILLEGAL_PARAM	：id_kind が不正
EAPI_ETC_ERROR	：指定された拡張通信機能の処理不可

（6）使用する構造体

```
typedef struct {
        short   ext_hed;        /* この構造体の種類を示すコード。
                                0x0001：セキュア通信指定*/
        shot    cipher;         /* 暗号化指定（方式指定含）。
                                0x0000：暗号化無し
                                0x0001：AES-CBC
                                0x0002～0xFFFF：for future reserved */
        short   authent;        /* アクセス制限レベル指定
```




```
                                その他：for future reserved */
        short   authentication          /* 認証有無指定 */
        long    makerKeyIndex           /* メーカ KeyIndex */
        short   makerKeysize            /* メーカ Key サイズ */
        char    makerKey                /* メーカ Key */
} EXT_CONT
```

（7）注意事項

特になし

４．３．１０ MidGetReceiveEpc, MidExtGetReceiveEpc

（1）名称

MidGetReceiveEpc, MidExtGetReceiveEpc－　ECHONET オブジェクトのプロパティからのデータの読出要求関数（1）

（2）機能

受信のあった ECHONET プロパティのデータを読み込む。

（3）構文

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

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

（4）説明［MidExtGetReceiveEpc は、Optional 関数］

MidGetReceiveEpc は、id,eoj_code,epc_code によって指定された ECHONET プロパティの受信データを読み込む。受信データを読み込みたい任意のタイミングで呼び出し可能。

MidExtGetReceiveEpc も、基本的には MidGetReceiveEpc と同じ機能を持つが、セキュア通信等拡張された設定のデータの読み出しに関する処理が行える関数である。

id_kind　：[in]ID 種別
　　APIVAL_NODE_KIND　　：0(機器 ID)
　　APIVAL_EA_KIND　　：1(ECHONET アドレス)
id　　：[in]機器 ID または ECHONET アドレス
eoj_code　：[in] EOJ コード (下位 3byte のみ使用、ない場合は－1)
（未解析のセキュア通信電文等 拡張電文要求の場合、－１となる。）
epc_code　：[in]EPC コード(下位 1byte のみ使用、ない場合は－1)
　　（未解析のセキュア通信電文等 拡張電文要求の場合、－１となる。）
buff_size　：[in]エリアサイズ
esv_code　：[out] ESV コード格納エリア(下位 1byte のみ使用、ない場合は－1)
　　（未解析のセキュア通信電文等 拡張電文要求の場合、－１となる。）
data　　：[out]データ内容格納エリア
data_size　：[out]読み込みデータサイズ
eoj_code2　：[out] 通信上の SEOJ コードまたは DEOJ コード
　　下位 3byte のみ使用、ない場合は－1
　　（本 eoj_code2 がある場合には、eoj_code が他 Node の
　　EOJ を指定する場合に通信上の DEOJ コードとなり、
　　eoj_code が自 Node の EOJ を指定する場合に通信上の
　　SEOJ コードを示す。）

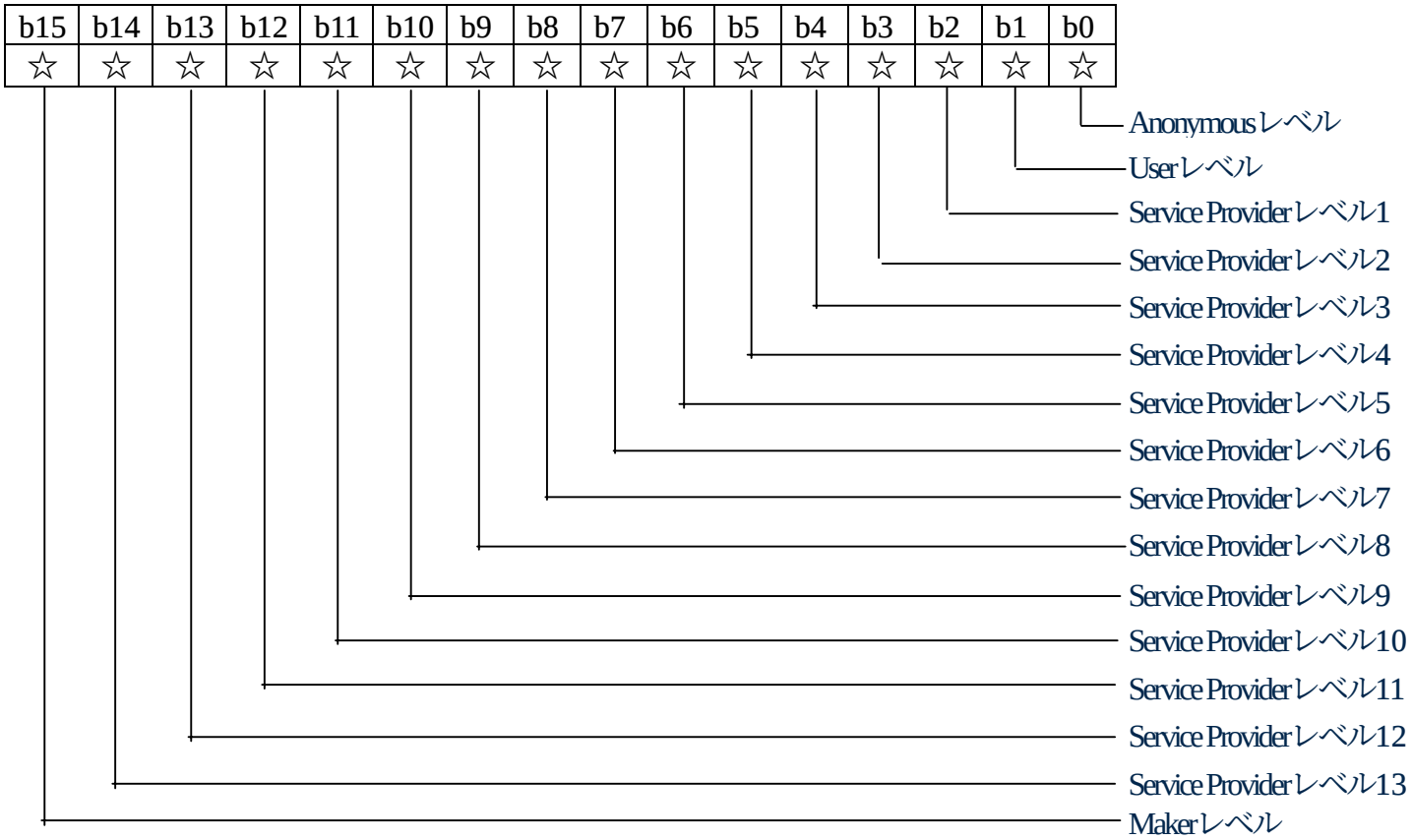
extcont　：[out]拡張通信オプション

（5）戻り値

EAPI_NO_EROR　：読み込み成功
EAPI_NOTOPEN　：非起動（セッションが未 open）
EAPI_ILLEGAL_PARAM　：id_kind が不正
EAPI_NOTFOUND_EPC　：プロパティが存在しない
EAPI_NORECEIVE　：受信データが存在しない
EAPI_MEMBER_EPC　：配列要素プロパティ
EAPI_DATASIZE_EROR　：データサイズが不正

（6）EAPI_ETC_ERROR　：指定された拡張通信機能の処理不可使用する構造体

```
typedef struct {
        short   ext_hed;    /* この構造体の種類を示すコード。
                            0x0001：セキュア通信指定*/
        short   cipher;     /* 暗号化指定（方式指定含）。
                            0x0000：暗号化無し
                            0x0001：AES-CBC
                            0x0002～0xFFFF：for future reserved */
        short   authent;    /* アクセス制限レベル指定
```




```
                その他：for future reserved */
        short   authentication      /* 認証有無指定 */
```

```
        long    makerKeyIndex       /* メーカ KeyIndex */
        short   makerKeysize        /* メーカ Key サイズ  */
        char    makerKey            /*  メーカ Key */
    } EXT_CONT
```

（7）注意事項

特に無し。

４．３．１１ MidGetEpc

（1）名称

MidGetEpc－　ECHONET オブジェクトのプロパティからのデータの読出要求関数（2）

（2）機能

受信に無関係に、ECHONET プロパティからのデータ読み込み要求。

（3）構文

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

（4）説明

ECHONET 通信処理部が管理している、id,eoj_code,epc_code によって指定された ECHONET プロパティの現状状態を取得する。状態を読み込みたい任意のタイミングで呼び出し可能。受信の有無に関わらず現在の状態を取得することができる。

id_kind　：[in]ID 種別
　　APIVAL_NODE_KIND　　：0(機器 ID)
　　APIVAL_EA_KIND　　：1(ECHONET アドレス)
id　　：[in]機器 ID または ECHONET アドレス
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)
epc_code　：[in]EPC コード(下位 1byte のみ使用)
buff_size　：[in]エリアサイズ
data　　：[out]データ内容格納エリア
data_size　：[out]読み込みデータサイズ

（5）戻り値

EAPI_NO_EROR	：読み込み成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NOTFOUND_EPC	：プロパティが存在しない
EAPI_MEMBER_EPC	：配列要素のプロパティ
EAPI_ILLEGAL_PARAM	：id_kind が不正
EAPI_NOCONDITION	：制御不可能なプロパティ
EAPI_DATASIZE_EROR	：データサイズエラー

（6）使用する構造体

特認無し。

（7）注意事項

特になし。

４．３．１２ MidSetSendCheckEpc，MidExtSetSendCheckEpc

（1）名称

MidSetSendCheckEpc　－　ECHONET オブジェクトのプロパティへのデータの書込確認関数

（2）機能

ECHONET プロパティへのデータ書込みの確認。

（3）構文

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

（4）説明

id,eoj_code,epc_code によって指定された ECHONET プロパティにデータを書き込めるかをチェックする。データを書き込めるかチェックしたい任意のタイミングで呼び出し可能、
書き込み不可の場合は、前回書き込んだ内容が未送信の場合も含むものとする。

id_kind　：[in]ID 種別
　　APIVAL_NODE_KIND　　：0(機器 ID)
　　APIVAL_EA_KIND　　：1(ECHONET アドレス)
id　　：[in]機器 ID または ECHONET アドレス
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)
epc_code　：[in]EPC コード(下位 1byte のみ使用)
extcont　：[in]拡張通信オプション

（5）戻り値

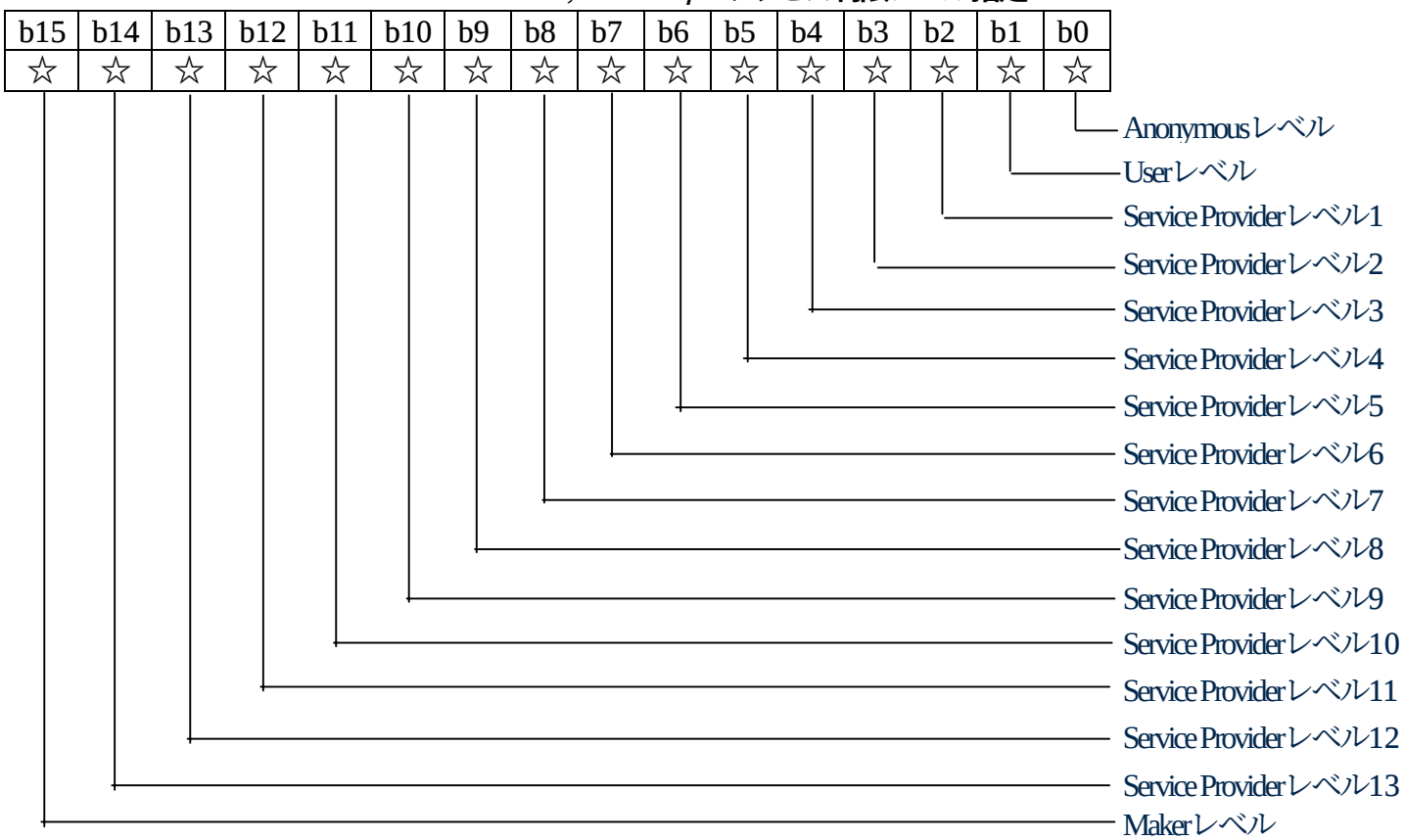
EAPI_NO_ERROR	：書き込み可能
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_ILLEGAL_PARAM	：id_kind が不正
EAPI_NOTFOUND_EPC	：プロパティが存在しない
EAPI_NOTSEND	：送信待ち中
EAPI_MEMBER_EPC	：配列要素のプロパティ
EAPI_NORESOURCE	：リソース不足
EAPI_NOCONDITION	：書き込み不可能なプロパティ
EAPI_ETC_NOCONDITION	：指定拡張通信機能での書き込み不可能なプロパティ

（6）使用する構造体

```
typedef struct {
        short   ext_hed;    /* この構造体の種類を示すコード。
                            0x0001：セキュア通信指定*/
        short   cipher;         /* 暗号化指定（方式指定含）。
                            0x0000：暗号化無し
```

0x0001：AES-CBC
0x0002～0xFFFF：for future reserved */

short authent; /* アクセス制限レベル指定



その他：for future reserved */

```
    short   authentication      /* 認証有無指定 */
    short   makerKeyIndex       /* メーカ KeyIndex */
    short   makerKeysize        /* メーカ Key サイズ */
    char    makerKey            /* メーカ Key */
} EXT_CONT
```

（7）注意事項

特に無し。

４．３．１３ MidSetSendEpcM，MidExtSetSendEpcM

（1）名称

MidSetSendEpcM, MidExtSetSendEpcM　－　ECHONET オブジェクトの配列のプロパティに対応した送信要求関数

（2）機能

配列の ECHONET プロパティに要素指定でデータを書き込み、サービスを送信する

（3）構文

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

（4）説明［Optional 関数］

MidSetSendEpcM は、id,eoj_code,epc_code によって指定された ECHONET プロパティの member_no の要素にデータを書き込み、esv_code のサービスを送信する。

データを書き込みたい任意のタイミングで呼び出し可。

書き込んだ時点でその要素は有効となる。

MidExtSetSendEpcM も、基本的には MidSetSendEpcM と同じ機能を持つが、書き込みデータに関するセキュア通信設定が行える関数である。

id_kind　　：[in]ID 種別
- APIVAL_NODE_KIND　：0(機器 ID)
- APIVAL_EA_KIND　：1(ECHONET アドレス)
- APIVAL_BROAD_KIND　：2(同報)

id　　：[in]機器 ID または ECHONET アドレスまたは同報アドレス

seoj_code　　：[in]SEOJ コード (下位 3byte のみ使用）
SEOJ なしの場合は、-1 を設定

deoj_code　　：[in]DEOJ コード（下位 3byte のみ使用）
DEOJ なしの場合は、-1 を設定

epc_code　　：[in]EPC コード(下位 1byte のみ使用）

esv_code　　：[in]ESV コード
- ESV_SetIM　　：0x64(応答不要要素指定プロパティ値書込要求)
- ESV_SetCM　　：0x65(応答要要素指定プロパティ値書込要求)
- ESV_GetM　　：0x66(要素指定プロパティ値読み出し要求)
- ESV_INFMReq　：0x67(要素指定プロパティ値通知要求)
- ESV_AddMI　　：0x68(応答不要要素指定プロパティ値追加要求)
- ESV_AddMC　　：0x69(応答要要素指定プロパティ値追加要求)

ESV_DelMI　：0x6A(応答不要要素指定プロパティ値削除要求)
ESV_DelMC　：0x6B(応答要要素指定プロパティ値削除要求)
ESV_CheckM　：0x6C(要素指定プロパティチェック要求)
ESV_AddMSI　：0x6D(応答不要要素追加要求)
ESV_AddMSC　：0x6E(応答要要素追加要求)
ESV_INFM　：0x77(要素指定プロパティ値通知)
ESV_INFM_AREQ　：0x78(プロパティ値の要素指定通知確認要求)

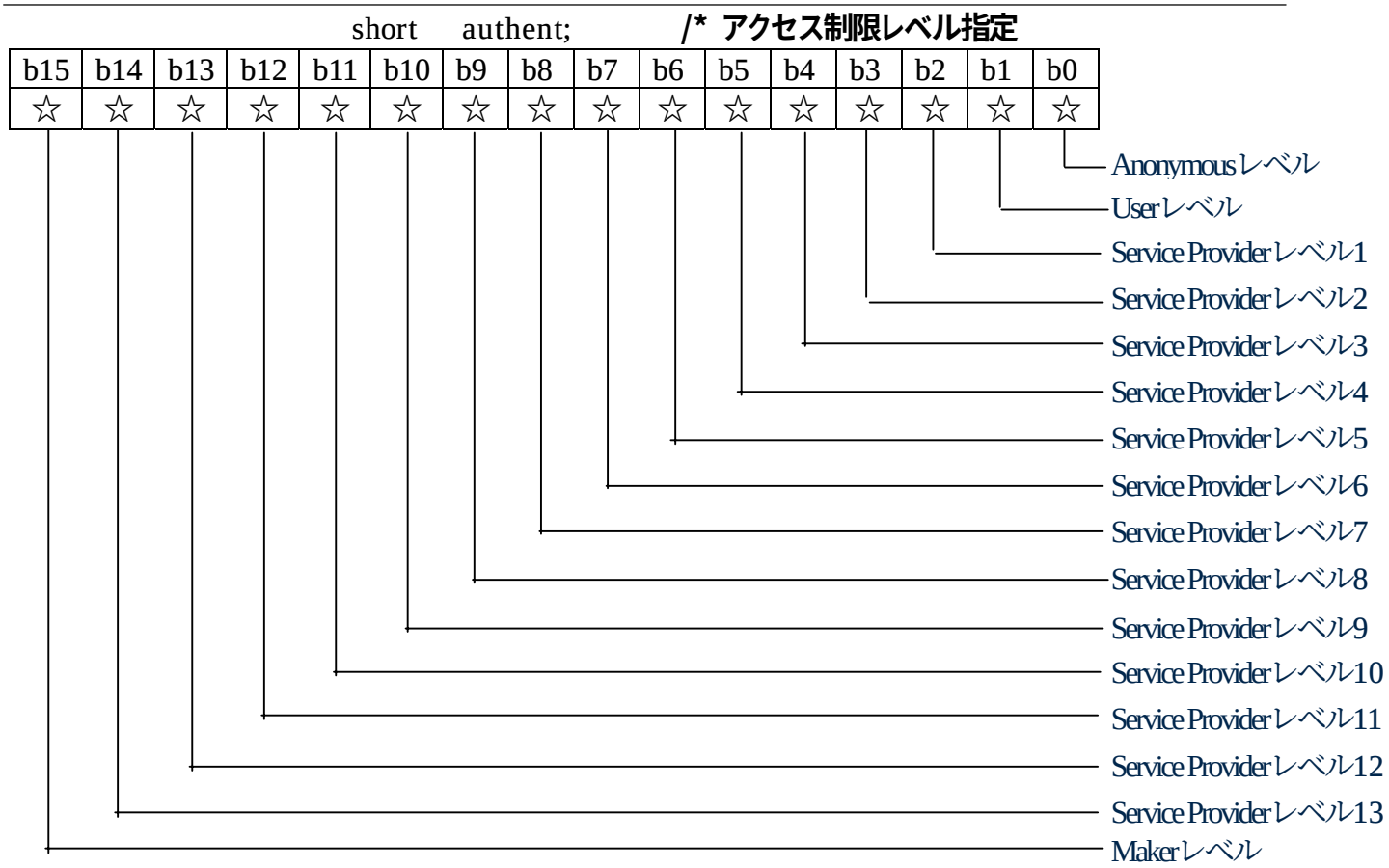
member_no　：[in]要素番号(０～0xFFFF)
data　：[in]設定データ
size　：[in]データサイズ
extcont　：[in] 拡張通信オプション

（5）戻り値

EAPI_NO_ERROR　：設定成功
EAPI_NOTOPEN　：非起動（セッションが未 open）
EAPI_ILLEGAL_PARAM　：id_kind もしくは esv_code が不正
EAPI_NOTFOUND_EPC　：プロパティが存在しない
EAPI_DATASIZE_EROR　：書き込みデータサイズが不正
EAPI_NORESOURCE　：リソース不足、
id_kind が EA_KIND・BROAD_KIND の時のみ
EAPI_NOCONDITION　：制御不可能なプロパティ
EAPI_NOTMEMBER_EPC　：配列要素のプロパティではない
EAPI_NOTFOUND_MNO　：指定された配列要素が存在しない
EAPI_NOTSEND　：未送信データあり
EAPI_TIMEOUT　：通信タイムアウト（同期の場合のみ）
EAPI_ETC_ERROR　：指定された拡張通信機能の処理不可

（6）使用する構造体

```
typedef struct {
        short   ext_hed;    /* この構造体の種類を示すコード。
                            0x0001：セキュア通信指定*/
        short   cipher;     /* 暗号化指定（方式指定含）。
                            0x0000：暗号化無し
                            0x0001：AES-CBC
                            0x0002～0xFFFF：for future reserved */
```



その他：for future reserved ***/**

```
        short   authentication      /* 認証有無指定 */
        long    makerKeyIndex       /* メーカ KeyIndex */
        short   makerKeysize        /* メーカ Key サイズ */
        short   makerKey            /* メーカ Key */
    } EXT_CONT
```

（7）注意事項

配列要素指定の以外の書き込みは不可。

４．３．１４ MidSetEpcM，MidExtSetEpcM

（1）名称

MidSetEpcM，MidExtSetEpcM　－　ECHONET オブジェクトの配列のプロパティへのデータの書込要求関数

（2）機能

配列の ECHONET プロパティに要素指定でデータを書き込む。

（3）構文

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

（4）説明［Optional 関数］

MidSetEpcM は、id,eoj_code,epc_code によって指定された ECHONET プロパティの member_no の要素にデータを書き込む。データを書き込みたい任意のタイミングで呼び出し可。

自機器への書き込みデータが前回のデータと変わった時で、状態変化通知処理設定のある場合のみ状態通知サービスを行なう。

MidExtSetEpcM も、基本的には MidSetEpcM と同じ機能を持つが、外部への通知データに関するセキュア通信設定が行える関数である。

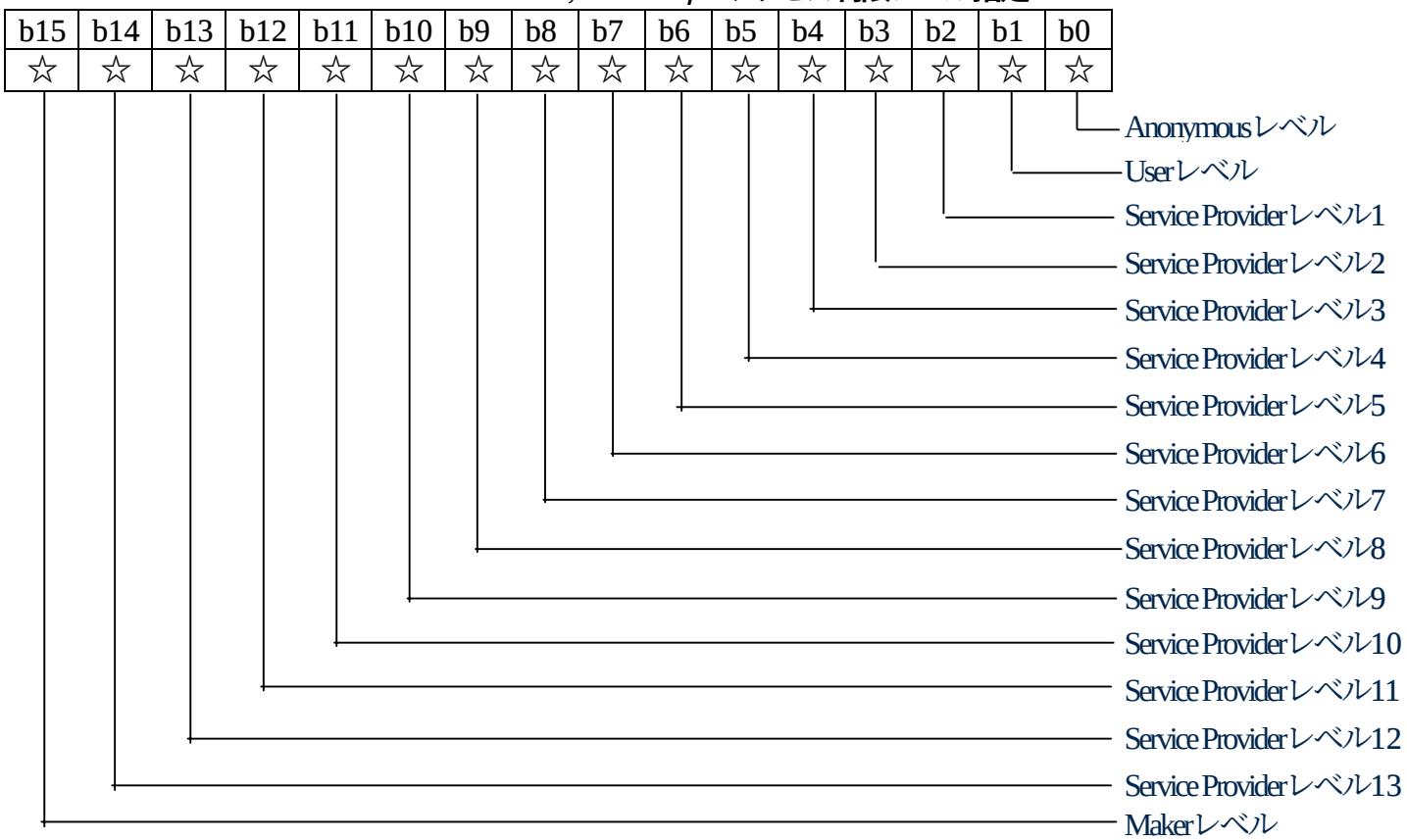
id_kind	：[in]ID 種別
APIVAL_NODE_KIND	:０(機器 ID)
APIVAL_EA_KIND	:１(ECHONET アドレス)
id	：[in]機器 ID または ECHONET アドレス
eoj_code	：[in]EOJ コード(下位 3byte のみ使用)
epc_code	：[in]EPC コード(下位 1byte のみ使用)
member_no	：[in]要素番号(０～0xFFFF)
data	：[in]設定データ
size	：[in]データサイズ
extcont	：[in] 拡張通信オプション

（5）戻り値

EAPI_NO_ERROR	：設定成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NOTFOUND_EPC	：プロパティが存在しない
EAPI_NOTMEMBER_EPC	：配列要素のプロパティではない
EAPI_NOTFOUND_MNO	：指定された配列要素が存在しない
EAPI_DATASIZE_EROR	：データサイズが不正
EAPI_ILLEGAL_PARAM	：id_kind が不正
EAPI_ETC_ERROR	：指定された拡張通信機能の処理不可

（6）使用する構造体

```
typedef struct {
        short   ext_hed;        /* この構造体の種類を示すコード。
                                0x0001：セキュア通信指定*/
        short   cipher;             /* 暗号化指定（方式指定含）。
                                0x0000：暗号化無し
                                0x0001：AES-CBC
                                0x0002～0xFFFF：for future reserved */
        short   authent;            /* アクセス制限レベル指定
```




```
                                その他：for future reserved */
        short   authentication      /* 認証有無指定 */
        logn    makerKeyIndex       /* メーカ KeyIndex */
        short   makerKeysize        /* メーカ Key サイズ */
        char    *makerKey           /* メーカ Key */
} EXT_CONT
```

（7）注意事項

配列要素指定の以外の書き込みは不可。
書き込んだ時点でその要素は有効となる。

４．３．１５ MidGetReceiveEpcM

（1）名称

MidGetReceiveEpcM　－　ECHONET オブジェクトの配列のプロパティからのデータの読込要求関数（1）

（2）機能

受信のあった配列の ECHONET プロパティの要素指定データを読み込む。

（3）構文

```
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）説明［Optional 関数］

id,eoj_code,epc_code によって指定された ECHONET プロパティの member_no の配列要素の受信データを読み込む。受信データを読み込みたい任意のタイミングで呼び出し可能。

id_kind	：[in]ID 種別
APIVAL_NODE_KIND	:0(機器 ID)
APIVAL_EA_KIND	:1(ECHONET アドレス)
id	：[in]機器 ID または ECHONET アドレス
eoj_code	：[in]EOJ コード (下位 3byte のみ使用)
epc_code	：[in]EPC コード(下位 1byte のみ使用)
member_no	：[in]要素番号(０～0xFFFF)
buff_size	：[in]エリアサイズ
esv_code	：[out]ESV コード格納エリア
data	：[out]データ内容格納エリア
data_size	：[out]読み込みデータサイズ
eoj_code2	：[out]通信上の SEOJ コードまたは DEOJ コード 下位 3byte のみ使用、ない場合は−1

（5）戻り値

EAPI_NO_ERROR	：読み込み成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_ILLEGAL_PARAM	：id_kind が不正
EAPI_NOTFOUND_EPC	：プロパティが存在しない
EAPI_NORECEIVE	：受信データが存在しない
EAPI_NOTMEMBER_EPC	：配列要素のプロパティではない
EAPI_NOTFOUND_MNO	：指定された配列要素が存在しない
EAPI_DATASIZE_EROR	：データサイズが不正

（7）注意事項

配列要素指定の以外の読み込みは不可。

４．３．１６ MidGetEpcM

（1）名称

MidGetEpcM　－　ECHONET オブジェクトの配列のプロパティからのデータの読込要求関数（2）

（2）機能

受信の無有に無関係に、配列でない ECHONET プロパティからのデータを取得する。

（3）構文

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）説明［Optional 関数］

id,eoj_code,epc_code によって指定された ECHONET プロパティの member_no の要素の現状状態を取得。状態を読み込みたい任意のタイミングで呼び出し可能。受信の有無に関わらず現在の状態を取得することができる。

id_kind　：[in]ID 種別
　　APIVAL_NODE_KIND　　：0(機器 ID)
　　APIVAL_EA_KIND　　：1(ECHONET アドレス)
id　　：[in]機器 ID または ECHONET アドレス
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)
epc_code　：[in]EPC コード(下位 1byte のみ使用)
member_no　　：[in]要素番号(0～0xFFFF)
buff_size　：[in]エリアサイズ
data　　：[out]データ内容格納エリア
data_size　：[out]読み込みデータサイズ

（5）戻り値

EAPI_NO_ERROR	：取得成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NOTFOUND_EPC	：プロパティが存在しない
EAPI_NOTMEMBER_EPC	：配列要素のプロパティではない
EAPI_NOTFOUND_MNO	：指定された配列要素が存在しない
EAPI_ILLEGAL_PARAM	：id_kind が不正
EAPI_NOCONDITION	：制御不可能なプロパティ
EAPI_DATASIZE_EROR	：データサイズエラー

（6）使用する構造体

特に無し。

（7）注意事項

配列要素指定の以外の読み込みは不可。

４．３．１７ MidSetSendCheckEpcM，MidExtSetSendCheckEpcM

（1）名称

MidSetSendCheckEpcM、MidExtSetSendCheckEpcM　－　ECHONET オブジェクトの配列であるプロパティへのデータの書込確認関数

（2）機能

配列の ECHONET プロパティへのデータ書込みの確認。

（3）構文

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

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

（4）説明［Optional 関数］

id,eoj_code,epc_code によって指定された ECHONET プロパティの member_no の配列要素にデータを書き込めるかをチェックする。データを書き込めるかチェックしたい任意のタイミングで呼び出し可能、書き込み不可の場合は、前回書き込んだ内容が未送信の場合がありえる。

id_kind　：[in]ID 種別
　APIVAL_NODE_KIND　：0(機器 ID)
　APIVAL_EA_KIND　：1(ECHONET アドレス)
id　：[in]機器 ID または ECHONET アドレス
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)
epc_code　：[in]EPC コード(下位 1byte のみ使用)
member_no　：[in]要素番号(0～0xFFFF)
extcont　：[in]拡張通信オプション

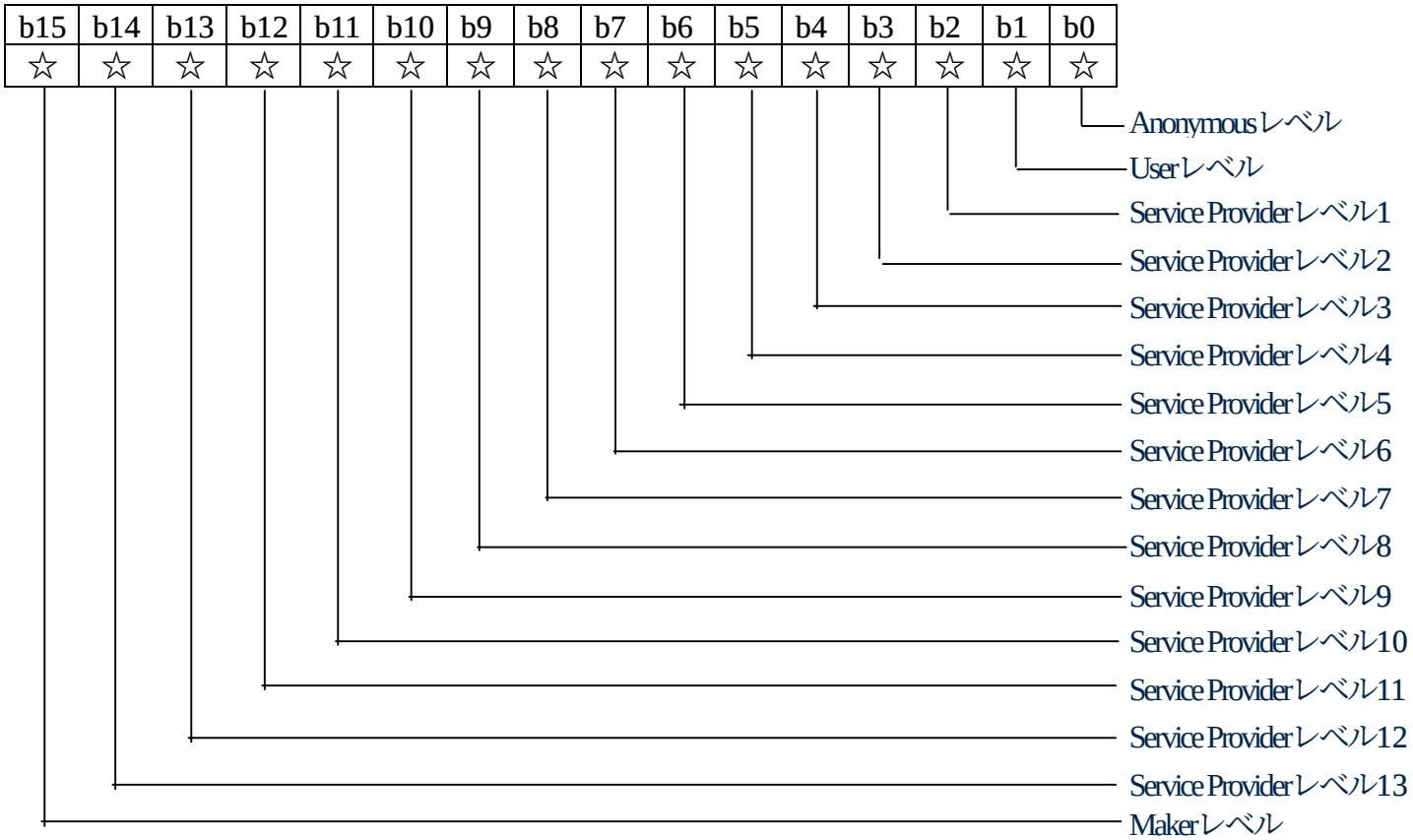
（5）戻り値

EAPI_NO_ERROR	：書き込み可能
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_ILLEGAL_PARAM	：id_kind が不正
EAPI_NOTFOUND_EPC	：プロパティが存在しない
EAPI_NOTSEND	：送信待ち中
EAPI_NOTMEMBER_EPC	：配列要素のプロパティではない
EAPI_NOTFOUND_MNO	：指定された配列要素が存在しない
EAPI_NORESOURCE	：リソース不足
EAPI_NOCONDITION	：書き込み不可能なプロパティ
EAPI_ETC_NOCONDITION	：指定拡張通信機能での書き込み不可能なプロパティ

（6）使用する構造体

typedef struct｛

```
short   ext_hed;    /* この構造体の種類を示すコード。
                    0x0001：セキュア通信指定*/
short   cipher;         /* 暗号化指定（方式指定含）。
                    0x0000：暗号化無し
                    0x0001：AES-CBC
                    0x0002～0xFFFF：for future reserved */
short   authent;        /* アクセス制限レベル指定
```




```
                    その他：for future reserved */
short   authentication      /* 認証有無指定 */
long    makerKeyIndex       /* メーカ KeyIndex */
short   makerKeysize        /* メーカ Key サイズ */
char    *makerKey           /* メーカ Key */
} EXT_CONT
```

（7）注意事項

特に無し。

４．３．１８ MidGetReceiveCheckEpc，MidExtGetReceiveCheckEpc

（1）名称

MidGetReceiveCheckEpc　－　ECHONET プロパティのデータ読出確認関数

（2）機能

受信した ECHONET プロパティを確認する。

（3）構文

```
long MidGetReceiveCheckEpc ( short buff_num, short *id_kind, short *id, short *EA,
        long *eoj_code, short *epc_code, short *esv_code,short *member_no,
        short *out_num )
long MidExtGetReceiveCheckEpc ( short buff_num, short *id_kind, short *id,
        short *EA, long *eoj_code, short *epc_code, short *esv_code,short *member_no,
        short *out_num )
```

（4）説明［Optional 関数］

MidGetReceiveCheckEpc は、全ての機器のオブジェクトを検索し、受信のあった EPC を受信順にリストアップする。受信を確認したい任意のタイミングで呼び出し可能。

MidExtGetReceiveCheckEpc も、基本的には MidGetReceiveCheckEpc と同じ機能を持つが、セキュア通信等拡張された設定の電文の受信をリストアップが行える関数である。受信を確認したい任意のタイミングで呼び出し可能。

buff_num	：[in]リストアップ最大要素数
id_kind	：[out]機器 ID 種類を示すコードの格納領域を指定するポインタ
id	：[out]機器 ID の格納領域を指定するポインタ(-1：ID 管理無し)
EA	：[out]ECHONET アドレス
eoj_code	：[out]EOJ コード (下位 3byte のみ使用) 解析できなかったセキュア電文受信を確認の場合－１
epc_code	：[out]受信オブジェクト EPC コード格納エリア(下位 1byte のみ使用) 解析できなかったセキュア電文受信を確認の場合－１
esv_code	：[out] ESV コード格納エリア 解析できなかったセキュア電文受信を確認の場合－１
member_no	：[out]配列要素番号格納エリア 配列要素オブジェクト以外の場合及び解析できなかったセキュア電文受信を確認の場合は-1 が格納される
out_num	：[out]リストアップ数格納エリア

（5）戻り値

EAPI_NO_ERROR	：リストアップ成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_ILLEGAL_PARAM	：buff_num が不正（リストアップ最大数を超える）

（6）注意事項

buff_num＜out_num の場合は、リストアップされていない受信データがあることを意味する。リストアップ最大数は 100（本数字は特に規定するものではない）とする。

４．３．１９ MidGetEpcSize

（1）名称

MidGetEpcSize　－　ECHONET プロパティのサイズ取得関数

（2）機能

ECHONET プロパティのデータサイズを取得する。

（3）構文

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

（4）説明［Optional 関数］

id,eoj_code,epc_code によって指定された ECHONET プロパティのデータサイズを取得する。取得したい任意のタイミングで呼び出し可能。

id_kind　：[in]ID 種別
　APIVAL_NODE_KIND　：0(機器 ID)
　APIVAL_EA_KIND　：1(ECHONET アドレス)
id　：[in]機器 ID または ECHONET アドレス
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)
epc_code　：[in]EPC コード(下位 1byte のみ使用)
size　：[out]プロパティデータサイズ(byte 数)格納エリア
　配列要素プロパティの場合は、各要素 byte 数が格納される
mem_num　：[out]配列要素数格納エリア
　通常プロパティの場合は、１固定となる

（5）戻り値

EAPI_NO_ERROR　：取得成功
EAPI_NOTOPEN　：非起動（セッションが未 open）
EAPI_NOTFOUND_EPC　：プロパティが存在しない
EAPI_ILLEGAL_PARAM　：id_kind が不正

（6）使用する構造体

特に無し。

（7）注意事項

特に無し。

４．３．２０ MidGetEpcAttrib

（1）名称

MidGetEpcAttrib　－　ECHONET プロパティの属性取得関数

（2）機能

機器オブジェクトのプロパティ属性を取得する。

（3）構文

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

（4）説明［Optional 関数］

id,eoj_code,epc_code で指定された ECHONET オブジェクトのプロパティの各属性を取得する。

id_kind　：[in]ID 種別
　　APIVAL_NODE_KIND　　：0(機器 ID)
　　APIVAL_EA_KIND　　：1(ECHONET アドレス)
id　　：[in]機器 ID または ECHONET アドレス
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)
epc_code　：[in]EPC コード(下位 1byte のみ使用)
data_type：[out]データ型取得エリア
　　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（データ型なし）
rule　　：[out]アクセスルール取得エリア（処理するものが全て OR された値）
　　APIVAL_RULE_SET　　：0x0001（Set）
　　APIVAL_RULE_GET　　：0x0002（Get）
　　APIVAL_RULE_SETM　　：0x0100（要素指定 Set）
　　APIVAL_RULE_GETM　　：0x0200（要素指定 Get）
　　APIVAL_RULE_ADDM　　：0x0400（要素指定追加要求）
　　APIVAL_RULE_DELM　　：0x0800（要素指定削除要求）
　　APIVAL_RULE_CHECKM　：0x1000（要素指定存在確認要求）
data_size ：[out]データサイズ取得エリア
　　配列要素オブジェクトの場合は各要素サイズが格納される

（5）戻り値：

EAPI_NO_ERROR　　：取得成功
EAPI_NOTOPEN　　：非起動（セッションが未 open）
EAPI_NOTFOUND_EPC　　：プロパティが存在しない

EAPI_ILLEGAL_PARAM　　：id_kind が不正

（6）使用する構造体

特に無し。

（7）注意事項

特に無し。

４．３．２１ MidGetEpcMember

（1）名称

MidGetEpcMember　－　ECHONET オブジェクトの配列のプロパティの配列要素取得関数

（2）機能

配列要素オブジェクト情報を取得する。

（3）構文：

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

（4）説明［Optional 関数］

id,eoj_code,epc_code によって指定された配列要素 ECHONET プロパティの配列要素数、要素データサイズ、各配列要素番号を buff_size 分取得する。取得したい任意のタイミングで呼び出し可能。

id_kind	：[in]ID 種別
APIVAL_NODE_KIND	:０(機器 ID)
APIVAL_EA_KIND	:１(ECHONET アドレス)
id	：[in]機器 ID または ECHONET アドレス
eoj_code	：[in]EOJ コード(下位 3byte のみ使用)
epc_code	：[in]EPC コード(下位 1byte のみ使用)
buff_size	：[in]要素番号格納可能数
member_no	：[out]要素番号格納エリア
member_num	：[out]要素数格納エリア
data_size	：[out]要素データサイズ

（5）戻り値

EAPI_NO_ERROR	：取得成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NOTFOUND_EPC	：プロパティが存在しない
EAPI_NOT_MOBJECT	：配列要素のプロパティではない
EAPI_ILLEGAL_PARAM	：id_kind が不正

（6）使用する構造体

特に無し。

（7）注意事項

buff_size＜member_num の場合は、未取得の配列要素がある事を意味する。

４．３．２２ MidCreateNode

（1）名称

MidCreateNode　－　管理機器追加作成関数

（2）機能

ECHONET 通信ミドルウェアで管理する他機器を追加作成する。

（3）構文

long MidCreateNode(short ea_code, short *node_id)

（4）説明［Optional 関数］

新たな管理他機器を指定 EA コードで作成する。（あくまで自ノード上のデータ操作。）機器 ID は既存機器と重ならない ID を ECHONET 通信ミドルウェアで自動付与する。

ea_code　：[in]設定エコーネットアドレスコード
node_id　：[out]作成機器 ID 格納エリア

（5）戻り値：

EAPI_NO_ERROR	：作成成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NORESORCE	：リソース不足
EAPI_EXIST_NODE	：指定 EA の機器が既に存在

（6）使用する構造体

特に無し。

（7）注意事項

特に無し。

４．３．２３ MidCreateObj

（1）名称

MidCreateObj　－　ECHONET オブジェクトの追加作成関数

（2）機能

ECHONET オブジェクトを追加作成する。

（3）構文

long MidCreateObj **(short node_id, long eoj_code,)**

（4）説明［Optional 関数］

node_id,eoj_code によって指定されたエコーネットオブジェクトを作成する。（あくまで自ノード上のデータ操作。）指定機器が既に存在することが必要。
ECHONET オブジェクトを作成したい任意のタイミングで呼び出し可能。
node_id　：[in]機器 ID
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)

（5）戻り値：

EAPI_NO_ERROR	：作成成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NORESORCE	：リソース不足
EAPI_EXIST_OBJ	：指定のオブジェクト既存
EAPI_NOTFOUND_NODE	：指定の管理機器が存在しない

（6）使用する構造体

特に無し。

（7）注意事項

特に無し。

４．３．２４ MidCreateEpc，MidCreateExtEpc

（1）名称

MidCreateEpc, MidCreateExtEpc　－　配列でない ECHONET プロパティの追加作成関数

（2）機能

ECHONET プロパティを追加作成する。

（3）構文

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

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

（4）説明［Optional 関数］

MidCreateEpc は、node_id,eoj_code,epc_code によって指定された ECHONET プロパティを指定機器、指定 ECHONET オブジェクトに作成する。指定機器、指定オブジェクトが存在する事が必要となる。ECHONET プロパティを作成したい任意のタイミングで呼び出し可能。

MidCreateExtEpc も、基本的には MidCreateEpc と同じ機能を持つが、設定するプロパティ情報が、拡張されたものとなる関数である。

node_id　：[in]機器 ID
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)
epc_code　：[in]EPC コード(下位 1byte のみ使用)
data_type：[in]データ型
　　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（データ型なし）
rule　　：[in]アクセスルール（以下のルールの内、処理するものを OR する）
　　APIVAL_RULE_SET　　：0x0001（Set）
　　APIVAL_RULE_GET　　：0x0002（Get）
　　APIVAL_RULE_ANNO　：0x0040（Anno）
anno　　：[in]状変時アナウンス有無（自機器の場合のみ有効）
　　APIVAL_ANNO_ON　　：1（アナウンスあり）
　　APIVAL_ANNO_OFF　　：0（アナウンスなし）
data_size ：[in]データエリアサイズ(バイト数)
extepc　：[in]セキュア通信等のための拡張されたプロパティ情報設定領域

（5）戻り値：

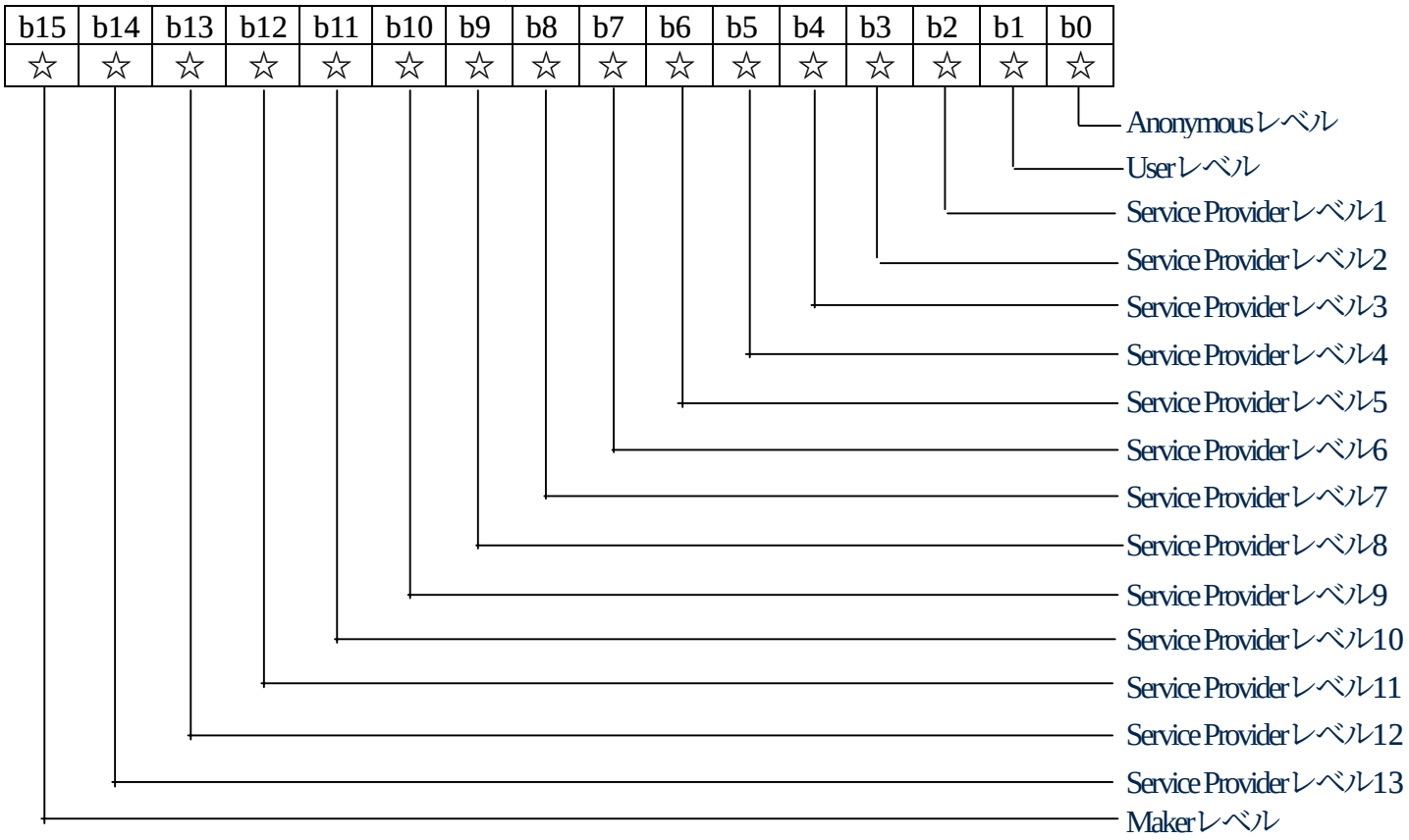
EAPI_NO_ERROR	：作成成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NORESORCE	：リソース不足
EAPI_EXIST_EPC	：プロパティ既存
EAPI_NOTFOUND_NODE	：管理機器が存在しない
EAPI_NOTFOUND_OBJ	：管理オブジェクトが存在しない
EAPI_ILLEGAL_PARAM	：data_type、rule、anno またはデータサイズが不正

（6）使用する構造体

```
typedef struct {

        short keykinds;     /* Set サービスに対するアクセス制限レベル  */
        short keykindg;     /* Get サービスに対するアクセス制限レベル  */
        short keykinda;     /* Anno サービスに対するアクセス制限レベル  */
} EXT_EPC
```

アクセス制限レベルは、以下の内、指定するものを OR した値とする。



（7）注意事項

配列の ECHONET プロパティの追加は扱えない。

４．３．２５ MidCreateEpcM，MidCreateExtEpcM

（1）名称

MidCreateEpcM, MidCreateExtEpcM　－　配列の ECHONET プロパティの追加作成関数

（2）機能

配列の **ECHONET** プロパティを作成する。

（3）構文

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

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

（4）説明［Optional 関数］

MidCreateEpcM は、node_id,eoj_code,epc_code によって指定された要素数 1 つの配列要素 ECHONET プロパティを指定機器、指定オブジェクトに作成する。指定機器、指定オブジェクトが存在する事が必要となる。配列要素プロパティを作成したい任意のタイミングで呼び出し可能。

MidCreateExtEpcM も、基本的には MidCreateEpcM と同じ機能を持つが、設定するプロパティ情報が、拡張されたものとなる関数である。

node_id　：[in]機器 ID
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)
epc_code　：[in]EPC コード(下位 1byte のみ使用)
data_type：[in]データ型
- 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（バイト配列）

rule　：[in]アクセスルール（以下のルールの内、処理するものを OR する）
- APIVAL_RULE_SETM　：0x0100（要素指定 Set）
- APIVAL_RULE_GETM　：0x0200（要素指定 Get）
- APIVAL_RULE_ADDM　：0x0400（要素指定追加要求）
- APIVAL_RULE_DELM　：0x0800（要素指定削除要求）
- APIVAL_RULE_CHECKM　：0x1000（要素指定存在確認要求）
- APIVAL_RULE_ADDMS　：0x2000（要素指追加要求）
- APIVAL_RULE_ANNOM　：0x4000（要素指定通知要求）

anno　：[in]状変時アナウンス有無（自機器の場合のみ有効）

APIVAL_ANNO_ON　　：1（アナウンスあり）
APIVAL_ANNO_OFF　　：0（アナウンスなし）

data_size　　：[in]要素サイズ(バイト数)
member_no　　：[in]作成要素番号（0～0xFFFF）

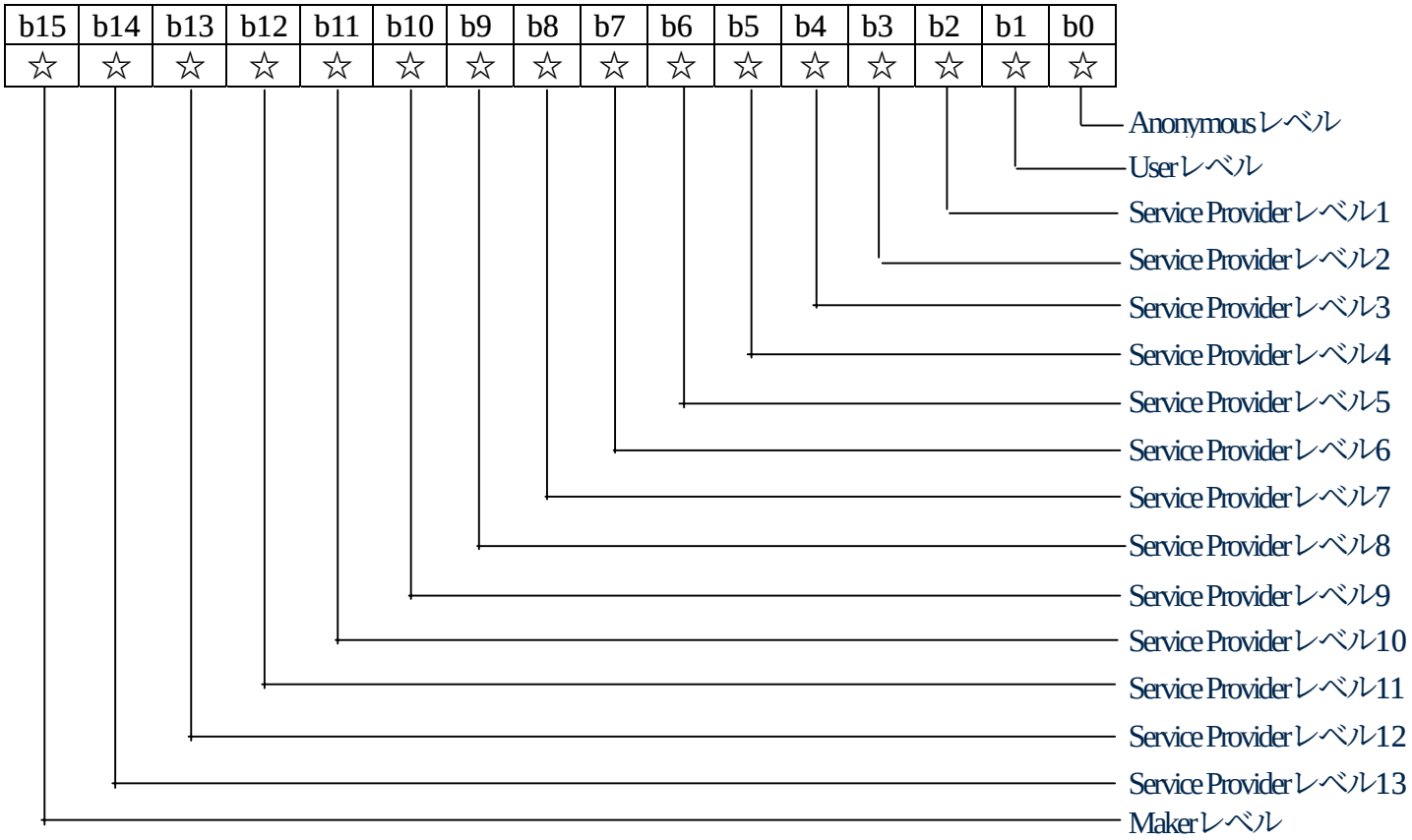
（5）戻り値

EAPI_NO_ERROR　　：作成成功
EAPI_NOTOPEN　　：非起動（セッションが未 open）
EAPI_NORESORCE　　：リソース不足
EAPI_EXIST_EPC　　：プロパティ既存
EAPI_NOTFOUND_NODE　　：管理機器が存在しない
EAPI_NOTFOUND_OBJ　　：管理オブジェクトが存在しない
EAPI_ILLEGAL_PARAM：data_type、rule、anno、data_size、または member_no が不正

（6）使用する構造体

```
typedef struct {
        shot ext_size;      /* この構造体のｻｲｽﾞ。Ver2.10 では、0x0E 。 */
        shot keykindsm;     /* SetM サービスに対するアクセス制限レベル */
        short keykindgm;    /* GetM サービスに対するアクセス制限レベル */
        short keykindadm;   /* AddM サービスに対するアクセス制限レベル */
        shot keykinddm;     /* DelM サービスに対するアクセス制限レベル */
        short keykindcm;    /* CheckM サービスに対するアクセス制限レベル */
        short keykindadms;  /* AddMS サービスに対するアクセス制限レベル */
        short keykindam;    /* AnnoM サービスに対するアクセス制限レベル */
} EXT_EPC_M
```

アクセス制限レベルは、以下の内、指定するものを OR した値とする。



（7）注意事項

配列の ECHONET プロパティ以外の作成できない。

４．３．２６ MidAddEpcMember

（1）名称

MidAddEpcMember　－　配列の EHCONET プロパティの配列要素追加（要素番号指定有）関数

（2）機能

要素番号を指定して配列のプロパティに配列要素を追加する。

（3）構文

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

（4）説明［Optional 関数］

node_id,eoj_code,epc_code によって指定された ECHONET プロパティに member_no の配列要素を追加する。すでに指定 ECHONET プロパティが存在する事が前提。

node_id	：[in]機器 ID
eoj_code	：[in]EOJ コード(下位 3byte のみ使用)
epc_code	：[in]EPC コード(下位 1byte のみ使用)
member_no	：[in]要素番号(０～0xFFFF)

（5）戻り値：

EAPI_NO_ERROR	：追加成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NOTFOUND_NODE	：管理機器が存在しない
EAPI_NOTFOUND_OBJ	：管理オブジェクトが存在しない
EAPI_NOTFOUND_EPC	：管理プロパティが存在しない
EAPI_NORESORCE	：リソース不足 or 合計要素数が 256 以上
EAPI_NOTMEMBER_EPC	：配列要素プロパティではない
EAPI_EXIST_MEMBER	：指定配列要素番号は既に存在する

（6）使用する構造体

特に無し。

（7）注意事項

特になし。

４．３．２７ MidAddEpcMemberS

（1）名称

MidAddEpcMemberS　－　配列の EHCONET プロパティの配列要素追加（要素番号指定無）関数

（2）機能

要素番号を指定せずに、配列のプロパティに配列要素を追加する。

（3）構文

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

（4）説明［Optional 関数］

node_id,eoj_code,epc_code によって指定された ECHONET プロパティに配列要素を追加する。配列要素番号は既存配列要素と重ならない番号が自動的に振られる。すでに指定 ECHONET プロパティが存在する事が前提。

node_id	：[in]機器 ID
eoj_code	：[in]EOJ コード(下位 3byte のみ使用)
epc_code	：[in]EPC コード(下位 1byte のみ使用)
member_no	：[out]要素番号格納エリア

（5）戻り値：

EAPI_NO_ERROR	：追加成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NOTFOUND_NODE	：管理機器が存在しない
EAPI_NOTFOUND_OBJ	：管理オブジェクトが存在しない
EAPI_NOTFOUND_EPC	：プロパティが存在しない
EAPI_NORESORCE	：リソース不足 or 合計要素数が 256 以上
EAPI_NOTMEMBER_EPC	：配列要素のプロパティではない

（6）使用する構造体

特に無し。

（7）注意事項

特になし。

４．３．２８ MidDeleteNode

（1）名称

MidDeleteNode　－　管理機器削除関数

（2）機能

ECHONET 通信ミドルウェアで管理している他機器を削除する。

（3）構文

long MidDeleteNode **(short node_id)**

（4）説明［Optional 関数］

node_id によって指定された管理機器を削除する。削除したい任意のタイミングで呼び出し可能。

node_id　：[in]機器 ID

（5）戻り値：

EAPI_NO_ERROR	：削除成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NOTFOUND_NODE	：指定の管理他機器が存在しない

（6）使用する構造体

特に無し。

（7）注意事項

機器を削除する事により、その機器に存在する全てのオブジェクト、プロパティが削除される。

４．３．２９ MidDeleteObj

（1）名称

MidDeleteObj　－　ECHONET オブジェクトの削除関数

（2）機能

ECHONET オブジェクトを削除する。

（3）構文

long MidDeleteObj **(short node_id, long eoj_code)**

（4）説明［Optional 関数］

node_id,eoj_code によって指定された ECHONET オブジェクトを削除する。
ECHONET オブジェクトを削除したい任意のタイミングで呼び出し可能。
node_id　：[in]機器 ID
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)

（5）戻り値：

EAPI_NO_ERROR	：削除成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NODELETE	：削除不可能
EAPI_NOTFOUND_OBJ	：指定のオブジェクトが存在しない

（6）使用する構造体

特に無し。

（7）注意事項

オブジェクトを削除する事により、そのオブジェクトに存在する全てのプロパティが削除される。これにより、指定機器に全てのプロパティが存在しなくなった場合も、機器のインスタンスは削除されない。機器のインスタンスを削除するには DeleteNode を呼び出す。

４．３．３０ MidDeleteEpc

（1）名称

MidDeleteEpc　－　ECHONET プロパティの削除関数

（2）機能

ECHONET プロパティを削除する。

（3）構文

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

（4）説明［Optional 関数］

node_id,eoj_code,epc_code によって指定された ECHONET プロパティを削除する。
ECHONET プロパティを削除したい任意のタイミングで呼び出し可能。

node_id　：[in]機器 ID
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)
epc_code　：[in]EPC コード(下位 1byte のみ使用)

（5）戻り値：

EAPI_NO_ERROR	：削除成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NODELETE	：削除不可能
EAPI_NOTFOUND_EPC	：指定のプロパティが存在しない

（6）使用する構造体

特に無し。

（7）注意事項

指定されたプロパティが配列要素プロパティの場合、全ての配列要素が削除される。これにより、指定オブジェクトに全てのプロパティが存在しなくなった場合も、オブジェクト自身は削除されない。オブジェクトを削除するには DeleteObj を呼び出す。

４．３．３１ MidDeleteEpcM

（1）名称

MidDeleteEpcM　－　配列の ECHONET プロパティの指定要素削除関数

（2）機能

配列の **ECHONET** プロパティの指定された要素を削除する。

（3）構文

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

（4）説明［Optional 関数］

node_id,eoj_code,epc_code によって指定された ECHONET プロパティの member_no で指定された配列要素を削除する。配列要素を無効にしたい任意のタイミングで呼び出し可能。

node_id	：[in]機器 ID
eoj_code	：[in]EOJ コード(下位 3byte のみ使用)
epc_code	：[in]EPC コード(下位 1byte のみ使用)
member_no	：[in]要素番号(０～0xFFFF)

（5）戻り値

EAPI_NO_ERROR	：設定成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_NOTFOUND_EPC	：プロパティが存在しない
EAPI_NOTMEMBER_EPC	：配列要素のプロパティではない
EAPI_NOTFOUND_MNO	：指定された配列要素が存在しない
EAPI_NODELETE	：削除不可能な配列要素

（6）使用する構造体

特に無し。

（7）注意事項

これにより指定プロパティのすべての配列要素が存在しなくなった場合も、プロパティ自身は削除されない。プロパティを削除するには DeleteEpc を呼び出す。

４．３．３２ MidGetState

（1）名称

MidGetState　－　ECHONET 通信処理部の状態取得関数

（2）機能

通信ミドルウェアの現在の状態を取得する。

（3）構文

long MidGetState **(short *state)**

（4）説明［Optional 関数］

現在の通信ミドルの状態を取得。

state　　：[out]通信ミドル状態格納エリア

MID_STS_STOP	：０(停止中)
MID_STS_INIT	：１(初期化中、初期化処理終了)
MID_STS_RUN	：２(通常処理中)
MID_STS_APL_ERR	：３(アプリ異常)
MID_STS_PRO_ERR	：４(プロトコル差異吸収処理部異常)
MID_STS_LOW_ERR	：５(下位通信ソフトウェア異常)

（5）戻り値

EAPI_NO_ERROR	：取得成功
EAPI_NOTOPEN	：非起動（セッションが未 open）

（6）使用する構造体

特に無し。

（7）注意事項

特に無し。

４．３．３３ MidSetRecvTargetList

（1）名称

MidSetRecvTargetList　－　データ受信通知対象リストの有効無効設定関数。

（2）機能

データ受信通知対象リストの有効無効設定を行う。

（3）構文

long MidSetRecvTargetList (**short setup**)

（4）説明［Optional 関数］

データ受信通知対象 Epc リストの有効・無効設定を行う。
有効とした場合は、AddTargetList によって指定された ECHONET プロパティに対する受信データのみを MidGetReceiveEPC、MidGetReceiveCheckEPC 対象とする。無効とした場合は、全ての受信データを MidGetReceiveEPC、MidGetReceiveCheckEPC 対象とする。

setup　　：[in]有効・無効（０：無効、１：有効）

（5）戻り値

EAPI_NOTOPEN　　：非起動（セッションが未 open）

（6）使用する構造体

特に無し。

（7）注意事項

有効中に有効設定された場合、無効中に無効設定された場合もエラーとはしない。

４．３．３４ MidAddRecvTargetList

（1）名称

MidAddRecvTargetList　－　データ受信通知対象リストの追加関数。

（2）機能

データ受信通知対象リストへの追加を行う。

（3）構文

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

（4）説明［Optional 関数］

データ受信通知の対象となる Epc を設定する。設定後は id,eoj_code,epc_code によって指定された ECHONET プロパティに対する受信データを MidGetReciveEPC、MidGetReciveCheckEPC 対象とする。

id_kind　：[in]ID 種別
　　APIVAL_NODE_KIND　　：0(機器 ID)
　　APIVAL_EA_KIND　　：1(ECHONET アドレス)
id　　：[in]機器 ID または ECHONET アドレス
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)
epc_code　：[in]EPC コード(下位 1byte のみ使用)

（5）戻り値

EAPI_NOTOPEN　　：非起動（セッションが未 open）
EAPI_NOTFOUND_OBJECT　　：プロパティが存在しない
EAPI_ILLEGAL_PARAM　　：id_kind が不正

（6）使用する構造体

特に無し。

（7）注意事項

配列要素ごとの設定は出来ない。
現在受信対象中の Epc を指定した場合もエラーとはしない。

４．３．３５ MidDeleteRecvTargetList

（1）名称

MidDeleteRecvTargetList　－　データ受信通知対象リストの削除関数。

（2）機能

データ受信通知対象リストの削除を行う。

（3）構文

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

（4）説明［Optional 関数］

指定 Epc を受信対通知象から削除する。
削除後は id,eoj_code,epc_code によって指定された ECHONET プロパティに対する受信データは MidGetReceiveEPC、MidGetReceiveCheckEPC 対象外とする。

id_kind　：[in]ID 種別
　　APIVAL_NODE_KIND　　:０(機器 ID)
　　APIVAL_EA_KIND　　:１(ECHONET アドレス)
id　　：[in]機器 ID または ECHONET アドレス
eoj_code　：[in]EOJ コード(下位 3byte のみ使用)
epc_code　：[in]EPC コード(下位 1byte のみ使用)

（5）戻り値

EAPI_NOTOPEN　　：非起動（セッションが未 open）
EAPI_NOTFOUND_OBJECT　　：プロパティが存在しない
EAPI_ILLEGAL_PARAM　　：id_kind が不正

（6）使用する構造体

特に無し。

（7）注意事項

配列要素ごとの設定は出来ない。
現在受信対象外の Epc を指定した場合もエラーとはしない。

４．３．３６ MidGetRecvTargetList

（1）名称

MidGetRecvTargetList　－　データ受信通知対象リストの取得関数。

（2）機能

データ受信通知対象リストの取得を行う。

（3）構文

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

（4）説明［Optional 関数］

データ受信通知の対象となる Epc リストを buff_num 分取得する。

buff_num：[in]リストバッファ数
setup　　：[out]リスト有効無効設定（０：無効、１：有効）
node_id_ ：[out]機器 ID リスト格納エリア
eoj_code ：[out]EOJ コードリスト格納エリア(下位 3byte のみ使用)
epc_code ：[out]EPC コードリスト格納エリア(下位 1byte のみ使用)
data_num：[out]データ数

（5）戻り値

EAPI_NOTOPEN　　：非起動（セッションが未 open）

（6）使用する構造体

特に無し。

（7）注意事項

buffnum<data_num の場合は、リストアップされていない受信対象 Epc が存在する事を意味する。

４．３．３７ MidStart

（1）名称

MidStart　－　ECHONET 通信処理部初期化関数。

（2）機能

通信ミドルウェアを起動し、ウォームスタートを行う。

（3）構文

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

（4）説明［Optional 関数］

mid_no で指定された通信ミドルウェアの ECHONET 通信処理部の ECHONET アドレスを保持したまま、ECHONET 通信処理部の起動、プロトコル差異吸収処理部、下位通信ソフトウェアのロードを行う。セッションの open は行わない。

mid_no	：[in] 通信ミドルウェアＮｏ
mid_name	：[in] 通信ミドルウェアプロセス名
p_init	：[in] プロトコル差異吸収処理部初期化データ
dev_num	：[in] 下位通信ソフトウェア搭載数
l_init	：[in] 下位通信ソフトウェア初期化データ dev_num 分のデータを用意する

（5）戻り値

EAPI_NO_ERROR	：初期化成功
EAPI_MID_ERROR	：ECHONET 通信処理部初期化失敗
EAPI_PRO_ERROR	：プロトコル差異吸収処理部初期化失敗
EAPI_LOW_ ERROR	：下位通信ソフトウェア初期化失敗
EAPI_ILLEGAL_PARAM	：下位通信ソフトウェア搭載数が不正

（6）構造体

特に無し。

（7）注意事項

void *p_init，void *l_init については、実装規定とする。

４．３．３８ MidReset

（1）名称

MidReset　－　ECHONET 通信処理部初期化関数。

（2）機能

通信ミドルウェアを起動、初期化し、コールドスタート(3)を行う。

（3）構文

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

（4）説明［Optional 関数］

mid_no で指定された通信ミドルウェアの ECHONET 通信処理部の ECHONET アドレスを破棄し、ECHONET 通信処理部の起動、プロトコル差異吸収処理部、下位通信ソフトウェアのロードを行う。セッションの open は行わない。

mid_no	：[in] 通信ミドルウェアＮｏ
mid_name	：[in] 通信ミドルウェアプロセス名
p_init	：[in] プロトコル差異吸収処理部初期化データ
dev_num	：[in] 下位通信ソフトウェア搭載数
l_init	：[in] 下位通信ソフトウェア初期化データ

dev_num 分のデータを用意する

（5）戻り値

EAPI_NO_ERROR	：初期化成功
EAPI_MID_ERROR	：ECHONET 通信処理部初期化失敗
EAPI_PRO_ERROR	：プロトコル差異吸収処理部初期化失敗
EAPI_LOW_ ERROR	：下位通信ソフトウェア初期化失敗
EAPI_ILLEGAL_PARAM	：下位通信ソフトウェア搭載数が不正

（6）構造体

特に無し。

（7）注意事項

void *p_init，void *l_init については、実装規定とする。

４．３．３９ MidInit

（1）名称

MidInit　－　ECHONET 通信処理部初期化関数。

（2）機能

通信ミドルウェアを起動、初期化し、コールドスタート(2)を行う。

（3）構文

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

（4）説明

mid_noで指定された通信ミドルウェアのECHONET通信処理部の初期化､起動､プロトコル差異吸収処理部、下位通信ソフトウェアのロードと初期化を行う。セッションの open は行わない。

mid_no	：[in] 通信ミドルウェアＮｏ
mid_name	：[in] 通信ミドルウェアプロセス名
p_init	：[in] プロトコル差異吸収処理部初期化データ
dev_num	：[in] 下位通信ソフトウェア搭載数
l_init	：[in] 下位通信ソフトウェア初期化データ

dev_num 分のデータを用意する

（5）戻り値

EAPI_NO_ERROR	：初期化成功
EAPI_MID_ERROR	：ECHONET 通信処理部初期化失敗
EAPI_PRO_ERROR	：プロトコル差異吸収処理部初期化失敗
EAPI_LOW_ ERROR	：下位通信ソフトウェア初期化失敗
EAPI_ILLEGAL_PARAM	：下位通信ソフトウェア搭載数が不正

（6）構造体

特に無し。

（7）注意事項

void *p_init，void *l_init については、実装規定とする。

４．３．４０ MidInitAll

（1）名称

MidInitAll　－　ECHONET 通信処理部初期化関数。

（2）機能

通信ミドルウェアを起動、初期化し、コールドスタート(1)を行う。

（3）構文

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

（4）説明［Optional 関数］

mid_noで指定された通信ミドルウェアのECHONET通信処理部の初期化、起動、プロトコル差異吸収処理部、下位通信ソフトウェアのロードと初期化を行う。セッションの open は行わない。

mid_no	：[in] 通信ミドルウェアＮｏ
mid_name	：[in] 通信ミドルウェアプロセス名
p_init	：[in] プロトコル差異吸収処理部初期化データ
dev_num	：[in] 下位通信ソフトウェア搭載数
l_init	：[in] 下位通信ソフトウェア初期化データ dev_num 分のデータを用意する

（5）戻り値

EAPI_NO_ERROR	：初期化成功
EAPI_MID_ERROR	：ECHONET 通信処理部初期化失敗
EAPI_PRO_ERROR	：プロトコル差異吸収処理部初期化失敗
EAPI_LOW_ ERROR	：下位通信ソフトウェア初期化失敗
EAPI_ILLEGAL_PARAM	：下位通信ソフトウェア搭載数が不正

（6）構造体

特に無し。

（7）注意事項

void *p_init，void *l_init については、実装規定とする。

４．３．４１ MidRequestRun

（1）名称

MidRequestRun　－　ECHONET 通信ミドルウェア動作開始関数。

（2）機能

通信ミドルウェアの動作開始を要求する。

（3）構文

long MidRequestRun(void)

（4）説明

MidInit 終了後の待機状態において、通信ミドルウェアの ECHONET 通信処理部の動作を開始する。

（5）戻り値

EAPI_NO_ERROR　　：開始成功
EAPI_NOTOPEN：非起動（セッションが未 open）
EAPI_MID_ERROR　　：ECHONET 通信処理部エラー

（6）注意事項

通信ミドルの ECHONET 通信処理部の動作を開始する。

４．３．４２ MidSuspend

（1）名称

MidSuspend　－　ECHONET 通信ミドルウェアの一時停止要求関数。

（2）機能

通信ミドルウェアの一時停止を要求する。

（3）構文

long MidSuspend(void)

（4）説明［Optional 関数］

ECHONET 通信処理部以下を全て一時停止する。
送信待ち電文、受信待ち電文は、クリアされない。

（5）戻り値

EAPI_NO_ERROR　　：停止成功
EAPI_NOTOPEN：非起動（セッションが未 open）
EAPI_MID_ERROR　　：ECHONET 通信処理部エラー
EAPI_PRO_ERROR　　：プロトコル差異吸収処理部エラー
EAPI_LOW_ ERROR　　：下位通信ソフトウェアエラー

（6）注意事項

動作の再開は、MidWakeUp 関数により行う。

４．３．４３ MidWakeUp

（1）名称

MidWakeUp　－　ECHONET 通信ミドルウェアの動作再開要求関数。

（2）機能

通信ミドルウェアの動作再開を要求する

（3）構文

long MidWakeUp(void)

（4）説明［Optional 関数］

ECHONET 通信処理部以下の全てを動作再開する

（5）戻り値

EAPI_NO_ERROR　　：再開成功
EAPI_NOTOPEN：非起動（セッションが未 open）
EAPI_MID_ERROR　　：ECHONET 通信処理部エラー
EAPI_PRO_ERROR　　：プロトコル差異吸収処理部エラー
EAPI_LOW_ ERROR　　：下位通信ソフトウェアエラー

（6）注意事項

MidSuspend で動作停止中のみ有効とする。

４．３．４４ MidSetSendMulti，MidExtSetSendMulti

（1）名称

MidSetSendMulti, MidExtSetSendMult　－　ECHONET オブジェクトの配列でない複数のプロパティへのデータの書込及び複合電文の送信要求関数。

（2）機能

配列でない複数の ECHONET プロパティにデータを書き込み、複合電文としてサービスを送信する。

（3）構文

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

（4）説明［Optional 関数］

MidSetSendMulti は、id,eoj_code,epc_code によって指定された複数の ECHONET プロパティにデータを書き込み、esv_code のサービスを送信する。
MidExtSetSendMulti も基本的には、MidSetSendMulti と同じ機能を持つが、セキュア通信設定が行える関数である。
データを書き込みたい任意のタイミングで呼び出し可。

id_kind　：[in]ID 種別
　APIVAL_NODE_KIND　：0(機器 ID)
　APIVAL_EA_KIND　：1(ECHONET アドレス)
　APIVAL_BROAD_KIND　：2(同報)
id　：[in]機器 ID または ECHONET アドレスまたは同報種別
seoj_code　：[in]SEOJ コード (下位 3byte のみ使用）
　SEOJ なしの場合は、-1 を設定
deoj_code　：[in]DEOJ コード（下位 3byte のみ使用）
　DEOJ なしの場合は、-1 を設定
esv_code　：[in]ESV コード
　ESV_SetI　：0x60(応答不要プロパティ値書き込み要求)
　ESV_SetC　：0x61(応答要プロパティ値書き込み要求)
　ESV_Get　：0x62(プロパティ値読み出し要求)
　ESV_Inf_Req：0x63(プロパティ値通知要求)
　ESV_INF　：0x73(プロパティ値通知)
opc_code　：[in]EPC の要素数を設定
pdc_code　：[in]各 EPC コードと EDT コードのサイズ情報が入る配列の先頭ポインタ。要素数は、opc_code の数。
epcedt_code　：[in]EPC コードと EDT コードが入る配列の先頭ポインタ。

要素数は、opc_code の数。
（セキュアを追加）

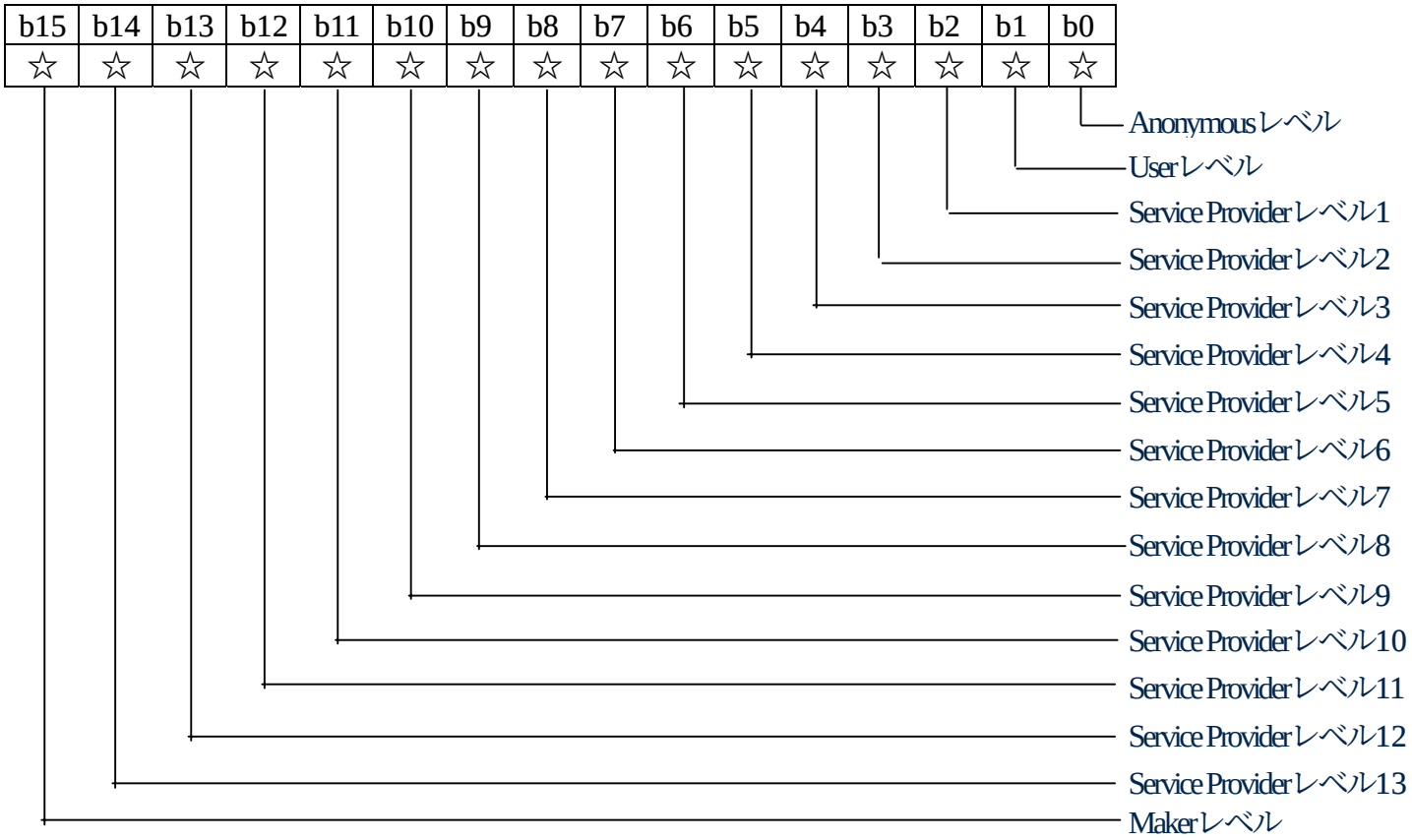
extcont　　：[in]拡張通信オプション

（5）戻り値

EAPI_NO_ERROR　　：設定成功
EAPI_NOTOPEN　　：非起動（セッションが未 open）
EAPI_ILIEGAL_PARAM　：ID_kind　もしくは　esv_code　が不正
EAPI_NOTFOUND_EPC　：プロパティが存在しない
EAPI_DATASIZE_EROR　：書き込みデータサイズが不正
EAPI_NORESOURCE　　：リソース不足
EAPI_NOCONDITION　　：制御不可能なプロパティ
EAPI_MEMBER_EPC　　：配列要素プロパティ
EAPI_NOTSEND　　：未送信データあり
EAPI_TIMEPOUT　　：通信タイムアウト（通信モードが同期の時）
EAPI_ETC_NOCONDITION　　：指定拡張通信機能での書き込み不可能なプロパティ

（6）使用する構造体

```
typedef struct {
        short   ext_hed;    /* この構造体の種類を示すコード。
                            0x0001：セキュア通信指定*/
        short   cipher;         /* 暗号化指定（方式指定含）。
                            0x0000：暗号化無し
                            0x0001：AES-CBC
                            0x0002～0xFFFF：for future reserved */
        short   authent;        /* アクセス制限レベル指定
```



その他：for future reserved ***/**

```
    short   authentication      /* 認証有無指定 */
    long    makerKeyIndex       /* メーカ KeyIndex */
    short   makerKeysize        /* メーカ Key サイズ */
    char    *makerKey           /* メーカ Key */
} EXT_CONT
```

（7）注意事項

特に無し。

４．３．４５ MidGetReceiveEpcMulti

（1）名称

MidGetReceiveEpcMulti－　ECHONETオブジェクトの配列でない複数のプロパティからのデータの読出要求関数

（2）機能

受信のあった配列でない複数の ECHONET プロパティのデータを読み込む。

（3）構文

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

（4）説明

id,eoj_codeによって指定されたオブジェクトの複数のECHONETプロパティの受信データを読み込む。受信データを読み込みたい任意のタイミングで呼び出し可能。

id_kind　：[in]ID 種別
　　APIVAL_NODE_KIND　　：0(機器 ID)
　　APIVAL_EA_KIND　　：1(ECHONET アドレス)
id　　：[in]機器 ID または ECHONET アドレス
eoj_code ： [in] EOJ コード (下位 3byte のみ使用、ない場合は－1)
buff_size ： [in]エリアサイズ
opc_code ： [in]OPC コード
esv_code ： [out]ESV コード
pdc_code ： [out]各 EPC コードと EDT コードのサイズ情報が入る配列の先頭ポインタ。要素数は、opc_code の数
epcedt_code ： [out]EPC コードと EDT コードが入る配列の先頭ポインタ。
eoj_code2 ： [out]通信上の SEOJ コードまたは DEOJ コード（本 eoj_code2 がある場合には、eoj_code が他 Node の EOJ を指定する場合に通信上の DEOJ コードとなり、eoj_code が自 NodeID の EOJ を指定する場合に通信上の SEOJ コードを示す。）

（5）戻り値

EAPI_NO_EROR	：読み込み成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_ILLEGAL_PARAM	：id_kind が不正
EAPI_NOTFOUND_EPC	：プロパティが存在しない
EAPI_NORECEIVE	：受信データが存在しない
EAPI_NOTSEND	：送信待ち中
EAPI_MEMBER_EPC	：配列要素プロパティ
EAPI_DATASIZE_EROR	：データサイズが不正
EAPI_NORESOURCE	：リソース不足

（6）使用する構造体

特に無し。

（7）注意事項

配列要素指定の読み込みは不可。セキュア通信処理されている電文の場合には、複合電文であっても　MidExtGetReceiveEpc によりアプリケーションに通知される。

４．３．４６ MidSetSecureContVal

（1）名称

MidSetSecureContVal　－　セキュア通信用シリアル Key 設定関数。

（2）機能

セキュア通信の初期共有鍵設定に必要となるシリアル Key を設定する。

（3）構文

long MidSetSecureContVal (short　serial_len, unsigned char *serial_key)

（4）説明［Optional 関数］

セキュア通信用の各種設定を行う。

Serial_len　：[in]　シリアル Key データサイズ

serial_key　：[in]　シリアル Key データの先頭ポインタ

（5）戻り値

EAPI_NO_ERROR	：設定成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_DATASIZE_EROR	：書き込みデータサイズが不正
EAPI_NORESOURCE	：リソース不足

（6）使用する構造体

特に無し。

（7）注意事項

特に無し。

４．３．４７ MidStop

（1）名称

MidStop　－　ECHONET 通信ミドルウェアの通信停止要求関数。

（2）機能

通信ミドルウェアに通信停止状態への遷移を要求する。

（3）構文

long MidStop(void)

（4）説明［Optional 関数］

ECHONET 通信処理部以下を通信停止状態にする。
送信待ち電文、受信待ち電文は、破棄される。

（5）戻り値

EAPI_NO_ERROR　　：停止成功
EAPI_NOTOPEN：非起動（セッションが未 open）
EAPI_MID_ERROR　　：ECHONET 通信処理部エラー
EAPI_PRO_ERROR　　：プロトコル差異吸収処理部エラー
EAPI_LOW_ ERROR　　：下位通信ソフトウェアエラー

（6）注意事項

４．３．４８ MidHalt

（1）名称

MidHalt　－　ECHONET 通信ミドルウェアの完全停止要求関数。

（2）機能

通信ミドルウェアに完全停止状態への遷移を要求する。

（3）構文

long MidHalt(void)

（4）説明［Optional 関数］

ECHONET 通信処理部以下を全て停止する。

送信待ち電文、受信待ち電文は、破棄される。

（5）戻り値

EAPI_NO_ERROR　　：停止成功
EAPI_NOTOPEN：非起動（セッションが未 open）
EAPI_MID_ERROR　　：ECHONET 通信処理部エラー
EAPI_PRO_ERROR　　：プロトコル差異吸収処理部エラー
EAPI_LOW_ ERROR　　：下位通信ソフトウェアエラー

（6）注意事項

４．３．４９ MidGetAddressTableDataSize

（1）名称

MidGetAddressTableDataSize　－　下位通信ソフトウェアアドレステーブルデータサイズ取得関数

（2）機能

下位通信ソフトウェアで保持している下位アドレステーブルデータの組数を取得する。

（3）構文

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

（4）説明

出力データはデータ組数のポインタからなる。

device_id　：下位通信ソフトウェア識別情報。

電灯線	0x11～0x1F
特定小電力無線	0x31～0x3F
拡張 HBS	0x41～0x4F
IrDA_Control	0x51～0x5F
LonTalk®	0x61～0x6F
Bluetooth™	0x71～0x7F
Ethernet	0x81～0x8F

data_number　：下位アドレステーブルデータで保持しているアドレステーブルの組数のポインタ。

（5）戻り値

EAPI_NO_ERROR	：取得成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_UNACCEPTABLE	：取得受付手段なしエラー
EAPI_MOMENTARY_ERROR	：一時的エラー

（6）注意事項

なし

４．３．５０ MidGetAddressTableData

（1）　名称

MidGetAddressTableData　－　下位通信ソフトウェアアドレステーブルデータ取得関数

（2）　機能

下位通信ソフトウェアで保持している下位アドレステーブルデータを取得する。

（3）　構文

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

（4）　説明

入力データ(data_number)は、MidGetAddressTableDataSize で取得したアドレステーブルの組数のポインタ。
出力データは、実際に格納したアドレステーブルの組数と、各データ組のハードウェアアドレス、NodeID、及びマスタルータであることを示すフラグからなる構造体の配列データからなる。

device_id　：下位通信ソフトウェア識別情報。
- 電灯線　0x11～0x1F
- 特定小電力無線　0x31～0x3F
- 拡張 HBS　0x41～0x4F
- IrDA_Control　0x51～0x5F
- LonTalk®　0x61～0x6F
- Bluetooth ™　0x71～0x7F
- Ethernet　0x81～0x8F

data_number　：下位アドレステーブルデータで保持しているアドレステーブルの組数のポインタ。

addresstable　：下位アドレステーブルデータで保持しているハードウェアアドレス、NodeID、及びマスタルータであることを示すフラグを収容するアドレステーブル構造体の配列の先頭ポインタ。

（5）　戻り値

EAPI_NO_ERROR	：取得成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_UNACCEPTABLE	：取得受付手段なしエラー
EAPI_MOMENTARY_ERROR	：一時的エラー

（6）　使用する構造体

```
typedef struct {
        unsigned char hardwareaddress_size;     /* ハードウェアアドレスのデータサイズ */
        unsigned char hardwareaddress[8];       /* ハードウェアアドレス。下位バイト詰めで格納。*/
        unsigned char node_id;                  /* NodeID */
        unsigned char masterrouter_flag;        /* 対応するノードがマスタルータであるか否かを示す識別子。マスタルータであれば 1、そうでなければ 0。 */
} ADDRESSTABLE
```

（7）　注意事項

第２引数 data_number　は、入出力兼用のためデータ内容が上書きされるので注意が必要。

４．３．５１ MidSetMasterRouterFlag

（1）名称

MidSetMasterRouterFlag　－　マスタルータ通知関数

（2）機能

通信ミドルウェアに対して、自ノードがマスタルータであるか否かを下位通信ソフトウェアに通知するように要求する。

（3）構文

long MidSetMasterRouterFlag (unsigned char device_id）

（4）説明

device_id　：下位通信ソフトウェア識別情報。

電灯線	0x11～0x1F
特定小電力無線	0x31～0x3F
拡張 HBS	0x41～0x4F
IrDA_Control	0x51～0x5F
LonTalk®	0x61～0x6F
Bluetooth ™	0x71～0x7F
Ethernet	0x81～0x8F

（5）戻り値

EAPI_NO_ERROR	：通知成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_UNACCEPTABLE	：通知受付手段なしエラー
EAPI_MOMENTARY_ERROR	：一時的エラー

（6）注意事項

なし

４．３．５２ MidGetHardwareAddress

（1）名称

MidGetHardwareAddress　－　ハードウェアアドレスデータ取得関数

（2）機能

下位通信ソフトウェアに対して保持しているハードウェアアドレスデータを取得する。

（3）構文

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

（4）説明

出力データはハードウェアアドレスである。

device_id　：下位通信ソフトウェア識別情報。

電灯線	0x11～0x1F
特定小電力無線	0x31～0x3F
拡張 HBS	0x41～0x4F
IrDA_Control	0x51～0x5F
LonTalk®	0x61～0x6F
Bluetooth ™	0x71～0x7F
Ethernet	0x81～0x8F

hardwareaddress_size：ハードウェアアドレスのデータサイズのポインタ。

hardwareaddress ：自ノードのハードウェアアドレスのポインタ。

（5）戻り値

EAPI_NO_ERROR	：取得成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_UNACCEPTABLE	：取得受付手段なしエラー
EAPI_MOMENTARY_ERROR	：一時的エラー

（6）注意事項

なし

４．３．５３ MidGetReceiveCheckEpcMulti

（1）名称

MidGetReceiveCheckEpcMulti　－　複合電文に対するデータ読出確認関数

（2）機能

受信した複合電文を確認する。

（3）構文

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

（4）説明［Optional 関数］

MidGetReceiveCheckEpcMulti は、受信した複合電文を受信順にリストアップします。受信を確認したい任意のタイミングで呼び出しが可能です。

buff_num	：[in]リストアップ最大要素数
id	：[out]機器 ID(-1：ID 管理無し)
EA	：[out]ECHONET アドレス
eoj_code	：[out]EOJ コード (下位 3byte のみ使用)
esv_code	：[out] ESV コード格納エリア
out_num	：[out]リストアップ数格納エリア

（5）戻り値

EAPI_NO_ERROR	：リストアップ成功
EAPI_NOTOPEN	：非起動（通信ミドルウェアが初期化されていません）
EAPI_ILLEGAL_PARAM	：buff_num が不正（buff_num≦0）またはポインタが NULL

（6）注意事項

なし。

４．３．５４ MidGetDevID

（1）　名称

MidGetDevID　－　下位通信ソフトウェア搭載情報要求関数

（2）　機能

操作が可能な下位通信ソフトウェアの数、および種類の識別を示す下位通信ソフトウェア ID 情報を要求する。

（3）　構文

```
long      MidGetDevID（
          unsigned char     *device_num      /* [OUT] 操作可能な下位通信ソフトウェア数*/
          unsigned char     *device_idset    /* [OUT] 操作可能な下位通信ソフトウェア ID */
）
```

（4）　説明

*device_num　：操作可能な下位通信ソフト数へのポインタ

*device_idset　：操作可能な下位通信ソフトウェア ID 情報へのポインタ。ポインタの先には、device_num で指定された数の情報が存在する。下位通信ソフトウェアの種類と対応する下位通信ソフトウェア ID の関係は、以下の通りである。

電灯線	0x11～0x1F
特定小電力無線	0x31～0x3F
拡張 HBS	0x41～0x4F
IrDA_Control	0x51～0x5F
LonTalk®	0x61～0x6F
Bluetooth™	0x71～0x7F
Ethernet	0x81～0x8F

（5）　戻り値

EAPI_NO_ERROR　：設定成功

EAPI_NOTOPEN　：非起動（セッションが未 open）

（6）　使用する構造体

特に無し。

（7）　注意事項・制限事項

この関数は、「初期化要求関数：MidInit」や「動作開始要求関数：MidRequestRun」より前に呼び出されることを前提とする。

４．３．５５ MidGetLastSendError

（1）　名称

MidGetLastSendError　－　最新送信エラー情報取得関数

（2）　機能

ECHONET 通信ミドルウェアが保持する最新の ECHONET 電文送信エラー情報を取得する。

（3）　構文

```
long      MidGetLastSendError（
          unsigned char      *last_err /* [OUT]  最新送信エラー情報*/
）
```

（4）　説明

*last_err　　最新送信エラー情報へのポインタ
0x00：送信成功
0x01：送信中止時
0x02：送信結果取得タイムアウト
0x03：下位通信ソフトウェア内部エラー
0x04：機器アダプタ処理失敗
0x05：下位通信ソフトウェアバッファフルエラー
0x06：下位通信ソフトウェアバッファサイズエラー
0x07：下位通信ソフトウェア送信エラー
0x08～0xFE　：for future reserved
0xFF：無応答

（5）　戻り値

EAPI_NO_ERROR	：取得成功
EAPI_NOTOPEN	：非起動（セッションが未 open）
EAPI_MID_ERROR	：ECHONET 通信処理部エラー

（6）　使用する構造体

特に無し。

（7）　注意事項・制限事項

特に無し。

第５章 レベル２ECHONET 基本 API 仕様（Java 言語版）

５．１ 基本的な考え方

本章では、Java 言語で書かれたアプリケーション向けの基本API仕様について規定する。本API仕様は、集中制御装置に搭載されJava 言語で書かれたアプリケーション向けのAPI仕様であり、そのアプリケーションの特長として以下を想定している。

・広域ネットワーク経由で配信され各家庭の集中制御装置上にロードされ動作する。
・集中制御装置はマルチベンダで開発され得る。
・本アプリケーションは、配信される先の集中制御装置の開発ベンダが何処であるかを意識せずに動作できることが望ましい。

以上のような想定のもと、本 API の規格化方針は以下とする。

（1）他機器の監視・制御を行うための API として、要求・応答のタイミングを、同期的に行うものと非同期的に行うものの２タイプを用意する。アプリケーションプログラムの目的やそれを書く人のスキルに応じて、どちらを用いるかは適宜選択できるようにする。

（2）他機器からのサービス要求に対する応答の処理の記述を、アプリケーションプログラマに期待する。

（3）基本的にオプション機能は無しとする。ただし、アプリケーションが ECHONET 通信ミドルウェア自身のオプション機能を使おうとする際には、それがサポートされていない場合も想定したプログラム記述が必要である。

（4）ECHONET 通信ミドルウェアは、上位で動作するアプリケーションが、それぞれ独立して動作できる環境を提供する。オブジェクト指向の考え方に基づき、各々のアプリケーションが保持するデータは各々のアプリケーションが独自に管理する。例えば、同一ノード上にアプリケーションA，Bが搭載され、アプリケーションAは ECHONET 通信ミドルウェアが他機器のデータを取得したとしても、アプリケーションBが取得したデータの値がそれと同じであるとは限らない。

（5）ECHONET 通信ミドルウェア内部で他機器の状態を保存することは行わない。これは、アプリケーションは集中制御装置に搭載され、アプリケーションレベルでも状態を保持する使われ方を想定し、アプリケーションに任せる方が効率的である、という判断によるものである。

（6）ドメイン内に存在するすべての ECHONET 機器の情報を管理するなどの所謂ネームサービス、アクセス制限、ネットワークトラヒックコントロールなどのより高度なサービスは、本 API よりも上位のサービスミドルウェアにて実現するものとする。これについては今後規格化を検討していく。

（7）プロファイルオブジェクト、通信定義オブジェクトのプログラムは、通信ミドルウェアを製品として実際に開発する開発者によって記述される。この初期化のための API や、ネットワーク経由でのアクセスが許されていないプロパティへのア

クセスのための API などは、規定しない。これは実装の問題である。

（8）セキュア通信のための API を提供するために、Ver.2.00 仕様に対して以下の改造を行なう。

・関連するメソッド各々に対し、セキュア通信を行なう構文形式を追加する。なお、例外については、想定されるケースを新たに追加する。

・セキュア通信の指定を表現する ECHONET セキュア通信オプションクラス EN_SecureOpt を新たに定義する。

・EN_Const クラスに、セキュア通信のための定数定義を追加する。

（9）複合電文用の API では、要求側のアプリケーションには複合電文を意識させた API とし、応答側のアプリケーションには複合電文を意識させない API とする。

５．２ API 構成

５．２．１ API のクラス

API は次のクラスから構成される。

EN_Object クラス	ECHONET オブジェクトの管理
EN_Node クラス	ノードとイベントの管理
EN_Property クラス	プロパティのラッパー
EN_Packet クラス	イベントラッパー
EN_EventListener インタフェース	イベントリスナ
EN_Exception 例外クラス	例外を表現するクラス
EN_Const インタフェース	API で使用する定数の定義
EN_SecureOpt クラス	セキュア通信のオプション指定

５．２．２ 各クラスの関連

各クラスの関連を図５．１に示す。ECHONET ノードは EN_Node クラスによって管理されイベント（受信電文）を受けると、ユーザーアプリケーションのメソッドを呼び出す。EN_Object は ECHONET ブジェクトを抽象化している。アプリケーションが EN_Object に対してプロパティアクセスメソッドを呼び出すと、実際に電文が発行される。受信電文は、プロパティ部(EDT)を表す EN_Property クラスとプロパティ以外の部分を表す EN_Packet クラスで管理される。以下、各クラスについて説明する。なお、図５．１は ECHONET 通信ミドルウェアの実装を規定するものではない。

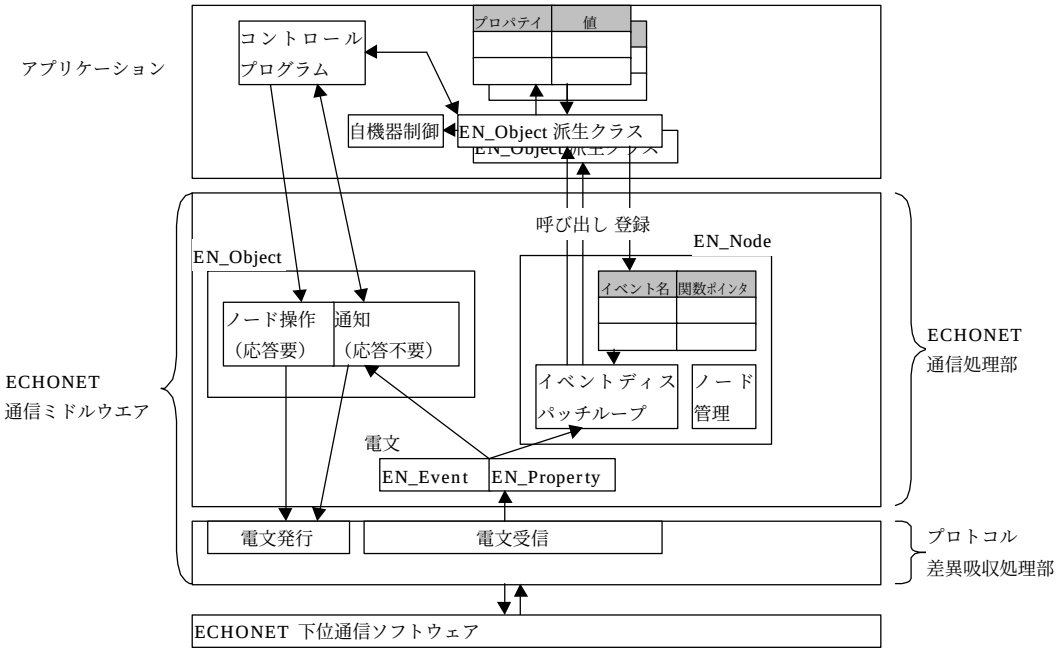


図５．１　各クラスの関連

５．２．３ EN_Object クラス

EN_Object クラスは ECHONET オブジェクトを抽象化したクラスである。役割は、ECHONET オブジェクトが自己のアプリケーションにある場合と他アプリケーションにある場合により異なる。

(注意)
ここでいう「他アプリケーション」は、自己のアプリケーション（以下、「自己アプリケーション」）以外を指す。なお、複数のアプリケーションが同一の ECHONET ノード上で動作している場合、それぞれの ECHONET アドレスは同一であっても、他のアプリケーション上のオブジェクトは「他アプリケーション」と考える。
例えば、下図において、ObjA にとって ObjB は自己アプリケーションであるが、ObjC, ObjD は他アプリケーションであるとする。なお、下図において ECHONET 通信ミドルウェアと ECHONET 通信ミドルウェア API は別々のブロックで書かれているが、この図は概念を説明するものであり、実装を規定するものではない。

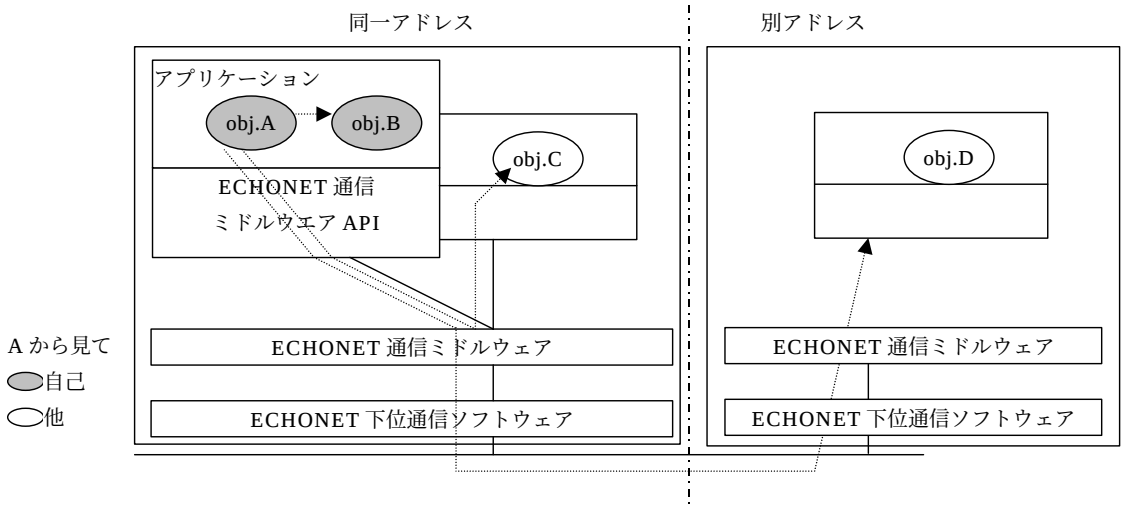


図５．２　自己アプリケーションと他アプリケーションの関連

自己 ECHONET オブジェクト
これは、たとえばエアコンが ECHONET エアコンオブジェクトを作成する場合に相当する。この場合アプリケーションは EN_Object を派生し新たなクラスを作成する。さらに、他アプリケーションから自己 ECHONET オブジェクトへのプロパティアクセス要求に答えるために、callbackReadMyProperty、callbackWriteMyProperty メソッドなどの”callback”で始まる名前のメソッド.をオーバライドする。最後にインスタンスを作成し、EN_Node クラスが提供するノードオブジェクトに、作成したインスタンスを登録する。
オーバライドした２つのメソッドは、EN_Node が他アプリケーションから自己 ECHONET オブジェクトのプロパティアクセス要求電文を受け取った場合など、プロパティの取得・設定が必要になったときにアクセスされる。

他 ECHONET オブジェクト

これは、たとえばコントローラが他アプリケーションのエアコン ECHONET オブジェクトを操作する場合に、自己アプリケーション内にこのエアコンオブジェクトに対応し作成されるエアコンオブジェクトのインスタンスに相当する。EN_Object のインスタンスは単にアクセスしたい ECHONET オブジェクトが存在する ECHONET アドレスとその EOJ（第 2 部 4.2.6）を持っているに過ぎない。このインスタンスの getProperty、setProperty メソッドは、API が提供するオブジェクトプロパティにアクセスするためのメソッドであり、API は実際に電文を発行し、必要な場合は応答を待ち、その応答をリターンコードとしてアプリケーションに返す。

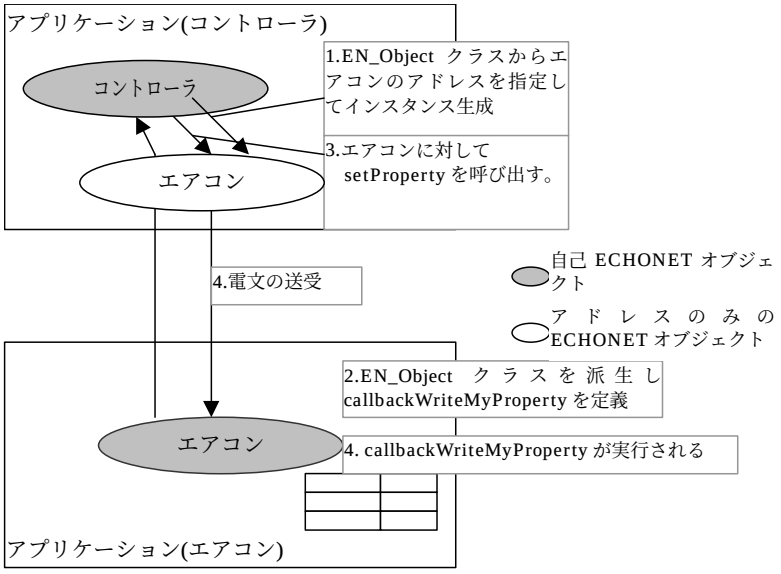


図５．３　自己 ECHONET オブジェクトと他 ECHONET オブジェクトとの関連

５．２．４ EN_Node クラス

EN_Node クラスは ECHONET ノードを抽象化したクラスである。これは ECHONET ノードとイベントの管理を行う。イベントとは自ノードに到着した ECHONET 電文のことを言う。

アプリケーションは、起動時に EN_Node のインスタンスをただ 1 つ作成しなければならない。EN_Node のコンストラクタではイベント待ちを行うイベントディスパッチループを実行するスレッド（イベントスレッド）を作成する。EN_Node で定義される add で始まるメソッドにより、インスタンスを登録できる。イベントスレッドは、イベントが発生すると、イベントに対応づけられたインスタンスの特定のメソッドを呼び出し、その処理が終わるとまたイベントの待ちを行う。

イベントは、自己 ECHONET オブジェクトに対するアクセス要求イベント、他アプリケーションが発した通知電文（応答電文を含む）を受け取ることによって生じる通知イベント、エラー通知イベントがある。

アクセス要求イベント

アプリケーションは、自己 ECHONET オブジェクトの実装として、EN_Object を継承して作成したクラスのインスタンスを作成し、EN_Node.addPropertyEventListener メソッドにより登録する。アクセス要求イベントが発生した場合、イベントスレッドは DEOJ（第 2 部 4.2.6 節）を調べ、EOJ が一致するインスタンスの callbackWriteMyProperty または callbackReadMyProperty メソッド（配列扱いのプロパティについては、後述）を呼び出す。API はこのメソッドのリターンコードから必要に応じて応答電文を送信する。

エラー通知イベント

アプリケーションは、エラー通知を受けるためにクラス EN_Object を継承し、メソッド callbackNotifyError をオーバライドする。そのクラスのインスタンスを、EN_Node.addNotifyErrorEventListener メソッドにより登録する。エラー通知イベントが発生した場合、イベントスレッドは呼び出すべきインスタンスを決め、callbackNotifyError メソッドを呼び出す。

５．２．５ EN_Property クラス

ECHONET オブジェクトが取り扱うプロパティは、ECHONET 電文中のバイト列 EDT(第 2 部 4.2.9)エリアに格納される値であるが、これを Java 言語の byte 型や int 型として参照したり作成できたりすると便利である。EN_Property クラスは EDT を保持し、その値を byte や int で作成、参照するためのメソッドを提供する。
これにより、アプリケーションは多様な型をもつプロパティを簡単な操作で扱うことができる一方、API は、プロパティの型によらないインタフェースの提供が可能になる。

５．２．６ EN_Packet クラス

イベントとは自ノードに到着した ECHONET 電文のことを言い、EN_Packet はイベントデータを保持するメンバをもつクラスである。これはイベント発生時に呼ばれるアプリケーション定義のメソッドに対する引数に用いられ、この EN_Packet クラスのインスタンスによってイベントの情報は API からアプリケーションに渡される。

５．２．７ EN_Exception 例外クラス

他ノードのオブジェクトにアクセスするため、getProperty、setProperty を呼び出した場合、そのアクセス要求が処理不可の場合、処理不可応答電文を受ける。この場合、EN_Exception 型の例外がスローされる。アプリケーションは、例外を補足(catch)し、アクセス要求不可の場合の処理を行う。

５．２．８ EN_EventListener インタフェース

イベントを受け付けるために必要となるインタフェース型である。EN_Object はこのインタフェースをインプリメントしているので、ユーザはこのインタフェースを意識する必要

はない。

５．２．９ EN_Const インタフェース

API で使用する定数群が定義されている。本定数を使用する API の各クラスは本インタフェースをインプリメントしている。API から返される定数を参照する各アプリケーションは、本インタフェースをインプリメントする必要がある。

５．２．１０ EN_SecureOpt クラス

セキュア通信を使用する際に、その実行形式を指示するオプションを表現するクラスである。

５．２．１１ EN_CpException 例外クラス

他ノードのオブジェクトにアクセスするため、getProperty、setProperty を複合電文形式で呼び出した場合、そのアクセス要求が処理不可の場合、処理不可応答電文を受ける。この場合、EN_CpException 型の例外がスローされる。アプリケーションは、例外を補足(catch)し、アクセス要求不可の場合の処理を行う。

５．３ API 詳細仕様

本章で述べる API 詳細仕様では、データ型を以下のように規定する。

“byte”	：符号付１バイト整数型
“short”	：符号付２バイト整数型
“int”	：符号付４バイト整数型
“long”	：符号付 8 バイト整数型
“boolean”	：論理型
“String”	：文字列型

以下で示すクラスの詳細では、クラスでカプセル化されるメソッド、メンバ（private メソッド、メンバ）は示していない。これは API 実装の問題であり、API 実装時に適時決めて良い。

５．３．１ EN_Object クラス

（1）名称

EN_Object　　－　ECHONET オブジェクトクラス

（2）機能

本クラスは、自ノードまたは他ノードの ECHONET オブジェクトのプロパティに対する操作を提供する。アプリケーションは、このクラスを用いることにより、ECHONET のオブジェクトを自ノードと他ノードの区別なく操作することができる。

アプリケーションは、自己が持つ ECHONET オブジェクトをこのクラスのサブクラスとして定義し、callbackReadMyProperty、callbackWriteMyProperty メソッドを適切にオーバライドしなければならない。他 ECHONET オブジェクトに対する操作は、そのノードに対して電文を発行し、必要であれば応答を待つ。自己 ECHONET オブジェクトに対する操作は、オーバライドした callbackReadMyProperty、callbackWriteMyProperty などの”callback”で始まるメソッドが呼ばれる。

（3）構文

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

（4）注意

・アプリケーションは、自ノード内の他のアプリケーションの ECHONET オブジェクトと同一の ECHONET オブジェクトコードをもつインスタンスを複数生成してはならない。自ノード内で複数のアプリケーションが共存している場合には注意が必要である。

・他ノードおよび自ノードの ECHONET アドレスが変化した場合、自動の追従は行わない。

5.3.1.1 EN_Object

（1）名称

EN_Object　　－　ECHONET オブジェクトのコンストラクタ

（2）機能

ECHONET オブジェクトの構築を行う。

（3）構文

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

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

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

構文4:　public EN_Object(EN_Object *eno*) throws EN_Exception;

（4）説明

ECHONET オブジェクトの構築を行う。

EN_Object はメンバにアドレス情報を持ち、おもに電文中の EA、EOJ、DEA、EHDb3 の作成に用いられる。構文 1 は自己 ECHONET オブジェクトのアドレスを、構文 2 は単一のノードのアドレスを、構文 3 は同報アドレス（ドメイン内同報、サブネット内同報のアドレス）をそれぞれ表す。構文 4 は引数に依存する。

EA	ECHONET アドレス。*EA* の下位 2 byte のうち、上位 8bit が NetID、下位 8bit が NodeID である。省略した場合（構文１）または値 EN_Object．MYSELF_NODE を指定した場合は自己 ECHONET オブジェクトを指すアドレスとなる。
EOJ	ECHONET オブジェクトコード(第 2 部 4.2.6)。下位 24bit を用い、クラスグループ、オブジェクトクラスコード、オブジェクトインスタンスコードを指定する。オブジェクトインスタンスコードを 0 にした場合は、クラスグループコード、オブジェクトクラスコードで指定されたすべてのインスタンスに対する同報を意味する特別な意味となる。また、0xFFFFFFFF を指定した場合（このコード 0xFFFFFFFF を 以下ワイルドカードコードと呼ぶこととする）には、すべてのクラスグループ、オブジェクトクラスコード、オブジェクトインスタンスコードを含む特別な意味となる。
eno	コピー元 EN_Object インスタンス。既存の EN_Object インスタンスと同じアドレス情報を持ったインスタンスを作成する場合に指定する。引き継ぐのはアドレス情報のみである。
broadcastArea	同報種別指定コード(第 2 部 4.2.2)を指定する。
broadcastGroup	同報対象指定コード(第 2 部 4.2.2)を指定する。

（5）リターンコード

なし

（6）例外

EAPI_ILLEGAL_PARAM　　：*EOJ* 異常 (EOJ として 3 byte を超える値を指定した場合)、*EA* 異常（EA として 2 byte を

超える値を指定した場合)、同報種別、同報対象指定コード誤り（規格で規定されていないコードを指定した場合)

（7） 注意

・アプリケーションが自己ECHONETオブジェクトのためにオーバライドするときには、かならずスーパークラスの呼び出しを処理前に行わなければならない。

・*EA* に MYSELF_NODE を指定した場合と EN_Node.getEA()の値を指定した場合の動作は異なる。前者のインスタンスに getProperty を呼び出した場合は、callbackReadMyProperty に呼び出しに変換されるが、後者はされない（ただし最終的に callbackReadMyProperty の呼び出しが行われることもありうる)。後者は自ノードの他アプリケーション上のECHONETオブジェクトにアクセスするために用いる。

5.3.1.2 setProperty

（1）名称

setProperty　　－　プロパティ値設定サービスの実行

（2）機能

ECHONET オブジェクトに対してプロパティ設定サービスを実行する。

（3）構文

構文1：

```
public void setProperty(
   EN_Object        sourceObject,    //発信元 ECHONET オブジェクト
   int              EPC,             //EPC
   EN_Property      p,               //プロパティ
   boolean          res,             //応答が必要なら true
   long       timeout          //タイムアウト時間
) throws EN_Exception;
```

構文2：

```
public void setProperty(
   int              EPC,             //EPC
   EN_Property      p,               //プロパティ
   boolean          res              //応答が必要なら true
) throws EN_Exception;
```

構文3：

```
public void setProperty(
   EN_Object        sourceObject,    //発信元 ECHONET オブジェクト
   int              EPC,             //EPC
   EN_Property      p,               //プロパティ
   long       timeout,         //タイムアウト時間
   EN_SecureOpt     secopt           //セキュア通信オプション
) throws EN_Exception;
```

構文4：

```
public void setProperty(
   int              EPC,             //EPC
   EN_Property      p,               //プロパティ
   EN_SecureOpt     secopt           //セキュア通信オプション
) throws EN_Exception;
```

構文5：

```
public void setProperty(
   EN_Object        sourceObject,    //発信元 ECHONET オブジェクト
   int              EPCnum,          //EPC の数
   int              EPC[],           //EPC の組
   EN_Property      p[],             //プロパティの組
   boolean          res,             //応答が必要なら true
```

```
    long        timeout         //タイムアウト時間
) throws EN_CpException;
```

構文６：

```
public void setProperty(
    int             EPCnum,     //EPC の数
    int             EPC[],      //EPC の組
    EN_Property     p[],        //プロパティの組
    boolean         res         //応答が必要なら true
) throws EN_CpException;
```

構文７：

```
public void setProperty(
    EN_Object       sourceObject,   //発信元 ECHONET オブジェクト
    int             EPCnum,     //EPC の数
    int             EPC[],      //EPC の組
    EN_Property     p[],        //プロパティの組
    long        timeout,        //タイムアウト時間
    EN_SecureOpt    secopt      //セキュア通信オプション
) throws EN_CpException;
```

構文８：

```
public void setProperty(
    int             EPCnum,     //EPC の数
    int             EPC[],      //EPC の組
    EN_Property     p[],        //プロパティの組
    EN_SecureOpt    secopt      //セキュア通信オプション
) throws EN_CpException;
```

（4）説明

ECHONET オブジェクトに対してプロパティ値設定サービスの実行を行う。構文１は送信元オブジェクトの指定ありの ECHONET 電文を作成するものであり、構文２は送信元オブジェクトの指定なしの ECHONET 電文を作成するものである。構文３，４は、セキュア通信を使用するためのものである。構文５～８は、それぞれ構文１～４を複合電文で行なうためのものである。

(a)this のアドレスが他アプリケーションを指している場合、他アプリケーションに対して電文を発行する。引数 *res* が true の場合は応答を待つ。このとき、SEA と SEOJ は *sourceObject* から、DEA と DEOJ が this から作成される。ただし、this のアドレスが同報（オブジェクトインスタンスコード=0 を含む）であれば、*res* の値によらず応答を待たない。なお、他アプリケーションが同一ノード内にある場合、実際に電文をネットワークに流すか否かは実装依存である。

(b)自己 ECHONET オブジェクトに対する操作を行なう場合には、this.callbackWriteMyProperty が呼ばれる。callbackWriteMyProperty の引数は、(a)と同様に作成される。

(c) *res* の値が true の場合は応答要電文(ESV=0x61)、false の場合は応答不要電文を発行する。*res* の値が true の場合は、応答を *timeout* 時間だけ待つ。*res* の値が false の場合は、応答を待たない。この場合、相手 ECHONET オブジェクトからは、正常処理された場合には応答は返って来ないが、処理不可の場合には処理不可応答電文が返される。これをアプリケーションが受信するには、予め後述の EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

(d) *timeout* 値が 0 である場合、あるいは構文２、６の場合には、応答を待たない。この場合、応答は通知イベントで取得できる。ただし、アプリケーションは予め後述の EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

(e) 構文５～８において、*EPC[i] と p[i] とは対応しているものとする（つまり、EPC[i] に対して p[i] を設定する）。*

(f) 構文５～８において、相手先から処理不可応答電文が返ってきた場合には API は例外を発生させるので、アプリケーションはこれをキャッチすることによって、複合電文形式で要求した複数の処理のうちどれが処理できなかったがわかる。

sourceObject	送信元オブジェクトの指定（すなわち、自己 ECHONET オブジェクト）。
EPCnum	書き込むプロパティの数。
EPC	EPC(第 2 部 4.2.7)の値。
P	書き込むプロパティ値。
res	応答が必要なら true
timeout	タイムアウト時間。単位はミリ秒とし、0～20000 が指定できる。ただし、実際に計測される時間は、その処理系に依存する。同報の場合には、0 を指定すること。0 以外のタイムアウト時間を指定しても 0 として処理する。
secopt	セキュア通信指定オプション。

（５）リターンコード

なし

（６）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NORESOURCE	：送信バッファフルのため送信を受け付けない
EAPI_TIMEOUT	：タイムアウト
EAPI_NOTSEND	：原因不明のエラーにより未送信データ有り
EAPI_NOTOPERATIVE	：処理不可応答電文受信
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー
EAPI_SEC_ERROR	：セキュア通信のエラー（認証エラー）

（７）注意

・*res* を true にし *timeout* 値を 0 に指定した場合には応答を待たず即リターンするが、

この応答は通知イベントで取得できる。宛先ノードへの要求と非同期に応答を受信し処理するようなプログラムは、この仕組みを利用するとよい。

・同一オブジェクト・同一プロパティに対する要求を、同一オブジェクトから複数同時に処理した場合の結果は保証しない。この場合、応答メッセージが同一になり、API はそれを区別できない。

・タイムアウト発生後、相手オブジェクトから正常な応答電文が返ってきた場合には、アプリケーションには通知イベントで返す。ただし、アプリケーションがこれを受取るには、予め EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

・*timeout* (!= 0) 指定をして応答待ちをした EN_Object には、そのメソッドに応答として返すが、これがトリガとなって引き起こされるイベントは、その応答が返るべき全ての EN_Object（但し同一アプリケーション）に対して応答が返る。なお、同一オブジェクトについては、メソッドに応答を返した以外には、リスナーが登録されていたとしても配信は行なわない。

・構文５～８において、アプリケーションから要求された複数の EPC[]を実際にどのように複合電文に分割するかについては、API の実装に任される。

5.3.1.3 getProperty

（1）名称

getProperty　　－　プロパティ取得サービスの実行

（2）機能

ECHONET オブジェクトに対してプロパティ取得サービスを実行する。

（3）構文

構文1：

```
public EN_Property getProperty(
   EN_Object        sourceObject,    //発信元 ECHONET オブジェクト
   int              EPC,             //EPC
   boolean          req_broadcast,   //プロパティ値通知要求サービスを
                                     //実行する場合には true
   long        timeout          //タイムアウト時間
) throws EN_Exception;
```

構文2：

```
public EN_Property getProperty(
   int              EPC,             //EPC
   boolean          req_broadcast    //プロパティ値通知要求サービスを
                                     //実行する場合には true
) throws EN_Exception;
```

構文3：

```
public EN_Property getProperty(
   EN_Object        sourceObject,    //発信元 ECHONET オブジェクト
   int              EPC,             //EPC
   boolean          req_broadcast,   //プロパティ値通知要求サービスを
                                     //実行する場合には true
   long        timeout,         //タイムアウト時間
   EN_SecureOpt     secopt           //セキュア通信オプション
) throws EN_Exception;
```

構文4：

```
public EN_Property getProperty(
   int              EPC,             //EPC
   boolean          req_broadcast,   //プロパティ値通知要求サービスを
                                     //実行する場合には true
   EN_SecureOpt     secopt           //セキュア通信オプション
) throws EN_Exception;
```

構文5：

```
public EN_Property[] getProperty(
   EN_Object        sourceObject,    //発信元 ECHONET オブジェクト
   int              EPCnum,          //EPC の数
   int              EPC[],           //EPC の組
```

```
    long        timeout          //タイムアウト時間
) throws EN_CpException;
```

構文６：

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

構文７：

```
public EN_Property[] getProperty(
    EN_Object       sourceObject,  //発信元 ECHONET オブジェクト
    int             EPCnum,      //EPC の数
    int             EPC[],       //EPC の組
    long        timeout,         //タイムアウト時間
    EN_SecureOpt    secopt       //セキュア通信オプション
) throws EN_Exception;
```

構文８：

```
public EN_Property[] getProperty(
    int             EPCnum,      //EPC の数
    int             EPC[],       //EPC の組
    EN_SecureOpt    secopt       //セキュア通信オプション
) throws EN_Exception;
```

（４）説明

this が指す ECHONET オブジェクトに対してプロパティ取得サービスの実行を行う。取得したプロパティを返す。構文１は送信元オブジェクトの指定ありの ECHONET 電文を作成するものであり、構文２は送信元オブジェクトの指定なしの ECHONET 電文を作成するものである。構文３，４は、セキュア通信を使用するためのものである。構文５～８は、それぞれ構文１～４を複合電文で行なうためのものである。

(a)this のアドレスが他アプリケーションを指している場合、他アプリケーションに対して電文を発行し応答を待つ。このとき、SEA と SEOJ は *sourceObject* から、DEA と DEOJ が this から作成される。ただし、this のアドレスが同報（オブジェクトインスタンスコード=0 を含む）であるか、または *req_broadcast* でプロパティ値通知要求サービス実行を指定した場合には、応答を待たない。この場合、応答は通知イベントで取得できる。ただし、アプリケーションは予め後述の EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。なお、他アプリケーションが同一ノード内にある場合、実際に電文をネットワークに流すか否かは実装依存である。

(b)this のアドレスが自ノードかつ自己 ECHONET オブジェクトのアドレスである場合は、this.callbackReadMyProperty が呼ばれる。なお、callbackReadMyProperty の引数は、(a)と同様に作成される。

(c) this のアドレスが自ノードでありかつ自己 ECHONET オブジェクトのアド

レスでない場合は、自ノードの他の適切な EN_Object が呼ばれる。

(d)*timeout* 値が 0 である場合、あるいは構文２、６の場合には、応答を待たない。この場合、応答は通知イベントで取得できる。ただし、アプリケーションは予め後述の EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

(e) プロパティ値通知要求サービスは、本メソッドにより実行可能。

(f) 構文５～８において、相手先から処理不可応答電文が返ってきた場合には API は例外を発生させるので、アプリケーションはこれをキャッチすることによって、複合電文形式で要求した複数の処理のうちどれが処理できなかったがわかる。

sourceObject	送信元オブジェクトの指定（すなわち、自己 ECHONET オブジェクト）。
EPCnum	読み出すプロパティの数。
EPC	EPC(第 2 部 4.2.7)の値。
req_broadcast	プロパティ値通知要求サービスを実行するかしないかの指定。する場合には true を指定。
timeout	タイムアウト時間。単位はミリ秒とし、0～20000 が指定できる。ただし、実際に計測される時間は、その処理系に依存する。同報または同報通知要求の場合には、0 を指定すること。0 以外のタイムアウト時間を指定しても 0 として処理する。
secopt	セキュア通信指定オプション。

（５）リターンコード

取得したプロパティ値（構文５～８の場合はその配列）。ただし、同報アドレス（ドメイン内同報、サブネット内同報のアドレス）に対する操作の場合、*timeout* に 0 を指定した場合、および構文２、構文４の場合は null とする。

構文５～８の場合、*EPCnum と同数のプロパティ値が返る。また、その並びについては、EN_Property[i] と EPC[i] とは対応しているものとする。ただし*、構文６、構文８の場合は null とする。

（６）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NORESOURCE	：送信バッファフルのため送信を受け付けない
EAPI_TIMEOUT	：タイムアウト
EAPI_NOTSEND	：原因不明のエラーにより未送信データ有り
EAPI_NOTOPERATIVE	：処理不可応答電文受信
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー
EAPI_SEC_ERROR	：セキュア通信のエラー（認証エラー）

（７）注意

・同一オブジェクト・同一プロパティに対する要求を、同一オブジェクトから複数同時に処理した場合の結果は保証しない。この場合、応答メッセージが同一になり、API

はそれを区別できない。

・this のアドレスを明示的に１つに指定している場合（同報でない場合）、取得したプロパティ値がリターンコードで返る。this が同報アドレス（ドメイン内同報、サブネット内同報のアドレス）の場合、または同報通知要求の場合には、リターンコードは null である。

・タイムアウト発生後、相手オブジェクトから正常な応答電文が返ってきた場合には、アプリケーションには通知イベントで返す。ただし、アプリケーションがこれを受取るには、予め EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

・*timeout* 値を 0 に指定した場合には応答を待たず即リターンするが、この応答は通知イベントで取得できる。宛先ノードへの要求と非同期に応答を受信し処理するようなプログラムは、この仕組みを利用するとよい。

・*timeout* (!= 0) 指定をして応答待ちをした EN_Object には、そのメソッドに応答として返すが、これがトリガとなって引き起こされるイベントは、その応答が返るべき全ての EN_Object（但し同一アプリケーション）に対して応答が返る。なお、同一オブジェクトについては、メソッドに応答を返した以外には、リスナーが登録されていたとしても配信は行なわない。

・構文５～８において、アプリケーションから要求された複数の EPC[]を実際にどのように複合電文に分割するかについては、API の実装に任される。分割した複数の複合電文のうち少なくとも１つに対して処理不可応答電文が返ってきた場合には、API は例外を発生させる。アプリケーションは、応答電文が ECHONET 電文の最大長を超えないように EPC[]を指定して本メソッドを呼び出すことが望ましい。

5.3.1.4 infProperty

（1）名称

infProperty　　－　プロパティ通知の発行

（2）機能

アプリケーションから通知電文を発行する。

（3）構文

構文1：

```
public void infProperty(
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC              //EPC
) throws EN_Exception;
```

構文2：

```
public void infProperty(
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   EN_Property       p                //プロパティ
) throws EN_Exception;
```

構文3：

```
public void infProperty(
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   EN_SecureOpt      secopt           //セキュア通信オプション
) throws EN_Exception;
```

構文4：

```
public void infProperty(
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   EN_Property       p,               //プロパティ
   EN_SecureOpt      secopt           //セキュア通信オプション
) throws EN_Exception;
```

構文5：

```
public void infProperty(
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   boolean           res,             //応答が必要なら true
   long              timeout          //タイムアウト時間
) throws EN_Exception;
```

構文6：

```
public void infProperty(
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
```

```
    EN_Property       p,          //プロパティ
    boolean           res,        //応答が必要なら true
    long              timeout     //タイムアウト時間
) throws EN_Exception;
```

構文７：

```
public void infProperty(
    EN_Object         sourceObject,  //発信元 ECHONET オブジェクト
    int               EPC,           //EPC
    boolean           res,           //応答が必要なら true
    long              timeout,       //タイムアウト時間
    EN_SecureOpt      secopt         //セキュア通信オプション
) throws EN_Exception;
```

構文８：

```
public void infProperty(
    EN_Object         sourceObject,  //発信元 ECHONET オブジェクト
    int               EPC,           //EPC
    EN_Property       p,             //プロパティ
    boolean           res,           //応答が必要なら true
    long              timeout,       //タイムアウト時間
    EN_SecureOpt      secopt         //セキュア通信オプション
) throws EN_Exception;
```

構文９：

```
public void infProperty(
    EN_Object         sourceObject,  //発信元 ECHONET オブジェクト
    int               EPCnum,        //EPC の数
    int               EPC[],         //EPC の組
    boolean           res,           //応答が必要なら true
    long              timeout        //タイムアウト時間
) throws EN_Exception;
```

構文１０：

```
public void infProperty(
    EN_Object         sourceObject,  //発信元 ECHONET オブジェクト
    int               EPCnum,        //EPC の数
    int               EPC[],         //EPC の組
    EN_Property       p[],           //プロパティの組
    boolean           res,           //応答が必要なら true
    long              timeout        //タイムアウト時間
) throws EN_Exception;
```

構文１１：

```
public void infProperty(
    EN_Object         sourceObject,  //発信元 ECHONET オブジェクト
```

```
    int             EPCnum,     //EPC の組の数
    int             EPC[],      //EPC
    boolean         res,        //応答が必要なら true
    long            timeout,    //タイムアウト時間
    EN_SecureOpt    secopt      //セキュア通信オプション
) throws EN_Exception;
```

構文１２：

```
public void infProperty(
    EN_Object       sourceObject,   //発信元 ECHONET オブジェクト
    int             EPCnum,     //EPC の数
    int             EPC[],      //EPC の組
    EN_Property     p[],        //プロパティの組
    boolean         res,        //応答が必要なら true
    long            timeout,    //タイムアウト時間
    EN_SecureOpt    secopt      //セキュア通信オプション
) throws EN_Exception;
```

（4）説明

アプリケーションから通知電文を発行する。これは指定したプロパティの状態が変化した場合に必要となる状変時アナウンス、あるいは一定時間毎のプロパティの値の通知を含む。

アプリケーションは、自己 ECHONET オブジェクトにおいて、状変時アナウンスが必須とされているプロパティが変化した場合はこのメソッドを呼び出さなければならない。また、一定時間毎の通知が必要であれば、アプリケーションはこのメソッドを一定時間毎に呼び出さなければならない。

本メソッドは、１２の構文をサポートする。

構文１の場合、電文の SEA と SEOJ は *sourceObject* から、DEA が this から作成される。また EDT は *sourceObject*.callbackReadMyProperty メソッドを API が呼び出し、その戻り値を用いるので、callbackReadMyProperty を *sourceObject* に実装しておかなければならない。

構文２の場合、電文の SEA と SEOJ は *sourceObject* から、DEA が this から作成される。また EDT は *EPC, p* から作成される。

構文１は、プロパティ値を定期的に通報したい場合に用いることを想定したもので、アプリケーションがプロパティ値として引数をその都度用意する必要がない。一方、構文２は、アプリケーションの状態変化が発生した場合にプロパティ値を通知する場合に用いることを想定したもので、引数に通知するプロパティ値を設定する。

構文３，４は、セキュア通信を使用するためのものである。

構文５～８は、構文１～４それぞれに対し、規格書 Ver.2.10 から新規追加された応答要のプロパティ通知サービスを実行するための構文である。

構文９～１２は、それぞれ構文５～８を複合電文で行なうためのものである。

(a)*res* で false を指定した場合には、それぞれ構文１～４と同等である。

(b)this のアドレスが同報（オブジェクトインスタンスコード=0 を含む）である

場合、*res* で true を指定できない。

sourceObject	送信元オブジェクトの指定（すなわち、自己 ECHONET オブジェクト）。
EPCnum	読み出すプロパティの数。
EPC	EPC(第２部 4.2.7)の値。
p	通知するプロパティ値。
res	応答要のプロパティ通知サービスを実行するか否かの指定。する場合には true を指定。
timeout	タイムアウト時間。単位はミリ秒とし、0～20000 が指定できる。ただし、実際に計測される時間は、その処理系に依存する。同報または同報通知要求の場合には、0 を指定すること。0 以外のタイムアウト時間を指定しても 0 として処理する。
secopt	セキュア通信指定オプション。

（5）リターンコード

なし

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NORESOURCE	：送信バッファフルのため送信を受け付けない
EAPI_TIMEOUT	：タイムアウト
EAPI_NOTSEND	：原因不明のエラーにより未送信データ有り
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー

（7）注意

・callbackReadMyProperty が例外をスローした場合は何もしない。

・個別通知以外は電文に DEOJ を付加しない。

5.3.1.5 callbackReadMyProperty

（1）名称

`callbackReadMyProperty`　　－　プロパティ値取得サービスの実装

（2）機能

自己 ECHONET オブジェクトに対してプロパティ値取得処理を行う。

（3）構文

```
public EN_Property callbackReadMyProperty(
   EN_Packet          ev                  //発生イベントの詳細。
) throws EN_Exception;
```

（4）説明

本メソッドは、API からの呼び出しに備えてアプリケーションがその処理を記述しておかなければならないものである。アプリケーションは自己 ECHONET オブジェクトに対してプロパティ値読み出し要求を受けたときに行なうべき処理を記述したメソッドをこの名称で用意しオーバーライドしなければならない。本 ECHONET オブジェクトに対する Get, GetM, INF_REQ, INFM_REQ サービス要求を受け付けたときに呼び出される。

アプリケーションは*ev* に指定された EPC のプロパティ値を返すものとする。また、配列要素のプロパティの読み出しの場合には、*ev* の EPC および elementNo に対応する値を返すものとする。

処理できなかった場合、アプリケーションによってオーバーライドされていない場合は例外を発生する。

これらの例外が発生した場合、API はサービス要求元に処理不可応答電文を返す。

ev　　発生イベントの詳細。

（5）リターンコード

返すべきプロパティ。

（6）例外

EAPI_NOTACCEPT　　：処理対象プロパティが存在しない、配列要素プロパティ（配列でないと指定した時）、配列要素プロパティではない（配列と指定した時）、処理対象配列要素が存在しないなど。また、アプリケーションによってオーバーライドされていない場合

（7）注意

・オーバライドしたメソッドはできるだけ早く終了しなければならない。
・本メソッドで API がアプリケーションに渡す EN_Packet には EN_Object 型の sourceObject, destinationObject が含まれるが、アプリケーションはこれを送信元の EA, EOJ および送信先の EA, EOJ 情報を取り出すこと以外に使用してはならない。

5.3.1.6 callbackWriteMyProperty

（1）名称

`callbackWriteMyProperty`　　－　プロパティ値設定サービスの実装

（2）機能

自己 ECHONET オブジェクトに対してプロパティ値設定を行う。

（3）構文

```
public boolean callbackWriteMyProperty (
   EN_Packet          ev                  //発生イベントの詳細。
) throws EN_Exception;
```

（4）説明

本メソッドは、API からの呼び出しに備えてアプリケーションがその処理を記述しておかなければならないものである。アプリケーションは自己 ECHONET オブジェクトに対してプロパティ値設定要求を受けたときに行なうべき処理を記述したメソッドをこの名称で用意しオーバーライドしなければならない。本 ECHONET オブジェクトに対する SetI, SetC, SetMI, SetMC サービス要求を受け付けたときに呼び出される。

ev　　発生イベントの詳細。

（5）リターンコード

通常 true を返す。false を返した場合、応答電文の発行を抑制する。

（6）例外

EAPI_NOTACCEPT　　：処理対象プロパティが存在しない、配列要素プロパティ（配列でないと指定した時）、配列要素プロパティではない（配列と指定した時）、処理対象配列要素が存在しないなど。また、アプリケーションによってオーバーライドされていない場合

（7）注意

・オーバライドしたメソッドはできるだけ早く終了しなければならない。実際の機器を制御する場合、制御の終了を待たずにこのメソッドを先に終了させるのが望ましい。
・本メソッドでＡＰＩがアプリケーションに渡す EN_Packet には EN_Object 型の sourceObject, destinationObject が含まれるが、アプリケーションはこれを送信元の EA, EOJ および送信先の EA, EOJ 情報を取り出すこと以外に使用してはならない。

5.3.1.7 callbackNotifyEvent

（1）名称

callbackNotifyEvent　－　通知処理

（2）機能

通知があったときに呼ばれるコールバックメソッド。

（3）構文

```
public void callbackNotifyEvent(
   EN_Packet          ev                  //発生イベントの詳細。
) throws EN_Exception;
```

（4）説明

本メソッドは、API からの呼び出しに備えてアプリケーションがその処理を記述しておかなければならないものである。本メソッドは、API からの通知があった場合に呼ばれるメソッドである（本 ECHONET オブジェクトに対する INF, INFM サービスを受け付けた場合も含む）。アプリケーションは通知イベントを受けたときに行うべき処理を記述したメソッドをこの名称で用意し、オーバライドしなければならない。

EN_Node.addNotifyEventListener を用いて、アプリケーションが作成したインスタンスをインスタンス登録すると、そのインスタンスの callbackNotifyEvent メソッドが呼ばれる。

処理できなかった場合、すなわち、アプリケーションによってオーバーライドされていない場合、API は例外を発生する。

ev　　発生イベントの詳細。

（5）リターンコード

なし。

（6）例外

EAPI_NOTACCEPT　　：アプリケーションによってオーバーライドされていない場合

（7）注意

・オーバライドしたメソッドはできるだけ早く終了しなければならない。実際の機器を制御する場合、制御の終了を待たずにこのメソッドを終了させるのが望ましい。

・本メソッドで API がアプリケーションに渡す EN_Packet には EN_Object 型の sourceObject, destinationObject が含まれるが、アプリケーションはこれを送信元の EA, EOJ および送信先の EA, EOJ 情報を取り出すこと以外に使用してはならない。

5.3.1.8 callbackNotifyError

（1）名称

`callbackNotifyError` ― エラー通知処理

（2）機能

エラー通知があったときに呼ばれるコールバックメソッド。

（3）構文

```
public void callbackNotifyError(
   int                errorCode        //発生エラーの詳細。
) throws EN_Exception;
```

（4）説明

本メソッドは、API からの呼び出しに備えてアプリケーションがその処理を記述しておかなければならないものである。本メソッドは、API のエラー通知があった場合に呼ばれるメソッド。アプリケーションはエラー通知を受けたときに行うべき処理を記述したメソッドをこの名称で用意し、オーバライドしなければならない。

EN_Node.addNotifyErrorEventListener を用いて、アプリケーションが作成したインスタンスをインスタンス登録しておくと、そのインスタンスの callbackNotifyError メソッドが呼ばれる。

処理できなかった場合、すなわち、アプリケーションによってオーバーライドされていない場合、API は例外 EAPI_NOTACCEPT を発生する。

エラーの内容は、*errorCode* で通知される。

errorCode　発生エラーの詳細。以下のエラーが上がるが、本メソッドがコールされる条件は ECHONET 通信ミドルウェアの実装による。

EAPI_LOW_ERROR：下位通信ソフトウェアのエラー
EAPI_PRO_ERROR：プロトコル差異吸収処理部ソフトウェアのエラー
EAPI_MID_ERROR：ECHONET 通信処理部ソフトウェアのエラー

（5）リターンコード

なし。

（6）例外

EAPI_NOTACCEPT　：アプリケーションによってオーバーライドされていない場合

（7）注意

・オーバライドしたメソッドはできるだけ早く終了しなければならない。実際の機器を制御する場合、制御の終了を待たずにこのメソッドを先に終了させるのが望ましい。
・本メソッドでＡＰＩがアプリケーションに渡す EN_Packet には EN_Object 型の sourceObject, destinationObject が含まれるが、アプリケーションはこれを送信元の EA, EOJ および送信先の EA, EOJ 情報を取り出すこと以外に使用してはならない。

5.3.1.9 getEA

（1）名称

getEA　　－　ECHONET アドレスを返す

（2）機能

ECHONET アドレスを返す。

（3）構文

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

（4）説明

ECHONET アドレスを返す。自己 ECHONET オブジェクトを指しているときは、EN_Object.MYSELF_NODE を、それ以外のときは ECHONET アドレスを返す。

（5）リターンコード

オブジェクトの ECHONET アドレス。下位２バイトのみ使用。

（6）例外

EAPI_NOTOPEN　　：requestStart() 完了前にコールされた場合

（7）注意

なし。

5.3.1.10 getEOJ

（1）名称

getEOJ　　－　ECHONET オブジェクトコードを返す

（2）機能

ECHONET オブジェクトコードを返す。

（3）構文

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

（4）説明

ECHONET オブジェクトコードを返す。

（5）リターンコード

ECHONET オブジェクトコード。下位3バイトのみ使用。

（6）例外

なし。

（7）注意

なし。

5.3.1.11 getAddrKind

（1）名称

getAddrKind　　－　アドレス種別取得

（2）機能

EN_Object は同報指定したものか個別を指定したものかを示すコードを返す。

（3）構文

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

（4）説明

ECHONET オブジェクトコードを返す。

（5）リターンコード

同報指定の場合、APIVAL_BROAD_KIND を返す。個別指定の場合、APIVAL_EA_KIND を返す。

（6）例外

なし。

（7）注意

なし。

5.3.1.12 setAccessRule

（1）名称

setAccessRule　　－　アクセスルール設定

（2）機能

プロパティのアクセスルールを API に設定する。

（3）構文

構文１：

```
public void setAccessRule(
   int              EPC,          // EPC
   int              accessRule    // アクセスルール
) throws EN_Exception;
```

構文２：

```
public void setAccessRule(
   int              EPC,          // EPC
   int              accessRule    // アクセスルール
   int              keyKind       // アクセス制限レベル
) throws EN_Exception;
```

（4）説明

アプリケーションは、自己 ECHONET オブジェクトを EN_Node に登録する前に、自己 ECHONET オブジェクトの全てのプロパティ EPC に対し、本メソッドを用いてそのアクセスルールを設定しなければならない。

EPC で指定したプロパティのアクセスルールとして、accessRule で指定したアクセスルールを API に設定する。

accessRule には EN_Const インタフェースに定義されている APIVAL_RULE で始まる定数を用いて指定する。複数のアクセスルールを設定する場合には、これらの論理和をとったものを指定する。

例）EN_Object のインスタンス obj に対し、配列でない EPC = 0x83、Set, Get 可の場合(implements EN_Const してあるとして)

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

なお、*accessRule* に 0x00000000 を指定した場合には、対象 EPC に関するアクセスルールは API から削除される。

なお、API では、設定されたアクセスルールを、アクセス要求イベントのフィルタリングに使用する。API は、受取ったサービスに対応するアクセスルールを基にこれを処理可能か否かを判断し、処理可能である場合にはそれぞれのサービスに対応するコールバックメソッドを呼び出す。処理不可と判断された場合には処理不可応答電文を作成し要求元に返す。

構文２は、セキュア通信のためのメソッドである。相手種別毎に異なるアクセスルールを設定することができる。指定できる相手種別は、以下の４つである。

APIVAL_ACCESS_ANO　：　Anonymous レベル
APIVAL_ACCESS_USER　：　User レベル
APIVAL_ACCESS_SP　：　Service Provider レベル
APIVAL_ACCESS_MAKER　：　Maker レベル
なお、構文１で設定した場合には、Anonymous レベルとして設定される。

（5）リターンコード
なし。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_NORESOURCE	：登録不可
EAPI_ILLEGAL_PARAM	：指定 EPC, アクセスルール, アクセス制限レベル不正
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー

（7）注意
・既に設定済みのプロパティに対して重ねてアクセスルールを設定した場合には、後から設定したアクセスルールが有効となる（上書きされる）。
・*accessRule* に 0x00000000 を指定した EPC に対して他ノードからサービス要求を受信しても API は callback で始まるメソッドを呼び出さない。

5.3.1.13 getAccessRule

（1）名称

getAccessRule　　－　アクセスルール読み出し

（2）機能

API に設定されたプロパティのアクセスルールを読み出す。

（3）構文

構文１：

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

構文２：

```
public int getAccessRule(
   int              EPC,          // EPC
   int              AccessLevel   // アクセス制限レベル
) throws EN_Exception;
```

（4）説明

EPC で指定したプロパティのアクセスルールを API から読み出す。

構文２は、セキュア通信のために相手アクセス制限レベル毎に設定されたアクセスルールを読み出す際に用いる。指定できる相手種別は、以下の４つである。

APIVAL_ACCESS_ANO　：　Anonymous レベル

APIVAL_ACCESS_USER　：　User レベル

APIVAL_ACCESS_SP　：　Service Provider レベル

APIVAL_ACCESS_MAKER　：　Maker レベル

なお、セキュア通信機能をサポートしている API に対し、構文１で読み出した場合には、Anonymous レベルのアクセスルールが返される。

（5）リターンコード

アクセスルール。setAccessRule() で指定する形式と同一のものである。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正（指定された *EPC* が、規格範囲外(int なので、0x100 以上または 0x80 未満)の場合、または *AccessLevel が範囲外の場合*）
EAPI_NOTARGET	：対象 EPC 未登録

（7）注意

・指定された *EPC* が setAccessRule() で設定されていなければ対象の EN_Object では取扱わない *EPC* とみなすと考えて、例外 EAPI_NOTARGET を発生する。

・指定された EPC が setAccessRule() で設定されていれば、対象の EN_Object で取扱うと考えている EPC ということで「正常なリターンコード」が得られる。

5.3.1.14 isIn

（1）名称

`isIn`　　― アドレス包含関係チェック

（2）機能

アドレスの包含関係を調べる。

（3）構文

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

（4）説明

x に this が含まれていれば true を返す。これは(a1)(a2)のいずれかの条件を満たし、(b1)(b2)(b3)のいずれかの条件をすべて満たす場合に相当する。

(a1)x の EOJ のオブジェクトインスタンスコードが 0 で、インスタンスコード以外は x と this の EOJ が等しい場合。

(a2)x の EOJ と this の EOJ が等しい。

(b1) *x* のアドレスと this のアドレスが等しい。

(b2) *x* のアドレスが同報アドレス（ドメイン内同報、サブネット内同報のアドレス）であり、this のアドレスは同報アドレスではなく、*x* のアドレスが this を含む。

(b3) *x*、this のアドレスが同報アドレスであり、*x* に含まれるアドレスのすべてが this に含まれる。

（5）リターンコード

x に this が含まれていれば true。そうでなければ false。

（6）例外

なし。

（7）注意

なし。

5.3.1.15 setMProperty

（1）名称

setMProperty　　－　配列扱いプロパティ値設定サービスの実行

（2）機能

ECHONET オブジェクトに対して、配列で構成されたプロパティに要素を設定するサービスを実行する。

（3）構文

構文1：

```
public void setMProperty(
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_Property       p,               //指定要素に設定するプロパティ値
   boolean           res,             //応答が必要なら true
   long        timeout          //タイムアウト時間
) throws EN_Exception;
```

構文2：

```
public void setMProperty(
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_Property       p,               //指定要素に設定するプロパティ値
   boolean           res,             //応答が必要なら true
) throws EN_Exception;
```

構文3：

```
public void setMProperty(
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_Property       p,               //プロパティ
   long        timeout,         //タイムアウト時間
   EN_SecureOpt      secopt           //セキュア通信オプション
) throws EN_Exception;
```

構文4：

```
public void setMProperty(
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_Property       p,               //プロパティ
   EN_SecureOpt      secopt           //セキュア通信オプション
) throws EN_Exception;
```

（4）説明

本メソッドは、this が指す ECHONET オブジェクトの配列プロパティに対して以

下のように要素の設定を行なうときに使用するものである。
elementNo には、要素を設定するプロパティの配列要素位置を指定する。
構文１は送信元オブジェクトの指定ありのECHONET電文を作成するものであり、
構文２は送信元オブジェクトの指定なしのECHONET電文を作成するものである。
構文３，４は、セキュア通信を使用するためのものである。

(a)this のアドレスが他アプリケーションである場合、他アプリケーションに対して電文を発行する。引数 *res* が true の場合は応答を待つ。このとき、SEA と SEOJ は *sourceObject* から、DEA と DEOJ が this から作成される。ただし、this のアドレスが同報（オブジェクトインスタンスコード=0 を含む）であれば、*res* の値によらず応答を待たない。なお、他アプリケーションが同一ノード内にある場合、実際に電文をネットワークに流すか否かは実装依存である。

(b)自己 ECHONET オブジェクトに対する操作を行なう場合は、this.callbackWriteMyProperty が呼ばれる。callbackWriteMyProperty の引数は、(a)と同様に作成される。

(c)*res* の値が true の場合は応答要電文(ESV=0x65)、false の場合は応答不要電文(ESV=0x64)を発行する。*res* の値が true の場合は、応答を timeout 時間だけ待つ。*res* の値が false の場合は、応答を待たない。この場合、相手 ECHONET オブジェクトからは、正常処理された場合には応答は返って来ないが、処理不可の場合には処理不可応答電文（ESV=0x54）が返される。これをアプリケーションが受信するには、予め後述の EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

(d)*timeout* 値が 0 である場合、あるいは構文２の場合には、応答を待たない。この場合、応答は通知イベントで取得できる。ただし、アプリケーションは予め後述の EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

sourceObject	送信元オブジェクトの指定（すなわち、自己 ECHONET オブジェクト）。
EPC	EPC(第 2 部 4.2.7)の値。
elementNo	書き込む配列要素の要素番号。
p	書き込むプロパティ値。
res	応答が必要なら true
timeout	タイムアウト時間。単位はミリ秒とし、0～20000 が指定できる。ただし、実際に計測される時間は、その処理系に依存する。同報の場合には、0 を指定すること。0 以外のタイムアウト時間を指定しても 0 として処理する。
secopt	セキュア通信指定オプション。

（5）リターンコード
なし。

（6）例外

EAPI_NOTOPEN　：requestStart() 完了前にコールされた場合

EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NORESOURCE	：送信バッファフルのため送信を受け付けない
EAPI_TIMEOUT	：タイムアウト
EAPI_NOTSEND	：原因不明のエラーにより未送信データ有り
EAPI_NOTOPERATIVE	：処理不可応答電文受信
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー
EAPI_SEC_ERROR	：セキュア通信のエラー（認証エラー）

（7）注意

・*res* を true にし *timeout* 値を 0 に指定した場合には応答を待たず即リターンするが、この応答は通知イベントで取得できる。宛先ノードへの要求と非同期に応答を受信し処理するようなプログラムは、この仕組みを利用するとよい。

・同一オブジェクト・同一プロパティに対する要求を、同一オブジェクトから複数同時に処理した場合の結果は保証しない。この場合、応答メッセージが同一になり、API はそれを区別できない。

・タイムアウト発生後、相手オブジェクトから正常な応答電文が返ってきた場合には、アプリケーションには通知イベントで返す。ただし、アプリケーションがこれを受取るには、予め EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

・*timeout* (!= 0) 指定をして応答待ちをした EN_Object には、そのメソッドに応答として返すが、これがトリガとなって引き起こされるイベントは、その応答が返るべき全ての EN_Object（但し同一アプリケーション）に対して応答が返る。なお、同一オブジェクトについては、メソッドに応答を返した以外には、リスナーが登録されていたとしても配信は行なわない。

5.3.1.16 getMProperty

（1）名称

getMProperty　　－　配列扱いプロパティ値取得サービスの実行

（2）機能

ECHONET オブジェクトに対して、配列で構成されたプロパティの要素を取得するサービスを実行する。

（3）構文

構文1：

```
public EN_Property getMProperty(
   EN_Object          sourceObject,    //発信元 ECHONET オブジェクト
   int                EPC,             //EPC
   int                elementNo,       //配列要素番号
   boolean            req_broadcast,   //同報通知要求する場合には true
   long       timeout          //タイムアウト時間
) throws EN_Exception;
```

構文2：

```
public EN_Property getMProperty(
   int                EPC,             //EPC
   int                elementNo,       //配列要素番号
   boolean            req_broadcast,   //同報通知要求する場合には true
) throws EN_Exception;
```

構文3：

```
public void getMProperty(
   EN_Object          sourceObject,    //発信元 ECHONET オブジェクト
   int                EPC,             //EPC
   boolean            req_broadcast,   //プロパティ値通知要求サービスを
                                       //実行する場合には true
   long       timeout,         //タイムアウト時間
   EN_SecureOpt       secopt           //セキュア通信オプション
) throws EN_Exception;
```

構文4：

```
public void getMProperty(
   int                EPC,             //EPC
   boolean            req_broadcast,   //プロパティ値通知要求サービスを
                                       //実行する場合には true
   EN_SecureOpt       secopt           //セキュア通信オプション
) throws EN_Exception;
```

（4）説明

本メソッドは、this が指す相手の ECHONET オブジェクトの配列プロパティに対して以下のように要素の値の取得を行なうときに使用するものである。

elementNo には、要素を取得するプロパティの配列要素位置を指定する。

構文１は送信元オブジェクトの指定ありのECHONET電文を作成するものであり、構文２は送信元オブジェクトの指定なしのECHONET電文を作成するものである。構文３，４は、セキュア通信を使用するためのものである。

(a) this のアドレスが他アプリケーションである場合、他アプリケーションに対して電文を発行し応答を待つ。このとき、SEA と SEOJ は *sourceObject* から、DEA と DEOJ が this から作成される。ただし、this のアドレスが同報（オブジェクトインスタンスコード=0 を含む）であるか、または req_broadcast でプロパティ値通知要求サービス実行を指定した場合には、応答を待たない。この場合、応答は通知イベントで取得できる。ただし、アプリケーションは予め後述の EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。なお、他アプリケーションが同一ノード内にある場合、実際に電文をネットワークに流すか否かは実装依存である。

(b) this のアドレスが自ノードかつ自己 ECHONET オブジェクトのアドレスである場合は、this.callbackReadMyProperty が呼ばれる。callbackReadMyProperty の引数は、(a)と同様に作成される。

(c) this のアドレスが自ノードでありかつ自己 ECHONET オブジェクトのアドレスでない場合は、自ノードの他の適切な EN_Object が呼ばれる。

(d) timeout 値が 0 である場合、あるいは構文２の場合には、応答を待たない。この場合、応答は通知イベントで取得できる。ただし、アプリケーションは予め後述の EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

(e) プロパティ値通知要求サービスは、本メソッドにより実行可能。

sourceObject	送信元オブジェクトの指定（すなわち、自己 ECHONET オブジェクト）。
EPC	読み出す EPC(第 2 部 4.2.7)の値。
elementNo	読み出す配列要素の要素番号。
req_broadcast	プロパティ値通知要求サービスを実行するかしないかの指定。する場合には true を指定。
timeout	タイムアウト時間。単位はミリ秒とし、0～20000 が指定できる。ただし、実際に計測される時間は、その処理系に依存する。同報の場合には、0 を指定すること。0 以外のタイムアウト時間を指定しても 0 として処理する。
secopt	セキュア通信指定オプション。

（5）リターンコード

配列構造のプロパティから取得したプロパティ値。ただし、同報アドレス（ドメイン内同報、サブネット内同報のアドレス）に対する操作の場合、*timeout* に 0 を指定した場合、および構文２の場合は null とする。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NORESOURCE	：送信バッファフルのため送信を受け付けない

EAPI_TIMEOUT	：タイムアウト
EAPI_NOTSEND	：原因不明のエラーにより未送信データ有り
EAPI_NOTOPERATIVE	：処理不可応答電文受信
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー
EAPI_SEC_ERROR	：セキュア通信のエラー（認証エラー）

（7）注意

・同一オブジェクト・同一プロパティに対する要求を、同一オブジェクトから複数同時に処理した場合の結果は保証しない。この場合、応答メッセージが同一になり、API はそれを区別できない。

・タイムアウト発生後、相手オブジェクトから正常な応答電文が返ってきた場合には、アプリケーションには通知イベントで返す。ただし、アプリケーションがこれを受取るには、予め EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

・*timeout* 値を 0 に指定した場合には応答を待たず即リターンするが、この応答は通知イベントで取得できる。宛先ノードへの要求と非同期に応答を受信し処理するようなプログラムは、この仕組みを利用するとよい。

・*timeout* (!= 0) 指定をして応答待ちをした EN_Object には、そのメソッドに応答として返すが、これがトリガとなって引き起こされるイベントは、その応答が返るべき全ての EN_Object（但し同一アプリケーション）に対して応答が返る。なお、同一オブジェクトについては、メソッドに応答を返した以外には、リスナーが登録されていたとしても配信は行なわない。

5.3.1.17 addMProperty

（1）名称

addMProperty　　－　配列扱いプロパティ追加要求

（2）機能

ECHONET オブジェクトに対して、配列で構成されたプロパティの要素を追加するサービスを実行する。

（3）構文

構文1：

```
public void addMProperty (
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_Property       p,               //指定要素に追加するプロパティ
   boolean           res,             //応答が必要なら true
   long        timeout          //タイムアウト時間
) throws EN_Exception;
```

構文2：

```
public void addMProperty (
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_Property       p,               //指定要素に追加するプロパティ
   boolean           res,             //応答が必要なら true
) throws EN_Exception;
```

構文1：

```
public void addMProperty (
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_Property       p,               //指定要素に追加するプロパティ
   boolean           res,             //応答が必要なら true
   long        timeout          //タイムアウト時間
   EN_SecureOpt      secopt           //セキュア通信オプション
) throws EN_Exception;
```

構文2：

```
public void addMProperty (
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_Property       p,               //指定要素に追加するプロパティ
   boolean           res,             //応答が必要なら true
   EN_SecureOpt      secopt           //セキュア通信オプション
) throws EN_Exception;
```

（4）説明

本メソッドは、this が指す相手の ECHONET オブジェクトの配列プロパティに対して要素の追加を行なうときに使用するものである。

p は、指定要素に追加するプロパティを指定する。

elementNo には、*p* を追加するプロパティの配列要素位置を指定する。

構文１は送信元オブジェクトの指定ありの ECHONET 電文を作成するものであり、構文２は送信元オブジェクトの指定なしの ECHONET 電文を作成するものである。

構文３，４は、セキュア通信を使用するためのものである。

(a) this のアドレスが他アプリケーションである場合、他アプリケーションに対して電文を発行する。引数 *res* が true の場合は応答を待つ。このとき、SEA と SEOJ は *sourceObject* から、DEA と DEOJ が this から作成される。ただし、this のアドレスが同報（オブジェクトインスタンスコード=0 を含む）であれば、応答を待たない。なお、他アプリケーションが同一ノード内にある場合、実際に電文をネットワークに流すか否かは実装依存である。

(b) 自己 ECHONET オブジェクトに対する操作を行なう場合は、this.callbackAddMyPropertyMember が呼ばれる。なお、this.callbackAddMyPropertyMember の引数は、(a)と同様に作成される。

(c) *res* の値が true の場合は応答要電文(ESV=0x69)、false の場合は応答不要電文(ESV=0x68)を発行する。*res* の値が true の場合は、応答を timeout 時間だけ待つ。*res* の値が false の場合は、応答を待たない。この場合、相手 ECHONET オブジェクトからは、正常処理された場合には応答は返って来ないが、処理不可の場合には処理不可応答電文（ESV=0x58）が返される。これをアプリケーションが受信するには、予め後述の EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

(d) *timeout* 値が 0 である場合、あるいは構文２の場合には、応答を待たない。この場合、応答は通知イベントで取得できる。ただし、アプリケーションは予め後述の EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

sourceObject	送信元オブジェクトの指定（すなわち、自己 ECHONET オブジェクト）。
EPC	追加する対象の EPC(第 2 部 4.2.7)の値。
elementNo	追加する配列要素の要素番号。
p	追加するプロパティ値。
res	応答が必要なら true。
timeout	タイムアウト時間。単位はミリ秒とし、0～20000 が指定できる。ただし、実際に計測される時間は、その処理系に依存する。同報の場合には、0 を指定すること。0 以外のタイムアウト時間を指定しても 0 として処理する。
secopt	セキュア通信指定オプション。

（5）リターンコード

なし。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NORESOURCE	：送信バッファフルのため送信を受け付けない
EAPI_TIMEOUT	：タイムアウト
EAPI_NOTSEND	：原因不明のエラーにより未送信データ有り
EAPI_NOTOPERATIVE	：処理不可応答電文受信
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー
EAPI_SEC_ERROR	：セキュア通信のエラー（認証エラー）

（7）注意

・*res* を true にし *timeout* 値を 0 に指定した場合には応答を待たず即リターンするが、この応答は通知イベントで取得できる。宛先ノードへの要求と非同期に応答を受信し処理するようなプログラムは、この仕組みを利用するとよい。

・同一オブジェクト・同一プロパティに対する要求を、同一オブジェクトから複数同時に処理した場合の結果は保証しない。この場合、応答メッセージが同一になり、API はそれを区別できない。

・タイムアウト発生後、相手オブジェクトから正常な応答電文が返ってきた場合には、アプリケーションには通知イベントで返す。ただし、アプリケーションがこれを受取るには、予め EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

・*timeout* (!= 0) 指定をして応答待ちをした EN_Object には、そのメソッドに応答として返すが、これがトリガとなって引き起こされるイベントは、その応答が返るべき全ての EN_Object（但し同一アプリケーション）に対して応答が返る。なお、同一オブジェクトについては、メソッドに応答を返した以外には、リスナーが登録されていたとしても配信は行なわない。

5.3.1.18 delMProperty

（1）名称

delMProperty　　－　配列扱いプロパティ削除要求

（2）機能

ECHONET オブジェクトに対して、配列で構成されたプロパティにおいて、要素を削除するサービスを行う。

（3）構文

構文1：

```
public void delMProperty (
   EN_Object          sourceObject,    //発信元 ECHONET オブジェクト
   int                EPC,             //EPC
   int                elementNo,       //配列要素番号
   boolean            res,             //応答が必要なら true
   long       timeout          //タイムアウト時間
) throws EN_Exception;
```

構文2：

```
public void delMProperty (
   int                EPC,             //EPC
   int                elementNo,       //配列要素番号
   boolean            res,             //応答が必要なら true
) throws EN_Exception;
```

構文3：

```
public void delMProperty (
   EN_Object          sourceObject,    //発信元 ECHONET オブジェクト
   int                EPC,             //EPC
   int                elementNo,       //配列要素番号
   boolean            res,             //応答が必要なら true
   long       timeout          //タイムアウト時間
   EN_SecureOpt       secopt           //セキュア通信オプション
) throws EN_Exception;
```

構文4：

```
public void delMProperty (
   int                EPC,             //EPC
   int                elementNo,       //配列要素番号
   boolean            res,             //応答が必要なら true
   EN_SecureOpt       secopt           //セキュア通信オプション
) throws EN_Exception;
```

（4）説明

本メソッドは、相手の ECHONET オブジェクトの配列プロパティに対して要素の削除を行なうときに使用するものである。

elementNo には、要素を削除するプロパティの配列要素位置を指定する。

構文１は送信元オブジェクトの指定ありのECHONET電文を作成するものであり、
構文２は送信元オブジェクトの指定なしのECHONET電文を作成するものである。
構文３，４は、セキュア通信を使用するためのものである。

(a)this のアドレスが他アプリケーションである場合、他アプリケーションに対して電文を発行する。引数 *res* が true の場合は応答を待つ。このとき、SEA と SEOJ は *sourceObject* から、DEA と DEOJ が this から作成される。ただし、this のアドレスが同報（オブジェクトインスタンスコード=0 を含む）であれば、応答を待たない。なお、他アプリケーションが同一ノード内にある場合、実際に電文をネットワークに流すか否かは実装依存である。

(b)自己 ECHONET オブジェクトに対する操作を行なう場合は、this.callbackDellMyPropertyMember が呼ばれる。なお、callbackDellMyPropertyMember の引数は、(a)と同様に作成される。

(c)*timeout* 値が 0 である場合、あるいは構文２の場合には、応答を待たない。この場合、応答は通知イベントで取得できる。ただし、アプリケーションは予め後述の EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

sourceObject	送信元オブジェクトの指定（すなわち、自己 ECHONET オブジェクト）。
EPC	EPC(第 2 部 4.2.7)の値。
elementNo	削除する配列要素の要素番号。
res	応答が必要なら true。
timeout	タイムアウト時間。単位はミリ秒とし、0～20000 が指定できる。ただし、実際に計測される時間は、その処理系に依存する。同報の場合には、0 を指定すること。0 以外のタイムアウト時間を指定しても 0 として処理する。
secopt	セキュア通信指定オプション。

（5）リターンコード

なし。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NORESOURCE	：送信バッファフルのため送信を受け付けない
EAPI_TIMEOUT	：タイムアウト
EAPI_NOTSEND	：原因不明のエラーにより未送信データ有り
EAPI_NOTOPERATIVE	：処理不可応答電文受信
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー
EAPI_SEC_ERROR	：セキュア通信のエラー（認証エラー）

（7）注意

・*res* を true にし *timeout* 値を 0 に指定した場合には応答を待たず即リターンするが、この応答は通知イベントで取得できる。宛先ノードへの要求と非同期に応答を受信し処理するようなプログラムは、この仕組みを利用するとよい。

・同一オブジェクト・同一プロパティに対する要求を、同一オブジェクトから複数同時に処理した場合の結果は保証しない。この場合、応答メッセージが同一になり、API はそれを区別できない。
・タイムアウト発生後、相手オブジェクトから正常な応答電文が返ってきた場合には、アプリケーションには通知イベントで返す。ただし、アプリケーションがこれを受取るには、予め EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。
・*timeout* 値を 0 に指定した場合には応答を待たず即リターンするが、この応答は通知イベントで取得できる。宛先ノードへの要求と非同期に応答を受信し処理するようなプログラムは、この仕組みを利用するとよい。
・*timeout* (!= 0) 指定をして応答待ちをした EN_Object には、そのメソッドに応答として返すが、これがトリガとなって引き起こされるイベントは、その応答が返るべき全ての EN_Object（但し同一アプリケーション）に対して応答が返る。なお、同一オブジェクトについては、メソッドに応答を返した以外には、リスナーが登録されていたとしても配信は行なわない。

5.3.1.19 checkMProperty

（1）名称

checkMProperty　　－　配列扱いプロパティ存在確認要求

（2）機能

ECHONET オブジェクトに対して、配列で構成されたプロパティの指定した要素番号の要素の存在を確認するサービスを行う。

（3）構文

構文1：

```
public boolean checkMProperty (
   EN_Object          sourceObject,     //発信元 ECHONET オブジェクト
   int                EPC,              //EPC
   int                elementNo,        //配列要素番号
   long         timeout           //タイムアウト時間
) throws EN_Exception;
```

構文2：

```
public boolean checkMProperty (
   int                EPC,              //EPC
   int                elementNo,        //配列要素番号
) throws EN_Exception;
```

構文3：

```
public boolean checkMProperty (
   EN_Object          sourceObject,     //発信元 ECHONET オブジェクト
   int                EPC,              //EPC
   int                elementNo,        //配列要素番号
   long         timeout           //タイムアウト時間
   EN_SecureOpt       secopt            //セキュア通信オプション
) throws EN_Exception;
```

構文4：

```
public boolean checkMProperty (
   int                EPC,              //EPC
   int                elementNo,        //配列要素番号
   EN_SecureOpt       secopt            //セキュア通信オプション
) throws EN_Exception;
```

（4）説明

本メソッドは、相手の ECHONET オブジェクトの配列プロパティに対して要素の存在確認を行なうときに使用するものである。

elementNo には、要素の存在を確認するプロパティの配列要素位置を指定する。

構文1は送信元オブジェクトの指定ありの ECHONET 電文を作成するものであり、構文2は送信元オブジェクトの指定なしの ECHONET 電文を作成するものである。

構文3，4は、セキュア通信を使用するためのものである。

(a)this のアドレスが他アプリケーションである場合、他アプリケーションに対

して電文を発行する。このとき、SEA と SEOJ は *sourceObject* から、DEA と DEOJ が this から作成される。ただし、this のアドレスが同報（オブジェクトインスタンスコード=0 を含む）であれば、応答を待たない。なお、他アプリケーションが同一ノード内にある場合、実際に電文をネットワークに流すか否かは実装依存である。

(b)自己 ECHONET オブジェクトに対する操作を行なう場合には、this.callbackCheckMyPropertyMember が呼ばれる。なお、callbackCheckMyPropertyMember の引数は、(a)と同様に作成される。

(c) *timeout* 値が 0 である場合、あるいは構文２の場合には、応答を待たない。この場合、応答は通知イベントで取得できる。ただし、アプリケーションは予め後述の EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

sourceObject	送信元オブジェクトの指定（すなわち、自己 ECHONET オブジェクト）。
EPC	EPC(第 2 部 4.2.7)の値。
elementNo	存在確認する配列要素の要素番号。
timeout	タイムアウト時間。単位はミリ秒とし、0～20000 が指定できる。ただし、実際に計測される時間は、その処理系に依存する。同報の場合には、0 を指定すること。0 以外のタイムアウト時間を指定しても 0 として処理する。
secopt	セキュア通信指定オプション。

（5）リターンコード

チェック対象のプロパティの有無が返される。存在する場合 true、存在しない場合 false が返される。なお、*timeout* =0 を指定した場合および構文２の場合には false を返すものとする。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NORESOURCE	：送信バッファフルのため送信を受け付けない
EAPI_TIMEOUT	：タイムアウト
EAPI_NOTSEND	：原因不明のエラーにより未送信データ有り
EAPI_NOTOPERATIVE	：処理不可応答電文受信
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー
EAPI_SEC_ERROR	：セキュア通信のエラー（認証エラー）

（7）注意

・同一オブジェクト・同一プロパティに対する要求を、同一オブジェクトから複数同時に処理した場合の結果は保証しない。この場合、応答メッセージが同一になり、API はそれを区別できない。

・this のアドレスを明示的に１つに指定している場合（同報でない場合）、要素の存在有無がリターンコードで返る。

・タイムアウト発生後、相手オブジェクトから正常な応答電文が返ってきた場合には、

アプリケーションには通知イベントで返す。ただし、アプリケーションがこれを受取るには、予め EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

・*timeout* 値を 0 に指定した場合には応答を待たず即リターンするが、この応答は通知イベントで取得できる。宛先ノードへの要求と非同期に応答を受信し処理するようなプログラムは、この仕組みを利用するとよい。

・*timeout* (!= 0) 指定をして応答待ちをした EN_Object には、そのメソッドに応答として返すが、これがトリガとなって引き起こされるイベントは、その応答が返るべき全ての EN_Object（但し同一アプリケーション）に対して応答が返る。なお、同一オブジェクトについては、メソッドに応答を返した以外には、リスナーが登録されていたとしても配信は行なわない。

5.3.1.20 addMSProperty

（1）名称

addMSProperty ― 要素指定なし配列扱いプロパティ追加要求

（2）機能

ECHONET オブジェクトに対して、配列で構成されたプロパティの要素を追加するサービスを行う。どの要素番号に追加されるかは、相手のオブジェクトでの処理に依存する。

（3）構文

構文１：

```
public int addMSProperty (
   EN_Object          sourceObject,     //発信元 ECHONET オブジェクト
   int                EPC,              //EPC
   EN_Property        p,                //プロパティ値
   boolean            res,              //応答が必要なら true
   long         timeout           //タイムアウト時間
) throws EN_Exception;
```

構文２：

```
public int addMSProperty (
   int                EPC,              //EPC
   EN_Property        p,                //プロパティ値
   boolean            res,              //応答が必要なら true
) throws EN_Exception;
```

構文３：

```
public int addMSProperty (
   EN_Object          sourceObject,     //発信元 ECHONET オブジェクト
   int                EPC,              //EPC
   EN_Property        p,                //プロパティ値
   boolean            res,              //応答が必要なら true
   long         timeout           //タイムアウト時間
   EN_SecureOpt       secopt            //セキュア通信オプション
) throws EN_Exception;
```

構文４：

```
public int addMSProperty (
   int                EPC,              //EPC
   EN_Property        p,                //プロパティ値
   boolean            res,              //応答が必要なら true
   EN_SecureOpt       secopt            //セキュア通信オプション
) throws EN_Exception;
```

（4）説明

本メソッドは、相手の ECHONET オブジェクトの配列プロパティに対して任意の位置への要素の追加を行なうときに使用するものである。

p には、追加するプロパティの要素を指定する。
構文１は送信元オブジェクトの指定ありのECHONET電文を作成するものであり、構文２は送信元オブジェクトの指定なしのECHONET電文を作成するものである。
構文３，４は、セキュア通信を使用するためのものである。

(a)this のアドレスが他アプリケーションである場合、他アプリケーションに対して電文を発行する。引数 *res* が true の場合は応答を待つ。このとき、SEA と SEOJ は *sourceObject* から、DEA と DEOJ が this から作成される。ただし、this のアドレスが同報（オブジェクトインスタンスコード=0 を含む）であれば、応答を待たない。なお、他アプリケーションが同一ノード内にある場合、実際に電文をネットワークに流すか否かは実装依存である。

(b)自己 ECHONET オブジェクトに対する操作を行なう場合は、this.callbackAddMyPropertyMemberAlt が呼ばれる。なお、callbackAddMyPropertyMemberAlt の引数は、(a)と同様に作成される

(c)*timeout* 値が 0 である場合、もしくは構文２を指定した場合には、応答を待たない。この場合、応答は通知イベントで取得できる。

sourceObject	送信元オブジェクトの指定（すなわち、自己 ECHONET オブジェクト）。
EPC	EPC(第 2 部 4.2.7)の値。
p	追加するプロパティ値。
res	応答が必要なら true
timeout	タイムアウト時間。単位はミリ秒とし、0～20000 が指定できる。ただし、実際に計測される時間は、その処理系に依存する。同報の場合には、0 を指定すること。0 以外のタイムアウト時間を指定しても 0 として処理する。
secopt	セキュア通信指定オプション。

（5）リターンコード

追加された要素の要素番号。構文１で *timeout* = 0 を指定した場合および構文２の場合、戻り値は -1 とする。また、*res* に false を指定した場合も戻り値は -1 とする。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NORESOURCE	：送信バッファフルのため送信を受け付けない
EAPI_TIMEOUT	：タイムアウト
EAPI_NOTSEND	：原因不明のエラーにより未送信データ有り
EAPI_NOTOPERATIVE	：処理不可応答電文受信
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー
EAPI_SEC_ERROR	：セキュア通信のエラー（認証エラー）

（7）注意

・*res* を true にし *timeout* 値を 0 に指定した場合には応答を待たず即リターンするが、この応答は通知イベントで取得できる。宛先ノードへの要求と非同期に応答を受信し

処理するようなプログラムは、この仕組みを利用するとよい。

・同一オブジェクト・同一プロパティに対する要求を、同一オブジェクトから複数同時に処理した場合の結果は保証しない。この場合、応答メッセージが同一になり、API はそれを区別できない。

・this のアドレスを明示的に１つに指定している場合（同報でない場合）、上述したコードがリターンコードで返る。this が同報アドレス（ドメイン内同報、サブネット内同報のアドレス）の場合、リターンコードは－１である。

・タイムアウト発生後、相手オブジェクトから正常な応答電文が返ってきた場合には、アプリケーションには通知イベントで返す。ただし、アプリケーションがこれを受取るには、予め EN_Node.addNotifyEventListener によって呼び出しリスナーを登録しておかなければならない。

・*timeout* (!= 0) 指定をして応答待ちをした EN_Object には、そのメソッドに応答として返すが、これがトリガとなって引き起こされるイベントは、その応答が返るべき全ての EN_Object（但し同一アプリケーション）に対して応答が返る。なお、同一オブジェクトについては、メソッドに応答を返した以外には、リスナーが登録されていたとしても配信は行なわない。

5.3.1.21 infPropertyMember

（1）名称

infPropertyMember　　－　配列扱いプロパティ通知要求

（2）機能

アプリケーションから配列扱いプロパティ要素値の通知電文を発行する。

（3）構文

構文1：

```
public void infPropertyMember (
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
) throws EN_Exception;
```

構文2：

```
public void infPropertyMember (
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_Property       p,               //プロパティ値
) throws EN_Exception;
```

構文3：

```
public void infPropertyMember (
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_SecureOpt      secopt           //セキュア通信オプション
) throws EN_Exception;
```

構文4：

```
public void infPropertyMember (
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_Property       p,               //プロパティ値
   EN_SecureOpt      secopt           //セキュア通信オプション
) throws EN_Exception;
```

構文５：
```
public void infPropertyMember (
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   boolean           res,             //応答が必要なら true
   long              timeout          //タイムアウト時間
) throws EN_Exception;
```
構文６：
```
public void infPropertyMember (
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_Property       p,               //プロパティ
   boolean           res,             //応答が必要なら true
   long              timeout          //タイムアウト時間
) throws EN_Exception;
```
構文７：
```
public void infPropertyMember (
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   boolean           res,             //応答が必要なら true
   long              timeout,         //タイムアウト時間
   EN_SecureOpt      secopt           //セキュア通信オプション
) throws EN_Exception;
```
構文８：
```
public void infPropertyMember (
   EN_Object         sourceObject,    //発信元 ECHONET オブジェクト
   int               EPC,             //EPC
   int               elementNo,       //配列要素番号
   EN_Property       p,               //プロパティ
   boolean           res,             //応答が必要なら true
   long              timeout,         //タイムアウト時間
   EN_SecureOpt      secopt           //セキュア通信オプション
) throws EN_Exception;
```

（4）説明

アプリケーションから通知電文を発行する。配列要素番号で指定したプロパティの要素の状態が変化した場合に必要となる状変時アナウンス、あるいは一定時間ごと

のプロパティ値の通知を含む。
本メソッドは、４つの構文をサポートする。
構文１の場合、電文の SEA と SEOJ は *sourceObject* から、DEA が this から作成される。また EDT は *sourceObject*.callbackReadMyProperty メソッドを API が呼び出し、その戻り値を用いるので、callbackReadMyProperty を *sourceObject* に実装しておかなければならない。
構文２の場合、電文の SEA と SEOJ は *sourceObject* から、DEA が this から作成される。また EDT は *EPC, elementNo, p から作成される。*
構文１は、プロパティ値を定期的に通報したいと場合に用いることを想定したもので、アプリケーションがプロパティ値として引数をその都度用意する必要がない。一方、構文２は、アプリケーションの状態変化が発生した場合にプロパティ値を通知する場合に用いることを想定したもので、引数に通知するプロパティ値を設定する。構文３，４は、セキュア通信を使用するためのものである。

sourceObject	送信元オブジェクトの指定（すなわち、自己 ECHONET オブジェクト）。
EPC	EPC(第 2 部 4.2.7)の値。
elementNo	通知するプロパティ配列要素の要素番号。
p	通知するプロパティ値。
timeout	タイムアウト時間。単位はミリ秒とし、0～20000 が指定できる。ただし、実際に計測される時間は、その処理系に依存する。同報または同報通知要求の場合には、0 を指定すること。0 以外のタイムアウト時間を指定しても 0 として処理する。
secopt	セキュア通信指定オプション。

（5）リターンコード
なし。
（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NORESOURCE	：送信バッファフルのため送信を受け付けない
EAPI_TIMEOUT	：タイムアウト
EAPI_NOTSEND	：原因不明のエラーにより未送信データ有り
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー

（7）注意
・構文１で callbackReadMyProperty が例外をスローした場合は何もしない。
・個別通知以外は DEOJ を電文に付加しない。

5.3.1.22 callbackAddMyPropertyMember

（1）名称

`callbackAddMyPropertyMember`　　－　配列扱いプロパティ値追加（要素指定あり）サービスの実装

（2）機能

自己 ECHONET オブジェクトに対して配列扱いプロパティの要素の追加（要素指定あり）を行う。

（3）構文

```
public boolean callbackAddMyPropertyMember (
   EN_Packet          ev                //発生イベントの詳細
) throws EN_Exception;
```

（4）説明

本メソッドは、API からの呼び出しに備えてアプリケーションがその処理を記述しておかなければならないものである。アプリケーションは自己 ECHONET オブジェクトに対して配列扱いプロパティの要素の追加（要素指定あり）要求を受けたときに行なうべき処理を記述したメソッドをこの名称で用意しオーバーライドしなければならない。、本 ECHONET オブジェクトに対する AddMI, AddMC サービス要求を受け付けたときに呼び出される。

処理できなかった場合、アプリケーションによってオーバーライドされていない場合は、API は例外 EAPI_NOTACCEPT を発生する。

この例外が発生した場合、API はサービス要求元に処理不可応答電文を返す。

（5）リターンコード

通常 true を返す。false を返した場合、応答電文の発行を抑制する。

（6）例外

EAPI_NOTACCEPT　　：処理対象プロパティが存在しない、配列要素プロパティではない、処理対象配列要素が存在しない場合。また、アプリケーションによってオーバーライドされていない場合。

（7）注意

・オーバライドしたメソッドはできるだけ早く終了しなければならない。本メソッドで API がアプリケーションに渡す EN_Packet には EN_Object 型の sourceObject, destinationObject が含まれるが、アプリケーションはこれを送信元の EA, EOJ および送信先の EA, EOJ 情報を取り出すこと以外に使用してはならない。

5.3.1.23 callbackDelMyPropertyMember

（1）名称

`callbackDelMyPropertyMember`　　－　配列扱いプロパティ値削除サービスの実装

（2）機能

自己 ECHONET オブジェクトに対して配列扱いプロパティ値の削除を行う。

（3）構文

```
public boolean callbackDelMyPropertyMember (
   EN_Packet        ev              //発生イベントの詳細
) throws EN_Exception;
```

（4）説明

本メソッドは、API からの呼び出しに備えてアプリケーションがその処理を記述しておかなければならないものである。アプリケーションは自己 ECHONET オブジェクトに対して配列扱いプロパティ値の削除要求を受けたときに行なうべき処理を記述したメソッドをこの名称で用意しオーバーライドしなければならない。、本 ECHONET オブジェクトに対する DelMI, DelMC サービス要求を受け付けたときに呼び出される。

処理できなかった場合、アプリケーションによってオーバーライドされていない場合は、API は例外 EAPI_NOTACCEPT を発生する。

この例外が発生した場合、API はサービス要求元に処理不可応答電文を返す。

（5）リターンコード

通常 true を返す。false を返した場合、応答電文の発行を抑制する。

（6）例外

EAPI_NOTACCEPT	：処理対象プロパティが存在しない、配列要素プロパティではない、処理対象配列要素が存在しない場合。また、アプリケーションによってオーバーライドされていない場合。

（7）注意

オーバライドしたメソッドはできるだけ早く終了しなければならない。本メソッドで API がアプリケーションに渡す EN_Packet には EN_Object 型の sourceObject, destinationObject が含まれるが、アプリケーションはこれを送信元の EA, EOJ および送信先の EA, EOJ 情報を取り出すこと以外に使用してはならない。

5.3.1.24 callbackCheckMyPropertyMember

（1）名称

`callbackCheckMyPropertyMember`　　― 配列扱いプロパティ値存在確認サービスの実装

（2）機能

自己ECHONETオブジェクトに対して配列扱いプロパティの要素の存在確認を行う。

（3）構文

```
public boolean callbackCheckMyPropertyMember (
   EN_Packet          ev              //発生イベントの詳細
) throws EN_Exception;
```

（4）説明

本メソッドは、API からの呼び出しに備えてアプリケーションがその処理を記述しておかなければならないものである。アプリケーションは自己 ECHONET オブジェクトに対して配列扱いプロパティの要素の存在確認要求を受けたときに行なうべき処理を記述したメソッドをこの名称で用意しオーバーライドしなければならない。、本 ECHONET オブジェクトに対する CheckM サービス要求を受け付けたときに呼び出される。

処理できなかった場合、アプリケーションによってオーバーライドされていない場合は、API は例外 EAPI_NOTACCEPT を発生する。

この例外が発生した場合、API はサービス要求元に処理不可応答電文を返す。

（5）リターンコード

指定された要素が存在する `true`、存在しない `false`。

（6）例外

EAPI_NOTACCEPT　　：処理対象プロパティが存在しない、配列要素プロパティではない、処理対象配列要素が存在しない場合。また、アプリケーションによってオーバーライドされていない場合。

（7）注意

オーバライドしたメソッドはできるだけ早く終了しなければならない。本メソッドで API がアプリケーションに渡す EN_Packet には EN_Object 型の sourceObject, destinationObject が含まれるが、アプリケーションはこれを送信元の EA, EOJ および送信先の EA, EOJ 情報を取り出すこと以外に使用してはならない。

5.3.1.25 callbackAddMyPropertyMemberAlt

（1）名称

`callbackAddMyPropertyMemberAlt`　－　配列扱いプロパティ値の任意要素番号への追加サービスの実装

（2）機能

自己 ECHONET オブジェクトに対して配列扱いプロパティ値の追加を行う。ただし、どの要素番号の位置に挿入するかはこれを処理する宛先 ECHONET ノードの ECHONET オブジェクト側での実装に任されている。

（3）構文

```
public boolean callbackAddMyPropertyMemberAlt (
   EN_Packet          ev                  //発生イベントの詳細
) throws EN_Exception;
```

（4）説明

本メソッドは、API からの呼び出しに備えてアプリケーションがその処理を記述しておかなければならないものである。アプリケーションは自己 ECHONET オブジェクトに対して配列扱いプロパティ値の追加要求を受けたときに行なうべき処理を記述したメソッドをこの名称で用意しオーバーライドしなければならない。、本 ECHONET オブジェクトに対する AddMSI, AddMSC サービス要求を受け付けたときに呼び出される。

ev の `elementNo` に、追加された要素の要素番号を返す。

処理できなかった場合、アプリケーションによってオーバーライドされていない場合は、API は例外 EAPI_NOTACCEPT を発生する。

この例外が発生した場合、API はサービス要求元に処理不可応答電文を返す。

（5）リターンコード

通常 true を返す。false を返した場合、応答電文の発行を抑制する。なお、追加された要素の要素番号を引数 ev の elementNo に返す。

（6）例外

EAPI_NOTACCEPT　：処理対象プロパティが存在しない、配列要素プロパティではない、処理対象配列要素が存在しない場合。また、アプリケーションによってオーバーライドされていない場合。

（7）注意

・オーバライドしたメソッドはできるだけ早く終了しなければならない。
・本メソッドで API がアプリケーションに渡す EN_Packet には EN_Object 型の sourceObject, destinationObject が含まれるが、アプリケーションはこれを送信元の EA, EOJ および送信先の EA, EOJ 情報を取り出すこと以外に使用してはならない。

５．３．２ EN_Node クラス

（1）名称

EN_Node　　－　ECHONET ノードとイベントの管理

（2）機能

本クラスは、自ノードに到着するイベントの管理を行う。アプリケーションはこのクラスのインスタンスをただ１つ作成しなければならない。

ノードに到着したイベントは、アプリケーションが用意したクラスに結び付けられる。このメカニズムにより、アプリケーションはプロパティに対する操作だけを記述すればよい。

（3）構文

```
public class EN_Node extends Object
        implements EN_Const;
```

５．３．２．１ EN_Node

（1）名称

EN_Node －　ECHONET ノードコンストラクタ

（2）機能

ECHONET ノードとイベントの管理の初期化と起動。

（3）構文

```
public EN_Node();
```

（4）説明

ECHONET 通信ミドルウェアに接続し、その管理を行う。
イベント待ちを行うイベントディスパッチループを実行するスレッド（イベントスレッド）を作成する。イベントスレッドは、イベントが発生すると、イベントに対応づけられたインスタンスの特定のメソッドを呼び出し、その処理が終わるとまたイベントの待ちを行う。
アプリケーションは、起動時に EN_Node のインスタンスをただ 1 つ作成しなければならない。

（5）リターンコード

なし。

（6）例外

なし。

（7）注意

・API はイベント発生によって呼ばれるメソッドが終了してから、次のイベントの処理を行う。このため、アプリケーションはイベント処理メソッドをリエントラントに書く必要はない。

５．３．２．２ getEA

（1）名称

getEA　　－　自ノードの ECHONET アドレスの取得

（2）機能

自ノードの ECHONET アドレスを返す。

（3）構文

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

（4）説明

自ノードの ECHONET アドレスを返す。

（5）リターンコード

自ノードの ECHONET アドレス。下位２バイトのみ使用。

（6）例外

EAPI_NOTOPEN　　：requestStart() 完了前にコールされた場合

（7）注意

なし。

５．３．２．３　addPropertyEventListener

（1）名称

addPropertyEventListener　－　プロパティ値イベントリスナの登録

（2）機能

他ノードから、自己 ECHONET オブジェクトのプロパティ取得・設定要求（プロパティ値イベント）発生時に呼ばれるリスナーオブジェクトを登録する。

（3）構文

```
public void addPropertyEventListener(
   EN_EventListener listener          //登録リスナーオブジェクト
) throws EN_Eception;
```

（4）説明

他ノードから、自己 ECHONET オブジェクトのプロパティ取得・設定要求（プロパティ値イベント）があった場合に呼び出すリスナーオブジェクトを登録する。
API は登録時に、listener.getEOJ()を呼び出し、結び付けるイベントを決定する。
なお、イベント発生時には イベントの DEOJ が getEOJ()で得た値と等しい *listener* を探す。そして、設定されているアクセスルールを参照し、アクセス可であれば listerner.callbackWriteMyProperty()、listener.callbackReadMyProperty()を呼び出す。最後に、必要に応じて応答電文を送信する。
listener が自己 ECHONET オブジェクトでない場合（listener.getEA()で判断する）は、例外を発生する。

（5）リターンコード

なし。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_NORESOURCE	：登録不可
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー

（7）注意

・同一 ECHONET オブジェクトコードの登録を複数行った場合は、最後に行った登録が有効になる。つまり、登録は同時には一つのみ可能であり、再度同じ EA, EOJ を持つ EN_Object が登録された場合には上書きされ、最後に登録したもののみ有効となる。

５．３．２．４　delPropertyEventListener

（1）名称

delPropertyEventListener　－　プロパティ値イベントリスナの削除

（2）機能

登録したリスナーオブジェクトを削除する。

（3）構文

```
public void delPropertyEventListener (
   EN_EventListener  listener          //削除リスナーオブジェクト
) throws EN_Exception;
```

（4）説明

addPropertyEventListner で登録したリスナーオブジェクトを削除する。
listener が自己 ECHONET オブジェクトでない場合（listener.getEA()で判断する）は、例外を発生する。また、指定されたリスナーオブジェクトが登録されていない場合は、例外を発生する。

（5）リターンコード

なし。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NOTARGET	：対象のリスナー未登録
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー

（7）注意

なし。

５．３．２．５　addNotifyEventListener

（1）名称

addNotifyEventListener　　－　通知イベントリスナの登録

（2）機能

他アプリケーションからの状変アナウンス、一定時間毎の通知、同報した要求の答えのイベント（通知イベント）発生時に呼ばれるリスナーオブジェクトを登録する。イベントは送信元オブジェクトコードにより結び付けられる。

（3）構文

```
public void addNotifyEventListener (
   EN_EventListener  listener          //登録リスナーオブジェクト
) throws EN_Exception;
```

（4）説明

他アプリケーションの状変アナウンス、一定時間毎の通知、同報した要求の答えのイベント（通知イベント）発生時に呼ばれるリスナーオブジェクトを登録する。API は登録時に、listener.getEA()、listener.getEOJ()を呼び出し、結び付けるイベントを決定する。

API は SEA、SEOJ をもつイベントが発生すると、対応する listener を次の順で探し、対応する listener の listener.callbackNotifyEvent()を呼ぶ。（getEA()、getEOJ()は登録時に１度だけ実行される。メソッド呼び出し形式を用いているのは説明の都合であり、何度も呼び出すわけではない）

登録リスナーには、２種類がある。１つは、自己 EA, EOJ を明示的に指定したものであり、これはこの EA, EOJ が受信電文の DEA, DEOJ に含まれていたときに必ず呼ばれるメソッドを登録するものである。これを今,「個別リスナー」と呼ぶ。

もう１つは、受信したい送信元を指定したものであり、同報アドレスという形式で登録されるものである。これは、登録した同報アドレス（ドメイン内同報、サブネット内同報のアドレス）の中に、受信した電文の SEA, SEOJ が含まれていたときに呼ばれるメソッドを登録するものである。これを今、「同報リスナー」と呼ぶ。

listener が自己 ECHONET オブジェクトでない場合（listener.getEA()で判断する）は、例外を発生する。

呼び出しリスナーの検索ロジックは以下の通り。

（検索ステップ１）登録されている個別リスナーの全てに対し、この EA, EOJ が受信電文の DEA, DEOJ に含まれているか否かを調べ、含まれている場合にはリスナーを呼び出す。受信電文中の DEA に、ドメイン内／サブネット内同報アドレスが格納されている場合には、同報の範囲か否かを判定する。また、受信電文中の DEOJ がインスタンス同報である場合には、これも登録リスナーの条件が同報の範囲に含まれているか否かも併せて判定する。

なお、1 回の電文受信で、条件に合致するリスナーが複数登録されている場合には、該当する複数のリスナーが全て呼び出される。例えば、コントローラインスタンス１、コントローラインスタンス２、コントローラインスタンス３の３つのオブジェクトが実装されている場合、それぞれのインスタンスでリスナーを登録しておくと、

エアコンオブジェクトをあて先としたインスタンス同報の受信電文が届いたときにはこの３つ全てのリスナーが呼び出される。
次に、getEA() == SEA、getEOJ() == SEOJ であるリスナを呼び出す。該当リスナがあり、呼び出した場合は検索ステップ５へ、なければ検索ステップ２へ。

（検索ステップ２） getEA()=SEA、getEOJ()と SEOJ のオブジェクトクラスグループとオブジェクトクラスコードがそれぞれ等しく、getEOJ()のインスタンスコードが 0 である *listener* を呼び出す。
ステップ３へ。

（検索ステップ３） getEA()が同報アドレス（ドメイン内同報、サブネット内同報のアドレス）であり、SEA を含み、getEOJ()=SEOJ である *listener* を呼び出す。該当リスナーがあり呼び出した場合には検索ステップ５へ。無ければ検索ステップ４へ。

（検索ステップ４） getEA()が同報アドレス（ドメイン内同報、サブネット内同報のアドレス）であり、SEA を含み、getEOJ()と SEOJ のオブジェクトクラスグループとオブジェクトクラスコードがそれぞれ等しく、getEOJ()のインスタンスコードが 0 である *listener* を呼び出す。
検索ステップ５へ。

（検索ステップ５） getEOJ()がワイルドカードコードである *listener*。
（注）なお、ワイルドカードコードである Listner 呼び出しは、システム内のすべてのオブジェクトの電文を受信するようなアプリケーションを想定した機能である。

（５）リターンコード
なし。

（６）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_NORESOURCE	：登録不可
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー

（７）注意
・同一 ECHONET オブジェクトコードの登録を複数行った場合は、最後に行った登録が有効になる。つまり、登録は同時には一つのみ可能であり、再度同じ EA, EOJ を持つ EN_Object が登録された場合には上書きされ、最後に登録したもののみ有効となる。

５．３．２．６　delNotifyEventListener

（1）名称

`delNotifyEventListener`　　－　通知イベントリスナの削除

（2）機能

他アプリケーションの状変アナウンス、一定時間毎の通知、同報した要求の答えのイベント（通知イベント）発生時に呼ばれるリスナーオブジェクトを削除する。

（3）構文

```
public void delNotifyEventListener(
   EN_EventListener listener          //削除リスナーオブジェクト
) throws EN_Exception;
```

（4）説明

addNotifyEventListner によって登録したリスナーオブジェクトを削除する。登録時に、listener.getEA()、listener.getEOJ()を呼び出し、結び付けるイベントが決定されているので、これらに関する情報も一括削除する。

なお、リスナーオブジェクトを削除せずに、これに結び付けるイベントのみを変更したい場合には、一旦 delNotifyEventListner で削除し、addNotifyEventListner で登録し直すこと。

listener が自己 ECHONET オブジェクトでない場合(listener.getEA()で判断する）は、例外を発生する。また、指定されたリスナーオブジェクトが登録されていない場合は、例外を発生する。

（5）リターンコード

なし。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NOTARGET	：対象のリスナー未登録
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー

（7）注意

なし。

５．３．２．７　addNotifyErrorEventListener

（1）名称

addNotifyErrorEventListener　　―　エラー通知イベントリスナの登録

（2）機能

致命的なエラー通知イベントのリスナーオブジェクトを登録する。

（3）構文

```
public void addNotifyErrorEventListener(
   EN_EventListener  listener          //登録リスナーオブジェクト
) throws EN_Exception;
```

（4）説明

ECHONET 通信ミドルウェア、ECHONET 下位通信ソフトウェアで発生した致命的なエラーを通知してもらう際に呼ばれるリスナーオブジェクトを登録する。

API は 致命的なエラーが発生すると、対応する *listener* の listener.callbackNotifyError ()を呼ぶ。

listener が自己 ECHONET オブジェクトでない場合（listener.getEA()で判断する）は、例外 EAPI_ILLEGAL_PARAM を発生する。

（5）リターンコード

なし。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_NORESOURCE	：:登録不可
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー

（7）注意

・同一 ECHONET オブジェクトコードの登録を複数行った場合は、最後に行った登録が有効になる。つまり、登録は同時には一つのみ可能であり、再度同じ EA, EOJ を持つ EN_Object が登録された場合には上書きされ、最後に登録したもののみ有効となる。

５．３．２．８　delNotifyErrorEventListener

（1）名称

`delNotifyErrorEventListener`　　―　エラー通知イベントリスナの削除

（2）機能

致命的なエラー通知イベントのリスナーとして登録したリスナーオブジェクトを削除する。

（3）構文

```
public void delNotifyErrorEventListener (
   EN_EventListener listener          //削除リスナーオブジェクト
) throws EN_Exception;
```

（4）説明

`addNotifyErrorEventListener` によって登録したリスナーオブジェクトを削除する。

listener が自己 ECHONET オブジェクトでない場合（listener.getEA()で判断する）は、例外を発生する。また、指定されたリスナーオブジェクトが登録されていない場合は、例外を発生する。

（5）リターンコード

なし。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_NOTARGET	：対象のリスナー未登録
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー

（7）注意

なし。

５．３．２．９　end

（1）名称

end　　－　アプリケーションの終了通知

（2）機能

アプリケーションソフトウェアは、終了前にこのメソッドを呼ぶことにより、API で管理している本アプリケーションのためのリソースを解放する。

（3）構文

```
public void end (
) throws EN_Exception;
```

（4）説明

ECHONET 通信ミドルウェア、ECHONET 下位通信ソフトウェアの終了を意味するものではない。

（5）リターンコード

なし。

（6）例外

EAPI_NOTOPEN　　：requestStart() 完了前にコールされた場合
EAPI_ETC_ERROR　　：リトライ可能な軽微なエラー

（7）注意

なし。

５．３．２．１０ notifyTrouble

（1）名称

notifyTrouble　　－　障害通知

（2）機能

アプリケーションソフトウェアの障害の状態をECHONET通信ミドルウェアへ通知する。本通知を受けたECHONET通信ミドルウェアは、アプリケーションソフトウェア異常を保持する。

（3）構文

```
public void notifyTrouble (
   int         Trouble  //障害内容（障害発生、障害解消）
) throws EN_Exception;
```

（4）説明

Trouble で指定された障害内容を必要に応じてミドルウェアへ通知する。

Trouble：トラブル番号

MID_STS_NO_ERR　トラブル解消

MID_STS_APL_ERR　アプリケーションソフト異常

（5）リターンコード

なし。

（6）例外

EAPI_NOTOPEN	：requestStart() 完了前にコールされた場合
EAPI_ILLEGAL_PARAM	：引数不正
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー

（7）注意

なし。

５．３．２．１１ requestInit

（1）名称

requestInit　　－　イニシャル要求

（2）機能

ECHONET 通信ミドルウェアおよび下位通信ソフトウェアに対してイニシャルを要求する。

（3）構文

```
public boolean requestInit (
   int        StartType        //初期化パラメータ
) throws EN_Exception;
```

（4）説明

本メソッドによりノードの ECHONET 通信ミドルウェアが状態遷移する。管理アプリケーションが使用することを想定したメソッド。

初期化パラメータで立ち上げのタイプを指定する。

StartType: 立上げタイプ

MID_WARM_START	ウォームスタート
MID_COLD_START	コールドスタート

（5）リターンコード

イニシャル成功 true、失敗 false。

（6）例外

EAPI_ILLEGAL_PARAM	：引数不正
EAPI_ETC_ERROR	：リトライ可能な軽微なエラー
EAPI_ALREADYOPEN	：既に起動中（requestStart() まで完了している場合）
EAPI_ALREADYINIT	：既に初期化済み（requestInit() が終わっていて、requestStart() はまだ発行されていない場合）

（7）注意

なし。

５．３．２．１２ requestStart

（1）名称

requestStart　　－　スタート要求

（2）機能

ECHONET 通信ミドルウェアおよび下位通信ソフトウェアに対してスタートを要求する。

（3）構文

```
public boolean requestStart (
) throws EN_Exception;
```

（4）説明

本メソッドによりノードの ECHONET 通信ミドルウェアが状態遷移する。管理アプリケーションが使用することを想定したメソッド。

（5）リターンコード

スタート成功 true、失敗 false。

（6）例外

EAPI_ETC_ERROR	：リトライ可能な軽微なエラー
EAPI_ALREADYOPEN	：既に起動中（requestStart() まで完了している場合）
EAPI_NOTINIT	：未初期化（requestInit() を一度も実行することなく requestStart() が呼ばれた場合）

（7）注意

なし。

５．３．３ EN_Property クラス

（1）名称

EN_Property　　－　プロパティラッパークラス

（2）機能

プロパティを表す電文のバイト列 EDT(第 2 部 4.2.9)を保持し、その値を byte や int で作成、参照するためのメソッドを提供する。

値は、コンストラクタで設定し、名前が get で始まるメソッドで取り出す。

（3）構文

public class EN_Property extends Object;

（4）注意

EDT を生で取り扱うメソッドは内部的に必要になるが、ここでは規定しない。

５．３．３．１ EN_Property

（1）名称

EN_Property　　― プロパティコンストラクタ

（2）機能

プロパティを作成する。

（3）構文

構文1:　`public EN_Property(byte b);`

構文2:　`public EN_Property(short s);`

構文3:　`public EN_Property(int i);`

構文4:　`public EN_Property(int m,int size) throws EN_Eception;`

構文5:　`public EN_Property(long l);`

構文6:　`public EN_Property(long m,int size) throws EN_Eception;`

構文7:　`public EN_Property(String st);`

構文8:　`public EN_Property(byte ba[]);`

（4）説明

プロパティ値より電文の EDT を作成し保持する。構文 4、6 ではデータ m を持ち、長さ *size* バイトの EDT を作成する。構文 4 では *size* は 1 から 4、構文 6 では size は 1 から 8 とする。

（5）リターンコード

なし。

（6）例外

EAPI_ILLEGAL_PARAM　　：引数不正（構文４，６で範囲外の size 値を指定した場合）

（7）注意

・setProperty()などで相手の ECHONET オブジェクトのプロパティに値を設定する際には、アプリケーションは該プロパティのデータ型を予め知っておく必要がある。アプリケーションは、API がこのデータ型に合うように電文の EDT を作成できるように適切に EN_Packet をコンストラクトしなければならない。

５．３．３．２ get

（1）名称

get　　－　プロパティ取得アクセサ

（2）機能

プロパティ値を取得するアクセサ。

（3）構文

構文1:
```
public byte getByte(byte b) throws EN_Exception;
```
構文2:
```
public short getShort() throws EN_Exception;
```
構文3:
```
public int getInt() throws EN_Exception;
```
構文4:
```
public long getLong() throws EN_Exception;
```
構文5:
```
public short getShortU() throws EN_Exception;
```
構文6:
```
public int getIntU() throws EN_Exception;
```
構文7:
```
public long getLongU() throws EN_Exception;
```
構文8:
```
public String getString() throws EN_Exception;
```
構文9:
```
public byte[] getByteArray() throws
EN_Exception;
```

（4）説明

プロパティ値を取得するアクセサ。保持しているデータを要求される型に強制的に変換して返す。メソッド名の最後に U が付くものは EN_Property が保持しているデータを符号無しとみなして変換して取得するものである。

なお、byte 型で unsigned とみなした場合の値は Java 言語の byte では表現しきれない値となるので getByteU() は存在しない。

変換規則は以下とする。

・API が EDT で保持しているデータ長を全長と考え、signed /unsigned の判定結果を基に解釈し、その結果を要求された型に納めて返す。

・最上位のビットが立っているデータについては、符号付きのものは元のサイズで表現される型のサイズ「未満」のサイズの型での取得要求には例外 EAPI_ILLEGAL_TYPE をスローする。また、元のサイズで表現される型のサイズ「以上」のサイズの型での取得要求には元のサイズでの評価結果（負値）を要求されている型に（値として）入れて返す。（下記(A)の場合）。

・符号無しのものは、変換元のバイト列の上位に 0 を補って表わされる符号無し整数を指示されたサイズおよび型に（値として）入れて返すのが原則である。元のサイズで表現される型「以下」（つまり丁度を含む）のサイズの型での取得要求には例外 EAPI_ILLEGAL_TYPE をスローし、また、元のサイズで表現される型を超えるサイズの型での取得要求には上位に 0x00 を補完した後の評価結果に従う。

例) {0x01, 0x02} --(getInt)--> 0x00000102
{0x80} --(getByte) --> (byte)-128
{0x80} --(getShortU)--> (short)128
{0x80} --(getShort)--> (short)-128 (byte で 0x80 は -128)
{0x80, 0x00} --(getByte) --> 例外

{0x80, 0x00} --(getShortU)--> 例外(Java 言語の short 型では表現しきれない)

{0x80, 0x00} --(getShort) --> -32768
{0x80, 0x00} --(getIntU)--> 32768
{0x80, 0x00} --(getInt) --> -32768
{0x80, 0x00, 0x00} --(getIntU)--> 8388608
{0x80, 0x00, 0x00} --(getInt) --> -8388608　----(A)
ただし、{0x80, 0x00, 0x00} は int, size = 3 として生成あるいは
byte[]で生成したものとする。
{0x80, 0x00, 0x00, 0x00} --(getIntU)--> 例外
{0x80, 0x00, 0x00, 0x00} --(getInt) --> -2147483648
{0x80, 0x00, 0x00, 0x00} --(getLongU)--> 2147483648
{0x80, 0x00, 0x00, 0x00} --(getLong) --> -2147483648
{0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00} - (getLongU)--> 例外
{0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00} - (getLong)
--> -3458764513820540928
{"01"} == {0x00, 0x30, 0x00, 0x31}--(getInt)--> 0x00300031　((int)1 とはしない)
{"123"} == {0x00, 0x31, 0x00, 0x32, 0x00, 0x33}--(getInt)-->例外
{"AB"} == {0x00, 0x41, 0x00, 0x42}--(getInt)--> 0x00410042
{"ABC"} == {0x00, 0x41, 0x00, 0x42, 0x00, 0x43}--(getInt)--> 例外

・getString() では EN_Property() で保持しているデータを変換できるサイズまで強制的に変換して返す。余ったデータは捨てられる。すなわち奇数バイト保持している場合のみ最後の１バイトが捨てられる。

例) {0x00, 0x30, 0x00, 0x31} --(getString)--> "01"
{0x00, 0x30, 0x00, 0x32, 0x00} --(getString)-->"02"

・getByteArray() は EDT のバイト列をそのまま byte 型の配列型で返す。このため、常に成功することになる。

（５）リターンコード

プロパティ値。

（６）例外

EAPI_ILLEGAL_TYPE　：不正な型指定

（７）注意

・getProperty()で相手の ECHONET オブジェクトのプロパティの値を参照する際には、アプリケーションは該プロパティのデータ型を予め知っておく必要がある。アプリケーションは、API が電文の EDT から正しい型のプロパティ値を作成できるように適切な構文で呼び出さなければならない。

５．３．４ EN_Packet クラス

（1）名称

EN_Packet　　－　ECHONET イベントクラス

（2）機能

本クラスは、ECHONET イベントを表現するためのクラスである。

（3）構文

```
public class EN_Packet extends Object implements EN_Const {
        public EN_Object sourceObject,        //要求元オブジェクト
        public EN_Object destinationObject,   //要求先オブジェクト
        public int EPC,                       //EPC
        public int elementNo,                 //配列要素番号(配列でない場合-1)
        public EN_Property property,          //プロパティ値
        public int esv                        //イベントの種類
}
```

（4）説明

sourceObject	送信元オブジェクトの指定。
destinationObject	送信先オブジェクトの指定。
EPC	EPC(第２部 4.2.7)の値。
elementNo	配列扱いプロパティの要素番号を示す。配列要素でない場合には、－１が入る。
property	プロパティ。電文の EDT に格納される値。配列扱いプロパティの場合、*elementNo* で示す要素の値である。
esv	電文の ESV に格納するコード。callbackWriteMyProperty、callbackReadMyProperty では参照不要。

（5）注意

・内部的に必要になる電文の情報（ただし EDT を除く）は private で持つ。

５．３．５ EN_Exception 例外クラス

（1）名称

EN_Exception　　－　例外クラス

（2）機能

API での例外を表現する。

（3）構文

```
public class EN_Exception extends Exception implements EN_Const
{
        public int type,                    //例外の種類
        public EN_Exception(int type)       //例外コンストラクタ
 }
```

（4）説明

API での例外を表現する。他アプリケーションに対するアクセス要求が処理不可の場合、処理不可応答電文を受ける。この場合、EN_Exception 型の例外がスローされる。アプリケーションは、例外を補足(catch)し、アクセス要求不可の場合の処理を行う。

自己 ECHONET オブジェクトの操作が不可の場合、アプリケーションは、EN_Exception 型の例外をスローすることができる。API は例外を補足し、処理不可応答電文を発行する。

（5）注意

なし。

５．３．６ EN_EventListener インタフェース

（1）名称

EN_EventListener　　－　イベントリスナインタフェース

（2）機能

イベントリスナのインタフェース

（3）構文

```
public interface EN_EventListener {
        //ECHONET アドレス取得
        public int getEA() throws EN_Exception;
        //オブジェクトコード取得
        public int getEOJ();
        //アドレス種別取得
        public int getAddrKind();
        //アクセスルール読み出し
        public int getAccessRule(int EPC) throws EN_Exception;
        //アドレス包含チェック
        public boolean isIn(EN_EventListener x);
        //プロパティ値取得
        public EN_Property callbackReadMyProperty(EN_Packet ev)
 throws EN_Exception;
        //プロパティ値設定
        public boolean callbackWriteMyProperty(EN_Packet ev)
 throws EN_Exception;
        //配列扱いプロパティ値追加
        public boolean callbackAddMyPropertyMember(EN_Packet
 ev) throws EN_Exception;
        //配列扱いプロパティ値削除
        public boolean callbackDelMyPropertyMember(EN_Packet
 ev) throws EN_Exception;
        //配列扱いプロパティ値存在確認
        public boolean callbackCheckMyPropertyMember
 (EN_Packet ev) throws EN_Exception;
        //配列扱いプロパティ値追加
        public boolean callbackAddMyPropertyMemberAlt
 (EN_Packet ev) throws EN_Exception;
        //通知
        public void callbackNotifyEvent(EN_Packet ev) throws
 EN_Exception;
        //エラー通知
        public void callbackNotifyError(int errorCode) throws
```

```
EN_Exception;
}
```

（4）説明

イベントを受け付けるために必要となるインタフェース型。EN_Object はこのインタフェースをインプリメントしているので、アプリケーションはこのインタフェースを意識する必要はない。

（5）注意

なし。

５．３．７ EN_Const インタフェース

（1）名称

EN_Const　　－　ECHONET Java 言語API定数定義インタフェース

（2）機能

本インタフェースは、API で用いられる各種定数を提供する。

（3）構文

```
public interface EN_Const {
        // 関数戻り値、例外タイプ

    // ECHONET 通信ミドルウェア未初期化
    public static final int EAPI_NOTINIT  = -1;
    // ECHONET 通信ミドルウェアは既に初期化済み
    public static final int EAPI_ALREADYINIT  = -2;
    //セッションが未 open 或いは、非起動中( requestStart()完了前に使用不可能な API がコールされた時)
    public static final int EAPI_NOTOPEN  = -3;
    // ECHONET 通信ミドルウェアは既に起動中
    public static final int EAPI_ALREADYOPEN  = -4;
    // 下位通信ソフトウェアエラー
    public static final int EAPI_LOW_ERROR  = -10;
    // プロトコル差異吸収処理部エラー
    public static final int EAPI_PRO_ERROR  = -11;
    // ECHONET 通信処理部エラー
    public static final int EAPI_MID_ERROR  = -12;
    // 一時的リソース不足(送信バッファフルのため送信を受け付けないなど)
    public static final int EAPI_NORESOURCE  = -20;
    #主にメモリ不足、バッファ不足が原因のエラー。しばらくすれば回復が見
    #込める。
    // 未送信データあり(送信を行えないまま指定時間が経過してしまった場合)。「指定時間」の値は、ミドルウェアの実装に依る。どこでエラーが発生したかは問わない。リトライして成功するか否かは不確か。
    public static final int EAPI_NOTSEND  = -21;

    //通信タイムアウト(送信は行えたが timeout 時間内に応答が返って来なかった場合
    public static final int EAPI_TIMEOUT  = -30;
    // 制御不可能（相手 ECHONET オブジェクトから処理不可応答電文を受取った場合）
    public static final int EAPI_NOTOPERATIVE  = -31;
```

```
// 認証エラー（相手 ECHONET オブジェクトから認証エラー電文を
受取った場合）
public static final int EAPI_SEC_ERROR = -32
// その他のリトライ可能な軽微なエラー
public static final int EAPI_ETC_ERROR  = -39;
// パラメータ不正
public static final int EAPI_ILLEGAL_PARAM = -40;
// 対象なし
public static final int EAPI_NOTARGET = -41
// 不正な型指定
public static final int EAPI_ILLEGAL_TYPE = -42
// ECHONET オブジェクトでの処理不可能
public static final int EAPI_NOTACCEPT = -100;

// ID 種別
public static final int APIVAL_NODE_KIND  = 0;   // 機器
ID
public static final int APIVAL_EA_KIND    =    1;    //
ECHONET アドレス
public static final int APIVAL_BROAD_KIND = 2;   // 同報

// ESV コード
public static final int ESV_SetI    = 0x60;// SetI
public static final int ESV_SetC    = 0x61;// SetC
public static final int ESV_Get     = 0x62;// Get
public static final int ESV_INF_REQ = 0x63;// INF_REQ
public static final int ESV_SetMI   = 0x64;// SetMI
public static final int ESV_SetMC   = 0x65;// SetMC
public static final int ESV_GetM    = 0x66;// GetM
public static final int ESV_INFM_REQ = 0x67;// INFM_REQ
public static final int ESV_AddMI   = 0x68;// AddMI
public static final int ESV_AddMC   = 0x69;// AddMC
public static final int ESV_DelMI   = 0x6A;// DelMI
public static final int ESV_DelMC   = 0x6B;// DelMC
public static final int ESV_CheckM  = 0x6C;// CheckM
public static final int ESV_AddMSI  = 0x6D;// AddMSI
public static final int ESV_AddMSC  = 0x6E;// AddMSC
public static final int ESV_Set_Res = 0x71;// Set_Res
public static final int ESV_Get_Res = 0x72;// Get_Res
public static final int ESV_INF     = 0x73;// INF
public static final int ESV_SetM_Res = 0x75;// SetM_Res
```

```
public static final int ESV_GetM_Res = 0x76;// GetM_Res
public static final int ESV_INFM   = 0x77;// INFM
public static final int ESV_AddM_Res = 0x79;// AddM_Res
public static final int ESV_DelM_Res = 0x7B;// DelM_Res
public static final int ESV_CheckM_Res = 0x7C;// CheckM_Res
public static final int ESV_AddMS_Res = 0x7E;// AddMS_Res

public static final int ESV_SetI_SNA = 0x50;// SetI_SNA
public static final int ESV_SetC_SNA = 0x51;// SetC_SNA
public static final int ESV_Get_SNA = 0x52;// Get_SNA
public static final int ESV_INF_SNA = 0x53;// INF_SNA
public static final int ESV_SetMI_SNA = 0x54;// SetMI_SNA
public static final int ESV_SetMC_SNA = 0x55;// SetMC_SNA
public static final int ESV_GetM_SNA = 0x56;// GetM_SNA
public static final int ESV_INFM_SNA = 0x57;// INFM_SNA
public static final int ESV_AddMI_SNA = 0x58;// AddMI_SNA
public static final int ESV_AddMC_SNA = 0x59;// AddMC_SNA
public static final int ESV_DelMI_SNA = 0x5A;// DelMI_SNA
public static final int ESV_DelMC_SNA = 0x5B;// DelMC_SNA
public static final int ESV_CheckM_SNA = 0x5C;// CheckM_SNA
public static final int ESV_AddMSI_SNA = 0x5D;// AddMSI_SNA
public static final int ESV_AddMSC_SNA = 0x5E;// AddMSC_SNA

// アクセスルール
public static final int APIVAL_RULE_SET = 0x0001; // Set
public static final int APIVAL_RULE_GET = 0x0002; // Get
public static final int APIVAL_RULE_ANNO = 0x0040; // Anno
public static final int APIVAL_RULE_SETM = 0x0100; // SetM
public static final int APIVAL_RULE_GETM = 0x0200; // GetM
public static final int APIVAL_RULE_ADDM = 0x0400; // AddM
public static final int APIVAL_RULE_DELM = 0x0800; // DelM
public static final int APIVAL_RULE_CHECKM = 0x1000; //
CheckM
public static final int APIVAL_RULE_ADDMS = 0x2000; // AddMS
public static final int APIVAL_RULE_ANNOM = 0x4000; // AnnoM

// 通信ミドルウェア状態
public static final int MID_STS_NO_ERR = -1;// トラブル解
消
public static final int MID_STS_APL_ERR = -3;// アプリケー
ション状態異常
```

```
        // 通信ミドルウェア初期化パラメータ
        public static final int MID_COLD_START = 0;// コールドスタート
        public static final int MID_WARM_START = 1;// ウォームスタート

        // セキュア通信アクセス制限レベル
        public static final int APIVAL_ACCESS_ANO = 0x0001; // Anonymous レベル
        public static final int APIVAL_ACCESS_USER = 0x0002; // User レベル
        public static final int APIVAL_ACCESS_SP = 0x0003; // Service Provider レベル
        public static final int APIVAL_ACCESS_MAKER = 0x0004; // Maker レベル

        // その他
        public static final int MYSELF_NODE = 0xFFFFFFFF;// 自分自身の EN_Object を指す
    }
```

（4）説明

API で使用する各種定数定義インタフェース。API は本インタフェースをインプリメントしている。アプリケーションは、このインタフェースをインプリメントすることにより、各種定数を参照することができる。

（5）注意

なし。

５．３．８ EN_SecureOpt クラス

（1）名称

EN_SecureOpt　　－　ECHONET セキュア通信オプションクラス

（2）機能

本クラスは、ECHONET セキュア通信オプションを表現する。

（3）構文

```
public class EN_SecureOpt extends Object implements EN_Const {
        public boolean      authentication,   //認証有無指定
        public int          keyIndex,         //セキュアユーザレベル
        public int          cipher,           //暗号方式
        public int          makerKeyIndex     //メーカ Key Index
        public int          makerKey          //メーカ Key
}
```

（4）説明

authentication	認証有無の指定。true: 有、false: 無。
keyIndex	セキュアユーザレベルの指定。 0x00: シリアル Key Index 0x01: user セキュア Key Index 0x03: maker セキュア Key Index 0x04～0x15: service provider セキュア Key Index
cipher	暗号化方式の指定。 ver.2.10 で有効な指定は、0x00（ブロック暗号）のみ。
makerKeyIndex	*keyIndex* でメーカ Key Index を指定した場合に用いる。メーカ Key インデックスは、上位から、メインインデックス MIX(3Byte)、およびサブインデックス(SIX)(1Byte)から構成する。ここで指定された値は、メーカ Key 暗号・認証ヘッダ形式、もしくはメーカ Key 暗号ヘッダ形式において、暗号・認証に用いる共有鍵のインデックスを示す際に用いられる。なお、*keyIndex* でメーカ Key Index 以外を指定した場合には、この値は評価しない。
makerKey	*keyIndex* でメーカ Key Index を指定した場合に用いる。メーカ Key そのものを格納する。

（5）注意

なし。

５．３．９ EN_CpException 例外クラス

（1）名称

EN_CpException　　－　複合電文処理例外クラス

（2）機能

複合電文処理に関する API での例外を表現する。

（3）構文

```
public class EN_CpException extends Exception implements
EN_Const {
        public int          type[],         //例外の種類
        public int          EPC[],          //EPC
        public EN_Property  p[],            //プロパティ
        public EN_CpException(int type)     //例外コンストラクタ
 }
```

（4）説明

API での複合電文処理に関する例外を表現する。他アプリケーションに対するアクセス要求が処理不可の場合、処理不可応答電文を受ける。この場合、EN_CpException 型の例外がスローされる。アプリケーションは、例外を補足(catch)し、アクセス要求不可の場合の処理を行う。

（5）注意

・*type* の値としては、以下のものがある。

EAPI_CpError_Success
EAPI_CpError_NotAccepted
EAPI_CpError_Unconfirm