OpenEL® 3.2 仕様書

02 版

2020 年 12 月 25 日

改版履歴

日付	版数	変更箇所	内容
2020 年 12 月 25 日	01	-	OpenEL 3.2 版策定
2021 年 1 月 26 日	02		命名規約

目次

1. はじめに

本書は、一般社団法人組込みシステム技術協会(以下 JASA という)が提唱する、OpenEL（Open Embedded Library）3.2 版の仕様を説明するものである。

本書の目的は、デバイスを作成するメーカ、デバイスを使用する開発者及びソフトウェア利用者が、OpenEL 仕様に準拠したインタフェースを構築するために、OpenEL 対応デバイスの API 仕様を規定することである。

本書は以下の読者を対象とする。

- OpenEL を用いて、ミドルウェアやソフトウェアを開発したり利用したりするソフトウェア技術者
- OpenEL に準拠したデバイスとコンポーネントを開発し供給するデバイスベンダ及びその技術者
- ロボット及び組込みソフトウェアの開発に興味のある技術者

2. ライセンス

本仕様書の著作権は JASA に帰属する。

OpenEL は、一般社団法人組込みシステム技術協会の日本における登録商標である。

OpenEL のロゴは、一般社団法人組込みシステム技術協会の商標である。

JASA は、本仕様書の全文または一部分を改変することなく複写し、無料または実費程度の費用で再配布することを許諾する。ただし、本仕様書の一部を抜粋して再配布する場合には、OpenEL 3.2 仕様書からの抜粋であること、抜粋した箇所、及び本仕様書の全文を入手する方法を明示(http://www.jasa.or.jp/ など)することを条件とする。

その他、本仕様ならびに本仕様書の利用条件の詳細については次節に記載する。

本仕様ならびに本仕様書に関する問合せ先は、巻末の「8.問合せ先」を参照のこと。

仕様の利用条件

OpenEL 3.2 仕様は、オープンな仕様である。誰でも自由に、OpenEL 3.2 仕様に準拠したソフトウェアを開発・使用・配布・販売することができる。

ただし、JASA は、OpenEL 3.2 仕様に準拠したソフトウェアの製品ドキュメントなどに、以下の文言を入れることを強く推奨する。

> OpenEL は、一般社団法人組込みシステム技術協会の日本における登録商標です。
> OpenEL のロゴは、一般社団法人組込みシステム技術協会の商標です。

また、OpenEL 3.2 仕様に準拠したソフトウェアの製品ドキュメントなどに、以下の文言を入れることを推奨する。

> OpenEL 3.2 仕様は、JASA が定めたオープンな（組込みシステム API）仕様です。
> OpenEL 3.2 仕様の仕様書は、JASA のホームページから入手することが出来ます。

仕様書を改変して製品ドキュメントを作成する許諾を受けた場合などは、これらの 2 つの推奨に従うことが条件となる。

仕様書の利用条件

OpenEL 3.2 仕様書の著作権は JASA に帰属する。

JASA は、OpenEL 3.2 仕様書の全文または一部分を改変することなく複写し、無料または実費程度の費用で再配布することを許諾する。ただし、OpenEL 3.2 仕様書の一部を再配布する場合には、OpenEL 3.2 仕様書からの抜粋である旨、抜粋した箇所、及び OpenEL 3.2 仕様書の全文を入手する方法を明示しなければならない。

また、JASA は事前に書面による承諾が無い限り、OpenEL 3.2 仕様書を改変することを堅く禁止する。JASA は、JASA 会員に対してのみ、OpenEL 3.2 仕様書を改変して製品ドキュメントを作成し、それを配布・販売することを許諾している。許諾条件やその申請方法については、JASA に問い合わせること。

無保証

JASA は、OpenEL 3.2 仕様及び仕様書に関して、いかなる保障も行わない。また、OpenEL 3.2 仕様及び仕様書を利用したことによって、直接的または間接的に発生した如何なる損害も補償しない。

また、JASA は、OpenEL 3.2 仕様書を予告無く改定する場合がある。

【今後開始予定】OpenEL 仕様準拠品登録制度

JASA では、OpenEL 仕様の普及と発展を促進するため、OpenEL 仕様準拠製品登録制度を設ける予定である。

この制度は、各社で開発された OpenEL 仕様準拠の製品の一覧を作成・保守し、JASA の広報活動などを通じて、OpenEL 仕様ならびに準拠製品の普及を図ることを目的としたものである。なお、この制度は、いわゆる検定制度とは異なり、登録された製品が OpenEL 仕様に準拠していることを認証するためのものではない。

この制度に登録されている製品の一覧は、JASA のホームページ(http://www.jasa.or.jp/)で閲覧可能となる予定である。

この制度は開始の準備が整い次第、JASA のホームページで制度開始のアナウンスと利用・登録方法を一般に公開する。

3. 用語

本仕様書で使用する用語を定義する。

但し以降の章・節で定義・説明される用語については、該当箇所への参照に留める。また、以降の本文では、固有名詞である OpenEL と JASA を除き、本仕様書固有の用語として太字・斜体（例：***Component***）で表記する。

表 3-1　用語一覧

No	用語	意味・用途
1	OpenEL	Open Embedded Library。制御システムのソフトウェアの実装仕様を標準化する組込みシステム向けのオープンなプラットフォーム。
3	JASA	一般社団法人 組込みシステム技術協会の略称。 組込みシステム（組込みソフトウェアを含めた組込みシステム技術をいう。以下同じ。）における応用技術に関する調査研究、標準化の推進、普及及び啓発等を行うことにより、組込みシステム技術の高度化及び効率化を図り、もって我が国の産業の健全な発展と国民生活の向上に寄与することを目的とする、業界団体である。 OpenEL 仕様の策定・管理も本団体にて行っている。
3	Surface レイヤ	OpenEL の構成要素。 詳細は 6.1 章に記載する。
4	Device レイヤ	OpenEL の構成要素。 詳細は 6.2 章に記載する。
5	Component	OpenEL 3.2 に準拠する ***Device レイヤ***を構成するモジュールの総称。モータやセンサなど、OpenEL 3.2 に準拠するハードウェアデバイスとともに、ベンダより提供される。詳細は 6.2 章に記載する。
6	OpenEL コンポーネント名	OpenEL の Component をユニークに判別する名称。 詳細は 6.2 章に記載する。
7	Device Kind (デバイス名／Device Kind ID)	モータやジャイロセンサなど、OpenEL 3.2 に準拠したデバイスの種類を表す。一つの***デバイス***毎に***デバイス名***と ***Device Kind ID*** が割り当てられる。***デバイス名***及び ***Device Kind ID*** は、***OpenEL コンポーネント名***のユニーク性を保つため JASA にて管理される。詳細は 6.2 章に記載する。
8	ベンダ (ベンダ名／ベンダ ID)	OpenEL 3.2 に準拠する ***Component*** の提供者・開発者の個人、法人、団体及びグループを表す略称。 ***ベンダ名***及び***ベンダ ID*** は、***OpenEL コンポーネント名***のユニーク性を保つため JASA にて管理される。詳細は 6.2 章に記載する。

4. OpenEL とは

OpenEL (Open Embedded Library)とは、ロボットや制御システムなどのソフトウェアの実装仕様を標準化する組込みシステム向けのオープンなプラットフォームである。具体的には、センサ入力やモータへの出力等、機器制御のための API(Application Program Interface)をミドルウェアよりも低いレイヤで標準化し、デバイスドライバ等のソフトウェアの移植性、再利用性、生産性を向上するための仕組みである。

組込みシステムの開発においてデバイスドライバをはじめ既存のソフトウェアを異なるシステムに移植するにはかなりの工数を必要とする。例えば、異なるハードウェア上の LED を点灯させたり、モータを動作させたりするだけで、何日も費やすこともあり得る。これは、センサの入出力やモータの制御などを行うアプリケーションプログラムのインタフェースが、各デバイスメーカにより独自に定義、実装されてきたためである。

OpenEL では、ロボットや制御システムなどのソフトウェアのベースとなる部分をプラットフォーム化することにより、異なるハードウェアでもすぐにアプリケーションを動作可能とすることを目的とする。これにより、ソフトウェアの移植性や再利用性が高まり、その結果、品質向上、コスト削減、生産性向上につながるなど、開発者および利用者の利便性が向上する。

図 4-1 に、OpenEL のユースケースを、図 4-2 に OpenEL のアーキテクチャを、図 4-3 に OpenEL の位置付けを示す。

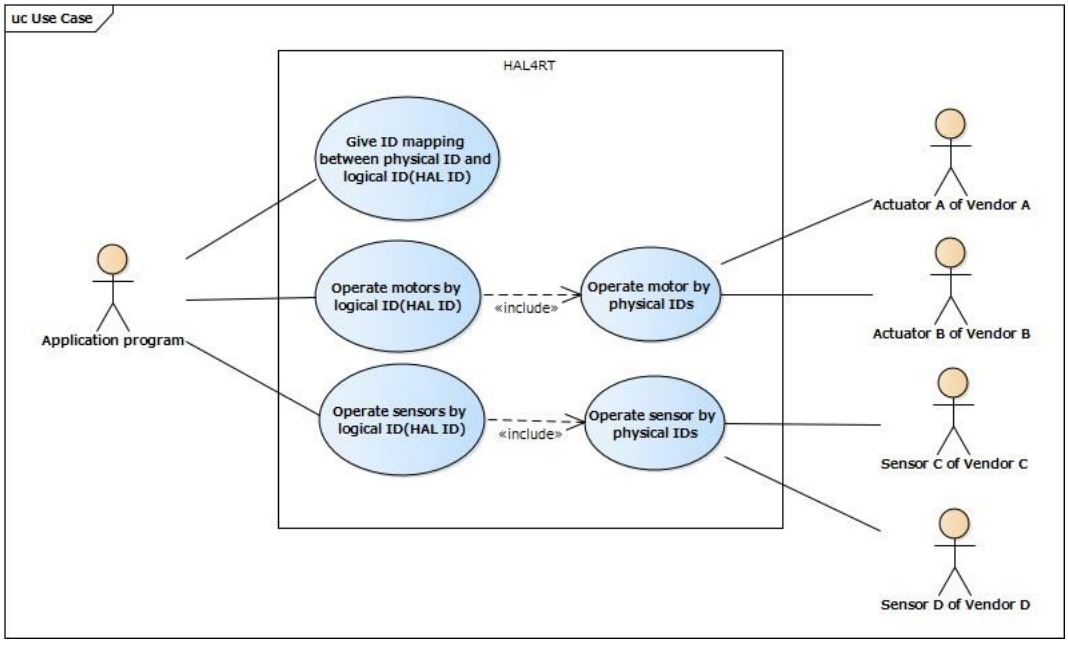

図 4-1 OpenEL のユースケース

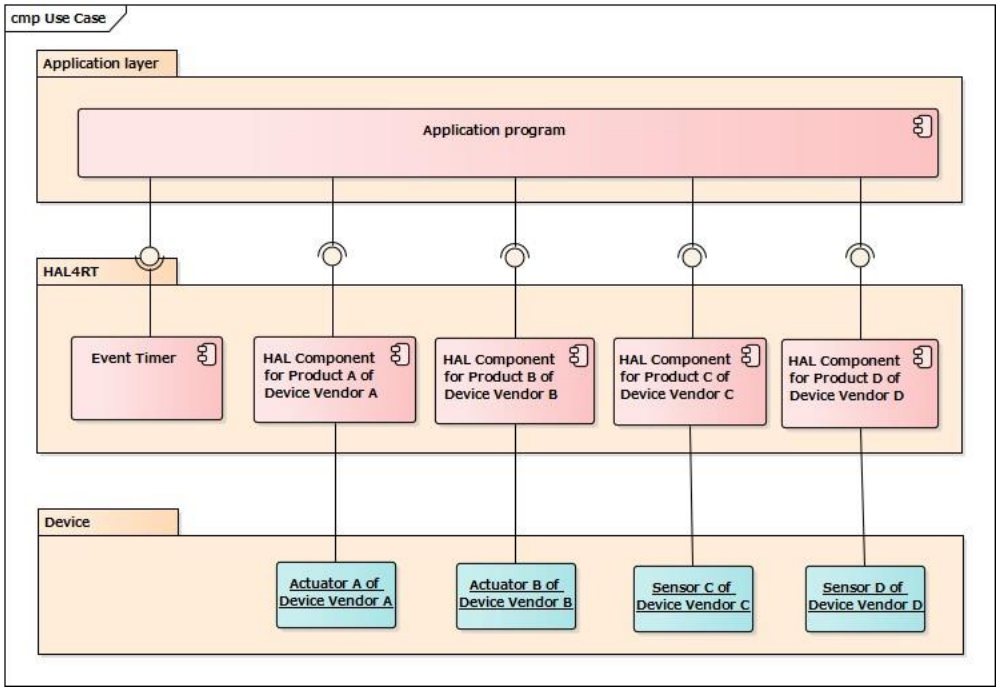

図 4-2 OpenEL のアーキテクチャ

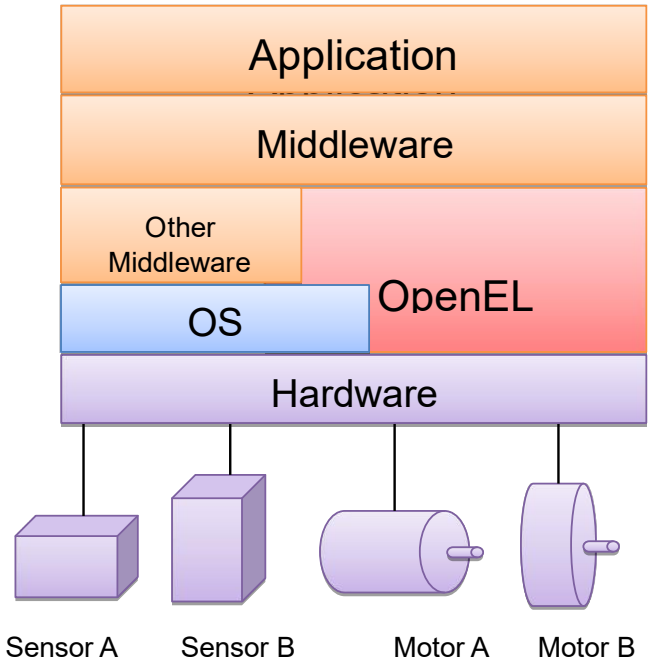

図 4-3 OpenEL の位置づけ

OpenEL は、***Surface*** と ***Device*** のレイヤから構成される。

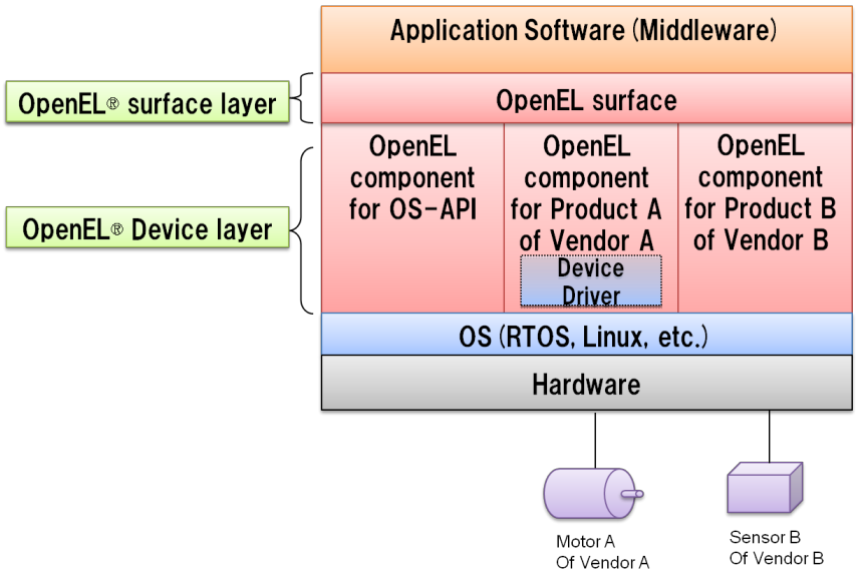

図 4-4 OpenEL の構成図

また、以下のように OS(Operating System)を搭載しない構成にも対応する。

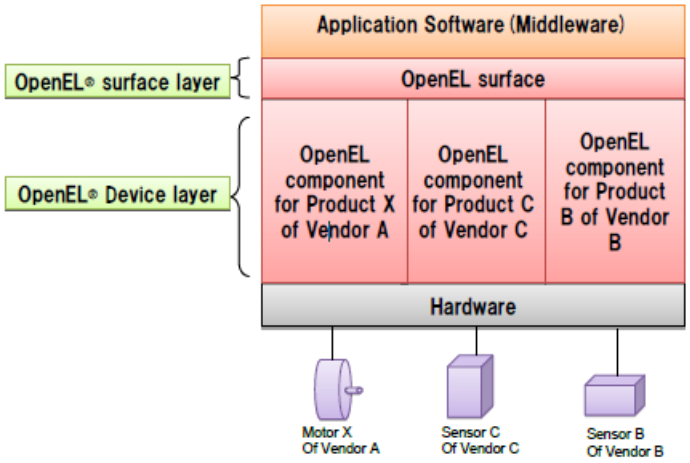

図 4-5 OS 非搭載時の構成

4.1 Surface(サーフェース) レイヤ

OpenEL 利用者向けに提供する API(Application Program Interface)を定義する。ミドルウェアを含むアプリケーションソフトウェアの開発者は、***Surface レイヤ***に定義された API のみを使用することによって、各デバイスハードウェアの差異を意識することなくソフトウェアの構築が可能となる。

Surface レイヤは通常 JASA より提供と公開がなされ、改版・追加・変更・削除は本仕様書に基づき JASA のみが実施する。

4.2 Device (デバイス)レイヤ

ミドルウェアを含むアプリケーションソフトウェアの開発者は ***Device レイヤ***の存在を意識する必要はない。アプリケーションソフトウェアから***デバイス ID***として渡された識別子を、各ハードウェアの実装に合わせ物理ポート番号などに変換し実際のデバイスハードウェアに動作指令を伝達する。

Device レイヤは、モータやセンサなどのハードウェアと共に、使用するデバイス毎にそのデバイスの製造者又は供給者により、OpenEL 仕様に準拠した ***Component***という形で提供される。この ***Component*** により、***Surface レイヤ***の API から実際のデバイス操作への橋渡しを行う。

Device レイヤはデバイスドライバソフトウェアを含む処理の実体であることも出来るし、デバイスドライバソフトウェアのラッパ関数であることも可能であり、実行可能バイナリモジュール又はソースファイルにて提供される。

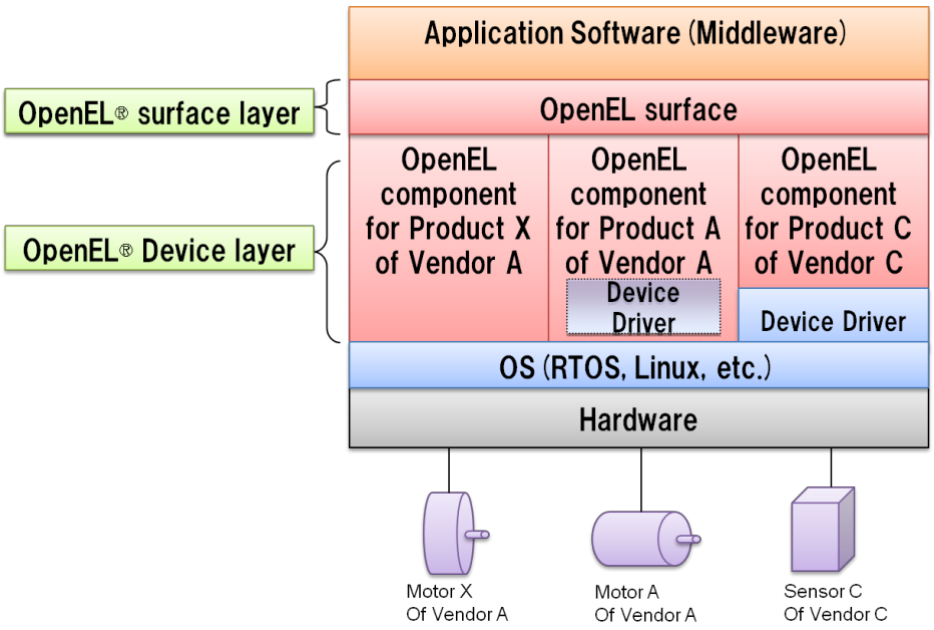

図 4-6 Device レイヤとデバイスドライバ

5. OpenEL の実装

5.1 バージョンと互換性

OpenEL の実装は、以下 3 要素からなるバージョン番号によって定義され公開される。

OpenEL (X). (Y). (Z)

表 5-1 OpenEL バージョン管理ルール

No	要素	意味	概要
1	(X)	メジャーバージョン番号	アーキテクチャや仕様の大幅変更があった場合に変更される。 メジャーバージョンが異なる場合は、仕様として一切の互換性を保証しない。
2	(Y)	マイナーバージョン番号	本仕様書が改版され公開される単位であり、API の追加、修正などにより変更される。 新しい公開ほど大きな数字が割り当てられ、同一メジャーバージョンであれば、可能な限り下位互換性を保つ努力が行われる。
3	(Z)	バグフィックス	バグフィックスや局所的な修正により変更される。新しい公開ほど大きな数字が割り当てられ、通常仕様上の差異は発生しない。

なお、OpenEL のバージョン変更は JASA の都合により適時実施され、(X)・(Y)・(Z)それぞれの値は、JASA により OpenEL 仕様の公開時に決定される。その際、バージョン変更や特定バージョンの公開中止に関する事前告知や互換性維持は JASA の努力義務であり、これらは完全に保証されるものではない。

5.2 実装方法

OpenEL 3.2 版は ISO/IEC 9899:1999 にて定義された C 言語(一般に C99 と呼ばれる)と ISO/IEC 14882:2011 にて定義された C++(一般に C++11 と呼ばれる)と ISO/IEC 23270:2003 にて定義された C#にて実装されることを前提としている。OpenEL 3.2 仕様の API は、C 言語もしくは C++もしくは C#の関数として定義される。

6. OpenEL の構成

6.1 OpenEL のモデル

図 6-1 に OpenEL のプラットフォーム非依存モデルのクラス図を示す。

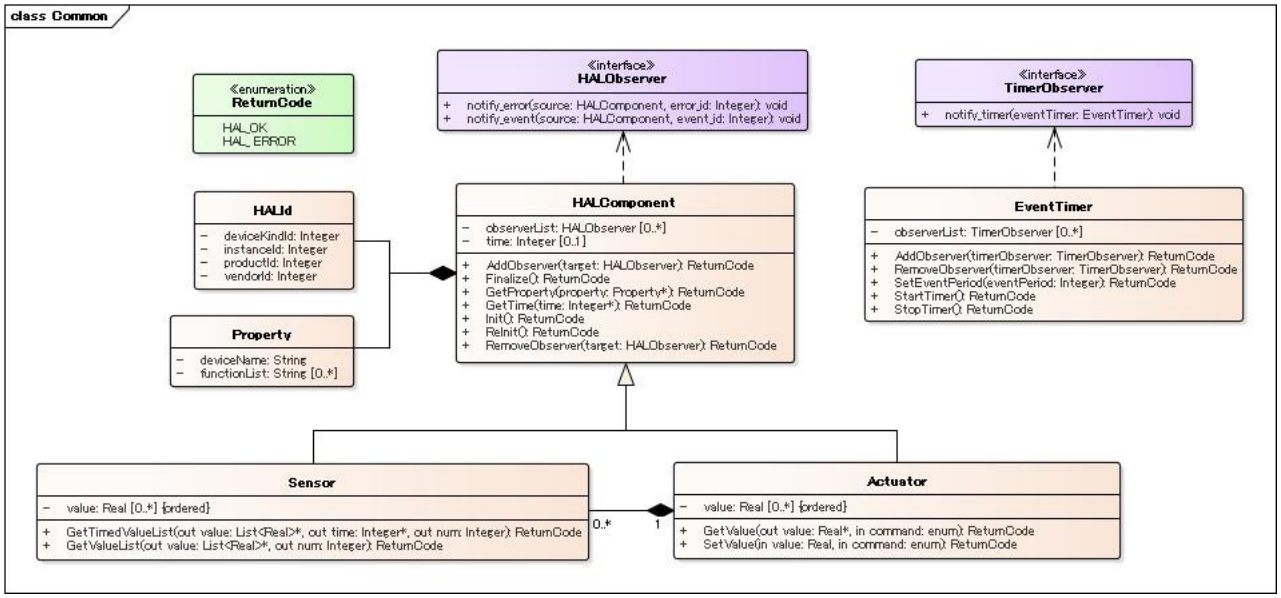

図 6-1 OpenEL のプラットフォーム非依存モデルのクラス図

6.2 OpenEL のコンポーネント

OpenEL のコンポーネントである HALComponent は全てのコンポーネントに共通な情報を持つ要素である。ジンバル付きカメラのように、Actuator と Sensor の両方の属性を持つ HAL コンポーネントを定義することも可能である。HALComponent は以下の識別子を具備する。

① Device Kind ID

OpenEL に準拠するデバイスは、モータやジャイロセンサなどのデバイス毎にその種別を表す ***Device Kind ID*** が JASA にて割り当てられる。***Device Kind ID*** は 32bit 符号無し整数型で定義され、0x00000000~0xFFFFFFFF の値を持つ。***Device Kind ID*** は、表 6-1.2 項に示される***プロファイル名***定義時に併せて発行され、全て JASA にて管理が行われる。

OpenEL3.2 に未定義のデバイスを使用した ***Component*** を新たに開発しようとする***ベンダ***は、JASA に申請し承認される必要がある。登録された ***Device Kind ID*** は、本仕様書別紙.Device Kind ID 一覧にて公開される。

② ベンダ ID

OpenEL に準拠する ***Component*** を開発・提供する***ベンダ***には、JASA から***ベンダ ID*** を割り当てる。これは、表 6-1.3 項に示される***ベンダ名***と 1 対 1 になり、***ベンダ名***と同時に払

い出される。

ベンダ ID は 32bit 符号無し整数型で定義され、0x00000000~0xFFFFFFFF の値を持ち、全ての値は JASA にて管理される。割り当て済み ***ベンダ ID*** は別紙の一覧にて開示される。

③ プロダクト ID

OpenEL に準拠する ***Component*** は、ベンダにより ***Component*** をユニークに識別可能なプロダクト ID が割り当てられる。製品名や型番を表す ***ベンダ*** 固有の ID であり、市場に存在する全 ***Component*** は、以下の形で必ずユニークな値となり、①で示す ***OpenEL コンポーネント名*** が特定可能となる。従って、表 6-1.4 項に示される製品シリーズ名と 1 対 1 になることが望ましい。

プロダクト ID は、32bit 符号無し整数型で定義され、0x00000000~0xFFFFFFFF の値を持ち、0x00000000 及び 0xFFFFFFFF を除き ***Component*** を開発する ***ベンダ*** により自由に割り当て可能である。但し、***OpenEL コンポーネント名*** 登録時にあわせて申請することが必要であり、JASA にて承認し管理される。

④ インスタンス ID

一つのアプリケーションや一つの処理単位に、同一 ***プロファイル*** の OpenEL に準拠するデバイスが複数接続された場合は、インスタンス ID により識別する。インスタンス ID は 32bit 符号無し整数型で定義され、0x00000000~0xFFFFFFFF の値を持つ。実際の割り当てにおけるルールは、デバイスの接続方法やデバイス固有の条件を伴うため、***Component*** を開発・提供する ***ベンダ*** からのドキュメントに記載する。

⑤ HAL ID

HAL ID は、Device Kind ID、ベンダ ID、プロダクト ID、インスタンス ID から構成される。HAL ID により、OpenEL デバイスを識別する。

⑥ プロパティ

プロパティは、デバイス名と関数リストにより構成される。プロパティにより、OpenEL コンポーネントの機能の詳細を知ることが可能である。

⑦ OpenEL コンポーネント名

Component は固有の名前 ***OpenEL コンポーネント名*** を割り当てられる。以下の規則にて決定し、割り当てられた ***OpenEL コンポーネント名*** は JASA にて承認し管理される。

Hal ＋(*デバイス名*) ＋(*ベンダ名*) ＋(製品シリーズ名)

表 6-1 コンポーネント命名規約

No	名称	決定者	概要
1	Hal	固定	OpenEL に準拠した Component であることを表す prefix。
2	(*デバイス名*)	OpenEL 仕様	デバイスの種類を表す名称。 (例:Motor, Sensor 等)
3	(*ベンダ名*)	JASA	デバイスのベンダを表す名称。 大文字から始まる 2～16 文字の英数字。 JASA が提案する単語からベンダが選定し申請する。
4	(製品シリーズ名)	*ベンダ*	製品名や型番を表すベンダ固有の名称。 大文字から始まる 2～16 文字の英数字。

例:ベンダ「XXX」、製品名「YYY」のモータ :「HalMotorXXXYYY」

図6-2にHALComponentの状態遷移図を示す。デバイス固有の状態遷移は、アクティブ状態内で定義される。

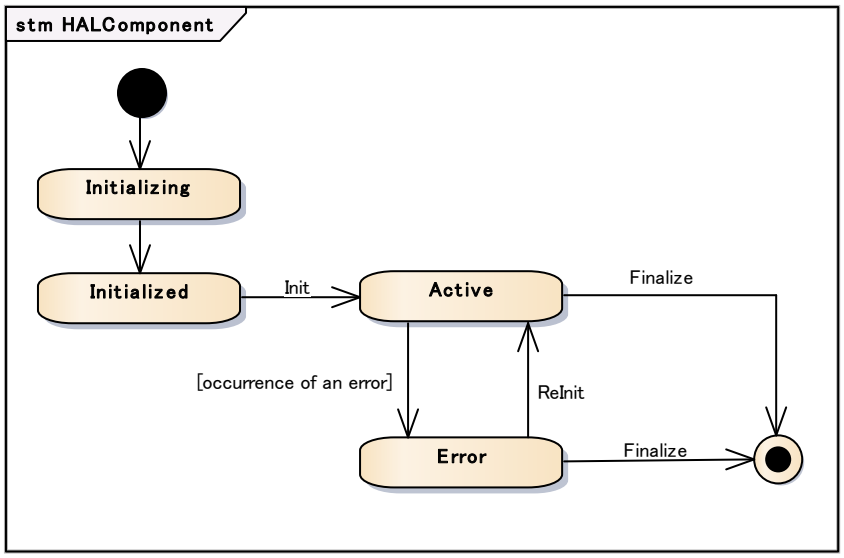

図 6-2 HALComponent の状態遷移図

各状態の詳細を下記に示す。

- 初期化中：デバイス固有の処理が実行されている状態。HALComponentのメソッドを呼び出すことはできない。
- 初期化完了：デバイスの初期化が完了した状態。Init()のみを呼び出すことができる、
- アクティブ：HALComponentとして動作している状態。Init()とReInit()以外のHALComponentの全てのAPIを呼び出すことができる。
- エラー：内部エラーなどにより動作を停止した状態。ReInit()とFinalize()を呼び出すことができる。

7. API 仕様(C 言語)

OpenEL 3.2 版にて定義する C 言語の API を示す。OpenEL 3.2 版に準拠する製品は、該当するデバイス毎に全 API を実装しなければならない。製品の特性上機能しない API(例えば、動作の開始・停止を制御できないセンサの場合など)は、API の戻り値としてエラー(表 7-2 参照)を返却する様に実装すること。

7.1 型定義

OpenEL 3.2 版では、5 章にて規定されるよう ISO/IEC 9899:1999 にて定義された C 言語(一般に C99 と呼ばれる)で実装される。従って、標準の型定義は標準ヘッダファイル<stdint.h>及び<stdbool.h>に従った定義を使用する。

それ以外の OpenEL 3.2 版固有型としては、以下を使用する。

表 7-1 OpenEL 3.2 固有型定義一覧

No	型定義	説明	備考
1	HALFLOAT_T	32bit 浮動小数 または 64bit 浮動小数	
2	HAL_FNCTBL_T	構造体	関数テーブル
3	HAL_REG_T	構造体	HALComponent レジストリ
4	HALRETURNCODE_T	列挙型	表 7-2 共通エラーを参照
5	HALCOMPONENT_T	構造体	
6	HAL_ARGUMENT_T	共用体	
7	HALEVENTTIMER_T	構造体	
8	HALTIMEROBSERVER_T	構造体	

下記に関数テーブルと HALComponent レジストリを示す。関数テーブルの配置は次のようにする。

基底クラス	0～15
派生クラス	16～27
デバイスベンダー	28～31
以降の拡張用に予約	32～ (openEL 3.2 で関数テーブルの実装をしない)

```
/** general component --- function table */
typedef struct HalFncTbl_st {
        /* 0x00 */ HALRETURNCODE_T (*pFncInit)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Initialize */ ¥
        /* 0x01 */ HALRETURNCODE_T (*pFncReInit)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< ReInit */ ¥
        /* 0x02 */ HALRETURNCODE_T (*pFncFinalize)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Finalize */ ¥
        /* 0x03 */ HALRETURNCODE_T (*pFncAddObserver)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< AddObserver */ ¥
        /* 0x04 */ HALRETURNCODE_T (*pFncRemoveObserver)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**<
RemoveObserver */ ¥
        /* 0x05 */ HALRETURNCODE_T (*pFncGetProperty)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< GetProperty */ ¥
        /* 0x06 */ HALRETURNCODE_T (*pFncGetTime)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< GetTime */ ¥
        /* 0x07 */ HALRETURNCODE_T (*pFncDummy07)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */

        /* 0x08 */ HALRETURNCODE_T (*pFncDummy08)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x09 */ HALRETURNCODE_T (*pFncDummy09)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x0A */ HALRETURNCODE_T (*pFncDummy0A)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x0B */ HALRETURNCODE_T (*pFncDummy0B)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x0C */ HALRETURNCODE_T (*pFncDummy0C)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x0D */ HALRETURNCODE_T (*pFncDummy0D)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x0E */ HALRETURNCODE_T (*pFncDummy0E)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x0F */ HALRETURNCODE_T (*pFncDummy0F)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */

        /* 0x10 */ HALRETURNCODE_T (*pFncDummy10)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x11 */ HALRETURNCODE_T (*pFncDummy11)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x12 */ HALRETURNCODE_T (*pFncDummy12)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x13 */ HALRETURNCODE_T (*pFncDummy13)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x14 */ HALRETURNCODE_T (*pFncDummy14)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x15 */ HALRETURNCODE_T (*pFncDummy15)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x16 */ HALRETURNCODE_T (*pFncDummy16)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */
        /* 0x17 */ HALRETURNCODE_T (*pFncDummy17)(HALCOMPONENT_T*,HAL_ARGUMENT_T*); /**< Reserved */

        /* 0x18 */ HALRETURNCODE_T (*pFncSetValue)(HALCOMPONENT_T *,HAL_ARGUMENT_T*); /**< SetValue */
        /* 0x19 */ HALRETURNCODE_T (*pFncGetValue)(HALCOMPONENT_T *,HAL_ARGUMENT_T*); /**< GetValue */
        /* 0x1A */ HALRETURNCODE_T (*pFncGetValueList)(HALCOMPONENT_T *halComponent,HAL_ARGUMENT_T*); /**<
GetValueList */
        /* 0x1B */ HALRETURNCODE_T (*pFncGetTimedValueList)(HALCOMPONENT_T *halComponent,HAL_ARGUMENT_T*);
/**< GetTimedValueList */
```

```
		/* 0x1C */ HALRETURNCODE_T
(*pFncDeviceVendor1C)(HALCOMPONENT_T*,HAL_ARGUMENT_T*,HAL_ARGUMENT_DEVICE_T *); /**< Device Vendor Function */
		/* 0x1D */ HALRETURNCODE_T
(*pFncDeviceVendor1D)(HALCOMPONENT_T*,HAL_ARGUMENT_T*,HAL_ARGUMENT_DEVICE_T *); /**< Device Vendor Function */
		/* 0x1E */ HALRETURNCODE_T
(*pFncDeviceVendor1E)(HALCOMPONENT_T*,HAL_ARGUMENT_T*,HAL_ARGUMENT_DEVICE_T *); /**< Device Vendor Function */
		/* 0x1F */ HALRETURNCODE_T
(*pFncDeviceVendor1F)(HALCOMPONENT_T*,HAL_ARGUMENT_T*,HAL_ARGUMENT_DEVICE_T *); /**< Device Vendor Function */
} HAL_FNCTBL_T;
/** HALComponent registry */
typedef struct HalReg_st {
		int32_t deviceKindID;
		int32_t vendorID;
		int32_t productID;
		HAL_FNCTBL_T *pFncTbl;
		int32_t szHalComponent;
} HAL_REG_T;
extern const HAL_REG_T HalRegTbl[]; /**< HALComponent registry table */
extern const int32_t hal_szRegTbl; /**< HALComponent registry table size */
```

7.2　共通エラー

OpenEL 3.2 に準拠する API は、共通の戻り値として以下を実装する。API に定義された処理が正常に完了した場合は、必ず HAL_OK を返却する。ベンダ固有エラーやデバイス固有のエラーは、この共通エラーに被らない範囲で定義すること。

表 7-2 共通エラー定義

定義名	値	説明
HAL_OK	0	正常
HAL_ERROR	1	異常

7.3 イベントタイマ

イベントタイマは、POSIX、組み込みシステム向け RTOS、その他の OS などで実装されているイベントタイマの操作を統一したものである。

表 7-3 イベントタイマ開始 API

関数名	EventTimerStartTimer	実装版数	3.0 以降
HALRETURNCODE_T EventTimerStartTimer(HALEVENTTIMER_T *eventTimer)			
イベントタイマを開始する。			
パラメータ	型	名前	説明
	HALEVENTTIMER_T	*eventTimer	イベントタイマ（OUT パラメータ）
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

表 7-4 イベントタイマ停止 API

関数名	EventTimerStopTimer	実装版数	3.0 以降
HALRETURNCODE_T EventTimerStopTimer(HALEVENTTIMER_T *eventTimer)			
イベントタイマを停止する。			
パラメータ	型	名前	説明
	HALEVENTTIMER_T	*eventTimer	イベントタイマ（OUT パラメータ）
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

表 7-5 イベントタイマ周期設定 API

関数名	EventTimerSetEventPeriod	実装版数	3.0 以降
HALRETURNCODE_T EventTimerSetEventPeriod(HALEVENTTIMER_T *eventTimer, int32_t eventPeriod)			
イベントタイマの周期を設定する。			
パラメータ	型	名前	説明
	HALEVENTTIMER_T	*eventTimer	イベントタイマ（OUT パラメータ）
	uint32_t	eventPeriod	イベントタイマの周期（単位[ミリ秒]）
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

表 7-6 Timer オブザーバ登録 API

関数名	EventTimerAddObserver	実装版数	3.0 以降
HALRETURNCODE_T EventTimerAddObserver(HALEVENTTIMER_T *eventTimer, HALTIMEROBSERVER_T *timerObserver)			
タイムアウトイベント発生時に呼び出される TimerObserver を追加する。			
パラメータ	型	名前	説明
	HALEVENTTIMER_T	*eventTimer	イベントタイマ（OUT パラメータ）
	HALTIMEROBSERVER_T	*timerObserver	タイマオブザーバ（IN パラメータ）
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

表 7-7 Timer オブザーバ削除 API

関数名	EventTimerRemoveObserver	実装版数	3.0 以降
HALRETURNCODE_T EventTimerRemoveObserver(HALEVENTTIMER_T *eventTimer, HALTIMEROBSERVER_T *timerObserver)			
タイムアウトイベント発生時に呼び出される TimerObserver を削除する。			
パラメータ	型	名前	説明
	HALEVENTTIMER_T	*eventTimer	イベントタイマ（OUT パラメータ）
	HALTIMEROBSERVER_T	*timerObserver	タイマオブザーバ（IN パラメータ）
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

7.4 オブザーバ

7.4.1 HALObserver

HALObserver は、OpenEL コンポーネントで発生したイベントをアプリケーションに通知するためのインターフェースである。

表 7-8 イベント通知 API

関数名	notify_event	実装版数	3.0 以降
void notify_event(HALCOMPONENT_T *halComponent, int32_t event_id)			
OpenEL コンポーネントで発生したイベントをアプリケーションに通知する。			
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント
	int32_t	event_id	イベント ID (IN パラメータ)
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

イベント ID	意味
0	予約
1	Completed：動作完了
2 以降	コンポーネント依存

表 7-9 エラー通知 API

関数名	notify_error	実装版数	3.0 以降
void notify_error(HALCOMPONENT_T *halComponent, int32_t error_id)			
OpenEL コンポーネントで発生したエラーをアプリケーションに通知する。			
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント
	uint32_t	error_id	エラーID (IN パラメータ)
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

エラーID	意味
0	予約
1 以降	コンポーネント依存

7.4.2 TimerObserver

TimerObserver は、イベントタイマで発生したタイムアウトイベントをアプリケーションに通知するためのインターフェースである。

表 7-10 タイムアウトイベント通知 API

関数名	notify_timer	実装版数	3.0 以降
void notify_timer(HALEVENTTIMER_T *eventTimer)			
イベントタイマで発生したタイムアウトイベントをアプリケーションに通知する。			
パラメータ	型	名前	説明
	HALEVENTTIMER_T	*eventTimer	イベントタイマ(IN パラメータ)
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

7.5 共通 API

■デバイス作成

表 7-11 デバイス作成 API

関数名	HalCreate	実装版数	3.1 以降
HALCOMPONENT_T HalCreate(int32_t vendorID,int32_t productID,int32_t instanceID)			
OpenEL デバイスの作成を行う。			
パラメータ	型	名前	説明
	int32_t	vendorID	ベンダーID
	int32_t	productID	プロダクト ID
	int32_t	instanceID	インスタンス ID
戻り値	HALCOMPONENT_T*	OpenEL コンポーネント	

■デバイス消去

表 7-12 デバイス消去 API

関数名	HalDestroy	実装版数	3.1 以降
void HalDestroy(HALCOMPONENT_T *halComponent)			
OpenEL デバイスの消去を行う。			
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント（IN/OUT パラメータ）
戻り値		なし	

■デバイス初期化

表 7-13 デバイス初期化 API

関数名	HalInit	実装版数	3.0 以降
HALRETURNCODE_T HalInit(HALCOMPONENT_T *halComponent)			
OpenEL デバイスの初期化を行う。			
コンポーネントテーブル名		pFuncInit	
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント（IN/OUT パラメータ）
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

■デバイス再初期化

表 7-14 デバイス終了 API

関数名	HalReInit	実装版数	3.0 以降
HALRETURNCODE_T HalReInit(HALCOMPONENT_T *halComponent)			
OpenEL デバイスの再初期化を行う。			
コンポーネントテーブル名		pFuncReInit	
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント（IN/OUT パラメータ）
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

■デバイス終了

表 7-15 デバイス終了 API

関数名	HalFinalize	実装版数	3.0 以降
HALRETURNCODE_T HalFinalize(HALCOMPONENT_T *halComponent)			
OpenEL デバイスを終了させる。 再度このデバイスを使用するためにはもう一度初期化を行う必要がある。			
コンポーネントテーブル名		pFuncFinalize	
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント（IN/OUT パラメータ）
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

■HAL オブザーバ登録

表 7-16 HAL オブザーバ登録 API

関数名	HalAddObserver	実装版数	3.0 以降
HALRETURNCODE_T HalAddObserver(HALCOMPONENT_T *halComponent, HALOBSERVER_T *halObserver)			
イベント発生時に呼び出される HalObserver を追加する。			
コンポーネントテーブル名		pFuncAddObserver	
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント
	HALOBSERVER_T	*halObserver	HAL オブザーバ（OUT パラメータ）
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

■HAL オブザーバ削除

表 7-17 HAL オブザーバ削除 API

関数名	HalRemoveObserver	実装版数	3.0 以降
HALRETURNCODE_T HalRemoveObserver(HALCOMPONENT_T *halComponent, HALOBSERVER_T *halObserver)			
イベント発生時に呼び出される HalObserver を削除する。			
コンポーネントテーブル名		pFuncRemoveObserver	
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント
	HALOBSERVER_T	*halObserver	HAL オブザーバ（OUT パラメータ）
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

■プロパティ情報取得

表 7-18 プロパティ情報取得 API

関数名	HalGetProperty	実装版数	3.0 以降
HALRETURNCODE_T HalGetProperty(HALCOMPONENT_T *halComponent, HALPROPERTY_T *property)			
プロパティ情報を取得する。			
コンポーネントテーブル名		pFuncGetProperty	
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント
	HALPROPERTY_T	*property	プロパティ(IN パラメータ)
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

■時間取得

表 7-19 時間取得 API

関数名	HalGetTime	実装版数	3.0 以降
HALRETURNCODE_T HalGetTime(HALCOMPONENT_T *halComponent, int32_t *timeValue)			
時間情報を取得する。			
コンポーネントテーブル名		pFuncGetTime	
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント
	int32_t	*timeValue	時間情報(IN パラメータ)
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

7.6 モータデバイス

■位置・速度・トルク制御と値の取得

表 7-20 HalActuatorSetValue API

関数名	HalActuatorSetValue	実装版数	3.1 以降
HALRETURNCODE_T HalActuatorSetValue(HALCOMPONENT_T *halComponent, int32_t request, HALFLOAT_T value)			
目標角度/位置、目標角速度/速度、目標トルクまでモータを動作させる. 本メソッドは非同期であり，モータが目標値に到達するまでは待たない. モータが目標値に到達する前に，再度本メソッドが呼ばれた場合には，目標値が更新される. モータが目標値に到達したことは，HALObserver を使用してアプリケーション側に通知される. ただし，この通知は，最終的な目標値に到達した場合のみ発行される. このため，本メソッドを複数回呼び出し，目標値が更新された場合には，最終的な目標値を設定したメソッドに対応した通知のみ行われる.			
コンポーネントテーブル名		pFncSetPosition	
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント
	int32_t	request	指令コマンド HAL_REQUEST_NO_EXCITE (0) HAL_REQUEST_POSITION_CONTROL (1) HAL_REQUEST_VELOCITY_CONTROL (2) HAL_REQUEST_TORQUE_CONTROL (3) 予約 (4)
	HALFLOAT_T	value	目標位置 単位[rad] または [m](製品仕様) 目標速度 単位[rad/s] または [m/s] (製品仕様) 目標トルク 単位[N・m] または [N] (製品仕様)
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

表 7-21 HalActuatorGetValue API

関数名	HalMotorGetPosition	実装版数	3.1 以降
HALRETURNCODE_T HalMotorGetPosition(HALCOMPONENT_T *halComponent, HALFLOAT_T *pPosition)			
対象モータの現在角度/位置、現在角速度/速度、現在トルク/力を取得する. 現在値が測定できない要素の場合には，推定値もしくは指令値を返す.			
コンポーネントテーブル名		pFncGetPosition	
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント
	int32_t	request	HAL_REQUEST_POSITION_ACTUAL (5) HAL_REQUEST_VELOCITY_ACTUAL (6) HAL_REQUEST_TORQUE_ACTUAL (7)
	HALFLOAT_T	*value	現在位置 単位[rad] または [m](製品仕様) 現在速度 単位[rad/s] または [m/s] (製品仕様) 現在トルク 単位[N・m] または [N] (製品仕様)
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

上記で定義した API を使用して，複数モータを同期制御する場合のサンプルシーケンス図を以下に示す.

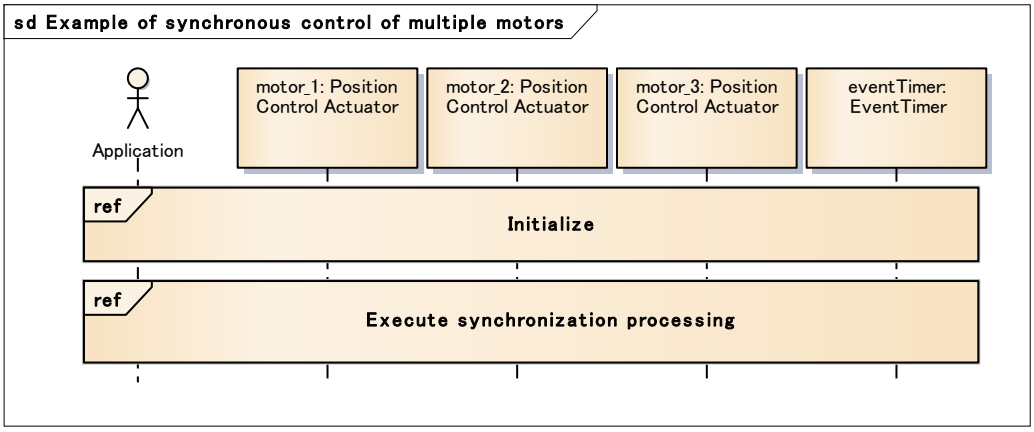

図 7.4 複数モータの同期制御

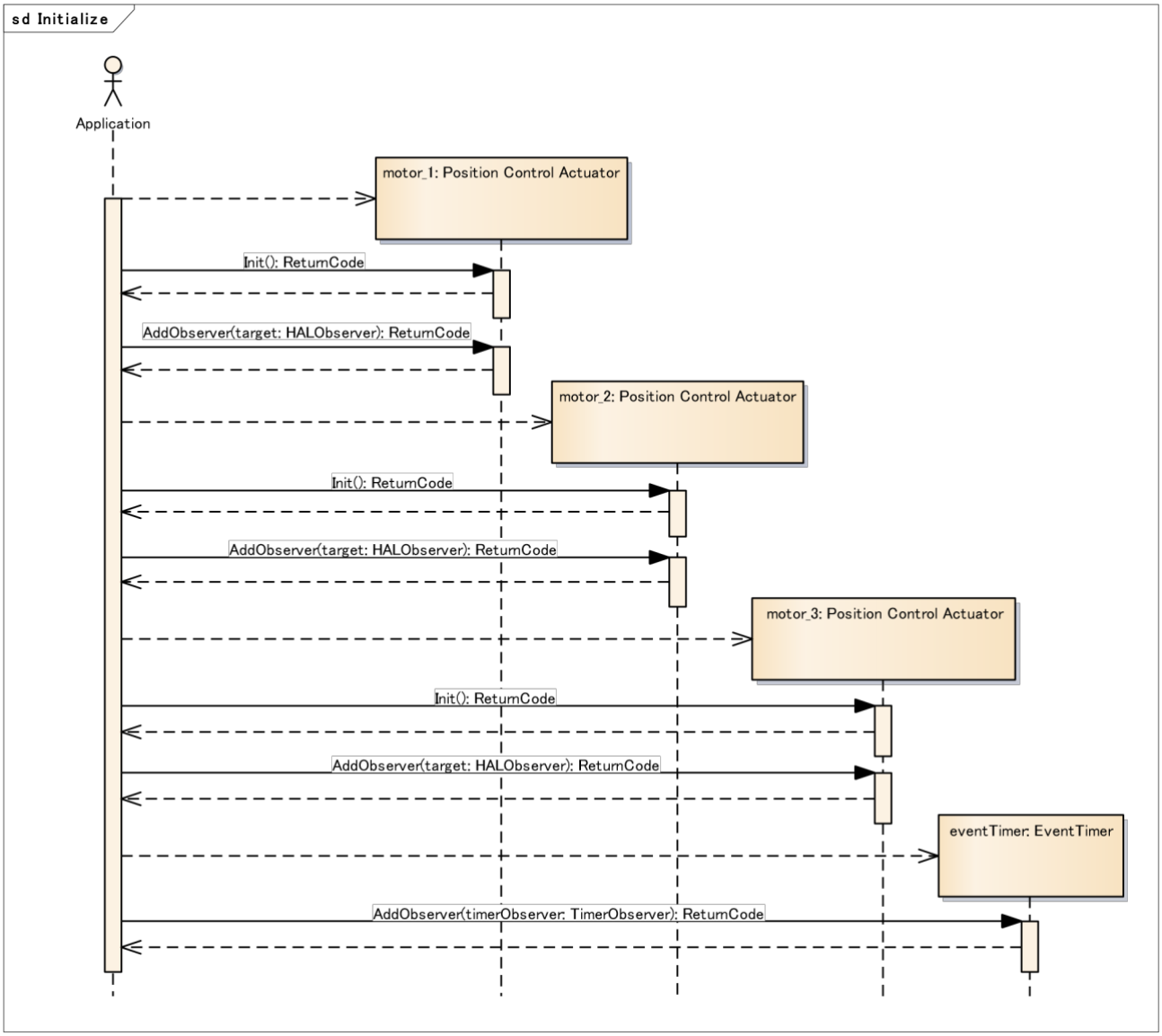

図 7.5 初期化処理を実行する

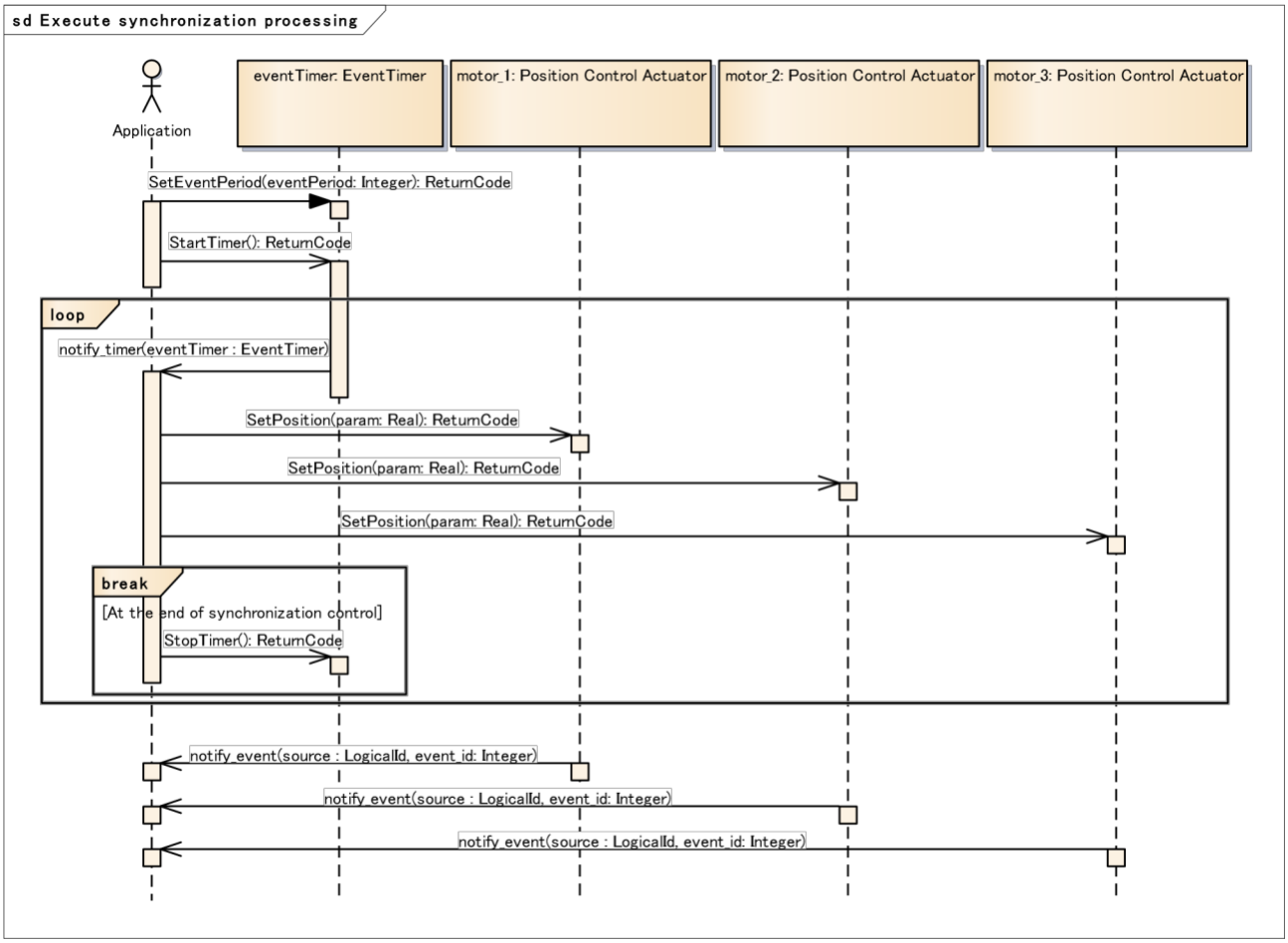

図 7.6 同期制御を実行する

7.7 センサデバイス

■センサ値取得

表 7-22 HalSensorGetValue API

関数名	HalGyroSensorGetValue	実装版数	3.1 以降
HALRETURNCODE_T HalSensorGetValue(HALCOMPONENT_T *halComponent, int32_t *size, HALFLOAT_T *valueList)			
センサーから値を取得する			
コンポーネントテーブル名		pFuncGetValue	
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント（IN/OUT パラメータ）
	int32_t	*size	[out パラメータ]データ数
	HALFLOAT_T	*valuetbl	[out パラメータ]データ実体
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

表 7-23 HalSensorGetTimedValue API

関数名	HalGyroSensorGetTimedValue	実装版数	3.1 以降
HALRETURNCODE_T HalSensorGetTimedValue(HALCOMPONENT_T *halComponent, int32_t *num, HALFLOAT_T *valueList, int32_t *timeValue)			
センサーから値と時間情報を取得する			
コンポーネントテーブル名		pFuncGetTimedValue	
パラメータ	型	名前	説明
	HALCOMPONENT_T	*halComponent	OpenEL コンポーネント（IN/OUT パラメータ）
	int32_t	*size	[out パラメータ]データ数
	HALFLOAT_T	*valueList	[out パラメータ]データ実体
	int32_t	*timeValue	[out パラメータ]時間情報
戻り値	HALRETURNCODE_T	HAL_OK:正常／HAL_ERROR:失敗	

上記で定義した API を使用して，複数センサのデータの同期をとる場合のサンプルシーケンス図を以下に示す.

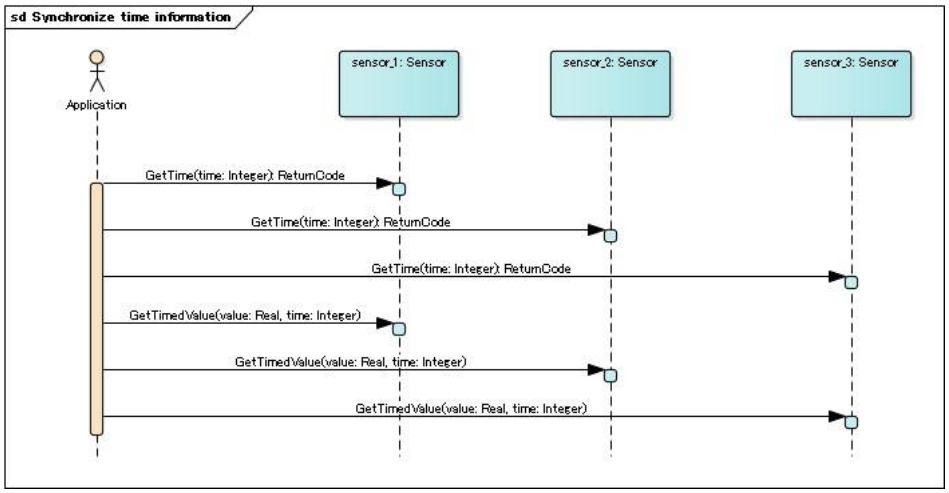

図 7.7 複数センサのデータの同期をとる場合のサンプルシーケンス図

8. API 仕様(C++)

OpenEL 3.2 版にて定義する C++の API を示す。OpenEL 3.2 版に準拠する製品は、該当するデバイス毎に全 API を実装しなければならない。製品の特性上機能しない API（例えば、動作の開始・停止を制御できないセンサの場合など）は、API の戻り値としてエラー（表 7-2 参照）を返却する様に実装すること。

8.1 型定義

型定義は C 言語と同様な定義を使用する。詳細は 7.1 章を参照すること。

ただし、HALRETURNCODE_T の型名を ReturnCode に変更する。

8.2 共通エラー

共通エラーは C 言語と同様なエラーを返却する。詳細は 7.2 章を参照すること。

8.3 例外処理

各関数の実装において、ファイルのオープンやリソースの確保の失敗などが考えられる場合は、適切な例外処理を実装すること。

8.4 イベントタイマ

イベントタイマは、POSIX、組み込みシステム向け RTOS、その他の OS などで実装されているイベントタイマの操作を統一したものである。イベントタイマ API は、EventTimer クラスのメンバ関数（メソッド）として定義する。

■イベントタイマ開始

表 8-1 イベントタイマ開始 API

関数名	StartTimer	実装版数	3.2 以降
ReturnCode StartTimer(void)			
タイマを開始する。			
パラメータ	型	名前	説明
			なし
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

■イベントタイマ終了

表 8-2 イベントタイマ停止 API

関数名	StopTimer	実装版数	3.2 以降
ReturnCode StopTimer(void)			
タイマを停止する。			
パラメータ	型	名前	説明
			なし
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

■イベントタイマ周期設定

表 8-3 イベントタイマ周期設定 API

関数名	SetEventPeriod	実装版数	3.2 以降
ReturnCode SetEventPeriod(int32_t eventPeriod)			
タイマの周期を設定する。			
パラメータ	型	名前	説明
	int32_t	eventPeriod	イベントタイマの周期
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

■オブザーバ登録

表 8-4 オブザーバ登録 API

関数名	AddObserver	実装版数	3.2 以降
ReturnCode AddObserver(TimerObserver *timerObserver)			
タイムアウトイベント発生時に呼び出される TimerObserver を追加する。			
パラメータ	型	名前	説明
	TimerObserver	*timerObserver	タイマオブザーバ（OUT パラメータ）
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

■オブザーバ削除

表 8-5 オブザーバ削除 API

関数名	RemoveObserver	実装版数	3.2 以降
ReturnCode RemoveObserver(TimerObserver *timerObserver)			
タイムアウトイベント発生時に呼び出される TimerObserver を削除する。			
パラメータ	型	名前	説明
	TimerObserver	*timerObserver	タイマオブザーバ（OUT パラメータ）
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

使用例：

```
  EventTimer *eventTimer101 = 0;
（中略）
  tmObs101 = new TmObs101();
  tmObs102 = new TmObs102();
  eventTimer101 = new EventTimer();
  eventTimer101->AddObserver(tmObs101);
  eventTimer101->AddObserver(tmObs102);
  eventTimer101->SetEventPeriod(10000); // 10 seconds
  eventTimer101->StartTimer();
（中略）
  eventTimer101->StopTimer();
  eventTimer101->RemoveObserver(tmObs101);
```

8.5 オブザーバ

8.5.1 HALObserver

HALObserver は、OpenEL コンポーネントで発生したイベントをアプリケーションに通知するためのインターフェースである。HALObserver クラスとして実装される。

表 8-6 イベント通知 API

関数名	notify_event	実装版数	3.2 以降
void notify_event(HalComponent *halComponent, int32_t event_id)			
OpenEL コンポーネントで発生したイベントをアプリケーションに通知する。			
パラメータ	型	名前	説明
	HalComponent	*halComponent	OpenEL コンポーネント
	uint32_t	event_id	イベント ID (IN パラメータ)
戻り値		なし	

イベント ID	意味
0	予約
1	Completed：動作完了
2 以降	コンポーネント依存

表 8-7 エラー通知 API

関数名	notify_error	実装版数	3.2 以降
void notify_error(HalComponent *halComponent, int32_t error_id)			
OpenEL コンポーネントで発生したエラーをアプリケーションに通知する。			
パラメータ	型	名前	説明
	HalComponent	*halComponent	OpenEL コンポーネント
	uint32_t	error_id	エラーID (IN パラメータ)
戻り値		なし	

エラーID	意味
0	予約
1 以降	コンポーネント依存

使用例：

```
Sensor *IMU = 0;

class HalObs101 : public HALObserver
{
   void notify_error(int error_id);
   void notify_event(int event_id);
};

void HalObs101::notify_error(int error_id)
{
   Serial.print("cbNotifyError101:");Serial.println(error_id);
}

void HalObs101::notify_event(int event_id)
{
   Serial.print("cbNotifyEvent101:");Serial.println(event_id);
}

HALId halid;
halid.deviceKindId = 2; // Sensor
halid.vendorId = 3; // M5
halid.productId = 4; // M5Stack Fire
halid.instanceId = 1;

IMU = new Sensor(halid);
HalObs101 *halObs101 = 0;
halObs101 = new HalObs101();
IMU->AddObserver(halObs101);
```

8.5.2 TimerObserver

TimerObserver は、イベントタイマで発生したタイムアウトイベントをアプリケーションに通知するためのインターフェースである。

表 8-8 タイムアウトイベント通知 API

関数名	notify_timer	実装版数	3.0 以降
void notify_timer(EventTimer *eventTimer)			
イベントタイマで発生したタイムアウトイベントをアプリケーションに通知する。			
パラメータ	型	名前	説明
	EventTimer	*eventTimer	イベントタイマ(IN パラメータ)
戻り値		なし	

8.6 共通 API

共通 API は、HalComponent クラスのメンバ関数（メソッド）として定義する。ただし、HalCreate(), HalDestroy()は HalComponent クラスから独立した関数として定義する。

■デバイス作成

表 8-9 デバイス作成 API

関数名	HalCreate	実装版数	3.2 以降
HALCOMPONENT * HalCreate(int32_t vendorID,int32_t productID,int32_t instanceID)			
OpenEL デバイスの作成を行う。			
パラメータ	型	名前	説明
	int32_t	vendorID	ベンダーID
	int32_t	productID	プロダクト ID
	int32_t	instanceID	インスタンス ID
戻り値	HALComponent	OpenEL コンポーネント	

■デバイス消去

表 8-10 デバイス消去 API

関数名	HalDestroy	実装版数	3.2 以降
void HalDestroy(HALComponent *halComponent)			
OpenEL デバイスの消去を行う。			
パラメータ	型	名前	説明
	HALComponent	*halComponent	OpenEL コンポーネント（IN/OUT パラメータ）
戻り値		なし	

■デバイス初期化

表 8-11 デバイス初期化 API

関数名	Init	実装版数	3.2 以降
ReturnCode Init(void)			
OpenEL デバイスの初期化を行う。			
コンポーネントテーブル名		pFuncInit	
パラメータ	型	名前	説明
			なし
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

■デバイス再初期化

表 8-12 デバイス終了 API

関数名	ReInit	実装版数	3.2 以降
ReturnCode ReInit(void)			
OpenEL デバイスの再初期化を行う。			
コンポーネントテーブル名		pFuncReInit	
パラメータ	型	名前	説明
			なし
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

■デバイス終了

表 8-13 デバイス終了 API

関数名	Finalize	実装版数	3.2 以降
ReturnCode Finalize(void)			
OpenEL デバイスを終了させる。 再度このデバイスを使用するためにはもう一度初期化を行う必要がある。			
コンポーネントテーブル名		pFuncFinalize	
パラメータ	型	名前	説明
			なし
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

■HAL オブザーバ登録

表 8-14 HAL オブザーバ登録 API

関数名	AddObserver	実装版数	3.2 以降
ReturnCode AddObserver(HalObserver *halObserver)			
イベント発生時に呼び出される HalObserver を追加する。			
パラメータ	型	名前	説明
	HalObserver	*halObserver	HAL オブザーバ（OUT パラメータ）
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗／HAL_ERROR:失敗	

■HAL オブザーバ削除

表 8-15 HAL オブザーバ削除 API

関数名	RemoveObserver	実装版数	3.2 以降
ReturnCode RemoveObserver(HalObserver *halObserver)			
イベント発生時に呼び出される HalObserver を削除する。			
パラメータ	型	名前	説明
	HalObserver	*halObserver	HAL オブザーバ（OUT パラメータ）
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

■プロパティ情報取得

表 8-16 プロパティ情報取得 API

関数名	GetProperty	実装版数	3.2 以降
ReturnCode HalGetProperty(Property *property)			
プロパティ情報を取得する。			
コンポーネントテーブル名		pFuncGetProperty	
パラメータ	型	名前	説明
	Property	*property	プロパティ(IN パラメータ)
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

■時間取得

表 8-17 時間取得 API

関数名	GetTime	実装版数	3.2 以降
ReturnCode GetTime(unsigned int *time)			
時間情報を取得する。			
コンポーネントテーブル名		pFuncGetTime	
パラメータ	型	名前	説明
	unsigned int	*time	時間情報(IN パラメータ)
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

8.7 モータデバイス

モータデバイス API は、Actuator クラスのメンバ関数（メソッド）として定義する。

■位置・速度・トルク制御と値の取得

表 8-18 SetValue API

関数名	SetValue	実装版数	3.2 以降
ReturnCode SetValue(int32_t request, HALFLOAT_T value)			
目標角度/位置、目標角速度/速度、目標トルクまでモータを動作させる. 本メソッドは非同期であり，モータが目標値に到達するまでは待たない. モータが目標値に到達する前に，再度本メソッドが呼ばれた場合には，目標値が更新される. モータが目標値に到達したことは，HALObserver を使用してアプリケーション側に通知される．ただし，この通知は，最終的な目標値に到達した場合のみ発行される．このため，本メソッドを複数回呼び出し，目標値が更新された場合には，最終的な目標値を設定したメソッドに対応した通知のみ行われる.			
コンポーネントテーブル名		pFncSetPosition	
パラメータ	型	名前	説明
	int32_t	request	指令コマンド HAL_REQUEST_NO_EXCITE (0) HAL_REQUEST_POSITION_CONTROL (1) HAL_REQUEST_VELOCITY_CONTROL (2) HAL_REQUEST_TORQUE_CONTROL (3) 予約 (4)
	HALFLOAT_T	value	目標位置 単位[rad] または [m](製品仕様) 目標速度 単位[rad/s] または [m/s]（製品仕様） 目標トルク 単位[N・m] または [N]（製品仕様）
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

表 8-19 GetValue API

関数名	GetPosition	実装版数	3.2 以降
ReturnCode GetPosition(HALFLOAT_T *pPosition)			
対象モータの現在角度/位置、現在角速度/速度、現在トルク/力を取得する. 現在値が測定できない要素の場合には，推定値もしくは指令値を返す.			
コンポーネントテーブル名		pFncGetPosition	
パラメータ	型	名前	説明
	int32_t	request	HAL_REQUEST_POSITION_ACTUAL (5) HAL_REQUEST_VELOCITY_ACTUAL (6) HAL_REQUEST_TORQUE_ACTUAL (7)
	HALFLOAT_T	*value	現在位置 単位[rad] または [m](製品仕様) 現在速度 単位[rad/s] または [m/s] (製品仕様) 現在トルク 単位[N・m] または [N] (製品仕様)
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

8.8 センサデバイス

センサデバイス API は、Sensor クラスのメンバ関数（メソッド）として定義する。

■センサ値取得

表 8-15 GetValueList API

関数名	GetValueList	実装版数	3.2 以降
ReturnCode GetValueList(float *valueList, int *num)			
センサーから値を取得する			
コンポーネントテーブル名		pFuncGetValueList	
パラメータ	型	名前	説明
	float	*valueList	[out パラメータ]データ実体
	int	*num	[out パラメータ]データ数
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

■センサ値・時間情報取得

表 8-16 GetTimedValueList API

関数名	GetTimedValueList	実装版数	3.2 以降
ReturnCode GetTimedValueList(float *valuelist, unsigned int *time, int *num)			
センサーから値と時間情報を取得する			
コンポーネントテーブル名		pFuncGetTimedValueList	
パラメータ	型	名前	説明
	float	*valueList	[out パラメータ]データ実体
	unsigned int	*time	[out パラメータ]時間情報
	int	*num	[out パラメータ]データ数
戻り値	ReturnCode	HAL_OK:正常／HAL_ERROR:失敗	

9. API 仕様(C#)

OpenEL 3.2 版にて定義する C#の API を示す。OpenEL 3.2 版に準拠する製品は、該当するデバイス毎に全 API を実装しなければならない。製品の特性上機能しない API(例えば、動作の開始・停止を制御できないセンサの場合など)は、API の戻り値としてエラー(表 7-2 参照)を返却する様に実装すること。

9.1 型定義

型定義は C 言語と同様な定義を使用する。詳細は 7.1 章を参照すること。

9.2 共通エラー

共通エラーは C 言語と同様なエラーを返却する。詳細は 7.2 章を参照すること。

9.3 例外処理

C++言語と同様。詳細は 8.3章を参照すること。

9.4 イベントタイマ

C++言語と同様。詳細は 8.4 章を参照すること。

9.5 オブザーバ

C++言語と同様。詳細は 8.5 章を参照すること。

9.6 共通 API

C++言語と同様。詳細は 8.6 章を参照すること。

9.7 モータデバイス

C++言語と同様。詳細は 8.7 章を参照すること。

9.8 センサデバイス

C++言語と同様。詳細は 8.8 章を参照すること。

10. お問合せ

OpenEL に関するお問い合わせは、下記までお願いいたします。

一般社団法人 組込みシステム技術協会
本部事務局 OpenEL 担当 宛
E-mail：jasainfo@jasa.or.jp
TEL：03-5643-0211
FAX：03-5643-0212

「OpenEL 3.2 仕様書 01 版」
2020 年 12 月 25 日発行
発行者　一般社団法人 組込みシステム技術協会
東京都中央区日本橋大伝馬町 6-7 住長第 2 ビル 3 階
TEL：03-5643-0211
FAX：03-5643-0212
URL：http://www.jasa.or.jp

No	Device Kind ID	Device name	Description	Applicant	Application date	Approval date	Authorizer	Remarks
0	0x00000000	Test	For development use	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
1	0x00000001	Motor	Motor	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
2	0x00000002	GyroSensor	Gyroscope Sensor	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
3	0x00000003	TorqueSensor	Torque sensor	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
4	0x00000004	AccelerationSensor	Acceleration Sensor	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
5	0x00000005	CompassSensor	Compass Sensor	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
6	0x00000006	DistanceSensor	Distance Sensor	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
7	0x00000007	ForceSensor	Force Sensor	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
8	0x00000008	TemperatureSensor	Temperature Sensor	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
9	0x00000009	HumiditySensor	Humidity Sensor	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
10	0x0000000A	CO2Sensor	CO2 Sensor	UTI	2021/2/1	2021/3/29	JASA	JASA OpenEL WG
11	0x0000000B	ColorSensor	Color Sensor	UTI	2021/4/19	2021/4/19	JASA	JASA OpenEL WG
12	0x0000000C	TouchSensor	Touch Sensor	UTI	2021/4/19	2021/4/19	JASA	JASA OpenEL WG
13	0x0000000D							
14	0x0000000E							
15	0x0000000F							
	0xFFFFFFFF	Test	For development use	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee

No	Vendor ID	Vendor Short Name	Vendor Name	Applicant	Application date	Approval date	Authorizer	Remarks
0	0x00000000	TEST	For development use	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
1	0x00000001	JASA	Japan Embedded Systems Technology Association	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
2	0x00000002	UTI	Upwind Technology, Inc.	UTI	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
3	0x00000003	OM	ORIENTAL MOTOR Co., Ltd.	OM	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
4	0x00000004	NIDEC	NIDEC CORPORATION	NIDEC	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
5	0x00000005	RT	RT Corporation	UTI	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
6	0x00000006	AG	ASAKUSAGIKEN	UTI	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee
7	0x00000007	SEN	Sensirion	UTI	2021/2/1	2021/3/29	JASA	JASA OpenEL WG
8	0x00000008	DIA	Diarkis	DIA	2021/2/1	2021/3/29	JASA	JASA OpenEL WG
9	0x00000009	LEGO	LEGO	UTI	2021/4/19	2021/4/19	JASA	JASA OpenEL WG
10	0x0000000A	M5	M5Stack	UTI	2021/4/19	2021/4/19	JASA	JASA OpenEL WG
11	0x0000000B							
12	0x0000000C							
13	0x0000000D							
14	0x0000000E							
15	0x0000000F							
	0xFFFFFFFF	TEST	For development use	JASA	2018/5/7	2018/5/7	JASA	JASA OpenEL International Standardization Committee

Vendor ID	Vendor Short Name	Vendor Name	Product ID	Product Name
0x00000000	TEST	For development use		
0x00000001	JASA	Japan Embedded Systems Technology Association		
0x00000002	UTI	Upwind Technology, Inc.	0x00000001	UTRX-17
0x00000003	OM	ORIENTAL MOTOR Co., Ltd.		
0x00000004	NIDEC	NIDEC CORPORATION		
0x00000005	RT	RT Corporation	0x00000001	USB-9AXIS
0x00000006	AG	ASAKUSAGIKEN	0x00000001	AGB65-DCM
0x00000007	SEN	Sensirion	0x00000001	SCD30
0x00000008	DIA	Diarkis	0x00000001	Diarkis
0x00000009	LEGO	LEGO	0x00000001	EV3
0x0000000A	M5	M5Stack	0x00000001	BALA2
0x0000000B				
0x0000000C				
0x0000000D				
0x0000000E				
0x0000000F				
0xFFFFFFFF	TEST	For development use		