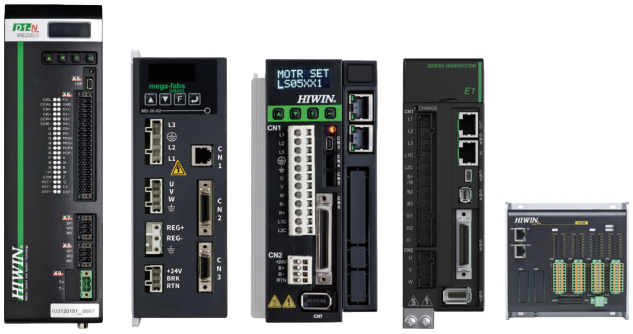
HIWIN® MIKROSYSTEM



API ライブラリ

リファレンスマニュアル

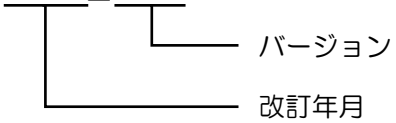
www.hiwinmikro.tw

MD23UJ01-2501_V1.9

改訂履歴

マニュアルのバージョンは表紙の下部にも記載されています。

MD23UJ01-2501_V1.9



日付	バージョン	DLL バージョン	改訂内容
2025 年 1 月	1.9	2.2.0.0	(1) 関連ドキュメントの追加。 (2) セクション 1.1 このマニュアルについて を更新。 (3) セクション 1.3.1 使用可能なスレーブを更新。 (4) セクション 1.4 デバイス通信設定を更新。 (5) セクション 2.1 MPI_CreateAxis を更新。 (6) セクション 3.1 MPI_ConnectMegaulink を更新。 (7) セクション 3.2 MPI_Disconnect を更新。 (8) セクション 3.6 MPI_Reconnect を更新。 (9) セクション 3.9 MPI_Connect Mega-ulink By Network Adapter を更新。 (10) セクション 3.10 MPI_ConnectUsbById を追加。 (11) セクション 3.11 MPI_ConnectUsbByInfo を追加。 (12) セクション 3.12 MPI_GetUsbDevice を追加。 (13) セクション 6.13 MPI_SetCallBackStates を更新。 (14) 第 10 章のエラーコード を更新。
2022 年 12 月 9 日	1.8	2.1.7.0	(1) セクション 1.3.2「前提条件」を更新
2022 年 3 月 23 日	1.7	2.1.7.0	(1) セクション 1.3.1 使用可能なスレーブを更新。 (2) セクション 1.3.2 前提条件を更新。 (3) セクション 2.2 MPI_DestroyAxis を更新。 (4) セクション 3.7 MPI_GetNetworkAdapterInfo を追加。 (5) セクション 3.8 MPI_ClearNetworkAdapterInfo を追加。 (6) セクション 3.9 MPI_Connect Mega-ulink By Network Adapter を追加。 (7) セクション 5.1 MPI_ServoOn を更新。 (8) セクション 5.3 MPI_MoveAbsolute を更新。 (9) セクション 5.4 MPI_MoveRelative を更新。 (10) セクション 5.11 MPI_ClearError を更新。

日付	バージョン	DLL バージョン	改訂内容
			(11) セクション 6.12 MPI_WaitMoveOver を更新。 (12) セクション 8.1 MPI_SetTriggerPosition を更新。 (13) セクション 8.2 MPI_SetTriggerInterval を更新。 (14) セクション 8.5 MPI_SetTriggerPositionArray を追加。 (15) セクション 8.6 MPI_Set Trigger Position State Array を追加。 (16) セクション 8.7 MPI_Set Trigger Position Start Index を追加。 (17) セクション 8.8 MPI_Set Trigger Position End Index を追加。 (18) セクション 9.5 MPI_Get Error Data を更新。 (19) セクション 9.6 MPI_Get Error Data Array を更新。 (20) セクション 9.7 MPI_Run Func PDL を更新。 (21) セクション 9.8 MPI_Is Task Running を更新。 (22) セクション 9.9 MPI_Kill Task を更新。 (23) セクション 9.11 MPI_Get Var Array を追加。 (24) セクション 9.12 MPI_Set Var Array を追加。 (25) セクション 9.13 MPI_Get Error Message を追加。 (26) 第 10 章 エラー コードを更新。 (27) 第 12 章 D シリーズドライバーエラーを追加。
2021 年 4 月 28 日	1.6	2.1.3.0	(1) セクション 3.6 MPI_Reconnect を追加。 (2) セクション 9.10 MPI_Send RAM to Flash を追加。 (3) 第 10 章 エラー コードを更新。 (4) 第 11 章 E1 シリーズドライバーアラームを更新。
2020 年 10 月 30 日	1.5	2.1.2.0	(1) セクション 1.2 mega-ulink 仕様を更新。 (2) セクション 1.3.2 前提条件を更新。 (3) セクション 2.1 MPI_Create Axis を更新。 (4) セクション 3.5 MPI_Set Communication Var を追加。 (5) セクション 6.13 MPI_Set Call Back States を追加。 (6) セクション 9.5 MPI_Get Error Data を追加。 (7) セクション 9.8 MPI_Is Task Running を追加。 (8) セクション 9.9 MPI_Kill Task を追加。 (9) 第 10 章 エラー コードを更新 (10) 第 11 章 E1 シリーズドライバーアラームを更新。
2020 年 5 月 5 日	1.4	2.0	(1) マニュアル名が、D シリーズドライバー用 API ライブラリリファレンスマニュアルからドライバー用 API ライブラリリファレンスマニュアルに変更。

日付	バージョン	DLL バージョン	改訂内容
			API 関数を使用して E1 シリーズドライバーまたは D シリーズドライバーを制御する場合は、このマニュアルを参照してください。 (2) セクション 1.1 このマニュアルについて を更新。 (3) セクション 1.3.1 使用可能なスレーブ を更新。 (4) セクション 1.3.2 前提条件 を更新。 (5) セクション 1.3.3 データベース検証 を更新。 (6) セクション 1.3.5 ドライバー設定 を更新。 (7) セクション 2.1 MPI_Create Axis を更新。 (8) セクション 2.3 MPI_Initial Axis を更新。 (9) セクション 4.17 MPI_Get Sw Limit Pos を更新。 (10) セクション 4.18 MPI_Set Sw Limit Pos を更新。 (11) セクション 5.1 MPI_Servo On を更新。 (12) セクション 5.2 MPI_Servo Off を更新。 (13) セクション 5.5 MPI_Jog Positive を更新。 (14) セクション 5.6 MPI_Jog Negative を更新。 (15) セクション 5.8 MPI_Start Home を更新。 (16) セクション 5.12 MPI_Enable Sw Limit を更新。 (17) セクション 5.13 MPI_Disable Sw Limit を更新。 (18) セクション 6.1 MPI_Is Drive Ready を追加。 (19) セクション 6.7 MPI_Is Sw Limit En を更新。 (20) セクション 6.9 MPI_Is Sw Limit を更新。 (21) セクション 8.1 MPI_Set Trigger Position を更新。 (22) セクション 8.2 MPI_Set Trigger Interval を更新。 (23) セクション 8.3 MPI_Enable Trigger を更新。 (24) セクション 8.4 MPI_Disable Trigger を更新。 (25) セクション 9.5 MPI_Get Error Data Array を追加。 (26) 第 10 章 エラー コードを更新。 (27) 第 11 章 E1 シリーズドライバーアラームを追加。
2020 年 1 月 8 日	1.3	1.17	セクション 1.1 に説明を追加： API ライブラリはフルクローズドループ (デュアル ループ) 機能をサポートしていません。
2018 年 1 月 27 日	1.2	1.17	HIOM をサポート。
2016 年 8 月 19 日	1.1	1.11	(1) いくつかの説明を修正。 (2) 引数の型を修正： MPI_Get Input Status、MPI_Get Output Status、MPI_Get State、および MPI_Set State。 (3) 引数の単位を修正： MPI_Set Trigger Position および MPI_Set Trigger

日付	バージョン	DLL バージョン	改訂内容
			Interval。 (4) 図 3.1.1 および図 3.1.2 を追加。
2016 年 2 月 29 日	1.0	1.09	初版

関連文書

関連ドキュメントを通じて、ユーザーはこのマニュアルの位置付けとマニュアルと製品の相関関係をすぐに理解できます。詳細については、HIWIN MIKROSYSTEM の公式 Web サイト → ダウンロード → マニュアルの概要 (https://www.hiwinmikro.tw/Downloads/ManualOverview_EN.htm) にアクセスしてください。

目次

(このページはブランクになっています)

1. 序文

1.1 このマニュアルについて

このマニュアルは、mega-ulink または USB 通信をサポートする HIWIN E1 シリーズおよび D シリーズドライバーに適用されます。USB 通信に適したハードウェアは、Thunder がサポートするドライバーハードウェアと同じです。mega-ulink は、サブデバイスをローカルコントローラーにチェーンできる EtherCAT バス タイプのハードウェアを使用します。標準コンポーネントを使用した合理化されたハードウェアとソフトウェアにより、ローカル操作またはセットアップでのシステム統合が簡単になります。このようなシステムでは、コンポーネントの可用性と分散型多軸システムが低コストで維持されます。コントローラーは、Microsoft Visual C++、Visual Basic .NET、C# インターフェース/ツール、またはカスタマイズされたアプリケーションを備えたコンピューターにすることができます。

HIWIN は、特定の機能とコマンドを使用して簡素化されたモーター制御を容易にする API (アプリケーション プロトコル インターフェース) ライブラリを提供します。API ライブラリは、プログラミング、通信、およびユーティリティ指向のソフトウェア インターフェースである MPI (megaulink プロトコル インターフェース) ライブラリの拡張です。HIWIN アプリケーション プログラミング ソフトウェアの PDL (プロセス記述言語) と組み合わせることで、ユーザーは効率的なカスタマイズが可能になります。

API ライブラリは、Axis.h、Axis.dll、Axis.lib の 3 つのファイルで構成されています。これは、mpi.lib、mpi.dll、canlib32.dll の 3 つのファイルで構成される MPI ライブラリと組み合わされています。API ライブラリは、Microsoft Visual C++、Visual Basic .NET、または C# プラットフォーム、および Windows XP/7/10 Professional オペレーティング システムに適用できます。

API ライブラリを介して次のことを実現できます。

- Mega-ulink または USB 経由で複数のドライバーと通信します。
- ドライバーのサーボ on/サーボ off。
- ドライバーの動作パラメーター（速度、加速、減速など）を処理します。
- プロセスモーションステータス（例：サーボ準備完了、サーボエラー、移動中など）。
- 絶対座標/相対座標のパラメーターでジョグ動作し、モーターの動作を中止して原点復帰手順を実行してモーターを駆動します。
- 入力/出力の状態を監視し、出力のオン/オフ状態を設定します。

注記：

(1) D シリーズドライバーの場合、API ライブラリはフルクローズドループ（デュアルループ）機能をサポートしていません。

(2) USB 通信を使用する場合は、十分な USB ポートがあることを確認してください。最大 5 つの USB 接続を確立できます。

(3) USB ドライバーは ARM アーキテクチャをサポートしていないため、ARM アーキテクチャ上の Windows で

は接続が正常に行われない可能性があります。

(4) mega-ulink は Windows 11 以降のオペレーティングシステムをサポートしていません。

1.2 mega-ulink 仕様

表 1.2.1

項目	仕様
Physical layer	Ethernet
Baud rate	100 Mbps
Transmission cycle time	> 500 us
Number of slaves	Max. 32
Topology	Bus or Line
Distributed clocks	Not support (Non synchronized)
基本機能	(1) モーターを絶対または相対目標位置まで駆動する (2) 原点復帰手順 (3) ソフトウェア制限保護 (4) 一般/特定入出力
高度な機能	(1) ユーザー定義の変数/状態へのアクセス (2) ユーザー定義の PDL 関数の実行

注記： 送信サイクル時間は、ホストコントローラーのアプリケーション負荷が変化すると変化します。

1.3 HIWIN API ライブラリを使用する前に

1.3.1 利用可能なスレーブ

- mega-ulink
  HIWIN D シリーズドライバーでは、mega-ulink 通信オプションのモデル指定は F です。HIWIN E1 シリーズドライバーでは、mega-ulink 通信オプションのモデル指定は H です。可能な組み合わせは、表 1.3.1.1 に示されています。
  API ライブラリは、E1 シリーズと D シリーズのドライバーの同時制御や、異なる通信の混在使用をサポートしていません。また、E2 シリーズ以上のドライバーもサポートしていません。

表 1.3.1.1

Slave	モデル番号
E1	ED1F-**H**□-□□□□-□□
D1	D1□-□□-**F**□-□-□
D2	D2□-□□□□-**F**-□□
D1-N	D1-N-□□-**F**□-□-□-□□
HIOM (HIWIN mega-ulink IO module)	HIOM-D32-□-□□

- USB
  すべての HIWIN E1 シリーズドライバーには USB 通信インターフェースがあり、最大 5 つの USB サブデバイスとの接続がサポートされています。mega-ulink 通信との同時使用はサポートされていません。

1.3.2 前提条件

HIWIN API ライブラリを使用する前に、次のソフトウェアをインストールしてください。

- Npcap 0.05 r16 (Windows 10 オペレーティング システム用)
  これは、Windows 環境でのリンク層ネットワーク アクセス用の業界標準のパケット キャプチャ ツールです。https://github.com/nmap/npcap/releases からダウンロードし、Npcap 0.05 r16 を選択すると、インストール ファイル npcap-nmap-0.05-r16.exe が Assets からダウンロードできます。Npcap 0.05 r16 のライセンスを参照してください。

- WinPcap (Windows 7 オペレーティング システム用)
  これは、Windows 環境でのリンク層ネットワーク アクセス用の業界標準のパケット キャプチャ ツールです。WinPcap の公式 Web サイトからダウンロードしてください:
  http://www.winpcap.org/install/default.htm

  WinPcap のライセンスを参照してください。

- Lightening
  これは、HIWIN D シリーズドライバー用のヒューマン マシン インターフェースです。

  HIWIN MIKROSYSTEM 公式 Web サイトからダウンロードしてください:
  https://www.hiwinmikro.tw/en/download

- Thunder
  これは、HIWIN E1 シリーズドライバー用のヒューマン マシン インターフェースです。

  HIWIN MIKROSYSTEM 公式 Web サイトからダウンロードしてください:
  https://www.hiwinmikro.tw/en/download

- MFC library
  Visual Studio をインストールするときにインストールするか、Microsoft の公式 Web サイトからダウンロードしてください。

1.3.3 データベースの検証

ドライバーのメインプログラムであるデータベースには、MDP ファイルと PDL ファイルが含まれています。これは、各ドライバーのレジスタと変数間の参照に使用されます。ドライバーのデータベースは、PC のデータベースと同じである必要があります。

ユーザーは、Wizalg プログラム (ソフトウェア インストール パッケージ内のツール) を使用して、ドライバーのデータベースを検証できます。操作手順を以下に示します。(注: E1 シリーズドライバーは、データベースの検証をサポートしていません。)

Step 1. ディレクトリ「C:¥HIWIN¥dce¥toolswin¥winkmi」から Wizalg プログラムを開きます。

Step 2. PC をドライバーに接続します。

Step 3. ファイルで Verify data base… in File を選択して、ファームウェアのウィンドウを開きます。

Step 4. ファームウェア ウィンドウの左側にあるスレーブ ツリーで、すべてのドライバーをクリックします。図 1.3.3.1 に示すように、2 列 3 行のテーブルが表示されます。1 列目はドライバーのファームウェアに関するもので、2 列目は PC の作業ディレクトリ内のファイルに関す

るものです。各 Csum サブ行のチェックサムは同じである必要があります。(最初の Csum サブ行 (Boot Dsp Program) は適用されません。)

Step 5. 青いテキストはチェックサムが一致していることを示します (図 1.3.3.1 を参照)。一方、赤いテキストはチェックサムが一致していないことを示します (図 1.3.3.2 を参照)。チェックサムが一致しない場合は、セクション 1.3.4 を参照してデータベースを更新してください。

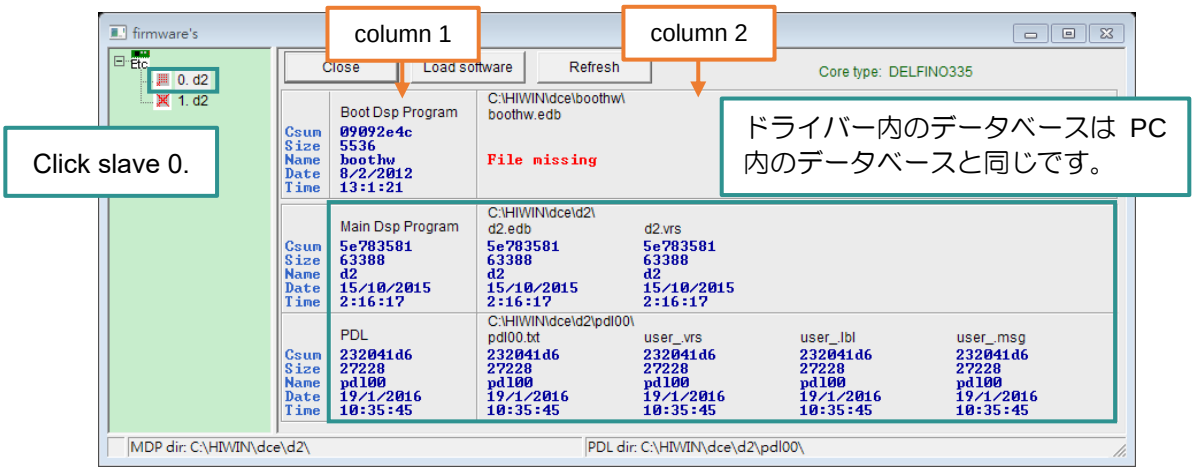


図 1.3.3.1 チェックサムの一致

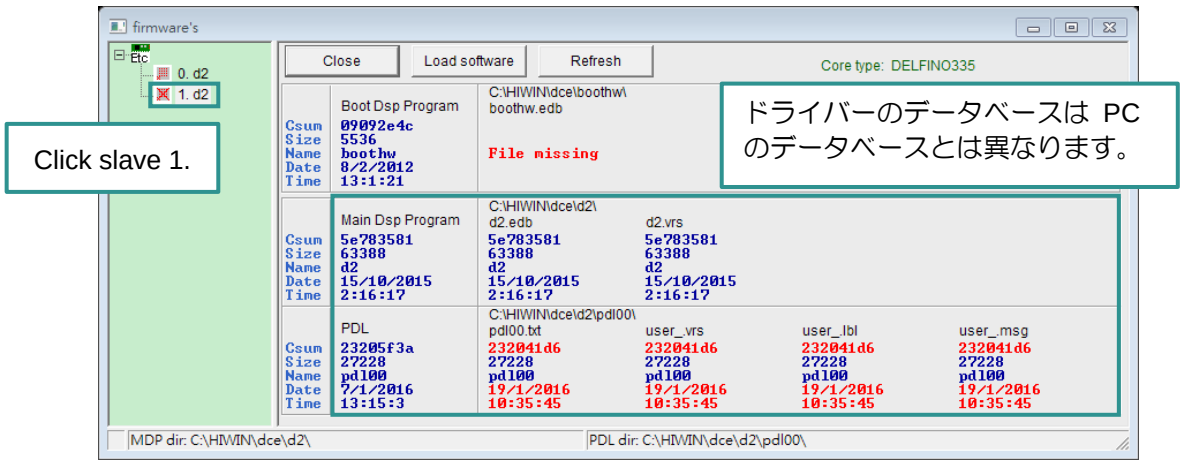


図 1.3.3.2 チェックサムが一致しない

1.3.4 データベースの更新

ユーザーは、Wizalg プログラムを利用して MDP ファイルと PDL ファイルを更新できます。操作手順を以下に示します。(注: E1 シリーズドライバーはこの機能をサポートしていません。複数のドライバーが接続されている場合、それらのファームウェア バージョンは同じである必要があります。)

Step 1. ディレクトリ C:¥HIWIN¥dce¥toolswin¥winkmi から Wizalg プログラムを開きます。

Step 2. PC をドライバーに接続します。

Step 3. ブートで Auto load programs… を選択して、Auto load programs ウィンドウを開きます。

Step 4. Compile PDL のチェックを外し、Run ボタンをクリックします。

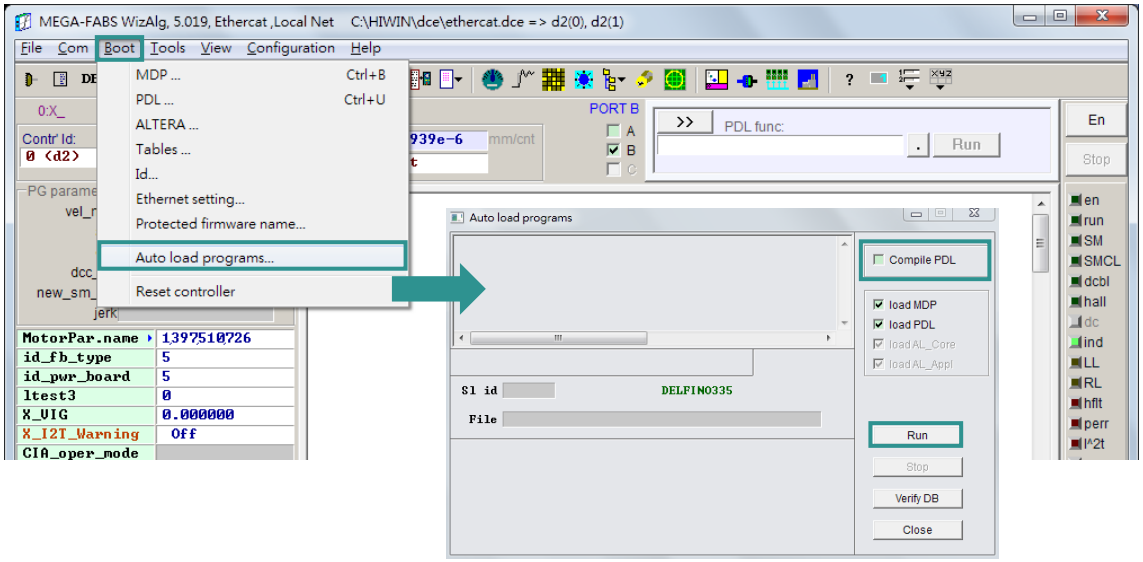


図 1.3.4.1

1.3.5 ドライバー設定

API (Axis.dll) を実装する前に、D シリーズドライバーは構成、チューニング、および動作の最適化を完了する必要があります。ドライバーが上記の設定を完了すると、Lightning の HMI パネルは図 1.3.5.1 のようになります。ステータス インジケータ「Software enabled」が緑色に点灯し、ドライバーの軸名は一意であるだけでなく、関数 MPI_CreateAxis の軸名と同じである必要があります。

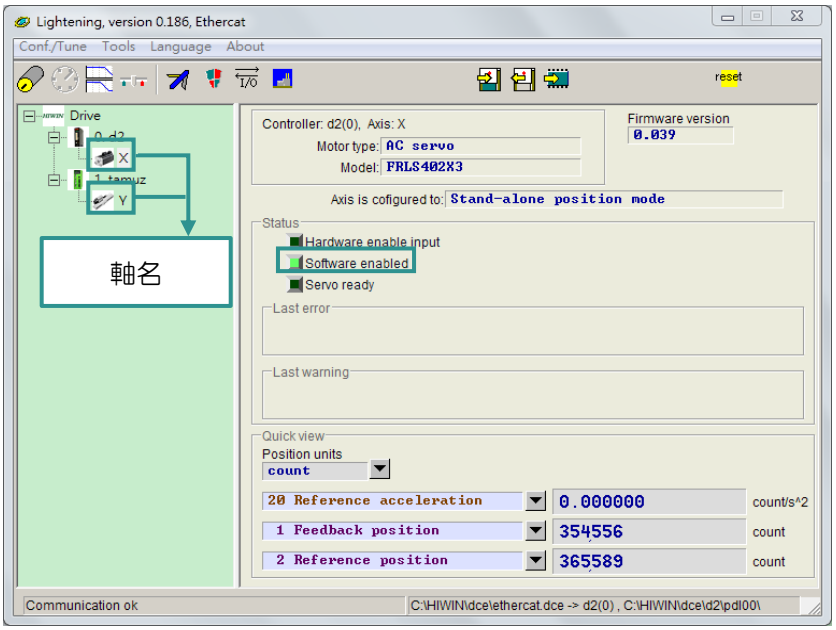


図 1.3.5.1

E1 シリーズドライバーは、構成、チューニング、および動作の最適化を完了する必要があります。ドライバーが上記の設定を完了すると、Thunder の HMI パネルは図 1.3.5.2 のようになります。ステータス インジケータ Drive ready が緑色に点灯します。

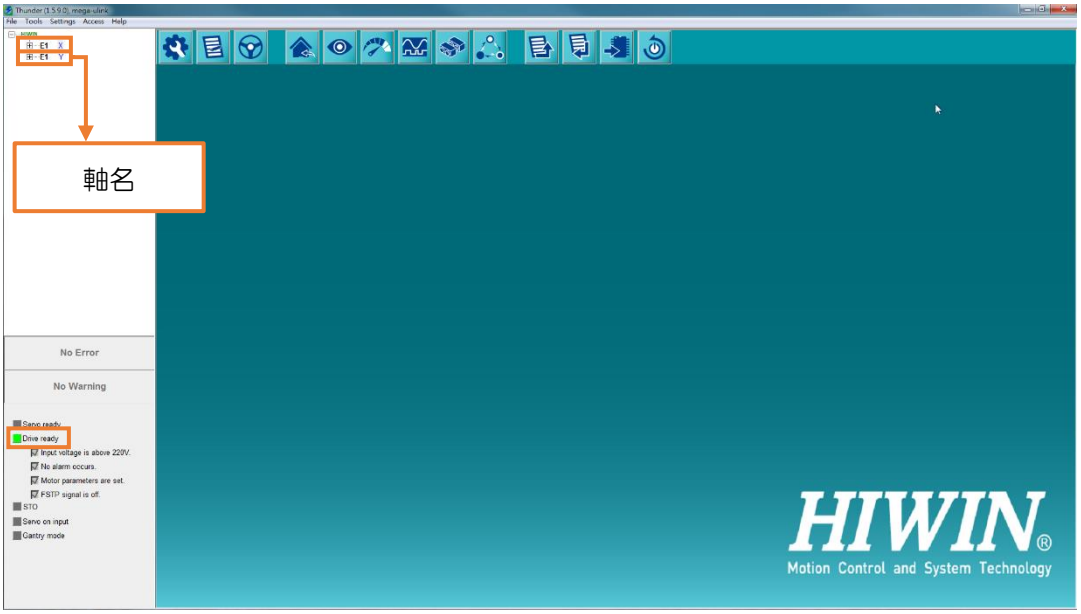


図 1.3.5.2

E1 シリーズドライバーの場合、最初に表示単位の設定を完了してください。対応する制御単位を 1 回転または 1 mm に設定します。

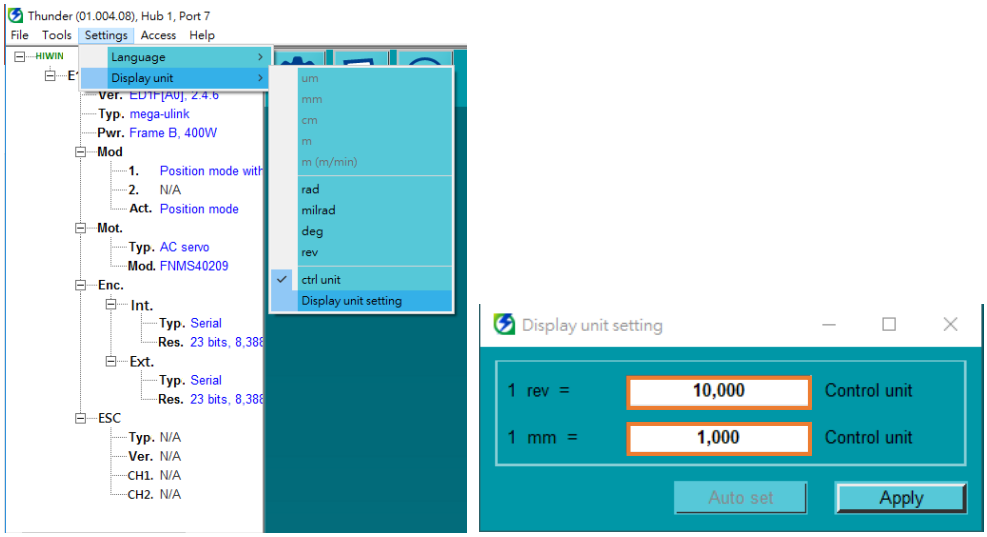


図 1.3.5.3

■ E1 シリーズドライバーの注意事項

1. 各パラメーターの設定範囲に注意してください。
2. API 関数によってドライバーにパラメーターを書き込むときに、ドライバーの単位がパラメーターの単位と異なる場合、API 関数は単位変換を実行します。パラメーター値は、ドライバーに書き込まれる前に丸められます。

1.4 デバイス通信設定

次の図に示すように、PC で使用するデバイスを手動で選択します。

mega-ulink 通信の場合、最初に関数 MPI_ConnectMegaulink(blNetWorkSetting=true) を呼び出します。次に、SetUp… ボタンを押して wizalg Communication Setup ウィンドウを開き、EtherCat… ボタンを押して目的のネットワークを選択します。その後、Apply ボタンを押して設定を保存します。ユーザーが EtherCAT mega-ulink に接続するたびに設定を復元できます。

USB 通信の場合は、関数 MPI_ConnectUsbById(*pAxis, blNetWorkSetting=true, nId) を呼び出し、SetUp… ボタンを押して、USB(Dragonfly) を選択します。

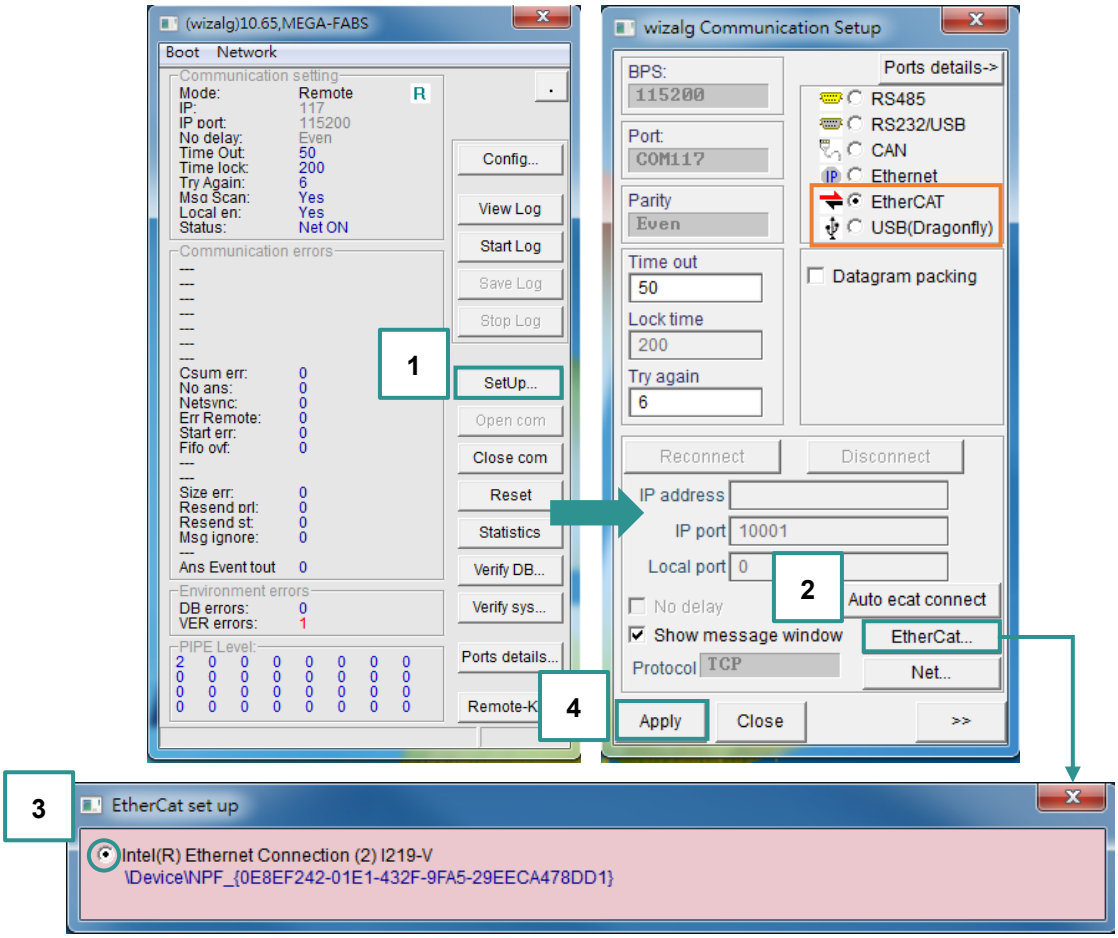


図 1.4.1

2. 初期化

2.1 MPI_CreateAxis

目的

スレーブのオブジェクト ポインターを作成します。複数のスレーブを mega-ulink 経由で制御する場合、ユーザーはそれらすべてに対して複数のオブジェクト ポインターを作成する必要があります。

プロトタイプ

```
CAxis* MPI_CreateAxis(char *pszAxisName, ENUM_UNIT_TYPE enumUnitType,
                      ENUM_DRIVE_TYPE enumDriveType,
                      ENUM_ROOT_FOLDER enumRootFolder=eC_HIWIN);
```

パラメーター

*pszAxisName	軸名
enumUnitType	ユニットタイプ eUT_ROTARY: 回転ユニットタイプ（単位：deg、deg/sec、deg/sec^2） eUT_LINEAR: リニアユニットタイプ（単位：mm、mm/sec、mm/sec^2）
enumDriveType	スレーブタイプ eDT_D1: D1 ドライバー　eDT_D1N: D1-N ドライバー eDT_D2: D2 ドライバー　eDT_HIOM: HIOM eDT_E1: E1 ドライバー eDT_E2: E2 ドライバー (USB 通信のみサポート)
enumRootFolder	ソフトウェアインストールのルートフォルダ eC_HIWIN: C:¥HIWIN; Thunder バージョン 1.5.10.0 未満の D シリーズドライバーおよび E1 シリーズドライバーで使用可能。 eC_Thunder: C:¥Thunder; Thunder バージョン 1.5.10.0 (付属) 以上の E1 シリーズドライバーで使用できます。

Return

CAxis クラス オブジェクトのポインター

備考

(1) この機能は HIOM でも利用可能です。
(2) 軸名はスレーブと同じである必要があります。軸名に応じてスレーブタイプも選択する必要があります。そうしないと、MPI_ConnectMegaulink が呼び出されたときにエラーが発生します。

2.2 MPI_DestroyAxis

目的

スレーブのオブジェクト ポインターを破棄します。

プロトタイプ

```
void MPI_DestroyAxis(CAxis *pAxis);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター

Return

N/A

備考

(1) この機能は HIOM でも利用可能です。

(2) アプリケーションウィンドウが閉じられる前にこの関数を呼び出します。

2.3 MPI_InitialAxis

目的

ドライバーのモーションパラメーターを初期化します。

プロトタイプ

```
int MPI_InitialAxis(CAxis *pAxis, double dVel, double dAcc, double dDec,
                    double dDecKill, int nSmoothTime, double dInPosWidth,
                    double dPosLimitPos, double dNegLimitPos);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター
dVel	最大速度（単位：mm/sec または deg/sec）
dAcc	加速度 (単位: mm/sec^2 または deg/sec^2)
dDec	減速度 (単位: mm/sec^2 または deg/sec^2)
dDecKill	緊急停止減速 (単位: mm/sec^2 または deg/sec^2)
nSmoothTime	スムーズタイム (単位: msec)
dInPosWidth	インポジション幅 (単位: mm または deg)
dPosLimitPos	正のソフトウェア制限位置 (単位: mm または deg)
dNegLimitPos	負のソフトウェア制限位置 (単位: mm または deg)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この関数は最初に呼び出す必要があります。MPI_ConnectMegaulink 関数が呼び出された後にこの関数を呼び出します。

(2) 特定の動作条件を変更するには、関数 MPI_SetVel、MPI_SetAcc、MPI_SetDec、MPI_SetDecKill、MPI_SetSmoothTime、MPI_SetInPosWidth、MPI_SetSwLimitPos を呼び出すことをお勧めします。

(3) E1 シリーズドライバーはソフトウェア制限機能をサポートしていません。パラメーター dPosLimitPos および dNegLimitPos には-1 を記述してください。関数 MPI_EnableSwLimit、MPI_DisableSwLimit、MPI_SetSwLimitPos および MPI_GetSwLimitPos はサポートされていません。

2.4 MPI_GetVersion

目的

HIWIN API バージョンの情報を取得します。

プロトタイプ

```
void MPI_GetVersion(char *pszVer);
```

パラメーター

*pszVer	HIWIN API バージョンの情報

Return

N/A

備考

(1) この機能は HIOM でも利用可能です。

(2) HIWIN API のバージョン情報は随時更新されます。

(このページはブランクになっています)

3. コミュニケーション

3.1 MPI_ConnectMegaulink

目的

mega-ulink 通信を開始します。

プロトタイプ

```
int MPI_ConnectMegaulink(bool blNetWorkSetting=true);
```

パラメーター

blNetWorkSetting　　true: デバイスの通信設定の構成ウィンドウを表示します。
false: デバイス通信設定の構成ウィンドウを表示しません。

Return

0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は HIOM でも利用可能ですが、E2 シリーズドライバーはサポートされていません。
(2) 図 3.1.1 に mega-ulink 通信開始手順を示す。
(3) デバイスの通信設定の詳細については、セクション 1.4 を参照してください。

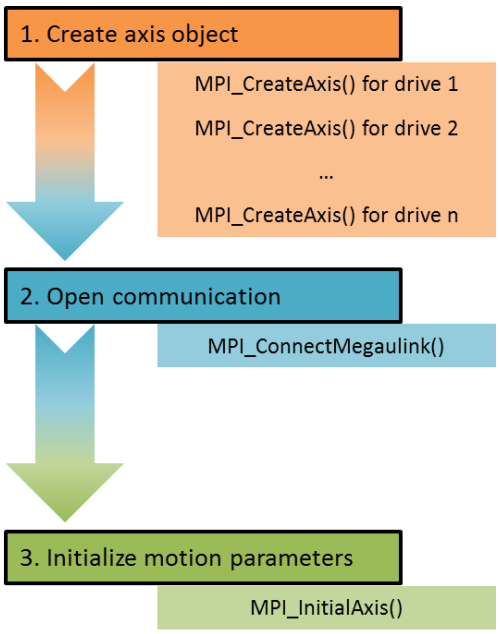


図 3.1.1

3.2 MPI_Disconnect

目的

mega-ulink 通信を終了します。

プロトタイプ

```
int MPI_Disconnect(CAxis *pAxis=nullptr);
```

パラメーター

*pAxis　　CAxis クラス オブジェクトのポインター
mega-ulink 通信が終了すると、このパラメーターは NULL ポインターになります。

Return

0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は HIOM でも利用可能です。
(2) 図 3.2,1 に mega-ulink 通信を終了する手順を示す。
(3) アプリケーションウィンドウが閉じられる前にこの関数を呼び出します。

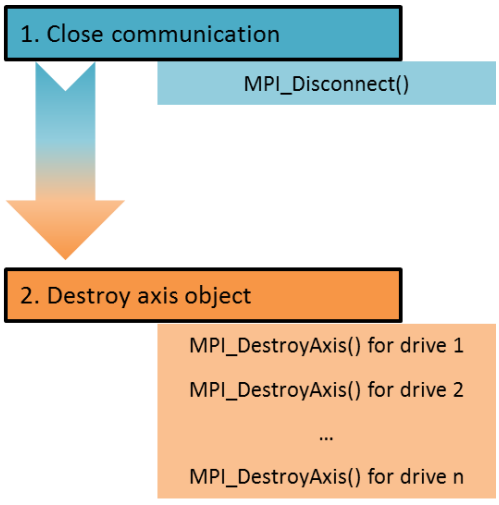


図 3.2.1

3.3 MPI_GetSlaveAmount

目的

mega-ulink 経由で接続されているスレーブの数を取得します。

プロトタイプ

```
int MPI_GetSlaveAmount(int *pnSlaveAmount);
```

パラメーター

*pnSlaveAmount　　スレーブの数。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラーコードの説明を参照してください。

備考

この機能は HIOM でも利用できます。

3.4 MPI_GetComSuccess

目的

通信状態を確認します。

プロトタイプ

```
void MPI_GetComSuccess(int *pnComState);
```

パラメーター

*pnComState 0: 通信に失敗しました。
1: 通信は成功しました。

Return

N/A

備考

この機能は HIOM でも利用できます。

3.5 MPI_SetCommunicationPar

目的

mega-ulink 通信パラメーターを設定します。

プロトタイプ

```
void MPI_SetComunicationPar(int nTimeOut=200, int nLockTime=200, int nIterNum=6);
```

パラメーター

nTimeOut	ドライバーがメッセージを送り返すのを待つ時間です。 デフォルトは 200 です。（単位：msec）
nLockTime	通信エラーが発生するのを許容する時間。 デフォルトは 200 です。（単位：msec）
nIterNum	ドライバーにメッセージを繰り返し送信する最大回数。 デフォルトは 6 です。

Return

N/A

備考

MPI_ConnectMegaulink 関数を呼び出す前にこの関数を呼び出します。

3.6 MPI_Reconnect

目的

mega-ulink 通信を切断して再開します。

プロトタイプ

```
int MPI_Reconnect(CAxis *pAxis=nullptr, unsigned long nWaitTime=100);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。 mega-ulink 通信が再接続されると、このパラメーターは NULL ポインターになります。
nWaitTime	接続が切断された後に、接続が再構築されるまで待機する時間。 デフォルトは 100 です。（単位：msec）

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

MPI_ConnectMegaulink、MPI_ConnectUsbById、または MPI_ConnectUsbByInfo 関数を呼び出す前に、この関数を呼び出さないでください。

3.7 MPI_GetNetworkAdapterInfo

目的

構造体 TNetworkAdapterInfo (PC 内のすべてのネットワーク インターフェース カードの情報を保存します) を取得します。

プロトタイプ

```
int MPI_GetNetworkAdapterInfo(TNetworkAdapterInfo *ptNetworkAdapterInfo);
```

パラメーター

*ptNetworkAdapterInfo　　TNetworkAdapterInfo のポインター。

Return

0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

3.7.1 TNetworkAdapterInfo

Axis.h で定義されている 構造体 TNetworkAdapterInfo は、PC 内のすべてのネットワーク インターフェース カードの情報を取得し、配列で格納します。メンバーは次のとおりです。

TNetworkAdapter *ptNetworkAdapter	PC 内のすべてのネットワーク インターフェース カードの情報を格納する配列。
int iCount	PC 内のすべてのネットワーク インターフェース カードの情報を格納する配列のサイズ。
char szNetworkAdapterErrorMessage	PC 内のすべてのネットワーク インターフェース カードの情報を取得するプロセス中に返されたエラー メッセージ。文字列の長さは 256 文字です。

3.7.2 TNetworkAdapter

Axis.h で定義されている構造体 TNetworkAdapter には、PC 内のネットワーク インターフェース カードの 1 つの名前と説明が格納されます。メンバーは次のとおりです。

char szNetworkAdapterName	PC 内のネットワーク インターフェース カードの 1 つの名前。 文字列の長さは 250 文字です。
char szNetworkAdapterDescription	PC のネットワーク インターフェース カードの 1 つに関する説明。 文字列の長さは 250 文字です。

3.8 MPI_ClearNetworkAdapterInfo

目的

構造体 TNetworkAdapterInfo をクリアします。

プロトタイプ

```
int MPI_ClearNetworkAdapterInfo(TNetworkAdapterInfo *ptNetworkAdapterInfo);
```

パラメーター

*ptNetworkAdapterInfo　　TNetworkAdapterInfo のポインター。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

この関数を呼び出すと、次の 3 つのことが完了します：

(1) ポインター配列 TNetworkAdapter *ptNetworkAdapter を削除します。

(2) int iCount を 0 に設定する。

(3) char szNetworkAdapterErrorMessage を空の文字列として設定します。

3.9 MPI_ConnectMegaulinkByNetworkAdapter

目的

PC のネットワーク インターフェース カードの 1 つを割り当てて、mega-ulink 通信を開始します。

プロトタイプ

```
int MPI_ConnectMegaulinkByNetworkAdapter(TNetworkAdapter *ptNetworkAdapter);
```

パラメーター

*ptNetworkAdapter　　TNetworkAdapter のポインター。

Return

0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) 関数 MPI_ConnectMegaulink と同じです。
　　違いは、この機能ではユーザーがネットワーク デバイスを手動で選択する必要がないことです。
(2) この関数は MPI_GetNetworkAdapterInfo 関数と一緒に使用する必要があります。
(3) TNetworkAdapter 構造体の詳細については、セクション 3.7.2 を参照してください。
(4) この機能は E2 シリーズドライバーをサポートしていません。

例

```
//  Declare the variable of TNetworkAdapterInfo
TNetworkAdapterInfo tNetworkAdapterInfo = {0};
//  Get the information of all network interface cards in PC
MPI_GetNetworkAdapterInfo(&tNetworkAdapterInfo);
//  Assign the first network interface card in PC to start mega-ulink communication
MPI_ConnectMegaulinkByNetworkAdapter(&(tNetworkAdapterInfo.ptNetworkAdapter[0]));
//  Clear the content of tNetworkAdapterInfo
MPI_ClearNetworkAdapterInfo(&tNetworkAdapterInfo);
```

3.10 MPI_ConnectUsbById

目的

USB の ID で USB 通信を開始します。

プロトタイプ

```
int MPI_ConnectUsbById(CAxis *pAxis, bool blNetWorkSetting, int nId);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
blNetWorkSetting	true: デバイスの通信設定の構成ウィンドウを表示します。複数の接続が実行された場合は、最後の接続の設定のみが表示されます。 false: デバイス通信設定の構成ウィンドウを表示しません。
nId	USB デバイスの接続 ID。1 つの USB デバイスには 1 つのポート ID があり、0 から始まり、Thunder 接続ウィンドウの USB ポートのシーケンスと同じです。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は HIOM では利用できません。
図 3.10.1 は USB の ID で USB 通信を開始する手順を示しています。
最大 5 つの接続を受け入れます。

(2) デバイスの通信設定の詳細については、セクション 1.4 を参照してください。

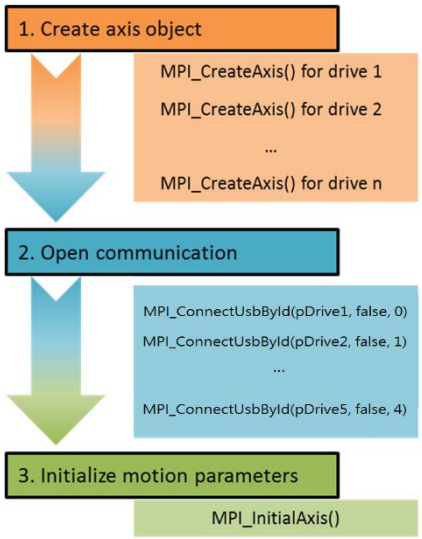


図 3.10.1

3.11 MPI_ConnectUsbByInfo

目的

USB の情報により USB 通信を開始します。

プロトタイプ

```
int MPI_ConnectUsbByInfo(CAxis *pAxis, const char* pszUsbInfo);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pszUsbInfo	USB デバイスの接続情報。1 つの USB ポートには固定の接続情報があり、関数 MPI_GetUsbDevice から取得できます。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は HIOM ではご利用いただけません。

(2) 図 3.11.1 は USB の情報により USB 通信を開始する手順を示します。
最大 5 つの接続を受け入れます。

(3) デバイスの通信設定の詳細については、セクション 1.4 を参照してください。

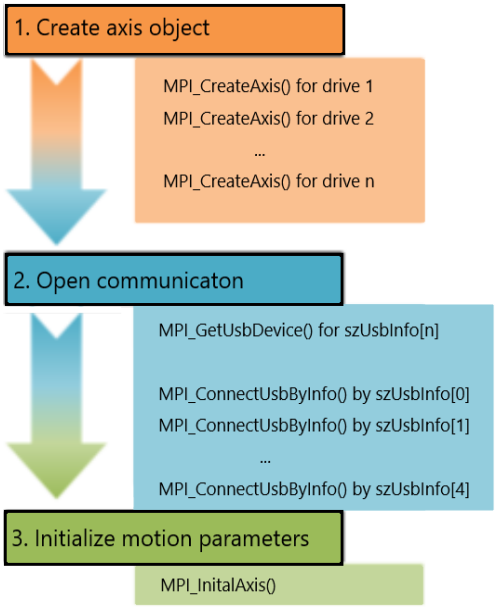


図 3.11.1

3.12 MPI_GetUsbDevice

目的

USB の接続情報を取得します。

プロトタイプ

```
int MPI_GetUsbDevice(char* szDeviceList[], const int nSize=MAX_USB_DEV_AMOUNT,
                     cont int nStringLength=MAX_USB_DEV_STR,
                     int* pnCount=nullptr);
```

パラメーター

*szDeviceList[]	USB デバイスのリスト。
nSize	取得する USB デバイスの数
nStringLength	USB デバイス名の文字列の長さ
*pnCount	USB デバイスが実際に取得されました。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は HIOM では使用できません。

(2) 接続された USB デバイスを最大 10 個までリスト表示できます。

(3) デバイスの通信設定の詳細については、セクション 1.4 を参照してください。

(このページはブランクになっています)

4. 変数にアクセスする

4.1 MPI_GetSlaveID

目的

スレーブ ID を取得します。

プロトタイプ

```
int MPI_GetSlaveID(CAxis *pAxis, int *pnSlaveID);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pnSlaveID	スレーブ ID。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

いた

備考

(1) この機能は HIOM でも利用可能です。

(2) ヒューマンマシンインターフェースにスレーブ ID を表示するには、この関数を呼び出します。

4.2 MPI_GetAxisName

目的
軸名を取得します。

プロトタイプ

```
int MPI_GetAxisName(CAxis *pAxis, char *pszAxisName);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pszAxisName	軸名

Return
0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考
(1) この機能は HIOM でも利用可能です。
(2) ヒューマンマシンインターフェース上に軸名を表示するには、この関数を呼び出します。

4.3 MPI_GetFeedbackPos

目的

フィードバック位置を取得します。

プロトタイプ

```
int MPI_GetFeedbackPos(CAxis *pAxis, double *pdPos);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pdPos	フィードバック位置。（単位：mm または deg）

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

4.4 MPI_GetFeedbackVel

目的

フィードバック速度を取得します。

プロトタイプ

```
int MPI_GetFeedbackVel(CAxis *pAxis, double *pdVel);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pdVel	フィードバック速度。(単位: mm/sec または deg/sec)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

4.5 MPI_GetVel

目的

ドライバーから最大速度を取得します。

プロトタイプ

```
int MPI_GetVel(CAxis *pAxis, double *pdVel);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pdVel	最大速度。（単位：mm/sec または deg/sec）

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

4.6 MPI_SetVel

目的
ドライバーに最大速度を設定します。

プロトタイプ

```
int MPI_SetVel(CAxis *pAxis, double dVel);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
dVel	設定する最大速度。(単位: mm/sec または deg/sec)

Return
0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考
設定された最大速度は即時有効になります。

4.7 MPI_GetAcc

目的

ドライバーから加速を取得します。

プロトタイプ

```
int MPI_GetAcc(CAxis *pAxis, double *pdAcc);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pdAcc	加速度. (単位: mm/sec^2 または deg/sec^2)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

4.8 MPI_SetAcc

目的
ドライバーに加速度を設定します。

プロトタイプ

```
int MPI_SetAcc(CAxis *pAxis, double dAcc);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
dAcc	設定する加速度 (単位: mm/sec^2 または deg/sec^2)

Return
0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考
設定された加速はすぐに有効になります。

4.9 MPI_GetDec

目的

ドライバーから減速を取得します。

プロトタイプ

```
int MPI_GetDec(CAxis *pAxis, double *pdDec);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pdDec	減速度. (単位: mm/sec^2 または deg/sec^2)

Return

0 –関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

4.10 MPI_SetDec

目的
ドライブに減速度を設定します。

プロトタイプ

```
int MPI_SetDec(CAxis *pAxis, double dDec);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
dDec	減速度設定. (単位: mm/sec^2 または deg/sec^2)

Return
0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考
設定された減速度は即時有効になります。

4.11 MPI_GetDecKill

目的

ドライバーから緊急減速停止を取得します。

プロトタイプ

```
int MPI_GetDecKill(CAxis *pAxis, double *pdDecKill);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pdDecKill	緊急減速停止。（単位：mm/sec^2 または deg/sec^2）

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

4.12 MPI_SetDecKill

目的

ドライバーに緊急減速停止を設定します。

プロトタイプ

```
int MPI_SetDecKill(CAxis *pAxis, double dDecKill);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
dDecKill	緊急停止時の減速度を設定します。（単位：mm/sec^2 または deg/sec^2）

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

設定された緊急減速停止が即時有効になります。

4.13 MPI_GetSmoothTime

目的

ドライバーからスムーズタイムを実現します。

プロトタイプ

```
int MPI_GetSmoothTime(CAxis *pAxis, int *pnSmoothTime);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pnSmoothTime	スムーズタイム. (単位: msec)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

4.14 MPI_SetSmoothTime

目的
ドライブにスムーズタイムを設定します。

プロトタイプ

```
int MPI_SetSmoothTime(CAxis *pAxis, int nSmoothTime);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
nSmoothTime	スムーズタイムの設定 (単位: msec)

Return
0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考
設定されたスムーズタイムは即時有効になります。

4.15 MPI_GetInPosWidth

目的

ドライバーからインポジション幅を取得します。

プロトタイプ

```
int MPI_GetInPosWidth(CAxis *pAxis, double *pdInPosWidth);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pdInPosWidth	インポジション幅. (単位: mm または deg)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

4.16 MPI_SetInPosWidth

目的
ドライバーのインポジション幅を設定します。

プロトタイプ

```
int MPI_SetInPosWidth(CAxis *pAxis, double dInPosWidth);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
dInPosWidth	設定するインポジション幅 (単位: mm または deg)

Return
0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考
設定されたインポジション幅は即時に有効になります。

4.17 MPI_GetSwLimitPos

目的

ドライブからソフトウェア リミット保護の正/負の位置を取得します。

プロトタイプ

```
int MPI_GetSwLimitPos(CAxis *pAxis, double *pdPosLimitPos, double *pdNegLimitPos);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pdPosLimitPos	正のソフトウェア制限位置 (単位: mm または deg)
*pdNegLimitPos	負のソフトウェア制限位置. (単位: mm または deg)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

E1 シリーズドライバーはこの機能をサポートしていません。

4.18 MPI_SetSwLimitPos

目的

ドライバーに対するソフトウェア制限保護の正/負の位置を設定します。

プロトタイプ

```
int MPI_SetSwLimitPos(CAxis *pAxis, double dPosLimitPos, double dNegLimitPos);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
dPosLimitPos	設定する正のソフトウェアリミット位置 (単位: mm または deg)
dNegLimitPos	設定する負のソフトウェアリミット位置 (単位: mm または deg)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) dPosLimitPos は dNegLimitPos より大きくなければなりません。
(2) ソフトウェアリミット保護の正負位置の設定は即時有効になります。
(3) ソフトウェア制限保護を有効にするには、関数 MPI_EnableSwLimit を呼び出します。ソフトウェア制限保護を無効にするには、関数 MPI_DisableSwLimit を呼び出します。
(4) E1 シリーズドライバーはこの機能をサポートしていません。

(このページはブランクになっています)

5. 軸動作

5.1 MPI_ServoOn

目的

ドライバーを有効にします。

プロトタイプ

```
int MPI_ServoOn(CAxis *pAxis);
```

パラメーター

*pAxis　　CAxis クラス オブジェクトのポインター。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) MPI_IsReady 関数でサーボオン/オフ信号を確認します。

(2) D シリーズドライバーを使用する場合、入力信号の 1 つを「Axis Enable」に設定する必要があります。そうしないと、この機能は失敗します。

5.2 MPI_ServoOff

目的
ドライブを無効にします。

プロトタイプ

```
int MPI_ServoOff(CAxis *pAxis);
```

パラメーター
*pAxis CAxis クラス オブジェクトのポインター。

Return
0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考
関数 MPI_IsReady でサーボのオン/オフ信号を確認します。

5.3 MPI_MoveAbsolute

目的

特定の動作条件下でモーターを絶対目標位置まで駆動します。

プロトタイプ

```
int MPI_MoveAbsolute(CAxis *pAxis, double dPos, double dVel=0, double dAcc=0,
                     double dDec=0);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
dPos	モーターのゼロ点を基準とした目標位置。 (単位: mm または deg)
dVel	動作速度 (単位: mm/sec または deg/sec)
dAcc	動作加速度 (単位: mm/sec^2 または deg/sec^2)
dDec	動作減速度 (単位: mm/sec^2 または deg/sec^2)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) パラメーターdVel、dAcc、dDec が 0（デフォルト値）に設定されている場合、モーターは最後に設定された動作条件で駆動されます。動作条件を設定できる関数は、MPI_InitialAxis、MPI_SetVel、MPI_SetAcc、MPI_SetDec、MPI_MoveAbsolute、MPI_MoveRelative です。

(2) この関数は、MPI_StartHome 関数によってゼロ点が正常に定義された後に呼び出されるべきです。

5.4 MPI_MoveRelative

目的

特定の動作条件下でモーターを相対的な目標位置まで駆動します。

プロトタイプ

```
int MPI_MoveRelative(CAxis *pAxis, double dPos, double dVel=0, double dAcc=0,
                     double dDec=0);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
dPos	モーターの現在のエンコーダー位置を基準としたターゲット位置。 (単位: mm または deg)
dVel	モーター速度 (単位: mm/sec または deg/sec)
dAcc	モーター加速度 (単位: mm/sec^2 または deg/sec^2)
dDec	モーター減速度 (単位: mm/sec^2 または deg/sec^2)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

パラメーター dVel、dAcc、dDec が 0 (デフォルト値) に設定されている場合、モーターは最後に設定された動作条件で駆動されます。動作条件を設定できる関数は、MPI_InitialAxis、MPI_SetVel、MPI_SetAcc、MPI_SetDec、MPI_MoveAbsolute、MPI_MoveRelative です。

5.5 MPI_JogPositive

目的

モーターを特定の動作速度で正方向に駆動します。

プロトタイプ

```
int MPI_JogPositive(CAxis *pAxis, double dVel);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
dVel	動作速度 (単位: mm/sec または deg/sec)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) モーターを負方向に駆動するには、関数 MPI_JogNegative を呼び出します。

(2) モーターを停止するには、MPI_StopMotion 関数を呼び出すか、速度を 0 に設定します。

5.6 MPI_JogNegative

目的

モーターを特定の動作速度で負方向に駆動します。

プロトタイプ

```
int MPI_JogNegative(CAxis *pAxis, double dVel);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
dVel	動作速度 (単位: mm/sec または deg/sec)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) モーターを正方向に駆動するには、関数 MPI_JogPositive を呼び出します。

(2) モーターを停止するには、MPI_StopMotion 関数を呼び出すか、速度を 0 に設定します。

5.7 MPI_StopMotion

目的

関数 MPI_InitialAxis または MPI_SetDecKill で定義された緊急減速停止でモーターを停止します。

プロトタイプ

```
int MPI_StopMotion(CAxis *pAxis);
```

パラメーター

*pAxis CAxis クラス オブジェクトのポインター。

Return

0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

現在の緊急減速停止を取得するには、関数 MPI_GetDecKill を呼び出します。

5.8 MPI_StartHome

目的
原点復帰手順を開始します。

プロトタイプ

```
int MPI_StartHome(CAxis *pAxis);
```

パラメーター
*pAxis CAxis クラス オブジェクトのポインター。

Return
0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考
原点復帰手順で使用されるすべてのパラメーターは、最初に Lightning または Thunder に設定する必要があります。

5.9 MPI_SetZero

目的

モーターの現在の位置をゼロ点として設定します。

プロトタイプ

```
int MPI_SetZero(CAxis *pAxis);
```

パラメーター

*pAxis CAxis クラス オブジェクトのポインター。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

通常、ゼロ ポイントは組み込みの原点復帰手順によって定義されます。ユーザーが外部原点復帰手順でゼロ ポイントを見つけることにした場合、この関数を呼び出してドライバーにゼロ ポイントを設定できます。

5.10 MPI_ResetDrive

目的
ドライバーをリセットします。

プロトタイプ

```
int MPI_ResetDrive(CAxis *pAxis);
```

パラメーター
*pAxis CAxis クラス オブジェクトのポインター。

Return
0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

5.11 MPI_ClearError

目的

ドライバーのエラー/アラームをクリアします。

プロトタイプ

```
int MPI_ClearError(CAxis *pAxis);
```

パラメーター

*pAxis CAxis クラス オブジェクトのポインター。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) D シリーズドライバーを使用する場合、入力信号の 1 つを「エラークリア」に設定する必要があります。そうしないと、この機能は失敗します。

(2) この機能は特定のエラー/アラームのみをクリアできます。E1 シリーズドライバー使用時に特定のアラームをクリアできるかどうかを確認するには、「E1 シリーズドライバーユーザーマニュアル」の表 13.2.1.1 の「アラームリセット」の列を参照してください。

5.12 MPI_EnableSwLimit

目的

ソフトウェア制限保護を有効にします。

プロトタイプ

```
int MPI_EnableSwLimit(CAxis *pAxis);
```

パラメーター

*pAxis CAxis クラス オブジェクトのポインター。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) ソフトウェア制限保護を無効にするには、関数 MPI_DisableSwLimit を呼び出します。

(2) ソフトウェアリミット保護の正負の位置を設定するには、関数 MPI_SetSwLimitPos を呼び出します

(3) E1 シリーズドライバーはソフトウェアリミット機能をサポートしていません。

5.13 MPI_DisableSwLimit

目的

ソフトウェア制限保護を無効にします。

プロトタイプ

```
int MPI_DisableSwLimit(CAxis *pAxis);
```

パラメーター

*pAxis CAxis クラス オブジェクトのポインター。

Return

0 –関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) ソフトウェア制限保護を有効にするには、関数 MPI_EnableSwLimit を呼び出します。

(2) ソフトウェアリミット保護の正負の位置を設定するには、関数 MPI_SetSwLimitPos を呼び出します。

(3) E1 シリーズドライバーはソフトウェアリミット機能をサポートしていません。

6. アクセス状態

6.1 MPI_IsDriveReady

目的

E1 シリーズドライバーの準備ができているかどうかを確認します。

プロトタイプ

```
int MPI_IsDriveReady(CAxis *pAxis, bool *pblReady);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pblReady	true: ドライバーの準備ができました。 false: ドライバーの準備ができていません。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

D シリーズドライバーはこの機能をサポートしていません。

6.2 MPI_IsReady

目的

ドライバーがサーボの準備ができているかどうかを確認します。

プロトタイプ

```
int MPI_IsReady(CAxis *pAxis, bool *pblReady);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pblReady	true: ドライバーはサーボ準備ができています。 false: ドライバーはサーボオフです。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

6.3 MPI_IsHomed

目的

ドライブが原点位置にあるかどうかを確認します。

プロトタイプ

```
int MPI_IsHomed(CAxis *pAxis, bool *pblHomed);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pblHomed	true: ドライバーが原点位置にあります。 false: ドライバーはまだ原点位置に戻っていません。

Return

0 –関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

6.4 MPI_IsMoving

目的

モーターが動いているか確認します。

プロトタイプ

```
int MPI_IsMoving(CAxis *pAxis, bool *pblMoving);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pblMoving	true: モーターが動いています。 false: モーターは停止しています。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

6.5 MPI_IsStopped

目的

モーターが停止しているかどうかを確認します。

プロトタイプ

```
int MPI_IsStopped(CAxis *pAxis, bool *pblStop);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pblStop	true: モーターが停止しました。 false: モーターが動いています。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

6.6 MPI_IsError

目的

ドライバーにエラー/アラームがないか確認します。

プロトタイプ

```
int MPI_IsError(CAxis *pAxis, bool *pblError);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pblError	true: ドライバーにエラー/アラームが発生しています。 false: ドライバーにエラー/アラームはありません。

Return

0 –関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

6.7 MPI_IsSwLimitEn

目的

ソフトウェア制限保護が有効になっているかどうかを確認します。

プロトタイプ

```
int MPI_IsSwLimitEn(CAxis *pAxis, bool *pbISwLimitEn);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pbISwLimitEn	true: ソフトウェア制限保護が有効になっています。 false: ソフトウェア制限保護は無効になっています。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

E1 シリーズドライバーはこの機能をサポートしていません。

6.8 MPI_IsHwLimit

目的

ハードウェア制限センサーがトリガーされているかどうかを確認します。

プロトタイプ

```
int MPI_IsHwLimit(CAxis *pAxis, bool *pblLeftLimit, bool *pblRightLimit);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pblLeftLimit	true: ハードウェア左リミットセンサーがトリガーされました。
	false: ハードウェア左リミットセンサーがトリガーされません。
*pblRightLimit	true: ハードウェア右リミットセンサーがトリガーされました。
	false: ハードウェア右リミットセンサーがトリガーされません。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

6.9 MPI_IsSwLimit

目的

ソフトウェア制限保護がトリガーされているかどうかを確認します。

プロトタイプ

```
int MPI_IsSwLimit(CAxis *pAxis, bool *pblLeftLimit, bool *pblRightLimit);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pblLeftLimit	true: ソフトウェア左制限保護がトリガーされます。 false: ソフトウェア左制限保護はトリガーされません。
*pblRightLimit	true: ソフトウェア権限制限保護がトリガーされます。 false: ソフトウェア権限制限保護はトリガーされません。

Return

0 –関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

E1 シリーズドライバーはこの機能をサポートしていません。

6.10 MPI_WaitServoReady

目的

ドライバーがサーボ準備完了になるまで待ちます。

プロトタイプ

```
int MPI_WaitServoReady(CAxis *pAxis, int nTimeOut);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
nTimeOut	タイムアウト値 (単位: ms; 最小: 10) ユーザーが 10 未満に設定した場合、自動的に 10 に設定されます。

Return

0 –関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

6.11 MPI_WaitHomeOver

目的

ドライバーが原点復帰手順を完了するまで待ちます。

プロトタイプ

```
int MPI_WaitHomeOver(CAxis *pAxis, int nTimeOut);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
nTimeOut	タイムアウト値 (単位: ms; 最小: 10) ユーザーが 10 未満に設定した場合、自動的に 10 に設定されます。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

6.12 MPI_WaitMoveOver

目的

モーターの動きが完了するまで待ちます。

プロトタイプ

```
int MPI_WaitMoveOver(CAxis *pAxis, int nTimeOut);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
nTimeOut	タイムアウト値 (単位: ms; 最小: 10) ユーザーが 10 未満に設定した場合、自動的に 10 に設定されます。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

E1 シリーズドライバーを使用する場合、モーターの移動が完了する条件はドライバーの COIN 信号の設定によって異なります。詳細については、「E1 シリーズドライバーユーザーズマニュアル」のセクション 8.4.4 位置決め完了出力（COIN）信号を参照してください。

6.13 MPI_SetCallBackStates

目的

状態が変化すると、ユーザーに積極的に通知され、対応するプロセスが実行されます。

プロトタイプ

```
int MPI_SetCallBackStates(CAxis *pAxis=nullptr, long lFuncAdd);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。 mega-ulink 通信が接続されている場合、このパラメーターは null ポインターになります。
lFuncAdd	関数ポインター。そのプロトタイプは void (*func)(BYTE, int) です。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

type と id は、ファイル <appl name.stt> 内の各状態に対して定義される状態属性です。状態が変更されると、関数 PutEventSt が呼び出され、type の最初の 2 ビットの少なくとも 1 つが 1 になります(これは、送信イベントの変更方向を定義します)。詳細については、「D シリーズドライバーの PDL リファレンス マニュアル」を参照してください。

例

```
void PutEventSt(BYTE type, int id)
{
    char str[200];
    sprintf(str, "state change, type=%02x, id=%ld", type, id);
    MessageBox(str);
}
……
MPI_SetCallBackStates(PutEventSt);
//  Set DLL with pointer of callback function
```

7. 一般的な入出力

7.1 MPI_GetInputStatus

目的

特定の入力ステータスを取得します。

プロトタイプ

```
int MPI_GetInputStatus(CAxis *pAxis, int nInputIndex, bool *pblStatus);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
nInputIndex	特定の入力のインデックス。
*pblStatus	入力ステータス true はオン、false はオフです。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

この機能は HIOM でも利用できます。

7.2 MPI_GetOutputStatus

目的

特定の出力ステータスを取得します。

プロトタイプ

```
int MPI_GetOutputStatus(CAxis *pAxis, int nOutputIndex, bool *pblStatus);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
nOutputIndex	特定の出力のインデックス。
*pblStatus	出力ステータス。 true はオン、false はオフです。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

この機能は HIOM でも利用できます。

7.3 MPI_SetOutput

目的

特定の出力に設定します。

プロトタイプ

```
int MPI_SetOutput(CAxis *pAxis, int nOutputIndex);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
nOutputIndex	特定の出力のインデックス

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

この機能は HIOM でも利用できます。

7.4 MPI_ClearOutput

目的
特定の出力を開始します。

プロトタイプ

```
int MPI_ClearOutput(CAxis *pAxis, int nOutputIndex);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
nOutputIndex	特定の出力のインデックス。

Return
0 – 関数は成功しました
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考
この機能は HIOM でも利用できます。

7.5 MPI_ToggleOutput

目的

特定の出力ステータスを切り替えます。

プロトタイプ

```
int MPI_ToggleOutput(CAxis *pAxis, int nOutputIndex);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
nOutputIndex	特定の出力のインデックス。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

この機能は HIOM でも利用できます。

8. 特定の入出力

8.1 MPI_SetTriggerPosition

目的

固定間隔位置トリガー機能の開始位置と終了位置を設定します。

プロトタイプ

```
int MPI_SetTriggerPosition(CAxis *pAxis, double dStartPos, double dEndPos,
                           double dInterval=0);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
dStartPos	固定間隔位置トリガー機能の開始位置 (単位: mm または deg)
dEndPos	固定間隔位置トリガー機能の終了位 (単位: mm または deg)
dInterval	間隔距離 (単位: mm または deg)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は E1 シリーズドライバーおよび D1-N ドライバーでのみ使用できます。

(2) E1 シリーズドライバーの位置トリガー機能の詳細については、「E1 シリーズドライバーユーザーマニュアル」のセクション 8.13「位置トリガー機能の設定」を参照してください。

8.2 MPI_SetTriggerInterval

目的

固定間隔位置トリガー機能の間隔距離を設定します。

プロトタイプ

```
int MPI_SetTriggerInterval(CAxis *pAxis, double dInterval);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
dInterval	間隔距離 (単位: mm または deg)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

この機能は E1 シリーズドライバーおよび D1-N ドライバーでのみ使用できます。

E1 シリーズドライバーの位置トリガー機能の詳細については、「E1 シリーズドライバーユーザーマニュアル」のセクション 8.13「位置トリガー機能の設定」を参照してください。

8.3 MPI_EnableTrigger

目的

位置トリガー機能を有効にします。

プロトタイプ

```
int MPI_EnableTrigger(CAxis *pAxis, ENUM_POLARITY_TYPE enumPolarity=ePT_LOW,
                      int nPulseWidth=0);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
enumPolarity	出力極性 ePT_LOW: 通常低 ePT_HIGH: 通常高 E1 シリーズドライバーはサポートされません。
nPulseWidth	パルス幅 (単位: 25 ns; 幅: 1~4095) ユーザーが 4095 より大きい値に設定した場合、自動的に 4095 に設定されます。 E1 シリーズドライバーはサポートされません。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は E1 シリーズドライバーおよび D1-N ドライバーでのみ使用できます。

(2) E1 シリーズドライバーで位置トリガー機能を有効にするためにこの関数を呼び出す前に、まず Thunder で出力極性とパルス幅を設定してください。「E1 シリーズドライバーユーザーマニュアル」のセクション 8.13 位置トリガー機能の設定を参照してください。

(3) この関数は、MPI_StartHome 関数によってゼロ点が正常に定義された後に使用してください。

8.4 MPI_DisableTrigger

目的

位置トリガー機能を無効にします。

プロトタイプ

```
int MPI_DisableTrigger(CAxis *pAxis);
```

パラメーター

*pAxis CAxis クラス オブジェクトのポインター。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は E1 シリーズドライバーおよび D1-N ドライバーでのみ使用できます。

(2) E1 シリーズドライバーの位置トリガー機能の詳細については、「E1 シリーズドライバーユーザーマニュアル」のセクション 8.13「位置トリガー機能の設定」を参照してください。

(3) モーター位置が MPI_SetTriggerPosition 関数で設定された dEndPos を超えると、位置トリガー機能は自動的に無効になります。

8.5 MPI_SetTriggerPositionArray

目的

ランダム間隔位置トリガー機能の位置配列を設定します。

プロトタイプ

```
int MPI_SetTriggerPositionArray(CAxis *pAxis, int iIndex, double dPosition);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
iIndex	インデックス
dPosition	位置データ

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は E1 シリーズドライバーでのみ使用できます。

(2) E1 シリーズドライバーの位置トリガー機能の詳細については、「E1 シリーズドライバーユーザーマニュアル」のセクション 8.13「位置トリガー機能の設定」を参照してください。

8.6 MPI_SetTriggerPositionStateArray

目的

ランダム間隔位置トリガー機能のステータス配列（ステータス出力）を設定します。

プロトタイプ

```
int MPI_SetTriggerPositionStateArray(CAxis *pAxis, int iIndex, int iState);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
iIndex	インデックス
iState	ステータスデータ

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は E1 シリーズドライバーでのみ使用できます。

(2) E1 シリーズドライバーの位置トリガー機能の詳細については、「E1 シリーズドライバーユーザーマニュアル」のセクション 8.13「位置トリガー機能の設定」を参照してください。

(3) ステータスデータの詳細については、「E1 シリーズドライバーユーザーズマニュアル」の表 8.13.13 を参照してください。

8.7 MPI_SetTriggerPositionStartIndex

目的

ランダム間隔位置トリガー機能の開始インデックスを設定します。

プロトタイプ

```
int MPI_SetTriggerPositionStartIndex(CAxis *pAxis, int iIndex);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
iIndex	開始インデックス ランダム間隔位置トリガー機能は、この位置データから開始されます。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は E1 シリーズドライバーでのみ使用できます。

(2) E1 シリーズドライバーの位置トリガー機能の詳細については、「E1 シリーズドライバーユーザーマニュアル」のセクション 8.13「位置トリガー機能の設定」を参照してください。

8.8 MPI_SetTriggerPositionEndIndex

目的

ランダム間隔位置トリガー機能の終了インデックスを設定します。

プロトタイプ

```
int MPI_SetTriggerPositionEndIndex(CAxis *pAxis, int iIndex);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
iIndex	終了インデックス ランダム間隔位置トリガー機能はこの位置データで終了します。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は E1 シリーズドライバーでのみ使用できます。

(2) E1 シリーズドライバーの位置トリガー機能の詳細については、「E1 シリーズドライバーユーザーマニュアル」のセクション 8.13「位置トリガー機能の設定」を参照してください。

(このページはブランクになっています)

9. 特定の変数/状態にアクセスする

9.1 MPI_GetVar

目的

ドライバーから特定の変数を取得します。

プロトタイプ

```
int MPI_GetVar(CAxis *pAxis, char *pszVarName, double *pdValue);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pszVarName	変数名
*pdValue	変数値

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

9.2 MPI_SetVar

目的

ドライバーに特定の変数を設定します。

プロトタイプ

```
int MPI_SetVar(CAxis *pAxis, char *pszVarName, double dValue);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pszVarName	変数名
dValue	設定する変数値 ドライバー内の正しい変数タイプに自動的に変換されます。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

9.3 MPI_GetState

目的

ドライバーから特定の状態を取得します。

プロトタイプ

```
int MPI_GetState(CAxis *pAxis, char *pszStateName, bool *pblStatus);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pszStateName	状態名
*pblStatus	変数の状態 true はオン、false はオフです。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

9.4 MPI_SetState

目的

ドライバーに特定の状態を設定します。

プロトタイプ

```
int MPI_SetState(CAxis *pAxis, char *pszStateName, bool blStatus);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pszStateName	状態名
blStatus	設定する変数の状態 true はオン、false はオフです。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

9.5 MPI_GetErrorData

目的

D シリーズドライバーのエラーを取得します。

プロトタイプ

```
int MPI_GetErrorData(CAxis *pAxis, int pnErrorData);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
pnErrorData	エラー番号

Return

0 – 関数は成功しました。
ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は D シリーズドライバーでのみ使用できます。
(2) エラー名については第 12 章を参照ください。

9.6 MPI_GetErrorDataArray

目的

E1 シリーズドライバーのアラームを取得します。

プロトタイプ

```
int MPI_GetErrorDataArray(CAxis *pAxis, int pnErrorData[3]);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
pnErrorData[0]	0～31 ビットは、第 11 章のアラーム 1～32 を表します。
pnErrorData[1]	0～31 ビットは、第 11 章のアラーム 33～64 を表します。
pnErrorData[2]	0～12 ビットは、第 11 章のアラーム 65～77 を表します。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) この機能は E1 シリーズドライバーでのみ使用できます。

(2) アラームの名称と説明については第 11 章を参照してください。

(3) この関数は第 11 章に記載されているアラームのみを取得できます。他のアラームを取得するには、関数 MPI_GetErrorMessage を呼び出します。

9.7 MPI_RunFuncPDL

目的

ドライバーから特定の PDL 機能を実行します。

プロトタイプ

```
int MPI_RunFuncPDL(CAxis *pAxis, char *pszPDLName, int *pnTaskID);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pszPDLName	PDL 関数名 PDL プログラム内の「_」（アンダースコア）で始まるラベルはすべてホストによって認識されます。
*pnTaskID	PDL のタスク ID。(範囲: 1～3)

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

9.8 MPI_IsTaskRunning

目的

ドライバーの PDL の特定のタスクが実行されているかどうかを確認します。

プロトタイプ

```
int MPI_IsTaskRunning(CAxis *pAxis, int nTaskID, bool *plRunning);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
nTaskID	PDL のタスク ID。(範囲: 1～3)
*plRunning	true: タスクは実行中です。 false: タスクは実行されていません。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

9.9 MPI_KillTask

目的

ドライバーの PDL の特定のタスクを強制終了、停止、または続行します。

プロトタイプ

```
int MPI_KillTask(CAxis *pAxis, int nTaskID,
                 ENUM_EXECTASK_TYPE enumExecTaskType=eETT_KILL);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
nTaskID	PDL のタスク ID。(範囲: 1～3)
enumExecTaskType	eETT_KILL: タスクを強制終了します。
	eETT_STOP: タスクを停止します。
	eETT_CONT: タスクを続行します。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

9.10 MPI_SendRAMtoFlash

目的

パラメーターデータをドライバーのメモリに保存します。

プロトタイプ

```
int MPI_SendRAMtoFlash(CAxis *pAxis);
```

パラメーター

*pAxis CAxis クラス オブジェクトのポインター。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

9.11 MPI_GetVarArray

目的

ドライバーから特定の配列型変数を取得します。

プロトタイプ

```
int MPI_GetVarArray(CAxis *pAxis, char *pszVarName, double *pdata, int iSize);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pszVarName	変数名
*pdata	変数値.
iSize	配列のサイズ

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

9.12 MPI_SetVarArray

目的

特定の配列型変数をドライバーに設定します。

プロトタイプ

```
int MPI_SetVarArray(CAxis *pAxis, char *pszVarName, double *pdata, int iSize);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pszVarName	変数名
*pdata	設定する変数値 ドライバー内の正しい変数タイプに自動的に変換されます。
iSize	配列のサイズ

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

9.13 MPI_GetErrorMessage

目的

ドライバーのエラー/アラームを取得します。

プロトタイプ

```
int MPI_GetErrorMessage(CAxis *pAxis, char *pszErrorMessage);
```

パラメーター

*pAxis	CAxis クラス オブジェクトのポインター。
*pszErrorMessage	エラー/アラーム。文字列の長さは 256 文字です。

Return

0 – 関数は成功しました。

ゼロ以外の値 – 関数は失敗しました。第 10 章のエラー コードの説明を参照してください。

備考

(1) E1 シリーズドライバーのアラームは AL□□□です。アラームの原因と対処方法については、「E1 シリーズドライバーユーザーズマニュアル」の 13.2.2 項を参照してください。

(2) D シリーズドライバーのエラー表示はエラー名です。エラーの原因と修正方法については、D シリーズドライバーユーザーマニュアルの関連章のエラーの説明を参照してください。

10. エラーコード

エラーコードのリストは以下の通りです。

表 10.1

コード番号	名称	説明
0	ERR_NO_ERROR	エラーはありません。
1	ERR_NO_OBJECT	オブジェクト ポインターが間違っています。 定義を確認してください。
2	ERR_WRONG_MOTOR_TYPE	モーターの種類が間違っています。 定義を確認してください。
3	ERR_WRONG_UNIT_TYPE	ユニットタイプが間違っています。 定義を確認してください。
4	ERR_WRONG_DRIVE_TYPE	スレーブタイプが間違っています。 定義を確認してください。
5	ERR_WRONG_ROOT_FOLDER	ソフトウェアのルートフォルダが間違っています。 ソフトウェアのバージョンを確認してください。
6	ERR_INVALID_ARGUMENT	引数が無効です。 定義を確認してください。
10	ERR_DISCONNECT	接続されていません。 配線を確認してください。
11	ERR_NO_SLAVE	スレーブが接続されていません。 配線を確認してください。
12	ERR_DUPLICATE_AXIS_NAME	2 つのスレーブが同じ軸名を持っています。 Lightning または Thunder でドライバー設定を確認してください。
13	ERR_MISMATCH_AXIS_NAME	ドライバーからの軸名と設定が一致しません。 Lightning または Thunder 経由でドライバー設定を確認してください。
14	ERR_NOT_INSTALLED	ソフトウェアがインストールされていません。 ドライバーの種類に応じて対応するソフトウェアをインストールしてください。
15	ERR_CONNECTION_OVERRANGE	接続数が制限を超えています。
16	ERR_NO_CONNECTION	軸の通信が存在しません。 接続を確立してください。
20	ERR_GET_VAR_ERROR	ドライバーから変数を取得できませんでした。 変数名またはドライバーのファームウェアバージョンを確認してください。
21	ERR_SET_VAR_ERROR	ドライバーへの変数の設定に失敗しました。 変数名またはドライバーのファームウェアバージョンを確認してください。
22	ERR_GET_STATE_ERROR	ドライバーから状態を取得できませんでした。 ドライバーの状態名またはファームウェアバージョンを確認してください。
23	ERR_SET_STATE_ERROR	ドライバーに状態を設定できませんでした。 ドライバーの状態名またはファームウェアバージョンを確認してください。
24	ERR_RUN_PDL_ERROR	ドライバーから PDL 機能を実行できません。 PDL 機能名またはドライバーのファームウェアバージョンを確認してください。

コード番号	名称	説明
25	ERR_SET_WRONG_INPUT_FUNC	入力機能の呼び出しに失敗しました。 Lightning または Thunder 経由でドライバーの入力機能設定を確認してください。
26	ERR_KILL_TASK_ERROR	PDL のタスクを強制終了、停止、または続行できません。
27	ERR_TASK_NUM_OVER_LIMIT	PDL のタスク ID が範囲外です。 最大値は 3 です。
28	ERR_TASK_NOT_RUN	PDL のタスクは実行されていません。したがって、終了、停止、または続行することはできません。
30	ERR_WAIT_TIMEOUT	タイムアウトです。 タイムアウト値または配線を確認してください。
31	ERR_NOT_HOMED_YET	ドライバーはまだ原点位置に戻っていません。 原点位置の状態を確認してください。
32	ERR_MPD_VERSION_NOT_SUPPORT	ファームウェアバージョンが低すぎるため、この機能をサポートできません。ドライバーのファームウェアバージョンを確認してください。 (このエラーは、E1 シリーズドライバーでのみ発生します。)
33	ERR_NOT_SUPPORT	API 関数はサポートされていません。前の章を参照してください。(このエラーは E1 シリーズドライバーでのみ発生します。)
34	ERR_RESOLUTION_VALUE_ERROR	分解能値が異常です。 Lightening や Thunder でドライバーの分解能設定を確認してください。
35	ERR_DISPLAY_UNIT_NOT_SET	表示単位が設定されていません。Thunder で表示単位を設定してください。(このエラーは E1 シリーズドライバーでのみ発生します。)
36	ERR_INVALID_STATE	ドライバーの状態が無効です。 ドライバーの状態名またはファームウェアバージョンを確認してください。
37	ERR_GET_ERROR_MESSAGE_ERROR	ドライバーのアラームを取得できませんでした。
38	ERR_NETWORK_ADAPTER_ERROR	ネットワークインターフェースカードの取得に失敗しました。
100	ERR_CANT_SAVE_WHEN_ENABLED	ドライバーが有効になっている場合、パラメーターデータをドライバーのメモリに保存することはできません。
101	ERR_SAVE_TO_FLASH_ERROR	パラメーターデータをドライバーのメモリに保存できませんでした。

(このページはブランクになっています)

11. E1 シリーズドライバーアラーム

アラームのリストは以下のように表示されます。

表 11.1

アラーム番号	名称	説明
1	-	予約
2	-	予約
3	-	予約
4	AL024	システムアラーム 1
5	AL025	システムアラーム 2
6	AL030	主回路故障
7	AL040	パラメーター設定エラー
8	AL050	組み合わせエラー
9	-	予約
10	-	予約
11	-	予約
12	AL070	モーターの変化を検出
13	-	予約
14	AL0b0	無効なサーボオンコマンド
15	AL100	過電流を検出しました
16	AL320	回生エネルギーのオーバーフロー
17	AL400	過電圧
18	AL410	低電圧
19	AL510	オーバースピード
20	AL511	エンコーダーパルス出力過速度
21	-	予約
22	-	予約
23	-	予約
24	AL710	過負荷（瞬間最大負荷）
25	AL720	過負荷（連続最大負荷）
26	-	予約
27	-	予約
28	-	予約
29	-	予約
30	AL7A2	電源ボード温度エラー
31	-	予約
32	-	予約
33	AL800	エンコーダーデータバックアップエラー
34	AL810	エンコーダーバッテリー電圧不足

アラーム番号	名称	説明
35	AL820	エンコーダー通信エラー
36	AL830	エンコーダーデータエラー
37	AL840	エンコーダー通信 CRC エラー
38	AL850	エンコーダーカウントエラー
39	AL860	エンコーダーデータ書き込みエラー
40	AL870	エンコーダーの過熱
41	AL880	インクリメンタルエンコーダー信号エラー
42	AL890	エクセレントスマートキューブ（ESC） - インクリメンタルエンコーダーの切断
43	AL8A0	最初のエンコーダーセット - Excellent Smart Cube (ESC) 信号エラー
44	AL8b0	最初のエンコーダーセット - エンコーダー信号エラー
45	AL8C0	2 番目のエンコーダーセット - Excellent Smart Cube (ESC) 信号エラー
46	AL8d0	2 番目のエンコーダーセット - エンコーダー信号エラー
47	AL8E0	デジタルエンコーダーの切断
48	AL8F0	エクセレントスマートキューブ（ESC）内部エラー
49	AL861	モーターの過熱
50	ALb10	速度コマンド A/D コンバータエラー
51	-	予約
52	ALb20	トルクコマンド A/D コンバータエラー
53	ALb33	電流検出異常
54	ALC10	モーターが制御不能
55	ALC20	位相検出エラー
56	ALC21	ホールセンサーエラー
57	-	予約
58	ALC50	電気角度検出障害
59	ALC51	電気角検出中にオーバートラベルを検出
60	ALC52	電気角度検出が不完全
61	-	予約
62	ALd00	位置偏差オーバーフロー
63	-	予約
64	-	予約
65	ALd10	モーター負荷位置偏差オーバーフロー
66	-	予約
67	ALEb1	安全機能信号入力タイミングエラー

アラーム番号	名称	説明
68	ALEb2	安全機能モジュールエラー
69	ALF10	電源ケーブル欠相
70	ALF50	モーター主回路ケーブル断線
71	ALFA0	エンコーダー電源エラー
72	ALFB0	フィールドバス通信ハードウェアの故障
73	ALFB1	フィールドバス通信エラー
74	ALFC0	ガントリー制御システムの通信エラー
75	ALFC1	ガントリー制御システムのスレーブ軸エラー
76	AL891	エクセレントスマートキューブ（ESC） - インクリメンタルエンコーダー信号エラー
77	ALFC2	フィールドバス通信設定エラー

注記：

アラームの原因と対処方法については、「E1 シリーズドライバーユーザーズマニュアル」のセクション 13.2.2 を参照してください。

12. D シリーズドライバーエラー

エラーのリストは以下のとおりです。

表 12.1

エラー番号 (16 進数で表示)	名称
0x00000001	モーターショート（過電流）が検出されました
0x00000002	過電圧を検出しました
0x00000004	位置誤差が大きすぎます
0x00000008	エンコーダーエラー
0x00000010	ソフトサーマルしきい値に到達
0x00000020	モーターが切断されている可能性があります
0x00000040	アンプ過熱
0x00000080	モーター温度センサーが作動しました
0x00000100	低電圧が検出されました
0x00000200	エンコーダーカードの故障の場合は 5V
0x00000400	フェーズ初期化エラー
0x00000800	シリアルエンコーダー通信エラー
0x00001000	ホールセンサーエラー
0x00002000	ホール位相チェックエラー
0x00004000	電流制御エラー
0x00008000	-
0x00010000	ハイブリッド偏差が大きすぎる
0x00020000	STO アクティブ
0x00040000	HFLT 不一致エラー
0x00080000	アートフェーズセンターが完了していないエラー
0x00100000	-
0x00200000	DC バス電圧異常
0x00400000	EtherCAT インターフェースが検出されません
0x00800000	CiA-402 原点復帰エラー
0x01000000	ファン障害エラー
0x02000000	ドライブ過負荷エラー

注記：

エラーの原因と対処方法については、D シリーズドライバーユーザーマニュアルの関連章のエラーの説明を参照してください。

API ライブラリ リファレンスマニュアル
バージョン：V1.9　　2025 年 1 月改訂

1．HIWIN は HIWIN Mikrosystem Corp., HIWIN Technologies Corp., ハイウィン株式会社の登録商標です。ご自身の権利を保護するため、模倣品を購入することは避けてください。
2．実際の製品は、製品改良等に対応するため、このカタログの仕様や写真と異なる場合があります。
3．HIWIN は「貿易法」および関連規制の下で制限された技術や製品を販売・輸出しません。制限された HIWIN 製品を輸出する際には、関連する法律に従って、所管当局によって承認を受けます。また、核・生物・化学兵器やミサイルの製造または開発に使用することは禁じます。
