リファレンスマニュアル

PLCopen仕様
MC API関数

目　次

第４章 モーション制御パラメータ

第５章 付録

はじめに

この度は、アルゴシステム製品をお買い上げ頂きありがとうございます。
弊社製品を安全かつ正しく使用していただくために、お使いになる前に本書をお読みいただき、十分に理解していただくようお願い申し上げます。

１） 適用範囲

本書では、PLCopen 仕様 MC ライブラリの使用方法について説明します。

２） データタイプ

本書で使用するデータタイプとその範囲を表１に示します。

表 1. データタイプ

分類	データ型名	サイズ	値の範囲
BOOL	BOOL	1bit	TRUE or FALSE
整数	BYTE	1byte	0～255
	SINT		-128～128
	USINT		0～255
	INT8		-128～128
	UINT8		0～255
	WORD	2byte	0～65535
	INT		-32768～32767
	UINT		0～65535
	INT16		-32768～32767
	UINT16		0～65535
	DWORD	4byte	0～4294967295
	DINT		-2147483648～2147483647
	UDINT		0～4294967295
	INT32		-2147483648～2147483647
	UINT32		0～4294967295
浮動小数点	LREAL	8byte	-1.79769313486231e+308 ～ -2.22507385850720e-308、0、2.22507385850720e-308 ～ 1.79769313486231e+308、+ ∞ / - ∞

第１章　PLCopen MCライブラリ

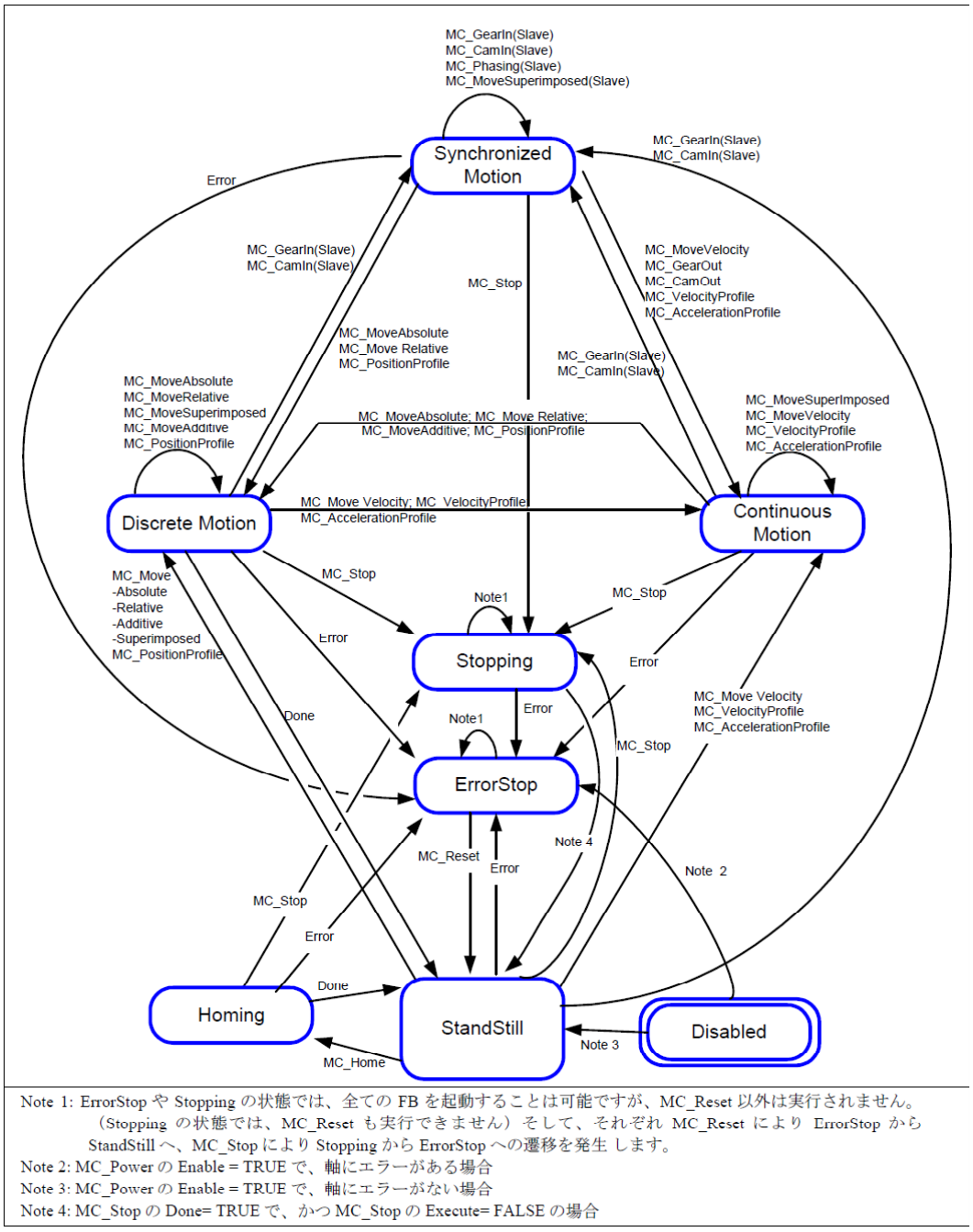
本章では MECHATROLINK-Ⅲ通信のサーボプロファイル、または、EtherCAT 通信のモーションコントロールのデバイスプロファイル（CiA402 デバイスプロファイル）を用いた『PLCopen 仕様 MC API 関数』の、基本的な仕様、構成について説明します。

１－１　PLCopen MC仕様とは

PLCopen とはプログラマブルコントローラ（PLC）のプログラミングの国際標準規格である IEC 61131-3 の普及促進・標準化推進団体であり、日本の主要メーカを含む世界 PLC 関連企業 46 社を含む 100 社以上が参加するワールド・ワイドな会員組織です。

この団体の規定するモーションコントロール仕様を PLCopen 仕様 MC と呼んでいます。

PLCopenのMC仕様では状態遷移が規定されており、この状態毎に実行可能なAPI関数が決まっています。この状態遷移図を図 1-1-1.　PLCopenMC状態遷移図に示します。



Note 1: ErrorStop や Stopping の状態では、全ての FB を起動することは可能ですが、MC_Reset 以外は実行されません。
（Stopping の状態では、MC_Reset も実行できません）そして、それぞれ MC_Reset により ErrorStop から StandStill へ、MC_Stop により Stopping から ErrorStop への遷移を発生 します。
Note 2: MC_Power の Enable = TRUE で、軸にエラーがある場合
Note 3: MC_Power の Enable = TRUE で、軸にエラーがない場合
Note 4: MC_Stop の Done= TRUE で、かつ MC_Stop の Execute= FALSE の場合

図 1-1-1. PLCopenMC状態遷移図

図 1-1-1に記載されていない以下のAPI関数については、軸の状態に影響しないため、状態を変化させずに実行する事が出来ます。
MC_ReadStatus; MC_ReadAxisError; MC_ReadParameter; MC_ReadBoolParameter; MC_ReadByteParameter; MC_ReadWordParameter; MC_ReadDwordParameter; MC_WriteParameter; MC_WriteBoolParameter; MC_WriteByteParameter; MC_WriteWordParameter; MC_WriteDwordParameter; MC_ReadActualPosition; MC_ReadActualVelocity; MC_CamTableSelect.

その他の詳細については、「技術仕様書 PLCopen - Technical Committee 2- 専門委員会 モーションコントロール用ファンクションブロック Version 1.1」を参照してください。

１－２　　全体構成

本ライブラリを使用した場合の全体構成図を、図 1-2-1、図 1-2-2に示します。

□　MECHATROLINK-Ⅲを使用した場合

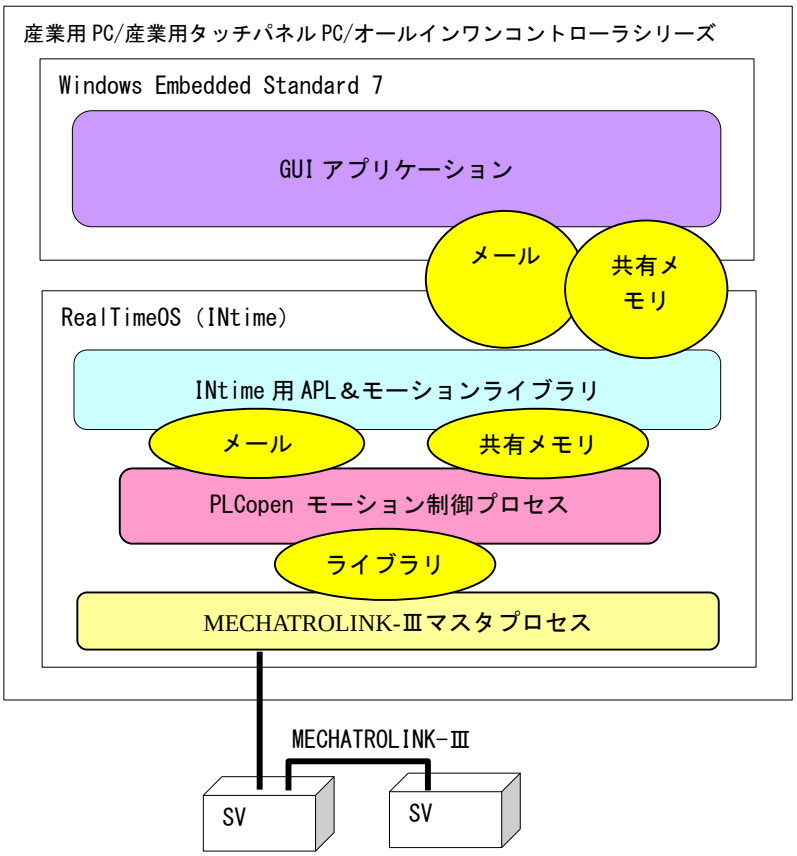


図 1-2-1.　全体構成図（MECHATROLINK-Ⅲ版）

Visual Studio 2008 で開発したプログラムを、実行します。
ProConOS から、PLCopen モーション制御プロセスへ命令が伝達されます。PLCopen モーション制御プロセスは MECHATROLINK-Ⅲマスタプロセスを通じてサーボパックを制御します。

□　EtherCAT を使用した場合

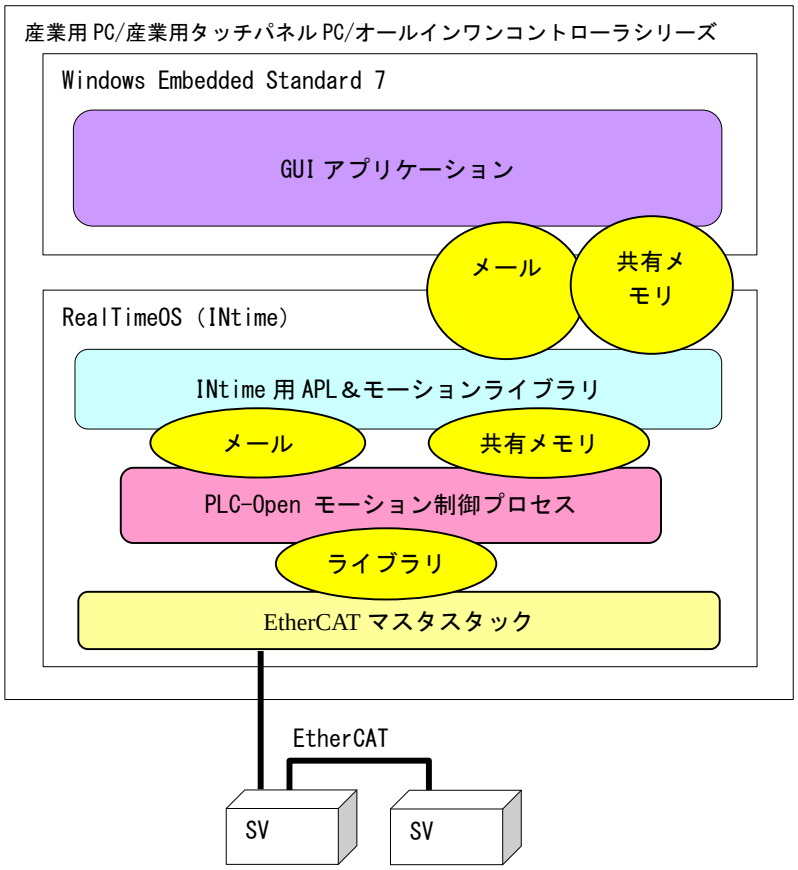


図 1-2-2.　全体構成図（EtherCAT版）

Visual Studio 2008 で開発したプログラムを、実行します。
ProConOS から、PLCopen モーション制御プロセスへ命令が伝達されます。PLCopen モーション制御プロセスは EtherCAT マスタスタック通じて、CiA402 準拠のサーボパックを制御します。

１－３　　MECHATROLINK-Ⅲ とは

１－３－１　概要

MECHATROLINK-Ⅲ通信とは、MECHATROLINK 協会の提唱するオープンな高速フィールドネットワークです。１台のコントローラで、複数のユニットを分散制御することが可能です。

MECHATROLINK-Ⅲの特徴は下記の通りです。

- サイクリック伝送による同期通信
- 100Mbps での高速伝送
- 伝送周期は接続局数、伝送データ量で最適値を選択可能（伝送周期 31.25us～64ms）
- 接続方法をカスケード形／スター形／Point to Point 形と装置に合わせた形で自由に構成可能
- MECHATROLINK 協会製「伝送 LSI」が、誤り検出と伝送周期内再送制御を含む伝送制御を行うため、FA コントローラの負荷低減が可能
- マスタとなるコントローラの他にサポートツールを接続可能

MECHATROLINK-Ⅲの接続形態は、C1 マスタ局が 1 局、スレーブ局が最大 62 局の Ethernet 接続によるネットワークシステムです。必要に応じて C2 マスタ局を 1 局接続できます。

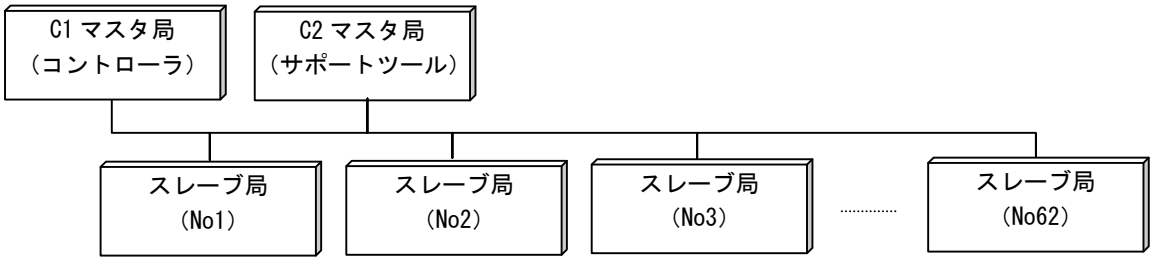


図 1-3-1.　MECHATROLINK-Ⅲ接続図

１－４　　EtherCATとは

この章では、EtherCAT ネットワーク通信の概要と CiA402 デバイスプロファイルについての仕様が簡単に記述されています。

お読みいただく方は、サーボアンプ、モーションコントロール、ネットワークと EtherCAT CoE(CANopen over EtherCAT)の基本的な知識を持つことを前提とします。

EtherCAT 仕様の詳細については、EtherCAT Technology Group から入手できます EtherCAT 仕様を参照いただくようにお願いします。

１－４－１　　概要

EtherCAT(Ethernet Control Automation Technology)は、Beckhoff 社により開発され、現在では EtherCAT Technology Group(ETG)により管理されています。

EtherCAT 接続は、新しいリアルタイムイーサネットを用いたネットワーク通信で、ツイストペア、または光ファイバケーブルで接続ができるとともに、ライン、ツリー、デイジーチェーン、ドロップラインをサポートします。

EtherCAT 転送方法はマスタから送信されたフレームがスレーブ通過時に出力データを取り出し、入力データを挿入します。Ethernet プロトコルは、IEEE802.3 に準拠した標準のイーサネットプロトコルが維持されていますので、新たにサブバスの構築は必要ありません。

EtherCAT プロトコルはプロセスデータ向けに最適化されています。EtherType により Ethernet フレーム内で直接転送されます。いくつかのサブ・テレグラムを構成しているかもしれませんが、それぞれ 4GB 容量までのロジック・プロセス・イメージを特定のメモリ・エリアに提供します。

１－４－２　　EtherCATプロファイル

EtherCAT は Ethernet をベースとしたネットワークの基本的な通信構造が定義されている IEC61158 の Section12 に定義されており、EtherCAT 通信プロファイルの EtherCAT ステートマシーン（ESM)、フィールドメモリ管理ユニット（FMMU）によるプロセスデータ通信方式、MailBox による CoE サービスチャンネル、シンクマネージャ（SM)、同期クロック方式による同期構造が説明されています。

ドライブおよびモーションコントロールのデバイスプロファイル（CiA402 デバイスプロファイル）は、サーボドライブ、正弦波インバータ、およびステッピングモーター用コントローラの機能動作を定義します。このプロファイルでは、複数の動作モードと対応する設定パラメータも規定されます。

この仕様には､状態ごとの内部および外部動作を規定する有限状態オートマトン(Finite State Automaton: FSA）も含まれます。受領されるコマンドや高出力を適用するかどうかは、ドライブの状態によって決まります。

状態はホストコントローラから受取るコントロールワードで変更されます。また、内部イベントによって変更することもできます。現在の状態はステータスワードで示されます。コントロールワードと各種コマンド値（速度など）はデフォルトの RxPDO（レシーブ PDO）にマッピングされます。ステータスワードと各種実査値（位置など）は TxPDO（トランスミット PDO）にマッピングされます。この規格には、すべてのドライブで使用できる汎用のデフォルト PDO と特定のドライブ（サーボドライブ、正弦波インバータ、ステッピングモーターなど）でのみ使用できるデフォルト PDO が用意されています。

オプション機能やパラメータが多いため、CiA 402 に準拠するデバイスは交換できない場合があります。

CiA 402 デバイスプロファイルは IEC 61800-7-201 および IEC 61800-7-301（（いずれも IEC から入手可能）で国際標準として定められています。

１－４－３　本システムでサポートできるサーボパック仕様

本システムでは、PLCopen プロセス処理で CiA402 準拠のサーボパック制御を行っています。本システムを使用するためには CiA402 仕様の中でサポートされていないと動作しない項目があります。以下に説明する項目がサポートされていないサーボパックについては本システムで動作させることはできません。

オペレーションモード

CiA402 では表 1-4-3-1に示すような動作モードが規定されています。本システムで必須となる動作モードは、プロファイル位置モード(pp)、プロファイル速度モード(pv)、ホーミングモード（hm）です。

表 1-4-3-1. CiA402 オペレーションモード一覧

オペレーションモード	記号	説明
プロファイル位置制御モード	pp	マスタはターゲット位置(0x607A)、プロファイル速度(0x6081)、プロファイル加減速度(0x6083,0x6084)を設定します。 スレーブはコントロールワード(0x6040)の bit4=1:NewSetpoint セットで軌道生成を行い目標位置へ移動します。
プロファイル速度制御モード	pv	マスタはターゲット速度(0x60FF)、プロファイル加減速度(0x6083,0x6084)を設定します。 オペレーションモードが切り替わった後、ターゲット速度まで加減速します。 軌道生成とパルス出力自体は、スレーブが行います。
ホーミングモード	hm	CiA402 で定義されたの原点復帰方法によって原点復帰を行います。 本システムで使用する原点復帰モードについては別項で説明します。
補間位置モード	ip	本システムでは、補間制御を行うことができません。
プロファイルトルクモード	tq	本システムでは、トルク制御ができません。
速度制御モード(インバータ等)	vl	本システムでは、インバータ等の速度制御モードには対応していません。
サイクル同期位置モード	csp	本システムでは、マスタ側で軌跡制御を行うことができません。
サイクル同期速度モード	csv	本システムでは、マスタ側で軌跡制御およびトルク制御ができません。
サイクル同期トルクモード	cst	本システムでは、マスタ側で軌跡制御およびトルク制御ができません。

必須CoEパラメータ

CiA402 で定義されているCoEパラメータのうち、本システムの制御時に使用しているパラメータについて、表 1-4-3-2に示します。

制御時に使用しているため、必須となります。
選択式です。システムパラメータでどちらを使うか選択できます。
必須ではありませんが、システムパラメータで設定している項目です。サーボパックにパラメータが無い場合は、システムパラメータで設定しても機能無効となります。

表 1-4-3-2. CiA402 CoEパラメーター覧

Index	Sub Index	Object Type	Name	Data Length	Dir	備考
0x6007	0x00	VAR	アボートコネクションオプションコード	INT16	RW	
0x603F	0x00	VAR	エラーコード	UINT16	RO	
0x6040	0x00	VAR	コントロールワード	UINT16	RW	
0x6041	0x00	VAR	ステータスワード	UINT16	RO	
0x605A	0x00	VAR	クイックストップオプションコード	INT16	RW	
0x605B	0x00	VAR	シャットダウンオプションコード	INT16	RW	
0x605C	0x00	VAR	ディセーブルオペレーションオプションコード	INT16	RW	
0x605D	0x00	VAR	ホールトオプションコード	INT16	RW	
0x605E	0x00	VAR	フォルトリアクションオプションコード	INT16	RW	
0x6060	0x00	VAR	オペレーションモード	INT8	RW	
0x6061	0x00	VAR	オペレーション表示	INT8	RO	
0x6062	0x00	VAR	指令位置	INT32	RO	
0x6063	0x00	VAR	内部実ポジション	INT32	RO	MC_ReadActualPosition 時の
0x6064	0x00	VAR	実ポジション（機械位置）	INT32	RO	読み出しパラメータ
0x6065	0x00	VAR	位置偏差ウインドウ	UINT32	RW	
0x6066	0x00	VAR	位置偏差過大タイムアウト	UINT16	RW	
0x6067	0x00	VAR	ポジションウインドウ（位置決め完了範囲）	UINT32	RW	
0x6068	0x00	VAR	ポジションウインドウタイム	UINT16	RW	
0x6069	0x00	VAR	実速度センサ値	INT32	RO	
0x606A	0x00	VAR	センサセレクションコード	INT16	RW	
0x606B	0x00	VAR	指令速度	INT32	RO	MC_ReadActualVelocity 時の
0x606C	0x00	VAR	実速度値（速度モニタ）	INT32	RO	読み出しパラメータ
0x606D	0x00	VAR	速度ウインドウ（速度一致範囲）	UINT16	RW	
0x606E	0x00	VAR	速度ウインドウタイム	UINT16	RW	
0x606F	0x00	VAR	速度スレッシュホールド	UINT16	RW	
0x6070	0x00	VAR	速度スレッシュホールドタイム	UINT16	RW	
0x6071	0x00	VAR	ターゲットトルク	INT16	RW	
0x6072	0x00	VAR	最大トルク	UINT16	RW	
0x6073	0x00	VAR	最大電流	UINT16	RW	
0x6074	0x00	VAR	指令トルク	INT16	RO	
0x6075	0x00	VAR	モータ定格電流	UINT32	RW	
0x6076	0x00	VAR	モータ定格トルク	UINT32	RW	
0x6077	0x00	VAR	実トルク値	INT16	RO	
0x6078	0x00	VAR	実電流値	INT16	RO	
0x6079	0x00	VAR	DC リンク回路電圧	UINT32	RO	
0x607A	0x00	VAR	ターゲット位置	INT32	RW	
0x607B	-	RECORD	ポジションレンジリミット	ー	ー	ー
	0x00	ー	エントリ数	UINT8	RO	No
	0x01	ー	最小位置レンジリミット	INT32	RW	Possible
	0x02	ー	最大位置レンジリミット	INT32	RW	Possible
0x607C	0x00	VAR	ホームオフセット	INT32	RW	Possible
0x607D	-	RECORD	ソフトウェア位置リミット値	ー	ー	ー
	0x00	ー	エントリ数	UINT8	RO	No
	0x01	ー	ソフトウェア最小位置リミット	INT32	RW	Possible
	0x02	ー	ソフトウェア最大位置リミット	INT32	RW	Possible

Index	Sub Index	Object Type	Name	Data Length	Dir	備考
0x607E	0x00	VAR	極性	UINT8	RO	
0x607F	0x00	VAR	最大プロファイル速度	UNIT32	RW	
0x6080	0x00	VAR	最大モータスピード	UINT32	RW	
0x6081	0x00	VAR	プロファイル速度	UNIT32	RW	
0x6082	0x00	VAR	エンド速度	UINT32	RW	
0x6083	0x00	VAR	プロファイル加速度	UINT32	RW	
0x6084	0x00	VAR	プロファイル減速度	UINT32	RW	
0x6085	0x00	VAR	クイックストップ減速度	UINT32	RW	MC_Stop 実行時の減速度として使用
0x6086	0x00	VAR	モーションプロファイルタイプ	INT16	RW	
0x6087	0x00	VAR	トルクスロープ	UINT32	RW	
0x6088	0x00	VAR	トルクプロファイルタイプ	INT16	RW	
0x608F	ー	RECORD	位置エンコーダ分解能	ー	ー	
	0x00	ー	エントリ数	UINT8	RO	
	0x01	ー	エンコーダ増分	UINT32	RW	
	0x02	ー	モータ回転数	UINT32	RW	
0x6090	ー	RECORD	速度エンコーダ分解能	ー	ー	
	0x00	ー	エントリ数	UINT8	RO	
	0x01	ー	エンコーダ増分/s	UINT32	RW	
	0x02	ー	モータ回転数/s	UINT32	RW	
0x6091	ー	RECORD	ギア比	ー	ー	
	0x00	ー	エントリ数	UINT8	RO	
	0x01	ー	エンコーダ増分	UINT32	RW	
	0x02	ー	シャフト回転数	UINT32	RW	
0x6092	ー	RECORD	フィード定数	ー	ー	
	0x00	ー	エントリ数	UINT8	RO	
	0x01	ー	フィード値	UINT32	RW	
	0x02	ー	シャフト回転数	UINT32	RW	
0x6098	0x00	VAR	ホーミング方式	INT8	RW	
0x6099	-	RECORD	ホーミング速度	ー	ー	
	0x00	ー	エントリ数	UINT8	RO	
	0x01	ー	スイッチサーチ速度	UINT32	RW	
	0x02	ー	ゼロサーチ速度	UINT32	RW	
0x609A	0x00	VAR	ホーミング加減速度	UINT32	RW	
0x60A3	0x00	VAR	プロファイルジャークユーズ	UINT8	RW	
0x60A4	ー	RECORD	プロファイルジャーク	ー	ー	
	0x00	ー	エントリ数	UINT8	RO	
	0x01	ー	プロファイルジャーク 1	UINT32	RW	
	0x02	ー	プロファイルジャーク 2	UINT32	RW	
	0x03	ー	プロファイルジャーク 3	UINT32	RW	
	0x04	ー	プロファイルジャーク 4	UINT32	RW	
	0x05	ー	プロファイルジャーク 5	UINT32	RW	
	0x06	ー	プロファイルジャーク 6	UINT32	RW	
0x60B0	0x00	VAR	位置オフセット（位置加算）	INT32	RW	
0x60B1	0x00	VAR	速度オフセット（速度加算）	INT32	RW	

Index	Sub Index	Object Type	Name	Data Length	Dir	備考
0x60B2	0x00	VAR	トルクオフセット（トルク加算）	INT16	RW	
0x60B8	0x00	VAR	タッチプローブ機能	UINT16	RW	
0x60B9	0x00	VAR	タッチプローブステータス	UINT16	RW	
0x60BA	0x00	VAR	タッチプローブ 1 位置立ち上がりエッジ	INT32	RW	
0x60BB	0x00	VAR	タッチプローブ 1 位置立ち下がりエッジ	INT32	RW	
0x60BC	0x00	VAR	タッチプローブ 2 位置立ち上がりエッジ	INT32	RW	
0x60BD	0x00	VAR	タッチプローブ 2 位置立ち下がりエッジ	INT32	RW	
0x60C0	0x00	VAR	補間サブモード選択		RW	
0x60C1	0x00	VAR	補間データレコード		RW	
0x60C2	―	RECORD	補間時間周期	―	―	
	0x00	―	エントリ数	UINT8	RO	
	0x01	―	補間時間単位	UINT8	RW	
	0x02	―	補間時間指数	INT8	RW	
0x60C4	―	RECORD	補間データ設定	―	―	
	0x00	―	エントリ数	UINT8	RO	
	0x01	―	最大バッファサイズ	UINT32	RW	
	0x02	―	実バッファサイズ	UINT32	RW	
	0x03	―	バッファ形式	BOOL	RW	
	0x04	―	バッファ位置	UINT16	RW	
	0x05	―	データサイズ	UINT8	RW	
	0x06	―	バッファクリア	BOOL	RW	
0x60C5	0x00	VAR	最大加速度	UINT32	RW	
0x60C6	0x00	VAR	最大減速度	UINT32	RW	
0x60E0	0x00	VAR	正方向トルクリミット値	UINT16	RW	
0x60E1	0x00	VAR	逆方向トルクリミット値	UINT16	RW	
0x60E2	0x00	VAR	モジュロ値	-	-	
0x60E3	―	RECORD	サポートホーミング方式	―	―	原点復帰時にホーミング方式がサポートしているか確認するときに使用。 パラメータが無い場合は、0x6098 にセットして原点復帰できるか確認している。
	0x00	―	エントリ数	UINT8	RO	
	0x01	―	サポートホーミング方式 1	UINT16	RO	
	0x02	―	サポートホーミング方式 2	UINT16	RO	
	0x03	―	サポートホーミング方式 7	UINT16	RO	
	0x04	―	サポートホーミング方式 8	UINT16	RO	
	0x05	―	サポートホーミング方式 11	UINT16	RO	
	0x06	―	サポートホーミング方式 12	UINT16	RO	
	0x07	―	サポートホーミング方式 23	UINT16	RO	
	0x08	―	サポートホーミング方式 27	UINT16	RO	
	0x09	―	サポートホーミング方式 35	UINT16	RO	
0x60E6	0x00	VAR	実位置計算方式	UINT8	RW	
0x60F2	0x00	VAR	位置オプションコード	UINT16	RW	
0x60F4	0x00	VAR	実位置偏差	INT32	RO	
0x60F8	0x00	VAR	最大偏差（最大ズレ量）	INT32	RW	
0x60FA	0x00	VAR	コントロールエフォート	INT32	RO	
0x60FC	0x00	VAR	内部位置コマンド値	INT32	RO	
0x60FD	0x00	VAR	デジタルインプット	UINT32	RO	
0x60FE	0x00	VAR	デジタルアウトプット	UINT32	RW	

Index	Sub Index	Object Type	Name	Data Length	Dir	備考
0x60FF	0x00	VAR	ターゲット速度	INT32	RW	
0x6402	0x00	VAR	モータタイプ	UINT16	RW	
0x6403	0x00	VAR	モータカタログナンバー	String	RW	
0x6404	0x00	VAR	モータ製造	String	RW	
0x6405	0x00	VAR	http モータカタログアドレス	String	RW	
0x6406	0x00	VAR	モータ校正日	Time of Day	RW	
0x6407	0x00	VAR	モータサービス期間	Time of Day	RW	
0x6502	0x00	VAR	サポートドライブモード	UINT32	RO	
0x6503	0x00	VAR	ドライブカタログナンバー	String	RW	
0x6505	0x00	VAR	http ドライバカタログアドレス	String	RW	

原点復帰シーケンス

PLCopenの原点復帰シーケンスは表 1-4-3-3で示されたAPI関数で行います。それぞれのAPI関数では、CiA402 に規定されている 32 種類の原点復帰シーケンスの中から、PLCopenで規定されている原点復帰シーケンスと一致したものを採用しています。

CiA402 の規定では、原点復帰シーケンス 32 種類すべてを対応する必要はないため、サーボパックメーカによってはサポートされていない原点復帰シーケンスがあります。サポートされていない原点復帰シーケンスを実行した場合はエラーとなります。

原点復帰動作の詳細については、『3-4-4 PLCopen仕様　原点復帰API関数』を参照してください。

表 1-4-3-3. PLC-Open原点復帰ファンクションブロックとCiA402 原点復帰シーケンス番号対応表

PLC-Open 原点復帰 ファンクションブロック	動作 方向	原点 エッジ	CiA402 原点復帰番号	備考
MC_StepAbsSwitch	+方向	OFF→ON	24	原点信号の左エッジで停止
		ON→OFF	26	原点信号の右エッジで停止
	-方向	OFF→ON	28	原点信号の右エッジで停止
		ON→OFF	30	原点信号の左エッジで停止
MC_StepLimitSwitch	+方向	ー	18	+EL 信号の ON→OFF エッジで停止
	-方向	ー	17	-EL 信号の ON→OFF エッジで停止
MC_StepRefPulse	+方向	ー	34	ZERO パルス信号の OFF→ON で停止
	-方向	ー	33	ZERO パルス信号の OFF→ON で停止
MC_StepDirect	ー	ー	35	現在位置を指定された値にセットする
MC_StepAbsolute	ー	ー	35	現在位置を原点位置とする
MC_StepBlock	未サポート			
MC_StepReferenceFlyingSwitch				
MC_StepReferenceFlyingRefPulse				
MC_AbortPassiveHoming				

第２章　開発環境

本章では、PLCopen MC ライブラリを使用するために必要な各種設定方法について説明します。

２－１　MECHATROLINK-Ⅲ動作環境

作成するアプリケーション内でPOMotion.RSLの関数をコールすることにより軸ユニットを制御することができます。
作成したアプリケーションと POMotion.RSL は同一フォルダ（ディレクトリ）に格納します。

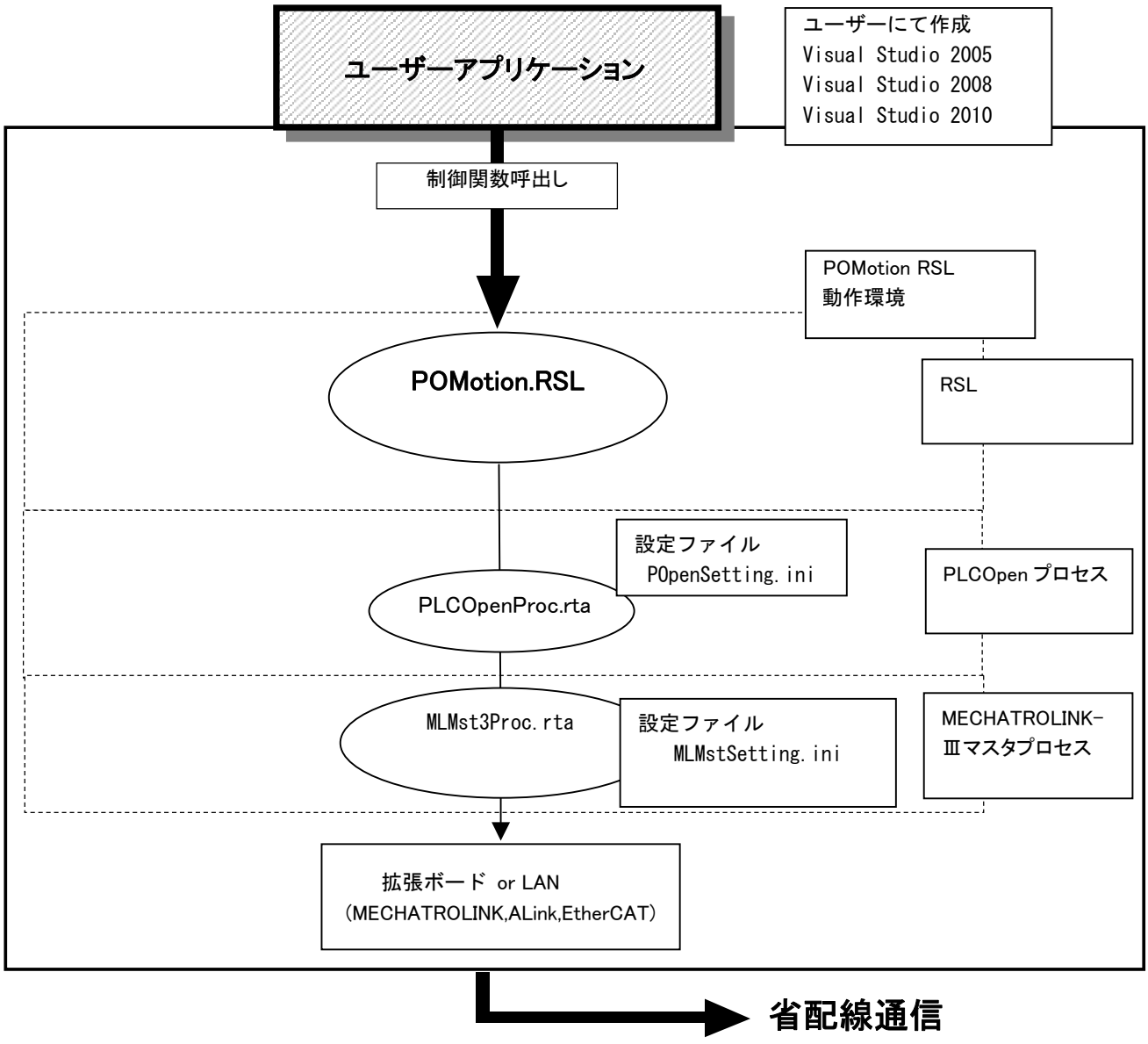


図 2-1-1. MECHATROLINK-Ⅲ構成図

２－２　EtherCAT動作環境

ユーザーは作成するアプリケーション内でPOMotion.RSLの関数をコールすることにより軸ユニットを制御することができます。
EtherCATマスタはシステムの情報としてconfig.xmlファイルを参照しますので、EtherCATマスタを使用したアプリケーションを動作させる前にconfig.xmlを作成する必要があります。
作成したアプリケーション、POMotion.RSL は同一フォルダ（ディレクトリ）に格納してアプリケーションを動作させます。config.xml は AcatProc.rta と同一フォルダ(ディレクトリ)に格納する必要があります。

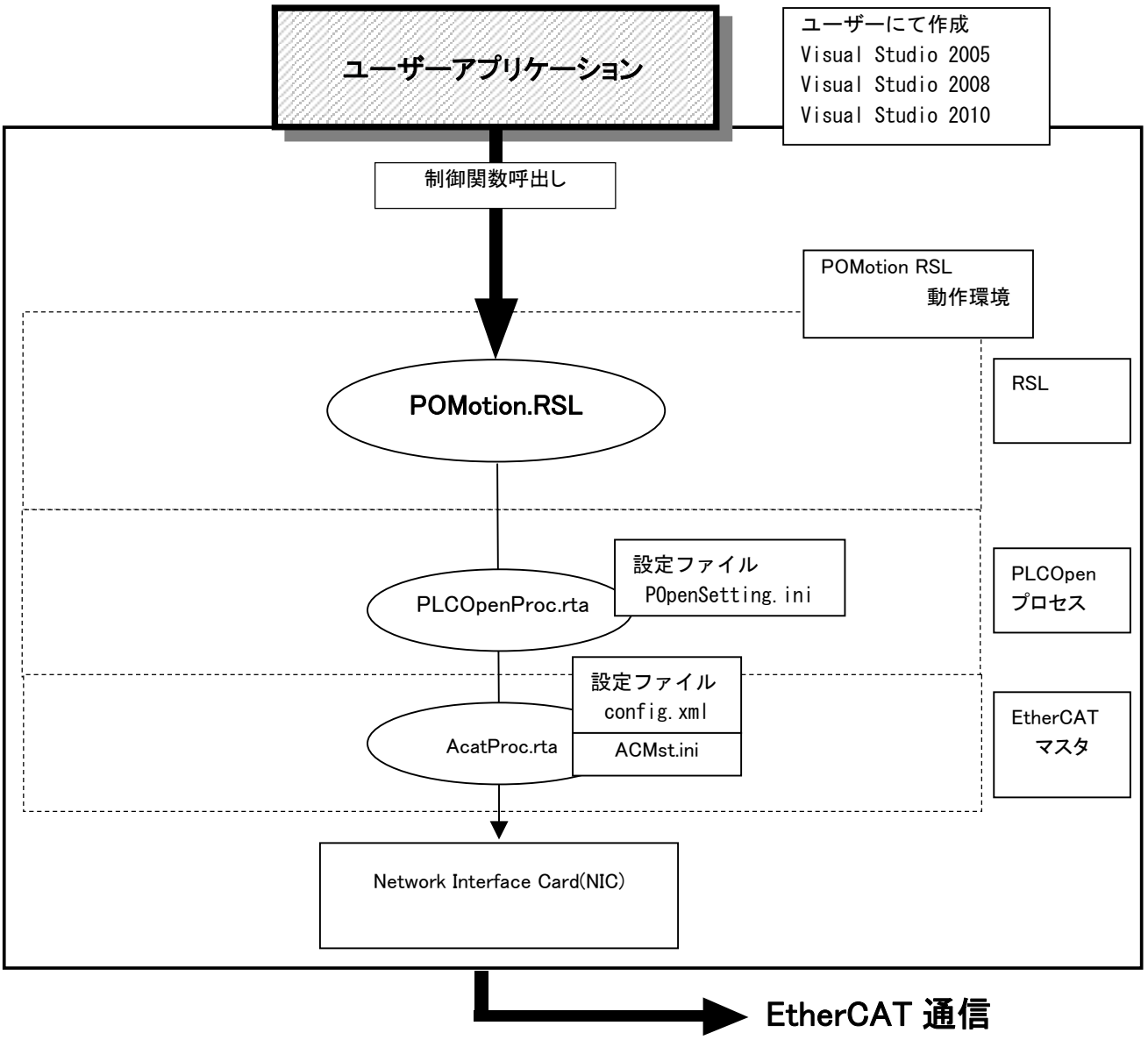


図 2-2-1. EtherCAT構成図

２－３　　アプリケーション開発の準備

開発アプリケーションから RSL をコールできるようにする為に、開発ユーザーは、下記の手順を実行します。

1）Microsoft Visual Studio 2005/2008/2010
プロジェクトのソースファイルがあるフォルダに、POMotion.CPP、POMotion.H、typeFbIO.H をコピーします。
RSL の関数をコールするソースファイルへ、POMotion.H をインクルードします。
プロジェクトへ POMotion.CPP を追加します。
プログラム起動時に実行する部分で、次の関数をコールして下さい。LoadPOMtnRsl(“POMotion.RSL”);
プログラム終了時に実行する部分で、次の関数をコールして下さい。UnLoadPOMtnRsl();

上記以外に C++のコンパイル設定で「プリコンパイルヘッダを使用しない」を指定して下さい。但し、「プリコンパイルヘッダを使用する」を指定する場合（ヘッダ指定が　stdafx.h　の時、つまりスイッチが /Yu"stdafx.h" の時）は、POMotion.cpp の上部 ヘッダ指定に次の１行を追加して下さい。

```
例：　　#include "stdafx.h"　　　＜－－追加行
　　#include "POMotion.h"
```

第３章　API関数

本章では、PLCopen 仕様のモーションコントロール API 関数を使用するために必要な内容について説明します。

３－１　　機能概要

C/C++言語を使って作成したアプリケーションで PLCopen に準拠した制御を実現するために、API 関数形式のライブラリを用意しました。各種コマンド毎の API 関数を使用することで、PLCopen 仕様モーションコントロールが可能となります。

API 関数はメインコマンド（共通）用、メインコマンド（サーボ動作）用、サブコマンド(原点復帰)用の３種類があります。

１）PLCopen 仕様 MC 初期化・終了処理 API 関数

PLCopen仕様で定義されているMC API関数の内、原点復帰用のAPI関数の一覧を表 3-1-1に示します。詳細は『3-4-1 PLCopen仕様　初期化・終了API関数』を参照してください。

表 3-1-1. PLCopen仕様MC初期化・終了処理API関数一覧

名称	API 関数名	サポート	PO_WaitForMotionRecv 関数呼び出しの必要性
ライブラリ内リソース生成	PO_Create	○	無し
ライブラリ内リソース破棄	PO_Destroy	○	無し
マスタプロセス処理の許可	PO_Open	○	無し
マスタプロセス処理の禁止	PO_Close	○	無し
マスタプロセス処理初期化命令	PO_ClearMstProc	○	無し
API 応答待ち処理	PO_WaitForMotionRecv	○	無し

管理・動作・原点復帰 API 関数は全て、関数に対する入力構造体ポインタと関数からの出力構造体ポインタの引数を持っています。

出力構造体ポインタに対して NULL を指定する事で非同期関数として機能しますが、非同期関数として使用する場合は、実行した関数の戻り値を使用して PO_ WaitForMotionRecv ()を実行し、結果を取得する必要があります。この待ち受け処理は、非同期関数として実行した際の戻り値全てに対して行う必要があります。

２）PLCopen 仕様 MC 管理 API 関数

PLCopen仕様で定義されているMC API関数の内、管理API関数の一覧を表 3-1-2に示します。詳細は『3-4-2 PLCopen仕様　管理API関数』を参照してください。

表 3-1-2. PLCopen仕様MC 管理API関数一覧

機能概略	API 関数名	サポート	制御軸	PO_WaitForMotionRecv 関数呼び出しの必要性
サーボ ON/OFF	MC_Power	○	単軸	無し
状態遷移ステータス読み込み	MC_ReadStatus	○	単軸	無し
軸エラー読み込み	MC_ReadAxisError	○	単軸	無し
軸パラメータ(LREAL 型)読み込み	MC_ReadParameter	○	単軸	無し
軸パラメータ(BOOL 型)読み込み	MC_ReadBoolParameter	○	単軸	無し
軸パラメータ(BYTE 型)読み込み	MC_ReadByteParameter	○	単軸	無し
軸パラメータ(WORD 型)読み込み	MC_ReadWordParameter	○	単軸	無し
軸パラメータ(DWORD 型)読み込み	MC_ReadDwordParameter	○	単軸	無し
軸パラメータ(LREAL 型)書き込み	MC_WriteParameter	○	単軸	無し
軸パラメータ(BOOL 型)書き込み	MC_WriteBoolParameter	○	単軸	無し
軸パラメータ(BYTE 型)書き込み	MC_WriteByteParameter	○	単軸	無し
軸パラメータ(WORD 型)書き込み	MC_WriteWordParameter	○	単軸	無し
軸パラメータ(DWORD 型)書き込み	MC_WriteDwordParameter	○	単軸	無し
現在位置読み込み	MC_ReadActualPosition	○	単軸	無し
現在速度読み込み	MC_ReadActualVelocity	○	単軸	無し
現在トルク読み込み	MC_ReadActualTorque	○	単軸	無し
エラーリセット	MC_Reset	○	単軸	無し
CAM 動作定義データ選択	MC_CamTableSelect	×	多軸	ー

○：仕様通りにサポート △：機能を限定してサポート ×：サポートしない
色が変わっている API 関数は、AI-Motion で追加されているものです。

３）PLCopen 仕様 MC 動作 API 関数

PLCopen仕様で定義されているMC API関数の内、動作API関数の一覧を表 3-1-3に示します。詳細は『3-4-3 PLCopen仕様　動作API関数』を参照してください。

表 3-1-3. PLCopen仕様MC 動作API関数一覧

名称	API 関数名	サポート	制御軸	PO_WaitForMotionRecv 関数呼び出しの必要性
絶対位置決め	MC_MoveAbsolute	○	単軸	有り
相対位置決め	MC_MoveRelative	○	単軸	有り
加算位置決め	MC_MoveAdditive	○	単軸	有り
位置決め中割り込み位置決め	MC_MoveSuperimposed	×	単軸	ー
定速動作	MC_MoveVelocity	○	単軸	有り
トルク制御	MC_TorqueControl	○	単軸	有り
原点復帰	MC_Home	×	単軸	ー
軸停止	MC_Stop	○	単軸	有り
位置データによる繰り返し	MC_PositionProfile	×	単軸	ー
速度データによる繰り返し	MC_VelocityProfile	×	単軸	ー
加速度データによる繰り返し	MC_AccelerationProfile	×	単軸	ー
主軸に対しての CAM 同期	MC_CamIn	×	多軸	ー
主軸からの CAM 同期解除	MC_CamOut	×	多軸	ー
主軸に対しての GEAR 同期	MC_GearIn	×	多軸	ー
主軸からの GEAR 同期解除	MC_GearOut	×	多軸	ー
主軸との位相同期	MC_Phasing	×	多軸	ー

○：仕様通りにサポート △：機能を限定してサポート ×：サポートしない

４）PLCopen 仕様 MC 原点復帰 API 関数

PLCopen仕様で定義されているMC API関数の内、原点復帰用のAPI関数の一覧を表 3-1-4に示します。詳細は『3-4-4 PLCopen仕様　原点復帰API関数』を参照してください。

表 3-1-4. PLCopen仕様MC 原点復帰API関数一覧

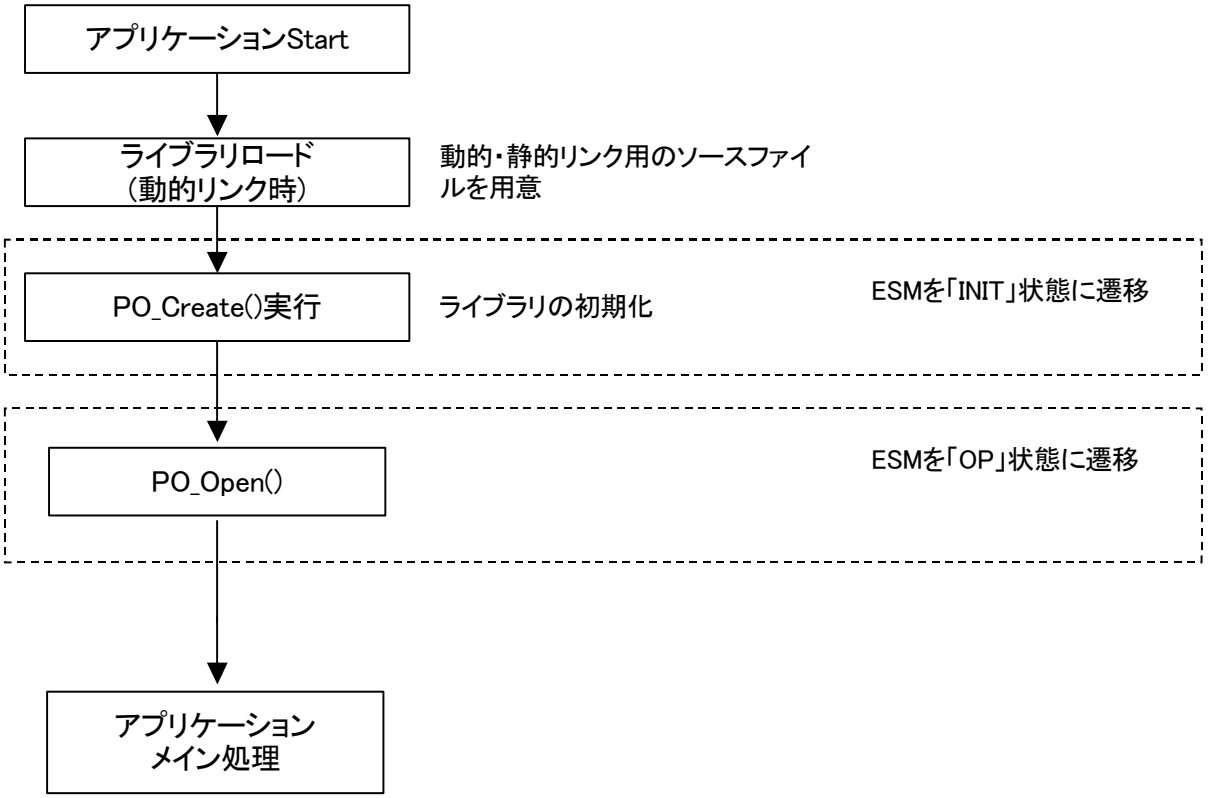
名称	API 関数名	サポート	制御軸	PO_WaitForMotionRecv 関数呼び出しの必要性
リミット/原点センサ使用	MC_StepAbsSwitch	○	単軸	有り
リミットセンサ使用	MC_StepLimitSwitch	○	単軸	有り
機械的限界	MC_StepBlock	×	単軸	ー
原点復帰終了	MC_FinishHoming	○	単軸	有り
Z 相検知による原点復帰	MC_StepRefPulse	○	単軸	有り
現在位置を指定して変更	MC_StepDirect	○	単軸	有り
現在位置を 0 位置に変更	MC_StepAbsolute	○	単軸	有り
位置決め中の SW 入力位置変更	MC_StepRefFlyingSwitc	×	単軸	ー
位置決め中の Z 相検知位置変更	MC_StepRefFlyingRefPulse	×	単軸	ー
原点復帰中断	MC_AbortPassiveHoming	×	単軸	ー

○：仕様通りにサポート △：機能を限定してサポート ×：サポートしない

３－２　　ライブラリ使用方法

３－２－１　アプリケーション開始

ライブラリを使用したアプリケーション開始のフローチャートを以下に示します。



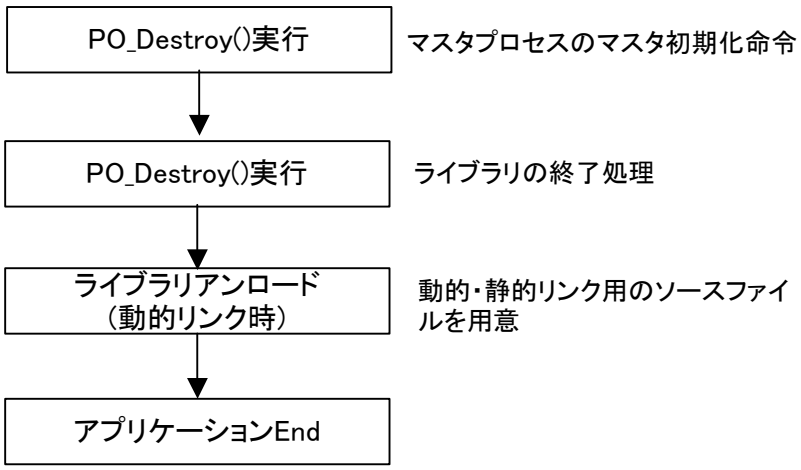
アルゴシステム製品のスレーブユニットのみを接続する場合は、各ユニット Open()関数を実行後、他のライブラリ関数を実行する事が出来ます。

他社製品との混在の場合は、他社製品の EtherCAT 仕様に基づいた状態遷移などが必要になります。
その場合は ACat_Mst_Set_State( )関数を用いて状態遷移を行います。

他社製品のみを接続する場合は、マスタアクセス関数を使用して EtherCAT 通信の開始、状態遷移などを行ってください。

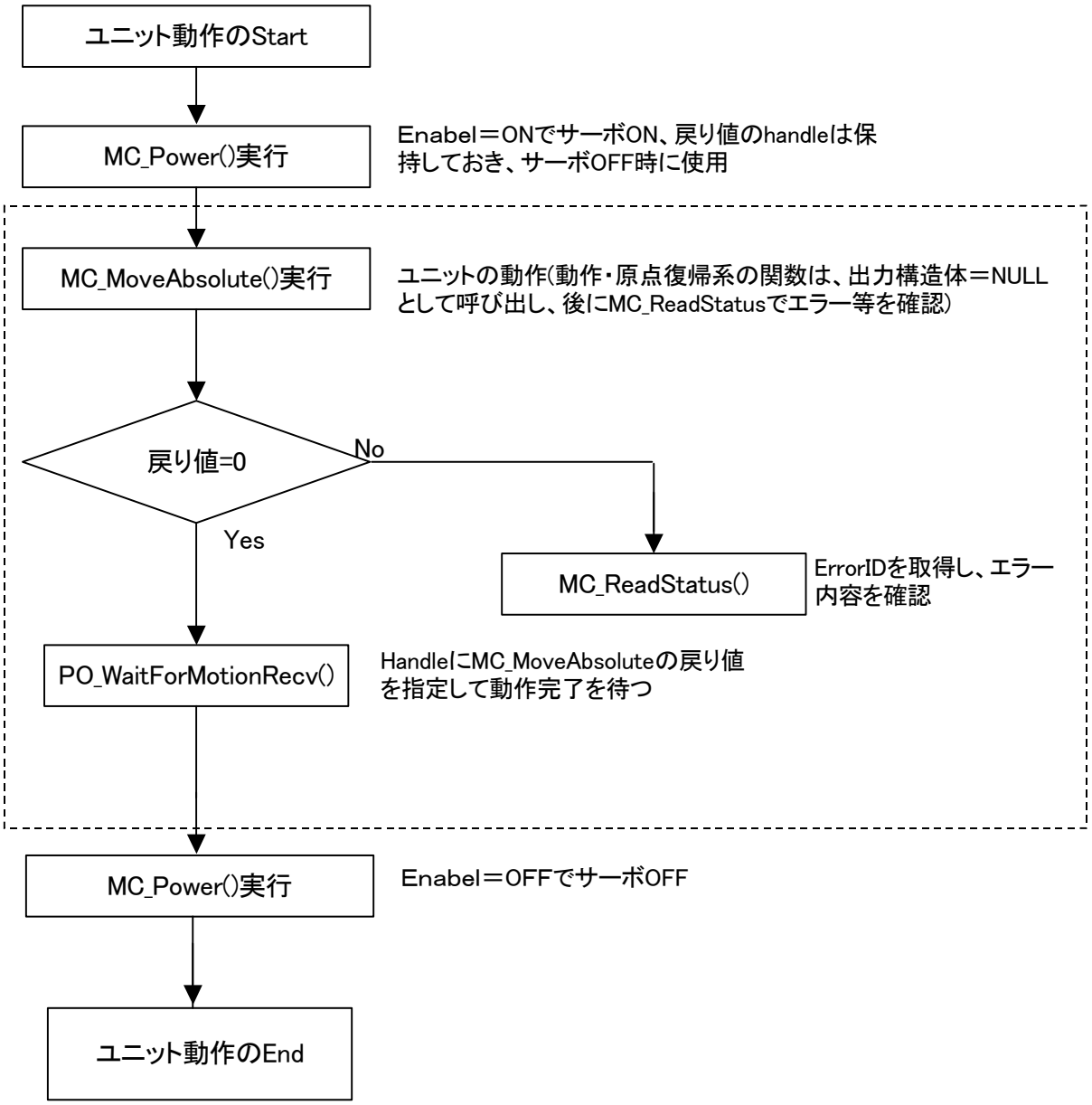
３－２－２　アプリケーション終了

ライブラリを使用したアプリケーション終了のフローチャートを以下に示します。



３－２－３　動作処理

ライブラリを使用したモーションユニットの動作のフローチャートを以下に示します。



３－３　　サンプルソース

C++ 用　デジタル入出力ユニットサンプル
RSL とのリンク部分とユニット制御関連のオープン部と実際の軸動作部分のみのサンプルを次に示します。

1）　RSL リンク

```
//--------------------------------------------------------------------------
//       RSL リンク
//--------------------------------------------------------------------------
  int ret;

  // RSL リンク(この関数はサンプルファイル POMotion.cpp/h に存在します)
  ret = LoadPOMtnRsl (POMTN_RSL_NAME);
  if (ret != E_OK) {
      printf("RSL LOAD 失敗 RET=%d", ret);
  }
  else {
      printf("RSL LOAD 完了");
  }

  // RSL 初期化
  ret = PO_Create ();
  if (ret != OK) {
      printf("RSL 初期化失敗 RET=%d", ret);
  }
  else {
      printf("RSL 初期化完了");
  }

//--------------------------------------------------------------------------
//      RSL リンク開放
//--------------------------------------------------------------------------
  int ret;

  // ライブラリ内リソース破棄
  ret = PO_ClearMstProc();
 if (ret != OK) {
      printf("リソース破棄失敗 RET=%d", ret);
  }
  else {
      printf("リソース破棄完了");
  }

  // RSL 終了
  ret = PO_Destroy ();
  if (ret != OK) {
      printf("RSL 終了失敗 RET=%d", ret);
```

```
  }
  else {
      printf("RSL 終了完了");
  }

  // RSL リンク開放(この関数はサンプルファイル POMotion.cpp/h に存在します)
  UnLoadPOMtnRsl();
```

2） ユニット制御オープン

```
//-------------------------------------------------------------------------------
//      モーションコントロールユニット オープン
//-------------------------------------------------------------------------------
  int ret;

  ret = PO_Open();                          // オープン
  if (ret) {
      printf("OPEN 失敗 RET=%d", ret);
  }
  else {
      printf("OPEN 完了");
  }

//-------------------------------------------------------------------------------
//      モーションコントロールユニット クローズ
//-------------------------------------------------------------------------------
  int ret;

  ret = PO_Close();                         // クローズ
  if (ret) {
      printf("CLOSE 失敗 RET=%d", ret);
  }
  else {
      printf("CLOSE 完了");
  }
```

3） デバイス入出力

```
//-------------------------------------------------------------------------------
//       サーボＯＮ(スレーブ ID=1)
//-------------------------------------------------------------------------------
int                 ret;
TFB_POWER_IN_LIB    power_in;
TFB_POWER_OUT       power_out;
RTHandle            h_power;

power_in.Axis               = 1;
power_in.BufferMode         = 0;
power_in.Enable             = 1;
```

```
power_in.Enable_Negative    = 0;
power_in.Enable_Positive    = 0;
p_smem_power.in.Power_hndle     = NULL;
h_power     = MC_Power(&power_in, &power_out);

if (!h_power) {
    printf("MC_Power 失敗 RET=%08x", power_out.ErrorID);
}
else {
    printf("MC_Power 完了: %08X", h_power);
}
//-------------------------------------------------------------------------------
//       絶対値移動(スレーブ ID=1)
//-------------------------------------------------------------------------------
int                 ret;
TFB_MVABS_IN_LIB    mvabs_in;
TFB_MVABS_OUT       mvabs_out;
RTHandle            h_mvabs;

mvabs_in.Axis             = 1;
mvabs_in.Execute          = 1;
mvabs_in.Position         = 2600000.0;
mvabs_in.Velocity         = 800000000.0;
mvabs_in.Acceleration     = 800000000.0;
mvabs_in.Deceleration     = 800000000.0;
mvabs_in.BufferMode       = 0;
mvabs_in.Direction        = 0;
mvabs_in.Jerk             = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, NULL);

if (!h_mvabs) {
    //エラー内容確認
    rdstatus_in.Axis          = 1;
    rdstatus_in.Enable        = 1;
    h_rdstatus = MC_ReadStatus(&rdstatus_in, &rdstatus_out);
    if(!h_rdstatus){
        printf("MC_MoveAbsolute 失敗 RET=%08x", rdstatus_out.ErrorID);
    }
}
else {
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
}
//動作完了待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_mvabs, &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
```

```
//-------------------------------------------------------------------------------
//       サーボ OFF(スレーブ ID=1)
//-------------------------------------------------------------------------------
int                 ret;
TFB_POWER_IN_LIB    power_in;
TFB_POWER_OUT       power_out;
RTHandle            h_power;

power_in.Axis               = 1;
power_in.BufferMode         = 0;
power_in.Enable             = 0;
power_in.Enable_Negative    = 0;
power_in.Enable_Positive    = 0;
p_smem_power.in.Power_hndle     = h_power;  //サーボON時に取得したハンドル
h_power     = MC_Power(&power_in, &power_out);

if (!h_power) {
    printf("MC_Power 失敗 RET=%d", power_out.ErrorID);
}
else {
    printf("MC_Power 完了: %08X", h_power);
}
```

３－４　　API関数リファレンス

本項では C 言語用に用意した PLCopen 仕様 MC API 関数を使用する為に必要な通信設定と各 MC API 関数について説明します。各 MC API 関数は PLCopen 仕様に従い作成されていますので、詳細については PLCopen が発行している技術仕様書「モーションコントロール用ファンクションブロック」なども参考にしてください。

３－４－１　　PLCopen仕様　初期化・終了API関数

本項では PLCopen MC 使用するために必要となる API 関数について説明します。

PO_Create 関数

機能 ライブラリ内リソースの生成を行います。

書式 int PO_Create (void)

引数 なし

戻り値

1	: 正常
0	: 異常

説明 ライブラリ内にある関数の初期化を行います。
本関数コール後、Open 関数をコールすることで PLCopen の機能 API を実行できるようになります。
アプリケーション起動時には、必ずコールする必要があります。

PO_Destroy 関数

機能 ライブラリ内リソースの破棄を行います。

書式 int PO_Destroy (void)

引数 なし

戻り値

1	: 正常
0	: 異常

説明 ライブラリ内にある関数の終了処理を行います。
この関数コール後は、PLCopen 機能 API を使用することはできません。
アプリケーションの終了時に必ずコールする必要があります。

PO_Open 関数

機能 マスタプロセス処理の許可を行います。

書式 int PO_Open (void)

引数 なし

戻り値

1	: 正常
0	: 異常

説明 マスタプロセス処理を許可します。
PLCopen 機能 API を使用するには、必ずコールする必要があります。

PO_Close 関数

機能 マスタプロセス処理の禁止を行います。

書式 int PO_Close (void)

引数 なし

戻り値

1	: 正常
0	: 異常

説明 マスタプロセス処理を禁止します。
アプリケーションの終了時に必ずコールする必要があります。

PO_ClearMstProc 関数

機能 マスタプロセスへのマスタ初期化命令を行います。

書式 int PO_ ClearMstProc (void)

引数 なし

戻り値

1	: 正常
0	: 異常

説明 マスタプロセスの初期化を行います。
アプリケーションの終了時に必ずコールする必要があります。

PO_WaitForMotionRecv 関数

機能　PLCopen 仕様 API 関数の応答待ち処理を行います。

書式

```
int  PO_WaitForMotionRecv (
        RTHANDLE            hndl,
        void                *out_dat,
        unsigned long       timeout
)
```

引数

hndl	: 応答待ち関数のハンドル
out_dat	: 出力用構造体(応答する関数により異なります)
timeout	: タイムアウト時間(×10msec) 0 の場合、無限待ち

戻り値

1	: 正常
0	: 異常

説明

非同期呼び出し可能 API 関数における完了応答を待ちます。
出力内容を確認する場合は、hndl で指定した関数の出力構造体に置き換えて参照してください。
本 API 関数により完了応答(Done=TRUE or Error=TRUE or CommandAborted=TRUE)を受けたハンドルは無効になります。
完了応答の確認後は、このハンドルを使用した応答待ち処理は行わないようにしてください。

３－４－２　　PLCopen仕様　管理API関数

本項では PLCopen MC 使用に定義されている管理系の API 関数について説明します。PLCopen 仕様の MC では状態遷移が定義されていますが、管理系の API 関数の多くは状態遷移上の状態に関わらず実行する事が可能になっています。

MC_Power 関数

機能 運転の可否を制御します。

書式
```
RTHANDLE  MC_Power (
        TFB_POWER_IN_LIB        *pwr_in,
        TFB_POWER_OUT           *pwr_out
)
```

引数
pwr_in : 入力用構造体
pwr_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明
本 API 関数は、入力用構造体内の Enable（有効）を TRUE にすると、Axis で指定された軸がサーボ ON され運転可能状態となります。
Enable（有効）を False にすると、Axis で指定された軸がサーボ OFF され運転可能状態を解除します。運転可能状態を解除した場合、軸は動作指令を受け付けず、軸制御ができません。

運転可能状態でない軸に対して、動作指令を実行した場合エラーとなります。
ただし、運転可能解除状態でも、MC_Power と MC_Reset は実行可能です。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :運転可能状態 FALSE:運転可能状態を解除
Enable_Positive	正転駆動許可	BOOL	TRUE or FALSE	FALSE	未サポート
Enable_Negative	逆転駆動許可	BOOL	TRUE or FALSE	FALSE	未サポート
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定 **本 API 関数では、0:Aborting のみサポート**

型名	型	値	モード	説明
MC_BufferMode	UINT16	0	Aborting	実行中の API 関数に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

出力用構造体詳細

変数	名称	データ型	範囲	内容
Status	運転可	BOOL	TRUE or FALSE	運転可能状態になったとき TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中のときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

MC_ReadStatus 関数

機能 現在処理中のモーション動作に対する軸の詳細ステータスを取得します。

書式
```
RTHANDLE  MC_ReadStatus (
        TFB_RDSTATUS_IN_LIB     *rds_in,
        TFB_RDSTATUS_OUT        *rds_out
)
```

引数
rds_in : 入力用構造体
rds_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 現在処理中のモーション動作に対する軸の状態遷移ステータスを出力します。
ConstantVelocity、Accelerating、Decelerating については MC_ReadActualVelocity と同じルーチンで、現在速度を読み出し、システム周期ごとに現在速度と前回の速度を比較して確認しています。
MC_ReadActualVelocity の読み出し速度が安定していない場合、誤動作する可能性がありますので、位置ループゲイン等のサーボパック固有パラメータの調整をしてください。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し

出力用構造体詳細

変数	名称	データ型	範囲	内容
Valid	出力有効	BOOL	TRUE or FALSE	各読み込みデータが有効の間 TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力
ErrorStop	ErrorStop 状態	BOOL	TRUE or FALSE	現在の軸状態を出力 いずれかの状態のみが TRUE となり、2 つ以上が同時に TRUE になることはありません。
Disabled	Disabled 状態	BOOL	TRUE or FALSE	
Stopping	Stopping 状態	BOOL	TRUE or FALSE	
StandStill	StandStill 状態	BOOL	TRUE or FALSE	
Discrete Motion	Discrete Motion 状態	BOOL	TRUE or FALSE	
Continuous Motion	Continuous Motion 状態	BOOL	TRUE or FALSE	
Synchronized Motion	Synchronized Motion 状態	BOOL	TRUE or FALSE	
Homing	Homing 状態	BOOL	TRUE or FALSE	
ConstantVelocity	定速動作中	BOOL	TRUE or FALSE	軸が定速動作中のとき TRUE
Accelerating	加速中	BOOL	TRUE or FALSE	軸が加速中のとき TRUE
Decelerating	減速中	BOOL	TRUE or FALSE	軸が減速中のとき TRUE

※1：エラーコード一覧を参照

MC_ReadAxisError 関数

機能 一般的な軸エラーを取得します。

書式
```
RTHANDLE  MC_ReadAxisError (
        TFB_RDERROR_IN_LIB      *rde_in,
        TFB_RDERROR_OUT         *rde_out
)
```

引数
rde_in : 入力用構造体
rde_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 軸のエラーコードを取得します。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し

出力構造体詳細

変数	名称	データ型	範囲	内容
Valid	出力有効	BOOL	TRUE or FALSE	各読み込みデータが有効の間 TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力
AxisErrorID	サーボパック エラー番号	DWORD	※1	サーボパック側でエラーが発生した場合はサーボパックエラーを出力

※1：エラーコード一覧を参照

MC_ReadParameter 関数

機能 LREAL 型パラメータを取得します。

書式
```
RTHANDLE  MC_ReadParameter (
        TFB_RDPARAM_IN_LIB      *rdp_in,
        TFB_RDPARAM_OUT         *rdp_out
)
```

引数
```
rdp_in              : 入力用構造体
rdp_out             : 出力用構造体
```

戻り値
```
0 以外              : 正常(RtHandle 値)
0                   : 異常
```

説明 ParameterNumber で指定された、パラメータ（LREAL 型）の値を取得します。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
ParameterNumber	パラメータ番号	DWORD	※1	0	読み込みパラメータ番号

※1：パラメーター覧を参照

出力構造体詳細

変数	名称	データ型	範囲	内容
Valid	出力有効	BOOL	TRUE or FALSE	各読み込みデータが有効の間 TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※2	異常が発生したときのエラーコードを出力
Value	読み出しパラメータ	LREAL	ー	読み出した値

※2：エラーコード一覧を参照

MC_ReadBoolParameter 関数

機能 BOOL 型パラメータを取得します。

書式
```
RTHANDLE  MC_ReadBoolParameter (
        TFB_RDBOOL_IN_LIB       *rdb_in,
        TFB_RDBOOL_OUT          *rdb_out
)
```

引数
rdb_in : 入力用構造体
rdb_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 ParameterNumber で指定された、パラメータ（BOOL 型）の値を取得します。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
ParameterNumber	パラメータ番号	DWORD	※1	0	読み込みパラメータ番号

※1：パラメーター覧を参照

出力構造体詳細

変数	名称	データ型	範囲	内容
Valid	出力有効	BOOL	TRUE or FALSE	各読み込みデータが有効の間 TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※2	異常が発生したときのエラーコードを出力
Value	読み出し パラメータ	BOOL	TRUE or FALSE	読み出した値

※2：エラーコード一覧を参照

MC_ReadByteParameter 関数

機能 BYTE 型パラメータを取得します。

書式
```
RTHANDLE  MC_ReadByteParameter (
        TFB_RDBYTE_IN_LIB       *rdb_in,
        TFB_RDBYTE_OUT          *rdb_out
)
```

引数
rdb_in : 入力用構造体
rdb_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 ParameterNumber で指定された、パラメータ（BYTE 型）の値を取得します。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
ParameterNumber	パラメータ番号	DWORD	※1	0	読み込みパラメータ番号

※1：パラメーター覧を参照

出力構造体詳細

変数	名称	データ型	範囲	内容
Valid	出力有効	BOOL	TRUE or FALSE	各読み込みデータが有効の間 TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※2	異常が発生したときのエラーコードを出力
Value	読み出しパラメータ	BYTE	ー	読み出した値

※2：エラーコード一覧を参照

MC_ReadWordParameter 関数

機能 WORD 型パラメータを取得します。

書式
```
RTHANDLE  MC_ReadWordParameter (
        TFB_RDWORD_IN_LIB       *rdw_in,
        TFB_RDWORD_OUT          *rdw_out
)
```

引数
rdw_in : 入力用構造体
rdw_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 ParameterNumber で指定された、パラメータ（WORD 型）の値を取得します。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
ParameterNumber	パラメータ番号	DWORD	※1	0	読み込みパラメータ番号

※1：パラメーター覧を参照

出力構造体詳細

変数	名称	データ型	範囲	内容
Valid	出力有効	BOOL	TRUE or FALSE	各読み込みデータが有効の間 TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※2	異常が発生したときのエラーコードを出力
Value	読み出し パラメータ	WORD	ー	読み出した値

※2：エラーコード一覧を参照

MC_ReadDwordParameter 関数

機能 DWORD 型パラメータを取得します。

書式
```
RTHANDLE  MC_ReadDwordParameter (
        TFB_RDDWORD_IN_LIB      *rdd_in,
        TFB_RDDWORD_OUT         *rdd_out
)
```

引数
rdd_in : 入力用構造体
rdd_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 ParameterNumber で指定された、パラメータ（DWORD 型）の値を取得します。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
ParameterNumber	パラメータ番号	DWORD	※1	0	読み込みパラメータ番号

※1：パラメーター覧を参照

出力構造体詳細

変数	名称	データ型	範囲	内容
Valid	出力有効	BOOL	TRUE or FALSE	各読み込みデータが有効の間 TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※2	異常が発生したときのエラーコードを出力
Value	読み出しパラメータ	DWORD	ー	読み出した値

※2：エラーコード一覧を参照

MC_WriteParameter 関数

機能 LREAL 型パラメータを書き込みます。

書式
```
RTHANDLE  MC_WriteParameter (
        TFB_WRPARAM_IN_LIB      *wrp_in,
        TFB_WRPARAM_OUT         *wrp_out
)
```

引数
wrp_in : 入力用構造体
wrp_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 ParameterNumber で指定された、パラメータ（LREAL 型）の値を書き込みます。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
ParameterNumber	パラメータ番号	DWORD	※1	0	書き込みパラメータ番号
Value	設定値	LREAL	ー	0	書き込みパラメータ値

※1：パラメーター覧を参照

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	書き込み完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※2	異常が発生したときのエラーコードを出力

※2：エラーコード一覧を参照

MC_WriteBoolParameter 関数

機能 BOOL 型パラメータを書き込みます。

書式
```
RTHANDLE  MC_WriteBoolParameter (
        TFB_WRBOOL_IN_LIB       *wrb_in,
        TFB_WRBOOL_OUT          *wrb_out
)
```

引数
wrb_in : 入力用構造体
wrb_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 ParameterNumber で指定された、パラメータ（BOOL 型）の値を書き込みます。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
ParameterNumber	パラメータ番号	DWORD	※1	0	書き込みパラメータ番号
Value	設定値	BOOL	TRUE or FALSE	FALSE	書き込みパラメータ値

※1：パラメーター覧を参照

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	書き込み完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※2	異常が発生したときのエラーコードを出力

※2：エラーコード一覧を参照

MC_WriteByteParameter 関数

機能 BYTE 型パラメータを書き込みます。

書式
```
RTHANDLE  MC_WriteByteParameter (
        TFB_WRBYTE_IN_LIB       *wrb_in,
        TFB_WRBYTE_OUT          *wrb_out
)
```

引数
wrb_in : 入力用構造体
wrb_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 ParameterNumber で指定された、パラメータ（BYTE 型）の値を書き込みます。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
ParameterNumber	パラメータ番号	DWORD	※1	0	書き込みパラメータ番号
Value	設定値	BYTE	ー	0	書き込みパラメータ値

※1：パラメーター覧を参照

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	書き込み完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※2	異常が発生したときのエラーコードを出力

※2：エラーコード一覧を参照

MC_WriteWordParameter 関数

機能 WORD 型パラメータを書き込みます。

書式
```
RTHANDLE  MC_WriteWordParameter (
        TFB_WRWORD_IN_LIB       *wrw_in,
        TFB_WRWORD_OUT          *wrw_out
)
```

引数
wrw_in : 入力用構造体
wrw_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 ParameterNumber で指定された、パラメータ（WORD 型）の値を書き込みます。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
ParameterNumber	パラメータ番号	DWORD	※1	0	書き込みパラメータ番号
Value	設定値	WORD	ー	0	書き込みパラメータ値

※1：パラメーター覧を参照

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	書き込み完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※2	異常が発生したときのエラーコードを出力

※2：エラーコード一覧を参照

MC_WriteDwordParameter 関数

機能 DWORD 型パラメータを書き込みます。

書式
```
RTHANDLE  MC_WriteDwordParameter (
        TFB_WRDWORD_IN_LIB      *wrd_in,
        TFB_WRDWORD_OUT         *wrd_out
)
```

引数
wrd_in : 入力用構造体
wrd_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 ParameterNumber で指定された、パラメータ（DWORD 型）の値を書き込みます。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
ParameterNumber	パラメータ番号	DWORD	※1	0	書き込みパラメータ番号
Value	設定値	DWORD	ー	0	書き込みパラメータ値

※1：パラメーター覧を参照

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	書き込み完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※2	異常が発生したときのエラーコードを出力

※2：エラーコード一覧を参照

MC_ReadActualPosition 関数

機能 現在位置を取得します。

書式
```
RTHANDLE  MC_ReadActualPosition (
        TFB_RDPOS_IN_LIB        *rap_in,
        TFB_RDPOS_OUT           *rap_out
)
```

引数
rap_in : 入力用構造体
rap_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 軸の現在位置を絶対座標で取得します。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し

出力構造体詳細

変数	名称	データ型	範囲	内容
Valid	出力有効	BOOL	TRUE or FALSE	各読み込みデータが有効の間 TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力
Position	現在位置	LREAL	倍精度実数値 【指令単位】	現在位置を絶対座標で出力

※1：エラーコード一覧を参照

MC_ReadActualVelocity 関数

機能 現在速度を取得します。

書式
```
RTHANDLE  MC_ReadActualVelocity (
        TFB_RDVEL_IN_LIB        *rav_in,
        TFB_RDVEL_OUT           *rav_out
)
```

引数
rav_in : 入力用構造体
rav_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 軸の現在速度を取得します。正の値なら正転、負の値なら逆転、0 なら停止中となります。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し

出力構造体詳細

変数	名称	データ型	範囲	内容
Valid	出力有効	BOOL	TRUE or FALSE	各読み込みデータが有効の間 TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力
Velocity	現在速度	LREAL	倍精度実数値 【指令単位/s】	現在速度を出力

※1：エラーコード一覧を参照

MC_ReadActualTorque 関数

機能 現在トルクを取得します。

書式
```
RTHANDLE  MC_ReadActualTorque(
    TFB_RDTRQ_IN_LIB        *rat_in,
    TFB_RDTRQ_OUT           *rat_out
)
```

引数
rat_in : 入力用構造体
rat_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 軸の現在トルクを取得します。正の値なら正転、負の値なら逆転、0 なら停止中となります。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し

出力構造体詳細

変数	名称	データ型	範囲	内容
Valid	出力有効	BOOL	TRUE or FALSE	各読み込みデータが有効の間 TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力
ActualTorque	現在トルク	LREAL	倍精度実数値 【N.m】or【%】	現在トルクを出力 単位はサーボパックにより異なる。<※2>

※1：エラーコード一覧を参照
※2：MECHATROLINK-Ⅲの場合 ：サーボパラメータ「トルク単位選択 (0x47)」の値により変わります。
EtherCAT の場合 ：サーボパックマニュアルの CiA402 パラメータ「目標トルク (0x6071)」または、「内部指令トルク (0x6074)」の単位を参照してください。

MC_Reset 関数

機能 指定した軸に関するエラーをリセットします。

書式
```
RTHANDLE  MC_Reset (
        TFB_RESET_IN_LIB        *rst_in,
        TFB_RESET_OUT           *rst_out
)
```

引数
rst_in : 入力用構造体
rst_out : 出力用構造体

戻り値
0 以外 : 正常(RtHandle 値)
0 : 異常

説明 軸でエラーが発生し、ErrorStop 状態に移行したとき、本 API 関数を実行することで、StandStill 状態へ復帰します。
サーボパックで発生したエラーについては、エラーリセット処理が実行されます。
通信異常等の外的要因で発生したエラーについては、解除できない場合があります。エラー内容からエラー要因を取り除いた上で実行してください。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	リセット完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

MC_CamTableSelect 関数

機能 未サポート

書式
```
RTHANDLE  MC_CamTableSelect (
        TFB_CAMTBL_IN_LIB       *cts_in,
        TFB_CAMTBL_OUT          *cts_out
)
```

引数

cts_in	: 入力用構造体
cts_out	: 出力用構造体

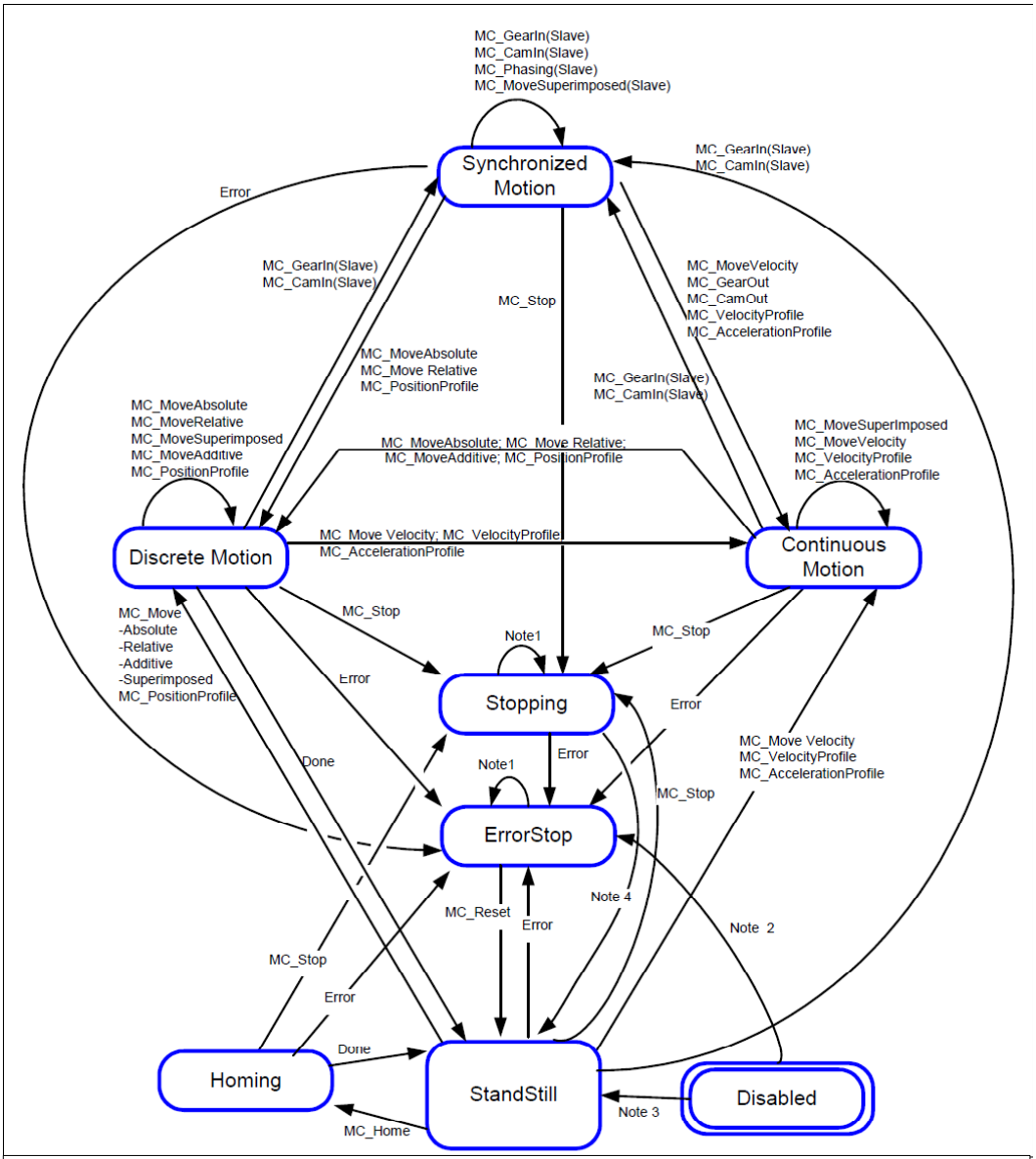
戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

３－４－３　　PLCopen仕様　動作API関数

本項ではPLCopen MC使用に定義されている動作系のAPI関数について説明します。本項で説明しますAPI関数は、PLCopenに定義されている状態遷移に従い動作を行います（図 3-4-3-1）。



Note 1:ErrorStop や Stopping の状態では、全ての API 関数をｺｰﾙすることは可能ですが、MC_Reset 以外は実行されません。
（Stopping の状態では、MC_Reset も実行できません）そして、それぞれ MC_Reset により ErrorStop から
StaindStill へ、MC_Stop により Stopping から ErrorStop への遷移を発生します。
Note 2:MC_Power の Enabel=TRUE で、軸にエラーがある場合
Note 3:MC_Power の Enable=TRUE で、軸にエラーがない場合
Note 4:MC_Stop の Done=TRUE で、かつ MC_Stop の Execute=FALSE の場合

図 3-4-3-1. PLCopenMC状態遷移図

MC_MoveAbsolute 関数

機能　絶対位置による位置決めを実行します。

書式
```
RTHANDLE  MC_MoveAbsolute (
        TFB_MVABS_IN_LIB        *abs_in,
        TFB_MVABS_OUT           *abs_out,
        unsigned long           timeout
)
```

引数

abs_in	: 入力用構造体
abs_out	: 出力用構造体
timeout	: タイムアウト時間 (x10msec) 0 の場合、無限待ち

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　絶対位置による位置決めを実行します。位置決め完了は、目標位置に対して設定された位置決め完了幅の範囲に到達する事で完了します。

非同期実行(出力用構造体=NULL による API 関数実行)時は、timeout は無効です。
終了待ちは PO_WaitForMotionRecv()関数を使用してください。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
Position	目標位置	LREAL	倍精度実数値 【指令単位】	0	位置決め目標位置を絶対位置で指定
Velocity	移動速度	LREAL	0,倍精度実数値正数 【指令単位/s】	0	位置決め時の速度
Acceleration	加速度	LREAL	倍精度実数値正数 【指令単位/s^2】	0	位置決め時の加速度
Deceleration	減速度	LREAL	倍精度実数値正数 【指令単位/s^2】	0	位置決め時の減速度
Jerk	加加速度	LREAL	ー	ー	未サポート
Direction	動作方向	MC_Direction	ー	ー	未サポート
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定

型名	型	値	モード	説明
MC_BufferMode	UINT16	0	Aborting	実行中の API 関数に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	位置決め完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中に TRUE
CommandAborted	実行中断	BOOL	TRUE or FALSE	命令が中止されたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

MC_MoveRelative 関数

機能 相対位置による位置決めを実行します。

書式
```
RTHANDLE  MC_MoveRelative (
        TFB_MVREL_IN_LIB        *rel_in,
        TFB_MVREL_OUT           *rel_out,
        unsigned long           timeout
)
```

引数

rel_in	: 入力用構造体
rel_out	: 出力用構造体
timeout	: タイムアウト時間 (x10msec) 0 の場合、無限待ち

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 相対位置による位置決めを実行します。位置決め完了は、目標位置に対して機器に設定された位置決め完了幅の範囲に到達する事で完了します。

非同期実行(出力用構造体=NULL による API 関数実行)時は、timeout は無効です。
終了待ちは PO_WaitForMotionRecv()関数を使用してください。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
Distance	移動量	LREAL	倍精度実数値 【指令単位】	0	位置決め目標位置を相対位置で指定
Velocity	移動速度	LREAL	0,倍精度実数値正数 【指令単位/s】	0	位置決め時の速度
Acceleration	加速度	LREAL	倍精度実数値正数 【指令単位/s^2】	0	位置決め時の加速度
Deceleration	減速度	LREAL	倍精度実数値正数 【指令単位/s^2】	0	位置決め時の減速度
Jerk	加加速度	LREAL	ー	ー	未サポート
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定

型名	型	値	モード	説明
MC_BufferMode	UINT16	0	Aborting	実行中の API 関数に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	位置決め完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中に TRUE
CommandAborted	実行中断	BOOL	TRUE or FALSE	命令が中止されたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

MC_MoveAdditive 関数

機能　直前に実行された位置決めに対して、本 API 関数で指定した相対位置を加算した位置に対して位置決めを実行します。

書式
```
RTHANDLE  MC_MoveAdditive (
        TFB_MVADD_IN_LIB        *add_in,
        TFB_MVADD_OUT           *add_out,
        unsigned long           timeout
)
```

引数

add_in	: 入力用構造体
add_out	: 出力用構造体
timeout	: タイムアウト時間 (x10msec) 0 の場合、無限待ち

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　直前のコマンドによる目標位置に、指定された相対位置を付加して移動します。位置決め完了は、目標位置に対して機器に設定された位置決め完了幅の範囲に到達する事で完了します。本 API 関数を単体で実行した場合の動作は、MC_MoveRelative と同等です。

非同期実行(出力用構造体=NULL による API 関数実行)時は、timeout は無効です。
終了待ちは PO_WaitForMotionRecv()関数を使用してください。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
Distance	加算移動量	LREAL	倍精度実数値 【指令単位】	0	加算移動量を相対位置で指定
Velocity	移動速度	LREAL	0,倍精度実数値正数 【指令単位/s】	0	位置決め時の速度
Acceleration	加速度	LREAL	倍精度実数値正数 【指令単位/s^2】	0	位置決め時の加速度
Deceleration	減速度	LREAL	倍精度実数値正数 【指令単位/s^2】	0	位置決め時の減速度
Jerk	加加速度	LREAL	ー	ー	未サポート
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定

型名	型	値	モード	説明
MC_BufferMode	UINT16	0	Aborting	実行中の API 関数に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	位置決め完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中に TRUE
CommandAborted	実行中断	BOOL	TRUE or FALSE	命令が中止されたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

MC_MoveSuperimposed 関数

機能 未サポート

書式
```
RTHANDLE  MC_MoveSuperimposed (
        TFB_MVSPIMP_IN_LIB      *spi_in,
        TFB_MVSPIMP_OUT         *spi_out,
        unsigned long           timeout
)
```

引数

spi_in	: 入力用構造体
spi_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

MC_MoveVelocity 関数

機能 指定速度による定速駆動を実行します。

書式
```
RTHANDLE  MC_MoveVelocity (
        TFB_MVVEL_IN_LIB        *vel_in,
        TFB_MVVEL_OUT           *vel_out,
        unsigned long           timeout
)
```

引数

vel_in	: 入力用構造体
vel_out	: 出力用構造体
timeout	: タイムアウト時間 (x10msec) 0 の場合、無限待ち

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 指定された速度での永久動作を命令します。
本 API 関数による動作を停止させるには、別の API 関数による指令を行う必要があります。

非同期実行(出力用構造体=NULL による API 関数実行)時は、timeout は無効です。
終了待ちは PO_WaitForMotionRecv()関数を使用してください。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
Velocity	指令速度	LREAL	0,倍精度実数値正数 【指令単位/s】	0	速度制御時の動作速度
Acceleration	加速度	LREAL	倍精度実数値正数 【指令単位/s^2】	0	速度制御時の加速度
Deceleration	減速度	LREAL	倍精度実数値正数 【指令単位/s^2】	0	速度制御時の減速度
Jerk	加加速度	LREAL	ー	ー	未サポート
Direction	動作方向	MC_Direction	0～3	0	動作方向を指定
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定

型名	型	値	モード	説明
MC_Direction	UINT16	0	正方向	正転方向に動作
		1	近回り	本 API 関数では常に正方向に動作
		2	逆方向	逆転方向に動作
		3	現在の方向	動作中の場合、同方向に動作。 停止中の場合は正転方向へ動作
MC_BufferMode	UINT16	0	Aborting	実行中の API 関数に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

出力構造体詳細

変数	名称	データ型	範囲	内容
InVelocity	速度到達通知	BOOL	TRUE or FALSE	指令速度に到達で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中に TRUE
CommandAborted	実行中断	BOOL	TRUE or FALSE	命令が中止されたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

MC_TorqueControl 関数

機能 指定トルクによるトルク制御を実行します。

書式
```
RTHANDLE  MC_TorqueControl(
        TFB_TRQCTRL_IN_LIB    *tqc_in,
        TFB_TRQCTRL_IN        *tqc_out,
        unsigned long         timeout
)
```

引数

tqc_in	: 入力用構造体
tqc_out	: 出力用構造体
timeout	: タイムアウト時間 (x10msec)

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 指定されたトルクでの永久動作を命令します。
本 API 関数による動作を停止させるには、別の API 関数による指令を行う必要があります。

非同期実行(出力用構造体=NULL による API 関数実行)時は、timeout は無効です。
終了待ちは PO_WaitForMotionRecv()関数を使用してください。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	立ち上がり時に命令を実行
Torque	指令トルク	LREAL	0,倍精度実数値正数 【N.m】or【%】	0	トルク制御時の動作トルク 単位はサーボパックにより異なる。〈※1〉
TorqueRamp	トルク傾斜度	LREAL	ー	ー	未サポート
Velocity	指令速度	LREAL	ー	ー	未サポート
Acceleration	加速度	LREAL	ー	ー	未サポート
Deceleration	減速度	LREAL	ー	ー	未サポート
Jerk	加加速度	LREAL	ー	ー	未サポート
Direction	動作方向	MC_Direction	0～3	0	動作方向を指定
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定

※1：MECHATROLINK-Ⅲの場合 ：サーボパラメータ「トルク単位選択（0x47)」の値により変わります。
EtherCAT の場合 ：サーボパックマニュアルの CiA402 パラメータ「目標トルク（0x6071)」または、「内部指令トルク（0x6074)」の単位を参照してください。

型名	型	値	モード	説明
MC_Direction	UINT16	0	正方向	正転方向に動作
		1	近回り	本 API 関数では常に正方向に動作
		2	逆方向	逆転方向に動作
		3	現在の方向	動作中の場合、同方向に動作。 停止中の場合は正転方向へ動作
MC_BufferMode	UINT16	0	Aborting	実行中の API 関数に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

出力構造体詳細

変数	名称	データ型	範囲	内容
InTorque	トルク到達通知	BOOL	TRUE or FALSE	指令トルクに到達で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中に TRUE
CommandAborted	実行中断	BOOL	TRUE or FALSE	命令が中止されたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

MC_Home 関数

機能 原点復帰シーケンスを実行します。本API関数の機能は「3-4-4 PLCopen仕様　原点復帰API関数」に分割して実装しています。

書式

```
RTHANDLE  MC_Home (
        TFB_HOME_IN_LIB     *hom_in,
        TFB_HOME_OUT        *hom_out,
        unsigned long       timeout
)
```

引数

hom_in	: 入力用構造体
hom_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

MC_Stop 関数

機能　位置決め実行中の軸動作を停止させます。

書式

```
RTHANDLE  MC_Stop (
        TFB_STOP_IN_LIB     *stp_in,
        TFB_STOP_OUT        *stp_out,
        unsigned long       timeout
)
```

引数

stp_in	: 入力用構造体
stp_out	: 出力用構造体
timeout	: タイムアウト時間 (x10msec) 0 の場合、無限待ち

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　軸の制御動作を停止させ、Stopping 状態に遷移します。軸停止後、Done 出力がセットされますが、Execute 入力が True の間は Stopping 状態のままになります。Done 出力セット後に再度本 API 関数を実行し Execute 入力を False にする事で StandStill 状態に遷移します。

非同期実行(出力用構造体=NULL による API 関数実行)時は、timeout は無効です。
終了待ちは PO_WaitForMotionRecv()関数を使用してください。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
Deceleration	減速度	LREAL	倍精度実数値正数 【指令単位/s^2】	0	停止時の減速度
Jerk	加加速度	LREAL	ー	ー	未サポート
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定 **本 API 関数では、0:Aborting のみサポート**

型名	型	値	モード	説明
MC_BufferMode	UINT16	0	Aborting	実行中の API 関数に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	ゼロ速度到達で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中に TRUE
CommandAborted	実行中断	BOOL	TRUE or FALSE	命令が中止されたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

MC_PositionProfile 関数

機能 未サポート

書式
```
RTHANDLE  MC_PositionProfile (
        TFB_POSPRO_IN_LIB       *ppr_in,
        TFB_POSPRO_OUT          *ppr_out,
        unsigned long           timeout
)
```

引数

ppr_in	: 入力用構造体
ppr_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

MC_VelocityProfile 関数

機能 未サポート

書式
```
RTHANDLE  MC_VelocityProfile (
        TFB_VELPRO_IN_LIB       *vpr_in,
        TFB_VELPRO_OUT          *vpr_out,
        unsigned long           timeout
)
```

引数

vpr_in	: 入力用構造体
vpr_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

MC_AccelerationProfile 関数

機能 未サポート

書式
```
RTHANDLE  MC_AccelerationProfile (
        TFB_ACCPRO_IN_LIB       *apr_in,
        TFB_ACCPRO_OUT          *apr_out,
        unsigned long           timeout
)
```

引数

apr_in	: 入力用構造体
apr_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

MC_CamIn 関数

機能 未サポート

書式
```
RTHANDLE  MC_CamIn (
        TFB_CAMIN_IN_LIB        *cmi_in,
        TFB_CAMIN_OUT           *cmi_out,
        unsigned long           timeout
)
```

引数

cmi_in	: 入力用構造体
cmi_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

MC_CamOut 関数

機能 未サポート

書式
```
RTHANDLE  MC_CamOut (
        TFB_CAMOUT_IN_LIB       *cmo_in,
        TFB_CAMOUT_OUT          *cmo_out,
        unsigned long           timeout
)
```

引数

cmo_in	: 入力用構造体
cmo_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

MC_GearIn 関数

機能 未サポート

書式
```
RTHANDLE  MC_GearIn (
        TFB_GEARIN_IN_LIB       *gri_in,
        TFB_GEARIN_OUT          *gri_out,
        unsigned long           timeout
)
```

引数

gri_in	: 入力用構造体
gri_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

MC_GearOut 関数

機能 未サポート

書式
```
RTHANDLE  MC_GearOut (
        TFB_GEAROUT_IN_LIB      *gro_in,
        TFB_GEAROUT_OUT         *gro_out,
        unsigned long           timeout
)
```

引数

gro_in	: 入力用構造体
gro_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

MC_Phasing 関数

機能 未サポート

書式
```
RTHANDLE  MC_Phasing (
        TFB_PHASE_IN_LIB        *phs_in,
        TFB_PHASE_OUT           *phs_out,
        unsigned long           timeout
)
```

引数

phs_in	: 入力用構造体
phs_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

３－４－４　　PLCopen仕様　原点復帰API関数

本項では PLCopen MC 使用に定義されている原点復帰系の API 関数について説明します。本項で説明します API 関数は、PLCopen の技術仕様書「Technical Paper PLCopen Technical Committee 2 – Task Force Function Blocks for motion control Part 5 - Homing」に定義されている仕様に従った API 関数になっています。これらの API 関数は MC_Home に相当する機能を個別の API 関数として定義されたものになっており、実行時には Homing 状態に状態遷移します。

MC_StepAbsSwitch 関数

機能 機械的に設置されたリミット SW、原点 SW を使用する事で原点復帰を実行します。

書式
```
RTHANDLE  MC_StepAbsSwitch (
        TFB_STEPABSSW_IN_LIB        *sas_in,
        TFB_STEPABSSW_OUT           *sas_out,
        unsigned long               timeout
)
```

引数

sas_in	: 入力用構造体
sas_out	: 出力用構造体
timeout	: タイムアウト時間 (x10msec) 0 の場合、無限待ち

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 原点復帰が実行可能な状態のとき、本 API 関数の実行により原点復帰を開始し、状態を Homing 状態に遷移します。本 API 関数では、停止位置の位置情報を更新しません。
本 API 関数は、正常終了時、StandStill 状態へ遷移しません。MC_FinishHoming を実行し、StandStill 状態へ遷移させてください。
異常終了時は ErrorStop 状態へ遷移します。

非同期実行(出力用構造体=NULL による API 関数実行)時は、timeout は無効です。
終了待ちは PO_WaitForMotionRecv()関数を使用してください。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
Direction	動作方向	MC_HomeDir	0～3	0	動作方向を指定
SwitchMode	センサモード	MC_SwitchMode	0～5	0	原点復帰完了センサモード
Velocity	移動速度	LREAL	0,倍精度実数値正数 【指令単位/s】	0	位置決め時の速度
TorqueLimit	トルクリミット	LREAL	ー	ー	未サポート
TimeLimit	タイマリミット	LREAL	0,倍精度実数値正数 【s】	0	原点復帰タイムアウト時間 0 の場合は無制限
DistanceLimit	移動量リミット	LREAL	0,倍精度実数値正数 【指令単位】	0	移動量のリミット値 0 の場合は無制限
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定 **本 FB では、0:Aborting のみサポート**

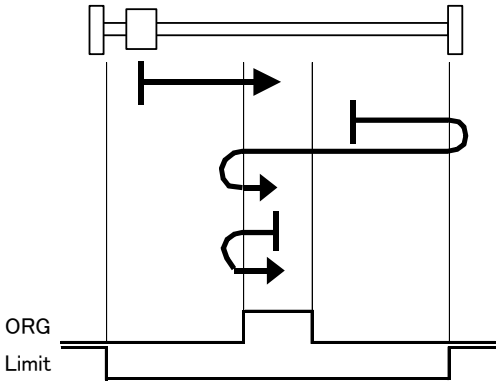
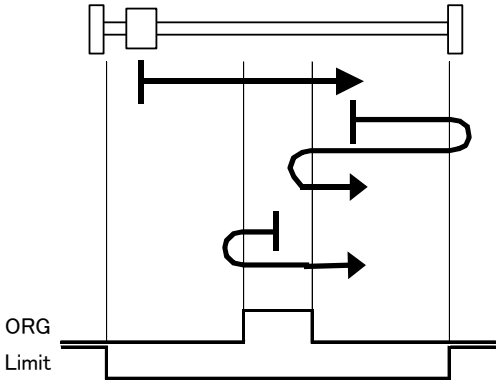
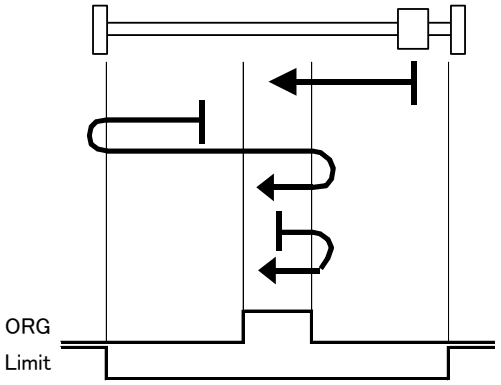
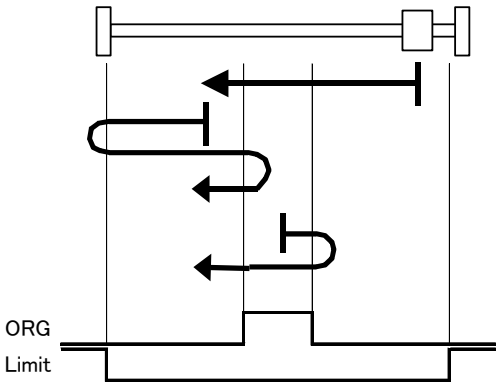
型名	型	値	モード	説明
MC_HomeDir	UINT16	0	MC_Positive	正方向へ開始（**未サポート**）
		1	MC_Negative	負方向へ開始（**未サポート**）
		2	MC_SwitchPositive	実行開始時に原点信号が ON していたら負方向へ開始、OFF していたら正方向へ開始
		3	MC_SwitchNegative	実行開始時に原点信号が ON していたら正方向へ開始、OFF していたら負方向へ開始
MC_SwitchMode	UINT16	0	MC_On	センサが ON 状態なら原点復帰完了（**未サポート**）
		1	MC_Off	センサが OFF 状態なら原点復帰完了（**未サポート**）
		2	MC_EdgeOn	センサが OFF→ON のエッジで原点復帰完了
		3	MC_EdgeOff	センサが ON→OFF のエッジで原点復帰完了
		4	MC_EdgeSwitchPositive	動作方向により原点復帰完了のエッジが変わる（**未サポート**）
		5	MC_EdgeSwitchNegative	動作方向により原点復帰完了のエッジが変わる（**未サポート**）
MC_BufferMode	UINT16	0	Aborting	実行中の FB に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

※注:MC_EdgeOff については、サーボパックが OFF エッジでの原点復帰を対応していないと使えません。

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	原点復帰完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中に TRUE
CommandAborted	実行中断	BOOL	TRUE or FALSE	命令が中止されたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

動作方向	センサモード	動作内容
MC_SwitchPositive	MC_EdgeOn	
	MC_EdgeOff	
MC_SwitchNegative	MC_EdgeOn	
	MC_EdgeOff	

MC_StepLimitSwitch 関数

機能 機械的に設置されたリミット SW を使用して原点復帰を行います。

書式
```
RTHANDLE  MC_StepLimitSwitch (
        TFB_STEPLMTSW_IN_LIB    *sls_in,
        TFB_STEPLMTSW_OUT       *sls_out,
        unsigned long           timeout
)
```

引数

sls_in	: 入力用構造体
sls_out	: 出力用構造体
timeout	: タイムアウト時間 (x10msec) 0 の場合、無限待ち

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 原点復帰が実行可能な状態のとき、本 API 関数の実行により原点復帰を開始し、状態を Homing 状態に遷移します。本 API 関数では、停止位置の位置情報を更新しません。本 API 関数は、正常終了時、StandStill 状態へ遷移しません。MC_FinishHoming を実行し、StandStill 状態へ遷移させてください。
異常終了時は ErrorStop 状態へ遷移します。

非同期実行(出力用構造体=NULL による API 関数実行)時は、timeout は無効です。
終了待ちは PO_WaitForMotionRecv()関数を使用してください。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
Direction	動作方向	MC_HomeDir	0～1	0	動作方向を指定
LimitSwitchMode	センサモード	MC_SwitchMode	0～3	0	原点復帰完了センサモード
Velocity	移動速度	LREAL	0,倍精度実数値正数 【指令単位/s】	0	位置決め時の速度
TorqueLimit	トルクリミット	LREAL	ー	ー	未サポート
TimeLimit	タイマリミット	LREAL	0,倍精度実数値正数 【s】	0	原点復帰タイムアウト時間 0 の場合は無制限
DistanceLimit	移動量リミット	LREAL	0,倍精度実数値正数 【指令単位】	0	移動量のリミット値 0 の場合は無制限
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定 **本 FB では、0:Aborting のみサポート**

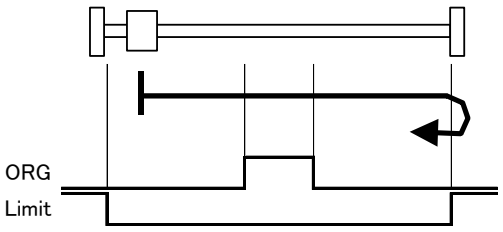
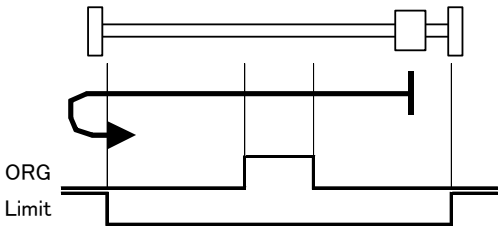
型名	型	値	モード	説明
MC_HomeDir	UINT16	0	MC_Positive	正方向へ開始
		1	MC_Negative	負方向へ開始
MC_SwitchMode	UINT16	0	MC_On	センサが ON 状態なら原点復帰完了 （**未サポート**）
		1	MC_Off	センサが OFF 状態なら原点復帰完了 （**未サポート**）
		2	MC_EdgeOn	センサが OFF→ON のエッジで原点復帰完了（**未サポート**）
		3	MC_EdgeOff	センサが ON→OFF のエッジで原点復帰完了
MC_BufferMode	UINT16	0	Aborting	実行中の FB に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

出力

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	原点復帰完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中に TRUE
CommandAborted	実行中断	BOOL	TRUE or FALSE	命令が中止されたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

動作方向	センサモード	動作内容
MC_Positive	MC_EdgeOff	
MC_Negative	MC_EdgeOff	

MC_StepBlock 関数

機能 未サポート

書式
```
RTHANDLE  MC_StepBlock (
        TFB_STPBLK_IN_LIB       *stb_in,
        TFB_STPBLK_OUT          *stb_out,
        unsigned long           timeoout
)
```

引数

stb_in	: 入力用構造体
stb_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

MC_FinishHoming 関数

機能　指定された軸の状態を Homing 状態から StandStill 状態に遷移させます。

書式
```
RTHANDLE  MC_FinishHoming (
        TFB_FINHOME_IN_LIB      *fhm_in,
        TFB_FINHOME_OUT         *fhm_out,
        unsigned long           timeout)
```

引数
fhm_in　：入力用構造体
fhm_out　：出力用構造体
timeout　：タイムアウト時間（x10msec）
0 の場合、無限待ち

戻り値
0 以外　：正常(RtHandle 値)
0　：異常

説明
指定された軸の状態を Homing 状態から StandStill 状態に遷移させます。
軸は動作しません。
MC_StepAbsSwitch と MC_StepLimitSwitch を実行した後に、本 API 関数を実行します。

非同期実行(出力用構造体=NULL による API 関数実行)時は、timeout は無効です。
終了待ちは PO_WaitForMotionRecv()関数を使用してください。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定 **本 FB では、0:Aborting のみサポート**

型名	型	値	モード	説明
MC_BufferMode	UINT16	0	Aborting	実行中の FB に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	原点復帰完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中に TRUE
CommandAborted	実行中断	BOOL	TRUE or FALSE	命令が中止されたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

MC_StepRefPulse 関数

機能　エンコーダからの Zero パルス（マーカー、またはリファレンスパルスとも呼びます。）を参照しながら原点復帰を行います。

書式

```
RTHANDLE  MC_StepRefPulse (
        TFB_STEPREFPLS_IN_LIB       *srp_in,
        TFB_STEPREFPLS_OUT          *srp_out,
        unsigned long               timeout
)
```

引数

srp_in	: 入力用構造体
srp_out	: 出力用構造体
timeout	: タイムアウト時間（x10msec） 0 の場合、無限待ち

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　エンコーダからの Zero パルス（マーカー、またはリファレンスパルスとも呼びます。）を参照しながら原点復帰を行います。
リファレンスパルスはエンコーダ 1 回転に 1 度発生します。
原点復帰にリファレンスパルスを使用する利点は伝統的な光学センサや磁気センサよりも高い精度を出す事が出来ることです。
実行開始時、状態が Homing 状態ではなければ Homing 状態に遷移します。
原点復帰完了位置を SetPosition で指定された位置に更新します。
正常終了時、Homing 状態から StandStill 状態に遷移します。

非同期実行(出力用構造体=NULL による API 関数実行)時は、timeout は無効です。
終了待ちは PO_WaitForMotionRecv()関数を使用してください。

入力構造体詳細

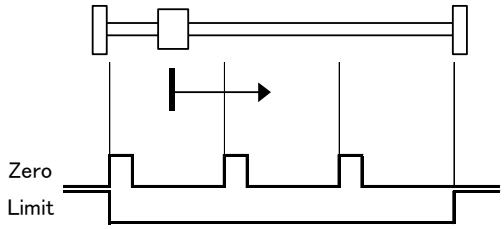
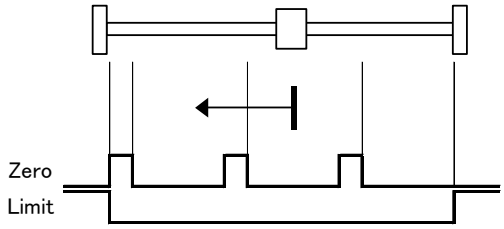
変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
Direction	動作方向	MC_HomeDir	0～1	0	動作方向を指定
Velocity	移動速度	LREAL	0,倍精度実数値正数 【指令単位/s】	0	位置決め時の速度
SetPosition	原点更新位置	LREAL	倍精度実数値 【指令単位】		
TorqueLimit	トルクリミット	LREAL	ー	ー	未サポート
TimeLimit	タイマリミット	LREAL	0,倍精度実数値正数 【s】	0	原点復帰タイムアウト時間 0 の場合は無制限
DistanceLimit	移動量リミット	LREAL	0,倍精度実数値正数 【指令単位】	0	移動量のリミット値 0 の場合は無制限
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定 **本 FB では、0:Aborting のみサポート**

型名	型	値	モード	説明
MC_HomeDir	UINT16	0	MC_Positive	正方向へ開始
		1	MC_Negative	負方向へ開始
MC_BufferMode	UINT16	0	Aborting	実行中の FB に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	原点復帰完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中に TRUE
CommandAborted	実行中断	BOOL	TRUE or FALSE	命令が中止されたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

動作方向	動作内容
MC_Positive	
MC_Negative	

MC_StepDirect 関数

機能　SetPosition 入力値を現在停止位置にセットすることで原点復帰を完了します。

書式

```
RTHANDLE  MC_StepDirect (
        TFB_STEPDIR_IN_LIB      *std_in,
        TFB_STEPDIR_OUT         *std_out,
        unsigned long           timeout
)
```

引数

std_in	: 入力用構造体
std_out	: 出力用構造体
timeout	: タイムアウト時間 (x10msec) 0 の場合、無限待ち

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明

SetPosition 入力値を現在停止位置にセットすることで原点復帰を完了します。
本 API 関数では軸は動作しません。
正常終了時、Homing 状態から StandStill 状態に遷移します。

非同期実行(出力用構造体=NULL による API 関数実行)時は、timeout は無効です。
終了待ちは PO_WaitForMotionRecv()関数を使用してください。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
SetPosition	原点更新位置	LREAL	倍精度実数値 【指令単位】		
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定 **本 FB では、0:Aborting のみサポート**

型名	型	値	モード	説明
MC_BufferMode	UINT16	0	Aborting	実行中の FB に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	原点復帰完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中に TRUE
CommandAborted	実行中断	BOOL	TRUE or FALSE	命令が中止されたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

MC_StepAbsolute 関数

機能　アブソリュートエンコーダに現在位置を原点位置としてセットします。

書式

```
RTHANDLE  MC_StepAbsolute (
        TFB_STEPABS_IN_LIB      *sta_in,
        TFB_STEPABS_OUT         *sta_out,
        unsigned long           timeout
)
```

引数

sta_in	: 入力用構造体
sta_out	: 出力用構造体
timeout	: タイムアウト時間 (x10msec) 0 の場合、無限待ち

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　アブソリュートエンコーダに現在位置を原点位置としてセットします。
軸動作は行われません。
正常終了時、Homing 状態から StandStill 状態に遷移します。

非同期実行(出力用構造体=NULL による API 関数実行)時は、timeout は無効です。
終了待ちは PO_WaitForMotionRecv()関数を使用してください。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
BufferMode	動作モード	MC_BufferMode	0～5	0	モーション命令 多重起動命令動作指定 **本 FB では、0:Aborting のみサポート**

型名	型	値	モード	説明
MC_BufferMode	UINT16	0	Aborting	実行中の FB に対して、重ねて次の動作指示を出す場合の制御方法を指定 詳細は「3-5-1 動作API関数の多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	

出力構造体詳細

変数	名称	データ型	範囲	内容
Done	実行完了通知	BOOL	TRUE or FALSE	原点復帰完了で TRUE
Busy	実行中	BOOL	TRUE or FALSE	命令を受け付けたときに TRUE
Active	軸制御中	BOOL	TRUE or FALSE	軸制御中に TRUE
CommandAborted	実行中断	BOOL	TRUE or FALSE	命令が中止されたときに TRUE
Error	エラー発生中	BOOL	TRUE or FALSE	異常が発生したとき TRUE
ErrorID	エラー番号	DWORD	※1	異常が発生したときのエラーコードを出力

※1：エラーコード一覧を参照

MC_StepRefFlyingSwitc 関数

機能 未サポート

書式
```
RTHANDLE  MC_StepRefFlyingSwitc (
        TFB_STPREFSW_IN_LIB     *srf_in,
        TFB_STPREFSW_OUT        *srf_out,
        unsigned long           timeout
)
```

引数

	srf_in	: 入力用構造体
	srf_out	: 出力用構造体
`	timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

MC_StepRefFlyingRefPulse 関数

機能 未サポート

書式
```
RTHANDLE  MC_StepRefFlyingRefPulse (
        TFB_STPREFPLS_IN_LIB       *srp_in,
        TFB_STPREFPLS_OUT          *srp_out,
        unsigned long              timeout
)
```

引数

srp_in	: 入力用構造体
srp_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

MC_AbortPassiveHoming 関数

機能 未サポート

書式
```
RTHANDLE  MC_AbortPassiveHoming (
        TFB_ABTHOME_IN_LIB      *aph_in,
        TFB_ABTHOME_OUT         *aph_out,
        unsigned long           timeout
)
```

引数

aph_in	: 入力用構造体
aph_out	: 出力用構造体
timeout	: タイムアウト時間

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明 未サポート

３－５　　モーション制御機能

本項では、PLCopen で規定されている、モーション制御の特殊な使い方について説明します。

３－５－１　動作API関数の多重起動

いくつかの API 関数は「BufferMode」と呼ばれる入力を持ちます。この入力により、API 関数は「Aborting mode」（デフォルト動作）と「Buffered mode」の両方で動作が可能です。これらのモードの相違点は、それらの動作がいつ開始されるかです。

- 非バッファリングモードでのコマンドは、他の動作を中断してでもすぐに動作します。
- バッファリングモードでのコマンドは、現在の API 関数が自身の「Done」（または「InPosition」や「InVelocity」）出力をセットするまで待ちます。

バッファリングモードには、いくつかのオプションがあります。この入力はMC_BUFFERMODE のENUM 型です。表 3-5-1-1に各バッファリングモードの一覧を示します。

表 3-5-1-1.　バッファリングモード一覧

値	モード	内容
0	Aborting	バッファリングしないデフォルトモード。次の API 関数は、実行中の動作を中断し、コマンドは直ちに軸に影響します。
1	Buffered	次の API 関数は、以前の動作が「Done」になると、直ちに軸に影響します。
2	BlendingLow	次の API 関数は、以前の API 関数が完了した後に軸を制御しますが、２つの動作間で軸は停止しません。 API 関数 1 の終了位置で API 関数 1 と API 関数 2 の低い速度とします。
3	BlendingPrevious	次の API 関数は、以前の API 関数が完了した後に軸を制御しますが、２つの動作間で軸は停止しません。 API 関数 1 の終了位置で API 関数 1 の速度とします。
4	BlendingNext	次の API 関数は、以前の API 関数が完了した後に軸を制御しますが、２つの動作間で軸は停止しません。 API 関数 1 の終了位置で API 関数 2 の速度とします。
5	BlendingHigh	次の API 関数は、以前の API 関数が完了した後に軸を制御しますが、２つの動作間で軸は停止しません。 API 関数 1 の終了位置で API 関数 1 と API 関数 2 の高い速度とします。

以下の例は、これらモードの動作の相違を記述しています：

- 連続した２つの絶対位置移動の標準動作

```
//-----------------------------------------------------------------------------
//       Aborting Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------------
int                 ret;
TFB_MVABS_IN_LIB    mvabs_in;
TFB_MVABS_OUT       mvabs_out;
RTHandle            h_mvabs;
RTHandle            h_wait_mvabs[2];

mvabs_in.Axis             = 1;
mvabs_in.Execute          = 1;
mvabs_in.Position         = 1000.0;
mvabs_in.Velocity         = 100.0;
mvabs_in.Acceleration     = 100.0;
mvabs_in.Deceleration     = 100.0;
mvabs_in.BufferMode       = 0;   //Aborting
mvabs_in.Direction        = 0;
mvabs_in.Jerk             = 0;
h_mvabs     = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //1つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[0]       = h_mvabs;
}
mvabs_in.Axis             = 1;
mvabs_in.Execute          = 1;
mvabs_in.Position         = 2000.0;
mvabs_in.Velocity         = 50.0;
mvabs_in.Acceleration     = 50.0;
mvabs_in.Deceleration     = 50.0;
mvabs_in.BufferMode       = 0;   //Aborting
mvabs_in.Direction        = 0;
mvabs_in.Jerk             = 0;
h_mvabs     = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //2つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[1]       = h_mvabs;
}
//動作完了待ち
 //先の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[0], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
```

```
}
 //後の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[1], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
```

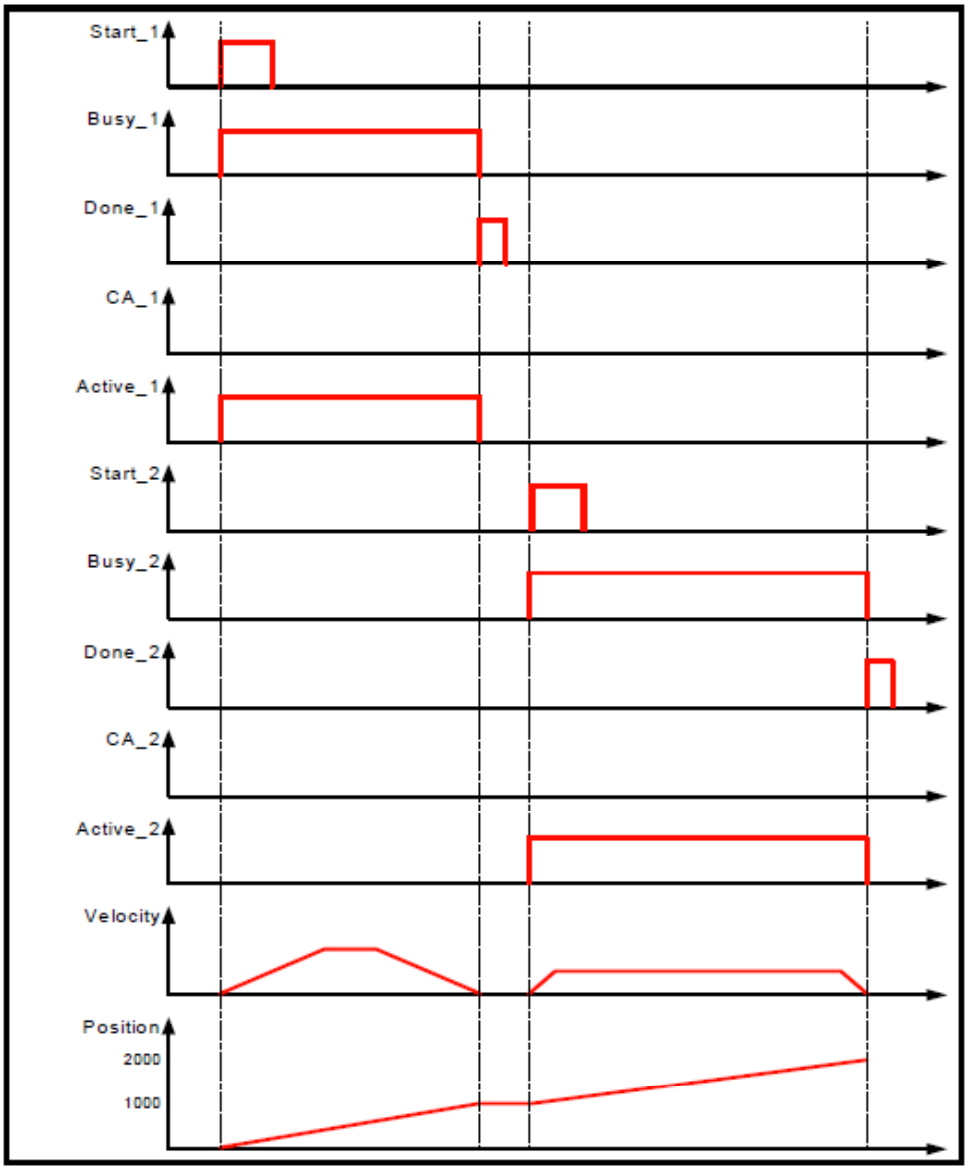


図 3-5-1-1.　上の例でAPI関数 1 とAPI関数 2 の間に干渉が無い場合でのタイムチャート（Aborting Mode）

- Aborting mode での動作

```
//-----------------------------------------------------------------------
//       Aborting Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------
int                ret;
TFB_MVABS_IN_LIB   mvabs_in;
TFB_MVABS_OUT      mvabs_out;
RTHandle           h_mvabs;
RTHandle           h_wait_mvabs[2];

mvabs_in.Axis             = 1;
mvabs_in.Execute          = 1;
mvabs_in.Position         = 1000.0;
mvabs_in.Velocity         = 100.0;
mvabs_in.Acceleration     = 100.0;
mvabs_in.Deceleration     = 100.0;
mvabs_in.BufferMode       = 0;   //Aborting
mvabs_in.Direction        = 0;
mvabs_in.Jerk             = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //1つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[0]       = h_mvabs;
}
mvabs_in.Axis             = 1;
mvabs_in.Execute          = 1;
mvabs_in.Position         = 2000.0;
mvabs_in.Velocity         = 50.0;
mvabs_in.Acceleration     = 50.0;
mvabs_in.Deceleration     = 50.0;
mvabs_in.BufferMode       = 0;   //Aborting
mvabs_in.Direction        = 0;
mvabs_in.Jerk             = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //2つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[1]       = h_mvabs;
}

//動作完了待ち
 //先の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[0], &base_out, 1000);
if(ret == OK){
```

```
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
 //後の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[1], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
```

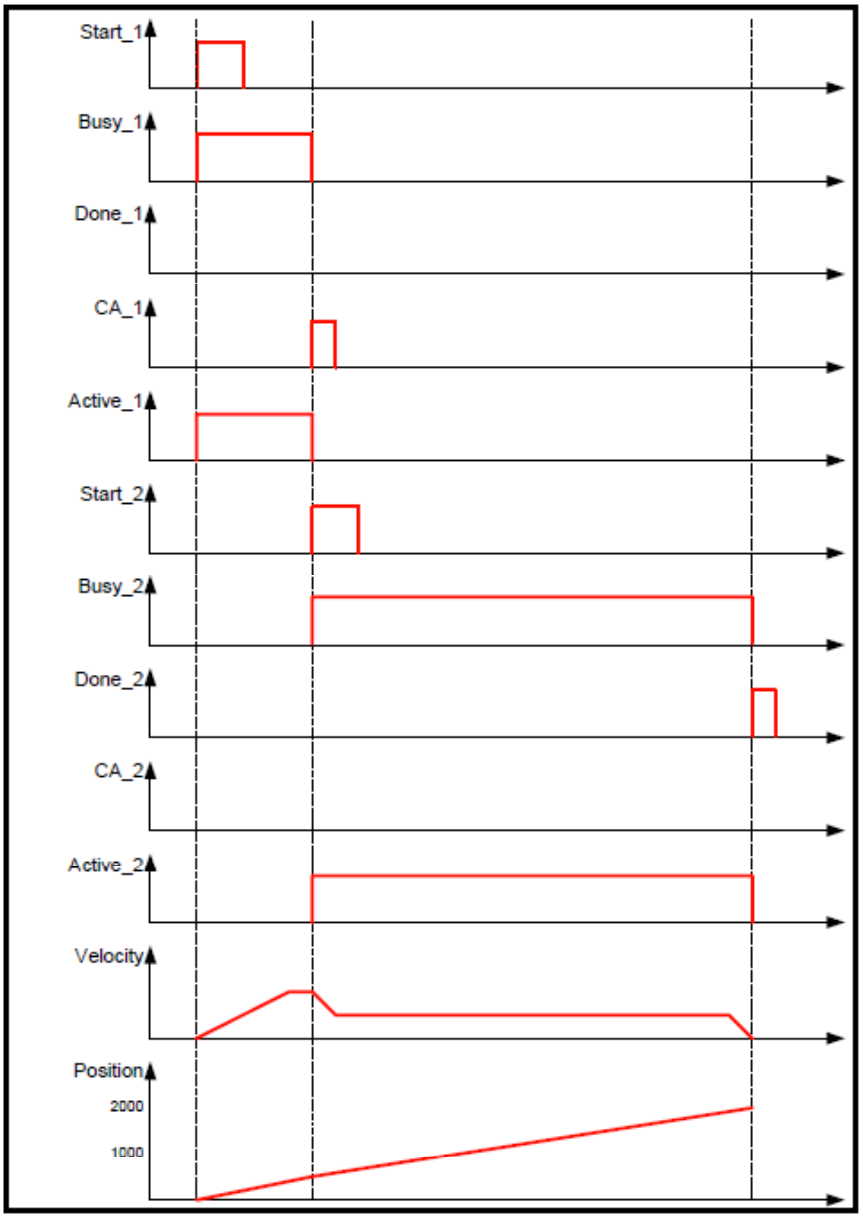


図 3-5-1-2.　上の例のAPI関数 2 がAPI関数 1 に割り込む場合でのタイムチャート（Aborting Mode）

- Bufferd mode での動作

```
//-----------------------------------------------------------------------------
//       Buffered Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------------
int                 ret;
TFB_MVABS_IN_LIB    mvabs_in;
TFB_MVABS_OUT       mvabs_out;
RTHandle            h_mvabs;
RTHandle            h_wait_mvabs[2];

mvabs_in.Axis             = 1;
mvabs_in.Execute          = 1;
mvabs_in.Position         = 1000.0;
mvabs_in.Velocity         = 100.0;
mvabs_in.Acceleration     = 100.0;
mvabs_in.Deceleration     = 100.0;
mvabs_in.BufferMode       = 0;   //Aborting
mvabs_in.Direction        = 0;
mvabs_in.Jerk             = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //1つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[0]       = h_mvabs;
}
mvabs_in.Axis             = 1;
mvabs_in.Execute          = 1;
mvabs_in.Position         = 2000.0;
mvabs_in.Velocity         = 50.0;
mvabs_in.Acceleration     = 50.0;
mvabs_in.Deceleration     = 50.0;
mvabs_in.BufferMode       = 1;   //Buffered
mvabs_in.Direction        = 0;
mvabs_in.Jerk             = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //2つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[1]       = h_mvabs;
}

//動作完了待ち
 //先の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[0], &base_out, 1000);
if(ret == OK){
```

```
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
 //後の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[1], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
```

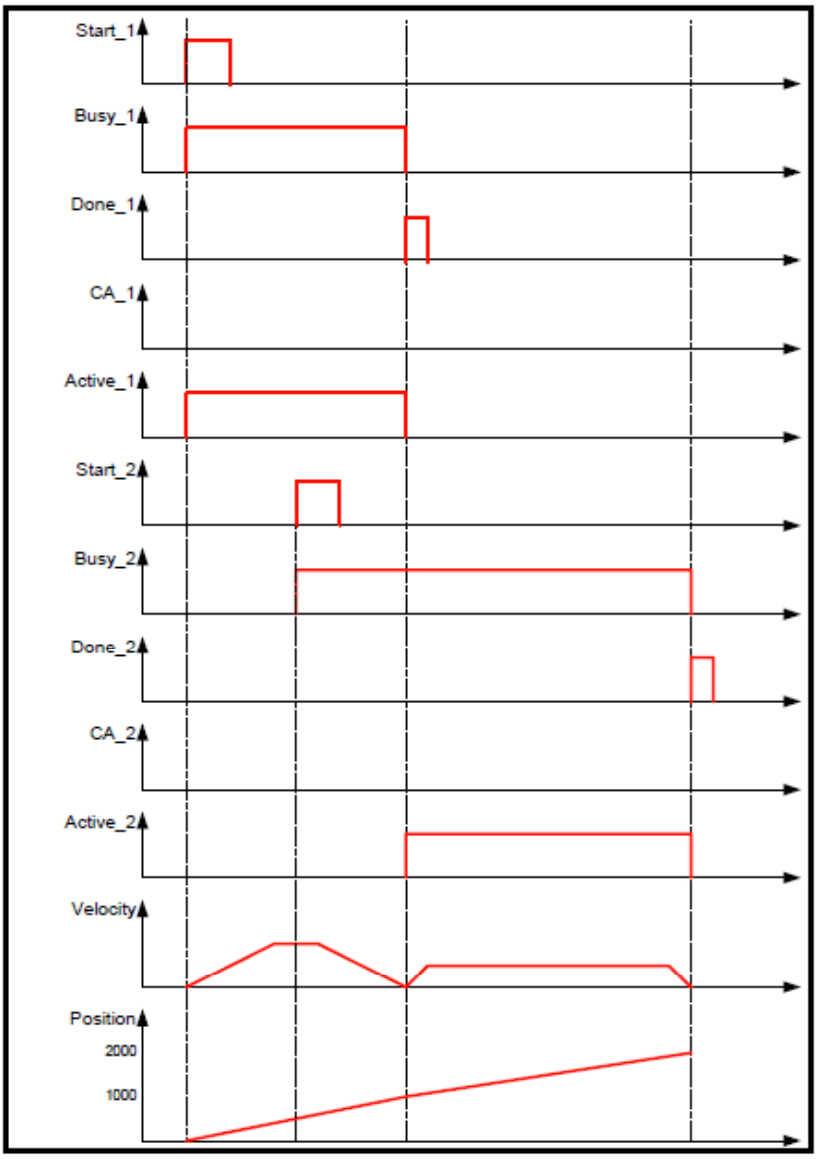


図 3-5-1-3.　上の例のBuffered Modeでのタイムチャート
（速度 0 で停止し、遅延なくその位置で API 関数 2 を開始する）

- BlendingLow mode での動作

```
//-----------------------------------------------------------------------------
//       BlendingLow Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------------
int                ret;
TFB_MVABS_IN_LIB   mvabs_in;
TFB_MVABS_OUT      mvabs_out;
RTHandle           h_mvabs;
RTHandle           h_wait_mvabs[3];

mvabs_in.Axis            = 1;
mvabs_in.Execute         = 1;
mvabs_in.Position        = 1000.0;
mvabs_in.Velocity        = 100.0;
mvabs_in.Acceleration    = 100.0;
mvabs_in.Deceleration    = 100.0;
mvabs_in.BufferMode      = 0;   //Aborting
mvabs_in.Direction       = 0;
mvabs_in.Jerk            = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //1つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[0]      = h_mvabs;
}
mvabs_in.Axis            = 1;
mvabs_in.Execute         = 1;
mvabs_in.Position        = 2000.0;
mvabs_in.Velocity        = 50.0;
mvabs_in.Acceleration    = 50.0;
mvabs_in.Deceleration    = 50.0;
mvabs_in.BufferMode      = 2;   // BlendingLow
mvabs_in.Direction       = 0;
mvabs_in.Jerk            = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //2つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[1]      = h_mvabs;
}
mvabs_in.Axis            = 1;
mvabs_in.Execute         = 1;
mvabs_in.Position        = 3000.0;
mvabs_in.Velocity        = 100.0;
mvabs_in.Acceleration    = 100.0;
mvabs_in.Deceleration    = 100.0;
```

```
mvabs_in.BufferMode       = 2;    // BlendingLow
mvabs_in.Direction        = 0;
mvabs_in.Jerk             = 0;
h_mvabs     = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //3つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[2]       = h_mvabs;
}
//動作完了待ち
 //先の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[0], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
 //2 番目の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[1], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
 //3 番目の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[2], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
```

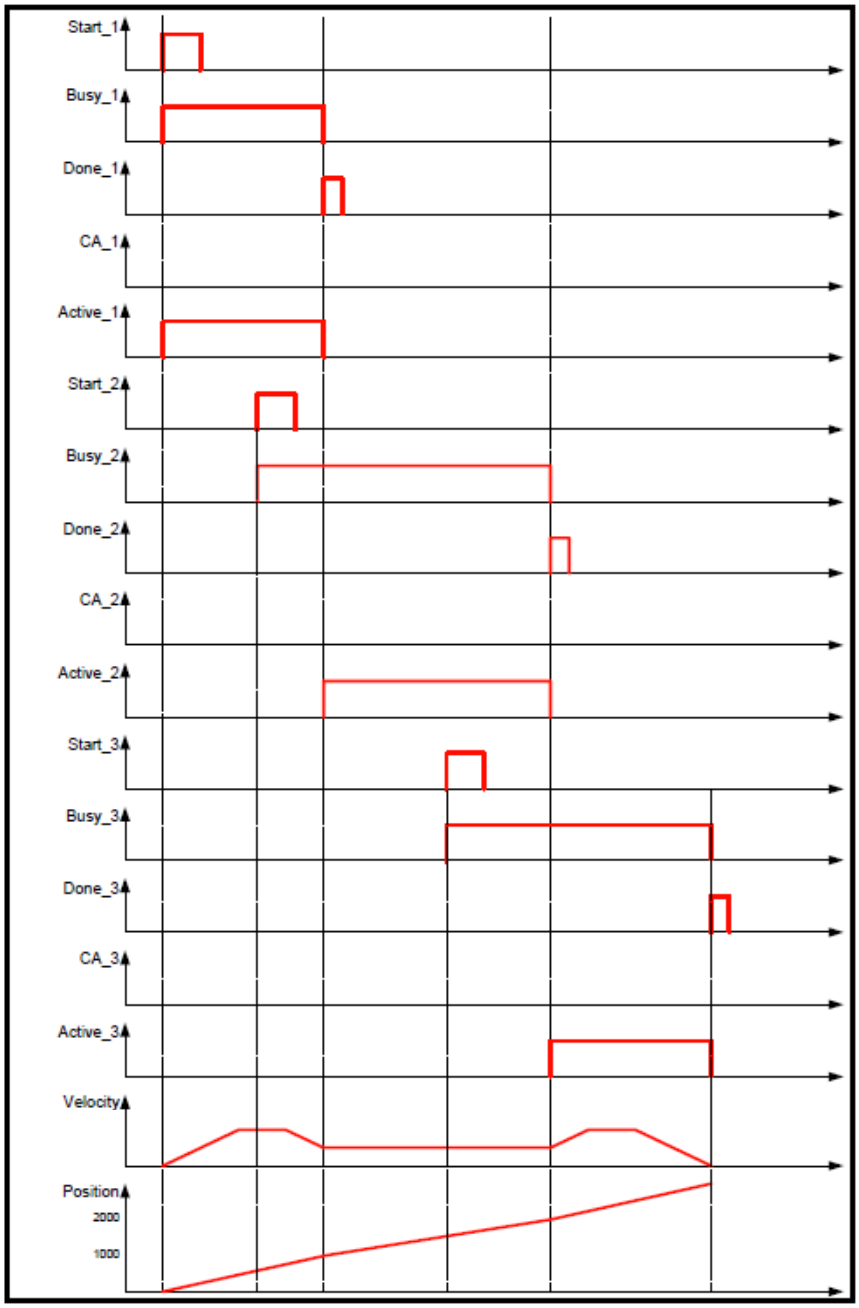


図 3-5-1-4.　上の例のBlendingLow Modeでのタイムチャート

（API 関数 1 の最終位置から API 関数 2 の最終位置まで低い方の速度（velocity2）を使用）

- BlendingPrevious mode での動作

```
//-----------------------------------------------------------------------
//       BlendingPrevious Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------
int                ret;
TFB_MVABS_IN_LIB   mvabs_in;
TFB_MVABS_OUT      mvabs_out;
RTHandle           h_mvabs;
RTHandle           h_wait_mvabs[3];

mvabs_in.Axis            = 1;
mvabs_in.Execute         = 1;
mvabs_in.Position        = 1000.0;
mvabs_in.Velocity        = 100.0;
mvabs_in.Acceleration    = 100.0;
mvabs_in.Deceleration    = 100.0;
mvabs_in.BufferMode      = 0;   //Aborting
mvabs_in.Direction       = 0;
mvabs_in.Jerk            = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //1つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[0]      = h_mvabs;
}
mvabs_in.Axis            = 1;
mvabs_in.Execute         = 1;
mvabs_in.Position        = 2000.0;
mvabs_in.Velocity        = 50.0;
mvabs_in.Acceleration    = 50.0;
mvabs_in.Deceleration    = 50.0;
mvabs_in.BufferMode      = 3;   // BlendingPrevious
mvabs_in.Direction       = 0;
mvabs_in.Jerk            = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //2つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[1]      = h_mvabs;
}
mvabs_in.Axis            = 1;
mvabs_in.Execute         = 1;
mvabs_in.Position        = 3000.0;
mvabs_in.Velocity        = 100.0;
mvabs_in.Acceleration    = 100.0;
mvabs_in.Deceleration    = 100.0;
```

```
mvabs_in.BufferMode       = 3;    // BlendingPrevious
mvabs_in.Direction        = 0;
mvabs_in.Jerk             = 0;
h_mvabs     = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //3つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[2]       = h_mvabs;
}
//動作完了待ち
 //先の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[0], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
 //2 番目の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[1], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
 //3 番目の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[2], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
```

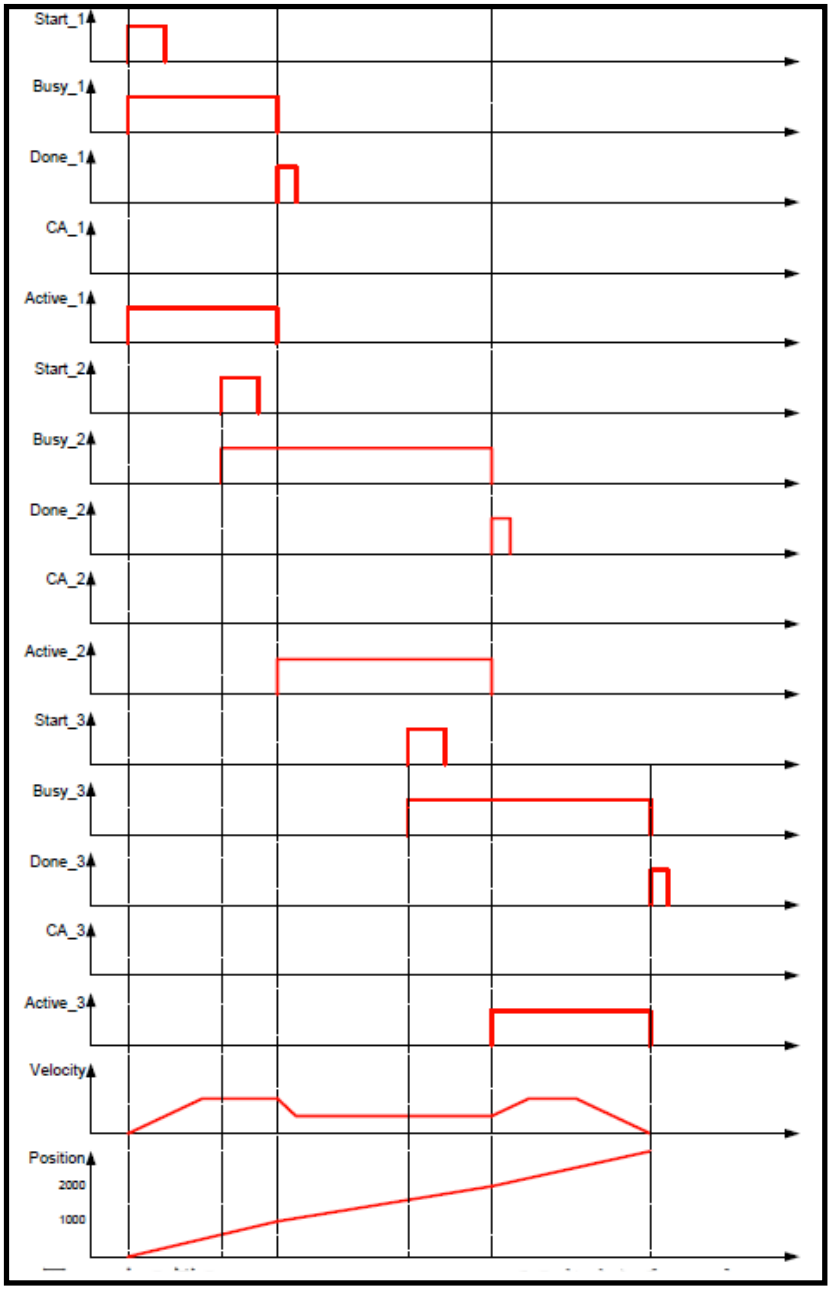


図 3-5-1-5.　上の例のBlendingPrevious modeでのタイムチャート
（API 関数 1 の最終位置では、API 関数 1 の速度を使用）

- BlendingNext mode での動作

```
//-----------------------------------------------------------------------
//      BlendingNext Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------
int                ret;
TFB_MVABS_IN_LIB   mvabs_in;
TFB_MVABS_OUT      mvabs_out;
RTHandle           h_mvabs;
RTHandle           h_wait_mvabs[3];

mvabs_in.Axis            = 1;
mvabs_in.Execute         = 1;
mvabs_in.Position        = 1000.0;
mvabs_in.Velocity        = 100.0;
mvabs_in.Acceleration    = 100.0;
mvabs_in.Deceleration    = 100.0;
mvabs_in.BufferMode      = 0;   //Aborting
mvabs_in.Direction       = 0;
mvabs_in.Jerk            = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //1つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[0]      = h_mvabs;
}
mvabs_in.Axis            = 1;
mvabs_in.Execute         = 1;
mvabs_in.Position        = 2000.0;
mvabs_in.Velocity        = 50.0;
mvabs_in.Acceleration    = 50.0;
mvabs_in.Deceleration    = 50.0;
mvabs_in.BufferMode      = 4;   // BlendingNext
mvabs_in.Direction       = 0;
mvabs_in.Jerk            = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //2つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[1]      = h_mvabs;
}
mvabs_in.Axis            = 1;
mvabs_in.Execute         = 1;
mvabs_in.Position        = 3000.0;
mvabs_in.Velocity        = 100.0;
mvabs_in.Acceleration    = 100.0;
mvabs_in.Deceleration    = 100.0;
```

```
mvabs_in.BufferMode       = 4;    // BlendingNext
mvabs_in.Direction        = 0;
mvabs_in.Jerk             = 0;
h_mvabs     = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //3つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[2]       = h_mvabs;
}
//動作完了待ち
 //先の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[0], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
 //2 番目の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[1], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
 //3 番目の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[2], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
```

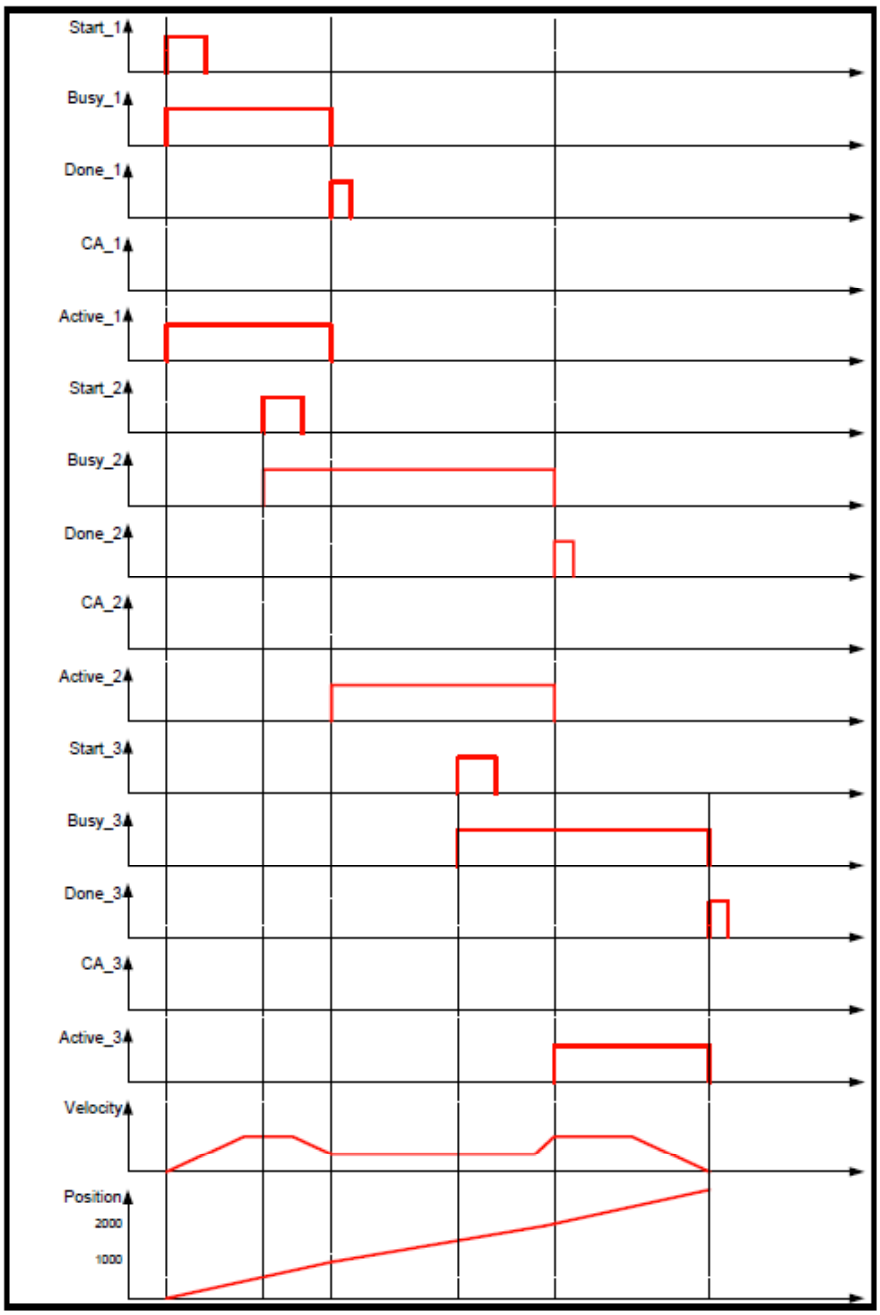


図 3-5-1-6.　上の例のBlendingNext modeでのタイムチャート
（前回の API 関数終了時点では、次の API 関数の速度を使用）

- BlendingHigh mode での動作

```
//-----------------------------------------------------------------------
//       BlendingHigh Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------
int                ret;
TFB_MVABS_IN_LIB   mvabs_in;
TFB_MVABS_OUT      mvabs_out;
RTHandle           h_mvabs;
RTHandle           h_wait_mvabs[3];

mvabs_in.Axis            = 1;
mvabs_in.Execute         = 1;
mvabs_in.Position        = 1000.0;
mvabs_in.Velocity        = 100.0;
mvabs_in.Acceleration    = 100.0;
mvabs_in.Deceleration    = 100.0;
mvabs_in.BufferMode      = 0;   //Aborting
mvabs_in.Direction       = 0;
mvabs_in.Jerk            = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //1つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[0]      = h_mvabs;
}
mvabs_in.Axis_id         = 1;
mvabs_in.Execute         = 1;
mvabs_in.Position        = 2000.0;
mvabs_in.Velocity        = 50.0;
mvabs_in.Acceleration    = 50.0;
mvabs_in.Deceleration    = 50.0;
mvabs_in.BufferMode      = 5;   //BlendingHigh
mvabs_in.Direction       = 0;
mvabs_in.Jerk            = 0;
h_mvabs    = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //2つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[1]      = h_mvabs;
}
mvabs_in.Axis            = 1;
mvabs_in.Execute         = 1;
mvabs_in.Position        = 3000.0;
mvabs_in.Velocity        = 100.0;
mvabs_in.Acceleration    = 100.0;
mvabs_in.Deceleration    = 100.0;
```

```
mvabs_in.BufferMode       = 5;    //BlendingHigh
mvabs_in.Direction        = 0;
mvabs_in.Jerk             = 0;
h_mvabs     = MC_MoveAbsolute(&mvabs_in, &mvabs_out);  //3つ目の動作指令

if(!h_mvabs){
    printf("MC_MoveAbsolute 失敗 RET=%08x", mvabs_out.ErrorID);
}else{
    printf("MC_MoveAbsolute 完了: %08X", h_mvabs);
    h_wait_mvabs[2]       = h_mvabs;
}
//動作完了待ち
 //先の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[0], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
 //2 番目の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[1], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
 //3 番目の動作指令に対する完了応答待ち
memset(&base_out, 0, sizeof(TFB_BASE_OUT));
ret = PO_WaitForMotionRecv(&h_wait_mvabs[2], &base_out, 1000);
if(ret == OK){
    memcpy(&mvabs_out, &base_out, sizeof(TFB_MVABS_OUT));
}
```

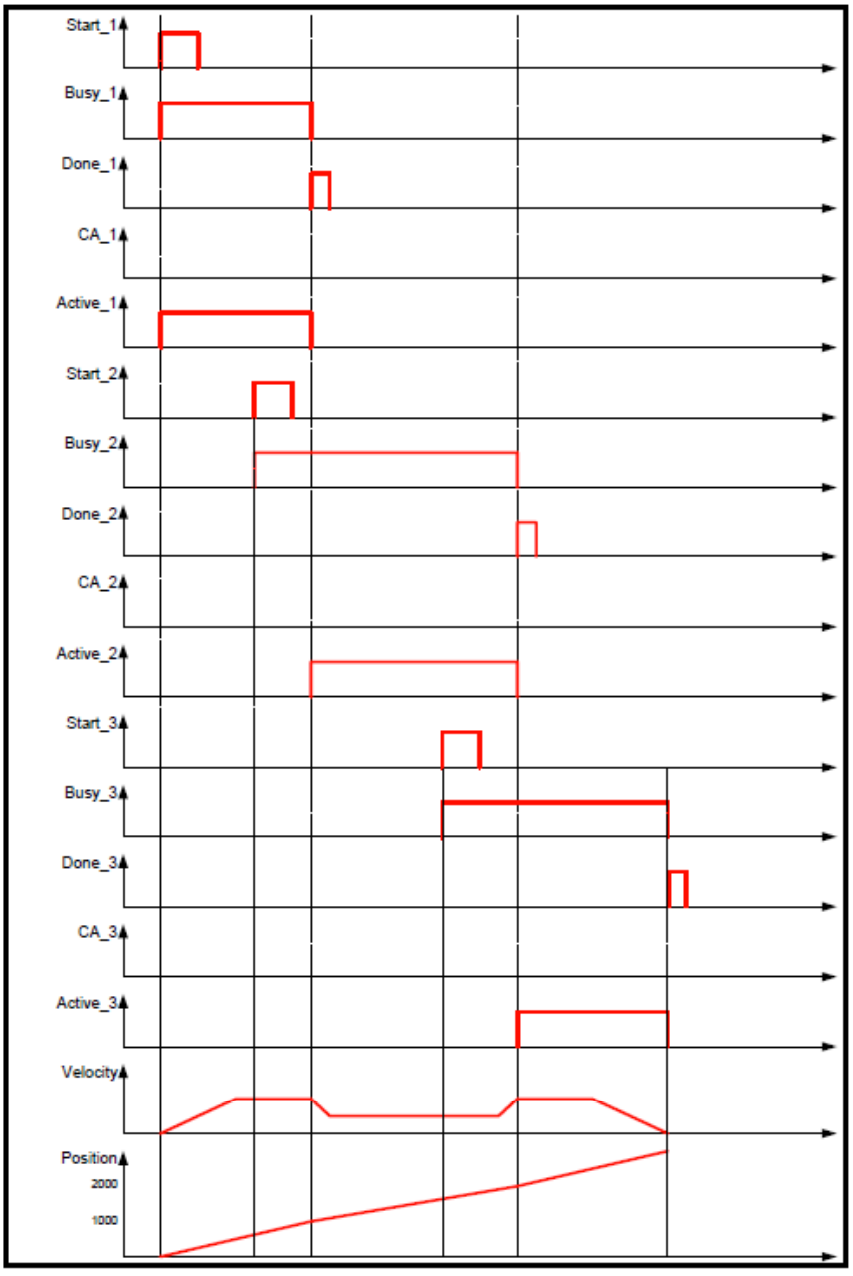


図 3-5-1-7.　上の例のBlendingHigh modeでのタイムチャート
（API 関数 1 最終位置では API 関数 1 の速度を使用、API 関数 2 の最終位置では API 関数 3 の速度を使用）

BufferedMode のうち、ブレンディング系（BlendingLow、BlendingPrevious、BlendingNext、BlendingHigh）の動作をさせる場合いくつか注意事項があります。

１．ブレンディングさせる動作の動作方向は同一方向としてください。反転動作となる指定をされた場合はエラーとなります。

２．MC_MoveVelocity の速度制御系の API 関数から、MC_MoveAbsolute 等の位置制御系の API 関数へのブレンディング動作は、すべて Buffered 動作となります。MC_MoveVelocity の InVelocity が ON した位置からの位置制御となります。
MC_MoveVelocity で加減速中に MC_MoveAbsolute をブレンディングモードで実行した場合で、動作方向が逆転する場合は、InVelocity が ON した後、減速し一度速度が 0 になってから反転動作します。

３．制御中の API 関数で、目標位置到達前に速度を変更する場合、指定された加減速度では、目標位置を越える場合は、加減速度値を急激にして目標位置到達時点で次の速度値になるようにします。

第４章　モーション制御パラメータ

本章では、PLCopen 仕様のモーション制御で使用する軸パラメータについて説明します。

４－１　概要

モーション制御パラメータとしては、PLCopen プロセス内で使用している PLCopen パラメータとサーボパックパラメータの２種類があります。

サーボパックのパラメータについては、各メーカのサーボパックのマニュアルを参照してください。

PLCopen プロセス内で使用しているパラメータについては、ini ファイルで初期値を設定することができます。

各パラメータをリード・ライトするときは、パラメータ型に合ったAPI関数を使用してください。パラメータリード・ライトで使用するAPI関数の一覧を表 4-1-1に示します。

表 4-1-1.　パラメータリード・ライトAPI関数一覧

API 関数名	パラメータ型例	サイズ	内容
MC_ReadParameter	LREAL	8byte	浮動小数点パラメータ読み出し
MC_ReadBoolParameter	BOOL	1bit	BOOL パラメータ読み出し
MC_ReadByteParameter	BYTE SINT USINT INT8 UINT8	1byte	1byte パラメータ読み出し
MC_ReadWordParameter	WORD INT UINT INT16 UINT16	2byte	2byte パラメータ読み出し
MC_ReadDwordParameter	DWORD DINT UDINT INT32 UINT32	4byte	4byte パラメータ読み出し
MC_WriteParameter	LREAL	8byte	浮動小数点パラメータ書き込み
MC_WriteBoolParameter	BOOL	1bit	BOOL パラメータ書き込み
MC_WriteByteParameter	BYTE SINT USINT INT8 UINT8	1byte	1byte パラメータ書き込み
MC_WriteWordParameter	WORD INT UINT INT16 UINT16	2byte	2byte パラメータ書き込み
MC_WriteDwordParameter	DWORD DINT UDINT INT32 UINT32	4byte	4byte パラメータ書き込み

※注：パラメータ型と範囲については、『はじめに ２データタイプ』を参照してください。

４－２　　PLCopenパラメーター覧

PLCopen プロセスで定義されているパラメータは、共通パラメータと各軸毎のパラメータがあります。

４－２－１　共通パラメータ

型名	TPOPEN_SMEM_CONTROL		説明	共通管理領域		
メンバ名	型	パラメータNo	説明			
UseAxis	UINT32	0x00000001	制御軸数	範囲	1～62	
				初期値	1	
EtherCATAddrKind	UINT16	0x00000002	0：EtherCATスレーブはノードアドレスで管理 1：EtherCATスレーブはDipSwで管理	範囲	0 or 1	
				初期値	0	

● MECHATROLINK-Ⅲの場合

MECHATROLINK-Ⅲ通信を使った PLCopen 仕様のスレーブ指定方法は下記のようになります。

・DipSw と MLMstSetting.ini 設定ファイルで管理された論理 ID

MECHATROLINK-Ⅲ通信の場合は、上記表中の「EtherCATAddrKind」は設定不要です。

● EtherCAT の場合

EtherCAT 通信を使った PLCopen 仕様のスレーブ指定方法は下記のようになります。

・DipSw と ACat.ini 設定ファイルで管理された論理 ID

AI-Motion での EtherCAT 通信の場合は、上記表中の「EtherCATAddrKind」は設定不要です。

４－２－２　軸毎パラメータ

<table>
<tr><td>型名</td><td colspan="2">AXIS_REF_CFG</td><td>説明</td><td colspan="3">軸基本設定</td></tr>
<tr><td>メンバ名</td><td>型</td><td>パラメータNo</td><td colspan="4">説明</td></tr>
<tr><td>AxisEnable</td><td>UINT16</td><td>0x00000100</td><td colspan="2">0：この軸は使用しない
1：この軸は使用する</td><td>範囲
初期値</td><td>0 or 1
0</td></tr>
<tr><td>AxisType</td><td>UINT16</td><td>0x00000101</td><td colspan="2">この軸で使用するユニットタイプを指定する
値：タイプ
0x00：EtherCAT：サーボ
0x01：EtherCAT：ALGOSYSTEM モーションコントロールユニット
0x02：EtherCAT：YASKAWA Σ-5シリーズ
0x03：EtherCAT：SANYO SANMORTION-Rシリーズ
0x04：EtherCAT：KOLLMORGEN AKDシリーズ
0x05-0x0F：EtherCAT：リザーブ
0x10-0x1F：MECHATROLINK-Ⅲ：リザーブ
0x20：仮想軸</td><td>範囲
初期値</td><td>0x00 ～ 0x20
0</td></tr>
<tr><td>NodeAddr</td><td>UINT16</td><td>0x00000102</td><td colspan="2">EtherCATの場合：制御するユニットのノードアドレス
Dipswの値、または、コンフィグ後のノードアドレス値</td><td>範囲
初期値</td><td>－
－</td></tr>
<tr><td>NodeSubCh</td><td>UINT16</td><td>0x00000103</td><td colspan="2">1ノードで複数軸動作できるユニットの場合の内部チャンネル指定</td><td>範囲
初期値</td><td>0～7
0</td></tr>
</table>

これらの値は、PLCopen プロセスが起動する前に設定されている必要があります。そのため、ini ファイルにより初期設定値を設定できるようにしてあります。これらの値を変更された後は、INtime のノードを再起動してください。

iniファイル設定方法は、『4-3 iniファイルによるパラメータ初期値設定方法』を参照ください。

<table>
<tr><td>型名</td><td colspan="2">AXIS_REF_SCALE</td><td>説明</td><td colspan="3">単位変換設定</td></tr>
<tr><td>メンバ名</td><td>型</td><td>パラメータNo</td><td colspan="4">説明</td></tr>
<tr><td>Num</td><td>UINT32</td><td>0x00000200</td><td colspan="2">モータ1回転のパルス数</td><td>範囲</td><td>0～
4294967295
【pulse】</td></tr>
<tr><td></td><td></td><td></td><td colspan="2"></td><td>初期値</td><td>0</td></tr>
<tr><td>Den</td><td>LREAL</td><td>0x00000201</td><td colspan="2">モータ1回転の移動量</td><td>範囲</td><td>実数値
【指令単位】</td></tr>
<tr><td></td><td></td><td></td><td colspan="2"></td><td>初期値</td><td>0</td></tr>
<tr><td>Units</td><td>UINT16</td><td>0x00000202</td><td colspan="2">指令単位を指定。
値 / 指令単位
0 パルス【pulse】
1 ミリメートル【mm】
2 マイクロメートル【um】
3 ナノメートル【nm】
4 度【degree】
5 インチ【inch】</td><td>範囲
初期値</td><td>0～5
0</td></tr>
<tr><td>A_Units</td><td>UINT32</td><td>0x00000203</td><td colspan="2">加減速度の内部計算値
安川電機製サーボパックの場合、加減速設定単位が初期値10000倍されています。この倍数を設定することで、PLC-Openプロセス内部計算値を調整します。</td><td>範囲</td><td>1～
4294967295
【pulse/s^2】</td></tr>
<tr><td></td><td></td><td></td><td colspan="2"></td><td>初期値</td><td>1</td></tr>
</table>

モーション制御で使用する際、指令単位とパルス単位の関係を設定するために電子ギアを使用します。電子ギア比の計算は下記のようになります。

- 電子ギア比（単位変換の式）
  指令位置【pulse】 = 指令位置【指令単位】 × 電子ギア比
  電子ギア比 = モータ 1 回転のパルス数【pulse】 ／ モータ 1 回転の移動量【指令単位】

モーション制御命令では目標位置や速度、加減速度を LREAL 型で指定しますが、電子ギア比を使って、パルス単位系に変換しています。変換した後の値がサーボパックで設定できる範囲を超える場合は、命令で異常が発生します。

【設定例】
モータ 1 回転のパルス数 = 1048576【pulse】
ボールネジピッチ = 6【mm】
減速比 = 1/3（モータ 1 回転でボールネジは 1/3 回転する）
モータ 1 回転の移動量 = ボールネジピッチ × 減速比 = 6 × 1/3 = 2【mm】
上記の構成の場合、設定値は下記のようになります。
Num = 1048576　Den = 2　Units = 1
この設定で、モーション制御命令での指令単位は 1【mm】となります。123.4【mm】の位置へ絶対位置移動するときは、MC_MoveAbsolute の Position に 123.4 を設定します。

上記の例で減速比が 1/7 の場合、モータ 1 回転の移動量は 6 × 1/7 = 0.857142857‥‥【mm】となります。
この場合、モータ 1 回転の移動量の設定値を四捨五入したりすると誤差が発生し、目的の位置になりません。
モータ 1 回転の移動量が割り切れない場合は、モータ 1 回転のパルス数とモータ 1 回転の移動量に同じ係数を掛けた値を設定します。
Num = 1048576×7 = 7340032　Den = 6 × 1/7 × 7 = 6　Units = 1

型名	AXIS_REF_MOVE		説明	軸動作設定	
メンバ名	型	パラメータNo	説明		
MaxSpeed	LREAL	0x00000300	最高速度設定 最高速度設定値を超える速度を設定された場合、この設定速度で動作します。	範囲	実数正数値【指令単位/s】
				初期値	400000000
FollowingErrorWindow	LREAL	0x00000301	位置偏差カウンタオーバーフロー値 位置要求値に相対的に許容可能な位置範囲を設定します。	範囲	実数値正数【指令単位】
				初期値	5000000
PositionWindow	LREAL	0x00000302	位置決め完了範囲 ターゲット位置到達として許容可能な範囲を設定します。	範囲	実数値正数、0【指令単位】
				初期値	100
VelocityWindow	LREAL	0x00000303	速度到達範囲 ターゲット速度到達として許容可能な範囲を設定します。	範囲	実数値正数【指令単位/s】
				初期値	50
HomeOffset	LREAL	0x00000304	ホームオフセット メカ原点位置をホームオフセット値で正規化します。	範囲	実数値【指令単位】
				初期値	0
CwSoftLimit	LREAL	0x00000305	正方向ソフトリミット値 正方向側のソフトリミット値を設定	範囲	実数値【指令単位】
				初期値	2147483647
CcwSoftLimit	LREAL	0x00000306	負方向ソフトリミット値 負方向側のソフトリミット値を設定	範囲	実数値【指令単位】
				初期値	-2147483648
ZeroSearchSpeed	LREAL	0x00000307	原点復帰ゼロ点サーチスピード 原点復帰時の低速スピードを設定	範囲	実数値正数【指令単位/s】
				初期値	100000
HomeAcceleration	LREAL	0x00000308	原点復帰加減速度 原点復帰時の加減速度を設定	範囲	実数値正数【指令単位/s】
				初期値	10000000
PositionReadProfileNo	UINT16	0x00000309	ポジション読み出しプロファイル番号 ブレンディングモード時の計算時に位置監視の対象パラメータとして使用。 ReadActualPosition時にも使用	範囲	0x6063 or 0x6064
				初期値	0x6064
VelocityReadProfileNo	UINT16	0x0000030A	速度読み出しプロファイル番号 ブレンディングモード時の計算時に速度監視の対象パラメータとして使用。 ReadActualVelocity時にも使用	範囲	0x606B or 0x606C
				初期値	0x606B
TorqueWindow	LREAL	0x0000030B	トルク到達範囲 ターゲットトルク到達として許容可能な範囲を設定します。	範囲	実数値正数【N.m】or【%】
				初期値	50

型名	AXIS_REF_COUNT		説明	軸カウンタ設定	
メンバ名	型	パラメータNo	説明		
CountMode	UINT16	0x00000400	カウントモード 0:リニアモード(有限長) 1:ロータリモード(無限長)	範囲	0 or 1
				初期値	0
MinPosLimit	LREAL	0x00000401	リングカウンタ下限設定値 カウントモードロータリモードとしたときのリングカウンタ下限値を設定	範囲	実数値【指令単位】
				初期値	-2147483648
MaxPosLimit	LREAL	0x00000402	リングカウンタ上限設定値 カウントモードロータリモードとしたときのリングカウンタ上限値を設定	範囲	実数値【指令単位】
				初期値	2147483647

型名	AXIS_REF_STATUS		説明	軸ステータス
メンバ名	型	パラメータNo	説明	
CtrlStatus	UINT16	0x00010000		範囲 ー 初期値 ー

BIT	名称	内容
0	原点確定	0:原点未確定
		1:原点確定
1	指令速度飽和	0:指令速度は無制限
		1:指令速度が最大速度で制限中
2	動作方向	0:正方向
		1:負方向
3	サーボReady	0:CiA402の状態B
		1:CiA402の状態C
4	主回路電源	0:主回路電源OFF状態
		1:主回路電源ON状態
5	サーボON	0:サーボOFF
		1:サーボON
6	＋EL	0:正方向エンドリミットOFF
		1:正方向エンドリミットON
7	ーEL	0:負方向エンドリミットOFF
		1:負方向エンドリミットON
8	ORG	0:原点信号OFF
		1:原点信号ON
9	ドライブアラーム	0:ドライブアラーム無し
		1:ドライブアラーム発生中
10	ドライブワーニング	0:ドライブワーニング無し
		1:ドライブワーニング発生中

４－２－３　サーボパックパラメータ

本項では、サーボパックで規定されているパラメータをアクセスする方法について説明します。

● MECHATROLINK-Ⅲの場合
MECHATROLINK-Ⅲ通信のサーボパックでは、共通パラメータ／機器パラメータと呼ばれるパラメータが存在し、これらのパラメータは RAM 領域／不揮発メモリ領域を選択して R/W する事が出来ます。

PLCopen 仕様 MC ファンクションブロックの MC_ReadParameter や MC_WriteParameter 等のリード・ライト系の FB では下記のようなパラメータ番号を入力することで、それぞれのサイズ毎にパラメータの読み書きを行います。

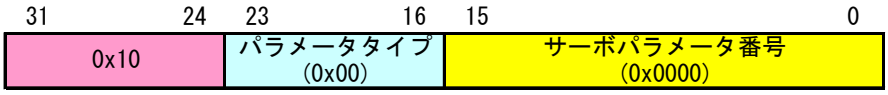


図 4-2-3-1.　R/Wパラメータ番号

パラメータタイプは以下のタイプがあります。

表 4-2-3-1.　サーボパラメータタイプ

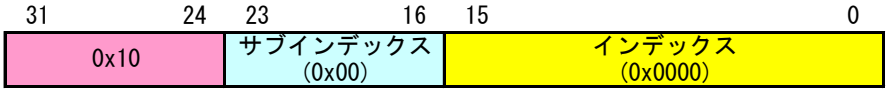
共通パラメータ	RAM 領域	0x00
	不揮発メモリ領域	0x01
機器パラメータ	RAM 領域	0x10
	不揮発メモリ領域	0x11

共通パラメータ／機器パラメータの詳細については、各サーボパックのマニュアルを参照してください。

● EtherCAT の場合

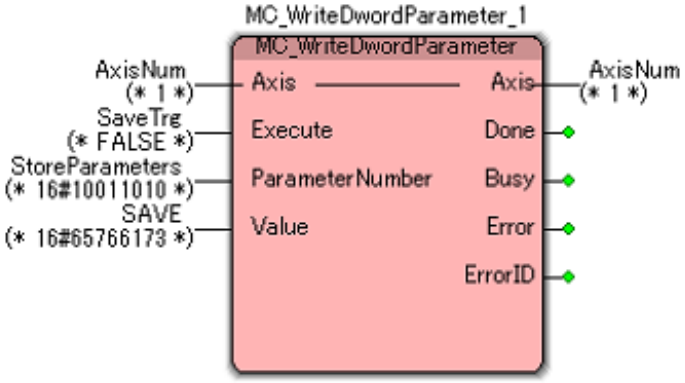
EtherCAT 通信のサーボパックでは、CiA402 で規定されているパラメータ（0x6000 番台）の他に、メーカ独自のパラメータ等が規定されています。

EtherCAT の場合は、インデックス番号（16bit）＋サブインデックス番号（8bit）でアクセスすることができます。MC_ReadParameter や MC_WriteParameter 等のリード・ライト系の API 関数では下記のようなパラメータ番号を入力することで、それぞれのサイズ毎にパラメータの読み書きを行います。



● 設定例

全サーボパラメータを EEPROM へ保存する場合
インデックス番号 = 0x1010
サブインデックス番号 = 0x01



SaveTrg を True にすると、サーボパックに設定されているすべてのパラメータが EEPROM へ保存されます。インデックス番号(0x1010)の使用方法については、各メーカのサーボパックマニュアルを参照してください。

４－３　　iniファイルによるパラメータ初期値設定方法

POpenSetting.ini ファイルは「INtime 版 PLCopen プロセス PLCOpenProc.RTA」を使用する際に必要な設定ファイルです。

本設定ファイルにより「INtime 版 PLCopen プロセス PLCOpenProc.RTA」で使用する軸数や、各軸のタイプ設定を変更することができます。本項では設定ファイル　POpenSetting.ini の設定法について解説します。

設定ファイル　POpenSetting.iniの構成図は図 4-3-1、図 4-3-2のようになります。

● MECHATROLINK-Ⅲの場合
MECHATROLINK-Ⅲ通信マスタが参照している MLMstSetting.ini については、「MLMstSetting.INI 設定マニュアル」を参照してください。

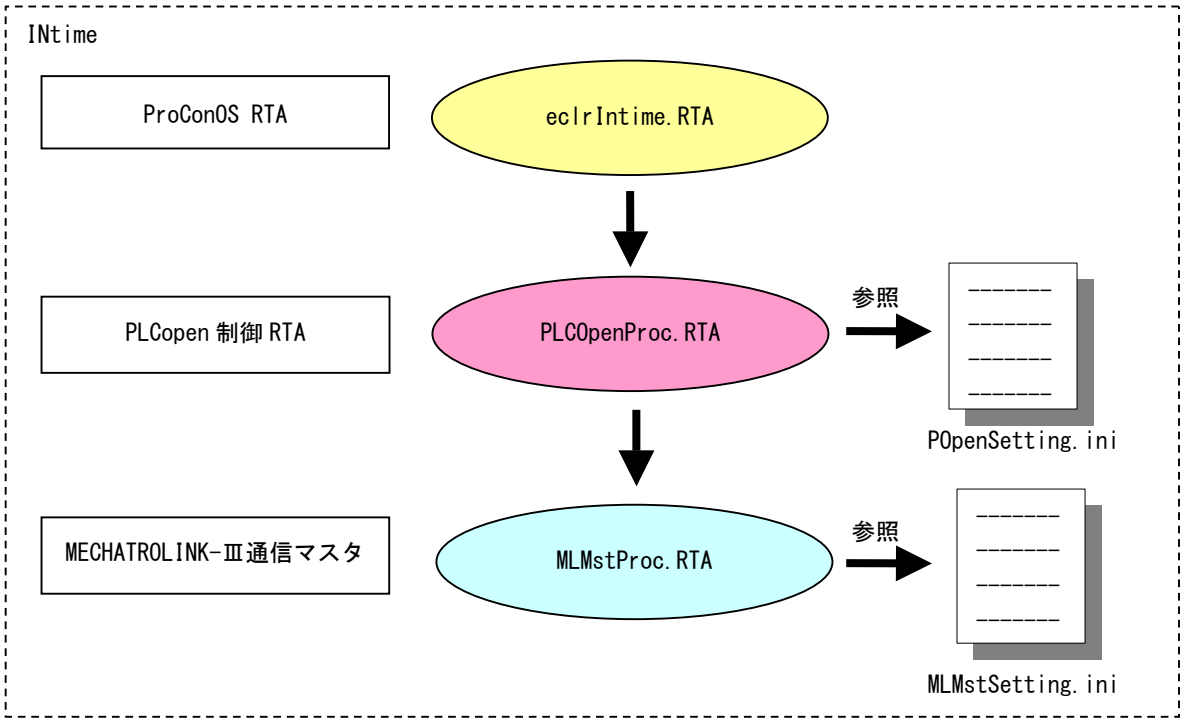


図 4-3-1. INtime版PLC-OPENプロセス構成図（MECHATROLINK-Ⅲ）

● EtherCAT の場合

EtherCAT 通信マスタが参照している ACMst.ini については、「ACMst.ini 設定マニュアル」を参照してください。

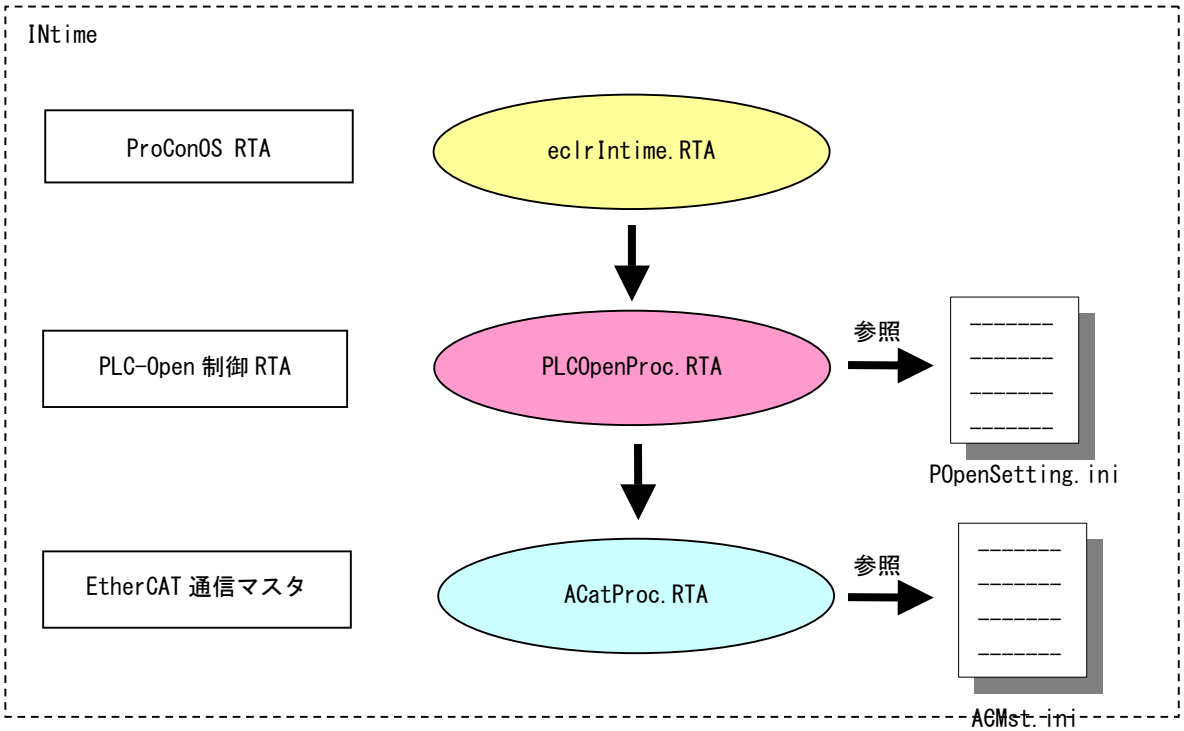


図 4-3-2. INtime版PLC-OPENプロセス構成図（EtherCAT）

４－３－１ POpenSettong.iniファイル

PLCOpenProc.rta は「POpenSetting.ini」ファイルを設定することによって軸タイプおよび通信設定を変更して動作させることができます。

４－３－２　ファイル書式

設定する項目は表 4-3-2-1のようになります。

表 4-3-2-1.　INIファイル設定

セクション名	キー名	備　　考
CONTROL	UseAxis	TPOPEN_SMEM_CONTROL パラメータ参照
	EtherCATAddrKind	
AXIS_n n：1,2,･･･62 (軸番号)	RefCfg_AxisEnable	AXIS_REF_CFG パラメータ参照
	RefCfg_AxisType	
	RefCfg_NodeAddr	
	RefCfg_NodeSubCh	
	RefScale_Num	AXIS_REF_SCALE パラメータ参照
	RefScale_Den	
	RefScale_Units	
	RefScale_A_Units	
	RefMove_MaxSpeed	AXIS_REF_MOVE パラメータ参照
	RefMove_FollowingErrorWindow	
	RefMove_PositionWindow	
	RefMove_VelocityWindow	
	RefMove_HomeOffset	
	RefMove_CwSoftLimit	
	RefMove_CcwSoftLimit	
	RefMove_ZeroSearchSpeed	
	RefMove_HomeAcceleration	
	RefMove_PositionReadProfileNo	
	RefMove_VelocityReadProfileNo	
	RefCount_CountMode	AXIS_REF_COUNT パラメータ参照
	RefCount_MinPosLimit	
	RefCount_MaxPosLimit	

POpenSetting.ini ファイルの例を以下に示します。

```
[CONTROL]
UseAxis=2
EtherCATAddrKind=0

[AXIS_1]
RefCfg_AxisEnable=0
RefCfg_AxisType=0
RefCfg_NodeAddr=1001
RefCfg_NodeSubCh=0
RefScale_Num=1
RefScale_Den=1
RefScale_Units=0
RefScale_A_Units=1
RefMove_MaxSpeed=400000000
RefMove_FollowingErrorWindow=5000000
```

```
RefMove_PositionWindow=100
RefMove_VelocityWindow=50
RefMove_HomeOffset=0
RefMove_CwSoftLimit=0
RefMove_CcwSoftLimit=0
RefMove_ZeroSearchSpeed=100000
RefMove_HomeAcceleration=10000000
RefMove_PositionReadProfileNo=24676
RefMove_VelocityReadProfileNo=24683
RefCount_CountMode=0
RefCount_MinPosLimit=-2147483648
RefCount_MaxPosLimit=2147483647

[AXIS_2]
RefCfg_AxisEnable=0
RefCfg_AxisType=0
RefCfg_NodeAddr=1002
RefCfg_NodeSubCh=0
RefScale_Num=1
RefScale_Den=1
RefScale_Units=0
RefScale_A_Units=1
RefMove_MaxSpeed=400000000
RefMove_FollowingErrorWindow=5000000
RefMove_PositionWindow=100
RefMove_VelocityWindow=50
RefMove_HomeOffset=0
RefMove_CwSoftLimit=0
RefMove_CcwSoftLimit=0
RefMove_ZeroSearchSpeed=100000
RefMove_HomeAcceleration=10000000
RefMove_PositionReadProfileNo=24676
RefMove_VelocityReadProfileNo=24683
RefCount_CountMode=0
RefCount_MinPosLimit=-2147483648
RefCount_MaxPosLimit=2147483647
```

値はすべて 10 進数で指定してください。16 進数での指定は無効です。

４－４　　エラー表示

MC_ReadAxisError で読み出せるエラー番号の詳細を示します。

出力用構造体の ErrorID、MC_ReadAxisError で読み出せるエラー番号の詳細を示します。

４－４－１　MECHATROLINK-Ⅲライブラリ異常

エラー番号	エラー名	内容
0x00000001	オープン済み	すでにオープンしている
0x00000002	マスタ未起動	MECHATROLINK-Ⅲマスタが起動していない
0x00000003	引数無効	無効な引数
0x00000004	オープン未完	オープンしていない
0x00000005	通信開始済み	すでに通信開始されている
0x00000006	通信開始未完	通信していない
0x00000007	通信リセット未完	リセットされていない
0x00000008	同期通信異常	同期通信されていない
0x00000009	通信初期化異常	通信初期化エラー
0x0000000A	コマンド無し	コマンドセットされていない
0x0000000B	レスポンス無し	レスポンスなし
0x00000101	内部異常	内部エラー
0x00000102	内部リソース生成失敗	各種デバイス生成失敗
0x00000103	タイムアウト異常	タイムアウトエラー
0x00000201	デバイスドライバ異常	デバイスドライバーロードエラー
0x00000202	プロファイル異常	プロファイルタイプが異なっている
0x00000203	ユニット異常	ユニットタイプが異なっている
0x00000204	コマンド重複異常	コマンド実行中に別のコマンドが実行された

４－４－２　EtherCAT通信異常

エラー番号	エラー名	内容
0x10000000	スレーブノードアドレス重複	EtherCAT のノードアドレスが重複しています。
0x10000001	リンクオフが発生	通信断が発生しました。
0x10000002	ネットワーク構成異常	ネットワークの構成情報に異常があります。
0x10000003	スレーブ初期化異常	スレーブの初期化に失敗しました。
0x10000004	プロセスデータ送受信エラー	データの送受信に失敗しました。
0x10000005	スレーブアドレスエラー	スレーブのアドレスに異常があります。
0x10000006	スレーブタイプが EtherCAT でない	スレーブタイプが EtherCAT でありません。

４－４－３　機器異常

エラー番号	エラー名	内容
0x2000XXXX	サーボパック内部エラー	サーボパック内部で発生したエラーです。 下位 16bit がサーボパック側のエラー番号となります。 エラー内容については、各社サーボパックマニュアルを参照してください。

４－４－４　コマンド異常

エラー番号	エラー名	内容
0x30000000	スレッド生成エラー	スレッドの生成に失敗しました。
0x30000001	CPOpenMCPart1 生成エラー	PLCopen Part1 制御クラスの生成に失敗しました。
0x30000002	CPOpenMCPart5 生成エラー	PLCopen Part5 制御クラスの生成に失敗しました。
0x30000101	MC_Power エラー	
0x30000102	MC_Stop エラー	
0x30000103	MC_Home エラー	
0x30000201	Homing 異常停止	原点復帰が異常停止しました。
0x30000202	Homing タイムアウト	原点復帰が指定された時間以内に完了しませんでした。
0x30000203	Homing ポジションリミット	原点復帰が指定された移動量以内で完了しませんでした。
0x30000204	Homing 未サポート	指定された原点復帰モードは、サーボパック側でサポートされていません。
0x30000301	Buffered 方向異常	API 関数多重起動のブレンディング動作時に、もともと動作していた API 関数の動作方向と、多重起動された API 関数の動作方向が違います。
0x30000401	型タイプ異常	パラメータの型があっていません。

４－４－５　API関数インスタンス異常

エラー番号	エラー名	内容
0x40000001	Buffered 未対応	仕様範囲外の Buffered 命令
0x40000002	ステータス異常	API 関数呼び出しが仕様範囲外の状態で実行された
0x40000003	API 関数入力値異常	API 関数入力値が異常です。
0x40000004	同時読み込み	パラメータ読み込み API 関数が同時に複数実行された
0x40000005	同時書き込み	パラメータ書き込み API 関数が同時に複数実行された

４－４－６　EtherCATマスタ異常

エラー番号	エラー名	内容
0x90000001	ACAT_ER_ALREADYOPEN	既にオープンしている
0x90000002	ACAT_ER_NOTOPEN	オープンされていない
0x90000003	ACAT_ER_INVALIDPARAM	無効な引数
0x90000004	ACAT_ER_OPENDEVICE	デバイスドライバー起動失敗
0x90000005	ACAT_ER_CREATETHREAD	スレッド作成失敗
0x90000006	ACAT_ER_CREATESEMAPH	セマフォ作成失敗
0x90000007	ACAT_ER_CREATEMAP	マップトファイル作成失敗
0x90000008	ACAT_ER_CREATEMAIL	メイルスロット作成失敗
0x90000009	ACAT_ER_INICONFIG	設定ファイルの記述エラー
0x9000000A	ACAT_ER_ALREADYSTART	スタートしている
0x9000000B	ACAT_ER_NOTSTART	スタートしていない

0x9000000C	ACAT_ER_DEVICE	デバイスアクセスエラー
0x9000000D	ACAT_ER_LOADDEVICE	デバイスロードエラー
0xFFFFFFFF	ACAT_ER_UNITTYPE	ユニットタイプエラー
0x90000101	ACAT_ER_MST_NOTSUPPORT	未サポートエラー
0x90000102	ACAT_ER_MST_INVALIDINDEX	無効なインデックス
0x90000103	ACAT_ER_MST_INVALIDOFFSET	無効なオフセット
0x90000104	ACAT_ER_MST_CANCEL	キャンセルエラー
0x90000105	ACAT_ER_MST_INVALIDSIZE	無効なサイズ
0x90000106	ACAT_ER_MST_INVALIDDATA	無効なデータ
0x90000107	ACAT_ER_MST_NOTREADY	準備ができていない
0x90000108	ACAT_ER_MST_BUSY	BUSY 状態
0x90000109	ACAT_ER_MST_ACYC_FRM_FREEQ_EMPTY	EtherCAT コマンドが送信できない
0x9000010A	ACAT_ER_MST_NOMEMORY	メモリ不足
0x9000010B	ACAT_ER_MST_INVALIDPARM	無効なパラメータ
0x9000010C	ACAT_ER_MST_NOTFOUND	EtherCAT マスタが見つからない
0x9000010E	ACAT_ER_MST_INVALIDSTATE	無効なステート
0x9000010F	ACAT_ER_MST_TIMER_LIST_FULL	タイマーリストにスレーブを追加できない
0x90000110	ACAT_ER_MST_TIMEOUT	タイムアウト
0x90000111	ACAT_ER_MST_OPENFAILED	オープンできない
0x90000112	ACAT_ER_MST_SENDFAILED	送信できない
0x90000113	ACAT_ER_MST_INSERTMAILBOX	MailBox に追加できない
0x90000114	ACAT_ER_MST_INVALIDCMD	無効な MailBox コマンド
0x90000115	ACAT_ER_MST_UNKNOWN_MBX_PROTOCOL	未定義な MailBox プロトコル
0x90000116	ACAT_ER_MST_ACCESSDENIED	アクセス拒否
0x9000011A	ACAT_ER_MST_PRODKEY_INVALID	無効なプロダクトキー
0x9000011B	ACAT_ER_MST_WRONG_FORMAT	コンフィグファイルフォーマット警告
0x9000011C	ACAT_ER_MST_FEATURE_DISABLED	特色無効
0x9000011D	ACAT_ER_MST_SHADOW_MEMORY	間違ったモードで要求されたメモリ
0x9000011E	ACAT_ER_MST_BUSCONFIG_MISMATCH	バス設定が誤り
0x9000011F	ACAT_ER_MST_CONFIGDATAREAD	コンフィグファイル読み込みエラー
0x90000121	ACAT_ER_MST_XML_CYCCMDS_MISSING	サイクリックコマンド送信失敗
0x90000122	ACAT_ER_MST_XML_ALSTATUS_READ_MISSING	ALSTATUS 読み込み失敗
0x90000123	ACAT_ER_MST_MCSM_FATAL_ERROR	McSm の致命的エラー
0x90000124	ACAT_ER_MST_SLAVE_ERROR	スレーブエラー
0x90000125	ACAT_ER_MST_FRAME_LOST	フレームロスト, IDX ミスマッチ
0x90000126	ACAT_ER_MST_CMD_MISSING	受信フレームの中で EtherCAT コマンドが失敗
0x90000128	ACAT_ER_MST_INVALID_DCL_MODE	無効な DC ラッチモード
0x90000129	ACAT_ER_MST_AI_ADDRESS	Auto increment address 異常(スレーブ異常)
0x9000012A	ACAT_ER_MST_INVALID_SLAVE_STATE	無効なステート(スレーブ異常)
0x9000012B	ACAT_ER_MST_SLAVE_NOT_ADDRESSABLE	無効なアドレス(スレーブ異常)
0x9000012C	ACAT_ER_MST_CYC_CMDS_OVERFLOW	コンフィグファイルで多くのサイクリックコマンドが実行された
0x9000012D	ACAT_ER_MST_LINK_DISCONNECTED	Ethernet ケーブルが切断されました
0x9000012E	ACAT_ER_MST_MASTERCORE_INACCESSIBLE	EtherCAT マスタにアクセスできません
0x9000012F	ACAT_ER_MST_COE_MBXSND_WKC_ERROR	CoE MailBox(送信)の working counter エラー
0x90000130	ACAT_ER_MST_COE_MBXRCV_WKC_ERROR	CoE MailBox(受信)の working counter エラー
0x90000131	ACAT_ER_MST_NO_MBX_SUPPORT	MailBox 未サポート
0x90000132	ACAT_ER_MST_NO_COE_SUPPORT	CoE 未サポート
0x90000133	ACAT_ER_MST_NO_EOE_SUPPORT	EoE 未サポート

0x90000134	ACAT_ER_MST_NO_FOE_SUPPORT	FoE 未サポート
0x90000135	ACAT_ER_MST_NO_SOE_SUPPORT	SoE 未サポート
0x90000136	ACAT_ER_MST_NO_VOE_SUPPORT	VoE 未サポート
0x90000137	ACAT_ER_MST_EVAL_VIOLATION	評価の構成に違反がありました
0x90000138	ACAT_ER_MST_EVAL_EXPIRED	評価は時間リミットに達しました
0x90000201	ACAT_ER_MST_SDO_ABORTCODE_TOGGLE	SDO Abort(トグルビットは交替されませんでした)
0x90000202	ACAT_ER_MST_SDO_ABORTCODE_TIMEOUT	SDO Abort(SDO プロトコル タイムアウト)
0x90000203	ACAT_ER_MST_SDO_ABORTCODE_CCS_SCS	SDO Abort(Client/Server Command が無効か無知)
0x90000204	ACAT_ER_MST_SDO_ABORTCODE_BLK_SIZE	SDO Abort(無効なブロックサイズ)
0x90000205	ACAT_ER_MST_SDO_ABORTCODE_SEQNO	SDO Abort(無効なシーケンス番号)
0x90000206	ACAT_ER_MST_SDO_ABORTCODE_CRC	SDO Abort(CRC エラー)
0x90000207	ACAT_ER_MST_SDO_ABORTCODE_MEMORY	SDO Abort(メモリ範囲外)
0x90000208	ACAT_ER_MST_SDO_ABORTCODE_ACCESS	SDO Abort(未サポートアクセス)
0x90000209	ACAT_ER_MST_SDO_ABORTCODE_WRITEONLY	SDO Abort(WriteOnly エリアを読み出した)
0x9000020A	ACAT_ER_MST_SDO_ABORTCODE_READONLY	SDO Abort(ReadOnly エリアに書き込んだ)
0x9000020B	ACAT_ER_MST_SDO_ABORTCODE_INDEX	SDO Abort(無効な Index 番号)
0x9000020C	ACAT_ER_MST_SDO_ABORTCODE_PDO_MAP	SDO Abort(PDO マッピングできないオブジェクト)
0x9000020D	ACAT_ER_MST_SDO_ABORTCODE_PDO_LEN	SDO Abort(PDO Length が間違えている)
0x9000020E	ACAT_ER_MST_SDO_ABORTCODE_P_INCOMP	SDO Abort(一般パラメータの不一致)
0x9000020F	ACAT_ER_MST_SDO_ABORTCODE_I_INCOMP	SDO Abort(内部情報の不一致)
0x90000210	ACAT_ER_MST_SDO_ABORTCODE_HARDWARE	SDO Abort(ハードウェア)
0x90000211	ACAT_ER_MST_SDO_ABORTCODE_DATA_SIZE	SDO Abort(データタイプ:パラメータミスマッチ)
0x90000212	ACAT_ER_MST_SDO_ABORTCODE_DATA_SIZE1	SDO Abort(データタイプ：パラメータ too long)
0x90000213	ACAT_ER_MST_SDO_ABORTCODE_DATA_SIZE2	SDO Abort(データタイプ：パラメータ too short)
0x90000214	ACAT_ER_MST_SDO_ABORTCODE_OFFSET	SDO Abort(サブインデックス)
0x90000215	ACAT_ER_MST_SDO_ABORTCODE_DATA_RANGE	SDO Abort(書込み：パラメータ範囲外)
0x90000216	ACAT_ER_MST_SDO_ABORTCODE_DATA_RANGE1	SDO Abort(書込み：パラメータ上限超)
0x90000217	ACAT_ER_MST_SDO_ABORTCODE_DATA_RANGE2	SDO Abort(書込み：パラメータ下限以下)
0x90000218	ACAT_ER_MST_SDO_ABORTCODE_MINMAX	SDO Abort(最大値が最小値よりも低い)
0x90000219	ACAT_ER_MST_SDO_ABORTCODE_GENERAL	SDO Abort(一般エラー)
0x9000021A	ACAT_ER_MST_SDO_ABORTCODE_TRANSFER	SDO Abort(データの転送/保存はできない)
0x9000021B	ACAT_ER_MST_SDO_ABORTCODE_TRANSFER1	SDO Abort(ローカル制御からデータの転送/保存はできない)
0x9000021C	ACAT_ER_MST_SDO_ABORTCODE_TRANSFER2	SDO Abort(現在の state ではデータの転送/保存はできない)
0x9000021D	ACAT_ER_MST_SDO_ABORTCODE_DICTIONARY	SDO Abort(オブジェクトディクショナリエラー)
0x9000021E	ACAT_ER_MST_SDO_ABORTCODE_UNKNOWN	SDO Abort(unknown code)
0x90000301	ACAT_ER_MST_CFGFILENOTFOUND	コンフィグファイルが見つからない
0x90000302	ACAT_ER_MST_EEPROMREADERROR	EEPROM 読み込みエラー
0x90000303	ACAT_ER_MST_EEPROMWRITEERROR	EEPROM 書込みエラー
0x90000304	ACAT_ER_MST_XML_CYCCMDS_SIZEMISMATCH	サイクリックコマンドサイズ不一致
0x90000305	ACAT_ER_MST_XML_INVALID_INP_OFF	サイクリックコマンド、無効入力オフセット
0x90000306	ACAT_ER_MST_XML_INVALID_OUT_OFF	サイクリックコマンド、無効出力オフセット
0x90000307	ACAT_ER_MST_PORTCLOSE	ポートクローズ失敗
0x90000308	ACAT_ER_MST_PORTOPEN	ポートオープン失敗
0x90000309	ACAT_ER_MST_SLAVE_NOT_PRESENT	スレーブは Bus に出席していません
0x9000030A	ACAT_ER_MST_NO_FOE_SUPPORT_BS	FoE プロトコルは Boot Strap をサポートしません
0x9000030B	ACAT_ER_MST_EEPROMRELOADERROR	EEPROM ReLoad エラー

0x9000030C	ACAT_ER_MST_SLAVECTRLRESETERROR	スレーブコントローラリセットエラー
0x9000030D	ACAT_ER_MST_SYSDRIVERMISSING	ドライバをオープンできません
0x9000030E	ACAT_ER_MST_BUSCONFIG_TOPOCHANGE	changed Topology のバス設定を検出できません
0x9000030F	ACAT_ER_MST_EEPROMASSIGNERROR	EEPROM の割り当てに失敗しました
0x90000310	ACAT_ER_MST_MBX_ERROR_TYPE	MailBox 受信エラー
0x90000311	ACAT_ER_MST_REDLINEBREAK	Redundancy line break
0x90000312	ACAT_ER_MST_XML_INVALID_CMD_WITH_RED	Redundancy の無効な EtherCAT コマンドを受信
0x90000313	ACAT_ER_MST_XML_PREV_PORT_MISSING	<PreviousPort>-tag 失敗
0x90000314	ACAT_ER_MST_XML_DC_NOT_ALLOWED_WITH_RED	Redundancy の DC アクセスを許さない
0x90000315	ACAT_ER_MST_DLSTATUS_IRQ_TOPOCHANGED	changed Topology の DL Status 割込み
0x9000031C	ACAT_ER_MST_DC_REF_CLOCK_SYNC_OUT_UNIT_DISABLED	DC リファレンスクロック無効
0x9000031D	ACAT_ER_MST_DC_REF_CLOCK_NOT_FOUND	DC リファレンスクロックが見つからない
0x9000031E	ACAT_ER_MST_XML_DC_REF_CLOCK_NOT_FIRST	最初のスレーブからリファレンスクロックが有効にならない
0x9000031F	ACAT_ER_MST_MBX_CMD_WKC_ERROR	MailBox コマンド working counter エラー
0x90000401	ACAT_ER_MAX_BUS_SLAVES_EXCEEDED	Bus Slaves が最大数を越えています
0x90000402	ACAT_ER_MBX_SYNTAX	MailBox ヘッダーが間違えています
0x90000403	ACAT_ER_MBX_UNSUPPORTEDPROTOCOL	MailBox プロトコルがサポートされていません
0x90000404	ACAT_ER_MBX_INVALIDCHANNEL	無効なチャンネルが選択されています
0x90000405	ACAT_ER_MBX_SERVICENOTSUPPORTED	MailBox プロトコルヘッダーが間違えています
0x90000406	ACAT_ER_MBX_INVALIDHEADER	MailBox プロトコルヘッダーが間違えています
0x90000407	ACAT_ER_MBX_SIZETOOSHORT	受信した MailBox データ長が Short ではありません
0x90000408	ACAT_ER_MBX_NOMOREMEMORY	MailBox プロトコルを受取るリソースがありません
0x90000409	ACAT_ER_MBX_INVALIDSIZE	無効なサイズです
0x90000501	ACAT_ER_DCM_NOTINITIALIZED	DC モード初期化に失敗しました
0x90000502	ACAT_ER_DCM_MAX_CTL_ERROR_EXCEED	DC 同期通信に失敗しました
0x90000503	ACAT_ER_DCM_NOMEMORY	十分なメモリが確保できません
0x90000504	ACAT_ER_DCM_INVALID_HWLAYER	無効なハードウェアレイヤー
0x90000505	ACAT_ER_DCM_TIMER_MODIFY_ERROR	ハードウェアレイヤーエラー（タイマ補正）
0x90000506	ACAT_ER_DCM_TIMER_NOT_RUNNING	ハードウェアレイヤーエラー（タイマ未動作）
0x90000507	ACAT_ER_DCM_WRONG_CPU	ハードウェアレイヤーエラー（Wrong CPU）
0x90000508	ACAT_ER_DCM_INVALID_SYNC_PERIOD	無効な同期ピリオド
0x90000509	ACAT_ER_DCM_INVALID_SETVAL	SetVal が小さい
0x9000050A	ACAT_ER_DCM_DRIFT_TO_HIGH	タイマとリファレンスクロックの間のドリフトが高い

第５章　付録

５－１　参考文献

●「IEC61131-3 を用いた PLC プログラミング」

著者　K.-H.John ／ M.Tiegelkamp
監訳者　PLCopen Japan
発行者　深田　良治
発行所　シュプリンガー・フェアラーク東京株式会社
発行年　2006 年

本 DVD には KW-Software 社提供の MULTIPROG に関するマニュアルも収録しております。
MULTIPROG の使用方法に関する詳細などはそちらを参照してください。
各マニュアルは<DVD>¥doc¥に収録されています。
また、サンプルコードも<DVD>¥sample¥に収録されています。こちらも参考にしてください。

このユーザーズマニュアルについて

(1) 本書の内容の一部又は全部を当社からの事前の承諾を得ることなく、無断で複写、複製、掲載することは固くお断りします。
(2) 本書の内容に関しては、製品改良のためお断りなく、仕様などを変更することがありますのでご了承ください。
(3) 本書の内容に関しては万全を期しておりますが、万一ご不審な点や誤りなどお気付きのことがございましたらお手数ですが巻末記載の弊社までご連絡ください。その際、巻末記載の書籍番号も併せてお知らせください。

７７ＰＯ１００００２Ｂ　２０１４年　９月　第２版
７７ＰＯ１００００２Ａ　２０１２年　７月　初版

株式会社アルゴシステム

本社
〒５８７－００２１　大阪府堺市美原区小平尾６５６番地
ＴＥＬ(０７２)３６２－５０６７
ＦＡＸ(０７２)３６２－４８５６

ホームページ　http://www.algosystem.co.jp