リファレンスマニュアル

同期モーション制御
(PLCopen 仕様)
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為に、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。

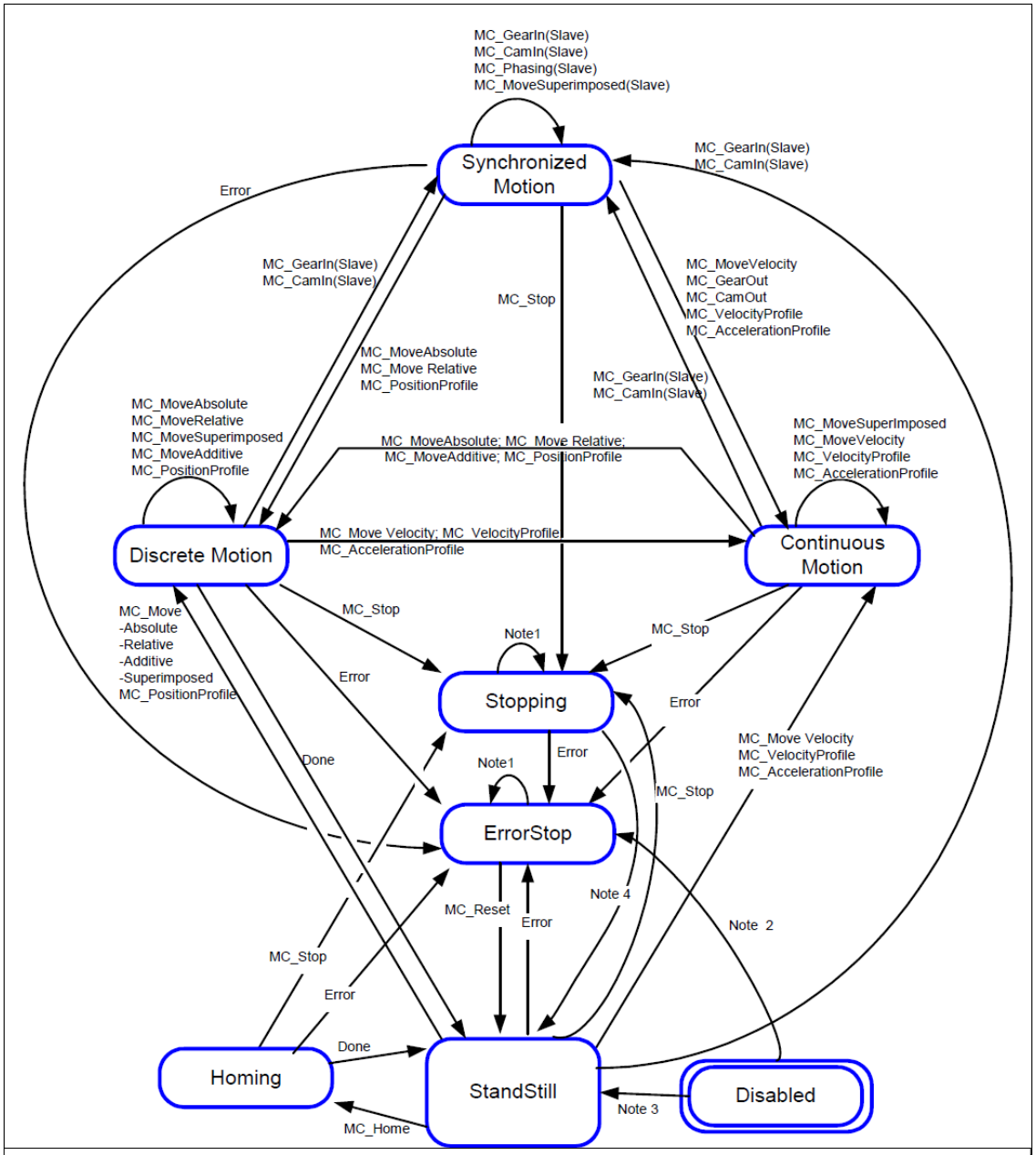
PLCopen の MC 仕様は動作仕様だけでなく、FB の起動方法や状態まで定義しています。これにより、ハードウェアへの依存性を低減しています。
PLCopen の MC 仕様は表 1-1-1 の 6 つの Part から構成されています。

表 1-1-1. PLCopenMC 技術仕様の種類と状況

	仕様	状況	内容
Part1	基本仕様	◎	単軸、多軸制御および、管理用の命令を準備
Part2	拡張仕様	◎	Part1 に対して拡張機能を追加 （Ver2.0 にて Part1 と Part2 は統合化される）
Part3	ユーザガイドライン	○	ユーザのモーションをガイドラインとして準備
Part4	補間制御仕様	○	補間制御および、管理用の命令を準備
Part5	原点復帰追加仕様	○	原点復帰に関して、Part1 への追加仕様を準備
Part6	流体パワー拡張仕様	△	流体パワーデバイスやシステムとのプログラミングの統合化のために規定

◎：英語、日本語仕様公開済み ○：英語仕様公開済み △：仕様策定中

PLCopen の MC 仕様 Part1、Part2、Part5 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ファンクションブロックが決まっています。この状態遷移図を図 1-1-1 に示します。



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 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ファンクションブロックについては、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_ReadActualTorque; 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 パート 1－Basics Version 1.1」と「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 パート 2－Extensions Version 1.0」、「Technical Paper PLCopen Technical Committee 2 – Task Force Function Blocks for motion control: Part 5 - Homing Version 0.99」を参照してください。

同期制御（直線補間や円弧補間）を行うためには、Part4 で規定されている仕様を満足する必要があります。Part4 の状態遷移図を図 1-1-2 に示します。

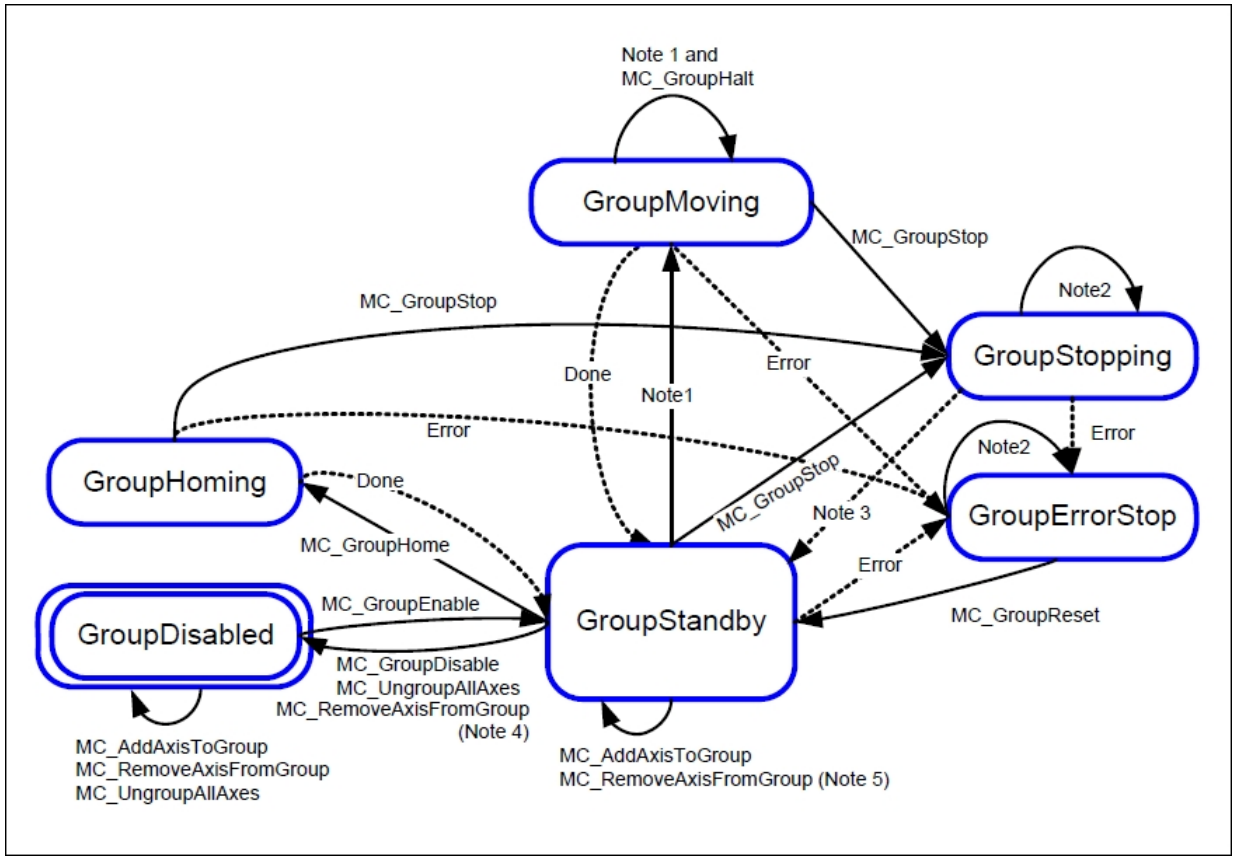


図 1-1-2. PLCopenMC Part4 状態遷移図

Note1:全ての動作系ファンクションブロックに適用されます。
Note2:MC_GroupReset によるエラー解除成功以外、全てのファンクションブロックに適用されます。
Note3:MC_GroupStop.DONE=TRUE かつ MC_GroupStop.EXECUTE=FALSE の時に遷移します。
Note4:最後の軸が Remove された際に遷移します。
Note5:グループが空ではない時に遷移します。

また、単軸の状態遷移図と Part4 状態との相関図を図 1-1-3 に示します。

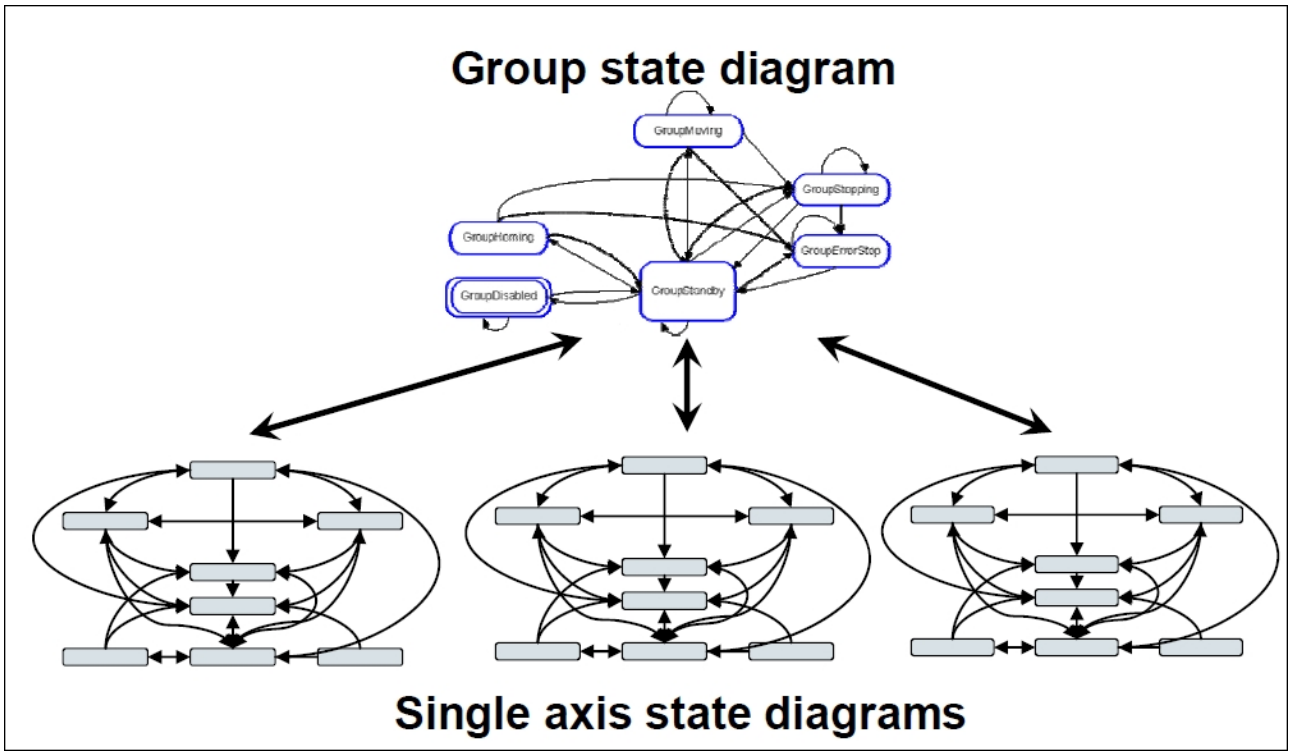


図 1-1-3. PLCopenMC Part4 単軸状態との相関図

動作系ファンクションブロックについて、軸グループに登録済み(MC_AddAxisToGroup 実行済み)の単軸に対して動作指令を実行すると単軸の動作指令はエラーとなります。(この時、軸グループの動作は停止せず、単軸エラーもエラー状態に遷移しません。)

その他の詳細については、「Technical Paper PLCopen Technical Committee 2 – Task Force Function Blocks for motion control:Part 4 –Coordinated Motion Version 1.0」を参照してください。

１－２　　MECHATROLINK-Ⅲ とは

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。

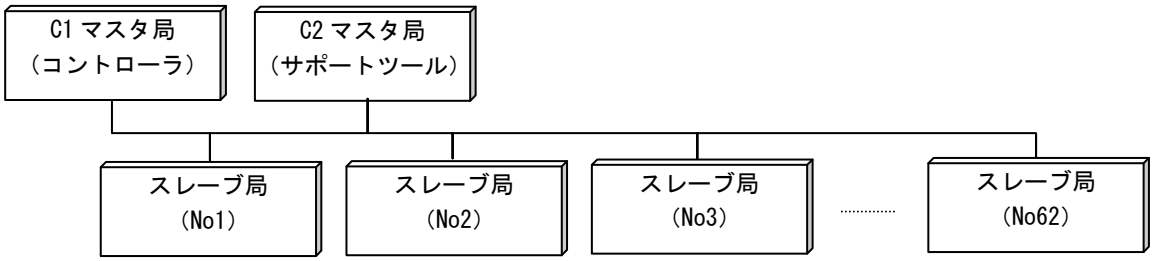


図 1-2-1. MECHATROLINK-Ⅲ接続図

モーション同期制御を実現するために、(株) テクノが開発したモーションライブラリを組み込んでいます。同期制御を組み込んでいないものと一部仕様が異なる部分があります。

１－３　　EtherCAT 通信とは

１－３－１　　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 転送方法はマスター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。

１－３－２　　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け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。

モーション同期制御を実現するために､(株)テクノが開発したモーションライブラリを組み込んでいます。同期制御を組み込んでいないものと一部仕様が異なる部分があります。

１－４　　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 1-3-1 に示します。

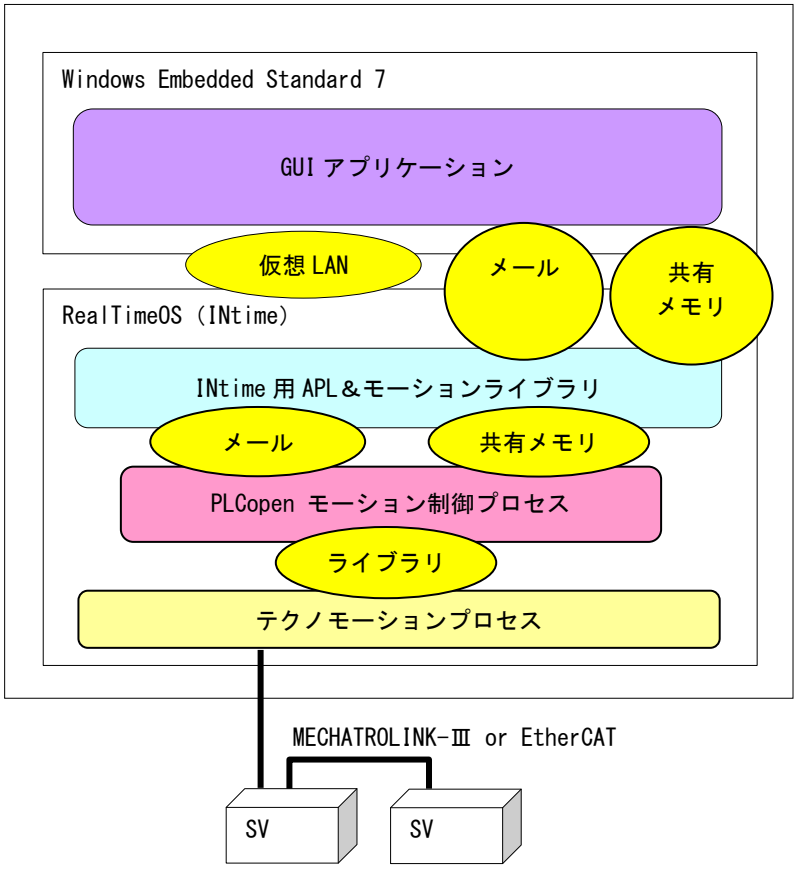


図 1-4-1. 全体構成図

Visual Studio で開発したプログラムを実行します。
Intime 用 APL・モーションライブライブラリから、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テクノモーションプロセスを通じてサーボパックを制御します。

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とPOMotionP4.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、POMotion.RSL、POMotionP4.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。

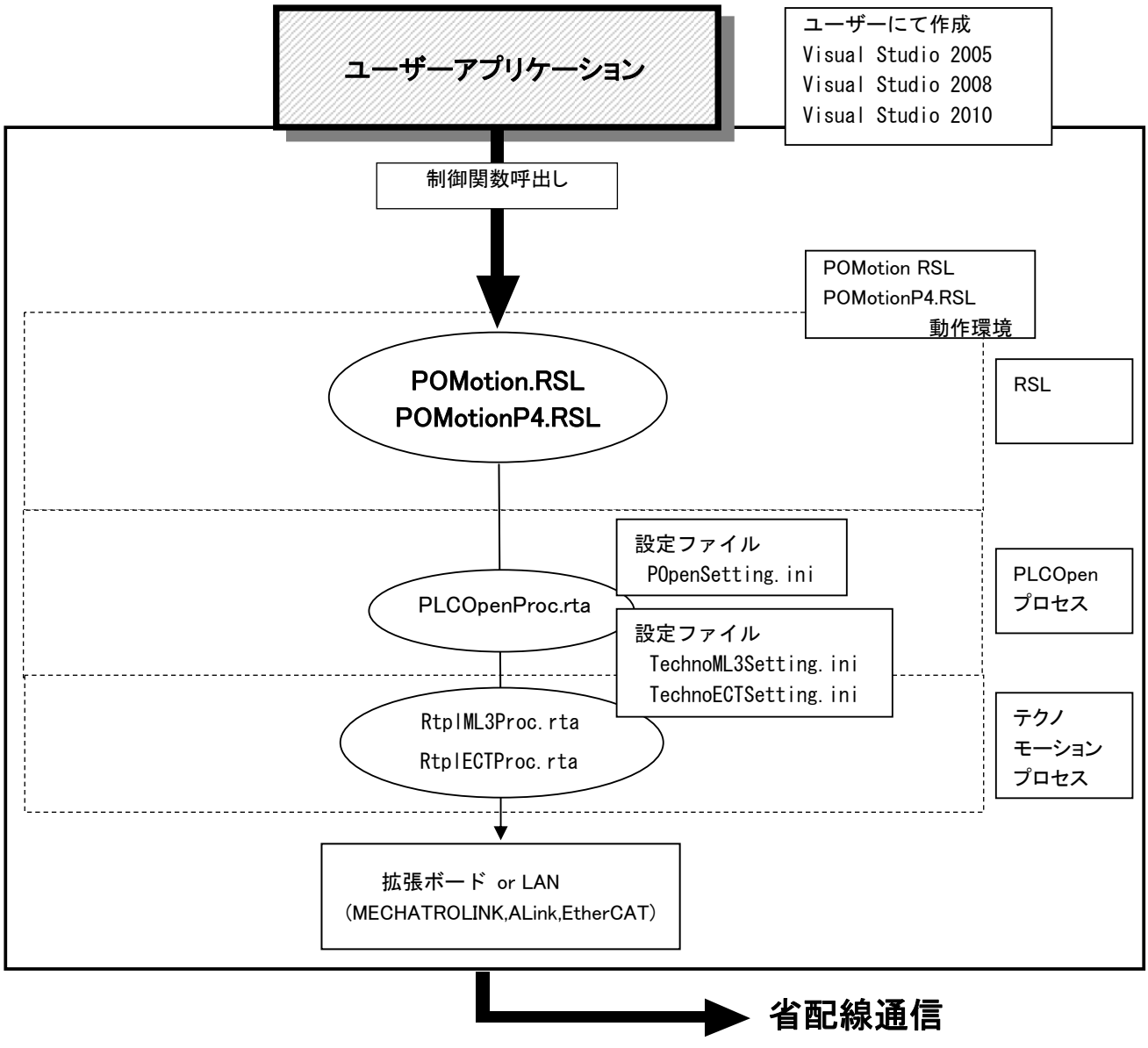


図 2-1-1. 構成図

２－２　　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 、typePOpenError.h をコピーします。同期制御を行いたい場合は、POMotionP4.cpp、POMotionP4.h、typeFbIO_p4.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 と POMotionP4.h をインクルードします。
プロジェクトへ POMotion.cpp と POMotionP4.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”);
LoadPOMtnP4Rsl(“POMotionP4.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();
UnLoadPOMtnP4Rsl();
```

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 と POMotionP4.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"

#include "stdafx.h"          ＜－－追加行
#include "POMotionP4.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言語の開発環境上で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 関数はライブラリ初期化･終了用、管理 API(サーボ ON/OFF･位置読込など軸動作なし)、動作 API の３種類があります。

３－１－１　PLCopen 仕様 Part1、Part2、Part5 関数一覧

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､ライブラリ初期化･終了用API関数の一覧を表3-1-1-1に示します。詳細は『3-4-1　PLCopen 仕様　初期化・終了処理 API 関数』を参照してください。

表 3-1-1-1. 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 3-1-1-2 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 3-1-1-2. 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
現在トルク読み出し	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
色が変わっているファンクションブロックは、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 3-1-1-3 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 3-1-1-3. 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	有り
主軸からの GEAR 同期解除	MC_GearOut	△	多軸	有り
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
色が変わっているファンクションブロックは、AI-Motion で追加されているものです。

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 3-1-1-4 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 3-1-1-4. 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

３－１－２　PLCopen Part4 仕様 関数一覧

５）PLCopen Part4 仕様 MC 初期化・終了処理 API 関数

PLCopen 仕様で定義されている Part4 MC API 関数の内、原点復帰用の API 関数の一覧を表 3-1-2-1 に示します。詳細は『3-4-5　PLCopen Part4 仕様 MC　初期化・終了処理 API 関数』を参照してください。

表 3-1-2-1. PLCopen Part4 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_WaitForP4MotionRecv 関数呼び出しの必要性
ライブラリ内リソース生成	PO_P4Create	○	無し
ライブラリ内リソース破棄	PO_P4Destroy	○	無し
マスタプロセス処理の許可	PO_P4Open	○	無し
マスタプロセス処理の禁止	PO_P4Close	○	無し
マスタプロセス処理初期化命令	PO_P4ClearMstProc	○	無し
API 応答待ち処理	PO_WaitForP4MotionRecv	○	無し

○：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

管理・動作 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_WaitForP4MotionRecv()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。

６）PLCopen Part4 仕様 MC 管理 API 関数

PLCopen Part4 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 3-1-2-2 に示します。詳細は『3-4-6　PLCopen Part4 仕様　管理 API 関数』を参照してください。

表 3-1-2-2. PLCopen Part4 仕様 MC 管理 API 関数一覧

機能概略	API 関数名	サポート	PO_WaitForP4MotionRecv 関数呼び出しの必要性
軸追加	MC_AddAxisToGroup	○	有り
軸削除	MC_RemoveAxisFromGroup	○	有り
軸グループ解除	MC_UngroupAllAxes	○	有り
軸グループ設定読出	MC_GroupReadConfiguration	○	有り
軸グループ有効	MC_GroupEnable	○	有り
軸グループ無効	MC_GroupDisable	○	有り
運動学的変換設定	MC_SetKinTransform	×	-
直交座標変換設定	MC_SetCartesianTransform	×	-
座標変換設定	MC_SetCoordinateTransform	×	-
運動学的変換読出	MC_ReadKinTransform	×	-
直交座標変換読出	MC_ReadCartesianTransform	×	-
座標変換読出	MC_ReadCoordinateTransform	×	-
軸グループ現在位置変更	MC_GroupSetPosition	○	有り
軸グループ現在位置読出	MC_GroupReadActualPosition	○	無し
軸グループ現在速度読出	MC_GroupReadActualVelocity	○	無し
軸グループ現在加速度読出	MC_GroupReadActualAcceleration	×	-
軸グループステータス読出	MC_GroupReadStatus	○	無し
軸グループエラー読出	MC_GroupReadError	○	無し
軸グループエラーリセット	MC_GroupReset	○	有り
経路選択	MC_PathSelect	×	-
軸グループオーバーライド値設定	MC_GroupSetOverride	○	有り
動的座標変換設定	MC_SetDynCoordTransform	×	-

○：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

７）PLCopen Part4 仕様 MC 動作 API 関数

PLCopen Part4 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 3-1-2-3 に示します。詳細は『3-4-7　PLCopen Part4 仕様　動作 API 関数』を参照してください。

表 3-1-2-3. PLCopen 仕様 MC 動作 API 関数一覧

機能概略	API 関数名	サポート	PO_WaitForP4MotionRecv 関数呼び出しの必要性
軸グループ原点復帰	MC_GroupHome	○	有り
軸グループ強制停止	MC_GroupStop	○	有り
軸グループ停止	MC_GroupHalt	○	有り
軸グループ一時停止	MC_GroupInterrupt	○	有り
軸グループ一時停止解除	MC_GroupContinue	○	有り
絶対値直線補間	MC_MoveLinearAbsolute	○	有り
相対値直線補間	MC_MoveLinearRelative	○	有り
絶対値円弧補間	MC_MoveCircularAbsolute	○	有り
相対値円弧補間	MC_MoveCircularRelative	○	有り
軸グループ絶対値位置決め	MC_MoveDirectAbsolute	×	-
軸グループ相対値位置決め	MC_MoveDirectRelative	×	-
指定経路移動	MC_MovePath	×	-
グループへの単軸同期動作	MC_SyncAxisToGroup	×	-
単軸へのグループ同期動作	MC_SyncGroupToAxis	×	-
コンベヤ追従動作	MC_TrackConveyorBelt	×	-
ロータリテーブル追従動作	MC_TrackRotaryTable	×	-

○：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。

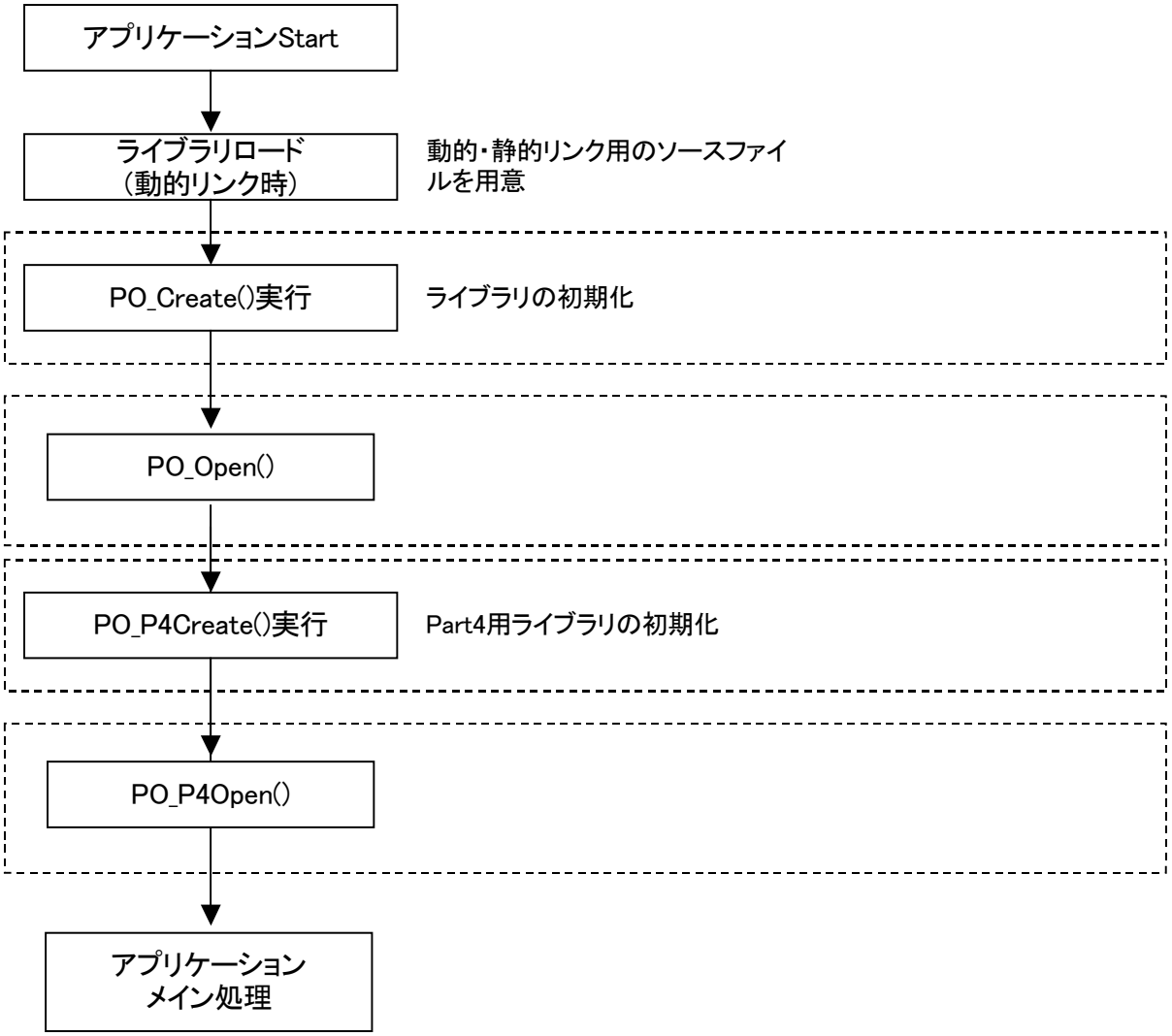


図 3-2-1-1. アプリケーション開始フローチャート

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。

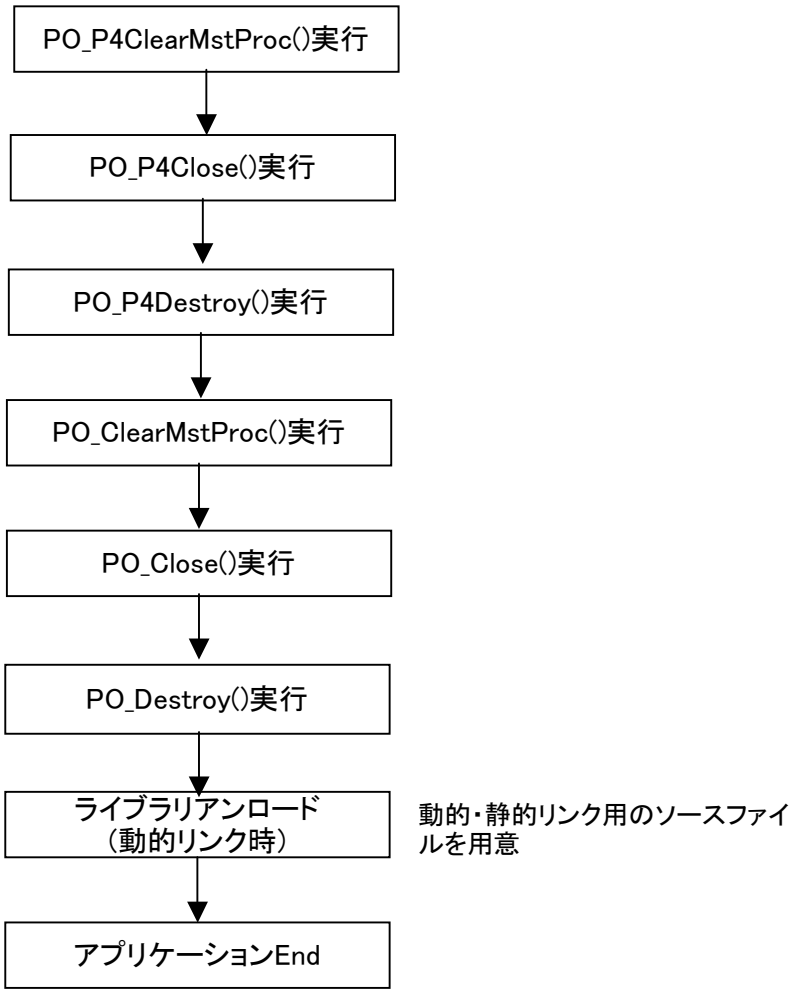


図 3-2-2-1. アプリケーション終了フローチャート

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。

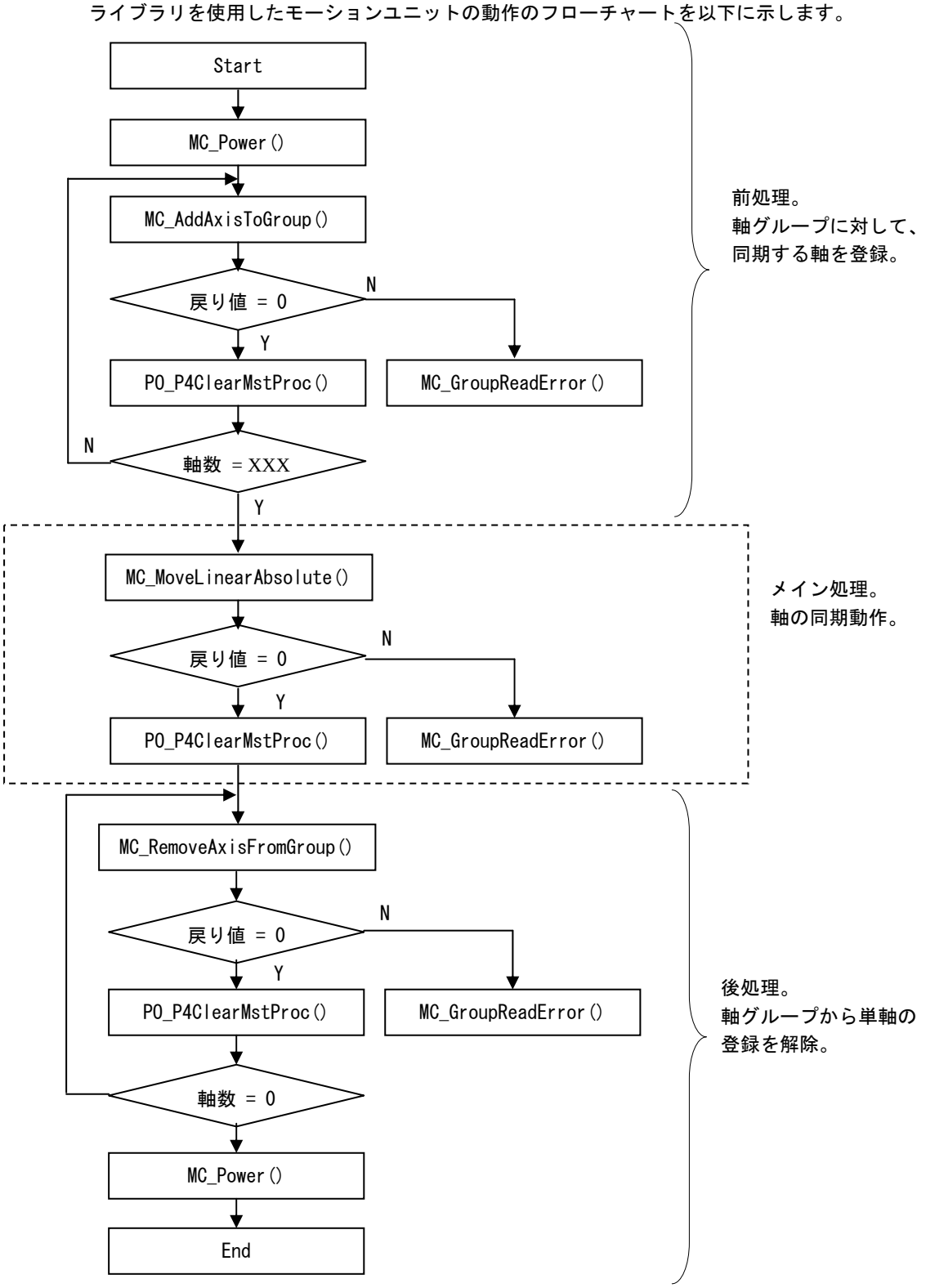


図 3-2-3-1. 動作処理フローチャート

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。
サンプルソースは AI-Motion 開発環境 DVD に格納されています。

３－３－１　Part1,Part2,Part5 のサンプルソース

```
void LibCompileTest(void);
int LibFuncLoadCheck(void);

void LibCompileTestPowerON(void);
void LibCompileTestPowerOFF(void);
void LibCompileTestReset(int slv);

void LibCompileTestMotion(int slv);
void LibCompileTestAdmini(int slv);
void LibCompileTestHoming(int slv);
void LibCompileTestMotionWait(int slv);



/*-----------------------------------------------------------------------------
   LibCompileTest()
-----------------------------------------------------------------------------*/
void LibCompileTest(void)
{
    int   ret;
    int   i;

    { // LoadPOMtnRsl
        DP(("---------------LoadPOMtnRsl start---------------¥n"));

        ret = LoadPOMtnRsl(POMTN_RSL_NAME);
        DP(("---LoadPOMtnRsl ret=0x%08x¥n", ret));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------LoadPOMtnRsl end-----------------¥n"));
    }

    { // LibFuncLoadCheck
        DP(("---------------LibFuncLoadCheck start---------------¥n"));

        ret = LibFuncLoadCheck();
        DP(("---LibFuncLoadCheck ret=0x%08x¥n", ret));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------LibFuncLoadCheck end-----------------¥n"));
    }
```

```
    { // PO_Create
        DP(("---------------PO_Create start---------------¥n"));

        ret = PO_Create();
        DP(("---PO_Create ret=0x%08x¥n", ret));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------PO_Create end-----------------¥n"));
    }

    { // PO_Open
        DP(("---------------PO_Open start---------------¥n"));

        ret = PO_Open();
        DP(("---PO_Open ret=0x%08x¥n", ret));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------PO_Open end-----------------¥n"));
    }

    /*****************************PLCopen*********************************/
    LibCompileTestPowerON();

    for(i = 1; i <= CTRL_AXIS_MAX; i++){
//      LibCompileTestReset(i);
        LibCompileTestMotion(i);
//      LibCompileTestAdmini(i);
//      LibCompileTestHoming(i);
//      LibCompileTestMotionWait(i);
    }

    LibCompileTestPowerOFF();
    /*********************************************************************/

    { // PO_ClearMstProc
        DP(("---------------PO_ClearMstProc start---------------¥n"));

        ret = PO_ClearMstProc();
        DP(("---PO_ClearMstProc ret=0x%08x¥n", ret));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------PO_ClearMstProc end-----------------¥n"));
    }

    { // PO_Close
        DP(("---------------PO_Close start---------------¥n"));

        ret = PO_Close();
        DP(("---PO_Close ret=0x%08x¥n", ret));
        RtSleep(RTSLEEP_TIME);
```

```
        DP(("---------------PO_Close end-----------------¥n"));
    }

    { // PO_Destroy
        DP(("---------------PO_Destroy start---------------¥n"));

        ret = PO_Destroy();
        DP(("---PO_Destroy ret=0x%08x¥n", ret));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------PO_Destroy end-----------------¥n"));
    }

    { // UnLoadPOMtnRsl
        DP(("---------------UnLoadPOMtnRsl start---------------¥n"));

        UnLoadPOMtnRsl();
        RtSleep(RTSLEEP_TIME);

        DP(("---------------UnLoadPOMtnRsl end-----------------¥n"));
    }
}
/*-----------------------------------------------------------------------------
    LibFuncLoadCheck
-----------------------------------------------------------------------------*/
int LibFuncLoadCheck(void)
{
    int ret = 0;

    if ( PO_Create                  == NULL ){ret = 1; DP(("PO_Create==NULL¥n"));
          }
    if ( PO_Destroy                     == NULL ){ret = 1; DP(("PO_Destroy==NULL¥n"));
             }
    if ( PO_Open                    == NULL ){ret = 1; DP(("PO_Open==NULL¥n"));
             }
    if ( PO_Close                   == NULL ){ret = 1; DP(("PO_Close==NULL¥n"));
          }
    if ( PO_ClearMstProc            == NULL ){ret = 1; DP(("PO_ClearMstProc==NULL¥n"));
             }
    if ( PO_WaitForMotionRecv           == NULL ){ret = 1;
DP(("PO_WaitForMotionRecv==NULL¥n"));      }

    if ( MC_Power                   == NULL ){ret = 1; DP(("MC_Power==NULL¥n"));
          }
    if ( MC_ReadStatus                  == NULL ){ret = 1; DP(("MC_ReadStatus==NULL¥n"));
             }
    if ( MC_ReadAxisError           == NULL ){ret = 1; DP(("MC_ReadAxisError==NULL¥n"));
          }
    if ( MC_ReadParameter           == NULL ){ret = 1; DP(("MC_ReadParameter==NULL¥n"));
          }
    if ( MC_ReadBoolParameter           == NULL ){ret = 1;
```

```
DP(("MC_ReadBoolParameter==NULL¥n"));      }
    if ( MC_ReadByteParameter           == NULL ){ret = 1;
DP(("MC_ReadByteParameter==NULL¥n"));      }
    if ( MC_ReadWordParameter           == NULL ){ret = 1;
DP(("MC_ReadWordParameter==NULL¥n"));      }
    if ( MC_ReadDwordParameter          == NULL ){ret = 1;
DP(("MC_ReadDwordParameter==NULL¥n"));     }
    if ( MC_WriteParameter              == NULL ){ret = 1;
DP(("MC_WriteParameter==NULL¥n"));         }
    if ( MC_WriteBoolParameter          == NULL ){ret = 1;
DP(("MC_WriteBoolParameter==NULL¥n"));     }
    if ( MC_WriteByteParameter          == NULL ){ret = 1;
DP(("MC_WriteByteParameter==NULL¥n"));     }
    if ( MC_WriteWordParameter          == NULL ){ret = 1;
DP(("MC_WriteWordParameter==NULL¥n"));     }
    if ( MC_WriteDwordParameter         == NULL ){ret = 1;
DP(("MC_WriteDwordParameter==NULL¥n"));    }
    if ( MC_ReadActualPosition          == NULL ){ret = 1;
DP(("MC_ReadActualPosition==NULL¥n"));     }
    if ( MC_Reset                   == NULL ){ret = 1; DP(("MC_Reset==NULL¥n"));
          }
    if ( MC_CamTableSelect              == NULL ){ret = 1;
DP(("MC_CamTableSelect==NULL¥n"));         }
    if ( MC_MoveAbsolute            == NULL ){ret = 1; DP(("MC_MoveAbsolute==NULL¥n"));
             }
    if ( MC_MoveRelative            == NULL ){ret = 1; DP(("MC_MoveRelative==NULL¥n"));
             }
    if ( MC_MoveAdditive            == NULL ){ret = 1; DP(("MC_MoveAdditive==NULL¥n"));
             }
    if ( MC_MoveSuperimposed        == NULL ){ret = 1;
DP(("MC_MoveSuperimposed==NULL¥n"));          }
    if ( MC_MoveVelocity            == NULL ){ret = 1; DP(("MC_MoveVelocity==NULL¥n"));
             }
    if ( MC_Home                    == NULL ){ret = 1; DP(("MC_Home==NULL¥n"));
             }
    if ( MC_Stop                    == NULL ){ret = 1; DP(("MC_Stop==NULL¥n"));
             }
    if ( MC_PositionProfile             == NULL ){ret = 1;
DP(("MC_PositionProfile==NULL¥n"));        }
    if ( MC_VelocityProfile             == NULL ){ret = 1;
DP(("MC_VelocityProfile==NULL¥n"));        }
    if ( MC_AccelerationProfile         == NULL ){ret = 1;
DP(("MC_AccelerationProfile==NULL¥n"));    }
    if ( MC_CamIn                   == NULL ){ret = 1; DP(("MC_CamIn==NULL¥n"));
          }
    if ( MC_CamOut                  == NULL ){ret = 1; DP(("MC_CamOut==NULL¥n"));
          }
    if ( MC_GearIn                  == NULL ){ret = 1; DP(("MC_GearIn==NULL¥n"));
          }
    if ( MC_GearOut                     == NULL ){ret = 1; DP(("MC_GearOut==NULL¥n"));
             }
```

```
    if ( MC_Phasing                    == NULL ){ret = 1; DP(("MC_Phasing==NULL¥n"));
            }
    if ( MC_ReadActualVelocity         == NULL ){ret = 1;
DP(("MC_ReadActualVelocity==NULL¥n"));     }
    if ( MC_StepAbsSwitch           == NULL ){ret = 1; DP(("MC_StepAbsSwitch==NULL¥n"));
          }
    if ( MC_StepLimitSwitch            == NULL ){ret = 1;
DP(("MC_StepLimitSwitch==NULL¥n"));        }
    if ( MC_StepBlock               == NULL ){ret = 1; DP(("MC_StepBlock==NULL¥n"));
          }
    if ( MC_FinishHoming            == NULL ){ret = 1; DP(("MC_FinishHoming==NULL¥n"));
            }
    if ( MC_StepRefPulse            == NULL ){ret = 1; DP(("MC_StepRefPulse==NULL¥n"));
            }
    if ( MC_StepDirect                 == NULL ){ret = 1; DP(("MC_StepDirect==NULL¥n"));
            }
    if ( MC_StepAbsolute            == NULL ){ret = 1; DP(("MC_StepAbsolute==NULL¥n"));
            }
    if ( MC_StepRefFlyingSwitc         == NULL ){ret = 1;
DP(("MC_StepRefFlyingSwitc==NULL¥n"));     }
    if ( MC_StepRefFlyingRefPulse      == NULL ){ret = 1;
DP(("MC_StepRefFlyingRefPulse==NULL¥n"));  }
    if ( MC_AbortPassiveHoming         == NULL ){ret = 1;
DP(("MC_AbortPassiveHoming==NULL¥n"));     }

    return ret;
}
/*-----------------------------------------------------------------------------
    LibCompileTestPowerON()
-----------------------------------------------------------------------------*/
void LibCompileTestPowerON(void)
{
    int ret;
    int i;

    TFB_POWER_IN_LIB  pwr_in;
    TFB_POWER_OUT     pwr_out;

    // MC_Power
    DP(("---------------MC_Power ON start--------------¥n"));
    for(i = 1; i <= CTRL_AXIS_MAX; i++){

        memset(&pwr_in.Axis, 0, sizeof(TFB_POWER_IN_LIB));
        memset(&pwr_out.Status, 0, sizeof(TFB_POWER_OUT));

        pwr_in.Axis              = i;
        pwr_in.Enable         = DEF_ENB;
        pwr_in.Enable_Positive   = DEF_EPS;
        pwr_in.Enable_Negative   = DEF_ENN;
        pwr_in.BufferMode     = DEF_BFF;
```

```
        //MC_Power(&pwr_in, &pwr_out);
        ret = MC_Power(&pwr_in, &pwr_out);
        DP(("---MC_Power slv=%d ret=0x%08x¥n", pwr_in.Axis, ret));

        DP(("---Status=0x%02x, Busy=0x%02x,  Active=0x%02x¥n", pwr_out.Status,
pwr_out.Busy, pwr_out.Active));
        DP(("---Error=0x%02x,  Error=0x%02x, ErrorID=0x%08x¥n", pwr_out.Error,
pwr_out.Error, pwr_out.ErrorID));
    }
    DP(("---------------MC_Power ON end-----------------¥n"));
}
/*-----------------------------------------------------------------------------
    LibCompileTestPowerOFF()
-----------------------------------------------------------------------------*/
void LibCompileTestPowerOFF(void)
{
    int ret;
    int i;

    TFB_POWER_IN_LIB  pwr_in;
    TFB_POWER_OUT     pwr_out;

    // MC_Power
    DP(("---------------MC_Power OFF start--------------¥n"));
    for(i = 1; i <= CTRL_AXIS_MAX; i++){

        memset(&pwr_in.Axis, 0, sizeof(TFB_POWER_IN_LIB));
        memset(&pwr_out.Status, 0, sizeof(TFB_POWER_OUT));

        pwr_in.Axis               = i;
        pwr_in.Enable         = DEF_DIS;
        pwr_in.Enable_Positive    = DEF_EPS;
        pwr_in.Enable_Negative    = DEF_ENN;
        pwr_in.BufferMode     = DEF_BFF;

        //MC_Power(&pwr_in, &pwr_out);
        ret = MC_Power(&pwr_in, &pwr_out);
        DP(("---MC_Power slv=%d ret=0x%08x¥n", pwr_in.Axis, ret));

        DP(("---Status=0x%02x, Busy=0x%02x,  Active=0x%02x¥n", pwr_out.Status,
pwr_out.Busy, pwr_out.Active));
        DP(("---Error=0x%02x,  Error=0x%02x, ErrorID=0x%08x¥n", pwr_out.Error,
pwr_out.Error, pwr_out.ErrorID));
    }
    DP(("---------------MC_Power OFF end----------------¥n"));
}

/*-----------------------------------------------------------------------------
    LibCompileTestReset()
-----------------------------------------------------------------------------*/
void LibCompileTestReset(int slv)
```

```
{
    int   ret;
    { // MC_Reset
        DP(("---------------MC_Reset start---------------¥n"));
        TFB_RESET_IN_LIB  rst_in;
        TFB_RESET_OUT     rst_out;

        memset(&rst_in.Axis, 0, sizeof(TFB_RESET_IN_LIB));
        memset(&rst_out.Done, 0, sizeof(TFB_RESET_OUT));

        rst_in.Axis          = slv;
        rst_in.Execute       = DEF_EXEC;

        ret = MC_Reset(&rst_in, &rst_out);
        DP(("---MC_Reset slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Done=0x%02x  ,Busy=0x%02x  ,Error=0x%02x¥n", rst_out.Done,
rst_out.Busy, rst_out.Error));
        DP(("---ErrorID=0x%08x¥n", rst_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_Reset end-----------------¥n"));
    }
}
/*-----------------------------------------------------------------------------
    LibCompileTestMotion()
-----------------------------------------------------------------------------*/
void LibCompileTestMotion(int slv)
{
    int   ret;

    DP(("--------------LibCompileTestMotion start--------------¥n"));

    { // MC_MoveAbsolute
        DP(("---------------MC_MoveAbsolute start---------------¥n"));
        TFB_MVABS_IN_LIB  abs_in;
        TFB_MVABS_OUT     abs_out;

        memset(&abs_in.Axis, 0, sizeof(TFB_MVABS_IN_LIB));
        memset(&abs_out.Done, 0, sizeof(TFB_MVABS_OUT));

        abs_in.Axis          = slv;
        abs_in.Execute       = DEF_EXEC;
        abs_in.Position      = DEF_POS;
        abs_in.Velocity      = DEF_VEL;
        abs_in.Acceleration  = DEF_ACC;
        abs_in.Deceleration  = DEF_DEC;
        abs_in.Jerk          = DEF_JRK;
        abs_in.Direction  = DEF_DIR;
        abs_in.BufferMode = DEF_BFF;
```

```
        ret = MC_MoveAbsolute(&abs_in, &abs_out, MOVE_TIMEOUT);
        DP(("---MC_MoveAbsolute slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Done=0x%02x  ,Busy=0x%02x  ,Active=0x%02x¥n", abs_out.Done,
abs_out.Busy, abs_out.Active));
        DP(("---CommandAborted=0x%02x  ,Error=0x%02x¥n", abs_out.CommandAborted,
abs_out.Error));
        DP(("---ErrorID=0x%08x¥n", abs_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_MoveAbsolute end-----------------¥n"));
    }

    { // MC_MoveRelative
        DP(("---------------MC_MoveRelative start---------------¥n"));
        TFB_MVREL_IN_LIB  rel_in;
        TFB_MVREL_OUT     rel_out;

        memset(&rel_in.Axis, 0, sizeof(TFB_MVREL_IN_LIB));
        memset(&rel_out.Done, 0, sizeof(TFB_MVREL_OUT));

        rel_in.Axis          = slv;
        rel_in.Execute       = DEF_EXEC;
        rel_in.Distance      = DEF_DST;
        rel_in.Velocity      = DEF_VEL;
        rel_in.Acceleration  = DEF_ACC;
        rel_in.Deceleration  = DEF_DEC;
        rel_in.Jerk          = DEF_JRK;
        rel_in.BufferMode = DEF_BFF;

        ret = MC_MoveRelative(&rel_in, &rel_out, MOVE_TIMEOUT);
        DP(("---MC_MoveRelative slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Done=0x%02x  ,Busy=0x%02x  ,Active=0x%02x¥n", rel_out.Done,
rel_out.Busy, rel_out.Active));
        DP(("---CommandAborted=0x%02x  ,Error=0x%02x¥n", rel_out.CommandAborted,
rel_out.Error));
        DP(("---ErrorID=0x%08x¥n", rel_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_MoveRelative end-----------------¥n"));
    }

    { // MC_MoveAdditive
        DP(("---------------MC_MoveAdditive start---------------¥n"));
        TFB_MVADD_IN_LIB  add_in;
        TFB_MVADD_OUT     add_out;

        memset(&add_in.Axis, 0, sizeof(TFB_MVADD_IN_LIB));
        memset(&add_out.Done, 0, sizeof(TFB_MVADD_OUT));
```

```
        add_in.Axis          = slv;
        add_in.Execute       = DEF_EXEC;
        add_in.Distance      = DEF_DST;
        add_in.Velocity      = DEF_VEL;
        add_in.Acceleration  = DEF_ACC;
        add_in.Deceleration  = DEF_DEC;
        add_in.Jerk          = DEF_JRK;
        add_in.BufferMode = DEF_BFF;

        ret = MC_MoveAdditive(&add_in, &add_out, MOVE_TIMEOUT);
        DP(("---MC_MoveAdditive slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Done=0x%02x  ,Busy=0x%02x  ,Active=0x%02x¥n", add_out.Done,
add_out.Busy, add_out.Active));
        DP(("---CommandAborted=0x%02x  ,Error=0x%02x¥n", add_out.CommandAborted,
add_out.Error));
        DP(("---ErrorID=0x%08x¥n", add_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_MoveAdditive end-----------------¥n"));
    }
    { // MC_MoveVelocity
        DP(("---------------MC_MoveVelocity start---------------¥n"));
        TFB_MVVEL_IN_LIB  vel_in;
        TFB_MVVEL_OUT     vel_out;

        memset(&vel_in.Axis, 0, sizeof(TFB_MVVEL_IN_LIB));
        memset(&vel_out.InVelocity, 0, sizeof(TFB_MVVEL_OUT));

        vel_in.Axis          = slv;
        vel_in.Execute       = DEF_EXEC;
        vel_in.Velocity      = DEF_VEL;
        vel_in.Acceleration  = DEF_ACC;
        vel_in.Deceleration  = DEF_DEC;
        vel_in.Jerk          = DEF_JRK;
        vel_in.Direction  = DEF_DIR;
        vel_in.BufferMode = DEF_BFF;

        ret = MC_MoveVelocity(&vel_in, &vel_out, MOVE_TIMEOUT);
        DP(("---MC_MoveVelocity slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---InVelocity=0x%02x   ,Busy=0x%02x  ,Active=0x%02x¥n",
vel_out.InVelocity, vel_out.Busy, vel_out.Active));
        DP(("---CommandAborted=0x%02x  ,Error=0x%02x¥n", vel_out.CommandAborted,
vel_out.Error));
        DP(("---ErrorID=0x%08x¥n", vel_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_MoveVelocity end-----------------¥n"));
    }
    { // MC_Stop
```

```
        DP(("---------------MC_Stop start---------------¥n"));
        TFB_STOP_IN_LIB       stp_in;
        TFB_STOP_OUT      stp_out;

        memset(&stp_in.Axis, 0, sizeof(TFB_STOP_IN_LIB));
        memset(&stp_out.Done, 0, sizeof(TFB_STOP_OUT));

        stp_in.Axis           = slv;
        stp_in.Execute        = DEF_EXEC;
        stp_in.Deceleration   = DEF_DEC;
        stp_in.Jerk           = DEF_JRK;
        stp_in.BufferMode = DEF_BFF;

        ret = MC_Stop(&stp_in, &stp_out, MOVE_TIMEOUT);
        DP(("---MC_Stop slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Done=0x%02x   ,Busy=0x%02x   ,Active=0x%02x¥n", stp_out.Done,
stp_out.Busy, stp_out.Active));
        DP(("---CommandAborted=0x%02x    ,Error=0x%02x¥n", stp_out.CommandAborted,
stp_out.Error));
        DP(("---ErrorID=0x%08x¥n", stp_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_Stop end-----------------¥n"));
    }
    DP(("---------------LibCompileTestMotion end-----------------¥n"));
}
/*-------------------------------------------------------------------------
    LibCompileTestGearMotion()
-------------------------------------------------------------------------*/
void LibCompileTestGearMotion(void)
{
    int   ret;
    { // MC_GearIn
        DP(("---------------MC_GearIn start---------------¥n"));
        TFB_GEARIN_IN_LIB gri_in;
        TFB_GEARIN_OUT    gri_out;

        memset(&gri_in.Master, 0, sizeof(TFB_GEARIN_IN_LIB));
        memset(&gri_out.InGear, 0, sizeof(TFB_GEARIN_OUT));

        gri_in.Master         = MST_ID;
        gri_in.slave_id           = SLV_ID;
        gri_in.Execute            = DEF_EXEC;
        gri_in.RatioNumerator = DEF_RTN;
        gri_in.RatioDenominator   = DEF_RTD;
        gri_in.Acceleration       = DEF_ACC;
        gri_in.Deceleration       = DEF_DEC;
        gri_in.Jerk               = DEF_JRK;
        gri_in.BufferMode     = DEF_BFF;
```

```
        ret = MC_GearIn(&gri_in, &gri_out, MOVE_TIMEOUT);
        DP(("---MC_GearIn ret=0x%08x¥n", ret));

        DP(("---InGear=0x%02x,Busy=0x%02x  ,Active=0x%02x¥n", gri_out.InGear,
gri_out.Busy, gri_out.Active));
        DP(("---CommandAborted=0x%02x   ,Error=0x%02x¥n", gri_out.CommandAborted,
gri_out.Error));
        DP(("---ErrorID=0x%08x¥n", gri_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_GearIn end----------------¥n"));
    }

    { // MC_GearOut
        DP(("---------------MC_GearOut start---------------¥n"));
        TFB_GEAROUT_IN_LIB    gro_in;
        TFB_GEAROUT_OUT       gro_out;

        memset(&gro_in.slave_id, 0, sizeof(TFB_GEAROUT_IN_LIB));
        memset(&gro_out.Done, 0, sizeof(TFB_GEAROUT_OUT));

        gro_in.slave_id       = SLV_ID;
        gro_in.Execute        = DEF_EXEC;

        ret = MC_GearOut(&gro_in, &gro_out, MOVE_TIMEOUT);
        DP(("---MC_GearOut ret=0x%08x¥n", ret));

        DP(("---Done=0x%02x  ,Busy=0x%02x  ,Error=0x%02x¥n", gro_out.Done,
gro_out.Busy, gro_out.Error));
        DP(("---ErrorID=0x%08x¥n", gro_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_GearOut end----------------¥n"));
    }
}

/*-------------------------------------------------------------------------------
    LibCompileTestAdmini()
-------------------------------------------------------------------------------*/
void LibCompileTestAdmini(int slv)
{
    int   ret;
//  int hoge;         //debug

    DP(("---------------LibCompileTestAdmini start---------------¥n"));
    { // MC_ReadStatus
        DP(("---------------MC_ReadStatus start---------------¥n"));
        TFB_RDSTATUS_IN_LIB      rds_in;
        TFB_RDSTATUS_OUT     rds_out;

        memset(&rds_in.Axis, 0, sizeof(TFB_RDSTATUS_IN_LIB));
```

```
        memset(&rds_out.Valid, 0, sizeof(TFB_RDSTATUS_OUT));

        rds_in.Axis          = slv;
        rds_in.Enable     = DEF_ENB;

        ret = MC_ReadStatus(&rds_in, &rds_out);
        DP(("---MC_ReadStatus slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x  ,Busy=0x%02x   ,Error=0x%02x¥n", rds_out.Valid,
rds_out.Busy, rds_out.Error));
        DP(("---ErrorID=0x%08x   ,ErrorStop=0x%02x¥n", rds_out.ErrorID,
rds_out.ErrorStop));
        DP(("---Disabled=0x%02x  ,Stopping=0x%02x  ,Error=0x%02x¥n", rds_out.Disabled,
rds_out.Stopping, rds_out.StandStill));
        DP(("---DiscreteMotion=0x%02x   ,ContinuousMotion=0x%02x¥n",
rds_out.DiscreteMotion, rds_out.ContinuousMotion));
        DP(("---SynchronizedMotion=0x%02x   ,Homing=0x%02x,ConstantVelocity=0x%02x¥n",
rds_out.SynchronizedMotion, rds_out.Homing, rds_out.ConstantVelocity));
        DP(("---Accelerating=0x%02x  ,Decelerating=0x%02x¥n", rds_out.Accelerating,
rds_out.Decelerating));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_ReadStatus end-----------------¥n"));
    }

    { // MC_ReadAxisError
        DP(("---------------MC_ReadAxisError start---------------¥n"));
        TFB_RDERROR_IN_LIB    rde_in;
        TFB_RDERROR_OUT       rde_out;

        memset(&rde_in.Axis, 0, sizeof(TFB_RDERROR_IN_LIB));
        memset(&rde_out.Valid, 0, sizeof(TFB_RDERROR_OUT));

        rde_in.Axis          = slv;
        rde_in.Enable     = DEF_ENB;

        ret = MC_ReadAxisError(&rde_in, &rde_out);
        DP(("---MC_ReadAxisError slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x      ,Busy=0x%02x¥n", rde_out.Valid, rde_out.Busy));
        DP(("---AxisErrorID=0x%02x   ,Error=0x%02x¥n", rde_out.AxisErrorID,
rde_out.Error));
        DP(("---ErrorID=0x%08x¥n", rde_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_ReadAxisError end-----------------¥n"));
    }

    { // MC_ReadParameter
        DP(("---------------MC_ReadParameter start---------------¥n"));
        TFB_RDPARAM_IN_LIB    rdp_in;
```

```
        TFB_RDPARAM_OUT       rdp_out;

        memset(&rdp_in.Axis, 0, sizeof(TFB_RDPARAM_IN_LIB));
        memset(&rdp_out.Valid, 0, sizeof(TFB_RDPARAM_OUT));

        rdp_in.Axis              = slv;
        rdp_in.Enable         = DEF_ENB;
        rdp_in.ParameterNumber    = DEF_PRN;

        ret = MC_ReadParameter(&rdp_in, &rdp_out);
        DP(("---MC_ReadParameter slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x ,Busy=0x%02x  ,Error=0x%02x¥n", rdp_out.Valid,
rdp_out.Busy, rdp_out.Error));
        DP(("---ErrorID=0x%08x¥n", rdp_out.ErrorID));
        DP(("---Value=0x%f¥n", rdp_out.Value));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_ReadParameter end-----------------¥n"));
    }

    { // MC_ReadBoolParameter
        DP(("---------------MC_ReadBoolParameter start---------------¥n"));
        TFB_RDBOOL_IN_LIB rdb_in;
        TFB_RDBOOL_OUT    rdb_out;

        memset(&rdb_in.Axis, 0, sizeof(TFB_RDBOOL_IN_LIB));
        memset(&rdb_out.Valid, 0, sizeof(TFB_RDBOOL_OUT));

        rdb_in.Axis              = slv;
        rdb_in.Enable         = DEF_ENB;
        rdb_in.ParameterNumber    = DEF_PRN;

        ret = MC_ReadBoolParameter(&rdb_in, &rdb_out);
        DP(("---MC_ReadBoolParameter slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x ,Busy=0x%02x  ,Error=0x%02x¥n", rdb_out.Valid,
rdb_out.Busy, rdb_out.Error));
        DP(("---ErrorID=0x%08x¥n", rdb_out.ErrorID));
        DP(("---Value=0x%02x¥n", rdb_out.Value));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_ReadBoolParameter end-----------------¥n"));
    }

    { // MC_ReadByteParameter
        DP(("---------------MC_ReadByteParameter start---------------¥n"));
        TFB_RDBYTE_IN_LIB rdb_in;
        TFB_RDBYTE_OUT    rdb_out;

        memset(&rdb_in.Axis, 0, sizeof(TFB_RDBYTE_IN_LIB));
```

```
        memset(&rdb_out.Valid, 0, sizeof(TFB_RDBYTE_OUT));

        rdb_in.Axis              = slv;
        rdb_in.Enable        = DEF_ENB;
        rdb_in.ParameterNumber   = DEF_PRN;

        ret = MC_ReadByteParameter(&rdb_in, &rdb_out);
        DP(("---MC_ReadByteParameter slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x ,Busy=0x%02x  ,Error=0x%02x¥n", rdb_out.Valid,
rdb_out.Busy, rdb_out.Error));
        DP(("---ErrorID=0x%08x¥n", rdb_out.ErrorID));
        DP(("---Value=0x%02x¥n", rdb_out.Value));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_ReadByteParameter end----------------¥n"));
    }

    { // MC_ReadWordParameter
        DP(("---------------MC_ReadWordParameter start---------------¥n"));
        TFB_RDWORD_IN_LIB rdw_in;
        TFB_RDWORD_OUT    rdw_out;

        memset(&rdw_in.Axis, 0, sizeof(TFB_RDWORD_IN_LIB));
        memset(&rdw_out.Valid, 0, sizeof(TFB_RDWORD_OUT));

        rdw_in.Axis              = slv;
        rdw_in.Enable        = DEF_ENB;
        rdw_in.ParameterNumber   = DEF_PRN;

        ret = MC_ReadWordParameter(&rdw_in, &rdw_out);
        DP(("---MC_ReadWordParameter slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x ,Busy=0x%02x  ,Error=0x%02x¥n", rdw_out.Valid,
rdw_out.Busy, rdw_out.Error));
        DP(("---ErrorID=0x%08x¥n", rdw_out.ErrorID));
        DP(("---Value=0x%04x¥n", rdw_out.Value));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_ReadWordParameter end----------------¥n"));
    }

    { // MC_ReadDwordParameter
        DP(("---------------MC_ReadDwordParameter start---------------¥n"));
        TFB_RDDWORD_IN_LIB    rdd_in;
        TFB_RDDWORD_OUT       rdd_out;

        memset(&rdd_in.Axis, 0, sizeof(TFB_RDDWORD_IN_LIB));
        memset(&rdd_out.Valid, 0, sizeof(TFB_RDDWORD_OUT));

        rdd_in.Axis              = slv;
```

```
        rdd_in.Enable         = DEF_ENB;
        rdd_in.ParameterNumber    = DEF_PRN;

        ret = MC_ReadDwordParameter(&rdd_in, &rdd_out);
        DP(("---MC_ReadDwordParameter slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x ,Busy=0x%02x  ,Error=0x%02x¥n", rdd_out.Valid,
rdd_out.Busy, rdd_out.Error));
        DP(("---ErrorID=0x%08x¥n", rdd_out.ErrorID));
        DP(("---Value=0x%08x¥n", rdd_out.Value));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_ReadDwordParameter end----------------¥n"));
    }

    { // MC_WriteParameter
        DP(("---------------MC_WriteParameter start---------------¥n"));
        TFB_WRPARAM_IN_LIB    wrp_in;
        TFB_WRPARAM_OUT       wrp_out;

        memset(&wrp_in.Axis, 0, sizeof(TFB_WRPARAM_IN_LIB));
        memset(&wrp_out.Valid, 0, sizeof(TFB_WRPARAM_OUT));

        wrp_in.Axis               = slv;
        wrp_in.Execute            = DEF_EXEC;
        wrp_in.ParameterNumber    = DEF_PRN;
        wrp_in.Value          = DEF_VOL_DBL;

        ret = MC_WriteParameter(&wrp_in, &wrp_out);
        DP(("---MC_WriteParameter slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x ,Busy=0x%02x  ,Error=0x%02x¥n", wrp_out.Valid,
wrp_out.Busy, wrp_out.Error));
        DP(("---ErrorID=0x%08x¥n", wrp_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_WriteParameter end----------------¥n"));
    }

    { // MC_WriteBoolParameter
        DP(("---------------MC_WriteBoolParameter start---------------¥n"));
        TFB_WRBOOL_IN_LIB wrb_in;
        TFB_WRBOOL_OUT    wrb_out;

        memset(&wrb_in.Axis, 0, sizeof(TFB_WRBOOL_IN_LIB));
        memset(&wrb_out.Valid, 0, sizeof(TFB_WRBOOL_OUT));

        wrb_in.Axis               = slv;
        wrb_in.Execute            = DEF_EXEC;
        wrb_in.ParameterNumber    = DEF_PRN;
        wrb_in.Value          = DEF_VOL_CH;
```

```
        ret = MC_WriteBoolParameter(&wrb_in, &wrb_out);
        DP(("---MC_WriteBoolParameter slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x ,Busy=0x%02x  ,Error=0x%02x¥n", wrb_out.Valid,
wrb_out.Busy, wrb_out.Error));
        DP(("---ErrorID=0x%08x¥n", wrb_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_WriteBoolParameter end----------------¥n"));
    }

    { // MC_WriteByteParameter
        DP(("---------------MC_WriteByteParameter start---------------¥n"));
        TFB_WRBYTE_IN_LIB wrb_in;
        TFB_WRBYTE_OUT    wrb_out;

        memset(&wrb_in.Axis, 0, sizeof(TFB_WRBYTE_IN_LIB));
        memset(&wrb_out.Valid, 0, sizeof(TFB_WRBYTE_OUT));

        wrb_in.Axis              = slv;
        wrb_in.Execute           = DEF_EXEC;
        wrb_in.ParameterNumber   = DEF_PRN;
        wrb_in.Value         = DEF_VOL_CH;

        ret = MC_WriteByteParameter(&wrb_in, &wrb_out);
        DP(("---MC_WriteByteParameter slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x ,Busy=0x%02x  ,Error=0x%02x¥n", wrb_out.Valid,
wrb_out.Busy, wrb_out.Error));
        DP(("---ErrorID=0x%08x¥n", wrb_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_WriteByteParameter end----------------¥n"));
    }

    { // MC_WriteWordParameter
        DP(("---------------MC_WriteWordParameter start---------------¥n"));
        TFB_WRWORD_IN_LIB wrw_in;
        TFB_WRWORD_OUT    wrw_out;

        memset(&wrw_in.Axis, 0, sizeof(TFB_WRWORD_IN_LIB));
        memset(&wrw_out.Valid, 0, sizeof(TFB_WRWORD_OUT));

        wrw_in.Axis              = slv;
        wrw_in.Execute           = DEF_EXEC;
        wrw_in.ParameterNumber   = DEF_PRN;
        wrw_in.Value         = DEF_VOL_SH;

        ret = MC_WriteWordParameter(&wrw_in, &wrw_out);
        DP(("---MC_WriteWordParameter slv=%d ret=0x%08x¥n", slv, ret));
```

```
        DP(("---Valid=0x%02x  ,Busy=0x%02x   ,Error=0x%02x¥n", wrw_out.Valid,
wrw_out.Busy, wrw_out.Error));
        DP(("---ErrorID=0x%08x¥n", wrw_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_WriteWordParameter end-----------------¥n"));
    }

    { // MC_WriteDwordParameter
        DP(("---------------MC_WriteDwordParameter start---------------¥n"));
        TFB_WRDWORD_IN_LIB    wrd_in;
        TFB_WRDWORD_OUT       wrd_out;

        memset(&wrd_in.Axis, 0, sizeof(TFB_WRDWORD_IN_LIB));
        memset(&wrd_out.Valid, 0, sizeof(TFB_WRDWORD_OUT));

        wrd_in.Axis              = slv;
        wrd_in.Execute           = DEF_EXEC;
        wrd_in.ParameterNumber   = DEF_PRN;
        wrd_in.Value          = DEF_VOL_LNG;

        ret = MC_WriteDwordParameter(&wrd_in, &wrd_out);
        DP(("---MC_WriteDwordParameter slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x  ,Busy=0x%02x   ,Error=0x%02x¥n", wrd_out.Valid,
wrd_out.Busy, wrd_out.Error));
        DP(("---ErrorID=0x%08x¥n", wrd_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_WriteDwordParameter end-----------------¥n"));
    }

    { // MC_ReadActualPosition
        DP(("---------------MC_ReadActualPosition start---------------¥n"));
        TFB_RDPOS_IN_LIB  rap_in;
        TFB_RDPOS_OUT     rap_out;

        memset(&rap_in.Axis, 0, sizeof(TFB_RDPOS_IN_LIB));
        memset(&rap_out.Valid, 0, sizeof(TFB_RDPOS_OUT));

        rap_in.Axis           = slv;
        rap_in.Enable     = DEF_ENB;

        ret = MC_ReadActualPosition(&rap_in, &rap_out);
        DP(("---MC_ReadActualPosition slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x  ,Busy=0x%02x   ,Error=0x%02x¥n", rap_out.Valid,
rap_out.Busy, rap_out.Error));
        DP(("---Position=0x%f¥n", rap_out.Position));
        DP(("---ErrorID=0x%08x¥n", rap_out.ErrorID));
```

```
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_ReadActualPosition end-----------------¥n"));
    }

    { // MC_ReadActualVelocity
        DP(("---------------MC_ReadActualVelocity start---------------¥n"));
        TFB_RDVEL_IN_LIB  rav_in;
        TFB_RDVEL_OUT     rav_out;

        memset(&rav_in.Axis, 0, sizeof(TFB_RDVEL_IN_LIB));
        memset(&rav_out.Valid, 0, sizeof(TFB_RDVEL_OUT));

        rav_in.Axis          = slv;
        rav_in.Enable     = DEF_ENB;

        ret = MC_ReadActualVelocity(&rav_in, &rav_out);
        DP(("---MC_ReadActualVelocity slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Valid=0x%02x ,Busy=0x%02x  ,Error=0x%02x¥n", rav_out.Valid,
rav_out.Busy, rav_out.Error));
        DP(("---Velocity=0x%f¥n", rav_out.Velocity));
        DP(("---ErrorID=0x%08x¥n", rav_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_ReadActualVelocity end-----------------¥n"));
    }

    DP(("---------------LibCompileTestAdmini end-----------------¥n"));
}
/*-------------------------------------------------------------------------------
    LibCompileTestHoming()
-------------------------------------------------------------------------------*/
void LibCompileTestHoming(int slv)
{
    int   ret;

    DP(("---------------LibCompileTestHoming start---------------¥n"));

    { // MC_StepAbsSwitch
        DP(("---------------MC_StepAbsSwitch start---------------¥n"));
        TFB_STEPABSSW_IN_LIB  sas_in;
        TFB_STEPABSSW_OUT     sas_out;

        memset(&sas_in.Axis, 0, sizeof(TFB_STEPABSSW_IN_LIB));
        memset(&sas_out.Done, 0, sizeof(TFB_STEPABSSW_OUT));

        sas_in.Axis              = slv;
        sas_in.Execute           = DEF_EXEC;
        sas_in.Direction      = DEF_DIR;
        sas_in.SwitchMode     = DEF_SWM;
```

```
        sas_in.Velocity          = DEF_VEL;
        sas_in.TorqueLimit       = DEF_TRL;
        sas_in.TimeLimit     = DEF_TML;
        sas_in.DistanceLimit  = DEF_DSL;
        sas_in.BufferMode    = DEF_BFF;

        ret = MC_StepAbsSwitch(&sas_in, &sas_out);
        DP(("---MC_StepAbsSwitch slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Done=0x%02x   ,Busy=0x%02x   ,Active=0x%02x¥n", sas_out.Done,
sas_out.Busy, sas_out.Active));
        DP(("---CommandAborted=0x%02x    ,Error=0x%02x¥n", sas_out.CommandAborted,
sas_out.Error));
        DP(("---ErrorID=0x%08x¥n", sas_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_StepAbsSwitch end-----------------¥n"));
    }

    { // MC_StepLimitSwitch
        DP(("---------------MC_StepLimitSwitch start---------------¥n"));
        TFB_STEPLMTSW_IN_LIB  sls_in;
        TFB_STEPLMTSW_OUT     sls_out;

        memset(&sls_in.Axis, 0, sizeof(TFB_STEPLMTSW_IN_LIB));
        memset(&sls_out.Done, 0, sizeof(TFB_STEPLMTSW_OUT));

        sls_in.Axis              = slv;
        sls_in.Execute           = DEF_EXEC;
        sls_in.Direction     = DEF_DIR;
        sls_in.LimitSwitchMode   = DEF_LSM;
        sls_in.Velocity          = DEF_VEL;
        sls_in.TorqueLimit       = DEF_TRL;
        sls_in.TimeLimit     = DEF_TML;
        sls_in.DistanceLimit  = DEF_DSL;
        sls_in.BufferMode    = DEF_BFF;

        ret = MC_StepLimitSwitch(&sls_in, &sls_out);
        DP(("---MC_StepLimitSwitch slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Done=0x%02x   ,Busy=0x%02x   ,Active=0x%02x¥n", sls_out.Done,
sls_out.Busy, sls_out.Active));
        DP(("---CommandAborted=0x%02x    ,Error=0x%02x¥n", sls_out.CommandAborted,
sls_out.Error));
        DP(("---ErrorID=0x%08x¥n", sls_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_StepLimitSwitch end-----------------¥n"));
    }

    { // MC_FinishHoming
```

```
        DP(("---------------MC_FinishHoming start---------------¥n"));
        TFB_FINHOME_IN_LIB    fhm_in;
        TFB_FINHOME_OUT       fhm_out;

        memset(&fhm_in.Axis, 0, sizeof(TFB_FINHOME_IN_LIB));
        memset(&fhm_out.Done, 0, sizeof(TFB_FINHOME_OUT));

        fhm_in.Axis           = slv;
        fhm_in.Execute        = DEF_EXEC;
        fhm_in.BufferMode = DEF_BFF;

        ret = MC_FinishHoming(&fhm_in, &fhm_out);
        DP(("---MC_FinishHoming slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Done=0x%02x   ,Busy=0x%02x   ,Active=0x%02x¥n", fhm_out.Done,
fhm_out.Busy, fhm_out.Active));
        DP(("---CommandAborted=0x%02x    ,Error=0x%02x¥n", fhm_out.CommandAborted,
fhm_out.Error));
        DP(("---ErrorID=0x%08x¥n", fhm_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_FinishHoming end-----------------¥n"));
    }

    { // MC_StepRefPulse
        DP(("---------------MC_StepRefPulse start---------------¥n"));
        TFB_STEPREFPLS_IN_LIB srp_in;
        TFB_STEPREFPLS_OUT       srp_out;

        memset(&srp_in.Axis, 0, sizeof(TFB_STEPREFPLS_IN_LIB));
        memset(&srp_out.Done, 0, sizeof(TFB_STEPREFPLS_OUT));

        srp_in.Axis              = slv;
        srp_in.Execute           = DEF_EXEC;
        srp_in.Direction     = DEF_DIR;
        srp_in.SetPosition       = DEF_STP;
        srp_in.Velocity          = DEF_VEL;
        srp_in.TorqueLimit       = DEF_TRL;
        srp_in.TimeLimit     = DEF_TML;
        srp_in.DistanceLimit  = DEF_DSL;
        srp_in.BufferMode    = DEF_BFF;

        ret = MC_StepRefPulse(&srp_in, &srp_out);
        DP(("---MC_StepRefPulse slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Done=0x%02x   ,Busy=0x%02x   ,Active=0x%02x¥n", srp_out.Done,
srp_out.Busy, srp_out.Active));
        DP(("---CommandAborted=0x%02x    ,Error=0x%02x¥n", srp_out.CommandAborted,
srp_out.Error));
        DP(("---ErrorID=0x%08x¥n", srp_out.ErrorID));
        RtSleep(RTSLEEP_TIME);
```

```
        DP(("----------------MC_StepRefPulse end-----------------¥n"));
    }

    { // MC_StepDirect
        DP(("----------------MC_StepDirect start----------------¥n"));
        TFB_STEPDIR_IN_LIB    std_in;
        TFB_STEPDIR_OUT       std_out;

        memset(&std_in.Axis, 0, sizeof(TFB_STEPDIR_IN_LIB));
        memset(&std_out.Done, 0, sizeof(TFB_STEPDIR_OUT));

        std_in.Axis           = slv;
        std_in.Execute        = DEF_EXEC;
        std_in.SetPosition    = DEF_STP;
        std_in.BufferMode = DEF_BFF;

        ret = MC_StepDirect(&std_in, &std_out);
        DP(("---MC_StepDirect slv=%d ret=0x%08x¥n", slv, ret));

        DP(("---Done=0x%02x   ,Busy=0x%02x   ,Active=0x%02x¥n", std_out.Done,
std_out.Busy, std_out.Active));
        DP(("---CommandAborted=0x%02x    ,Error=0x%02x¥n", std_out.CommandAborted,
std_out.Error));
        DP(("---ErrorID=0x%08x¥n", std_out.ErrorID));
        RtSleep(RTSLEEP_TIME);

        DP(("----------------MC_StepDirect end-----------------¥n"));
    }

    { // MC_StepAbsolute
        DP(("----------------MC_StepAbsolute start----------------¥n"));
        TFB_STEPABS_IN_LIB    sta_in;
        TFB_STEPABS_OUT       sta_out;

        memset(&sta_in.Axis, 0, sizeof(TFB_STEPABS_IN_LIB));
        memset(&sta_out.Done, 0, sizeof(TFB_STEPABS_OUT));

        sta_in.Axis           = slv;
        sta_in.Execute        = DEF_EXEC;
        sta_in.BufferMode = DEF_BFF;

        ret = MC_StepAbsolute(&sta_in, &sta_out);
        DP(("---MC_StepAbsolute slv=%d, ret=0x%08x¥n", slv, ret));

        DP(("---Done=0x%02x   ,Busy=0x%02x   ,Active=0x%02x¥n", sta_out.Done,
sta_out.Busy, sta_out.Active));
        DP(("---CommandAborted=0x%02x    ,Error=0x%02x¥n", sta_out.CommandAborted,
sta_out.Error));
        DP(("---ErrorID=0x%08x¥n", sta_out.ErrorID));
        RtSleep(RTSLEEP_TIME);
```

```
        DP(("---------------MC_StepAbsolute end-----------------¥n"));
    }

    DP(("--------------LibCompileTestHoming end-----------------¥n"));
}

/*-----------------------------------------------------------------------------
    LibCompileTestMotionWait()
-----------------------------------------------------------------------------*/
void LibCompileTestMotionWait(int slv)
{
    int   ret;

    DP(("--------------LibCompileTestMotion Wait start--------------¥n"));

    { // MC_MoveAbsolute
        DP(("---------------MC_MoveAbsolute start---------------¥n"));
        TFB_MVABS_IN_LIB  abs_in;
        TFB_MVABS_OUT     abs_out;

        memset(&abs_in.Axis, 0, sizeof(TFB_MVABS_IN_LIB));
        memset(&abs_out.Done, 0, sizeof(TFB_MVABS_OUT));

        abs_in.Axis          = slv;
        abs_in.Execute       = DEF_EXEC;
        abs_in.Position      = DEF_POS;
        abs_in.Velocity      = DEF_VEL;
        abs_in.Acceleration  = DEF_ACC;
        abs_in.Deceleration  = DEF_DEC;
        abs_in.Jerk          = DEF_JRK;
        abs_in.Direction  = DEF_DIR;
        abs_in.BufferMode = DEF_BFF;

        ret = MC_MoveAbsolute(&abs_in, NULL, 0);
        DP(("---MC_MoveAbsolute slv=%d ret=0x%08x¥n", slv, ret));
        //Add wait
        if(ret != 0){
            ret = PO_WaitForMotionRecv(ret, &abs_out, MOVE_TIMEOUT);
            DP(("---MC_MoveAbsolute ret=0x%08x¥n", ret));

            DP(("---Done=0x%02x  ,Busy=0x%02x  ,Active=0x%02x¥n", abs_out.Done,
abs_out.Busy, abs_out.Active));
            DP(("---CommandAborted=0x%02x   ,Error=0x%02x¥n", abs_out.CommandAborted,
abs_out.Error));
            DP(("---ErrorID=0x%08x¥n", abs_out.ErrorID));
        }
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_MoveAbsolute end-----------------¥n"));
    }
```

```
    { // MC_MoveRelative
        DP(("---------------MC_MoveRelative start---------------¥n"));
        TFB_MVREL_IN_LIB  rel_in;
        TFB_MVREL_OUT     rel_out;

        memset(&rel_in.Axis, 0, sizeof(TFB_MVREL_IN_LIB));
        memset(&rel_out.Done, 0, sizeof(TFB_MVREL_OUT));

        rel_in.Axis          = slv;
        rel_in.Execute       = DEF_EXEC;
        rel_in.Distance      = DEF_DST;
        rel_in.Velocity      = DEF_VEL;
        rel_in.Acceleration  = DEF_ACC;
        rel_in.Deceleration  = DEF_DEC;
        rel_in.Jerk          = DEF_JRK;
        rel_in.BufferMode = DEF_BFF;

        ret = MC_MoveRelative(&rel_in, NULL, 0);
        DP(("---MC_MoveRelative slv=%d ret=0x%08x¥n", slv, ret));
        //Add wait
        if(ret != 0){
            ret = PO_WaitForMotionRecv(ret, &rel_out, MOVE_TIMEOUT);
            DP(("---MC_MoveRelative ret=0x%08x¥n", ret));

            DP(("---Done=0x%02x   ,Busy=0x%02x   ,Active=0x%02x¥n", rel_out.Done,
rel_out.Busy, rel_out.Active));
            DP(("---CommandAborted=0x%02x    ,Error=0x%02x¥n", rel_out.CommandAborted,
rel_out.Error));
            DP(("---ErrorID=0x%08x¥n", rel_out.ErrorID));
        }
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_MoveRelative end-----------------¥n"));
    }

    { // MC_MoveAdditive
        DP(("---------------MC_MoveAdditive start---------------¥n"));
        TFB_MVADD_IN_LIB  add_in;
        TFB_MVADD_OUT     add_out;

        memset(&add_in.Axis, 0, sizeof(TFB_MVADD_IN_LIB));
        memset(&add_out.Done, 0, sizeof(TFB_MVADD_OUT));

        add_in.Axis          = slv;
        add_in.Execute       = DEF_EXEC;
        add_in.Distance      = DEF_DST;
        add_in.Velocity      = DEF_VEL;
        add_in.Acceleration  = DEF_ACC;
        add_in.Deceleration  = DEF_DEC;
        add_in.Jerk          = DEF_JRK;
```

```
        add_in.BufferMode = DEF_BFF;

        ret = MC_MoveAdditive(&add_in, NULL, 0);
        DP(("---MC_MoveAdditive slv=%d ret=0x%08x¥n", slv, ret));
        //Add wait
        if(ret != 0){
            ret = PO_WaitForMotionRecv(ret, &add_out, MOVE_TIMEOUT);
            DP(("---MC_MoveAdditive ret=0x%08x¥n", ret));

            DP(("---Done=0x%02x   ,Busy=0x%02x   ,Active=0x%02x¥n", add_out.Done,
add_out.Busy, add_out.Active));
            DP(("---CommandAborted=0x%02x    ,Error=0x%02x¥n", add_out.CommandAborted,
add_out.Error));
            DP(("---ErrorID=0x%08x¥n", add_out.ErrorID));
        }
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_MoveAdditive end-----------------¥n"));
    }

    { // MC_MoveVelocity
        DP(("---------------MC_MoveVelocity start---------------¥n"));
        TFB_MVVEL_IN_LIB  vel_in;
        TFB_MVVEL_OUT     vel_out;

        memset(&vel_in.Axis, 0, sizeof(TFB_MVVEL_IN_LIB));
        memset(&vel_out.InVelocity, 0, sizeof(TFB_MVVEL_OUT));

        vel_in.Axis          = slv;
        vel_in.Execute       = DEF_EXEC;
        vel_in.Velocity      = DEF_VEL;
        vel_in.Acceleration  = DEF_ACC;
        vel_in.Deceleration  = DEF_DEC;
        vel_in.Jerk          = DEF_JRK;
        vel_in.Direction  = DEF_DIR;
        vel_in.BufferMode = DEF_BFF;

        ret = MC_MoveVelocity(&vel_in, NULL, 0);
        DP(("---MC_MoveVelocity slv=%d ret=0x%08x¥n", slv, ret));
        //Add wait
        if(ret != 0){
            ret = PO_WaitForMotionRecv(ret, &vel_out, MOVE_TIMEOUT);
            DP(("---MC_MoveVelocity ret=0x%08x¥n", ret));

            DP(("---InVelocity=0x%02x    ,Busy=0x%02x   ,Active=0x%02x¥n",
vel_out.InVelocity, vel_out.Busy, vel_out.Active));
            DP(("---CommandAborted=0x%02x    ,Error=0x%02x¥n", vel_out.CommandAborted,
vel_out.Error));
            DP(("---ErrorID=0x%08x¥n", vel_out.ErrorID));
        }
        RtSleep(RTSLEEP_TIME);
```

```
        DP(("----------------MC_MoveVelocity end-----------------¥n"));
    }

    { // MC_Stop
        DP(("----------------MC_Stop ON start---------------¥n"));
        TFB_STOP_IN_LIB      stp_in;
        TFB_STOP_OUT      stp_out;

        memset(&stp_in.Axis, 0, sizeof(TFB_STOP_IN_LIB));
        memset(&stp_out.Done, 0, sizeof(TFB_STOP_OUT));

        stp_in.Axis          = slv;
        stp_in.Execute       = DEF_EXEC;
        stp_in.Deceleration  = DEF_DEC;
        stp_in.Jerk          = DEF_JRK;
        stp_in.BufferMode = DEF_BFF;

        ret = MC_Stop(&stp_in, NULL, 0);
        DP(("---MC_Stop slv=%d ret=0x%08x¥n", slv, ret));
        //Add wait
        if(ret != 0){
            ret = PO_WaitForMotionRecv(ret, &stp_out, MOVE_TIMEOUT);
            DP(("---MC_Stop ret=0x%08x¥n", ret));

            DP(("---Done=0x%02x   ,Busy=0x%02x   ,Active=0x%02x¥n", stp_out.Done,
stp_out.Busy, stp_out.Active));
            DP(("---CommandAborted=0x%02x    ,Error=0x%02x¥n", stp_out.CommandAborted,
stp_out.Error));
            DP(("---ErrorID=0x%08x¥n", stp_out.ErrorID));
        }
        RtSleep(RTSLEEP_TIME);

        DP(("----------------MC_Stop ON end-----------------¥n"));
    }

    { // MC_Stop
        DP(("----------------MC_Stop OFF start---------------¥n"));
        TFB_STOP_IN_LIB      stp_in;
        TFB_STOP_OUT      stp_out;

        memset(&stp_in.Axis, 0, sizeof(TFB_STOP_IN_LIB));
        memset(&stp_out.Done, 0, sizeof(TFB_STOP_OUT));

        stp_in.Axis          = slv;
        stp_in.Execute       = 0;
        stp_in.Deceleration  = DEF_DEC;
        stp_in.Jerk          = DEF_JRK;
        stp_in.BufferMode = DEF_BFF;

        ret = MC_Stop(&stp_in, NULL, 0);
```

```
        DP(("---MC_Stop slv=%d ret=0x%08x¥n", slv, ret));
        //Add wait
        if(ret != 0){
            ret = PO_WaitForMotionRecv(ret, &stp_out, MOVE_TIMEOUT);
            DP(("---MC_Stop ret=0x%08x¥n", ret));

            DP(("---Done=0x%02x   ,Busy=0x%02x   ,Active=0x%02x¥n", stp_out.Done,
stp_out.Busy, stp_out.Active));
            DP(("---CommandAborted=0x%02x    ,Error=0x%02x¥n", stp_out.CommandAborted,
stp_out.Error));
            DP(("---ErrorID=0x%08x¥n", stp_out.ErrorID));
        }
        RtSleep(RTSLEEP_TIME);

        DP(("---------------MC_Stop OFF end-----------------¥n"));
    }

    DP(("---------------LibCompileTestMotion Wait end-----------------¥n"));
}
```

３－３－２　Part4 サンプルソース

```
void LibCompileTest(void);

int LibFuncLoadCheck(void);
int LibFuncLoadCheckP4(void);

void LibCompileTestPowerON(void);
void LibCompileTestPowerOFF(void);

int LibCompileTestGroupAdd(void);
int LibCompileTestGroupEnable(void);

void LibCompileTestLineAbsoluteMove(void);

/*-------------------------------------------------------------------------
    LibCompileTest()
-------------------------------------------------------------------------*/
void LibCompileTest(void)
{
    int ret = 0;

    { // LoadPOMtnRsl
        DP(("---------------LoadPOMtnRsl start---------------¥n"));

        ret = LoadPOMtnRsl(POMTN_RSL_NAME);
        DP(("---LoadPOMtnRsl ret=0x%08x¥n", ret));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------LoadPOMtnRsl end-----------------¥n"));
```

```
    }

    { // LibFuncLoadCheck
        DP(("---------------LibFuncLoadCheck start---------------¥n"));

        ret = LibFuncLoadCheck();
        DP(("---LibFuncLoadCheck ret=0x%08x¥n", ret));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------LibFuncLoadCheck end-----------------¥n"));
    }

    { // PO_Create
        DP(("---------------PO_Create start---------------¥n"));

        ret = PO_Create();
        DP(("---PO_Create ret=0x%08x¥n", ret));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------PO_Create end-----------------¥n"));
    }

    { // PO_Open
        DP(("---------------PO_Open start---------------¥n"));

        ret = PO_Open();
        DP(("---PO_Open ret=0x%08x¥n", ret));
        RtSleep(RTSLEEP_TIME);

        DP(("---------------PO_Open end-----------------¥n"));
    }

    { // LoadPOP4Rsl
        printf("---------------LoadPOP4Rsl start---------------¥n");

        ret = LoadPOMtnP4Rsl(POMTNP4_RSL_NAME);
        printf("---LoadPOP4Rsl ret=0x%08x¥n", ret);
        RtSleep(RTSLEEP_TIME);

        printf("---------------LoadPOP4Rsl end-----------------¥n");
    }
    if ( ret!=0 ) return ;

    { // LibFuncLoadCheck
        printf("---------------LibFuncLoadCheck start---------------¥n");

        ret = LibFuncLoadCheckP4();
        RtSleep(RTSLEEP_TIME);

        printf("---------------LibFuncLoadCheck end-----------------¥n");
    }
```

```
    if ( ret!=0 ) return ;

    { // PO_P4Create
        printf("---------------PO_P4Create start---------------¥n");

        ret = PO_P4Create();
        printf("---PO_P4Create ret=0x%08x¥n", ret);
        RtSleep(RTSLEEP_TIME);

        printf("---------------PO_P4Create end-----------------¥n");
    }

    { // PO_P4Open
        printf("---------------PO_P4Open start---------------¥n");

        ret = PO_P4Open();
        printf("---PO_P4Open ret=0x%08x¥n", ret);
        RtSleep(RTSLEEP_TIME);

        printf("---------------PO_P4Open end-----------------¥n");
    }

    /*****************************PLCopen*******************************/
    LibCompileTestPowerON();
    RtSleep(RTSLEEP_TIME);

    if(LibCompileTestGroupAdd() == 0){
        RtSleep(RTSLEEP_TIME);
        if(LibCompileTestGroupEnable() == 0){
            while(1){
                LibCompileTestLineAbsoluteMove();
            }
        }
    }

    LibCompileTestPowerOFF();
    RtSleep(RTSLEEP_TIME);
    /*********************************************************************/

    { // PO_P4ClearMstProc
        printf("---------------PO_P4ClearMstProc start---------------¥n");

        ret = PO_P4ClearMstProc();
        printf("---PO_P4ClearMstProc ret=0x%08x¥n", ret);
        RtSleep(RTSLEEP_TIME);

        printf("---------------PO_P4ClearMstProc end-----------------¥n");
    }

    { // PO_P4Close
        printf("---------------PO_P4Close start---------------¥n");
```

```
    ret = PO_P4Close();
    printf("---PO_P4Close ret=0x%08x¥n", ret);
    RtSleep(RTSLEEP_TIME);

    printf("---------------PO_P4Close end-----------------¥n");
}

{ // PO_P4Destroy
    printf("---------------PO_P4Destroy start---------------¥n");

    ret = PO_P4Destroy();
    printf("---PO_P4Destroy ret=0x%08x¥n", ret);
    RtSleep(RTSLEEP_TIME);

    printf("---------------PO_P4Destroy end-----------------¥n");
}

{ // UnloadPOP4Rsl
    printf("---------------UnloadPOP4Rsl start---------------¥n");

    UnLoadPOMtnP4Rsl();
    RtSleep(RTSLEEP_TIME);

    printf("---------------UnloadPOP4Rsl end-----------------¥n");
}
{ // PO_ClearMstProc
    DP(("---------------PO_ClearMstProc start---------------¥n"));

    ret = PO_ClearMstProc();
    DP(("---PO_ClearMstProc ret=0x%08x¥n", ret));
    RtSleep(RTSLEEP_TIME);

    DP(("---------------PO_ClearMstProc end-----------------¥n"));
}

{ // PO_Close
    DP(("---------------PO_Close start---------------¥n"));

    ret = PO_Close();
    DP(("---PO_Close ret=0x%08x¥n", ret));
    RtSleep(RTSLEEP_TIME);

    DP(("---------------PO_Close end-----------------¥n"));
}

{ // PO_Destroy
    DP(("---------------PO_Destroy start---------------¥n"));

    ret = PO_Destroy();
    DP(("---PO_Destroy ret=0x%08x¥n", ret));
```

```
        RtSleep(RTSLEEP_TIME);

        DP(("---------------PO_Destroy end-----------------¥n"));
    }

    { // UnLoadPOMtnRsl
        DP(("---------------UnLoadPOMtnRsl start---------------¥n"));

        UnLoadPOMtnRsl();
        RtSleep(RTSLEEP_TIME);

        DP(("---------------UnLoadPOMtnRsl end-----------------¥n"));
    }
}

/*-----------------------------------------------------------------------------
    LibFuncLoadCheck
-----------------------------------------------------------------------------*/
int LibFuncLoadCheck(void)
{
    int ret = 0;

    if ( PO_Create                    == NULL ){ret = 1; DP(("PO_Create==NULL¥n"));
        }
    if ( PO_Destroy                   == NULL ){ret = 1; DP(("PO_Destroy==NULL¥n"));
        }
    if ( PO_Open                      == NULL ){ret = 1; DP(("PO_Open==NULL¥n"));
        }
    if ( PO_Close                     == NULL ){ret = 1; DP(("PO_Close==NULL¥n"));
        }
    if ( PO_ClearMstProc              == NULL ){ret = 1; DP(("PO_ClearMstProc==NULL¥n"));
        }
    if ( PO_WaitForMotionRecv         == NULL ){ret = 1; DP(("PO_WaitForMotionRecv==NULL¥n"));
    }

    if ( MC_Power                     == NULL ){ret = 1; DP(("MC_Power==NULL¥n"));
        }
    if ( MC_ReadStatus                == NULL ){ret = 1; DP(("MC_ReadStatus==NULL¥n"));
        }
    if ( MC_ReadAxisError             == NULL ){ret = 1; DP(("MC_ReadAxisError==NULL¥n"));
        }
    if ( MC_ReadParameter             == NULL ){ret = 1; DP(("MC_ReadParameter==NULL¥n"));
        }
    if ( MC_ReadBoolParameter         == NULL ){ret = 1; DP(("MC_ReadBoolParameter==NULL¥n"));
    }
    if ( MC_ReadByteParameter         == NULL ){ret = 1; DP(("MC_ReadByteParameter==NULL¥n"));
    }
    if ( MC_ReadWordParameter         == NULL ){ret = 1; DP(("MC_ReadWordParameter==NULL¥n"));
    }
    if ( MC_ReadDwordParameter        == NULL ){ret = 1; DP(("MC_ReadDwordParameter==NULL¥n"));
    }
```

```
    if ( MC_WriteParameter              == NULL ){ret = 1; DP(("MC_WriteParameter==NULL¥n"));
       }
    if ( MC_WriteBoolParameter          == NULL ){ret = 1; DP(("MC_WriteBoolParameter==NULL¥n"));
    }
    if ( MC_WriteByteParameter          == NULL ){ret = 1; DP(("MC_WriteByteParameter==NULL¥n"));
    }
    if ( MC_WriteWordParameter          == NULL ){ret = 1; DP(("MC_WriteWordParameter==NULL¥n"));
    }
    if ( MC_WriteDwordParameter         == NULL ){ret = 1; DP(("MC_WriteDwordParameter==NULL¥n"));
       }
    if ( MC_ReadActualPosition          == NULL ){ret = 1; DP(("MC_ReadActualPosition==NULL¥n"));
    }
    if ( MC_Reset                       == NULL ){ret = 1; DP(("MC_Reset==NULL¥n"));
       }
    if ( MC_CamTableSelect              == NULL ){ret = 1; DP(("MC_CamTableSelect==NULL¥n"));
       }
    if ( MC_MoveAbsolute                == NULL ){ret = 1; DP(("MC_MoveAbsolute==NULL¥n"));
       }
    if ( MC_MoveRelative                == NULL ){ret = 1; DP(("MC_MoveRelative==NULL¥n"));
       }
    if ( MC_MoveAdditive                == NULL ){ret = 1; DP(("MC_MoveAdditive==NULL¥n"));
       }
    if ( MC_MoveSuperimposed            == NULL ){ret = 1; DP(("MC_MoveSuperimposed==NULL¥n"));
       }
    if ( MC_MoveVelocity                == NULL ){ret = 1; DP(("MC_MoveVelocity==NULL¥n"));
       }
    if ( MC_Home                        == NULL ){ret = 1; DP(("MC_Home==NULL¥n"));
       }
    if ( MC_Stop                        == NULL ){ret = 1; DP(("MC_Stop==NULL¥n"));
       }
    if ( MC_PositionProfile             == NULL ){ret = 1; DP(("MC_PositionProfile==NULL¥n"));
       }
    if ( MC_VelocityProfile             == NULL ){ret = 1; DP(("MC_VelocityProfile==NULL¥n"));
       }
    if ( MC_AccelerationProfile         == NULL ){ret = 1; DP(("MC_AccelerationProfile==NULL¥n"));
       }
    if ( MC_CamIn                       == NULL ){ret = 1; DP(("MC_CamIn==NULL¥n"));
       }
    if ( MC_CamOut                      == NULL ){ret = 1; DP(("MC_CamOut==NULL¥n"));
       }
    if ( MC_GearIn                      == NULL ){ret = 1; DP(("MC_GearIn==NULL¥n"));
       }
    if ( MC_GearOut                     == NULL ){ret = 1; DP(("MC_GearOut==NULL¥n"));
       }
    if ( MC_Phasing                     == NULL ){ret = 1; DP(("MC_Phasing==NULL¥n"));
       }
    if ( MC_ReadActualVelocity          == NULL ){ret = 1; DP(("MC_ReadActualVelocity==NULL¥n"));
    }
    if ( MC_StepAbsSwitch               == NULL ){ret = 1; DP(("MC_StepAbsSwitch==NULL¥n"));
       }
    if ( MC_StepLimitSwitch             == NULL ){ret = 1; DP(("MC_StepLimitSwitch==NULL¥n"));
```

```
        }
    if ( MC_StepBlock                   == NULL ){ret = 1; DP(("MC_StepBlock==NULL¥n"));
        }
    if ( MC_FinishHoming                == NULL ){ret = 1; DP(("MC_FinishHoming==NULL¥n"));
        }
    if ( MC_StepRefPulse                == NULL ){ret = 1; DP(("MC_StepRefPulse==NULL¥n"));
        }
    if ( MC_StepDirect                  == NULL ){ret = 1; DP(("MC_StepDirect==NULL¥n"));
        }
    if ( MC_StepAbsolute                == NULL ){ret = 1; DP(("MC_StepAbsolute==NULL¥n"));
        }
    if ( MC_StepRefFlyingSwitc          == NULL ){ret = 1; DP(("MC_StepRefFlyingSwitc==NULL¥n"));
    }
    if ( MC_StepRefFlyingRefPulse       == NULL ){ret = 1; DP(("MC_StepRefFlyingRefPulse==NULL¥n"));
    }
    if ( MC_AbortPassiveHoming          == NULL ){ret = 1; DP(("MC_AbortPassiveHoming==NULL¥n"));
    }

    return ret;
}

/*-----------------------------------------------------------------------------
    LibFuncLoadCheck
-----------------------------------------------------------------------------*/
int LibFuncLoadCheckP4(void)
{
    int ret = 0;

    // General IF関数
    if ( PO_P4Create                        == NULL ) { ret = -1; printf("PO_P4Create==NULL¥n"); }
    if ( PO_P4Destroy                       == NULL ) { ret = -1; printf("PO_P4Destroy==NULL¥n"); }
    if ( PO_P4Open                          == NULL ) { ret = -1; printf("PO_P4Open==NULL¥n"); }
    if ( PO_P4Close                         == NULL ) { ret = -1; printf("PO_P4Close==NULL¥n"); }
    if ( PO_P4ClearMstProc                  == NULL ) { ret = -1;
printf("PO_P4ClearMstProc==NULL¥n"); }

    // Administrative関数
    if ( MC_AddAxisToGroup                  == NULL ) { ret = -1;
printf("MC_AddAxisToGroup==NULL¥n"); }
    if ( MC_RemoveAxisFromGroup             == NULL ) { ret = -1;
printf("MC_RemoveAxisFromGroup==NULL¥n"); }
    if ( MC_UngroupAllAxes                  == NULL ) { ret = -1;
printf("MC_UngroupAllAxes==NULL¥n"); }
    if ( MC_GroupReadConfiguration          == NULL ) { ret = -1;
printf("MC_GroupReadConfiguration==NULL¥n"); }
    if ( MC_GroupEnable                     == NULL ) { ret = -1; printf("MC_GroupEnable==NULL¥n"); }
    if ( MC_GroupDisable                    == NULL ) { ret = -1;
printf("MC_GroupDisable==NULL¥n"); }
    if ( MC_SetKinTransform                 == NULL ) { ret = -1;
printf("MC_SetKinTransform==NULL¥n"); }
    if ( MC_SetCartesianTransform           == NULL ) { ret = -1;
```

```
printf("MC_SetCartesianTransform==NULL¥n"); }
    if ( MC_SetCoordinateTransform          == NULL ) { ret = -1;
printf("MC_SetCoordinateTransform==NULL¥n"); }
    if ( MC_ReadKinTransform                == NULL ) { ret = -1;
printf("MC_ReadKinTransform==NULL¥n"); }
    if ( MC_ReadCartesianTransform          == NULL ) { ret = -1;
printf("MC_ReadCartesianTransform==NULL¥n"); }
    if ( MC_ReadCoordinateTransform         == NULL ) { ret = -1;
printf("MC_ReadCoordinateTransform==NULL¥n"); }
    if ( MC_GroupSetPosition                == NULL ) { ret = -1;
printf("MC_GroupSetPosition==NULL¥n"); }
    if ( MC_GroupReadActualPosition         == NULL ) { ret = -1;
printf("MC_GroupReadActualPosition==NULL¥n"); }
    if ( MC_GroupReadActualVelocity         == NULL ) { ret = -1;
printf("MC_GroupReadActualVelocity==NULL¥n"); }
    if ( MC_GroupReadActualAcceleration     == NULL ) { ret = -1;
printf("MC_GroupReadActualAcceleration==NULL¥n"); }
    if ( MC_GroupReadStatus                 == NULL ) { ret = -1;
printf("MC_GroupReadStatus==NULL¥n"); }
    if ( MC_GroupReadError                  == NULL ) { ret = -1;
printf("MC_GroupReadError==NULL¥n"); }
    if ( MC_GroupReset                      == NULL ) { ret = -1; printf("MC_GroupReset==NULL¥n"); }
    if ( MC_PathSelect                      == NULL ) { ret = -1; printf("MC_PathSelect==NULL¥n"); }
    if ( MC_GroupSetOverride                == NULL ) { ret = -1;
printf("MC_GroupSetOverride==NULL¥n"); }
    if ( MC_SetDynCoordTransform            == NULL ) { ret = -1;
printf("MC_SetDynCoordTransform==NULL¥n"); }

    // Motion関数
    if ( MC_GroupHome                       == NULL ) { ret = -1; printf("MC_GroupHome==NULL¥n"); }
    if ( MC_GroupStop                       == NULL ) { ret = -1; printf("MC_GroupStop==NULL¥n"); }
    if ( MC_GroupHalt                       == NULL ) { ret = -1; printf("MC_GroupHalt==NULL¥n"); }
    if ( MC_GroupInterrupt                  == NULL ) { ret = -1;
printf("MC_GroupInterrupt==NULL¥n"); }
    if ( MC_GroupContinue                   == NULL ) { ret = -1;
printf("MC_GroupContinue==NULL¥n"); }
    if ( MC_MoveLinearAbsolute              == NULL ) { ret = -1;
printf("MC_MoveLinearAbsolute==NULL¥n"); }
    if ( MC_MoveLinearRelative              == NULL ) { ret = -1;
printf("MC_MoveLinearRelative==NULL¥n"); }
    if ( MC_MoveCircularAbsolute            == NULL ) { ret = -1;
printf("MC_MoveCircularAbsolute==NULL¥n"); }
    if ( MC_MoveCircularRelative            == NULL ) { ret = -1;
printf("MC_MoveCircularRelative==NULL¥n"); }
    if ( MC_MoveDirectAbsolute              == NULL ) { ret = -1;
printf("MC_MoveDirectAbsolute==NULL¥n"); }
    if ( MC_MoveDirectRelative              == NULL ) { ret = -1;
printf("MC_MoveDirectRelative==NULL¥n"); }
    if ( MC_MovePath                        == NULL ) { ret = -1; printf("MC_MovePath==NULL¥n"); }
    if ( MC_SyncAxisToGroup                 == NULL ) { ret = -1;
printf("MC_SyncAxisToGroup==NULL¥n"); }
```

```
    if ( MC_SyncGroupToAxis                == NULL ) { ret = -1;
printf("MC_SyncGroupToAxis==NULL¥n"); }
    if ( MC_TrackConveyorBelt              == NULL ) { ret = -1;
printf("MC_TrackConveyorBelt==NULL¥n"); }
    if ( MC_TrackRotaryTable               == NULL ) { ret = -1;
printf("MC_TrackRotaryTable==NULL¥n"); }

    // ライブラリ関数
    if ( PO_WaitForP4MotionRecv            == NULL ) { ret = -1;
printf("PO_WaitForP4MotionRecv==NULL¥n"); }

    return ( ret ) ;
}
/*-------------------------------------------------------------------------------
    LibCompileTestPowerON()
-------------------------------------------------------------------------------*/
void LibCompileTestPowerON(void)
{
    int ret;
    int i;

    TFB_POWER_IN_LIB    pwr_in;
    TFB_POWER_OUT       pwr_out;

    // MC_Power
    DP(("---------------MC_Power ON start---------------¥n"));
    for(i = 1; i <= CTRL_AXIS_MAX; i++){

        memset(&pwr_in.Axis, 0, sizeof(TFB_POWER_IN_LIB));
        memset(&pwr_out.Status, 0, sizeof(TFB_POWER_OUT));

        pwr_in.Axis             = i;
        pwr_in.Enable           = DEF_ENB;
        pwr_in.Enable_Positive  = DEF_EPS;
        pwr_in.Enable_Negative  = DEF_ENN;
        pwr_in.BufferMode       = DEF_BFF;

        //MC_Power(&pwr_in, &pwr_out);
        ret = MC_Power(&pwr_in, &pwr_out);
        DP(("---MC_Power slv=%d ret=0x%08x¥n", pwr_in.Axis, ret));

        DP(("---Status=0x%02x, Busy=0x%02x,  Active=0x%02x¥n", pwr_out.Status, pwr_out.Busy,
pwr_out.Active));
        DP(("---Error=0x%02x,  Error=0x%02x, ErrorID=0x%08x¥n", pwr_out.Error, pwr_out.Error,
pwr_out.ErrorID));
    }
    DP(("---------------MC_Power ON end-----------------¥n"));
}
/*-------------------------------------------------------------------------------
    LibCompileTestPowerOFF()
-------------------------------------------------------------------------------*/
```

```
void LibCompileTestPowerOFF(void)
{
    int ret;
    int i;

    TFB_POWER_IN_LIB    pwr_in;
    TFB_POWER_OUT       pwr_out;

    // MC_Power
    DP(("---------------MC_Power OFF start---------------¥n"));
    for(i = 1; i <= CTRL_AXIS_MAX; i++){

        memset(&pwr_in.Axis, 0, sizeof(TFB_POWER_IN_LIB));
        memset(&pwr_out.Status, 0, sizeof(TFB_POWER_OUT));

        pwr_in.Axis             = i;
        pwr_in.Enable           = DEF_DIS;
        pwr_in.Enable_Positive  = DEF_EPS;
        pwr_in.Enable_Negative  = DEF_ENN;
        pwr_in.BufferMode       = DEF_BFF;

        //MC_Power(&pwr_in, &pwr_out);
        ret = MC_Power(&pwr_in, &pwr_out);
        DP(("---MC_Power slv=%d ret=0x%08x¥n", pwr_in.Axis, ret));

        DP(("---Status=0x%02x, Busy=0x%02x,  Active=0x%02x¥n", pwr_out.Status, pwr_out.Busy,
pwr_out.Active));
        DP(("---Error=0x%02x,  Error=0x%02x, ErrorID=0x%08x¥n", pwr_out.Error, pwr_out.Error,
pwr_out.ErrorID));
    }
    DP(("---------------MC_Power OFF end-----------------¥n"));
}
/*-------------------------------------------------------------------------
    LibCompileTestGroupAdd()
-------------------------------------------------------------------------*/
int LibCompileTestGroupAdd(void)
{
    int ret;
    int i;
    unsigned char done;

    done = 0;

    TFB_ADDAXISTOGRP_IN_LIB addgrp_in;
    TFB_ADDAXISTOGRP_OUT    addgrp_out;

    // MC_AddAxisToGroup
    DP(("---------------MC_AddAxisToGroup start---------------¥n"));
    for(i = 1; i <= CTRL_AXIS_MAX; i++){

        memset(&addgrp_in, 0, sizeof(TFB_ADDAXISTOGRP_IN_LIB));
```

```
        memset(&addgrp_out, 0, sizeof(TFB_ADDAXISTOGRP_OUT));

        addgrp_in.AxesGroup = DEF_GRPNO;
        addgrp_in.AxisNo = i;
        addgrp_in.IdentInGroup = i;
        addgrp_in.Execute = DEF_EXEC;

        ret = MC_AddAxisToGroup(&addgrp_in, &addgrp_out);
        DP(("---MC_AddAxisToGroup slv=%d ret=0x%08x¥n", addgrp_in.AxisNo, ret));
        DP(("---Done=0x%02x, Busy=0x%02x¥n", addgrp_out.Done, addgrp_out.Busy));
        DP(("---Error=0x%02x, ErrorID=0x%08x¥n", addgrp_out.Error, addgrp_out.ErrorID));
        if(addgrp_out.Error) done = 1;
    }
    DP(("---------------MC_AddAxisToGroup end-----------------¥n"));

    if(done) return -1;
    else return 0;
}
/*-----------------------------------------------------------------------------
    LibCompileTestGroupEnable()
-----------------------------------------------------------------------------*/
int LibCompileTestGroupEnable(void)
{
    int ret;
    unsigned char done;

    done = 0;

    TFB_GRPENA_IN_LIB   grpena_in;
    TFB_GRPENA_OUT      grpena_out;

    // MC_GroupEnable
    DP(("---------------MC_GroupEnable start---------------¥n"));

    memset(&grpena_in, 0, sizeof(TFB_GRPENA_IN_LIB));
    memset(&grpena_out, 0, sizeof(TFB_GRPENA_OUT));

    grpena_in.AxesGroup = DEF_GRPNO;
    grpena_in.Execute = DEF_EXEC;

    ret = MC_GroupEnable(&grpena_in, &grpena_out);
    DP(("---MC_GroupEnable grp=%d ret=0x%08x¥n", grpena_in.AxesGroup, ret));
    DP(("---Done=0x%02x, Busy=0x%02x¥n", grpena_out.Done, grpena_out.Busy));
    DP(("---Error=0x%02x, ErrorID=0x%08x¥n", grpena_out.Error, grpena_out.ErrorID));

    DP(("---------------MC_GroupEnable end-----------------¥n"));

    if(grpena_out.Done) return 0;
    else return -1;
}
```

```
/*--------------------------------------------------------------------------------
    LibCompileTestLineAbsoluteMove()
--------------------------------------------------------------------------------*/
void LibCompileTestLineAbsoluteMove(void)
{
    int ret;


    TFB_MVLINABS_IN_LIB mvlinabs_in;
    TFB_MVLINABS_OUT    mvlinabs_out;

    // MC_MoveLinearAbsolute
    DP(("--------------MC_MoveLinearAbsolute start--------------¥n"));

    memset(&mvlinabs_in, 0, sizeof(TFB_MVLINABS_IN_LIB));
    DP(("1¥n"));
    memset(&mvlinabs_out, 0, sizeof(TFB_MVLINABS_OUT));
    DP(("2¥n"));

    mvlinabs_in.AxesGroup = DEF_GRPNO;
    mvlinabs_in.Execute = DEF_EXEC;
    mvlinabs_in.Position[0] = 1000000.0;
    mvlinabs_in.Position[1] = 2000000.0;
    mvlinabs_in.Position[2] = 1500000.0;
    mvlinabs_in.Position[3] = 2500000.0;
    mvlinabs_in.Velocity = 1000000.0;
    mvlinabs_in.Acceleration = 10000000.0;
    mvlinabs_in.Deceleration = 10000000.0;
    DP(("3¥n"));

    ret = MC_MoveLinearAbsolute(&mvlinabs_in, NULL, 0);
    DP(("---MC_MoveLinearAbsolute grp=%d ret=0x%08x¥n", mvlinabs_in.AxesGroup, ret));
    //Add wait
    if(ret != 0){
        while(1){
            ret = PO_WaitForP4MotionRecv(ret, &mvlinabs_out, MOVE_TIMEOUT);
            DP(("---Done=0x%02x, Busy=0x%02x¥n", mvlinabs_out.Done, mvlinabs_out.Busy));
            if(mvlinabs_out.Done) break;
            RtSleep(RTSLEEP_TIME);
        }
    }

    memset(&mvlinabs_in, 0, sizeof(TFB_MVLINABS_IN_LIB));
    DP(("4¥n"));
    memset(&mvlinabs_out, 0, sizeof(TFB_MVLINABS_OUT));
    DP(("5¥n"));

    mvlinabs_in.AxesGroup = DEF_GRPNO;
    mvlinabs_in.Execute = DEF_EXEC;
    mvlinabs_in.Position[0] = 0.0;
    mvlinabs_in.Position[1] = 0.0;
```

```
    mvlinabs_in.Position[2] = 0.0;
    mvlinabs_in.Position[3] = 0.0;
    mvlinabs_in.Velocity = 1000000.0;
    mvlinabs_in.Acceleration = 10000000.0;
    mvlinabs_in.Deceleration = 10000000.0;
    DP(("6¥n"));

    ret = MC_MoveLinearAbsolute(&mvlinabs_in, NULL, 0);
    DP(("---MC_MoveLinearAbsolute grp=%d ret=0x%08x¥n", mvlinabs_in.AxesGroup, ret));
    //Add wait
    if(ret != 0){
        while(1){
            ret = PO_WaitForP4MotionRecv(ret, &mvlinabs_out, MOVE_TIMEOUT);
            if(mvlinabs_out.Done) break;
            RtSleep(RTSLEEP_TIME);
        }
    }

    DP(("---------------MC_MoveLinearAbsolute end-----------------¥n"));

}
```

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 を使用する事は出来ません。
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。
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_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 の時無限待ち)

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：パラメーター一覧を参照

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：パラメーター一覧を参照

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：パラメーター一覧を参照

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：パラメーター一覧を参照

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：パラメーター一覧を参照

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

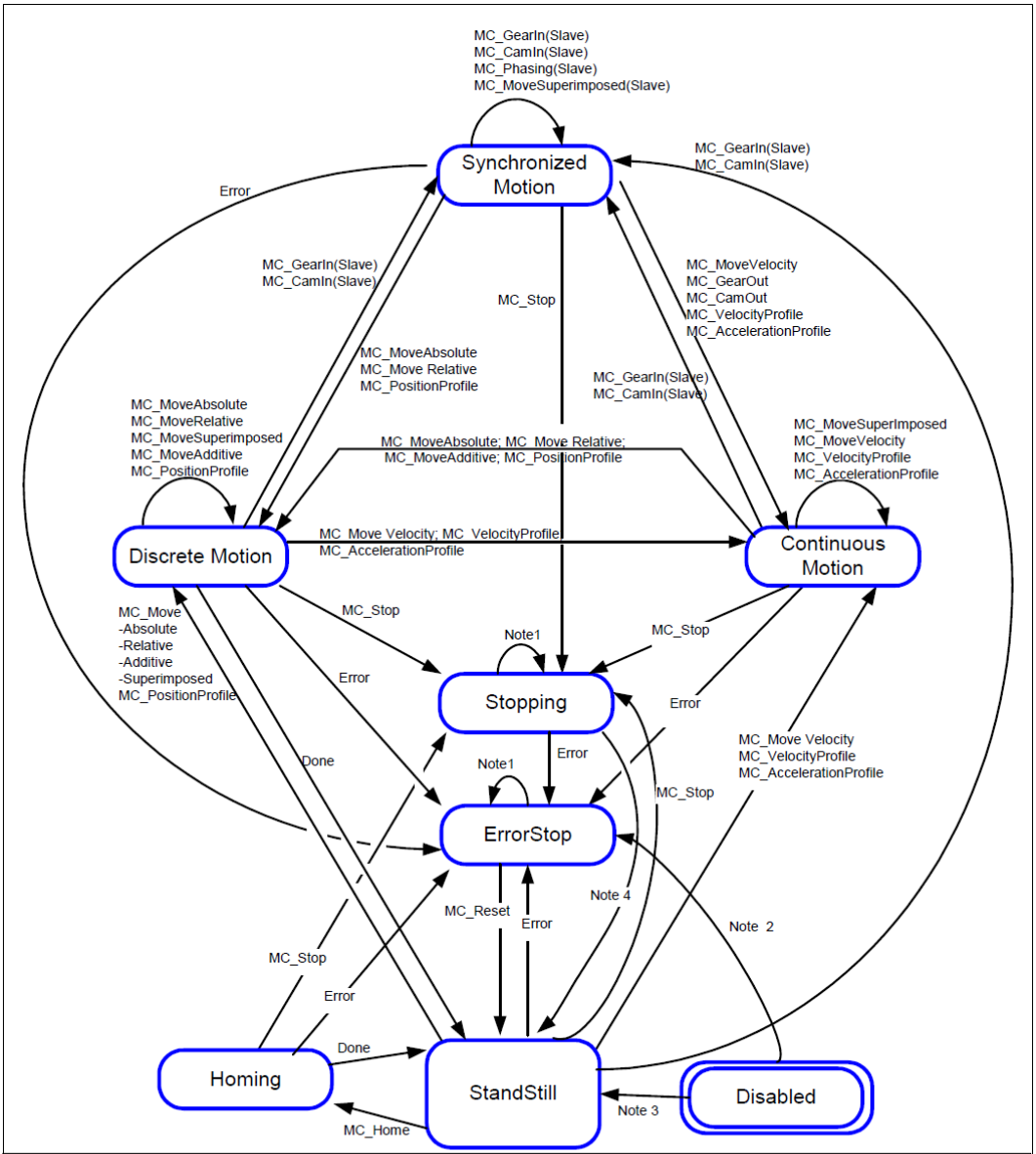
戻り値

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 参照）。



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(×10msec)(0 の時無限待ち)

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	―	―	未サポート
Direction	動作方向	MC_Direction (UINT16)	―	―	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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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 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 動作ファンクションブロックの多重起動」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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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　マスタ軸とスレーブ軸間の連動比率を設定します。

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(×10msec)(0 の時無限待ち)

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　指定したスレーブ軸を指定したマスタ軸に連動させます。スレーブ軸は SynchronizedMotion 状態に遷移され、マスタ軸が動作すると連動してスレーブ軸も動作します。

本 FB の制約として下記の３点があります。

① マスタ軸とスレーブ軸が停止しているときのみ実行可能です。
② 設定できる比率は 1:1 のみです。
③ GearOut が実行されるまで、スレーブ軸については、動作コマンドは受け付けません。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
slave_id	スレーブ軸	AXIS_REF (UINT)	1～62	-	スレーブとなる論理軸番号を指定
Master	マスタ軸	AXIS_REF (UINT)	1～62	-	マスタとなる論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	立ち上がり時に命令を実行
RatioNumerator	ギア比（分子）	UINT	1	1	本 FB では 1 固定
RatioDenominator	ギア比（分母）	UINT	1	1	本 FB では 1 固定
Acceleration	加速度	LREAL	―	―	未サポート
Deceleration	減速度	LREAL	―	―	未サポート
Jerk	加加速度	LREAL	―	―	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 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 動作ファンクションブロックの多重起動」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
InGear	同期比率到達	BOOL	TRUE or FALSE	設定した比率に到達したとき 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_GearOut 関数

機能　マスタ軸とスレーブ軸の連動を解除します。

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(×10msec)(0 の時無限待ち)

戻り値

0 以外	：正常(RtHandle 値)
0	：異常

説明　マスタ軸とスレーブ軸の連動動作を解除します。マスタ軸とスレーブ軸が停止している状態でのみ実行可能です。

本 FB の制約として下記の 2 点があります。

① マスタ軸とスレーブ軸が停止しているときのみ実行可能です。

② 実行完了後、スレーブの状態は StandStill 状態へ遷移します。

入力構造体詳細

変数	名称	データ型	範囲	初期値	内容
slave_id	スレーブ軸	AXIS_REF (UINT)	1～62	-	スレーブとなる論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	立ち上がり時に命令を実行

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	連動解除で 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_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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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	0～655 【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**

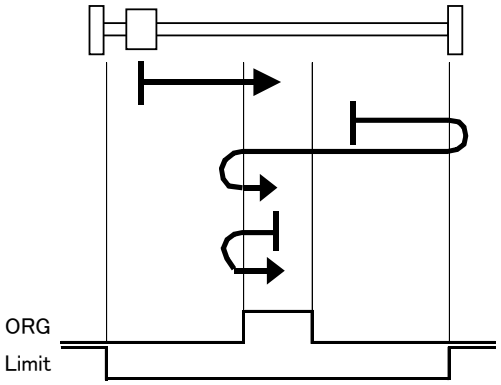
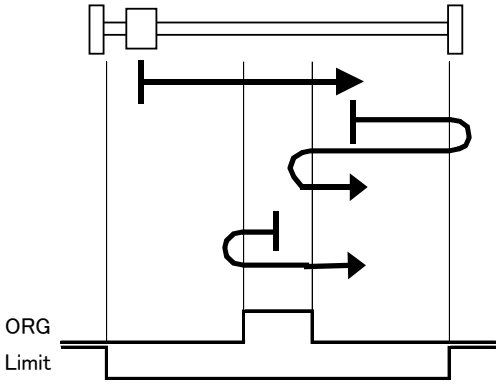
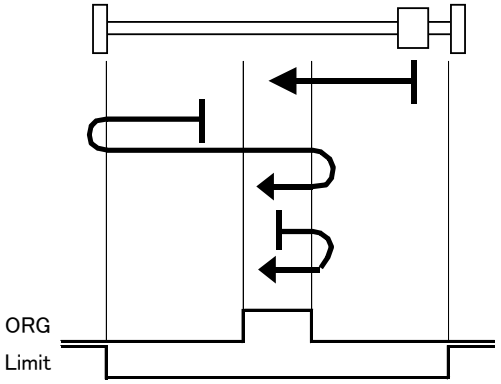
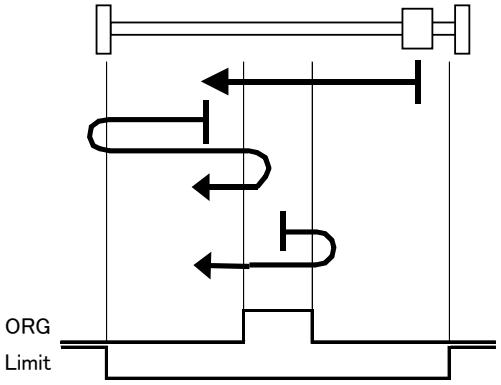
型名	型	値	モード	説明
MC_HomeDir	UINT16	0	MC_Positive	正方向へ開始（**未サポート**）
		1	MC_Negative	負方向へ開始（**未サポート**）
		2	MC_SwitchPositive	実行開始時に原点信号が ON していたら負方向へ開始、OFF していたら正方向へ開始
		3	MC_SwitchNegative	実行開始時に原点信号が ON していたら正方向へ開始、OFF していたら負方向へ開始
MC_SwitchMode	UINT16	0	MC_On	センサが ON 状態なら原点復帰完了（**未サポート**）
		1	MC_Off	センサが 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 動作ファンクションブロックの多重起動」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(×10msec)(0 の時無限待ち)

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

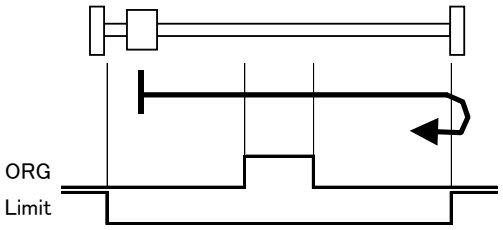
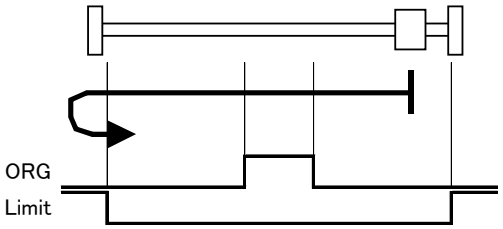
変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE ：実行 FALSE:実行無し
Direction	動作方向	MC_HomeDir	0～1	0	動作方向を指定
LimitSwitchMode	センサモード	MC_SwitchMode	0～3	0	原点復帰完了センサモード
Velocity	移動速度	LREAL	0,倍精度実数値正数 【指令単位/s】	0	位置決め時の速度
TorqueLimit	トルクリミット	LREAL	ー	ー	未サポート
TimeLimit	タイマリミット	LREAL	0～655 【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
		3	MC_EdgeOff	センサが ON→OFF のエッジで原点復帰完了（**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 動作ファンクションブロックの多重起動」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	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           timeout
);
```

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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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 動作ファンクションブロックの多重起動」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(×10msec)(0 の時無限待ち)

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

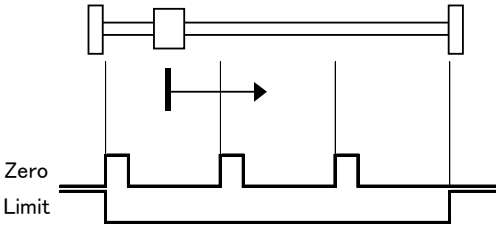
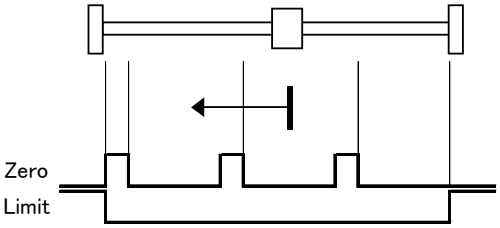
変数	名称	データ型	範囲	初期値	内容
Axis	軸	AXIS_REF (UINT)	1～62	-	論理軸番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE ：実行 FALSE:実行無し
Direction	動作方向	MC_HomeDir	0～1	0	動作方向を指定
Velocity	移動速度	LREAL	0,倍精度実数値正数 【指令単位/s】	0	位置決め時の速度
SetPosition	原点更新位置	LREAL	倍精度実数値 【指令単位】		
TorqueLimit	トルクリミット	LREAL	―	―	未サポート
TimeLimit	タイマリミット	LREAL	0～655 【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 動作ファンクションブロックの多重起動」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(×10msec)(0 の時無限待ち)

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 動作ファンクションブロックの多重起動」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(×10msec)(0 の時無限待ち)

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 動作ファンクションブロックの多重起動」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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

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(×10msec)(0 の時無限待ち)

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　未サポート

３－４－５　　PLCopen 仕様　MC Part4 初期化・終了 API 関数

本項では PLCopen 使用 MC Part4 機能を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_P4Create 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_P4Create (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ユニットにアクセス可能となります。
ユニットを使用する際には、必ずコールする必要があります。

PO_P4Destroy 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_P4Destroy (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、ユニットにアクセス不可になります。
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。

P0_P40pen 関数

機能　マスタプロセス処理の許可を行います。

書式　int　P0_P40pen (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。
ユニットを使用する際には、必ずコールする必要があります。

PO_P4Close 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_P4Close (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_P4ClearMstProc 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_P4ClearMstProc (void)

引数　なし

戻り値

1	: 正常
0	: 異常

説明　マスタプロセスへ初期化を行います。
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_WaitForP4MotionRecv 関数

機能　API 関数の応答待ち処理を行います。

書式　int　PO_WaitForP4MotionRecv (RTHANDLE hndl, void *out, unsigned long timeout)

引数

handle	: 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)

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、handle 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。

３－４－６　　PLCopen 仕様　MC Part4 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_AddAxisToGroup 関数

機能　軸追加

書式

```
RTHANDLE MC_AddAxisToGroup(
    TFB_ADDAXISTOGRP_IN_LIB *in,
    TFB_ADDAXISTOGRP_OUT    *out
);
```

引数

in	: 入力用構造体
out	: 出力用構造体

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　指定した軸グループに対して軸を追加します。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
AxisNo	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
IdentInGroup	軸識別情報	IDENT_IN_GROUP_REF (UINT)	1～16	-	グループに追加する軸の識別番号を指定

出力用構造体詳細

変数	名称	データ型	範囲	内容
Done	完了	BOOL	TRUE or FALSE	正常終了時に 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_RemoveAxisFromGroup 関数

機能　軸削除

書式

```
RTHANDLE MC_RemoveAxisFromGroup(
    TFB_REMAXISFROMGRP_IN_LIB *in,
    TFB_REMAXISFROMGRP_OUT    *out
);
```

引数

in	：入力用構造体
out	：出力用構造体

戻り値

0 以外	：正常(RtHandle 値)
0	：異常

説明　指定した軸グループから軸を削除します。
削除後に軸グループから軸が無くなった場合、グループの状態を GroupDisabled 状態に変更します。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE ：実行 FALSE:実行無し
IdentInGroup	軸識別情報	IDENT_IN_GROUP_REF (UINT)	1～16	-	グループに追加する軸の識別番号を指定

出力用構造体詳細

変数	名称	データ型	範囲	内容
Done	完了	BOOL	TRUE or FALSE	正常終了時に 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_UngroupAllAxes 関数

機能　軸グループ解除

書式
```
RTHANDLE MC_UngroupAllAxes(
    TFB_UNGRPALLAXES_IN_LIB *in,
    TFB_UNGRPALLAXES_OUT    *out
);
```

引数
in　：入力用構造体
out　：出力用構造体

戻り値
0 以外　：正常(RtHandle 値)
0　：異常

説明　指定した軸グループから軸を全て削除します。
削除完了後、グループの状態を GroupDisabled 状態に変更します。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し

出力用構造体詳細

変数	名称	データ型	範囲	内容
Done	完了	BOOL	TRUE or FALSE	正常終了時に 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_GroupReadConfiguration 関数

機能　軸グループ設定読出

書式
```
RTHANDLE MC_GroupReadConfiguration(
    TFB_GRPRDCFG_IN_LIB *in,
    TFB_GRPRDCFG_OUT    *out
);
```

引数
in	：	入力用構造体
out	：	出力用構造体

戻り値
0 以外	：	正常(RtHandle 値)
0	：	異常

説明　指定した軸グループに追加されている軸番号を読み出します。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE ：実行 FALSE:実行無し
IdentInGroup	軸識別情報	IDENT_IN_GROUP_REF (UINT)	1～16	FALSE	グループに追加した軸の識別番号を指定
CoordSystem	座標系選択	ENUM (UINT)	-	-	未サポート

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
Axis	軸	AXIS_REF (UINT)	1～62	論理軸番号
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_GroupEnable 関数

機能　軸グループ有効

書式
```
RTHANDLE MC_GroupEnable(
    TFB_GRPENA_IN_LIB *in,
    TFB_GRPENA_OUT    *out
);
```

引数
in　：入力用構造体
out　：出力用構造体

戻り値
0 以外　：正常(RtHandle 値)
0　：異常

説明　軸グループの状態を GroupDisabled から GroupStandby に遷移します。
本 API 関数が正常終了後、動作 API 関数を実行する事ができます。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し

出力用構造体詳細

変数	名称	データ型	範囲	内容
Done	完了	BOOL	TRUE or FALSE	正常終了時に 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_GroupDisable 関数

機能　軸グループ無効

書式
```
RTHANDLE MC_GroupDisable(
    TFB_GRPDIS_IN_LIB *in,
    TFB_GRPDIS_OUT    *out
);
```

引数
in　：入力用構造体
out　：出力用構造体

戻り値
0 以外　：正常(RtHandle 値)
0　：異常

説明　軸グループの状態を GroupDisabled に遷移します。
本 API 関数実行後は動作 API 関数は実行できません。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し

出力用構造体詳細

変数	名称	データ型	範囲	内容
Done	完了	BOOL	TRUE or FALSE	正常終了時に 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_SetKinTransform 関数

機能　運動学的変換設定

書式

```
RTHANDLE MC_SetKinTransform(
    TFB_SETKINTRANS_IN_LIB *in,
    TFB_SETKINTRANS_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_SetCartesianTransform 関数

機能　直交座標変換設定

書式
```
RTHANDLE MC_SetCartesianTransform(
    TFB_SETCARTTRANS_IN_LIB *in,
    TFB_SETCARTTRANS_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_SetCoordinateTransform 関数

機能　座標変換設定

書式
```
RTHANDLE MC_SetCoordinateTransform(
    TFB_SETCOORDTRANS_IN_LIB *in,
    TFB_SETCOORDTRANS_OUT    *out
);
```

引数

in	：入力用構造体
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

MC_ReadKinTransform 関数

機能　運動学的変換読出

書式
```
RTHANDLE MC_ReadKinTransform(
    TFB_RDKINTRANS_IN_LIB *in,
    TFB_RDKINTRANS_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_ReadCartesianTransform 関数

機能　直交座標変換読出

書式

```
RTHANDLE MC_ReadCartesianTransform(
    TFB_RDCARTTRANS_IN_LIB *in,
    TFB_RDCARTTRANS_OUT    *out
);
```

引数

in	：入力用構造体
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

MC_ReadCoordinateTransform 関数

機能　座標変換読出

書式

```
RTHANDLE MC_ReadCoordinateTransform(
    TFB_RDCOORDTRANS_IN_LIB *in,
    TFB_RDCOORDTRANS_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_GroupSetPosition 関数

機能　軸グループ現在位置変更

書式

```
RTHANDLE MC_GroupSetPosition(
    TFB_GRPSETPOS_IN_LIB *in,
    TFB_GRPSETPOS_OUT    *out
);
```

引数

in	：	入力用構造体
out	：	出力用構造体

戻り値

0 以外	：	正常(RtHandle 値)
0	：	異常

説明　軸グループの現在位置を変更します。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE ：実行 FALSE:実行無し
Position	目標座標配列	ARRAY [1..16] of LREAL	倍精度実数値 【指令単位】	-	目標位置を指定 （位置モードに従う）
Relative	位置モード	BOOL	TRUE or FALSE	FALSE	TRUE ：相対位置指令 FALSE:絶対位置指令
CoordSystem	座標系選択	ENUM (UINT)	-	-	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 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「3-5-2 Part4 動作ファンクションブロックの多重起動」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
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
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_GroupReadActualPosition 関数

機能　軸グループ現在位置読出

書式

```
RTHANDLE MC_GroupReadActualPosition(
    TFB_GRPRDPOS_IN_LIB *in,
    TFB_GRPRDPOS_OUT    *out
);
```

引数

in	：	入力用構造体
out	：	出力用構造体

戻り値

0 以外	：	正常(RtHandle 値)
0	：	異常

説明　軸グループの現在位置を読み出します。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE ：実行 FALSE:実行無し
CoordSystem	座標系選択	ENUM (UINT)	-	-	未サポート

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
Position	現在座標配列	ARRAY [1..16] of LREAL	倍精度実数値 【指令単位】	絶対座標による現在位置

※1：エラーコード一覧を参照

MC_GroupReadActualVelocity 関数

機能　軸グループ現在速度読出

書式

```
RTHANDLE MC_GroupReadActualVelocity(
    TFB_GRPRDVEL_IN_LIB *in,
    TFB_GRPRDVEL_OUT    *out
);
```

引数

in	: 入力用構造体
out	: 出力用構造体

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　軸グループの現在速度を読み出します。
読み出す現在速度は、グループ内の軸毎の現在速度と、同期のための合成速度になります。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
CoordSystem	座標系選択	ENUM (UINT)	-	-	未サポート

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
Velocity	現在速度配列	ARRAY [1..16] of LREAL	倍精度実数値【指令単位】	軸毎の移動速度
PathVelocity	経路速度	LREAL	倍精度実数値【指令単位】	経路上の移動速度

※1：エラーコード一覧を参照

MC_GroupReadActualAcceleration 関数

機能　軸グループ現在加速度読出

書式

```
RTHANDLE MC_GroupReadActualAcceleration(
    TFB_GRPRDACC_IN_LIB *in,
    TFB_GRPRDACC_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_GroupReadStatus 関数

機能　　軸グループステータス読出

書式

```
RTHANDLE MC_GroupReadStatus(
    TFB_GRPRDSTS_IN_LIB *in,
    TFB_GRPRDSTS_OUT    *out
);
```

引数

in	：	入力用構造体
out	：	出力用構造体

戻り値

0 以外	：	正常(RtHandle 値)
0	：	異常

説明　　指定したグループの状態遷移図上の状態を読み出します。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
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
GroupMoving	Group Moving 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。
GroupHoming	Group Homing 状態	BOOL	TRUE or FALSE	
GroupErrorStop	Group ErrorStop 状態	BOOL	TRUE or FALSE	
GroupStandby	Group Standby 状態	BOOL	TRUE or FALSE	
GroupStopping	Group Stopping 状態	BOOL	TRUE or FALSE	
GroupDisabled	Group Disabled 状態	BOOL	TRUE or FALSE	
ConstantVelocity	定速動作中	BOOL	TRUE or FALSE	グループが定速動作中のとき TRUE
Accelerating	加速中	BOOL	TRUE or FALSE	グループが加速中のとき TRUE
Decelerating	減速中	BOOL	TRUE or FALSE	グループが減速中のとき TRUE
InPosition	位置範囲内	BOOL	TRUE or FALSE	グループが目標位置範囲内の時 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_GroupReadError 関数

機能　軸グループエラー読出

書式

```
RTHANDLE MC_GroupReadError(
    TFB_GRPRDERR_IN_LIB *in,
    TFB_GRPRDERR_OUT    *out
);
```

引数

in	：	入力用構造体
out	：	出力用構造体

戻り値

0 以外	：	正常(RtHandle 値)
0	：	異常

説明　軸グループに発生しているエラーを読み出します。
軸毎に発生しているエラーについては、MC_ReadError をご使用ください。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
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
GroupErrorID	グループ エラー番号	DWORD	※1	グループ内のエラーコードを出力

※1：エラーコード一覧を参照

MC_GroupReset 関数

機能　軸グループエラーリセット

書式
```
RTHANDLE MC_GroupReset(
    TFB_GRPRESET_IN_LIB *in,
    TFB_GRPRESET_OUT    *out
);
```

引数
in　：入力用構造体
out　：出力用構造体

戻り値
0 以外　：正常(RtHandle 値)
0　：異常

説明　軸グループに発生しているエラーをリセットします。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し

出力用構造体詳細

変数	名称	データ型	範囲	内容
Done	完了	BOOL	TRUE or FALSE	正常終了時に 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_PathSelect 関数

機能　経路選択

書式
```
RTHANDLE MC_PathSelect(
    TFB_PATHSEL_IN_LIB *in,
    TFB_PATHSEL_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_GroupSetOverride 関数

機能　軸グループオーバーライド値設定

書式
```
RTHANDLE MC_GroupSetOverride(
    TFB_GRPSETOVERRIDE_IN_LIB *in,
    TFB_GRPSETOVERRIDE_OUT    *out
);
```

引数
in　：入力用構造体
out　：出力用構造体

戻り値
0 以外　：正常(RtHandle 値)
0　：異常

説明　実行中の位置決め動作の、速度・加速(減速)度に対して倍率指定によりパラメータを変更します。
速度倍率に対して 0.0 を指定した場合、状態遷移を発生させずに軸を停止します。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Enable	有効	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
VelFactor	速度倍率	LREAL	倍精度実数値正数 【0.0<=値<=2.0】	-	現在の速度に対する 倍率を指定
AccFactor	加速度倍率	LREAL	倍精度実数値正数 【0.0<値<=2.0】	-	現在の加速度に対する 倍率を指定
JerkFactor	加加速度倍率	LREAL	倍精度実数値正数 【0.0<値<=2.0】	-	未サポート

出力用構造体詳細

変数	名称	データ型	範囲	内容
Enabled	有効化完了	BOOL	TRUE or FALSE	正常終了時に 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_SetDynCoordTransform 関数

機能　動的座標変換設定

書式
```
RTHANDLE MC_SetDynCoordTransform(
    TFB_SETDYNCOORDTRANS_IN_LIB *in,
    TFB_SETDYNCOORDTRANS_OUT    *out
);
```

引数

in	: 入力用構造体
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

３－４－７　　PLCopen 仕様　MC Part4 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。

MC_GroupHome 関数

機能　未サポート

書式

```
RTHANDLE MC_GroupHome(
    TFB_GRPHOME_IN_LIB *in,
    TFB_GRPHOME_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_GroupStop 関数

機能　軸グループ強制停止

書式
```
RTHANDLE MC_GroupStop(
    TFB_GRPSTOP_IN_LIB *in,
    TFB_GRPSTOP_OUT    *out
);
```

引数
in　：入力用構造体
out　：出力用構造体

戻り値
0 以外　：正常(RtHandle 値)
0　：異常

説明　軸グループの位置決めを停止します。
実行後 GroupStopping 状態に遷移し、停止完了後に GroupStandby 状態に遷移します。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
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	位置決め時の減速度
Jerk	加加速度	LREAL	-	-	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 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「3-5-2 Part4 動作ファンクションブロックの多重起動」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
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
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_GroupHalt 関数

機能　軸グループ停止

書式

```
RTHANDLE MC_GroupHalt(
    TFB_GRPHALT_IN_LIB *in,
    TFB_GRPHALT_OUT    *out
);
```

引数

in	：	入力用構造体
out	：	出力用構造体

戻り値

0 以外	：	正常(RtHandle 値)
0	：	異常

説明

軸グループの位置決めを停止します。
本 API 関数は GroupStopping 状態への遷移を発生させずに位置決めを停止します。
停止後に GroupStandby 状態に遷移します。
GroupStopping 状態へ遷移しないため、本 API 関数実行中に位置決め API 関数の起動が可能です。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
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	位置決め時の減速度
Jerk	加加速度	LREAL	-	-	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 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「3-5-2 Part4 動作ファンクションブロックの多重起動」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
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
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_GroupInterrupt 関数

機能　軸グループ一時停止

書式
```
RTHANDLE MC_GroupInterrupt(
    TFB_GRPINT_IN_LIB *in,
    TFB_GRPINT_OUT    *out
);
```

引数
in	: 入力用構造体
out	: 出力用構造体

戻り値
0 以外	: 正常(RtHandle 値)
0	: 異常

説明　軸グループの位置決めを一時停止します。
一時停止中、停止後も状態遷移は発生しません。
本 API 関数後に MC_GroupContinue を実行する事で、続きの動作を行うことができます。
また、一時停止前とは違う位置決め関数の実行もできます。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
Deceleration	減速度	LREAL	-	-	未サポート
Jerk	加加速度	LREAL	-	-	未サポート

出力用構造体詳細

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
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_GroupContinue 関数

機能　軸グループ一時停止解除

書式

```
RTHANDLE MC_GroupContinue(
    TFB_GRPCONT_IN_LIB *in,
    TFB_GRPCONT_OUT    *out
);
```

引数

in	: 入力用構造体
out	: 出力用構造体

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　MC_GroupInterrupt により一時停止された動作を再開する事ができます。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し

出力用構造体詳細

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
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_MoveLinearAbsolute 関数

機能　絶対値直線補間

書式

```
RTHANDLE MC_MoveLinearAbsolute(
    TFB_MVLINABS_IN_LIB *in,
    TFB_MVLINABS_OUT    *out
);
```

引数

in	: 入力用構造体
out	: 出力用構造体

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　絶対座標指令による直線補間を行います。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
Position	目標座標配列	ARRAY [1..16] of LREAL	倍精度実数値 【指令単位】	-	目標位置を指定 (位置モードに従う)
Velocity	速度	LREAL	倍精度実数値正数 【指令単位/s】	-	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】	-	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	-	-	未サポート
CoordSystem	座標系選択	ENUM (UINT)	-	-	未サポート
BufferMode	動作モード	MC_BufferMode (UINT16)	0～5	0	モーション命令 多重起動命令動作指定 **0: Aborting** **1: Buffered** **3: BlendingPrevious のみ**
Transition Mode	トランジションモード	MC_TransitionMode (UINT16)	0 or 10	0	FB 同士を接続したときの遷移カーブを決定します。 **0: TMNone** **10: TMSmoothing のみ**
Transition Parameter	トランジションパラメータ	ARRAY [1..4] of LREAL	-	-	未サポート

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「3-5-2 Part4 動作ファンクションブロックの多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	
MC_TransitionMode	UINT16	0	TMNone	重ねて次の動作を行うときの速度および軌跡の遷移カーブを指定 詳細は「3-5-2 Part4 動作ファンクションブロックの多重起動」を参照してください。
		1～9	PLCopen で予約	
		10	TMSmoothing	

出力用構造体詳細

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
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_MoveLinearRelative 関数

機能　相対値直線補間

書式
```
RTHANDLE MC_MoveLinearRelative(
    TFB_MVLINREL_IN_LIB *in,
    TFB_MVLINREL_OUT    *out
);
```

引数

in	: 入力用構造体
out	: 出力用構造体

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　現在位置に対する相対座標指定による直線補間を行います。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE :実行 FALSE:実行無し
Distance	目標座標配列	ARRAY [1..16] of LREAL	倍精度実数値 【指令単位】	-	目標位置を指定 (位置モードに従う)
Velocity	速度	LREAL	倍精度実数値正数 【指令単位/s】	-	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】	-	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	-	-	未サポート
CoordSystem	座標系選択	ENUM (UINT)	-	-	未サポート
BufferMode	動作モード	MC_BufferMode (UINT16)	0～5	0	モーション命令 多重起動命令動作指定 **0: Aborting** **1: Buffered** **3: BlendingPrevious のみ**
Transition Mode	トランジションモード	MC_TransitionMode (UINT16)	0 or 10	0	FB 同士を接続したときの遷移カーブを決定します。 **0: TMNone** **10: TMSmoothing のみ**
Transition Parameter	トランジションパラメータ	ARRAY [1..4] of LREAL	-	-	未サポート

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「3-5-2 Part4 動作ファンクションブロックの多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	
MC_TransitionMode	UINT16	0	TMNone	重ねて次の動作を行うときの速度および軌跡の遷移カーブを指定 詳細は「3-5-2 Part4 動作ファンクションブロックの多重起動」を参照してください。
		1～9	PLCopen で予約	
		10	TMSmoothing	

出力用構造体詳細

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
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_MoveCircularAbsolute 関数

機能　絶対値円弧補間

書式

```
RTHANDLE MC_MoveCircularAbsolute(
    TFB_MVCIRCABS_IN_LIB *in,
    TFB_MVCIRCABS_OUT    *out
);
```

引数

in	: 入力用構造体
out	: 出力用構造体

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　絶対座標指令による 2 軸円弧補間を行います。2 軸以上登録されているグループに対して実行した場合はエラーとなります。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE ：実行 FALSE:実行無し
CircMode	円弧補間モード	ENUM (UINT)	0～2	0	AuxPoint,EndPoint の意味を決定
AuxPoint	補助点	ARRAY [1..16] of LREAL	倍精度実数値【指令単位】	-	CircMode に従う。 絶対位置指定
EndPoint	終点	ARRAY [1..16] of LREAL	倍精度実数値【指令単位】	-	CircMode に従う。 絶対位置指定
PathChoice	経路選択	MC_CIRC_PATHCHOICE (UINT)	0～1	0	0:時計回り 1:反時計回り
Velocity	速度	LREAL	倍精度実数値正数【指令単位/s】	-	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】	-	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	-	-	未サポート
CoordSystem	座標系選択	ENUM (UINT)	-	-	未サポート
BufferMode	動作モード	MC_BufferMode (UINT16)	0～5	0	モーション命令 多重起動命令動作指定 **0: Aborting** **1: Buffered のみ**
Transition Mode	トランジションモード	MC_TransitionMode (UINT16)	0 or 10	0	FB 同士を接続したときの遷移カーブを決定します。 **0: TMNone** **10: TMSmoothing のみ**
Transition Parameter	トランジションパラメータ	ARRAY [1..4] of LREAL	-	-	未サポート

型名	型	値	モード	説明
MC_CircMode	UINT16	0	BORDER	「AuxPoint」はスタート位置から、「EndPoint」で構成される円周上の点。
		1	CENTER	「AuxPoint」は中心点。
		2	RADIUS	「AuxPoint」は、右母指の規則による円平面の垂直の先頭ポイントを定義。 円の半径はベクトルの長さ。
MC_PathChoice	UINT16	0	時計回り（CW）	スタート位置から「EndPoint」へ時計回りに円弧補間を実行します。
		1	反時計回り（CCW）	スタート位置から「EndPoint」へ反時計回りに円弧補間を実行します。

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「3-5-2 Part4 動作ファンクションブロックの多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	
MC_TransitionMode	UINT16	0	TMNone	重ねて次の動作を行うときの速度および軌跡の遷移カーブを指定 詳細は「3-5-2 Part4 動作ファンクションブロックの多重起動」を参照してください。
		1～9	PLCopen で予約	
		10	TMSmoothing	

出力用構造体詳細

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
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_MoveCircularRelative 関数

機能　相対値円弧補間

書式

```
RTHANDLE MC_MoveCircularRelative(
    TFB_MVCIRCREL_IN_LIB *in,
    TFB_MVCIRCREL_OUT    *out
);
```

引数

in	: 入力用構造体
out	: 出力用構造体

戻り値

0 以外	: 正常(RtHandle 値)
0	: 異常

説明　現在位置に対する相対座標指定による円弧補間を行います。2 軸以上登録されているグループに対して実行した場合はエラーとなります。

入力用構造体詳細

変数	名称	データ型	範囲	初期値	内容
group_id	軸グループ	AXES_GROUP_REF (UINT)	1～32	-	論理グループ番号を指定
Execute	起動トリガ	BOOL	TRUE or FALSE	FALSE	TRUE ：実行 FALSE:実行無し
CircMode	円弧補間モード	ENUM (UINT)	0～2	0	AuxPoint,EndPoint の意味を決定
AuxPoint	補助点	ARRAY [1..16] of LREAL	倍精度実数値【指令単位】	-	CircMode に従う。 相対位置指定
EndPoint	終点	ARRAY [1..16] of LREAL	倍精度実数値【指令単位】	-	CircMode に従う。 絶対位置指定
PathChoice	経路選択	MC_CIRC_PATHCHOICE (UINT)	0～1	0	0:時計回り 1:反時計回り
Velocity	速度	LREAL	倍精度実数値正数【指令単位/s】	-	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】	-	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	-	-	未サポート
CoordSystem	座標系選択	ENUM (UINT)	-	-	未サポート
BufferMode	動作モード	MC_BufferMode (UINT16)	0～5	0	モーション命令 多重起動命令動作指定 **0: Aborting** **1: Buffered のみ**
Transition Mode	トランジションモード	MC_TransitionMode (UINT16)	0 or 10	0	FB 同士を接続したときの遷移カーブを決定します。 **0: TMNone** **10: TMSmoothing のみ**
Transition Parameter	トランジションパラメータ	ARRAY [1..4] of LREAL	倍精度実数値【指令単位】	-	トランジションモードにより変化

型名	型	値	モード	説明
MC_CircMode	UINT16	0	BORDER	「AuxPoint」はスタート位置から、「EndPoint」で構成される円周上の点。
		1	CENTER	「AuxPoint」は中心点。
		2	RADIUS	「AuxPoint」は、右母指の規則による円平面の垂直の先頭ポイントを定義。 円の半径はベクトルの長さ。
MC_PathChoice	UINT16	0	時計回り（CW）	スタート位置から「EndPoint」へ時計回りに円弧補間を実行します。
		1	反時計回り（CCW）	スタート位置から「EndPoint」へ反時計回りに円弧補間を実行します。

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「3-5-2 Part4 動作ファンクションブロックの多重起動」を参照してください。
		1	Buffered	
		2	BlendingLow	
		3	BlendingPrevious	
		4	BlendingNext	
		5	BlendingHigh	
MC_TransitionMode	UINT16	0	TMNone	重ねて次の動作を行うときの速度および軌跡の遷移カーブを指定 詳細は「3-5-2 Part4 動作ファンクションブロックの多重起動」を参照してください。
		1～9	PLCopen で予約	
		10	TMSmoothing	

出力用構造体詳細

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
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_MoveDirectAbsolute 関数

機能　軸グループ絶対値位置決め

書式

```
RTHANDLE MC_MoveDirectAbsolute(
    TFB_MVDIRABS_IN_LIB *in,
    TFB_MVDIRABS_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_MoveDirectRelative 関数

機能　軸グループ相対値位置決め

書式

```
RTHANDLE MC_MoveDirectRelative(
    TFB_MVDIRREL_IN_LIB *in,
    TFB_MVDIRREL_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_MovePath 関数

機能　指定経路移動

書式

```
RTHANDLE MC_MovePath(
    TFB_MVPATH_IN_LIB *in,
    TFB_MVPATH_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_SyncAxisToGroup 関数

機能　グループへの単軸同期動作

書式
```
RTHANDLE MC_SyncAxisToGroup(
    TFB_SYNCAXISTOGRP_IN_LIB *in,
    TFB_SYNCAXISTOGRP_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_SyncGroupToAxis 関数

機能　単軸へのグループ同期動作

書式

```
RTHANDLE MC_SyncGroupToAxis(
    TFB_SYNCGRPTOAXIS_IN_LIB *in,
    TFB_SYNCGRPTOAXIS_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_TrackConveyorBelt 関数

機能　コンベヤ追従動作

書式

```
RTHANDLE MC_TrackConveyorBelt(
    TFB_TRACKCONVBELT_IN_LIB *in,
    TFB_TRACKCONVBELT_OUT    *out
);
```

引数

in	: 入力用構造体
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

MC_TrackRotaryTable 関数

機能　ロータリテーブル追従動作

書式
```
RTHANDLE MC_TrackRotaryTable(
    TFB_TRACKROTTBL_IN_LIB *in,
    TFB_TRACKROTTBL_OUT    *out
);
```

引数

in	: 入力用構造体
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

３－５　　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        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));
}
```

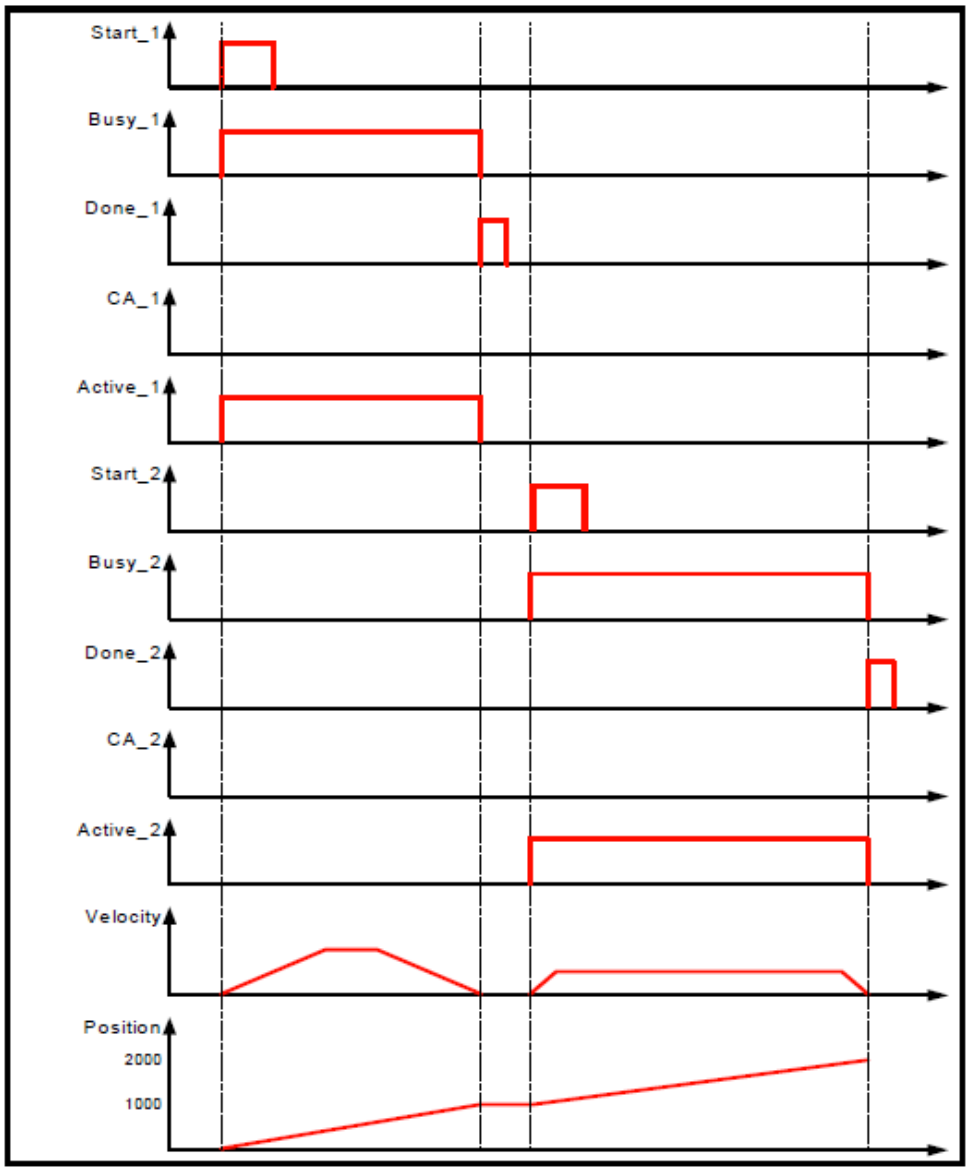


図 3-5-1-1. 上の例で API 関数 1 と API 関数 2 の間に干渉が無い場合でのタイムチャート (Aborting Mode)

● Aborting mode での動作

```
//-----------------------------------------------------------------------------
//       Aborting Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------------
int                 ret;
TFB_MVABS_IN        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));
}
```

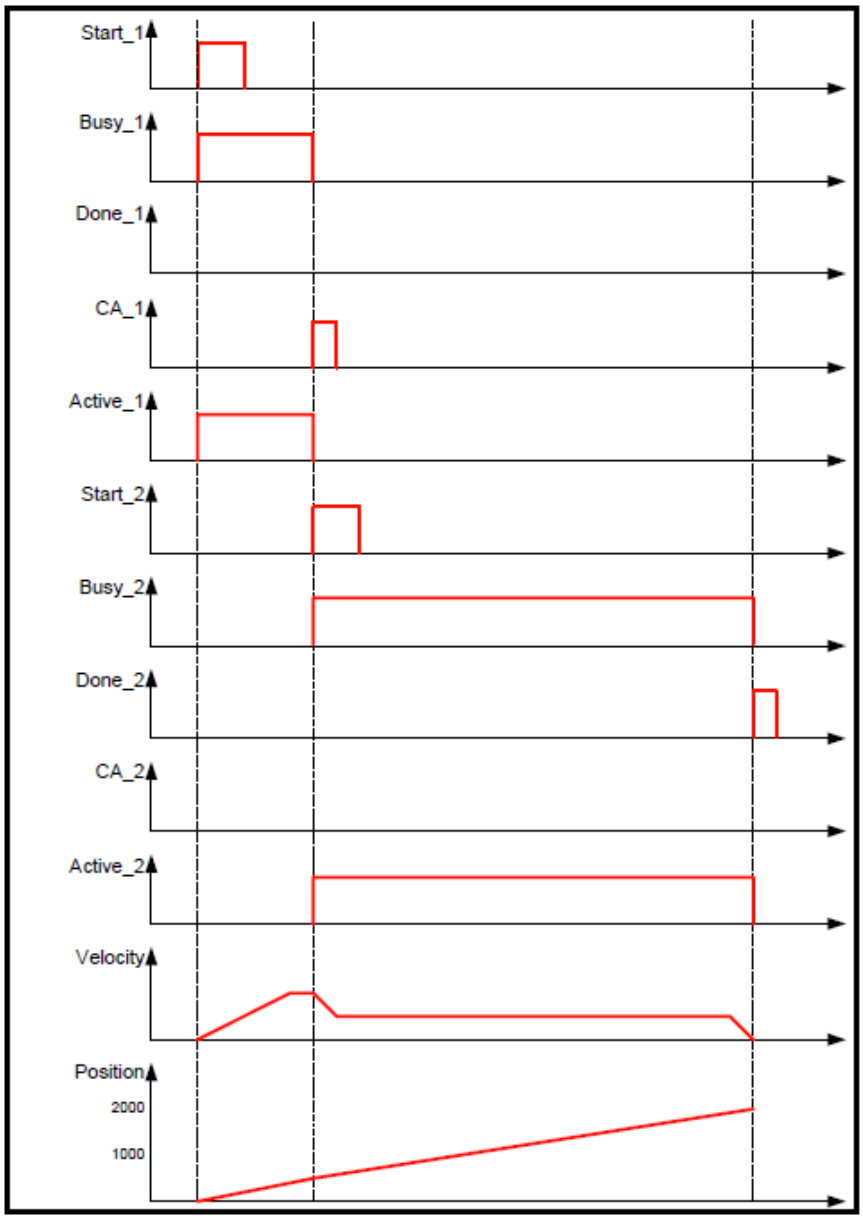


図 3-5-1-2. 上の例の API 関数 2 が API 関数 1 に割り込む場合でのタイムチャート（Aborting Mode）

● Bufferd mode での動作

```
//-----------------------------------------------------------------------------
//       Buffered Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------------
int                ret;
TFB_MVABS_IN       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));
}
```

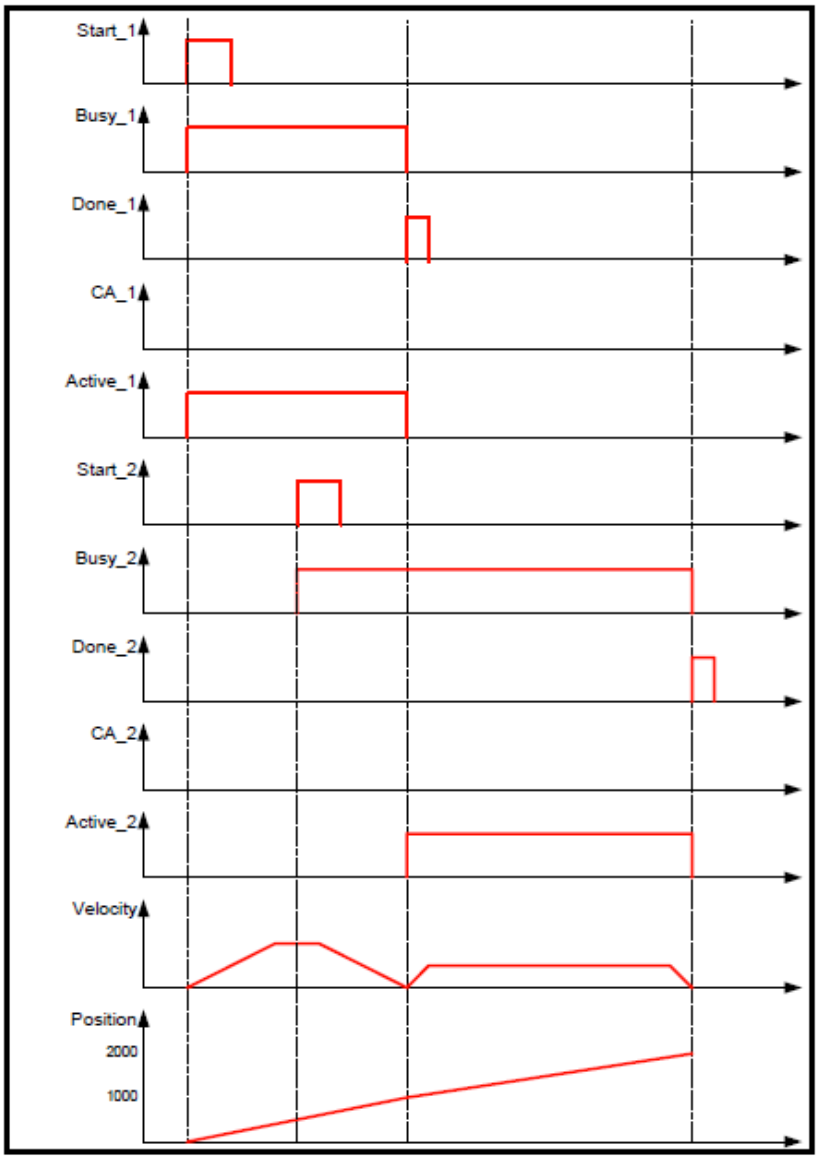


図 3-5-1-3. 上の例の Buffered Mode でのタイムチャート
（速度 0 で停止し、遅延なくその位置で API 関数 2 を開始する）

- BlendingLow mode での動作

```
//-----------------------------------------------------------------------------
//       BlendingLow Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------------
int               ret;
TFB_MVABS_IN      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));
}
```

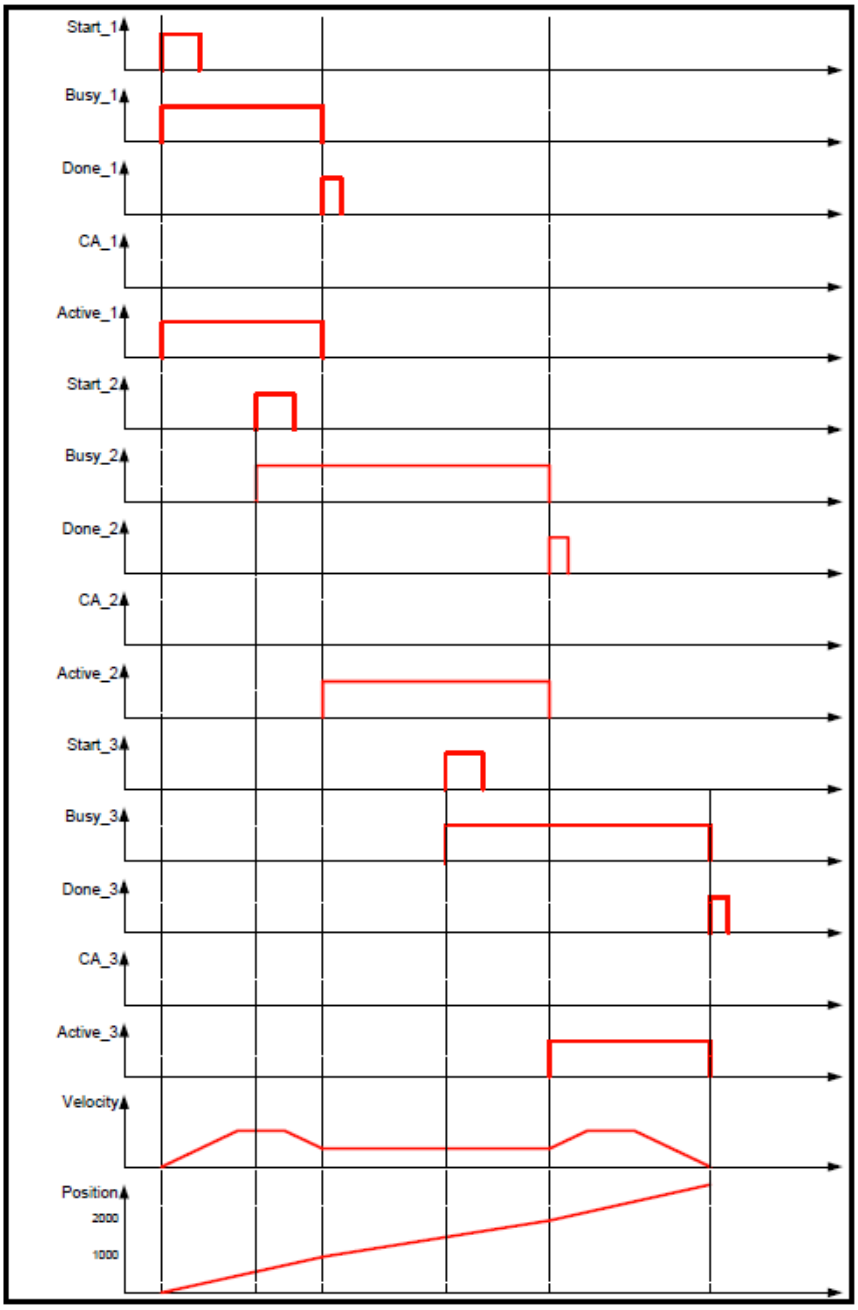


図 3-5-1-4. 上の例の BlendingLow Mode でのタイムチャート
（API 関数 1 の最終位置から API 関数 2 の最終位置まで低い方の速度（velocity2）を使用）

- BlendingPrevious mode での動作

```
//-----------------------------------------------------------------------
//       BlendingPrevious Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------
int                ret;
TFB_MVABS_IN       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));
}
```

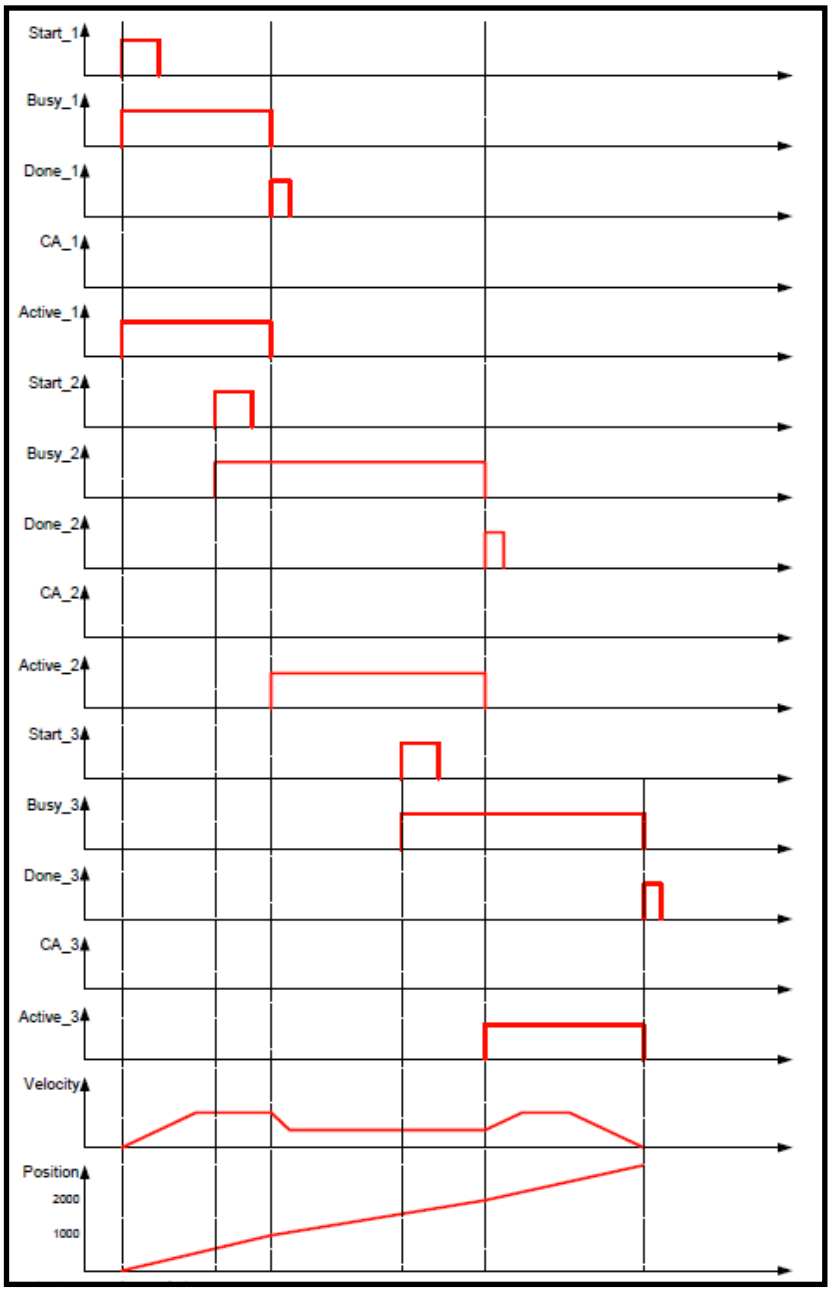


図 3-5-1-5. 上の例の BlendingPrevious mode でのタイムチャート
（API 関数１の最終位置では、API 関数１の速度を使用）

- BlendingNext mode での動作

```
//-----------------------------------------------------------------------
//       BlendingNext Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------
int               ret;
TFB_MVABS_IN      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));
}
```

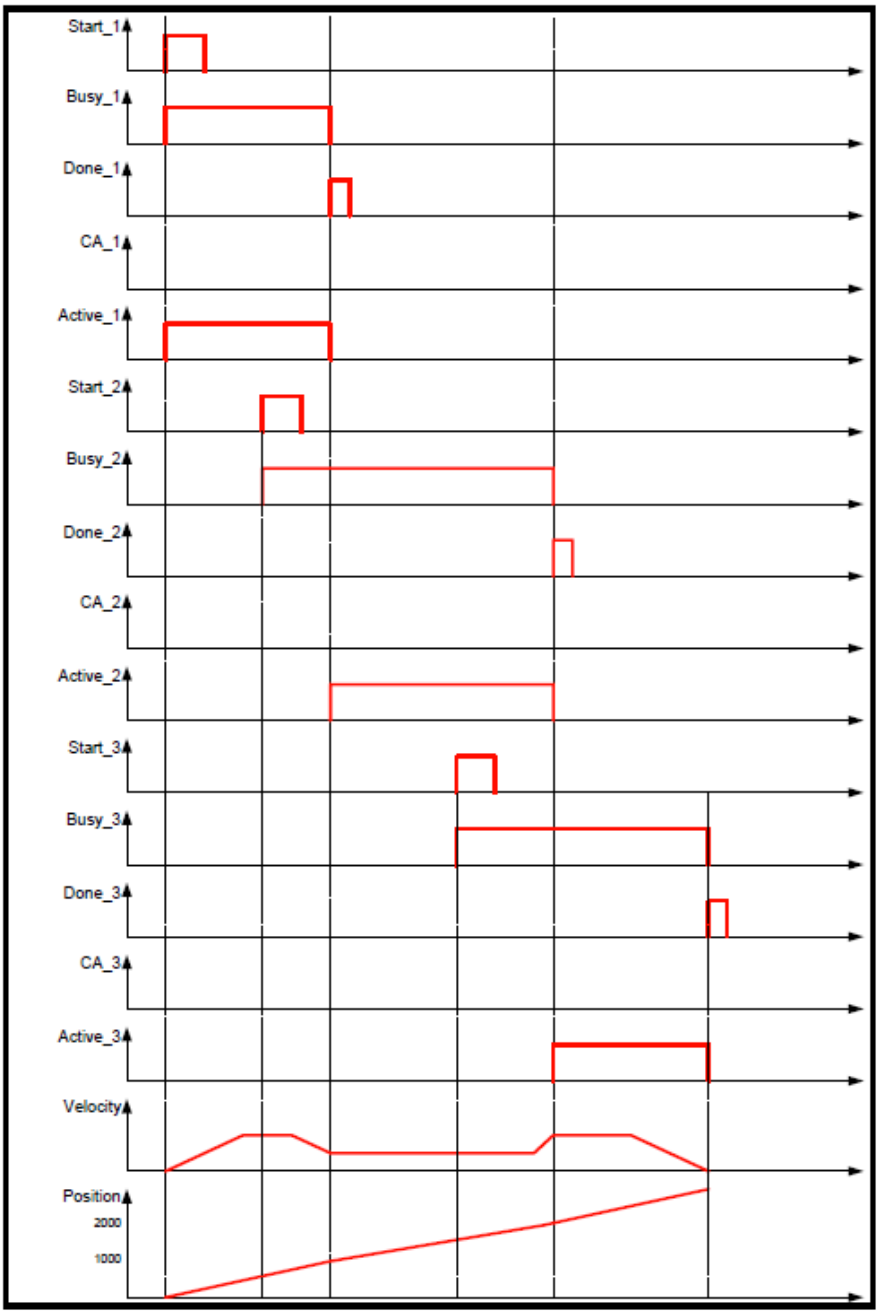


図 3-5-1-6. 上の例の BlendingNext mode でのタイムチャート
（前回の API 関数終了時点では、次の API 関数の速度を使用）

- BlendingHigh mode での動作

```
//-----------------------------------------------------------------------
//       BlendingHigh Mode [絶対値移動](スレーブ ID=1)
//-----------------------------------------------------------------------
int               ret;
TFB_MVABS_IN      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       = 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));
}
```

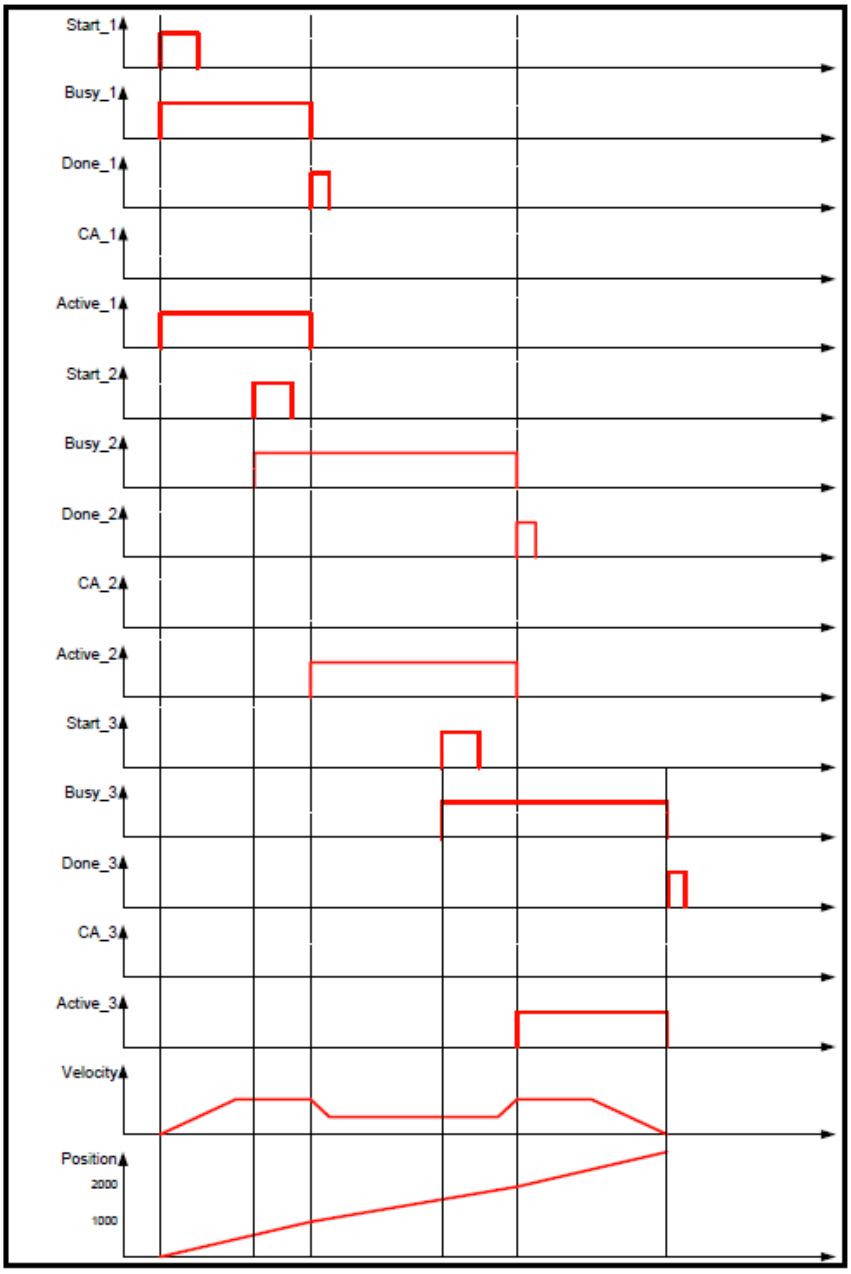


図 3-5-1-7. 上の例の BlendingHigh mode でのタイムチャート

（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。

３－５－２　Part4 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、多重起動を行った際に後から実行した動作の開始タイミングです。

・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」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 3-5-2-1 に各バッファリングモードの一覧を示します。

表 3-5-2-1. 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。 弊社 Part4 仕様ではサポートしていません。
3	BlendingPrevious	次の FB は、以前の FB 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。 合成速度の接続はトランジションモードにより決まります。
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。 弊社 Part4 仕様ではサポートしていません。
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。 弊社 Part4 仕様ではサポートしていません。

また、位置決め動作を伴う API 関数には、BufferMode 適用時の曲線を指定するために「TransitionMode」と「TransitionParameter」という入力を持ちます。
「TransitionParameter」は「TransitionMode」により Parameter の意味が変わります。

表 3-5-2-2. トランジションモード一覧

値	Transition Mode	Transition Parameter	Parameter 内容
0	TMNone	TMNone	TransitionParameter 入力は評価されません。
1	TMStartVelocity	TMStartVelocity	弊社 Part4 仕様ではサポートしていません。
2	TMConstantVelocity	TMConstantVelocity	弊社 Part4 仕様ではサポートしていません。
3	TMCornerDistance	TMCornerDistance	弊社 Part4 仕様ではサポートしていません。
4	TMMaxCornerDeviation	TMMaxCornerDeviation	弊社 Part4 仕様ではサポートしていません。
5～9	PLCopen で予約	－	－
10	TMSmoothing	TMSmoothing	TransitionParameter 入力は評価されません。 内部で自動的に速度合成が実行されます。

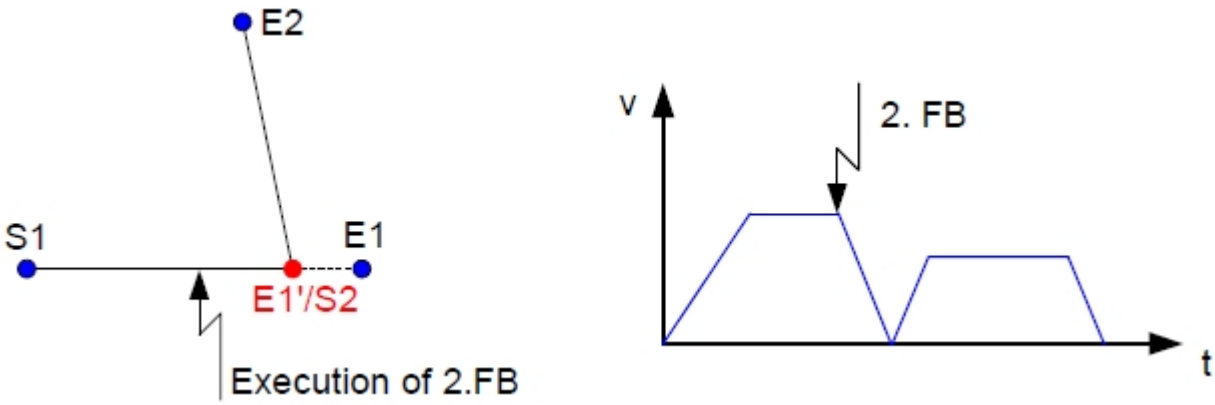
トランジションモードとバッファリングモードの対応表を表 3-4-2-3 に示します。

表 3-5-2-3. トランジションモードとバッファリングモード対応表

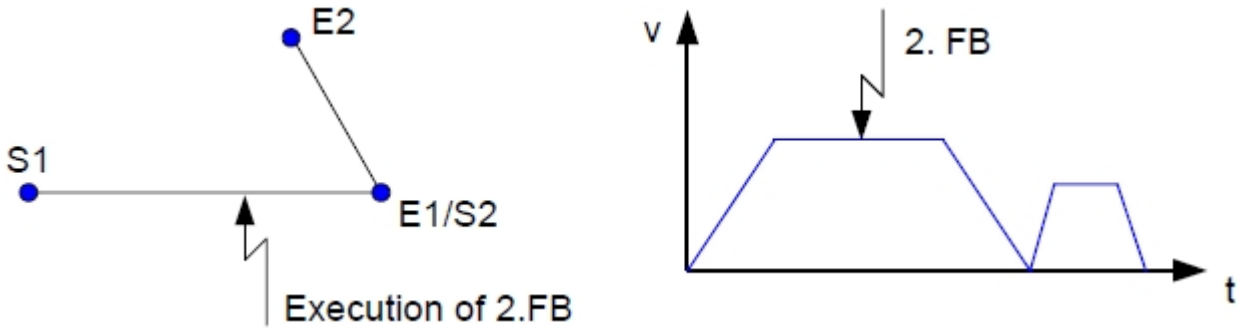
値	TransitionMode	Aborting	Buffered	BlendingPrevious
0	TMNone	○	○	Buffered の動作
10	TMSmoothing	○	Previous の動作	○

以下の例は、バッファリング・トランジションモードの組み合わせによる動作の相違を記述しています。
プログラミングに際しての指定方法等は単軸用のマニュアルを参照してください。

バッファリングモード「Aborting」では、実行中の動作を即座に停止し、新しい動作を開始します。



バッファリングモード「Buffered」では、実行中の動作完了後に新しい動作を開始します。



- Part4 BufferMode “BlendingPrevious”

補間機能の設定により，補間の速度合成によるつなぎ動作を行うことができます。つなぎ機能を有効にすると補間のブロック間で停止せずに速度がつながれるため，サイクルタイムの短縮が可能です。直線補間または円弧補間実行中に補間機能のつなぎ機能を有効に設定した直線補間または円弧補間を指令することで，つなぎ動作が行われます。つなぎ動作が行われた場合，最初の補間の動作完了は速度合成終了時に発行されます。

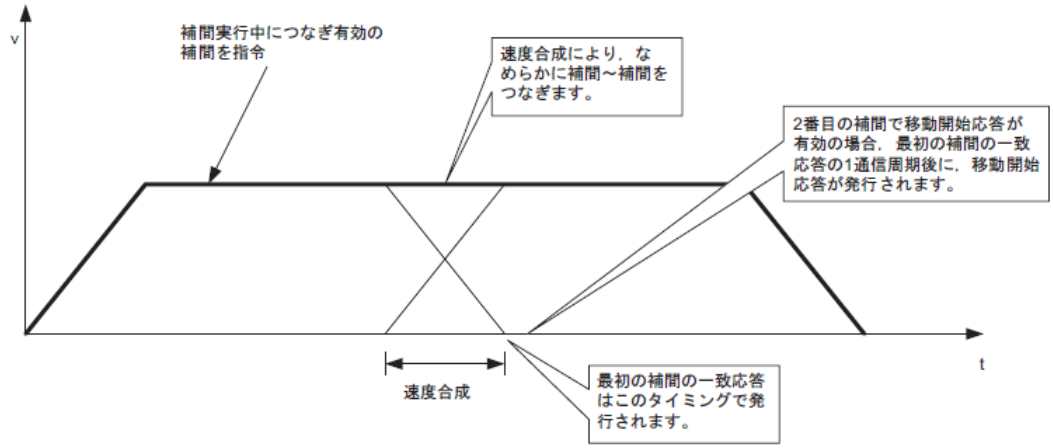


図 3-4-2-3. BlendingPrevious 速度合成図（速度一定、加速度一定）

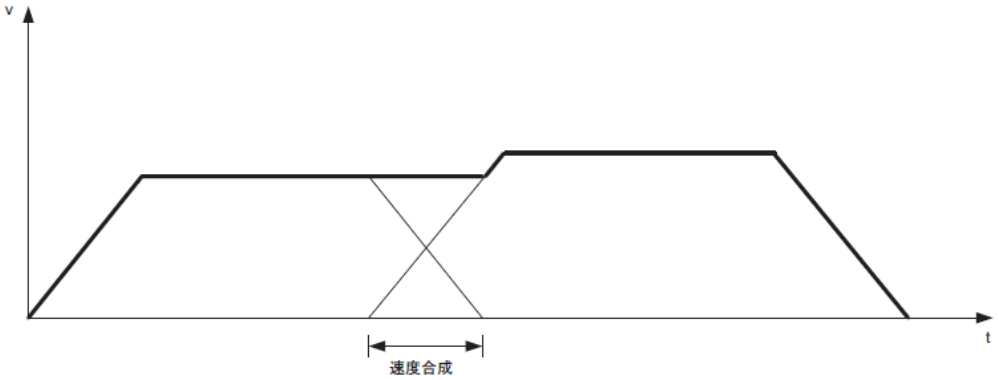


図 3-4-2-4. BlendingPrevious 速度合成図（速度上昇、加速度一定）

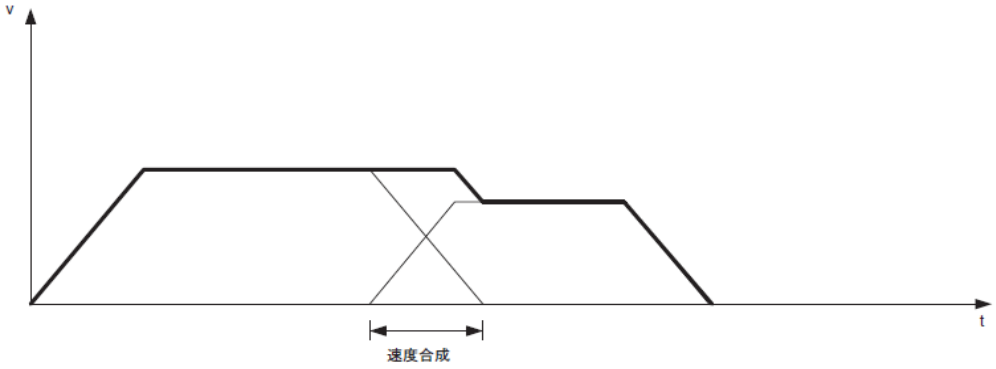


図 3-4-2-5. BlendingPrevious 速度合成図（速度下降、加速度一定）

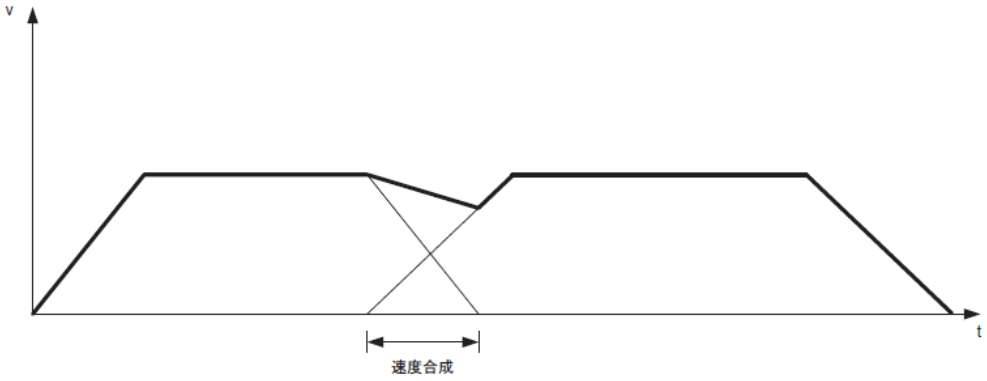


図 3-4-2-6. BlendingPrevious 速度合成図（速度一定、加速度減少）

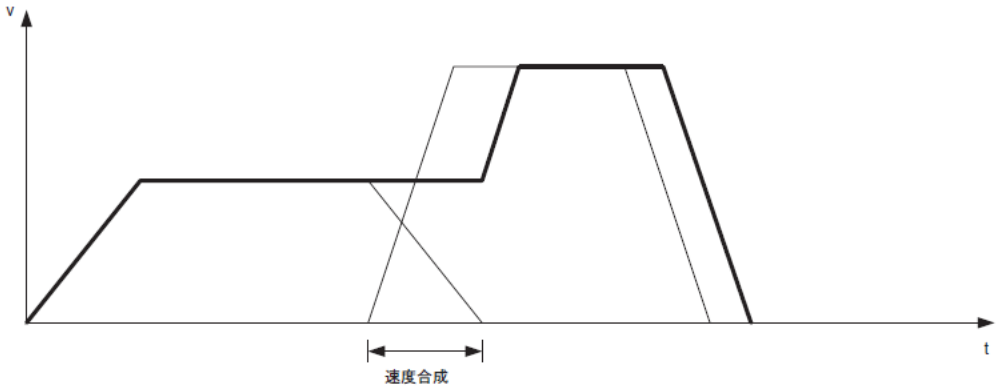


図 3-4-2-7. BlendingPrevious 速度合成図（速度上昇、加速度上昇）

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 FB 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 FB 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・ライトファンクションブロック一覧

FB 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 **※注：AI-Motionでは使用しません。**	範囲	0 or 1	
				初期値	0	
UseGroup	UINT32	0x00000003	制御グループ数	範囲	1～31	
				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 TechnoML3Setting.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 TechnoECTSetting.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

型名	AXIS_REF_CFG		説明	軸基本設定		
メンバ名	型	パラメータNo	説明			
AxisEnable	UINT16	0x00000100	0：この軸は使用しない 1：この軸は使用する		範囲	0 or 1
					初期値	0
AxisType	UINT16	0x00000101	この軸で使用するユニットタイプを指定する		範囲	0x00 ～ 0x20
					初期値	0

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	MECHATROLINK-Ⅲ通信サーボパック
0x11-0x1F	MECHATROLINK-Ⅲ：リザーブ
0x20	仮想軸

NodeAddr	UINT16	0x00000102	EtherCATの場合：TechnoECTSetting.ini設定ファイルで設定した論理ID MECHATROLINK-Ⅲの場合：TechnoML3Setting.ini設定ファイルで設定した論理ID		範囲	ー
					初期値	ー
NodeSubCh	UINT16	0x00000103	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		範囲	0～7
					初期値	0

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>
</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:サーボ準備未完
		1:サーボ準備完了
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	－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。

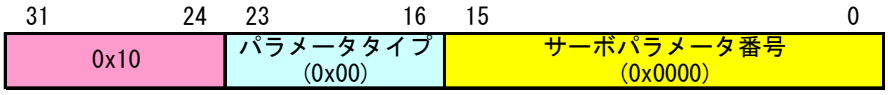


図 4-2-3-1. R/W パラメータ番号

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

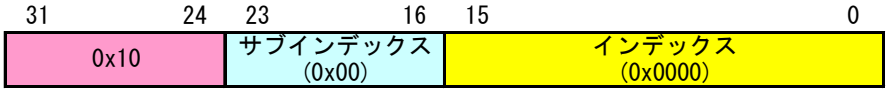
共通パラメータ	RAM 領域	0x00
	不揮発メモリ領域	0x01
機器パラメータ	RAM 領域	0x10
	不揮発メモリ領域	0x11

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

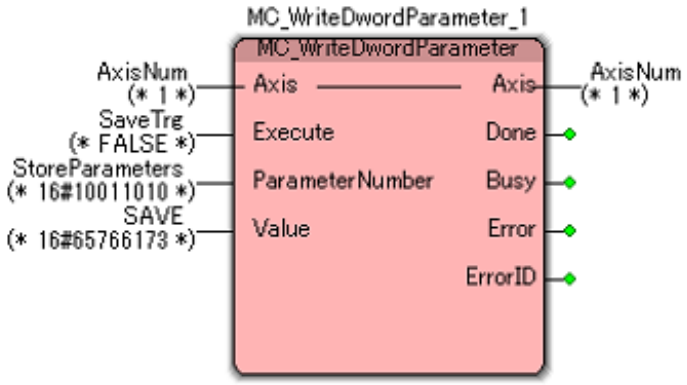
EtherCAT 通信のサーボパックでは、CiA402 で規定されているパラメータ（0x6000 番台）の他に、メーカ独自のパラメータ等が規定されています。

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 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。



● 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 のようになります。

TechnoML3Setting.ini ファイルは「MECHATROLINK-Ⅲ用テクノモーションプロセス RtpIML3Proc.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、MECHATROLINK-Ⅲの通信構成を構築し、各種通信設定を変更して動作させることができます

TechnoECTSetting.ini ファイルは「EtherCAT 用テクノモーションプロセス RtpIECTProc.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、EtherCAT の通信構成を構築し、各種通信設定を変更して動作させることができます

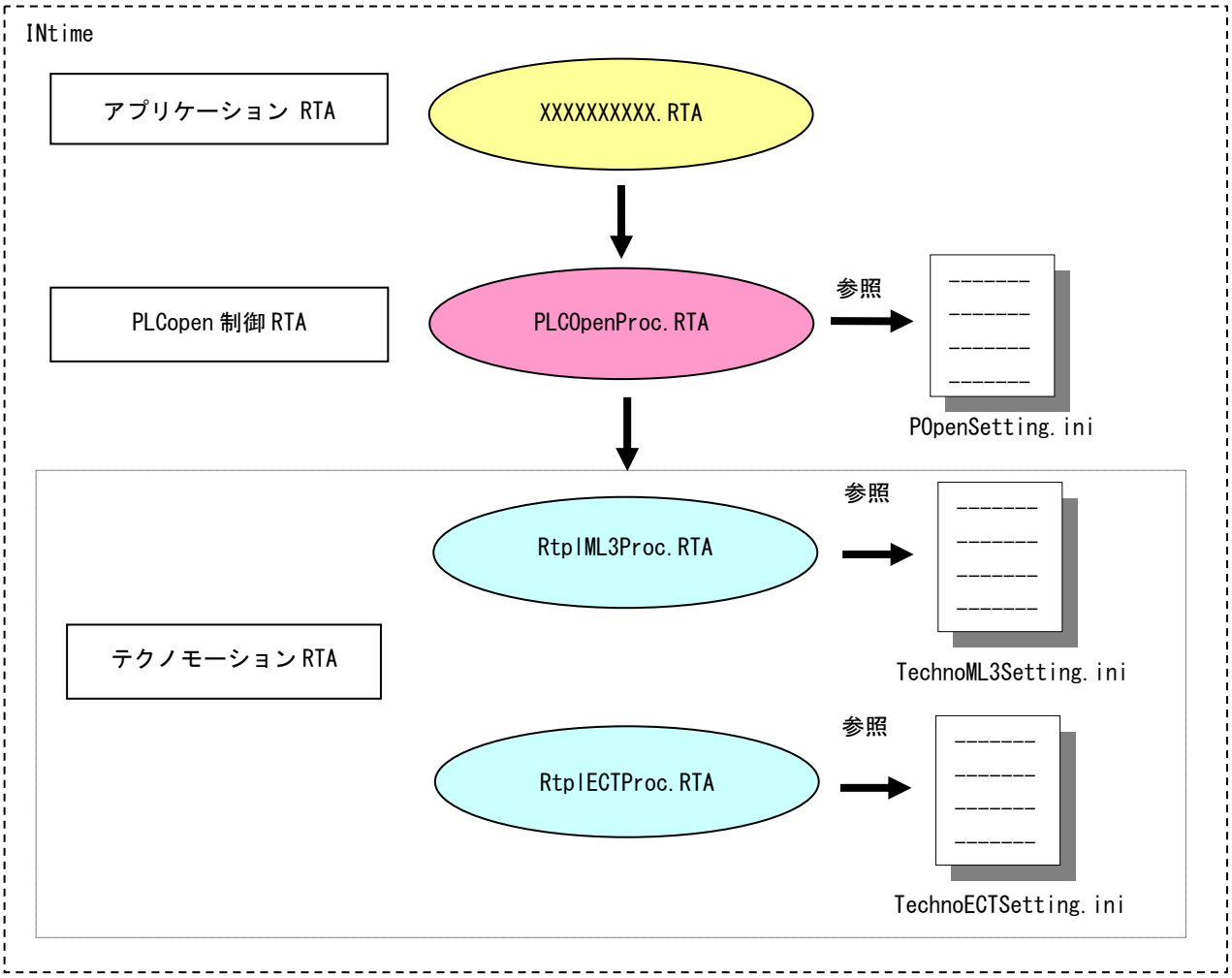


図 4-3-1　INtime 版 PLCopen プロセス構成図

４－３－１　POpenSetting.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。I/O ユニットの制御は I/O ユニット制御用の関数にて直接行いますので、あくまで、制御軸数を設定してください。

設定する項目は表 4-3-1-1 のようになります。

表 4-3-1-1　INI ファイル項目

セクション名	キー名	備　　考
CONTROL	UseAxis	TPOPEN_SMEM_CONTROL パラメータ参照
	EtherCATAddrKind	
	UseGroup	
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 ファイルの例をリスト 4-3-1-1 に示します。

リスト 4-3-1-1　INI ファイル例

```
[CONTROL]
UseAxis=2
UseGroup=1
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。

４－３－２　TechnoML3Setting.ini ファイル

TPAPIM3.RSL は「TechnoML3Setting.ini」ファイルを設定することによって MECHATROLINK-Ⅲ通信設定を行い、スレーブのタイプを設定することができます。
サーボパックだけでなく、I/O ユニットを使用する場合もこちらで設定します。PLCOpen MC として制御するのはサーボパックのみです。I/O ユニットについては、I/O ユニット用の関数により直接制御します。
設定する項目は表 4-3-2-1、表 4-3-2-2 のようになります。

表 4-3-2-1　INI ファイル項目（マスタ設定）

セクション名	キー名	タイプ	備　　考
SYSTEM	CommMode	16 進数	これらの設定値の詳細については、下記を参照してください。
	MaxSlave	10 進数	
	CycleTime	10 進数	
	IntOffset	10 進数	
	C2MstDly	10 進数	
	CommProtocol	10 進数	
	MaxRetry	10 進数	
	HostWdt	10 進数	

CommMode　：通信モード設定　（初期値：0x8002）

CommMode（論理和）	内容
SYS_MOD_TYPE_C1MST *1	0x00000002：動作タイプ C1MST
SYS_MOD_INTLV_PLS	0x00000800：INTOL 信号をパルス出力設定
SYS_MOD_ESYNC	0x00001000：RTCIL 信号入力に同期
SYS_MOD_INT_FR　*1	0x00008000：ハード同期有効

*1 必ず指定する必要があります。

MaxSlave　：最大接続スレーブ数　（初期値：1）
［ 1 ～ 62 ］　単位：【局】

CycleTime　：伝送周期　（初期値：50000）
［ 3125(31.25us) ～ 6400000(64ms) ］ 単位：【10ns】

IntOffset　：割込遅延時間　（初期値：25000）
［ 0 ～ 伝送周期設定値－500(5us) ］ 単位：【10ns】

C2MstDly　：C2 マスタ送信開始時間　（初期値：0）
［ 0：C2 マスタ使用しない］
［ 1 ～ 伝送周期設定値－500(5us) ］ 単位：【10ns】

CommProtocol　：通信プロトコル選択　（初期値：0）
［ 0:サイクリック通信］
［ 1:イベントドリブン通信］

MaxRetry　：最大リトライ回数　（初期値：1）
［ 0 ～ 62 ］　単位：【回】

HostWdt　：ホスト監視用 WDT 設定　（初期値：16384）
［ 0：機能無効］

[1(8us) ～ 16384(131072us)]　　単位：【8us】

表 4-3-2-2　INI ファイル項目（スレーブ設定）

セクション名	キー名	タイプ	備考
SLAVE_n （n = 1 ～ 62）	SlaveID	10 進数	これらの設定値の詳細については、下記を参照してください。
	Cd_Rd_Len	10 進数	
	ResponseTime	10 進数	
	Com_Mod	16 進数	
	Com_Tim	10 進数	
	Profile_Type	16 進数	

SlaveID　：局アドレス　（初期値：3）
[0：スレーブなし]
拡張アドレス（上位バイト）[0x00 ～ 0x3D]
局アドレス　（下位バイト）[0x03 ～ 0xEF]

Cd_Rd_Len　：送受信データ長　（初期値：48）
[8/16/32/48/64]　　単位【Byte】

ResponseTime　：応答監視時間　（初期値：5000）
[500(5us) ～ 伝送周期設定値]　　単位【10ns】

Com_Mod　：通信モード　（初期値：0x82）

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
SUBCMD	0	0	0	0	0	SYNCMODE	0

SYNCMODE:同期設定　[0:非同期通信 1:同期通信]
SUBCMD:サブコマンド設定　[0:サブコマンド無効 1:サブコマンド有効]

Com_Tim　：通信周期（通信周期 = 設定値×伝送周期）　（初期値：1）
[1 ～ 255]　　単位【倍】

Profile_Type　：プロファイルタイプ

Profile_Type	内容
SLV_PROFILETYPE_SRV	0x10：サーボパックプロファイルタイプ（初期値）
SLV_PROFILETYPE_IO	0x30：I/O プロファイルタイプ

TechnoML3Setting.ini ファイルの例をリスト 4-3-2-1 に示します。

リスト 4-3-2-1　INI ファイル例

```
[SYSTEM]
CommMode=8002h
MaxSlave=2
CycleTime=50000
IntOffset=25000
C2MstDly=0
CommProtocol=0
MaxRetry=1
HostWdt=16384

[SLAVE_1]
SlaveID=3
Cd_Rd_Len=48
ResponseTime=5000
Com_Mod=82h
Com_Tim=1
Profile_Type=10h

[SLAVE_2]
SlaveID=4
Cd_Rd_Len=48
ResponseTime=5000
Com_Mod=82h
Com_Tim=1
Profile_Type=10h
```

MECHATROLINK-Ⅲ通信設定

h が付いている設定値は 16 進数で設定

各スレーブの通信設定
接続するスレーブ毎に用意

各スレーブの通信設定
接続するスレーブ毎に用意

４－３－３　TechnoECTSetting.ini ファイル

RTPLECTPROC.RSL は「TechnoECTSetting.ini」ファイルを設定することによって EtherCAT 通信設定を行い、スレーブのタイプを設定することができます。
サーボパックだけでなく、I/O ユニットを使用する場合もこちらで設定します。PLCOpen MC として制御するのはサーボパックのみです。I/O ユニットについては、ProConOS により直接制御します。
設定する項目は表 4-3-3-1、表 4-3-3-2 のようになります。

表 4-3-3-1　INI ファイル項目（マスタ設定）

セクション名	キー名	タイプ	備　　考
SYSTEM	MaxSlave	10 進数	これらの設定値の詳細については、下記を参照してください。

MaxSlave　　：最大接続スレーブ数　　（初期値：1）
［ 1 ～ 62 ］　　単位：【局】

表 4-3-3-2　INI ファイル項目（スレーブ設定）

セクション名	キー名	タイプ	備考
SLAVE_n （n = 1 ～ 62）	StationAlias	10 進数	これらの設定値の詳細については、下記を参照してください。
	SlaveType	10 進数	

StationAlias　：局アドレス　　（初期値：1）
［ 0：スレーブなし ］
［ 0x01 ～ 0xEF ］

SlaveType　　：スレーブタイプ　　（初期値：0）
［ 0：CiA402 準拠サーボパック］
［ 0 以外：その他の EtherCAT スレーブ］

TechnoECTSetting.ini ファイルの例をリスト 4-3-3-1 に示します。

リスト 4-3-3-1　INI ファイル例

```
[SYSTEM]
MaxSlave=3

[SLAVE_1]
StationAlias=1
SlaveType=0

[SLAVE_2]
StationAlias=2
SlaveType=0

[SLAVE_3]
StationAlias=3
SlaveType=1
```

EtherCAT 通信設定

各スレーブの通信設定
接続するスレーブ毎に用意（サーボ）

各スレーブの通信設定
接続するスレーブ毎に用意（サーボ）

各スレーブの通信設定
接続するスレーブ毎に用意（I/O）

PLCopen から制御できるのは、サーボパックだけです。

４－４　　エラーコード一覧

ファンクションブロック出力の 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　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。

４－４－２　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	予約。本エラーは発生しません。
0x30000102	MC_Stop エラー	予約。本エラーは発生しません。
0x30000103	MC_Home エラー	予約。本エラーは発生しません。
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	FB 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 FB の動作方向と、多重起動された FB 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。
0x30000801	グループへの軸登録未完	補間移動グループに軸が登録されていません。
0x30000802	単軸動作中	補間移動グループに登録しようとしている軸が単軸で動作しています。
0x30000803	単軸エラー発生中	補間移動グループに登録しようとしている軸がエラー状態です。
0x30000804	グループ登録軸数オーバ	グループに登録できる軸数を越えています。直線補間時は 16 軸まで、円弧補間時は 2 軸まで登録できます。

４－４－３　FB 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	FB 呼び出しが仕様範囲外の状態で実行された
0x40000003	FB 入力値異常	FB 入力値が異常です。
0x40000004	同時読み込み	パラメータ読み込み FB 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 FB が同時に複数実行された
0x40000801	グループへ登録済み	すでにグループに登録されています。

４－４－４　MECHATROLINK-Ⅲコマンド異常

エラー番号	エラー名	内容
0x5000XXXX	テクノモーションライブラリ 内部エラー	テクノモーションライブラリ内部で発生したエラーです。 下位 16bit がテクノモーションライブラリ側のエラー番号となります。 エラー番号は「4-4-5 テクノモーションライブラリエラー番号」を参照してください。
0x5001XXXX	MECHATROLINK-Ⅲワーニング	MECHATROLINK-Ⅲ通信コマンドでワーニングが発生しています。 下位 16bit が MECHATORLINK-Ⅲ通信レスポンスデータの CMD_STAT の情報です。

４－４－５　テクノモーションライブラリエラー

● MECHATROLINK-Ⅲ

エラー番号	エラー名	内容
1	引数エラー	
2	共通データ不正エラー	
3	MECHATROLINK-Ⅲデータ不正エラー	
100	プロテクト認証エラー	
101	ライブラリ初期化済	
102	ライブラリ未初期化	
103	スレッドリソース初期化済	
104	スレッドリソース未初期化	
105	スレッド数フル	
106	マスタ初期化済	
107	マスタ未初期化	
108	スレーブ未接続	
109	プロファイル違いエラー	
110	サーボ OFF エラー	
111	指令有りエラー	
112	ガントリ軸設定済	
113	ガントリ軸未設定	
114	ガントリスレーブ軸への指令	
115	ガントリサポート対象外指令	
116	レスポンスフルエラー	
200	バッファフルエラー	
201	バッファ空エラー	
202	コマンド違いエラー	
203	稼働済移動グループ登録エラー	
204	登録済移動グループ登録エラー	
205	バッファグループ即実行コマンド登録エラー	
206	応答待ち時タイムアウトエラー	
300	非移動中エラー	
301	一時停止無視エラー	
302	加減速フィルターバッファフル	
303	加速時定数オーバーエラー	
304	減速時定数オーバーエラー	
305	S 字加減速時定数オーバーエラー	
306	円弧指令回転方向違いエラー	
307	円弧半径オーバーフローエラー	
308	円弧中心点違いエラー	
309	円弧中心点計算エラー	
400	サーボ主電源 OFF	
401	＋方向ソフトリミット	
402	－方向ソフトリミット	
403	＋方向オーバートラベル	
404	－方向オーバートラベル	
500	ML コマンドタイムアウトエラー	
501	ML 通信エラー	
502	ML WDT エラー	

503	ML データ受信無しエラー	
504	ML アラーム	
505	ML ワーニング	
1000	INtime 未起動	
1001	INtime プロセス無し	
1002	位置決め LIB 未起動	
1003	パラメータエラー	
1004	空き通信ハンドル無し	
1005	無効通信ハンドル	
1006	多重ハンドル	
1007	通信ビジー	
1008	通信データサイズエラー	
1100	通信デバイス未初期化	
1101	通信タイムアウト	
1102	リトライオーバー	
1103	多重リトライ	
1104	通信シーケンスエラー	
1999	通信中不明エラー	
9999	不明エラー	
-1	その他のエラー	

● EtherCAT

エラー番号	エラー名	内容
1	引数エラー	
2	共通データ不正エラー	
3	EtherCAT 設定データ不正エラー	
100	プロテクト認証エラー	
101	ライブラリ初期化済	
102	ライブラリ未初期化	
103	スレッドリソース初期化済	
104	スレッドリソース未初期化	
105	スレッド数フル	
106	マスタ初期化済	
107	マスタ未初期化	
108	スレーブ未接続	
109	プロファイル違いエラー	
110	サーボ OFF エラー	
111	指令有りエラー	
112	ガントリ軸設定済	
113	ガントリ軸未設定	
114	ガントリスレーブ軸への指令	
115	ガントリサポート対象外指令	
116	レスポンスフルエラー	
200	バッファフルエラー	
201	バッファ空エラー	
202	コマンド違いエラー	
203	稼働済移動グループ登録エラー	
204	登録済移動グループ登録エラー	
205	バッファグループ即実行コマンド登録エラー	
206	応答待ち時タイムアウトエラー	
300	非移動中エラー	
301	一時停止無視エラー	
302	加減速フィルターバッファフル	
303	加速時定数オーバーエラー	
304	減速時定数オーバーエラー	
305	S 字加減速時定数オーバーエラー	
306	円弧指令回転方向違いエラー	
307	円弧半径オーバーフローエラー	
308	円弧中心点違いエラー	
309	円弧中心点計算エラー	
310	位置制御不可エラー	
311	ラッチ方法選択不可エラー	
312	トルク制御不可エラー	
400	サーボ主電源 OFF	
401	＋方向ソフトリミット	
402	－方向ソフトリミット	
403	＋方向オーバートラベル	
404	－方向オーバートラベル	
600	ENI ファイルとスレーブ情報が異なる	

601	未対応動作ステータスエラー	
602	未対応動作モード設定エラー	
603	未対応 PDO の呼び出しエラー	
604	モード変更失敗エラー	
605	コマンドタイムアウト	
606	アラーム発生	
607	ワーニング発生	
608	即停止中指令エラー	
609	マスタスタックエラー	
610	原点復帰エラー	
1000	INtime 未起動	
1001	INtime プロセス無し	
1002	位置決め LIB 未起動	
1003	パラメータエラー	
1004	空き通信ハンドル無し	
1005	無効通信ハンドル	
1006	多重ハンドル	
1007	通信ビジー	
1008	通信データサイズエラー	
1100	通信デバイス未初期化	
1101	通信タイムアウト	
1102	リトライオーバー	
1103	多重リトライ	
1104	通信シーケンスエラー	
1999	通信中不明エラー	
9999	不明エラー	
-1	その他のエラー	

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

⑴ 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。
⑵ 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。
⑶ 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。

７７ＰＯ１００００５Ａ　　２０１３年　９月　初版
７７ＰＯ１００００５Ｃ　　２０１４年　３月　第３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