株式会社ワイツー

PMC-Mシリーズ　ソフトウェアマニュアル

このマニュアルについて

このソフトウェアマニュアルにはソフトウェアに関する情報が記載されています。
取扱説明書（ハードウェアのマニュアル）も併せてお読み下さい。

ソフトウェアについて

本ソフトウェアは弊社モーター制御ボード、PMC-M2C-Uを制御する為のソフトウェアです。
モーターの制御は、提供されるDLLの関数をコールすることで実現できますので、開発者はUSB接続であるという事を意識せずに使用することができます。

用語説明

ID

ボード識別スイッチにて設定された値
（設定方法については取扱説明書を参照してください）

基本仕様

接続台数

1台のパソコンから制御できるボードの最大数は16台です。

ハードウェア仕様

ハードウェア仕様については取扱説明書を参照してください。

注意事項

パソコンがスタンバイや休止状態とならないようにOSを設定してください。
スタンバイや休止状態になると、以降ボードが動作しません。

動作環境（Windows）

PC

IBM PC/AT互換機（DOS/V機）

OS

Windows 11 x64
Windows 10 x86, x64
Windows 10 IoT Enterprise[1]
Windows 8.1 x86, x64
Windows 8 x86, x64[2]
Windows 7 x86, x64[2]
Windows Vista x86, x64[3]
Windows XP x64[4]
Windows XP[4]
Windows 2000[4]
Windows Me[5]
Windows 98, 98SE[5]

対応言語

Microsoft Visual C++（6.0、.NET2002以降）
Microsoft Visual C#
Microsoft Visual Basic（6.0、.NET2002以降）
VBA
Python3
その他、Win32API関数をサポートしているプログラミング言語

---

1. Windows 10 IoT Enterprise以外のWindows 10 IoTでは使用できません ↩
2. 2015年10月20日リリースの旧バージョンのドライバを使用します ↩↩
3. 2014年2月28日リリースの旧バージョンのドライバを使用します ↩
4. 2012年6月29日リリースの旧バージョンのドライバを使用します ↩↩↩
5. 2009年10月26日リリースの旧バージョンのドライバを使用します ↩↩

動作環境（Linux）

PC

IBM PC/AT互換機（DOS/V機）

シングルボードコンピュータ

Raspberry Pi 5/4/3
Jetson Nano

OS

PC

Ubuntu Desktop 24.04 LTS
Ubuntu Desktop 22.04 LTS
Ubuntu Desktop 20.04 LTS
Ubuntu Desktop 18.04 LTS
Debian 12（amd64, i386）
Debian 11（amd64, i386）
Debian 10（amd64, i386）
Debian 9（amd64, i386）
CentOS 8（x86_64）

Raspberry Pi 5

Raspberry Pi OS bookworm（64-bit, 32-bit）
Ubuntu Server 24.04 LTS
Ubuntu Desktop 24.04 LTS

Raspberry Pi 4

Raspberry Pi OS bookworm（64-bit, 32-bit）
Raspberry Pi OS bullseye（64-bit, 32-bit）
Raspberry Pi OS（Raspbian） buster
Ubuntu Server 24.04 LTS
Ubuntu Server 22.04 LTS（arm64, armhf）
Ubuntu Server 20.04 LTS（arm64, armhf）
Ubuntu Server 18.04 LTS（arm64, armhf）
Ubuntu Desktop 24.04 LTS
Ubuntu Desktop 22.04 LTS

Raspberry Pi 3

Raspberry Pi OS bullseye（64-bit, 32-bit）
Raspberry Pi OS（Raspbian） buster

Raspberry Pi OS（Raspbian） stretch
Ubuntu Server 22.04 LTS（arm64, armhf）
Ubuntu Server 20.04 LTS（arm64, armhf）
Ubuntu Server 18.04 LTS（arm64, armhf）

Jetson Nano

Jetson Nano Developer Kit SD Card Image

対応言語

GCC6+
Python3

必要なソフトウェアパッケージ

libusb-1.0
GCC6+

機能一覧

- エンドリミット信号（+EL、-EL）
- スローダウン信号（SD）
- 原点信号（ORG）、マーカ信号（Z）
- アラーム信号（ALM）
- 位置決め完了信号（INP）
- カウンタラッチ信号（LTC）
- 位置決め開始信号（PCS）
- 偏差カウンタクリア信号（ERC）
- 非常停止信号（EMG）
- パルス出力
- カウンタ
- コンパレータ
- イベント

エンドリミット信号（+EL、-EL）

概要

動作中に動作方向のEL信号がONすると、即停止／減速停止します。
動作開始時に動作方向のEL信号がON状態の場合は動作できません。

設定項目

- 入力論理
  PmcmSetSensorConfig関数で入力論理を選択できます。
- 停止方法
  PmcmSetSensorConfig関数で即停止／減速停止を選択できます。
- フィルタ
  PmcmSetSensorConfig関数でフィルタを挿入することができます。

備考

入力論理の初期値はオフで検知（正論理）です。
EL信号を接続していない場合やオンで検知（負論理）するリミットスイッチを接続した場合は動作しません。
その場合、入力論理をオンで検知（負論理）に変更してください。

> ⚠ EL信号を接続していない場合は機械系の衝突にご注意ください

減速停止を選択した場合でも、起動方法を定速起動でモーターを動作させた場合は即停止します。
停止方法を減速停止とした場合には、停止するまでEL入力をON状態に保持してください。

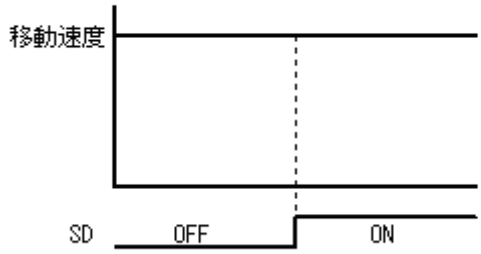
スローダウン信号（SD）

概要

モーター動作中にSD信号がONすると、減速方法と入力方法の設定により減速動作をおこないます。

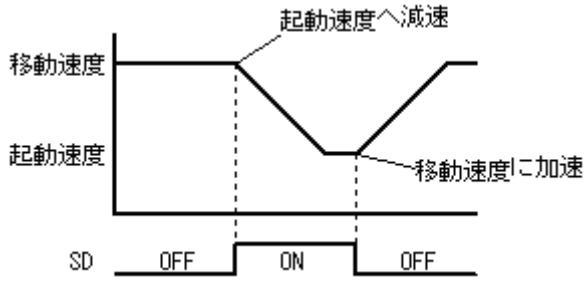
減速

- 減速方法 : 減速
- 入力方法 : レベル入力
- 起動モード :
  - 定速起動



定速起動ではSD信号を無視します。

  - 定速－減速起動、加減速起動

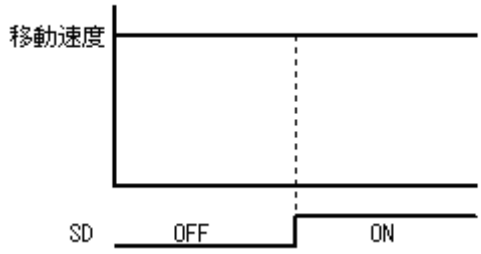


定速－減速起動・加減速起動ではSD信号ONにより起動速度まで減速します。
減速後、または減速中にSD信号がOFFになると移動速度まで加速します。
定速－減速起動・加減速起動時での動作開始時にSD信号がONしている場合は起動速度で動作しますが、SD信号がOFFになると移動速度まで加速します。

ラッチ・減速

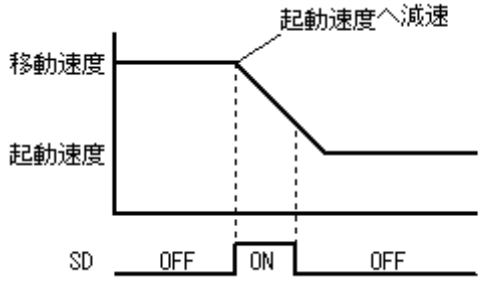
- 減速方法 : 減速
- 入力方法 : ラッチ入力
- 起動モード :

- 定速起動



定速起動ではSD信号を無視します。

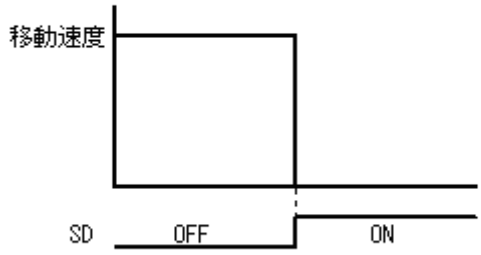
- 定速－減速起動、加減速起動



定速－減速起動・加減速起動ではSD信号ONにより起動速度まで減速します。
減速後、または減速中にSD信号がOFFになっても起動速度を維持し、移動速度に加速しません。
定速－減速起動・加減速起動時での動作開始時にSD信号がONしている場合は起動速度で動作しますが、SD信号がOFFになっても移動速度に加速はしません。

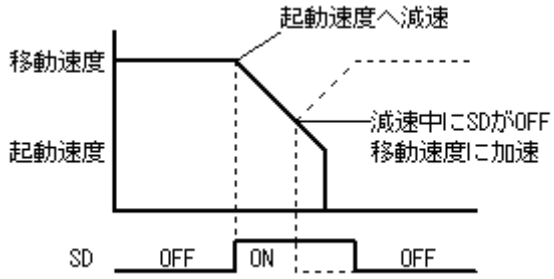
減速停止

- 減速方法 : 減速停止
- 入力方法 : レベル入力
- 起動モード :
  - 定速起動



定速起動ではSD信号がONすると停止します。
動作開始時にSD信号がONしている場合は、動作を開始せず動作完了になります。

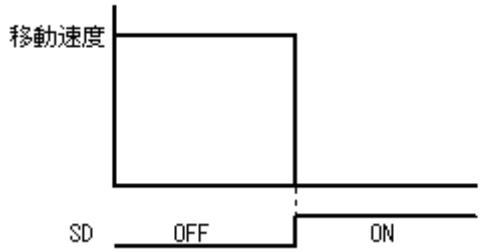
- 定速－減速起動、加減速起動



定速－減速起動・加減速起動ではSD信号ONにより、起動速度まで減速後停止します。
減速中にSD信号がOFFになると移動速度まで加速します。
動作開始時にSD信号がONしている場合は、動作を開始せず動作完了になります。

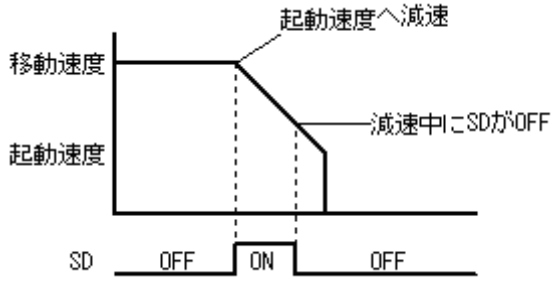
ラッチ・減速停止

- 減速方法 : 減速停止
- 入力方法 : ラッチ入力
- 起動モード :
  - 定速起動



定速起動ではSD信号がONすると停止します。
動作開始時にSD信号がONしている場合は、動作を開始せず動作完了になります。

- 定速－減速起動、加減速起動



定速－減速起動・加減速起動ではSD信号ONにより、起動速度まで減速後停止します。
減速中にSD信号がOFFになっても加速しません。
動作開始時にSD信号がONしている場合は、動作を開始せず動作完了になります。

設定項目

- 入力論理
  PmcmSetSensorConfig関数で入力論理を選択できます。
- 減速方法
  PmcmSetSensorConfig関数で減速／減速停止を選択できます。

- 入力方法
  PmcmSetSensorConfig関数でレベル入力／ラッチ入力を選択できます。
- フィルタ
  PmcmSetSensorConfig関数でフィルタを挿入することができます。

備考

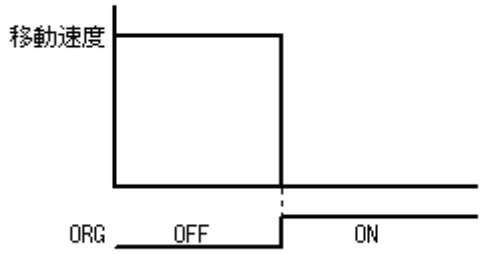
入力方法にラッチ入力を選択した場合、ラッチの解除は、次の動作開始時にSD信号がOFFであればラッチはリセットされます。
また、レベル入力を選択してもリセットされます。

原点信号（ORG）、マーカ信号（Z）

概要

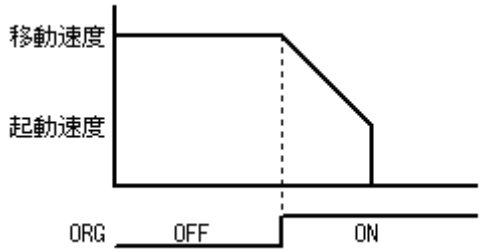
ORG信号のみによる原点復帰動作

- 原点復帰方法 : ORG信号のみ使用
- 起動モード :
  - 定速起動



ORG入力OFF→ONで即停止します。

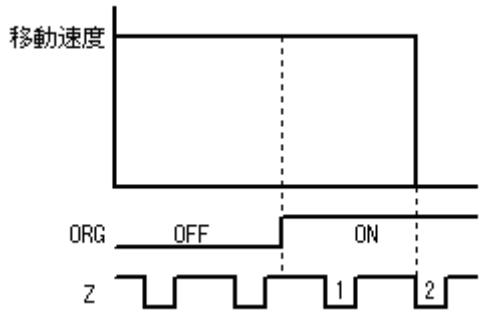
  - 定速－減速起動、加減速起動



ORG入力OFF→ONで減速を開始し、起動速度まで減速すると停止します。
ORG入力前に、SD入力により起動速度まで減速完了していたときには、ORG入力OFF→ONにより即停止します。

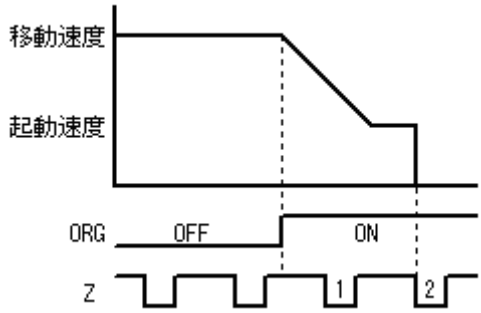
ORG信号とZ信号による原点復帰動作（動作の例はZ信号カウント数が2の場合）

- 原点復帰方法 : ORG信号とZ信号を使用
- 起動モード :
  - 定速起動



ORG入力OFF→ON後に指定回数のZ信号入力で即停止します。

- 定速－減速起動、加減速起動



ORG入力OFF→ONで減速を開始し、指定回数のZ信号入力で停止します。
ORG入力前に、SD入力により起動速度まで減速完了していたときには、ORG入力OFF→ON後の指定回数のZ信号入力により即停止します。

設定項目

- ORG信号入力論理
  PmcmSetSensorConfig関数でORG信号の入力論理を選択できます。
- 原点復帰方法
  PmcmSetSensorConfig関数で原点復帰方法を選択できます。
- Z信号入力論理
  PmcmSetSensorConfig関数でZ信号の入力論理を選択できます。
- Z信号カウント数
  PmcmSetSensorConfig関数でZ信号のカウント数を設定できます。
- フィルタ
  PmcmSetSensorConfig関数でフィルタを挿入することができます。

アラーム信号（ALM）

概要

動作中にALM信号がONになると即停止／減速停止します。
動作開始時にALM信号がON状態の場合はパルスを出力しません。

設定項目

- 入力論理
  PmcmSetSensorConfig関数で入力論理を選択できます。
- 停止方法
  PmcmSetSensorConfig関数で即停止／減速停止を選択できます。
- フィルタ
  PmcmSetSensorConfig関数でフィルタを挿入することができます。

備考

停止方法を減速停止とした場合には、停止するまでALM入力をON状態に保持してください。
停止方法を減速停止とした場合でも、定速起動にて動作させた場合は即停止します。

位置決め完了信号（INP）

概要

パルス列入力タイプのサーボドライバには、内部に指令パルス入力とフィードバックパルス入力との差をカウントする偏差カウンタがあり、その差が0になるようにモーターを制御します。 つまり、サーボドライバは指令パルスよりも遅れて動作し、指令パルスを停止してもドライバ内の偏差カウンタが0になるまで停止が遅れます。

動作完了判断をサーボドライバからの位置決め完了信号（INP信号）の入力時とすることができます。

設定項目

- 入力論理
  PmcmSetSensorConfig関数で入力論理を選択できます。
- フィルタ
  PmcmSetSensorConfig関数でフィルタを挿入することができます。

備考

パルス出力完了時に、既にINP信号ONの場合には、遅延なしで動作完了になります。

カウンタラッチ信号（LTC）

概要

出力パルスカウンタ、エンコーダカウンタをLTC信号入力によりラッチすることができます。

設定項目

- 入力論理
  PmcmSetSensorConfig関数で入力論理を選択できます。

位置決め開始信号（PCS）

PCS信号は位置のオーバーライド信号、または外部スタート信号として使用します。

概要

- 位置のオーバーライド
  動作開始後、PCS信号がONのタイミングから、移動パルス数に設定した移動量分の位置決め動作をおこなうことができます。
- 外部スタート
  動作開始要求後、PCS信号がONのタイミングから、実際に動作を開始することができます。

設定項目

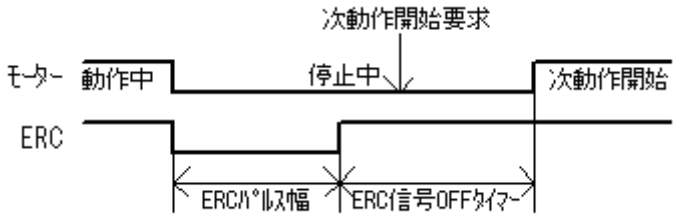
- 入力論理
  PmcmSetSensorConfig関数で入力論理を選択できます。

偏差カウンタクリア信号（ERC）

概要

サーボモーターは指令パルスを停止してもサーボドライバ内の偏差カウンタが0になるまで停止が遅れます。
原点復帰完了時などにサーボモーターを即停止させるためには、サーボドライバ内の偏差カウンタをクリアする必要があります。

ERC信号で、サーボドライバ内の偏差カウンタをクリアするための信号を出力することができます。
サーボドライバでERC信号がOFFに復帰してから指令パルスを受け付けるまでに時間を必要とする場合はOFFタイマー時間を選択できます。



設定項目

- 出力論理
  PmcmSetErcConfig関数で出力論理を選択できます。
- 出力幅
  PmcmSetErcConfig関数でERC信号の出力パルス幅を選択できます。
- OFFタイマー
  PmcmSetErcConfig関数でOFFタイマー時間を選択できます。

備考

EL、ALM、EMG入力停止時にERC信号を出力する設定であっても、減速停止の場合はERC信号を出力しません。

非常停止信号（EMG）

概要

動作中にEMG入力がONになると、全軸即停止となります。

設定項目

なし

備考

通常の停止動作では最終パルス幅が確保されますが、非常停止動作では最終パルス幅が確保されずにヒゲ状になる場合があります。
ヒゲ状になった時には、モータードライバでは受け付けられずに、内部カウンタだけはカウントすることが考えられます（位置管理のずれ）。
そのため、非常停止後には原点復帰動作をおこなって、内部位置（カウンタ）と機械位置とを一致させる必要があります。

パルス出力

パルス出力モード

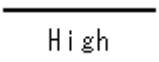
パルス出力モードとして、共通パルスモード4種類と2パルスモード2種類、90度位相差モード2種類が選択できます。

- 共通パルスモード
  OUT端子から動作用パルスを出力し、DIR端子から方向判別信号を出力するモード（0〜3）
- 2パルスモード
  OUT端子から+方向動作用パルスを出力し、DIR端子から-方向動作用パルスを出力するモード（4,7）
- 90度位相差モード（4逓倍）
  OUT端子とDIR端子から、90度位相差信号を出力するモード（5,6）

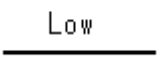
0

共通パルスモード
OUT：負論理
DIR：（＋方向）High、（―方向）Low

CW（＋）方向動作時

OUT出力	DIR出力
	

CCW（―）方向動作時

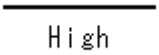
OUT出力	DIR出力
	

1

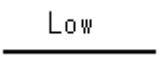
共通パルスモード
OUT：正論理
DIR：（＋方向）High、（―方向）Low

CW（＋）方向動作時

OUT出力	DIR出力

OUT出力	DIR出力
	

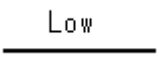
CCW（—）方向動作時

OUT出力	DIR出力
	

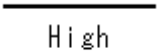
2

共通パルスモード
OUT：負論理
DIR：（＋方向）Low、（—方向）High

CW（＋）方向動作時

OUT出力	DIR出力
	

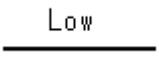
CCW（—）方向動作時

OUT出力	DIR出力
	

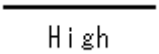
3

共通パルスモード
OUT：正論理
DIR：（＋方向）Low、（—方向）High

CW（＋）方向動作時

OUT出力	DIR出力
	

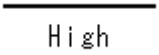
CCW（—）方向動作時

OUT出力	DIR出力
	

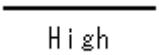
4

2パルスモード
負論理

CW（＋）方向動作時

OUT出力	DIR出力
	

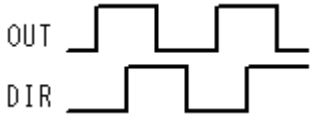
CCW（—）方向動作時

OUT出力	DIR出力
	

5

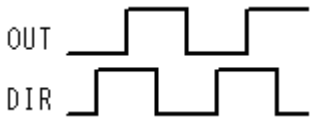
90度位相差モード（4逓倍）
OUT：進みパルス
DIR：遅れパルス

CW（＋）方向動作時

OUT/DIR出力


CCW（—）方向動作時

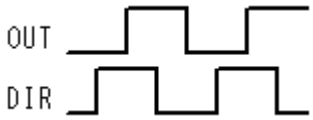
OUT/DIR出力

OUT/DIR出力


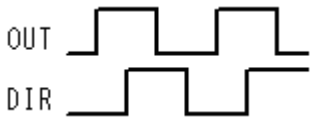
6

90度位相差モード（4逓倍）
OUT：遅れパルス
DIR：進みパルス

CW（＋）方向動作時

OUT/DIR出力


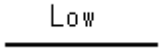
CCW（―）方向動作時

OUT/DIR出力


7

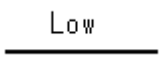
2パルスモード
正論理

CW（＋）方向動作時

OUT出力	DIR出力
	

CCW（―）方向動作時

OUT出力	DIR出力

OUT出力	DIR出力
	

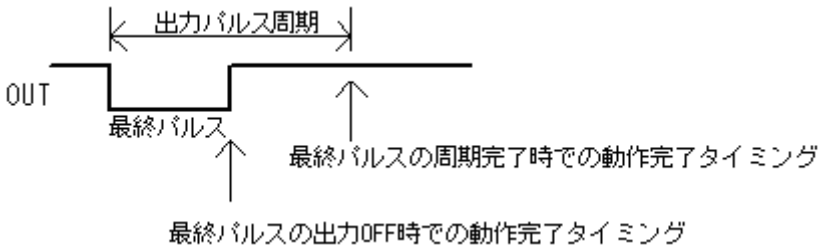
方向変化時の遅延

共通パルスモードを使用するモータードライバで、方向判別信号が変化してから指令パルスを受け付けるまでに時間が必要な場合に使用してください。
遅延ありに設定すると、方向判別信号の変化からパルス出力を0.2msec遅らせることができます。

動作完了タイミング

動作完了のタイミングを選択します。

設定	内容
最終パルスの周期完了時	（下図を参照してください）
最終パルスの出力OFF時	（下図を参照してください）
INP信号入力時	サーボドライバの位置決め完了信号の入力時に動作完了となります



移動速度補正（三角駆動回避）

移動速度補正をONにした場合、位置決め動作またはCP動作で加減速動作させた時に、移動量が少ないと自動的に移動速度を低下させて三角駆動を回避します。

> 直線加減速の時のみ設定が有効となり、S字加減速の時は自動的にONとなります

カウンタ

出力パルスカウンタ

出力するパルスをカウントします。
カウンタ値の設定・取得ができます。

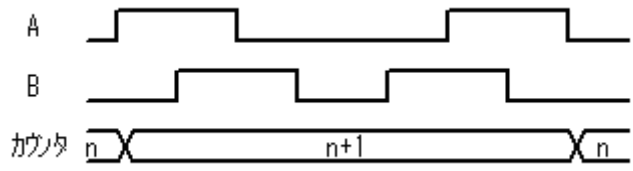
> モーター動作パラメータにて絶対座標指定をした場合、このカウンタ値を基準として動作します

エンコーダカウンタ

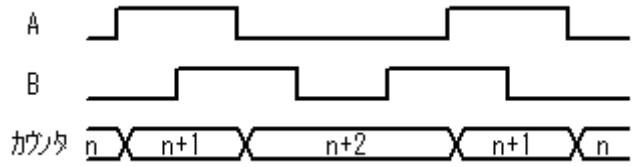
エンコーダパルスをカウントします。
カウンタ値の設定・取得ができます。

エンコーダパルス入力の設定は4種類から選択できます。

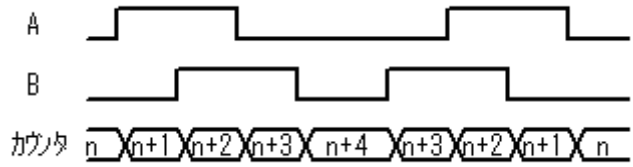
- 90度位相差信号1逓倍



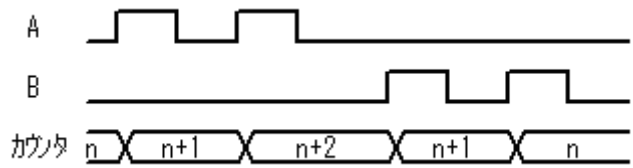
- 90度位相差信号2逓倍



- 90度位相差信号4逓倍



- 2パルス入力



カウンタラッチ機能

以下の条件でカウンタをラッチすることができます。

- LTC信号OFF→ON時
- ORG信号OFF→ON時

カウンタ値のラッチ直後にカウンタをクリアすることができます（ラッチ＆クリア機能）。
ラッチ＆クリア機能を使用した場合、クリア（ラッチ）タイミングとカウントタイミングが一致した時には、ラッチ直後に0にならずに+1または-1になります。

コンパレータ

出力パルスカウンタ用とエンコーダカウンタ用の2つのコンパレータを各軸ごとに内蔵しています。

PmcmSetComparatorConfig関数で設定した比較条件は、内部同期信号として使用します。
内部同期信号の設定はPmcmSetSynchroOut関数、内部同期信号によるスタートはPmcmSetSynchroConfig関数で設定します。

イベント

イベントが発生した時にユーザーに通知させることができます。

イベント発生要因

停止イベント

- 自動停止時
- +EL入力ONによる停止時
- -EL入力ONによる停止時
- ALM入力ONによる停止時
- EMG入力ONによる停止時
- SD入力ONによる減速停止時

状態イベント

- 加速開始時
- 加速終了時
- 減速開始時
- 減速終了時
- コンパレータ（出力パルス）条件成立時
- コンパレータ（エンコーダ）条件成立時
- LTC入力によるカウント値のラッチ時
- ORG入力ONによるカウント値のラッチ時
- SD入力ON時

イベント設定順序

1. PmcmSetEventMask関数でイベントマスクを設定する
2. PmcmSetEvent関数でイベント開始する

イベント発生確認順序

1. PmcmGetEventFactor関数でイベント発生要因を取得する
2. 第3引数のEVENTFACTORPMCM構造体のnResult
   - 0
     イベントが発生しています。
     続けてwStop、wStateを参照し、イベントの発生要因を確認してください。
   - 0以外
     エラーによりイベントは発生していません。

例）PmcmClose関数の実行によるクローズ、USBケーブルが抜けた、など。

連続動作

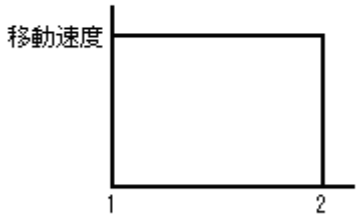
動作概要

停止要求があるまでモーターを動作させます。

動作例

定速起動

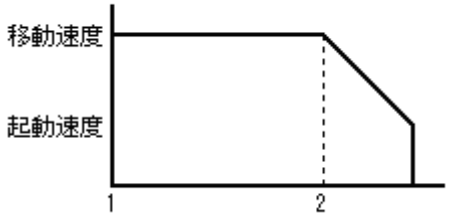
動作パラメータの起動モードは定速起動を指定します。
動作停止は即停止を指定します。



1. 動作開始
2. 動作停止要求による即停止

定速－減速起動

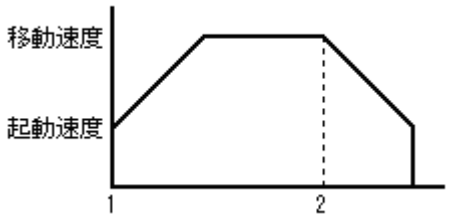
動作パラメータの起動モードは定速－減速起動を指定します。
動作停止は減速停止を指定します。



1. 動作開始
2. 動作停止要求による減速開始
   即停止を指定した場合は即停止します。

加減速起動

動作パラメータの起動モードは加減速起動を指定します。
動作停止は減速停止を指定します。



1. 動作開始
2. 動作停止要求による減速開始
   即停止を指定した場合は即停止します。

動作パラメータ設定例

定速起動

動作パラメータ	設定値	備考
動作モード[wMoveMode]	PMCM_JOG	連続動作を指定する
起動モード[wStartMode]	PMCM_CONST	定速起動を指定する
速度倍率[fSpeedRate]	1	
加減速モード[wAccDecMode]	PMCM_ACC_LINEAR	
起動速度[fLowSpeed]	1000	移動速度と同じ値を設定する
移動速度[fSpeed]	1000	
加速時間[wAccTime]	0	定速起動では0を設定する
減速時間[wDecTime]	0	定速起動では0を設定する
加速S字区間[fSAccSpeed]	0	定速起動では0を設定する
減速S字区間[fSDecSpeed]	0	定速起動では0を設定する
スローダウンポイント[lSlowdown]	-1	連続動作では-1を設定する
移動方向[lStep]	PMCM_DIR_CW	-方向に移動する場合はPMCM_DIR_CCW
絶対座標指定[bAbsolutePtp]	0	連続動作では0を指定する

定速ー減速起動

動作パラメータ	設定値	備考
動作モード[wMoveMode]	PMCM_JOG	連続動作を指定する

動作パラメータ	設定値	備考
起動モード[wStartMode]	PMCM_CONST_DEC	定速－減速起動を指定する
速度倍率[fSpeedRate]	1	
加減速モード[wAccDecMode]	PMCM_ACC_LINEAR	
起動速度[fLowSpeed]	100	
移動速度[fSpeed]	1000	
加速時間[wAccTime]	0	定速－減速起動では0を設定する
減速時間[wDecTime]	500	
加速S字区間[fSAccSpeed]	0	定速－減速起動では0を設定する
減速S字区間[fSDecSpeed]	0	加減速モードでS字加減速を選択した場合は値を設定する
スローダウンポイント[lSlowdown]	-1	連続動作では-1を設定する
移動方向[lStep]	PMCM_DIR_CW	-方向に移動する場合はPMCM_DIR_CCW
絶対座標指定[bAbsolutePtp]	0	連続動作では0を指定する

加減速起動

動作パラメータ	設定値	備考
動作モード[wMoveMode]	PMCM_JOG	連続動作を指定する
起動モード[wStartMode]	PMCM_ACC_DEC	加減速起動を指定する
速度倍率[fSpeedRate]	1	
加減速モード[wAccDecMode]	PMCM_ACC_SCURVE	
起動速度[fLowSpeed]	100	
移動速度[fSpeed]	1000	
加速時間[wAccTime]	2000	
減速時間[wDecTime]	2000	
加速S字区間[fSAccSpeed]	300	加減速モードでPMCM_ACC_LINEARを指定した場合は0を設定する
減速S字区間[fSDecSpeed]	300	加減速モードでPMCM_ACC_LINEARを指定した場合は0を設定する
スローダウンポイント[lSlowdown]	-1	連続動作では-1を設定する
移動方向[lStep]	PMCM_DIR_CW	-方向に移動する場合はPMCM_DIR_CCW
絶対座標指定[bAbsolutePtp]	0	連続動作では0を指定する

原点復帰動作

動作概要

ORG信号の入力があるまでモーターを動作させます。

動作例

動作例については原点信号を参照してください。

動作パラメータ設定例

動作パラメータ	設定値	備考
動作モード[wMoveMode]	PMCM_ORG	原点復帰動作を指定する
起動モード[wStartMode]	PMCM_ACC_DEC	
速度倍率[fSpeedRate]	1	
加減速モード[wAccDecMode]	PMCM_ACC_LINEAR	
起動速度[fLowSpeed]	100	
移動速度[fSpeed]	1000	
加速時間[wAccTime]	1000	
減速時間[wDecTime]	1000	
加速S字区間[fSAccSpeed]	0	加減速モードでS字加減速を選択した場合は値を設定する
減速S字区間[fSDecSpeed]	0	加減速モードでS字加減速を選択した場合は値を設定する
スローダウンポイント[lSlowdown]	-1	原点復帰動作では-1を指定する
移動方向[lStep]	PMCM_DIR_CW	-方向に移動する場合はPMCM_DIR_CCW
絶対座標指定[bAbsolutePtp]	0	原点復帰動作では0を指定する

位置決め動作（PTP動作）

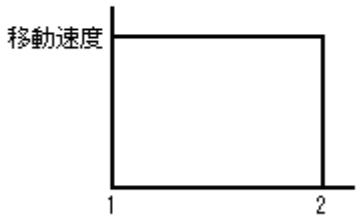
動作概要

設定されたパルス数だけパルスを出力し、モーターを動作させます。

動作例

定速起動

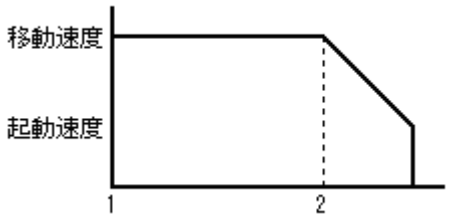
動作パラメータの起動モードは定速起動を指定します。



1. 動作開始
2. 移動パルス数出力終了による停止

定速－減速起動

動作パラメータの起動モードは定速－減速起動を指定します。

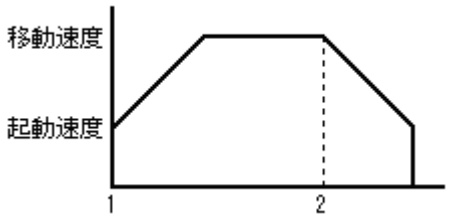


1. 動作開始
2. スローダウンポイント点到達による減速開始
   スローダウンポイントが0に設定されている場合は即停止します。

> スローダウンポイントをマニュアル設定（-1以外を設定）した場合に、スローダウンポイントが移動パルス数よりも大きい場合は起動速度で動作します

加減速起動

動作パラメータの起動モードは加減速起動を指定します。

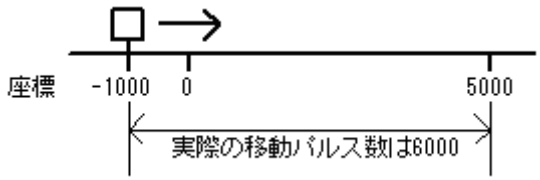


1. 動作開始
2. スローダウンポイント点到達による減速開始
   スローダウンポイントが0に設定されている場合は即停止します。

> スローダウンポイントをマニュアル設定（-1以外を設定）した場合に、スローダウンポイントが移動パルス数よりも大きい場合は起動速度で動作します

備考

動作パラメータの絶対座標指定で1を指定した場合はPmcmStartMotion関数を実行した時点での出力パルスカウンタ値を基準として移動量を算出しています。
従って、**モーターが停止している時にPmcmStartMotion関数を実行しないと正しく動作しません。**
下図の例は出力パルスカウンタ値が-1000で、移動パルス数に5000を設定、絶対座標指定を1に指定した場合です。



動作パラメータ設定例

定速起動

動作パラメータ	設定値	備考
動作モード[wMoveMode]	PMCM_PTP	位置決め動作を指定する
起動モード[wStartMode]	PMCM_CONST	定速起動を指定する
速度倍率[fSpeedRate]	1	
加減速モード[wAccDecMode]	PMCM_ACC_LINEAR	
起動速度[fLowSpeed]	1000	移動速度と同じ値を設定する
移動速度[fSpeed]	1000	
加速時間[wAccTime]	0	定速起動では0を設定する

動作パラメータ	設定値	備考
減速時間[wDecTime]	0	定速起動では0を設定する
加速S字区間[fSAccSpeed]	0	定速起動では0を設定する
減速S字区間[fSDecSpeed]	0	定速起動では0を設定する
スローダウンポイント[lSlowdown]	-1	定速起動では-1を設定する
移動パルス数[lStep]	1000	-方向に移動する場合はマイナスの値を設定する
絶対座標指定[bAbsolutePtp]	0	絶対座標指定の場合は1を指定する

定速－減速起動

動作パラメータ	設定値	備考
動作モード[wMoveMode]	PMCM_PTP	位置決め動作を指定する
起動モード[wStartMode]	PMCM_CONST_DEC	定速－減速起動を指定する
速度倍率[fSpeedRate]	1	
加減速モード[wAccDecMode]	PMCM_ACC_LINEAR	
起動速度[fLowSpeed]	100	
移動速度[fSpeed]	1000	
加速時間[wAccTime]	0	定速－減速起動では0を設定する
減速時間[wDecTime]	500	
加速S字区間[fSAccSpeed]	0	定速－減速起動では0を設定する
減速S字区間[fSDecSpeed]	0	加減速モードでS字加減速を選択した場合は値を設定する
スローダウンポイント[lSlowdown]	-1	マニュアル設定の場合は値を設定する
移動パルス数[lStep]	1000	-方向に移動する場合はマイナスの値を設定する
絶対座標指定[bAbsolutePtp]	0	絶対座標指定の場合は1を指定する

加減速起動

動作パラメータ	設定値	備考
動作モード[wMoveMode]	PMCM_PTP	位置決め動作を指定する
起動モード[wStartMode]	PMCM_ACC_DEC	加減速起動を指定する

動作パラメータ	設定値	備考
速度倍率[fSpeedRate]	1	
加減速モード[wAccDecMode]	PMCM_ACC_SCURVE	
起動速度[fLowSpeed]	100	
移動速度[fSpeed]	1000	
加速時間[wAccTime]	2000	
減速時間[wDecTime]	2000	
加速S字区間[fSAccSpeed]	300	加減速モードでPMCM_ACC_LINEARを指定した場合は0を設定する
減速S字区間[fSDecSpeed]	300	加減速モードでPMCM_ACC_LINEARを指定した場合は0を設定する
スローダウンポイント[lSlowdown]	-1	マニュアル設定の場合は値を設定する
移動パルス数[lStep]	1000	-方向に移動する場合はマイナスの値を設定する
絶対座標指定[bAbsolutePtp]	0	絶対座標指定の場合は1を指定する

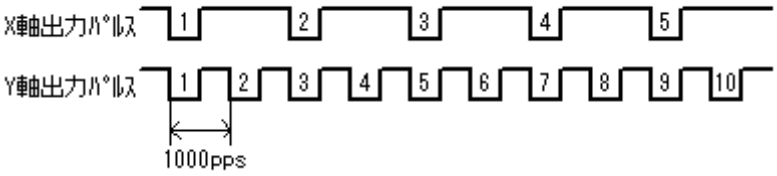
直線補間動作

動作概要

2軸の直線補間動作ができます。

動作例

下記動作パラメータ設定例の内容で設定した場合の動作です。



備考

各動作パラメータは最大移動量の軸に対する設定値を設定してください。

> 絶対座標指定を1に設定して動作を開始したときに、移動量が-134217728 ～ +134217727の範囲を超えていた場合は動作しません

動作パラメータ設定例

動作パラメータ	設定値	備考
起動モード[wStartMode]	PMCM_CONST	
速度倍率[fSpeedRate]	1	
加減速モード[wAccDecMode]	PMCM_ACC_LINEAR	
起動速度[fLowSpeed]	1000	
移動速度[fSpeed]	1000	
加速時間[wAccTime]	0	定速起動では0を設定する
減速時間[wDecTime]	0	定速起動では0を設定する
加速S字区間[fSAccSpeed]	0	定速起動では0を設定する
減速S字区間[fSDecSpeed]	0	定速起動では0を設定する
スローダウンポイント[lSlowdown]	-1	定速起動では-1を設定する

動作パラメータ	設定値	備考
移動方向[lStep]	X軸[0] : 5 Y軸[1] : 10	
絶対座標指定[bAbsolutePtp]	X軸[0] : 0 Y軸[1] : 0	

CP（Continuous Path）動作

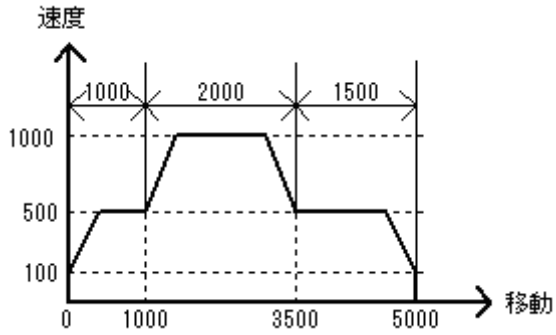
動作概要

位置決め動作を連続して実行します。

設定できる動作数は最大で96動作です。
それ以上の動作を設定したい場合はPmcmGetEmptyCp関数で空きを確認し、動作を追加設定します。

動作例

下記動作パラメータ設定例の内容で設定した場合の動作です。



備考

動作方向のEL信号の入力により即停止します。

動作パラメータ設定例

動作パラメータ	設定値（1回目）	設定値（2回目）	設定値（3回目）	備考
動作モード[wMoveMode]	PMCM_PTP	PMCM_PTP	PMCM_PTP	位置決め動作を指定する
起動モード[wStartMode]	PMCM_ACC_DEC	PMCM_ACC_DEC	PMCM_CONST_DEC	
速度倍率[fSpeedRate]	1	1	1	
加減速モード[wAccDecMode]	PMCM_ACC_LINEAR	PMCM_ACC_LINEAR	PMCM_ACC_LINEAR	
起動速度[fLowSpeed]	100	500	100	
移動速度[fSpeed]	500	1000	500	
加速時間[wAccTime]	500	500	0	
減速時間[wDecTime]	500	500	500	
加速S字区間[fSAccSpeed]	0	0	0	

動作パラメータ	設定値（1回目）	設定値（2回目）	設定値（3回目）	備考
減速S字区間[fSDecSpeed]	0	0	0	
スローダウンポイント[lSlowdown]	0	-1	-1	
移動方向[lStep]	1000	2000	1500	
絶対座標指定[bAbsolutePtp]	0	0	0	CP動作では0を指定する

実行手順

1. 準備
   ボードとパソコンをUSBケーブルで接続し、ボードへ電源を供給してください。
   （ボードへ電源供給後、パソコンがボードを認識するまでに数秒程度必要となる場合があります）
2. ボードをオープン
   PmcmOpen関数を使用してボードをオープンします。
   PmcmOpen関数にてオープンしたボードは、アプリケーション終了時に必ずクローズ（PmcmClose関数）するようにしてください。
3. 各種設定
   PmcmSetSensorConfig関数やPmcmSetPulseConfig関数などを使用して各種設定をおこないます。
4. 動作パラメータ設定
   - 各軸の動作
     PmcmSetMotion関数を使用して動作パラメータの設定をおこないます。
   - 直線補間動作
     PmcmSetMotionLine関数を使用して動作パラメータの設定をおこないます。
   - CP動作
     PmcmSetMotionCp関数を使用して動作パラメータの設定をおこないます。
5. モーター動作
   - 各軸の動作
     PmcmStartMotion関数を使用してモーター動作を開始します。
   - 直線補間動作
     PmcmStartMotionLine関数を使用してモーター動作を開始します。
   - CP動作
     PmcmStartMotionCp関数を使用してモーター動作を開始します。
6. モーター停止
   PmcmStopMotion関数を使用してモーターを停止します。
7. ボードをクローズ
   PmcmClose関数を使用してボードをクローズします。
8. 終了
   ボードへの電源供給を止めます。

戻り値一覧

エラーコード (16進数値/10進数値)	意味	対処方法
PMCM_RESULT_SUCCESS (H'00000000 / 0)	正常終了	
PMCM_RESULT_ERROR (H'CE000001 / -838860799)	エラー	エラーが発生しました。USBケーブルが抜けている事などが考えられます。
PMCM_RESULT_NOT_OPEN (H'CE000002 / -838860798)	オープンされていない	オープンされていないボードに対して操作がおこなわれました。PmcmOpen関数にてオープンされたユニットに対して操作をおこなってください。
PMCM_RESULT_ALREADY_OPEN (H'CE000003 / -838860797)	オープン済み	既にオープンされているボードに対してPmcmOpen関数が実行されました
PMCM_RESULT_INVALID_ID (H'CE000004 / -838860796)	IDが不正	指定されたIDが不正です。IDは0～15を指定してください。
PMCM_RESULT_CANNOT_OPEN (H'CE000006 / -838860794)	ボードがオープンできない	ボードがオープンできませんでした。USBケーブルが接続されていること、ボードの電源が入っていることを確認してください。
PMCM_RESULT_PARAMETER_ERROR (H'CE000008 / -838860792)	引数不正	関数に渡された引数が不正です。関数仕様やサンプルプログラムを参照し、引数の確認をしてください。
PMCM_RESULT_MEM_ALLOC_ERROR (H'CE000009 / -838860791)	メモリ確保エラー	利用可能なメモリが不足しています。不要なアプリケーションを終了するなどし、利用可能なメモリを増やしてください。
PMCM_RESULT_MODELNAME_ERROR (H'CE00000A / -838860790)	型名指定が不正	指定された型名は存在していないか、サポートしていません。型名を再度確認してください。
PMCM_RESULT_HARDWARE_ERROR (H'CE00000B / -838860789)	ハードウェアエラー	以下の原因などが考えられます。 1. 動作中に供給電源電圧が定格範囲を外れてしまっている（供給電源の電流容量不足による電圧低下など） 2. 外部との接続方法に問題がある 上記を確認しても問題が見当たらない場合は、ボードのハードウェアに問題が発生している可能性があります。現象を弊社サポートまでご連絡ください。
PMCM_RESULT_NOT_SUPPORTED (H'CE00000C / -838860788)	サポートされていない	サポートされていない関数が実行されました
PMCM_RESULT_ALREADY_EVENT (H'CE00000D / -838860787)	イベント設定済み	PmcmSetEvent関数が2回実行されました
PMCM_RESULT_THREAD_FAILED (H'CE00000E / -838860786)	スレッド起動失敗	DLL内部でスレッドの起動に失敗しました。不要なアプリケーションを終了するなどし、利用可能なメモリを増やしてください。
PMCM_RESULT_MOTION_NOT_READY (H'CE00000F / -838860785)	MOTIONデータ未設定	動作パラメータが設定されていないため、モーター動作を開始できません。動作パラメータを設定後に再度実行してください。
PMCM_RESULT_CP_NOT_READY (H'CE000010 / -838860784)	CPデータ未設定	CPデータが未設定のため、CP動作を開始できません。CPデータを設定後に再度実行してください。

エラーコード (16進数値/10進数値)	意味	対処方法
PMCM_RESULT_CP_DATA_FULL (H'CE000011 / -838860783)	CPデータフル	CPデータがいっぱいです。空きを確認の上、CPデータを再度設定してください。
PMCM_RESULT_RANGE_OVER (H'CE000012 / -838860782)	設定範囲オーバー	直線補間動作で絶対座標指定をおこないましたが、移動パルス数が移動可能範囲を超えています
PMCM_RESULT_FATAL_ERROR (H'CEFFFFFF / -822083585)	致命的なエラー	致命的なエラーです。どのような状況で発生したかを弊社サポートまでご連絡ください。

C#での注意事項

構造体

C#では以下の構造体を使用する場合、インスタンス生成後にメンバ関数のInitialize()を実行する必要があります。

MOTIONLINEPMCM

- 使用する関数
  PmcmSetMotionLine関数
  PmcmGetMotionLine関数
- サンプルコード

```
Pmcm.MOTIONLINEPMCM motionLine = new Pmcm.MOTIONLINEPMCM();
motionLine.Initialize();
```

EVENTFACTORPMCM

- 使用する関数
  PmcmGetEventFactor関数
- サンプルコード

```
Pmcm.EVENTFACTORPMCM eventFactor = new Pmcm.EVENTFACTORPMCM();
eventFactor.Initialize();
```

関数

C#で関数を使用する場合、Pmcmの後に"."（ドット）を付加して使用します（サンプルプログラムも参照してください）。

- 例

C#以外	C#
PmcmOpen	Pmcm.Open
PmcmSetSensorConfig	Pmcm.SetSensorConfig

Visual Basic（.NET2002以降）での注意事項

構造体

Visual Basic（.NET2002以降）では以下の構造体を使用する場合、変数宣言後にメンバ関数のInitialize()を実行する必要があります。

MOTIONLINEPMCM

- 使用する関数
  PmcmSetMotionLine関数
  PmcmGetMotionLine関数
- サンプルコード

```
Dim motionLine As MOTIONLINEPMCM
motionLine.Initialize()
```

EVENTFACTORPMCM

- 使用する関数
  PmcmGetEventFactor関数
- サンプルコード

```
Dim eventFactor As EVENTFACTORPMCM
eventFactor.Initialize()
```

関数一覧

関数	機能
PmcmOpen	ボードのオープンをおこないます
PmcmClose	ボードのクローズをおこないます

PmcmOpen

機能

ボードのオープンをおこない、ボードへのアクセスをおこなえるようにします。

書式

```
INT PmcmOpen(
    WORD wID,
    LPCSTR lpszModelName
);
```

パラメータ

wID

オープンするボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

lpszModelName

オープンするボードの型名を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LPCSTR	String^	string	String	String	const char*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
オープンに失敗した場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

> **Warning**
>
> PmcmOpen関数でオープンしたボードは、アプリケーション終了時に必ずPmcmClose関数でクローズしてください。

使用例

IDが0に設定されているPMC-M2C-Uをオープンします。

C/C++

```
int nResult;
nResult = PmcmOpen(0, "PMC-M2C-U");
```

C++/CLI

```
int result;
result = PmcmOpen(0, "PMC-M2C-U");
```

C#

```
int result;
result = Pmcm.Open(0, "PMC-M2C-U");
```

VB（.NET2002以降）

```
Dim result As Integer
result = PmcmOpen(0, "PMC-M2C-U")
```

VB6.0/VBA

```
Dim lngResult As Long
Dim strModelName As String
strModelName = "PMC-M2C-U"
lngResult = PmcmOpen(0, strModelName)
```

GCC

```
int32_t result;
result = PmcmOpen(0, "PMC-M2C-U");
```

PmcmClose

機能

ボードのクローズをおこない、ボードへのアクセスを禁止します。

書式

```
BOOL PmcmClose(
    WORD wID
);
```

パラメータ

wID

クローズするボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合はTRUEが返ります。
クローズに失敗した場合はFALSEが返ります。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	BOOL	bool	bool	Boolean	Boolean	int32_t

備考

> **Warning**
>
> PmcmOpen関数でオープンしたボードは、アプリケーション終了時に必ずPmcmClose関数でクローズしてください。

使用例

IDが0のボードのクローズ処理をおこないます。

C/C++

```
BOOL bResult;
bResult = PmcmClose(0);
```

C++/CLI

```
bool result;
result = PmcmClose(0);
```

C#

```
bool result;
result = Pmcm.Close(0);
```

VB（.NET2002以降）

```
Dim result As Boolean
result = PmcmClose(0)
```

VB6.0/VBA

```
Dim blnResult As Boolean
blnResult = PmcmClose(0)
```

GCC

```
int32_t result;
result = PmcmClose(0);
```

関数一覧

設定

関数	機能
PmcmSetSensorConfig	センサの設定をおこないます
PmcmSetPulseConfig	パルス出力の設定をおこないます
PmcmSetSynchroConfig	起動・停止条件の設定をおこないます
PmcmSetSynchroOut	内部同期信号出力の設定をおこないます
PmcmSetComparatorConfig	コンパレータの設定をおこないます
PmcmSetLatchConfig	カウンタラッチの設定をおこないます
PmcmSetEncoderConfig	エンコーダ入力の設定をおこないます
PmcmSetErcConfig	ERC信号の設定をおこないます
PmcmSetMotion	各軸の動作パラメータの設定をおこないます
PmcmSetMotionLine	直線補間動作パラメータの設定をおこないます
PmcmSetMotionCp	CP動作パラメータの設定をおこないます

設定取得

関数	機能
PmcmGetSensorConfig	センサの設定を取得します
PmcmGetPulseConfig	パルス出力の設定を取得します
PmcmGetSynchroConfig	起動・停止条件の設定を取得します
PmcmGetSynchroOut	内部同期信号出力の設定を取得します
PmcmGetComparatorConfig	コンパレータの設定を取得します

関数	機能
PmcmGetLatchConfig	カウンタラッチの設定を取得します
PmcmGetEncoderConfig	エンコーダ入力の設定を取得します
PmcmGetErcConfig	ERC信号の設定を取得します
PmcmGetMotion	各軸の動作パラメータの設定を取得します
PmcmGetMotionLine	直線補間動作パラメータの設定を取得します
PmcmGetMotionCp	CP動作パラメータの設定を取得します

動作開始・停止

関数	機能
PmcmStartMotion	各軸の動作を開始します
PmcmStartMotionLine	直線補間動作を開始します
PmcmStartMotionCp	CP動作を開始します
PmcmStopMotion	モーターを停止します
PmcmChangeSpeed	動作中に速度を変更します
PmcmChangeStep	動作中に目標位置を変更します

状態取得

関数	機能
PmcmGetStatus	動作状態や各信号状態を取得します

CP動作

関数	機能
PmcmGetEmptyCp	CPの空き動作パラメータ数を取得します

イベント

関数	機能
PmcmSetEventMask	イベントマスクの設定をおこないます
PmcmGetEventMask	イベントマスクの設定を取得します
PmcmSetEvent（Windows）	イベントの設定をおこないます
PmcmSetEvent（Linux）	イベントの設定をおこないます
PmcmWaitForEvent（Linux）	イベントの発生を待ちます
PmcmGetEventFactor	イベント発生要因を取得します

カウンタ

関数	機能
PmcmSetCounter	出力パルスカウンタの値を設定します
PmcmGetCounter	出力パルスカウンタの値を取得します
PmcmGetLatchCounter	出力パルスカウンタのラッチカウンタの値を取得します
PmcmSetEncoder	エンコーダカウンタの値を設定します
PmcmGetEncoder	エンコーダカウンタの値を取得します
PmcmGetLatchEncoder	エンコーダカウンタのラッチカウンタの値を取得します

PmcmChangeSpeed

機能

動作中に速度の変更をします。

書式

```
INT PmcmChangeSpeed(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    PFLOAT pfSpeed
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

速度を変更する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

設定する項目を指定します。

設定値	内容

設定値	内容
PMCM_CHG_IMM_LOW_SPEED	起動速度へ即変更
PMCM_CHG_IMM_SPEED	移動速度へ即変更
PMCM_CHG_DEC_LOW_SPEED	起動速度へ減速して変更
PMCM_CHG_ACC_SPEED	移動速度へ加速して変更
PMCM_CHG_NEW_SPEED	指定速度へ変更

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pfSpeed

速度を指定します。
wModeの設定値によって設定する内容が異なります。

> バッファは2軸分用意してください

wModeの設定値	設定値
PMCM_CHG_IMM_LOW_SPEED PMCM_CHG_IMM_SPEED PMCM_CHG_DEC_LOW_SPEED PMCM_CHG_ACC_SPEED	0を指定してください
PMCM_CHG_NEW_SPEED	速度を指定します。 SetMotionで指定した速度倍率によって設定範囲は異なります。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PFLOAT	float*	float	Single	Single	float*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

PMCM_CHG_NEW_SPEEDをおこなった後はPMCM_CHG_IMM_SPEEDとPMCM_CHG_ACC_SPEEDは無効となります。

複数軸同時に設定しない場合でも、速度バッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
FLOAT fSpeed[2]; ←2軸分用意
fSpeed[0] ←X軸の速度
fSpeed[1] ←Y軸の速度
```

使用例

IDが0のボードの、Y軸の速度を2000ppsに変更します。

C/C++

```
int nResult;
WORD wAxis;
FLOAT fSpeed[2];
wAxis = PMCM_AXIS_Y;
fSpeed[0] = 0;
fSpeed[1] = 2000;
nResult = PmcmChangeSpeed(0, wAxis, PMCM_CHG_NEW_SPEED, fSpeed);
```

C++/CLI

```
int result;
unsigned short axis;
float speed[2];
axis = PMCM_AXIS_Y;
speed[0] = 0;
speed[1] = 2000;
result = PmcmChangeSpeed(0, axis, PMCM_CHG_NEW_SPEED, speed);
```

C#

```
int result;
ushort axis;
float[] speed = new float[2];
axis = Pmcm.PMCM_AXIS_Y;
speed[0] = 0;
speed[1] = 2000;
result = Pmcm.ChangeSpeed(0, axis, Pmcm.PMCM_CHG_NEW_SPEED, speed);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim speed(1) As Single
axis = PMCM_AXIS_Y
speed(0) = 0
speed(1) = 2000
result = PmcmChangeSpeed(0, axis, PMCM_CHG_NEW_SPEED, speed)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim sngSpeed(1) As Single
intAxis = PMCM_AXIS_Y
sngSpeed(0) = 0
sngSpeed(1) = 2000
lngResult = PmcmChangeSpeed(0, intAxis, PMCM_CHG_NEW_SPEED, sngSpeed(0))
```

GCC

```
int32_t result;
uint16_t axis;
float speed[2];
axis = PMCM_AXIS_Y;
speed[0] = 0;
speed[1] = 2000;
result = PmcmChangeSpeed(0, axis, PMCM_CHG_NEW_SPEED, speed);
```

PmcmChangeStep

機能

動作中に目標位置の変更をします。
位置決め動作中のみ実行可能です。

書式

```
INT PmcmChangeStep(
    WORD wID,
    WORD wAxis,
    PLONG plStep
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

目標位置を変更する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

plStep

目標位置を設定します。
設定範囲は-134217728 ～ +134217727です。

> バッファは2軸分用意してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PLONG	long*	int	Integer	Long	int32_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

複数軸同時に設定しない場合でも、目標位置バッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
long lStep[2]; ←2軸分用意
lStep[0] ←X軸の目標位置
lStep[1] ←Y軸の目標位置
```

使用例

IDが0のボードの、Y軸の目標位置を5000に変更します。

C/C++

```
int nResult;
WORD wAxis;
long lStep[2];
lStep[0] = 0;
lStep[1] = 5000;
wAxis = PMCM_AXIS_Y;
nResult = PmcmChangeStep(0, wAxis, lStep);
```

C++/CLI

```
int result;
unsigned short axis;
long step[2];
step[0] = 0;
step[1] = 5000;
axis = PMCM_AXIS_Y;
result = PmcmChangeStep(0, axis, step);
```

C#

```
int result;
ushort axis;
int[] step = new int[2];
step[0] = 0;
step[1] = 5000;
axis = Pmcm.PMCM_AXIS_Y;
result = Pmcm.ChangeStep(0, axis, step);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim step(1) As Integer
axis = PMCM_AXIS_Y
step(0) = 0
step(1) = 5000
result = PmcmChangeStep(0, axis, step)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim lngStep(1) As Long
intAxis = PMCM_AXIS_Y
lngStep(0) = 0
lngStep(1) = 5000
lngResult = PmcmChangeStep(0, intAxis, lngStep(0))
```

GCC

```
int32_t result;
uint16_t axis;
int32_t step[2];
step[0] = 0;
step[1] = 5000;
axis = PMCM_AXIS_Y;
result = PmcmChangeStep(0, axis, step);
```

PmcmGetComparatorConfig

機能

コンパレータ設定の取得をします。

コンパレータについて

書式

```
INT PmcmGetComparatorConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    PCOMPPMCM pComp
);

typedef struct {
    WORD wConfig;
    LONG lCount;
} COMPPMCM, *PCOMPPMCM;
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

取得する項目を指定します。

設定値	内容
PMCM_COMP_COUNTER	出力パルスコンパレータの比較条件
PMCM_COMP_ENCODER	エンコーダコンパレータの比較条件

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pComp

設定パラメータを格納するバッファへのポインタを指定します。

> バッファは2軸分用意してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PCOMPPMCM	COMPPMCM*	COMPPMCM	COMPPMCM	COMPPMCM	PCOMPPMCM

wConfig

コンパレータ機能設定
取得する項目（wMode）によって値が異なります。

- wMode : PMCM_COMP_COUNTER

設定値	内容
0	コンパレータ機能OFF
1	設定値 = 出力パルスカウンタ
2	設定値 > 出力パルスカウンタ
3	設定値 < 出力パルスカウンタ

  初期値 : 0

- wMode : PMCM_COMP_ENCODER

設定値	内容

設定値	内容
0	コンパレータ機能OFF
1	設定値 = エンコーダカウンタ
2	設定値 > エンコーダカウンタ
3	設定値 < エンコーダカウンタ

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

lCount

コンパレータの設定値 : -134217728 ～ +134217727

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

複数軸同時に取得しない場合でも、設定パラメータバッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
COMPPMCM Comp[2]; ←2軸分用意
Comp[0] ←X軸の設定パラメータ
Comp[1] ←Y軸の設定パラメータ
```

使用例

IDが0のボードの、X軸の出力パルスコンパレータ設定を取得します。

C/C++

```
int nResult;
WORD wAxis;
COMPPMCM Comp[2];
wAxis = PMCM_AXIS_X;
nResult = PmcmGetComparatorConfig(0, wAxis, PMCM_COMP_COUNTER, Comp);
```

C++/CLI

```
int result;
unsigned short axis;
COMPPMCM comp[2];
axis = PMCM_AXIS_X;
result = PmcmGetComparatorConfig(0, axis, PMCM_COMP_COUNTER, comp);
```

C#

```
int result;
ushort axis;
Pmcm.COMPPMCM[] comp = new Pmcm.COMPPMCM[2];
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.GetComparatorConfig(0, axis, Pmcm.PMCM_COMP_COUNTER, comp);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim comp(1) As COMPPMCM
axis = PMCM_AXIS_X
result = PmcmGetComparatorConfig(0, axis, PMCM_COMP_COUNTER, comp)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim Comp(1) As COMPPMCM
intAxis = PMCM_AXIS_X
lngResult = PmcmGetComparatorConfig(0, intAxis, PMCM_COMP_COUNTER, Comp(0))
```

GCC

```
int32_t result;
uint16_t axis;
COMPPMCM comp[2];
axis = PMCM_AXIS_X;
result = PmcmGetComparatorConfig(0, axis, PMCM_COMP_COUNTER, comp);
```

PmcmGetCounter

機能

出力パルスカウンタ値の取得をします。

書式

```
INT PmcmGetCounter(
    WORD wID,
    WORD wAxis,
    PLONG plData
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

取得する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

plData

カウンタ値を格納するバッファへのポインタを指定します。

> バッファは2軸分用意してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PLONG	long*	int	Integer	Long	int32_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

複数軸同時に取得しない場合でも、カウンタ値バッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
long lData[2]; ←2軸分用意
lData[0] ←X軸のカウンタ値
lData[1] ←Y軸のカウンタ値
```

使用例

IDが0のボードの、X軸とY軸の出力パルスカウンタ値を取得します。

C/C++

```
int nResult;
WORD wAxis;
long lData[2];
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmGetCounter(0, wAxis, lData);
```

C++/CLI

```
int result;
unsigned short axis;
long data[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmGetCounter(0, axis, data);
```

C#

```
int result;
ushort axis;
int[] data = new int[2];
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.GetCounter(0, axis, data);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim data(1) As Integer
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmGetCounter(0, axis, data)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim lngData(1) As Long
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmGetCounter(0, intAxis, lngData(0))
```

GCC

```
int32_t result;
uint16_t axis;
int32_t data[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmGetCounter(0, axis, data);
```

PmcmGetEmptyCp

機能

CPの空き動作パラメータ数の取得をします。

書式

```
INT PmcmGetEmptyCp(
    WORD wID,
    WORD wAxis,
    PWORD pwNumofEmpty
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

取得する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pwNumofEmpty

CP空き動作パラメータ数を格納するバッファへのポインタを指定します。

> バッファは2軸分用意してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PWORD	unsigned short*	ushort	Short	Integer	uint16_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

複数軸同時に取得しない場合でも、CP空き動作パラメータ数バッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
WORD wEmpty[2]; ←2軸分用意
wEmpty[0] ←X軸のCP空き動作パラメータ数
wEmpty[1] ←Y軸のCP空き動作パラメータ数
```

使用例

IDが0のボードの、X軸とY軸のCP空き動作パラメータ数を取得します。

C/C++

```
int nResult;
WORD wAxis;
WORD wEmpty[2];
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmGetEmptyCp(0, wAxis, wEmpty);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned short empty[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmGetEmptyCp(0, axis, empty);
```

C#

```
int result;
ushort axis;
ushort[] empty = new ushort[2];
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.GetEmptyCp(0, axis, empty);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim empty(1) As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmGetEmptyCp(0, axis, empty)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intEmpty(1) As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmGetEmptyCp(0, intAxis, intEmpty(0))
```

GCC

```
int32_t result;
uint16_t axis;
uint16_t empty[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmGetEmptyCp(0, axis, empty);
```

PmcmGetEncoder

機能

エンコーダカウンタ値の取得をします。

書式

```
INT PmcmGetEncoder(
    WORD wID,
    WORD wAxis,
    PLONG plData
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

取得する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

plData

カウンタ値を格納するバッファへのポインタを指定します。

> バッファは2軸分用意してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PLONG	long*	int	Integer	Long	int32_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

複数軸同時に取得しない場合でも、カウンタ値バッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
long lData[2]; ←2軸分用意
lData[0] ←X軸のカウンタ値
lData[1] ←Y軸のカウンタ値
```

使用例

IDが0のボードの、X軸とY軸のエンコーダカウンタ値を取得します。

C/C++

```
int nResult;
WORD wAxis;
long lData[2];
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmGetEncoder(0, wAxis, lData);
```

C++/CLI

```
int result;
unsigned short axis;
long data[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmGetEncoder(0, axis, data);
```

C#

```
int result;
ushort axis;
int[] data = new int[2];
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.GetEncoder(0, axis, data);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim data(1) As Integer
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmGetEncoder(0, axis, data)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim lngData(1) As Long
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmGetEncoder(0, intAxis, lngData(0))
```

GCC

```
int32_t result;
uint16_t axis;
int32_t data[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmGetEncoder(0, axis, data);
```

PmcmGetEncoderConfig

機能

エンコーダ入力設定の取得をします。

カウンタについて

書式

```
INT PmcmGetEncoderConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    PWORD pwConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

取得する項目を指定します。

設定値	内容
PMCM_ENCODER_MODE	エンコーダ入力モード
PMCM_ENCODER_DIR	A/B入力カウント方向
PMCM_ENCODER_FILTER	入力フィルタ

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pwConfig

設定値を格納するバッファへのポインタを指定します。 取得する項目（wMode）によって値が異なります。

- wMode : PMCM_ENCODER_MODE

設定値	内容
0	1逓倍
1	2逓倍
2	4逓倍
3	アップ・ダウンの2パルス入力

  初期値 : 0

- wMode : PMCM_ENCODER_DIR

設定値	内容
0	Aの位相が進んでいる時、またはAの立ち上がりでカウントアップ
1	Bの位相が進んでいる時、またはBの立ち上がりでカウントアップ

  初期値 : 0

- wMode : PMCM_ENCODER_FILTER

設定値	内容
0	フィルタ機能OFF

設定値	内容
1	フィルタ機能ON

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PWORD	unsigned short*	out ushort	Short	Integer	uint16_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸のエンコーダ入力モードを取得します。

C/C++

```
int nResult;
WORD wAxis;
WORD wConfig;
wAxis = PMCM_AXIS_X;
nResult = PmcmGetEncoderConfig(0, wAxis, PMCM_ENCODER_MODE, &wConfig);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned short config;
axis = PMCM_AXIS_X;
result = PmcmGetEncoderConfig(0, axis, PMCM_ENCODER_MODE, &config);
```

C#

```
int result;
ushort axis;
ushort config;
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.GetEncoderConfig(0, axis, Pmcm.PMCM_ENCODER_MODE, out config);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim config As Short
axis = PMCM_AXIS_X
result = PmcmGetEncoderConfig(0, axis, PMCM_ENCODER_MODE, config)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intConfig As Integer
intAxis = PMCM_AXIS_X
lngResult = PmcmGetEncoderConfig(0, intAxis, PMCM_ENCODER_MODE, intConfig)
```

GCC

```
int32_t result;
uint16_t axis;
uint16_t config;
axis = PMCM_AXIS_X;
result = PmcmGetEncoderConfig(0, axis, PMCM_ENCODER_MODE, &config);
```

PmcmGetErcConfig

機能

ERC信号設定の取得をします。

書式

```
INT PmcmGetErcConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    PWORD pwConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

取得する項目を指定します。

設定値	内容

設定値	内容
PMCM_ERC_LOGIC	出力論理
PMCM_ERC_AUTOOUT_1	自動出力設定1
PMCM_ERC_AUTOOUT_2	自動出力設定2
PMCM_ERC_WIDTH	出力幅
PMCM_ERC_OFF_TIMER	OFFタイマ

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pwConfig

設定値を格納するバッファへのポインタを指定します。
取得する項目（wMode）によって値が異なります。

- wMode : PMCM_ERC_LOGIC

  ERC信号の出力論理

設定値	内容
0	負論理
1	正論理

  初期値 : 0

- wMode : PMCM_ERC_AUTOOUT_1

  ERC信号の自動出力設定

設定値	内容
0	EL、ALM、EMG入力停止時にERC信号を出力しない
1	EL、ALM、EMG入力停止時にERC信号を自動出力 （減速停止の場合は出力されません）

  初期値 : 0

- wMode : PMCM_ERC_AUTOOUT_2

  ERC信号の自動出力設定

設定値	内容
0	原点復帰完了時にERC信号を出力しない
1	原点復帰完了時にERC信号を自動出力

  初期値 : 0

- wMode : PMCM_ERC_WIDTH

  ERC信号の出力幅

設定値	内容
0	12μsec
1	102μsec
2	408μsec
3	1.6msec
4	13msec
5	52msec
6	104msec
7	レベル出力

  初期値 : 0

- wMode : PMCM_ERC_OFF_TIMER

  ERC信号のOFFタイマ

設定値	内容
0	0μsec
1	12μsec
2	1.6msec

設定値	内容
3	104msec

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PWORD	unsigned short*	out ushort	Short	Integer	uint16_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸の出力論理を取得します。

C/C++

```
int nResult;
WORD wAxis;
WORD wConfig;
wAxis = PMCM_AXIS_X;
nResult = PmcmGetErcConfig(0, wAxis, PMCM_ERC_LOGIC, &wConfig);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned short config;
axis = PMCM_AXIS_X;
result = PmcmGetErcConfig(0, axis, PMCM_ERC_LOGIC, &config);
```

C#

```
int result;
ushort axis;
ushort config;
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.GetErcConfig(0, axis, Pmcm.PMCM_ERC_LOGIC, out config);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim config As Short
axis = PMCM_AXIS_X
result = PmcmGetErcConfig(0, axis, PMCM_ERC_LOGIC, config)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intConfig As Integer
intAxis = PMCM_AXIS_X
lngResult = PmcmGetErcConfig(0, intAxis, PMCM_ERC_LOGIC, intConfig)
```

GCC

```
int32_t result;
uint16_t axis;
uint16_t config;
axis = PMCM_AXIS_X;
result = PmcmGetErcConfig(0, axis, PMCM_ERC_LOGIC, &config);
```

PmcmGetEventFactor

機能

イベント発生要因の取得をします。

イベントについて

書式

```
INT PmcmGetEventFactor(
    WORD wID,
    PEVENTFACTORPMCM pEventFactor
);

typedef struct {
    int nResult;
    WORD wStop[2];
    WORD wState[2];
} EVENTFACTORPMCM, *PEVENTFACTORPMCM;
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pEventFactor

イベントの発生要因を格納するバッファへのポインタを指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PEVENTFACTORPMCM	EVENTFACTORPMCM*	out EVENTFACTORPMCM	EVENTFACTORPMCM	EVENTFACTORPMCM	PEVENTFACTORPMCM

nResult

結果が格納されます。
0ならば正常にイベントが発生しています。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	int	int	int	Integer	Long	int32_t

wStop[2]

停止イベント要因が格納されます。

bit	15	14	13	12	11	10	9	8
信号	-	-	-	-	-	-	-	-

bit	7	6	5	4	3	2	1	0
信号	-	-	SD	EMG	ALM	-EL	+EL	STOP

bit5 : SD入力ONによる減速停止時
bit4 : EMG入力ONによる停止時
bit3 : ALM入力ONによる停止時
bit2 : -EL入力ONによる停止時
bit1 : +EL入力ONによる停止時
bit0 : 自動停止時

各ビットの設定値	内容
0	イベント未発生
1	イベント発生

2軸の停止イベント要因が格納されます。
wStop[0] : X軸の停止イベント要因
wStop[1] : Y軸の停止イベント要因

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wState[2]

状態イベント要因が格納されます。

bit	15	14	13	12	11	10	9	8
信号	-	-	-	-	-	-	-	SD

bit	7	6	5	4	3	2	1	0
信号	ORG	LTC	CMP2	CMP1	DEC FIN	DEC STA	ACC FIN	ACC STA

bit8 : SD入力ON時
bit7 : ORG入力ONによるカウント値のラッチ時
bit6 : LTC入力によるカウント値のラッチ時

bit5 : コンパレータ（エンコーダ）条件成立時
bit4 : コンパレータ（出力パルス）条件成立時
bit3 : 減速終了時
bit2 : 減速開始時
bit1 : 加速終了時
bit0 : 加速開始時

各ビットの設定値	内容
0	イベント未発生
1	イベント発生

2軸の状態イベント要因が格納されます。
wState[0] : X軸の状態イベント要因
wState[1] : Y軸の状態イベント要因

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、イベント要因を取得します。

C/C++

```
int nResult;
EVENTFACTORPMCM EventFactor;
nResult = PmcmGetEventFactor(0, &EventFactor);
```

C++/CLI

```
int result;
EVENTFACTORPMCM eventFactor;
result = PmcmGetEventFactor(0, &eventFactor);
```

C#

```
int result;
Pmcm.EVENTFACTORPMCM eventFactor = new Pmcm.EVENTFACTORPMCM();
eventFactor.Initialize();
result = Pmcm.GetEventFactor(0, out eventFactor);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim eventFactor As EVENTFACTORPMCM
eventFactor.Initialize()
result = PmcmGetEventFactor(0, eventFactor)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim EventFactor As EVENTFACTORPMCM
lngResult = PmcmGetEventFactor(0, EventFactor)
```

GCC

```
int32_t result;
EVENTFACTORPMCM event_factor;
result = PmcmGetEventFactor(0, &event_factor);
```

PmcmGetEventMask

機能

イベントマスク設定の取得をします。

イベントについて

書式

```
INT PmcmGetEventMask(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    PWORD pwConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

取得する項目を指定します。

設定値	内容
PMCM_EVENT_STOP	停止イベント
PMCM_EVENT_STATE	状態イベント

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pwConfig

設定値を格納するバッファへのポインタを指定します。
取得する項目（wMode）によって値が異なります。

- wMode : PMCM_EVENT_STOP

bit	15	14	13	12	11	10	9	8
信号	-	-	-	-	-	-	-	-

bit	7	6	5	4	3	2	1	0
信号	-	-	SD	EMG	ALM	-EL	+EL	STOP

bit5 : SD入力ONによる減速停止時
bit4 : EMG入力ONによる停止時
bit3 : ALM入力ONによる停止時
bit2 : -EL入力ONによる停止時
bit1 : +EL入力ONによる停止時
bit0 : 自動停止時

各ビットの設定値	内容
0	イベント発生マスク
1	イベント発生許可

初期値 : H'0

- wMode : PMCM_EVENT_STATE

bit	15	14	13	12	11	10	9	8
信号	-	-	-	-	-	-	-	SD

bit	7	6	5	4	3	2	1	0
信号	ORG	LTC	CMP2	CMP1	DEC FIN	DEC STA	ACC FIN	ACC STA

bit8 : SD入力ON時
bit7 : ORG入力ONによるカウント値のラッチ時
bit6 : LTC入力によるカウント値のラッチ時
bit5 : コンパレータ（エンコーダ）条件成立時
bit4 : コンパレータ（出力パルス）条件成立時
bit3 : 減速終了時
bit2 : 減速開始時
bit1 : 加速終了時
bit0 : 加速開始時

各ビットの設定値	内容
0	イベント発生マスク
1	イベント発生許可

初期値 : H'0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PWORD	unsigned short*	out ushort	Short	Integer	uint16_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸の停止イベントマスクを取得します。

C/C++

```
int nResult;
WORD wAxis;
WORD wConfig;
wAxis = PMCM_AXIS_X;
nResult = PmcmGetEventMask(0, wAxis, PMCM_EVENT_STOP, &wConfig);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned short config;
axis = PMCM_AXIS_X;
result = PmcmGetEventMask(0, axis, PMCM_EVENT_STOP, &config);
```

C#

```
int result;
ushort axis;
ushort config;
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.GetEventMask(0, axis, Pmcm.PMCM_EVENT_STOP, out config);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim config As Short
axis = PMCM_AXIS_X
result = PmcmGetEventMask(0, axis, PMCM_EVENT_STOP, config)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intConfig As Integer
intAxis = PMCM_AXIS_X
lngResult = PmcmGetEventMask(0, intAxis, PMCM_EVENT_STOP, intConfig)
```

GCC

```
int32_t result;
uint16_t axis;
uint16_t config;
axis = PMCM_AXIS_X;
result = PmcmGetEventMask(0, axis, PMCM_EVENT_STOP, &config);
```

PmcmGetLatchConfig

機能

カウンタラッチ設定の取得をします。

カウンタについて

書式

```
INT PmcmGetLatchConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    PWORD pwConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

取得する項目を指定します。

設定値	内容
PMCM_COUNTER_LATCH_LTC	LTC信号によるラッチ（出力パルスカウンタ）
PMCM_COUNTER_LATCH_ORG	原点復帰によるラッチ（出力パルスカウンタ）
PMCM_COUNTER_CLEAR_LATCH	ラッチ後のクリア（出力パルスカウンタ）
PMCM_ENCODER_LATCH_LTC	LTC信号によるラッチ（エンコーダカウンタ）
PMCM_ENCODER_LATCH_ORG	原点復帰によるラッチ（エンコーダカウンタ）
PMCM_ENCODER_CLEAR_LATCH	ラッチ後のクリア（エンコーダカウンタ）

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pwConfig

設定値を格納するバッファへのポインタを指定します。
取得する項目（wMode）によって値が異なります。

- wMode : PMCM_COUNTER_LATCH_LTC、PMCM_ENCODER_LATCH_LTC

設定値	内容
0	LTC信号によりカウンタをラッチしません
1	LTC信号によりカウンタをラッチします

  初期値 : 0

- wMode : PMCM_COUNTER_LATCH_ORG、PMCM_ENCODER_LATCH_ORG

設定値	内容
0	原点復帰によりカウンタをラッチしません
1	原点復帰によりカウンタをラッチします

  初期値 : 0

- wMode : PMCM_COUNTER_CLEAR_LATCH、PMCM_ENCODER_CLEAR_LATCH

設定値	内容

設定値	内容
0	ラッチ後にカウンタをクリアしません
1	ラッチ後にカウンタをクリアします

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PWORD	unsigned short*	out ushort	Short	Integer	uint16_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸のLTC信号によるラッチ（出力パルスカウンタ）を取得します。

C/C++

```
int nResult;
WORD wAxis;
WORD wConfig;
wAxis = PMCM_AXIS_X;
nResult = PmcmGetLatchConfig(0, wAxis, PMCM_COUNTER_LATCH_LTC, &wConfig);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned short config;
axis = PMCM_AXIS_X;
result = PmcmGetLatchConfig(0, axis, PMCM_COUNTER_LATCH_LTC, &config);
```

C#

```
int result;
ushort axis;
ushort config;
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.GetLatchConfig(0, axis, Pmcm.PMCM_COUNTER_LATCH_LTC, out config);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim config As Short
axis = PMCM_AXIS_X
result = PmcmGetLatchConfig(0, axis, PMCM_COUNTER_LATCH_LTC, config)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intConfig As Integer
intAxis = PMCM_AXIS_X
lngResult = PmcmGetLatchConfig(0, intAxis, PMCM_COUNTER_LATCH_LTC, intConfig)
```

GCC

```
int32_t result;
uint16_t axis;
uint16_t config;
axis = PMCM_AXIS_X;
result = PmcmGetLatchConfig(0, axis, PMCM_COUNTER_LATCH_LTC, &config);
```

PmcmGetLatchCounter

機能

出力パルスカウンタのラッチカウンタ値の取得をします。

書式

```
INT PmcmGetLatchCounter(
    WORD wID,
    WORD wAxis,
    PLONG plData
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

取得する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

plData

ラッチカウンタ値を格納するバッファへのポインタを指定します。

> バッファは2軸分用意してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PLONG	long*	int	Integer	Long	int32_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

複数軸同時に取得しない場合でも、ラッチカウンタ値バッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
long lData[2]; ←2軸分用意
lData[0] ←X軸のラッチカウンタ値
lData[1] ←Y軸のラッチカウンタ値
```

使用例

IDが0のボードの、X軸とY軸の出力パルスカウンタのラッチカウンタ値を取得します。

C/C++

```
int nResult;
WORD wAxis;
long lData[2];
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmGetLatchCounter(0, wAxis, lData);
```

C++/CLI

```
int result;
unsigned short axis;
long data[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmGetLatchCounter(0, axis, data);
```

C#

```
int result;
ushort axis;
int[] data = new int[2];
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.GetLatchCounter(0, axis, data);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim data(1) As Integer
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmGetLatchCounter(0, axis, data)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim lngData(1) As Long
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmGetLatchCounter(0, intAxis, lngData(0))
```

GCC

```
int32_t result;
uint16_t axis;
int32_t data[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmGetLatchCounter(0, axis, data);
```

PmcmGetLatchEncoder

機能

エンコーダカウンタのラッチカウンタ値の取得をします。

書式

```
INT PmcmGetLatchEncoder(
    WORD wID,
    WORD wAxis,
    PLONG plData
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

取得する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

plData

ラッチカウンタ値を格納するバッファへのポインタを指定します。

> バッファは2軸分用意してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PLONG	long*	int	Integer	Long	int32_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

複数軸同時に取得しない場合でも、ラッチカウンタ値バッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
long lData[2]; ←2軸分用意
lData[0] ←X軸のラッチカウンタ値
lData[1] ←Y軸のラッチカウンタ値
```

使用例

IDが0のボードの、X軸とY軸のエンコーダカウンタのラッチカウンタ値を取得します。

C/C++

```
int nResult;
WORD wAxis;
long lData[2];
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmGetLatchEncoder(0, wAxis, lData);
```

C++/CLI

```
int result;
unsigned short axis;
long data[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmGetLatchEncoder(0, axis, data);
```

C#

```
int result;
ushort axis;
int[] data = new int[2];
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.GetLatchEncoder(0, axis, data);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim data(1) As Integer
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmGetLatchEncoder(0, axis, data)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim lngData(1) As Long
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmGetLatchEncoder(0, intAxis, lngData(0))
```

GCC

```
int32_t result;
uint16_t axis;
int32_t data[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmGetLatchEncoder(0, axis, data);
```

PmcmGetMotion

機能

動作パラメータを取得します。

連続動作について
原点復帰動作について
位置決め動作について

書式

```
INT PmcmGetMotion(
    WORD wID,
    WORD wAxis,
    PMOTIONPMCM pMotion
);

typedef struct {
    WORD wMoveMode;
    WORD wStartMode;
    FLOAT fSpeedRate;
    WORD wAccDecMode;
    FLOAT fLowSpeed;
    FLOAT fSpeed;
    WORD wAccTime;
    WORD wDecTime;
    FLOAT fSAccSpeed;
    FLOAT fSDecSpeed;
    LONG lSlowdown;
    LONG lStep;
    BOOL bAbsolutePtp;
} MOTIONPMCM, *PMOTIONPMCM;
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。複数の軸を指定することができます。

設定値	内容

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pMotion

動作パラメータを格納するバッファへのポインタを指定します。

> バッファは2軸分用意してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PMOTIONPMCM	MOTIONPMCM*	MOTIONPMCM	MOTIONPMCM	MOTIONPMCM	PMOTIONPMCM

wMoveMode

動作モード

設定値	内容
PMCM_JOG	連続動作
PMCM_ORG	原点復帰動作
PMCM_PTP	位置決め動作

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wStartMode

起動モード

設定値	内容
PMCM_CONST	定速起動

設定値	内容
PMCM_CONST_DEC	定速－減速起動
PMCM_ACC_DEC	加減速起動

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fSpeedRate

速度倍率 : 0.3 ～ 600

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

wAccDecMode

加減速モード

設定値	内容
PMCM_ACC_LINEAR	直線加減速
PMCM_ACC_SCURVE	S字加減速

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fLowSpeed

起動速度 : 0.3 ～ 9829800[pps]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

fSpeed

移動速度 : 0.3 ～ 9829800[pps]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

wAccTime

加速時間[msec]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wDecTime

減速時間[msec]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fSAccSpeed

加速S字区間 : 0, 0.3 ～ 4914600[pps]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

fSDecSpeed

減速S字区間 : 0, 0.3 ～ 4914600[pps]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

lSlowdown

スローダウンポイント

設定値	内容
-1	スローダウンポイントは自動設定
0 ～ 16777215	スローダウンポイントはマニュアル設定

連続動作、原点復帰動作の時は0が設定されます

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

lStep

移動パルス数、移動方向

動作モード	設定値
PMCM_JOG PMCM_ORG	PMCM_DIR_CW : +方向 PMCM_DIR_CCW : -方向
PMCM_PTP	-134217728 ～ +134217727

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

bAbsolutePtp

絶対座標指定

設定値	内容
0	移動パルス数の設定値は移動量
1	移動パルス数の設定値は絶対座標

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	BOOL	long	int	Integer	Long	int32_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

複数軸同時に取得しない場合でも、動作パラメータバッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
MOTIONPMCM Motion[2]; ←2軸分用意
Motion[0] ←X軸の動作パラメータ
Motion[1] ←Y軸の動作パラメータ
```

使用例

IDが0のボードの、X軸の動作パラメータを取得します。

C/C++

```
int nResult;
WORD wAxis;
MOTIONPMCM Motion[2];
wAxis = PMCM_AXIS_X;
nResult = PmcmGetMotion(0, wAxis, Motion);
```

C++/CLI

```
int result;
unsigned short axis;
MOTIONPMCM motion[2];
axis = PMCM_AXIS_X;
result = PmcmGetMotion(0, axis, motion);
```

C#

```
int result;
ushort axis;
Pmcm.MOTIONPMCM[] motion = new Pmcm.MOTIONPMCM[2];
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.GetMotion(0, axis, motion);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim motion(1) As MOTIONPMCM
axis = PMCM_AXIS_X
result = PmcmGetMotion(0, axis, motion)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim Motion(1) As MOTIONPMCM
```

```
intAxis = PMCM_AXIS_X
lngResult = PmcmGetMotion(0, intAxis, Motion(0))
```

GCC

```
int32_t result;
uint16_t axis;
MOTIONPMCM motion[2];
axis = PMCM_AXIS_X;
result = PmcmGetMotion(0, axis, motion);
```

PmcmGetMotionCp

機能

CPの動作パラメータを取得します。

CP動作について

書式

```
INT PmcmGetMotionCp(
    WORD wID,
    WORD wAxis,
    PMOTIONPMCM pMotion,
    PWORD pwCount
);

typedef struct {
    WORD wMoveMode;
    WORD wStartMode;
    FLOAT fSpeedRate;
    WORD wAccDecMode;
    FLOAT fLowSpeed;
    FLOAT fSpeed;
    WORD wAccTime;
    WORD wDecTime;
    FLOAT fSAccSpeed;
    FLOAT fSDecSpeed;
    LONG lSlowdown;
    LONG lStep;
    BOOL bAbsolutePtp;
} MOTIONPMCM, *PMOTIONPMCM;
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸

設定値	内容
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pMotion

動作パラメータを格納するバッファへのポインタを指定します。
バッファは取得する数分用意してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PMOTIONPMCM	MOTIONPMCM*	MOTIONPMCM	MOTIONPMCM	MOTIONPMCM	PMOTIONPMCM

wMoveMode

動作モード

設定値	内容
PMCM_PTP	位置決め動作

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wStartMode

起動モード

設定値	内容
PMCM_CONST	定速起動
PMCM_CONST_DEC	定速－減速起動
PMCM_ACC_DEC	加減速起動

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fSpeedRate

速度倍率 : 0.3 ～ 600

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

wAccDecMode

加減速モード

設定値	内容
PMCM_ACC_LINEAR	直線加減速
PMCM_ACC_SCURVE	S字加減速

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fLowSpeed

起動速度 : 0.3 ～ 9829800[pps]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

fSpeed

移動速度 : 0.3 ～ 9829800[pps]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

wAccTime

加速時間[msec]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wDecTime

減速時間[msec]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fSAccSpeed

加速S字区間 : 0, 0.3 ～ 4914600[pps]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

fSDecSpeed

減速S字区間 : 0, 0.3 ～ 4914600[pps]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

lSlowdown

スローダウンポイント

設定値	内容
-1	スローダウンポイントは自動設定
0 ～ 16777215	スローダウンポイントはマニュアル設定

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

lStep

移動パルス数、移動方向 : -134217728 ～ +134217727

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

bAbsolutePtp

絶対座標指定

設定値	内容
0	移動パルス数の設定値は移動量

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	BOOL	long	int	Integer	Long	int32_t

pwCount

取得する動作パラメータ数が格納されているバッファへのポインタ
設定範囲は1 ～ 96

関数実行後は実際に取得した動作パラメータ数が格納されます。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PWORD	unsigned short*	ref ushort	Short	Integer	uint16_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸のCP動作パラメータを2つ取得します。

C/C++

```
int nResult;
WORD wAxis;
MOTIONPMCM Motion[2];
WORD wCount;
```

```
wAxis = PMCM_AXIS_X;
wCount = 2;
nResult = PmcmGetMotionCp(0, wAxis, Motion, &wCount);
```

C++/CLI

```
int result;
unsigned short axis;
MOTIONPMCM motion[2];
unsigned short count;
axis = PMCM_AXIS_X;
count = 2;
result = PmcmGetMotionCp(0, axis, motion, &count);
```

C#

```
int result;
ushort axis;
Pmcm.MOTIONPMCM[] motion = new Pmcm.MOTIONPMCM[2];
ushort count;
axis = Pmcm.PMCM_AXIS_X;
count = 2;
result = Pmcm.GetMotionCp(0, axis, motion, ref count);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim motion(1) As MOTIONPMCM
Dim count As Short
axis = PMCM_AXIS_X
count = 2
result = PmcmGetMotionCp(0, axis, motion, count)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim Motion(1) As MOTIONPMCM
Dim intCount As Integer
intAxis = PMCM_AXIS_X
intCount = 2
lngResult = PmcmGetMotionCp(0, intAxis, Motion(0), intCount)
```

GCC

```
int32_t result;
uint16_t axis;
MOTIONPMCM motion[2];
uint16_t count;
axis = PMCM_AXIS_X;
count = 2;
result = PmcmGetMotionCp(0, axis, motion, &count);
```

PmcmGetMotionLine

機能

直線補間動作パラメータを取得します。

直線補間動作について

書式

```
INT PmcmGetMotionLine(
    WORD wID,
    WORD wAxis,
    PMOTIONLINEPMCM pMotionLine
);

typedef struct {
    WORD wStartMode;
    FLOAT fSpeedRate;
    WORD wAccDecMode;
    FLOAT fLowSpeed;
    FLOAT fSpeed;
    WORD wAccTime;
    WORD wDecTime;
    FLOAT fSAccSpeed;
    FLOAT fSDecSpeed;
    LONG lSlowdown;
    LONG lStep[2];
    BOOL bAbsolute[2];
} MOTIONLINEPMCM, *PMOTIONLINEPMCM;
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。2軸以上指定してください。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

PMC-M2C-UのwAxisに設定する値はPMCM_AXIS_X + PMCM_AXIS_Yで固定です

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pMotionLine

動作パラメータを格納するバッファへのポインタを指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PMOTIONLINEPMCM	MOTIONLINEPMCM*	out MOTIONLINEPMCM	MOTIONLINEPMCM	MOTIONLINEPMCM	PMOTIONLINEPMCM

wStartMode

起動モード

設定値	内容
PMCM_CONST	定速起動
PMCM_CONST_DEC	定速－減速起動
PMCM_ACC_DEC	加減速起動

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fSpeedRate

速度倍率 : 0.3 ～ 600

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

wAccDecMode

加減速モード

設定値	内容
PMCM_ACC_LINEAR	直線加減速

設定値	内容
PMCM_ACC_SCURVE	S字加減速

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fLowSpeed

起動速度 : 0.3 ～ 9829800[pps]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

fSpeed

移動速度 : 0.3 ～ 9829800[pps]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

wAccTime

加速時間[msec]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wDecTime

減速時間[msec]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fSAccSpeed

加速S字区間 : 0, 0.3 ～ 4914600[pps]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

fSDecSpeed

減速S字区間 : 0, 0.3 ～ 4914600[pps]

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

lSlowdown

スローダウンポイント

設定値	内容
-1	スローダウンポイントは自動設定
0 ～ 16777215	スローダウンポイントはマニュアル設定

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

lStep

移動パルス数 : -134217728 ～ +134217727

2軸の移動パルス数を取得します。
lStep[0] : X軸の移動パルス数
lStep[1] : Y軸の移動パルス数

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

bAbsolute

絶対座標指定

設定値	内容
0	移動パルス数の設定値は移動量
1	移動パルス数の設定値は絶対座標

2軸の絶対座標指定を取得します。
bAbsolute[0] : X軸の絶対座標指定
bAbsolute[1] : Y軸の絶対座標指定

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	BOOL	long	int	Integer	Long	int32_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、直線補間動作パラメータを取得します。

C/C++

```
int nResult;
WORD wAxis;
MOTIONLINEPMCM MotionLine;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmGetMotionLine(0, wAxis, &MotionLine);
```

C++/CLI

```
int result;
unsigned short axis;
MOTIONLINEPMCM motionLine;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmGetMotionLine(0, axis, &motionLine);
```

C#

```
int result;
ushort axis;
Pmcm.MOTIONLINEPMCM motionLine = new Pmcm.MOTIONLINEPMCM();
motionLine.Initialize();
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.GetMotionLine(0, axis, out motionLine);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim motionLine As MOTIONLINEPMCM
motionLine.Initialize()
```

```
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmGetMotionLine(0, axis, motionLine)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim MotionLine As MOTIONLINEPMCM
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmGetMotionLine(0, intAxis, MotionLine)
```

GCC

```
int32_t result;
uint16_t axis;
MOTIONLINEPMCM motion_line;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmGetMotionLine(0, axis, &motion_line);
```

PmcmGetPulseConfig

機能

パルス出力設定の取得をします。

パルス出力について

書式

```
INT PmcmGetPulseConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    PWORD pwConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

取得する項目を指定します。

設定値	内容
PMCM_PULSE_OUT	パルス出力モード
PMCM_DIR_WAIT	方向変化時の遅延
PMCM_FIN_TIMING	動作完了タイミング
PMCM_FH_REVISE	移動速度補正（三角駆動回避）

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

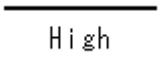
pwConfig

設定値を格納するバッファへのポインタを指定します。
取得する項目（wMode）によって値が異なります。

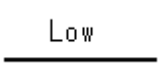
- wMode : PMCM_PULSE_OUT
  - 0

    共通パルスモード
    OUT：負論理
    DIR：（＋方向）High、（―方向）Low

    CW（＋）方向動作時

OUT出力	DIR出力
	

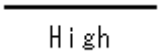
    CCW（―）方向動作時

OUT出力	DIR出力
	

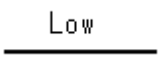
  - 1

    共通パルスモード
    OUT：正論理
    DIR：（＋方向）High、（―方向）Low

    CW（＋）方向動作時

OUT出力	DIR出力
	

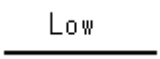
CCW（—）方向動作時

OUT出力	DIR出力
	

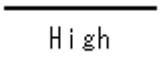
- 2

  共通パルスモード
  OUT：負論理
  DIR：（＋方向）Low、（—方向）High

  CW（＋）方向動作時

OUT出力	DIR出力
	

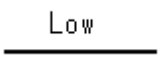
  CCW（—）方向動作時

OUT出力	DIR出力
	

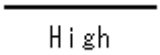
- 3

  共通パルスモード
  OUT：正論理
  DIR：（＋方向）Low、（—方向）High

  CW（＋）方向動作時

OUT出力	DIR出力
	

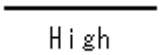
  CCW（—）方向動作時

OUT出力	DIR出力
	

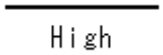
- 4

  2パルスモード
  負論理

  CW（＋）方向動作時

OUT出力	DIR出力
	

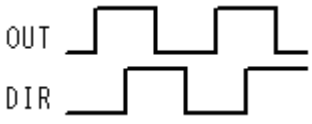
  CCW（―）方向動作時

OUT出力	DIR出力
	

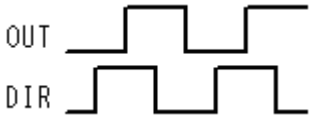
- 5

  90度位相差モード（4逓倍）
  OUT：進みパルス
  DIR：遅れパルス

  CW（＋）方向動作時

OUT/DIR出力


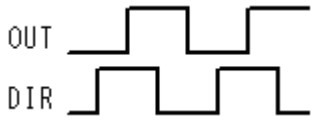
  CCW（―）方向動作時

OUT/DIR出力


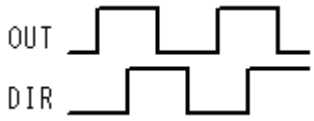
- 6

  90度位相差モード（4逓倍）
  OUT：遅れパルス
  DIR：進みパルス

  CW（＋）方向動作時

OUT/DIR出力


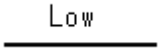
  CCW（―）方向動作時

OUT/DIR出力


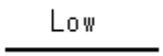
- 7

  2パルスモード
  正論理

  CW（＋）方向動作時

OUT出力	DIR出力
	

  CCW（―）方向動作時

OUT出力	DIR出力
	

初期値 : 7

- wMode : PMCM_DIR_WAIT

  モーターの回転方向が変わるときにパルス出力を遅延させることができます。

設定値	内容
0	方向変化時に200μsecの遅延あり
1	方向変化時に遅延なし

> PMCM_PULSE_OUTの設定が0～3の場合のみ有効です

初期値 : 0

- wMode : PMCM_FIN_TIMING

設定値	内容
0	最終パルスの周期完了時
1	最終パルスの出力OFF時
2	INP信号入力時

初期値 : 0

- wMode : PMCM_FH_REVISE

設定値	内容
0	移動速度補正OFF
1	移動速度補正ON

初期値 : 1

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PWORD	unsigned short*	out ushort	Short	Integer	uint16_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸のパルス出力設定を取得します。

C/C++

```
int nResult;
WORD wAxis;
WORD wConfig;
wAxis = PMCM_AXIS_X;
nResult = PmcmGetPulseConfig(0, wAxis, PMCM_PULSE_OUT, &wConfig);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned short config;
axis = PMCM_AXIS_X;
result = PmcmGetPulseConfig(0, axis, PMCM_PULSE_OUT, &config);
```

C#

```
int result;
ushort axis;
ushort config;
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.GetPulseConfig(0, axis, Pmcm.PMCM_PULSE_OUT, out config);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim config As Short
axis = PMCM_AXIS_X
result = PmcmGetPulseConfig(0, axis, PMCM_PULSE_OUT, config)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intConfig As Integer
intAxis = PMCM_AXIS_X
lngResult = PmcmGetPulseConfig(0, intAxis, PMCM_PULSE_OUT, intConfig)
```

GCC

```
int32_t result;
uint16_t axis;
uint16_t config;
```

```
axis = PMCM_AXIS_X;
result = PmcmGetPulseConfig(0, axis, PMCM_PULSE_OUT, &config);
```

PmcmGetSensorConfig

機能

センサ設定の取得をします。

書式

```
INT PmcmGetSensorConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    PWORD pwConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

取得する項目を指定します。

設定値	内容

設定値	内容
PMCM_LOGIC	入力論理（センサ極性）
PMCM_EL_FUNC	EL信号入力ON時の処理
PMCM_SD_FUNC	SD信号入力ON時の処理
PMCM_SD_ACTIVE	SD信号入力方法
PMCM_ORG_FUNC	原点復帰方法
PMCM_EZ_FUNC	Z信号入力論理
PMCM_EZ_COUNT	Z信号カウント数
PMCM_ALM_FUNC	ALM信号入力ON時の処理
PMCM_LTC_FUNC	LTC信号入力論理
PMCM_PCS_FUNC	PCS信号入力の処理
PMCM_SENS_FILTER	入力フィルタの設定

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pwConfig

設定値を格納するバッファへのポインタを指定します。
取得する項目（wMode）によって値が異なります。

- wMode : PMCM_LOGIC

  各センサの入力論理

bit	15	14	13	12	11	10	9	8
信号	-	-	-	-	-	-	-	-

bit	7	6	5	4	3	2	1	0
信号	-	-	PCS	INP	ALM	ORG	SD	±EL

各ビットの設定値	内容
0	オフで検知（正論理）
1	オンで検知（負論理）

> PCSはX軸のみ。Y軸のPCSは常に1が読み出されます

初期値 : H'0（X軸）、H'20（Y軸）

- wMode : PMCM_EL_FUNC

  +EL、-EL信号がONになった時の停止方法

設定値	内容
0	即停止
1	減速停止

初期値 : 0

- wMode : PMCM_SD_FUNC

  SD信号がONになった時の減速方法

設定値	内容
0	減速のみ
1	減速停止
2	SD無効

初期値 : 0

- wMode : PMCM_SD_ACTIVE

  SD信号の入力方法

設定値	内容
0	レベル入力
1	ラッチ入力

初期値 : 0

- wMode : PMCM_ORG_FUNC

  ORG、Z信号を使用した原点復帰方法

設定値	内容
0	ORG信号のみ使用
1	ORG信号とZ信号を使用

初期値 : 0

- wMode : PMCM_EZ_FUNC

  Z信号の入力論理

設定値	内容
0	立ち下がりエッジ
1	立ち上がりエッジ

初期値 : 0

- wMode : PMCM_EZ_COUNT

  原点復帰完了条件のZ信号カウント数 : 1 ～ 16

  初期値 : 1

- wMode : PMCM_ALM_FUNC

  ALM信号がONになった時の停止方法

設定値	内容
0	即停止
1	減速停止

初期値 : 0

- wMode : PMCM_LTC_FUNC

  LTC信号の入力論理

設定値	内容

設定値	内容
0	LTC信号の立ち下がりエッジでラッチ
1	LTC信号の立ち上がりエッジでラッチ

初期値 : 0

- wMode : PMCM_PCS_FUNC

  PCS信号の機能

設定値	内容
0	PCS信号として使用しない
1	PCS信号として使用する（位置のオーバーライド）
2	自軸のみに有効な外部スタート/同時スタート信号

初期値 : 0

- wMode : PMCM_SENS_FILTER

設定値	内容
0	フィルタなし
1	3.2μsec以下のパルス幅は無視
2	25μsec以下のパルス幅は無視
3	100μsec以下のパルス幅は無視
4	1.6msec以下のパルス幅は無視

+EL、-EL、SD、ORG、ALM、INPにフィルタを挿入します。
フィルタを挿入すると、指定パルス幅以下の信号は無視されます。

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PWORD	unsigned short*	out ushort	Short	Integer	uint16_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸の入力論理（センサ極性）設定を取得します。

C/C++

```
int nResult;
WORD wAxis;
WORD wConfig;
wAxis = PMCM_AXIS_X;
nResult = PmcmGetSensorConfig(0, wAxis, PMCM_LOGIC, &wConfig);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned short config;
axis = PMCM_AXIS_X;
result = PmcmGetSensorConfig(0, axis, PMCM_LOGIC, &config);
```

C#

```
int result;
ushort axis;
ushort config;
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.GetSensorConfig(0, axis, Pmcm.PMCM_LOGIC, out config);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim config As Short
axis = PMCM_AXIS_X
result = PmcmGetSensorConfig(0, axis, PMCM_LOGIC, config)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intConfig As Integer
intAxis = PMCM_AXIS_X
lngResult = PmcmGetSensorConfig(0, intAxis, PMCM_LOGIC, intConfig)
```

GCC

```
int32_t result;
uint16_t axis;
uint16_t config;
axis = PMCM_AXIS_X;
result = PmcmGetSensorConfig(0, axis, PMCM_LOGIC, &config);
```

PmcmGetStatus

機能

各ステータスを取得します。

書式

```
INT PmcmGetStatus(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    PWORD pwStatus
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

取得する項目を指定します。

設定値	内容

設定値	内容
PMCM_BUSY	動作状態
PMCM_SIG_STATUS	制御信号入力／出力状態

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pwStatus

設定値を格納するバッファへのポインタを指定します。
取得する項目（wMode）によって値が異なります。

- wMode : PMCM_BUSY

値	内容
0	停止中
1	内部同期信号待ち
2	他軸の停止待ち
3	ERCタイマー完了待ち
4	方向変化タイマー完了待ち
5	起動速度で動作中
6	移動速度で動作中
7	加速中
8	減速中
9	INP信号入力待ち状態
10	その他（状態遷移中です。再取得してください）

- wMode : PMCM_SIG_STATUS

bit	15	14	13	12	11	10	9	8

bit	15	14	13	12	11	10	9	8
信号	ERC	-	-	-	-	-	EMG	Z

bit	7	6	5	4	3	2	1	0
信号	PCS	LTC	INP	ALM	ORG	SD	-EL	+EL

各ビットの値	内容
0	OFF
1	ON

> PCSはX軸のみ

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PWORD	unsigned short*	out ushort	Short	Integer	uint16_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸の制御信号入力／出力状態を取得します。

C/C++

```
int nResult;
WORD wAxis;
WORD wStatus;
wAxis = PMCM_AXIS_X;
nResult = PmcmGetStatus(0, wAxis, PMCM_SIG_STATUS, &wStatus);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned short status;
axis = PMCM_AXIS_X;
result = PmcmGetStatus(0, axis, PMCM_SIG_STATUS, &status);
```

C#

```
int result;
ushort axis;
ushort status;
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.GetStatus(0, axis, Pmcm.PMCM_SIG_STATUS, out status);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim status As Short
axis = PMCM_AXIS_X
result = PmcmGetStatus(0, axis, PMCM_SIG_STATUS, status)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intStatus As Integer
intAxis = PMCM_AXIS_X
lngResult = PmcmGetStatus(0, intAxis, PMCM_SIG_STATUS, intStatus)
```

GCC

```
int32_t result;
uint16_t axis;
uint16_t status;
axis = PMCM_AXIS_X;
result = PmcmGetStatus(0, axis, PMCM_SIG_STATUS, &status);
```

PmcmGetSynchroConfig

機能

起動・停止条件設定の取得をします。

書式

```
INT PmcmGetSynchroConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    PWORD pwConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

設定する項目を指定します。

設定値	内容

設定値	内容
PMCM_SYNC_START_MODE	起動条件
PMCM_SYNC_START	他軸の停止によるスタート条件
PMCM_SYNC_SIG_START	内部同期信号によるスタート条件
PMCM_SYNC_STOP_MODE	停止条件

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pwConfig

設定値を格納するバッファへのポインタを指定します。
取得する項目（wMode）によって値が異なります。

- wMode : PMCM_SYNC_START_MODE

設定値	内容
0	スタート条件なし（即スタート）
1	PCS入力によるスタート
2	内部同期信号によるスタート
3	他軸の停止によるスタート

  初期値 : 0

- wMode : PMCM_SYNC_START

  停止確認する軸の設定

設定値	内容
PMCM_AXIS_X	X軸の停止でスタート
PMCM_AXIS_Y	Y軸の停止でスタート

  初期値 : 0（設定軸なし）

- wMode : PMCM_SYNC_SIG_START

  同期信号の設定

設定値	内容
0	X軸が出力した内部同期信号でスタート
1	Y軸が出力した内部同期信号でスタート

  初期値 : 0

- wMode : PMCM_SYNC_STOP_MODE

  他軸の異常停止による停止の設定

設定値	内容
0	他軸の異常停止で停止しない
1	他軸の異常停止で停止する

  初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PWORD	unsigned short*	out ushort	Short	Integer	uint16_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸の起動条件を取得します。

C/C++

```
int nResult;
WORD wAxis;
WORD wConfig;
wAxis = PMCM_AXIS_X;
nResult = PmcmGetSynchroConfig(0, wAxis, PMCM_SYNC_START_MODE, &wConfig);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned short config;
axis = PMCM_AXIS_X;
result = PmcmGetSynchroConfig(0, axis, PMCM_SYNC_START_MODE, &config);
```

C#

```
int result;
ushort axis;
ushort config;
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.GetSynchroConfig(0, axis, Pmcm.PMCM_SYNC_START_MODE, out config);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim config As Short
axis = PMCM_AXIS_X
result = PmcmGetSynchroConfig(0, axis, PMCM_SYNC_START_MODE, config)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intConfig As Integer
intAxis = PMCM_AXIS_X
lngResult = PmcmGetSynchroConfig(0, intAxis, PMCM_SYNC_START_MODE, intConfig)
```

GCC

```
int32_t result;
uint16_t axis;
uint16_t config;
axis = PMCM_AXIS_X;
result = PmcmGetSynchroConfig(0, axis, PMCM_SYNC_START_MODE, &config);
```

PmcmGetSynchroOut

機能

内部同期信号出力設定の取得をします。

書式

```
INT PmcmGetSynchroOut(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    PWORD pwConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定を取得する軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

取得する項目を指定します。

設定値	内容

設定値	内容
PMCM_SYNC_OUT_MODE	内部同期信号出力タイミング

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pwConfig

設定値を格納するバッファへのポインタを指定します。

- wMode : PMCM_SYNC_OUT_MODE

設定値	内容
0	内部同期信号出力なし
1	出力パルスコンパレータ条件成立時
2	エンコーダコンパレータ条件成立時
3	加速開始時
4	加速完了時
5	減速開始時
6	減速完了時

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PWORD	unsigned short*	out ushort	Short	Integer	uint16_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸内部同期信号出力タイミングを取得します。

C/C++

```
int nResult;
WORD wAxis;
WORD wConfig;
wAxis = PMCM_AXIS_X;
nResult = PmcmGetSynchroOut(0, wAxis, PMCM_SYNC_OUT_MODE, &wConfig);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned short config;
axis = PMCM_AXIS_X;
result = PmcmGetSynchroOut(0, axis, PMCM_SYNC_OUT_MODE, &config);
```

C#

```
int result;
ushort axis;
ushort config;
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.GetSynchroOut(0, axis, Pmcm.PMCM_SYNC_OUT_MODE, out config);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim config As Short
axis = PMCM_AXIS_X
result = PmcmGetSynchroOut(0, axis, PMCM_SYNC_OUT_MODE, config)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intConfig As Integer
intAxis = PMCM_AXIS_X
lngResult = PmcmGetSynchroOut(0, intAxis, PMCM_SYNC_OUT_MODE, intConfig)
```

GCC

```
int32_t result;
uint16_t axis;
uint16_t config;
axis = PMCM_AXIS_X;
result = PmcmGetSynchroOut(0, axis, PMCM_SYNC_OUT_MODE, &config);
```

PmcmSetComparatorConfig

機能

コンパレータの設定をします。

コンパレータについて

書式

```
INT PmcmSetComparatorConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    PCOMPPMCM pComp
);

typedef struct {
    WORD wConfig;
    LONG lCount;
} COMPPMCM, *PCOMPPMCM;
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

設定する項目を指定します。

設定値	内容
PMCM_COMP_COUNTER	出力パルスコンパレータの比較条件
PMCM_COMP_ENCODER	エンコーダコンパレータの比較条件

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pComp

設定パラメータが格納されているバッファへのポインタを指定します。

> バッファは2軸分用意してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PCOMPPMCM	COMPPMCM*	COMPPMCM	COMPPMCM	COMPPMCM	PCOMPPMCM

wConfig

コンパレータ機能設定
設定する項目（wMode）によって値が異なります。

- wMode : PMCM_COMP_COUNTER

設定値	内容
0	コンパレータ機能OFF
1	設定値 = 出力パルスカウンタ
2	設定値 > 出力パルスカウンタ
3	設定値 < 出力パルスカウンタ

  初期値 : 0

- wMode : PMCM_COMP_ENCODER

設定値	内容

設定値	内容
0	コンパレータ機能OFF
1	設定値 = エンコーダカウンタ
2	設定値 > エンコーダカウンタ
3	設定値 < エンコーダカウンタ

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

lCount

コンパレータの設定値
設定範囲は-134217728 ～ +134217727

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

複数軸同時に設定しない場合でも、設定パラメータバッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
COMPPMCM Comp[2]; ←2軸分用意
Comp[0] ←X軸の設定パラメータ
Comp[1] ←Y軸の設定パラメータ
```

使用例

IDが0のボードの、X軸の出力パルスコンパレータをOFFに設定します。

C/C++

```
int nResult;
WORD wAxis;
COMPPMCM Comp[2];
wAxis = PMCM_AXIS_X;
Comp[0].wConfig = 0;
Comp[0].lCount = 0;
nResult = PmcmSetComparatorConfig(0, wAxis, PMCM_COMP_COUNTER, Comp);
```

C++/CLI

```
int result;
unsigned short axis;
COMPPMCM comp[2];
axis = PMCM_AXIS_X;
comp[0].wConfig = 0;
comp[0].lCount = 0;
result = PmcmSetComparatorConfig(0, axis, PMCM_COMP_COUNTER, comp);
```

C#

```
int result;
ushort axis;
Pmcm.COMPPMCM[] comp = new Pmcm.COMPPMCM[2];
axis = Pmcm.PMCM_AXIS_X;
comp[0].wConfig = 0;
comp[0].lCount = 0;
result = Pmcm.SetComparatorConfig(0, axis, Pmcm.PMCM_COMP_COUNTER, comp);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim comp(1) As COMPPMCM
axis = PMCM_AXIS_X
comp(0).wConfig = 0
comp(0).lCount = 0
result = PmcmSetComparatorConfig(0, axis, PMCM_COMP_COUNTER, comp)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim Comp(1) As COMPPMCM
intAxis = PMCM_AXIS_X
Comp(0).wConfig = 0
Comp(0).lCount = 0
lngResult = PmcmSetComparatorConfig(0, intAxis, PMCM_COMP_COUNTER, Comp(0))
```

GCC

```
int32_t result;
uint16_t axis;
COMPPMCM comp[2];
axis = PMCM_AXIS_X;
comp[0].wConfig = 0;
comp[0].lCount = 0;
result = PmcmSetComparatorConfig(0, axis, PMCM_COMP_COUNTER, comp);
```

PmcmSetCounter

機能

出力パルスカウンタ値の設定をします。

書式

```
INT PmcmSetCounter(
    WORD wID,
    WORD wAxis,
    PLONG plData
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

plData

カウンタ値が格納されているバッファへのポインタを指定します。
設定範囲は-134217728 ～ +134217727

> バッファは2軸分用意してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PLONG	long*	int	Integer	Long	int32_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

複数軸同時に設定しない場合でも、カウンタ値バッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
long lData[2]; ←2軸分用意
lData[0] ←X軸のカウンタ値
lData[1] ←Y軸のカウンタ値
```

使用例

IDが0のボードの、X軸の出力パルスカウンタ値を1000、Y軸の出力パルスカウンタ値を-1000に設定します。

C/C++

```
int nResult;
WORD wAxis;
long lData[2];
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
lData[0] = 1000;
lData[1] = -1000;
nResult = PmcmSetCounter(0, wAxis, lData);
```

C++/CLI

```
int result;
unsigned short axis;
long data[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
data[0] = 1000;
data[1] = -1000;
result = PmcmSetCounter(0, axis, data);
```

C#

```
int result;
ushort axis;
int[] data = new int[2];
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
data[0] = 1000;
data[1] = -1000;
result = Pmcm.SetCounter(0, axis, data);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim data(1) As Integer
axis = PMCM_AXIS_X + PMCM_AXIS_Y
data(0) = 1000
data(1) = -1000
result = PmcmSetCounter(0, axis, data)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim lngData(1) As Long
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngData(0) = 1000
lngData(1) = -1000
lngResult = PmcmSetCounter(0, intAxis, lngData(0))
```

GCC

```
int32_t result;
uint16_t axis;
int32_t data[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
data[0] = 1000;
data[1] = -1000;
result = PmcmSetCounter(0, axis, data);
```

PmcmSetEncoder

機能

エンコーダカウンタ値の設定をします。

書式

```
INT PmcmSetEncoder(
    WORD wID,
    WORD wAxis,
    PLONG plData
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

plData

カウンタ値が格納されているバッファへのポインタを指定します。
設定範囲は-134217728 ～ +134217727

> バッファは2軸分用意してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PLONG	long*	int	Integer	Long	int32_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

複数軸同時に設定しない場合でも、カウンタ値バッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
long lData[2]; ←2軸分用意
lData[0] ←X軸のカウンタ値
lData[1] ←Y軸のカウンタ値
```

使用例

IDが0のボードの、X軸のエンコーダカウンタ値を1000、Y軸のエンコーダカウンタ値を-1000に設定します。

C/C++

```
int nResult;
WORD wAxis;
long lData[2];
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
lData[0] = 1000;
lData[1] = -1000;
nResult = PmcmSetEncoder(0, wAxis, lData);
```

C++/CLI

```
int result;
unsigned short axis;
long data[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
data[0] = 1000;
data[1] = -1000;
result = PmcmSetEncoder(0, axis, data);
```

C#

```
int result;
ushort axis;
int[] data = new int[2];
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
data[0] = 1000;
data[1] = -1000;
result = Pmcm.SetEncoder(0, axis, data);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim data(1) As Integer
axis = PMCM_AXIS_X + PMCM_AXIS_Y
data(0) = 1000
data(1) = -1000
result = PmcmSetEncoder(0, axis, data)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim lngData(1) As Long
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngData(0) = 1000
lngData(1) = -1000
lngResult = PmcmSetEncoder(0, intAxis, lngData(0))
```

GCC

```
int32_t result;
uint16_t axis;
int32_t data[2];
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
data[0] = 1000;
data[1] = -1000;
result = PmcmSetEncoder(0, axis, data);
```

PmcmSetEncoderConfig

機能

エンコーダ入力の設定をします。

カウンタについて

書式

```
INT PmcmSetEncoderConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    WORD wConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

設定する項目を指定します。

設定値	内容
PMCM_ENCODER_MODE	エンコーダ入力モード
PMCM_ENCODER_DIR	A/B入力カウント方向
PMCM_ENCODER_FILTER	入力フィルタ

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wConfig

設定値を指定します。
設定する項目（wMode）によって値が異なります。

- wMode : PMCM_ENCODER_MODE

設定値	内容
0	1逓倍
1	2逓倍
2	4逓倍
3	アップ・ダウンの2パルス入力

  初期値 : 0

- wMode : PMCM_ENCODER_DIR

設定値	内容
0	Aの位相が進んでいる時、またはAの立ち上がりでカウントアップ
1	Bの位相が進んでいる時、またはBの立ち上がりでカウントアップ

  初期値 : 0

- wMode : PMCM_ENCODER_FILTER

設定値	内容

設定値	内容
0	フィルタ機能OFF
1	フィルタ機能ON

A+、A-、B+、B-、Z+、Z-にフィルタを挿入します。
フィルタ機能をONにすると、0.15μsec以下のパルス入力を無効にします。

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸とY軸のエンコーダ入力モードをアップ・ダウンの2パルス入力に設定します。

C/C++

```
int nResult;
WORD wAxis;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmSetEncoderConfig(0, wAxis, PMCM_ENCODER_MODE, 3);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetEncoderConfig(0, axis, PMCM_ENCODER_MODE, 3);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.SetEncoderConfig(0, axis, Pmcm.PMCM_ENCODER_MODE, 3);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmSetEncoderConfig(0, axis, PMCM_ENCODER_MODE, 3)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmSetEncoderConfig(0, intAxis, PMCM_ENCODER_MODE, 3)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetEncoderConfig(0, axis, PMCM_ENCODER_MODE, 3);
```

PmcmSetErcConfig

機能

ERC信号の設定をします。

書式

```
INT PmcmSetErcConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    WORD wConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

設定する項目を指定します。

設定値	内容

設定値	内容
PMCM_ERC_LOGIC	出力論理
PMCM_ERC_AUTOOUT_1	自動出力設定1
PMCM_ERC_AUTOOUT_2	自動出力設定2
PMCM_ERC_WIDTH	出力幅
PMCM_ERC_OFF_TIMER	OFFタイマ
PMCM_ERC_SIG_ON	ERC信号を出力
PMCM_ERC_SIG_OFF	ERC信号の出力リセット

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wConfig

設定値を指定します。
設定する項目（wMode）によって値が異なります。

- wMode : PMCM_ERC_LOGIC

  ERC信号の出力論理

設定値	内容
0	負論理
1	正論理

  初期値 : 0

- wMode : PMCM_ERC_AUTOOUT_1

  ERC信号の自動出力設定

設定値	内容
0	EL、ALM、EMG入力停止時にERC信号を出力しない

設定値	内容
1	EL、ALM、EMG入力停止時にERC信号を自動出力 （減速停止の場合は出力されません）

初期値 : 0

- wMode : PMCM_ERC_AUTOOUT_2

  ERC信号の自動出力設定

設定値	内容
0	原点復帰完了時にERC信号を出力しない
1	原点復帰完了時にERC信号を自動出力

初期値 : 0

- wMode : PMCM_ERC_WIDTH

  ERC信号の出力幅

設定値	内容
0	12μsec
1	102μsec
2	408μsec
3	1.6msec
4	13msec
5	52msec
6	104msec
7	レベル出力

初期値 : 0

- wMode : PMCM_ERC_OFF_TIMER

  ERC信号のOFFタイマ

設定値	内容
0	0μsec
1	12μsec
2	1.6msec
3	104msec

初期値 : 0

- wMode : PMCM_ERC_SIG_ON

  ERC信号を出力します。

  wConfigは使用しません（0を設定してください）

- wMode : PMCM_ERC_SIG_OFF

  ERC信号をレベル出力に設定した時に出力をリセットします。

  wConfigは使用しません（0を設定してください）

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸とY軸の出力論理を負論理に設定します。

C/C++

```
int nResult;
WORD wAxis;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmSetErcConfig(0, wAxis, PMCM_ERC_LOGIC, 0);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetErcConfig(0, axis, PMCM_ERC_LOGIC, 0);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.SetErcConfig(0, axis, Pmcm.PMCM_ERC_LOGIC, 0);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmSetErcConfig(0, axis, PMCM_ERC_LOGIC, 0)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmSetErcConfig(0, intAxis, PMCM_ERC_LOGIC, 0)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetErcConfig(0, axis, PMCM_ERC_LOGIC, 0);
```

PmcmSetEvent（Windows）

> この関数はWindows用です。LinuxではPmcmSetEvent関数（Linux）を使用してください。

機能

イベントの設定をし、イベント発生を開始します。

イベントについて

書式

```
INT PmcmSetEvent(
    WORD wID,
    HANDLE hEvent
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA
型	WORD	unsigned short	ushort	Short	Integer

hEvent

イベントオブジェクトのハンドルを指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0	VBA
型	HANDLE	IntPtr	IntPtr	IntPtr	Long	LongPtr

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA
型	INT	int	int	Integer	Long

使用例

IDが0のボードへ、イベントの設定します。

C/C++

```
int nResult;
HANDLE hEvent;
hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
nResult = PmcmSetEvent(0, hEvent);
```

C++/CLI

```
int result;
ManualResetEvent^ eventHandle;
eventHandle = gcnew ManualResetEvent(false);
result = PmcmSetEvent(id, eventHandle->Handle);
```

C#

```
int result;
ManualResetEvent eventHandle;
eventHandle = new ManualResetEvent(false);
result = Pmcm.SetEvent(0, eventHandle.Handle);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim eventHandle As ManualResetEvent
eventHandle = New ManualResetEvent(False)
result = PmcmSetEvent(0, eventHandle.Handle)
```

VB6.0

```
Dim lngResult As Long
Dim lngEvent As Long
lngEvent = CreateEvent(0, True, False, 0)
lngResult = PmcmSetEvent(0, lngEvent)
```

VBA

```
Dim lngResult As Long
Dim lngEvent As LongPtr
lngEvent = CreateEvent(0, True, False, 0)
lngResult = PmcmSetEvent(0, lngEvent)
```

PmcmSetEvent（Linux）

> この関数はLinux用です。WindowsではPmcmSetEvent関数（Windows）を使用してください。

機能

イベントの設定をし、イベント発生を開始します。

イベントについて

書式

```
int32_t PmcmSetEvent(
    uint16_t wID
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	GCC
型	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	GCC
型	int32_t

使用例

IDが0のボードへ、イベントの設定します。

GCC

```
int32_t result;
result = PmcmSetEvent(0);
```

PmcmSetEventMask

機能

イベントマスクの設定をします。

イベントについて

書式

```
INT PmcmSetEventMask(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    WORD wConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

設定する項目を指定します。

設定値	内容
PMCM_EVENT_STOP	停止イベント
PMCM_EVENT_STATE	状態イベント

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wConfig

設定値を指定します。
設定する項目（wMode）によって値が異なります。

- wMode : PMCM_EVENT_STOP

bit	15	14	13	12	11	10	9	8
信号	-	-	-	-	-	-	-	-

bit	7	6	5	4	3	2	1	0
信号	-	-	SD	EMG	ALM	-EL	+EL	STOP

bit5 : SD入力ONによる減速停止時
bit4 : EMG入力ONによる停止時
bit3 : ALM入力ONによる停止時
bit2 : -EL入力ONによる停止時
bit1 : +EL入力ONによる停止時
bit0 : 自動停止時

各ビットの設定値	内容
0	イベント発生マスク
1	イベント発生許可

初期値 : H'0

- wMode : PMCM_EVENT_STATE

bit	15	14	13	12	11	10	9	8
信号	-	-	-	-	-	-	-	SD

bit	7	6	5	4	3	2	1	0
信号	ORG	LTC	CMP2	CMP1	DEC FIN	DEC STA	ACC FIN	ACC STA

bit8 : SD入力ON時
bit7 : ORG入力ONによるカウント値のラッチ時
bit6 : LTC入力によるカウント値のラッチ時
bit5 : コンパレータ（エンコーダ）条件成立時
bit4 : コンパレータ（出力パルス）条件成立時
bit3 : 減速終了時
bit2 : 減速開始時
bit1 : 加速終了時
bit0 : 加速開始時

各ビットの設定値	内容
0	イベント発生マスク
1	イベント発生許可

初期値 : H'0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸とY軸の停止イベントを全てマスクに設定します。

C/C++

```
int nResult;
WORD wAxis;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmSetEventMask(0, wAxis, PMCM_EVENT_STOP, 0);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetEventMask(0, axis, PMCM_EVENT_STOP, 0);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.SetEventMask(0, axis, Pmcm.PMCM_EVENT_STOP, 0);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmSetEventMask(0, axis, PMCM_EVENT_STOP, 0)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmSetEventMask(0, intAxis, PMCM_EVENT_STOP, 0)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetEventMask(0, axis, PMCM_EVENT_STOP, 0);
```

PmcmSetLatchConfig

機能

カウンタラッチの設定をします。

カウンタについて

書式

```
INT PmcmSetLatchConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    WORD wConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

設定する項目を指定します。

設定値	内容
PMCM_COUNTER_LATCH_LTC	LTC信号によるラッチ（出力パルスカウンタ）
PMCM_COUNTER_LATCH_ORG	原点復帰によるラッチ（出力パルスカウンタ）
PMCM_COUNTER_CLEAR_LATCH	ラッチ後のクリア（出力パルスカウンタ）
PMCM_ENCODER_LATCH_LTC	LTC信号によるラッチ（エンコーダカウンタ）
PMCM_ENCODER_LATCH_ORG	原点復帰によるラッチ（エンコーダカウンタ）
PMCM_ENCODER_CLEAR_LATCH	ラッチ後のクリア（エンコーダカウンタ）

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wConfig

設定値を指定します。
設定する項目（wMode）によって値が異なります。

- wMode : PMCM_COUNTER_LATCH_LTC、PMCM_ENCODER_LATCH_LTC

設定値	内容
0	LTC信号によりカウンタをラッチしません
1	LTC信号によりカウンタをラッチします

  初期値 : 0

- wMode : PMCM_COUNTER_LATCH_ORG、PMCM_ENCODER_LATCH_ORG

設定値	内容
0	原点復帰によりカウンタをラッチしません
1	原点復帰によりカウンタをラッチします

  初期値 : 0

- wMode : PMCM_COUNTER_CLEAR_LATCH、PMCM_ENCODER_CLEAR_LATCH

設定値	内容

設定値	内容
0	ラッチ後にカウンタをクリアしません
1	ラッチ後にカウンタをクリアします

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸とY軸のLTC信号によるラッチ（出力パルスカウンタ）をなしに設定します。

C/C++

```
int nResult;
WORD wAxis;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmSetLatchConfig(0, wAxis, PMCM_COUNTER_LATCH_LTC, 0);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetLatchConfig(0, axis, PMCM_COUNTER_LATCH_LTC, 0);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.SetLatchConfig(0, axis, Pmcm.PMCM_COUNTER_LATCH_LTC, 0);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmSetLatchConfig(0, axis, PMCM_COUNTER_LATCH_LTC, 0)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmSetLatchConfig(0, intAxis, PMCM_COUNTER_LATCH_LTC, 0)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetLatchConfig(0, axis, PMCM_COUNTER_LATCH_LTC, 0);
```

PmcmSetMotion

機能

動作パラメータを設定します。
動作パラメータに初期設定値はありませんので、動作開始前に動作パラメータを設定してください。

連続動作について
原点復帰動作について
位置決め動作について

書式

```
INT PmcmSetMotion(
    WORD wID,
    WORD wAxis,
    PMOTIONPMCM pMotion
);

typedef struct {
    WORD wMoveMode;
    WORD wStartMode;
    FLOAT fSpeedRate;
    WORD wAccDecMode;
    FLOAT fLowSpeed;
    FLOAT fSpeed;
    WORD wAccTime;
    WORD wDecTime;
    FLOAT fSAccSpeed;
    FLOAT fSDecSpeed;
    LONG lSlowdown;
    LONG lStep;
    BOOL bAbsolutePtp;
} MOTIONPMCM, *PMOTIONPMCM;
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することができます。

設定値	内容

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pMotion

動作パラメータが格納されているバッファへのポインタを指定します。

> **バッファは2軸分用意してください**

動作パラメータ計算方法

> **Warning**
>
> ドライバ内での計算による誤差のため、動作パラメータとして設定した値と実際に設定された値が異なる場合があります。
> PmcmGetMotion関数を使用して実際に設定された値を確認することができます。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PMOTIONPMCM	MOTIONPMCM*	MOTIONPMCM	MOTIONPMCM	MOTIONPMCM	PMOTIONPMCM

wMoveMode

動作モード

設定値	内容
PMCM_JOG	連続動作
PMCM_ORG	原点復帰動作
PMCM_PTP	位置決め動作

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wStartMode

起動モード

設定値	内容
PMCM_CONST	定速起動
PMCM_CONST_DEC	定速－減速起動
PMCM_ACC_DEC	加減速起動

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fSpeedRate

速度倍率
設定範囲は0.3 ～ 600

動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

wAccDecMode

加減速モード

設定値	内容
PMCM_ACC_LINEAR	直線加減速
PMCM_ACC_SCURVE	S字加減速

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fLowSpeed

起動速度
設定範囲は0.3 ～ 9829800[pps]

> 移動速度（fSpeed）以下の値を設定してください

速度倍率により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

fSpeed

移動速度
設定範囲は0.3 ～ 9829800[pps]

> 起動速度（fLowSpeed）以上の値を設定してください

速度倍率により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

wAccTime

加速時間
設定単位はmsec

速度倍率・起動速度・移動速度により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wDecTime

減速時間
設定単位はmsec

速度倍率・起動速度・移動速度により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fSAccSpeed

加速S字区間
設定範囲は0, 0.3 ～ 4914600[pps]
0を設定した場合は直線加速部分のないS字加速動作となります。

> 加減速モードに直線加減速を指定した場合は0を設定してください

速度倍率により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

fSDecSpeed

減速S字区間
設定範囲は0, 0.3 ～ 4914600[pps]
0を設定した場合は直線減速部分のないS字減速動作となります。

> 加減速モードに直線加減速を指定した場合は0を設定してください

速度倍率により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

lSlowdown

スローダウンポイント

設定値	内容
-1	スローダウンポイントは自動設定 減速開始点は内部で計算されます。
0 ～ 16777215	スローダウンポイントはマニュアル設定 残パルス数が設定値以下になると減速を開始します。 動作モードがPMCM_PTPの時に有効となります。

> **Tip**
>
> 位置決め動作の時のみ有効となります。
> 連続動作、原点復帰動作の時は-1または0を設定してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

lStep

移動パルス数、移動方向
動作モードによって設定値が異なります。

動作モード	設定値
PMCM_JOG PMCM_ORG	PMCM_DIR_CW : +方向 PMCM_DIR_CCW : -方向
PMCM_PTP	-134217728 〜 +134217727

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

bAbsolutePtp

絶対座標指定
移動パルス数の設定値を絶対座標とすることができます。
動作モードがPMCM_PTPの時に有効となります。

設定値	内容
0	移動パルス数の設定値は移動量
1	移動パルス数の設定値は絶対座標

> **Tip**
>
> 動作モードがPMCM_JOG・PMCM_ORGの時は0を設定してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	BOOL	long	int	Integer	Long	int32_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

複数軸同時に設定しない場合でも、動作パラメータバッファは2軸分用意し、配列の先頭アドレスを関数へ設定するようにしてください。

```
MOTIONPMCM Motion[2]; ←2軸分用意
Motion[0] ←X軸の動作パラメータ
Motion[1] ←Y軸の動作パラメータ
```

> **Warning**
>
> 本関数を実行した後、PmcmStartMotion関数を実行するまでの間に各種設定関数（PmcmSetxxxxConfig関数）を実行しても、設定内容が反映されずに動作する場合があります。
> 各種設定をおこなう場合は本関数を実行する前におこなってください。

使用例

IDが0のボードへ、X軸の動作パラメータを設定します。

C/C++

```
int nResult;
WORD wAxis;
MOTIONPMCM Motion[2];
wAxis = PMCM_AXIS_X;
Motion[0].wMoveMode = PMCM_JOG;
Motion[0].wStartMode = PMCM_CONST;
Motion[0].fSpeedRate = 1;
Motion[0].wAccDecMode = PMCM_ACC_LINEAR;
Motion[0].fLowSpeed = 100;
Motion[0].fSpeed = 100;
Motion[0].wAccTime = 0;
Motion[0].wDecTime = 0;
Motion[0].fSAccSpeed = 0;
Motion[0].fSDecSpeed = 0;
Motion[0].lSlowdown = -1;
Motion[0].lStep = PMCM_DIR_CW;
Motion[0].bAbsolutePtp = 0;
nResult = PmcmSetMotion(0, wAxis, Motion);
```

C++/CLI

```
int result;
unsigned short axis;
MOTIONPMCM motion[2];
axis = PMCM_AXIS_X;
motion[0].wMoveMode = PMCM_JOG;
motion[0].wStartMode = PMCM_CONST;
motion[0].fSpeedRate = 1;
motion[0].wAccDecMode = PMCM_ACC_LINEAR;
motion[0].fLowSpeed = 100;
motion[0].fSpeed = 100;
motion[0].wAccTime = 0;
motion[0].wDecTime = 0;
motion[0].fSAccSpeed = 0;
motion[0].fSDecSpeed = 0;
motion[0].lSlowdown = -1;
motion[0].lStep = PMCM_DIR_CW;
motion[0].bAbsolutePtp = 0;
result = PmcmSetMotion(0, axis, motion);
```

C#

```
int result;
ushort axis;
Pmcm.MOTIONPMCM[] motion = new Pmcm.MOTIONPMCM[2];
axis = Pmcm.PMCM_AXIS_X;
motion[0].wMoveMode = Pmcm.PMCM_JOG;
motion[0].wStartMode = Pmcm.PMCM_CONST;
motion[0].fSpeedRate = 1;
motion[0].wAccDecMode = Pmcm.PMCM_ACC_LINEAR;
motion[0].fLowSpeed = 100;
motion[0].fSpeed = 100;
motion[0].wAccTime = 0;
motion[0].wDecTime = 0;
motion[0].fSAccSpeed = 0;
motion[0].fSDecSpeed = 0;
motion[0].lSlowdown = -1;
motion[0].lStep = Pmcm.PMCM_DIR_CW;
motion[0].bAbsolutePtp = 0;
result = Pmcm.SetMotion(0, axis, motion);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim motion(1) As MOTIONPMCM
axis = PMCM_AXIS_X
motion(0).wMoveMode = PMCM_JOG
motion(0).wStartMode = PMCM_CONST
motion(0).fSpeedRate = 1
motion(0).wAccDecMode = PMCM_ACC_LINEAR
motion(0).fLowSpeed = 100
motion(0).fSpeed = 100
motion(0).wAccTime = 0
motion(0).wDecTime = 0
motion(0).fSAccSpeed = 0
motion(0).fSDecSpeed = 0
motion(0).lSlowdown = -1
motion(0).lStep = PMCM_DIR_CW
```

```
motion(0).bAbsolutePtp = 0
result = PmcmSetMotion(0, axis, motion)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim Motion(1) As MOTIONPMCM
intAxis = PMCM_AXIS_X
Motion(0).wMoveMode = PMCM_JOG
Motion(0).wStartMode = PMCM_CONST
Motion(0).fSpeedRate = 1
Motion(0).wAccDecMode = PMCM_ACC_LINEAR
Motion(0).fLowSpeed = 100
Motion(0).fSpeed = 100
Motion(0).wAccTime = 0
Motion(0).wDecTime = 0
Motion(0).fSAccSpeed = 0
Motion(0).fSDecSpeed = 0
Motion(0).lSlowdown = -1
Motion(0).lStep = PMCM_DIR_CW
Motion(0).bAbsolutePtp = 0
lngResult = PmcmSetMotion(0, intAxis, Motion(0))
```

GCC

```
int32_t result;
uint16_t axis;
MOTIONPMCM motion[2];
axis = PMCM_AXIS_X;
motion[0].wMoveMode = PMCM_JOG;
motion[0].wStartMode = PMCM_CONST;
motion[0].fSpeedRate = 1;
motion[0].wAccDecMode = PMCM_ACC_LINEAR;
motion[0].fLowSpeed = 100;
motion[0].fSpeed = 100;
motion[0].wAccTime = 0;
motion[0].wDecTime = 0;
motion[0].fSAccSpeed = 0;
motion[0].fSDecSpeed = 0;
motion[0].lSlowdown = -1;
motion[0].lStep = PMCM_DIR_CW;
motion[0].bAbsolutePtp = 0;
result = PmcmSetMotion(0, axis, motion);
```

PmcmSetMotionCp

機能

CPの動作パラメータを設定します。
動作パラメータに初期設定値はありませんので、動作開始前に動作パラメータを設定してください。

CP動作について

書式

```
INT PmcmSetMotionCp(
    WORD wID,
    WORD wAxis,
    PMOTIONPMCM pMotion,
    WORD wCount
);

typedef struct {
    WORD wMoveMode;
    WORD wStartMode;
    FLOAT fSpeedRate;
    WORD wAccDecMode;
    FLOAT fLowSpeed;
    FLOAT fSpeed;
    WORD wAccTime;
    WORD wDecTime;
    FLOAT fSAccSpeed;
    FLOAT fSDecSpeed;
    LONG lSlowdown;
    LONG lStep;
    BOOL bAbsolutePtp;
} MOTIONPMCM, *PMOTIONPMCM;
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することはできません。

設定値	内容

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pMotion

動作パラメータが格納されているバッファへのポインタを指定します。

動作パラメータ計算方法

> **Warning**
>
> ドライバ内での計算による誤差のため、動作パラメータとして設定した値と実際に設定された値が異なる場合があります。
> PmcmGetMotionCp関数を使用して実際に設定された値を確認することができます。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PMOTIONPMCM	MOTIONPMCM*	MOTIONPMCM	MOTIONPMCM	MOTIONPMCM	PMOTIONPMCM

wMoveMode

動作モード

設定値	内容
PMCM_PTP	位置決め動作

> PMCM_PTPのみ設定可能です

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wStartMode

起動モード

設定値	内容
PMCM_CONST	定速起動
PMCM_CONST_DEC	定速ー減速起動
PMCM_ACC_DEC	加減速起動

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fSpeedRate

速度倍率
設定範囲は0.3 ～ 600

動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

wAccDecMode

加減速モード

設定値	内容
PMCM_ACC_LINEAR	直線加減速
PMCM_ACC_SCURVE	S字加減速

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fLowSpeed

起動速度
設定範囲は0.3 ～ 9829800[pps]

> 移動速度（fSpeed）以下の値を設定してください

速度倍率により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

fSpeed

移動速度
設定範囲は0.3 ～ 9829800[pps]

> 起動速度（fLowSpeed）以上の値を設定してください

速度倍率により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

wAccTime

加速時間
設定単位はmsec

速度倍率・起動速度・移動速度により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wDecTime

減速時間
設定単位はmsec

速度倍率・起動速度・移動速度により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fSAccSpeed

加速S字区間
設定範囲は0.3 ～ 4914600[pps]

> 加減速モードに直線加減速を指定した場合は0を設定してください

速度倍率により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

fSDecSpeed

減速S字区間
設定範囲は0.3 ～ 4914600[pps]

> 加減速モードに直線加減速を指定した場合は0を設定してください

速度倍率により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

lSlowdown

スローダウンポイント

設定値	内容
-1	スローダウンポイントは自動設定 減速開始点は内部で計算されます。
0 ～ 16777215	スローダウンポイントはマニュアル設定 残パルス数が設定値以下になると減速を開始します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

lStep

移動パルス数、移動方向
設定範囲は-134217728 ～ +134217727

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

bAbsolutePtp

絶対座標指定

設定値	内容
0	移動パルス数の設定値は移動量

> 0のみ指定可能です

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	BOOL	long	int	Integer	Long	int32_t

wCount

設定する動作パラメータ数
設定範囲は1 ～ 96

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

> **Warning**
>
> 本関数を実行した後、PmcmStartMotionCp関数を実行するまでの間に各種設定関数（PmcmSetxxxxConfig関数）を実行しても、設定内容が反映されずに動作する場合があります。
> 各種設定をおこなう場合は本関数を実行する前におこなってください。

使用例

IDが0のボードへ、X軸のCP動作パラメータを2つ設定します。

C/C++

```
int nResult;
WORD wAxis;
MOTIONPMCM Motion[2];
wAxis = PMCM_AXIS_X;
Motion[0].wMoveMode = PMCM_PTP;
Motion[0].wStartMode = PMCM_CONST;
Motion[0].fSpeedRate = 1;
Motion[0].wAccDecMode = PMCM_ACC_LINEAR;
Motion[0].fLowSpeed = 100;
Motion[0].fSpeed = 100;
Motion[0].wAccTime = 0;
Motion[0].wDecTime = 0;
Motion[0].fSAccSpeed = 0;
Motion[0].fSDecSpeed = 0;
Motion[0].lSlowdown = -1;
Motion[0].lStep = 1000;
Motion[0].bAbsolutePtp = 0;
Motion[1].wMoveMode = PMCM_PTP;
Motion[1].wStartMode = PMCM_CONST;
Motion[1].fSpeedRate = 1;
Motion[1].wAccDecMode = PMCM_ACC_LINEAR;
Motion[1].fLowSpeed = 100;
Motion[1].fSpeed = 100;
Motion[1].wAccTime = 0;
Motion[1].wDecTime = 0;
Motion[1].fSAccSpeed = 0;
Motion[1].fSDecSpeed = 0;
Motion[1].lSlowdown = -1;
Motion[1].lStep = 1000;
Motion[1].bAbsolutePtp = 0;
nResult = PmcmSetMotionCp(0, wAxis, Motion, 2);
```

C++/CLI

```
int result;
unsigned short axis;
MOTIONPMCM motion[2];
axis = PMCM_AXIS_X;
motion[0].wMoveMode = PMCM_PTP;
motion[0].wStartMode = PMCM_CONST;
motion[0].fSpeedRate = 1;
motion[0].wAccDecMode = PMCM_ACC_LINEAR;
motion[0].fLowSpeed = 100;
motion[0].fSpeed = 100;
motion[0].wAccTime = 0;
motion[0].wDecTime = 0;
motion[0].fSAccSpeed = 0;
motion[0].fSDecSpeed = 0;
motion[0].lSlowdown = -1;
motion[0].lStep = 1000;
motion[0].bAbsolutePtp = 0;
motion[1].wMoveMode = PMCM_PTP;
motion[1].wStartMode = PMCM_CONST;
motion[1].fSpeedRate = 1;
```

```
motion[1].wAccDecMode = PMCM_ACC_LINEAR;
motion[1].fLowSpeed = 100;
motion[1].fSpeed = 100;
motion[1].wAccTime = 0;
motion[1].wDecTime = 0;
motion[1].fSAccSpeed = 0;
motion[1].fSDecSpeed = 0;
motion[1].lSlowdown = -1;
motion[1].lStep = 1000;
motion[1].bAbsolutePtp = 0;
result = PmcmSetMotionCp(0, axis, motion, 2);
```

C#

```
int result;
ushort axis;
Pmcm.MOTIONPMCM[] motion = new Pmcm.MOTIONPMCM[2];
axis = Pmcm.PMCM_AXIS_X;
motion[0].wMoveMode = Pmcm.PMCM_PTP;
motion[0].wStartMode = Pmcm.PMCM_CONST;
motion[0].fSpeedRate = 1;
motion[0].wAccDecMode = Pmcm.PMCM_ACC_LINEAR;
motion[0].fLowSpeed = 100;
motion[0].fSpeed = 100;
motion[0].wAccTime = 0;
motion[0].wDecTime = 0;
motion[0].fSAccSpeed = 0;
motion[0].fSDecSpeed = 0;
motion[0].lSlowdown = -1;
motion[0].lStep = 1000;
motion[0].bAbsolutePtp = 0;
motion[1].wMoveMode = Pmcm.PMCM_PTP;
motion[1].wStartMode = Pmcm.PMCM_CONST;
motion[1].fSpeedRate = 1;
motion[1].wAccDecMode = Pmcm.PMCM_ACC_LINEAR;
motion[1].fLowSpeed = 100;
motion[1].fSpeed = 100;
motion[1].wAccTime = 0;
motion[1].wDecTime = 0;
motion[1].fSAccSpeed = 0;
motion[1].fSDecSpeed = 0;
motion[1].lSlowdown = -1;
motion[1].lStep = 1000;
motion[1].bAbsolutePtp = 0;
result = Pmcm.SetMotionCp(0, axis, motion, 2);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim motion(1) As MOTIONPMCM
axis = PMCM_AXIS_X
motion(0).wMoveMode = PMCM_PTP
motion(0).wStartMode = PMCM_CONST
motion(0).fSpeedRate = 1
motion(0).wAccDecMode = PMCM_ACC_LINEAR
motion(0).fLowSpeed = 100
motion(0).fSpeed = 100
motion(0).wAccTime = 0
motion(0).wDecTime = 0
```

```
motion(0).fSAccSpeed = 0
motion(0).fSDecSpeed = 0
motion(0).lSlowdown = -1
motion(0).lStep = 1000
motion(0).bAbsolutePtp = 0
motion(1).wMoveMode = PMCM_PTP
motion(1).wStartMode = PMCM_CONST
motion(1).fSpeedRate = 1
motion(1).wAccDecMode = PMCM_ACC_LINEAR
motion(1).fLowSpeed = 100
motion(1).fSpeed = 100
motion(1).wAccTime = 0
motion(1).wDecTime = 0
motion(1).fSAccSpeed = 0
motion(1).fSDecSpeed = 0
motion(1).lSlowdown = -1
motion(1).lStep = 1000
motion(1).bAbsolutePtp = 0
result = PmcmSetMotionCp(0, axis, motion, 2)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim Motion(1) As MOTIONPMCM
intAxis = PMCM_AXIS_X
Motion(0).wMoveMode = PMCM_PTP
Motion(0).wStartMode = PMCM_CONST
Motion(0).fSpeedRate = 1
Motion(0).wAccDecMode = PMCM_ACC_LINEAR
Motion(0).fLowSpeed = 100
Motion(0).fSpeed = 100
Motion(0).wAccTime = 0
Motion(0).wDecTime = 0
Motion(0).fSAccSpeed = 0
Motion(0).fSDecSpeed = 0
Motion(0).lSlowdown = -1
Motion(0).lStep = 1000
Motion(0).bAbsolutePtp = 0
Motion(1).wMoveMode = PMCM_PTP
Motion(1).wStartMode = PMCM_CONST
Motion(1).fSpeedRate = 1
Motion(1).wAccDecMode = PMCM_ACC_LINEAR
Motion(1).fLowSpeed = 100
Motion(1).fSpeed = 100
Motion(1).wAccTime = 0
Motion(1).wDecTime = 0
Motion(1).fSAccSpeed = 0
Motion(1).fSDecSpeed = 0
Motion(1).lSlowdown = -1
Motion(1).lStep = 1000
Motion(1).bAbsolutePtp = 0
lngResult = PmcmSetMotionCp(0, intAxis, Motion(0), 2)
```

GCC

```
int32_t result;
uint16_t axis;
MOTIONPMCM motion[2];
axis = PMCM_AXIS_X;
```

```
motion[0].wMoveMode = PMCM_PTP;
motion[0].wStartMode = PMCM_CONST;
motion[0].fSpeedRate = 1;
motion[0].wAccDecMode = PMCM_ACC_LINEAR;
motion[0].fLowSpeed = 100;
motion[0].fSpeed = 100;
motion[0].wAccTime = 0;
motion[0].wDecTime = 0;
motion[0].fSAccSpeed = 0;
motion[0].fSDecSpeed = 0;
motion[0].lSlowdown = -1;
motion[0].lStep = 1000;
motion[0].bAbsolutePtp = 0;
motion[1].wMoveMode = PMCM_PTP;
motion[1].wStartMode = PMCM_CONST;
motion[1].fSpeedRate = 1;
motion[1].wAccDecMode = PMCM_ACC_LINEAR;
motion[1].fLowSpeed = 100;
motion[1].fSpeed = 100;
motion[1].wAccTime = 0;
motion[1].wDecTime = 0;
motion[1].fSAccSpeed = 0;
motion[1].fSDecSpeed = 0;
motion[1].lSlowdown = -1;
motion[1].lStep = 1000;
motion[1].bAbsolutePtp = 0;
result = PmcmSetMotionCp(0, axis, motion, 2);
```

PmcmSetMotionLine

機能

直線補間動作パラメータを設定します。
動作パラメータに初期設定値はありませんので、動作開始前に動作パラメータを設定してください。

直線補間動作について

書式

```
INT PmcmSetMotionLine(
    WORD wID,
    WORD wAxis,
    PMOTIONLINEPMCM pMotionLine
);

typedef struct {
    WORD wStartMode;
    FLOAT fSpeedRate;
    WORD wAccDecMode;
    FLOAT fLowSpeed;
    FLOAT fSpeed;
    WORD wAccTime;
    WORD wDecTime;
    FLOAT fSAccSpeed;
    FLOAT fSDecSpeed;
    LONG lSlowdown;
    LONG lStep[2];
    BOOL bAbsolute[2];
} MOTIONLINEPMCM, *PMOTIONLINEPMCM;
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。2軸以上指定してください。

設定値	内容
PMCM_AXIS_X	X軸

設定値	内容
PMCM_AXIS_Y	Y軸

> PMC-M2C-UのwAxisに設定する値はPMCM_AXIS_X + PMCM_AXIS_Yで固定です

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pMotionLine

動作パラメータが格納されているバッファへのポインタを指定します。

動作パラメータ計算方法

> **Warning**
>
> ドライバ内での計算による誤差のため、動作パラメータとして設定した値と実際に設定された値が異なる場合があります。
> PmcmGetMotionLine関数を使用して実際に設定された値を確認することができます。

下記パラメータについては、X軸・Y軸で移動量の多い軸のパラメータを設定してください。
fSpeedRate
fLowSpeed
fSpeed
wAccTime
wDecTime
fSAccSpeed
fSDecSpeed
lSlowdown

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PMOTIONLINEPMCM	MOTIONLINEPMCM*	ref MOTIONLINEPMCM	MOTIONLINEPMCM	MOTIONLINEPMCM	PMOTIONLINEPMCM

wStartMode

起動モード

設定値	内容
PMCM_CONST	定速起動
PMCM_CONST_DEC	定速ー減速起動

設定値	内容
PMCM_ACC_DEC	加減速起動

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fSpeedRate

速度倍率
設定範囲は0.3 ～ 600

動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

wAccDecMode

加減速モード

設定値	内容
PMCM_ACC_LINEAR	直線加減速
PMCM_ACC_SCURVE	S字加減速

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fLowSpeed

起動速度
設定範囲は0.3 ～ 9829800[pps]

> 移動速度（fSpeed）以下の値を設定してください

速度倍率により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

fSpeed

移動速度
設定範囲は0.3 ～ 9829800[pps]

> 起動速度（fLowSpeed）以上の値を設定してください

速度倍率により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

wAccTime

加速時間
設定単位はmsec

速度倍率・起動速度・移動速度により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wDecTime

減速時間
設定単位はmsec

速度倍率・起動速度・移動速度により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

fSAccSpeed

加速S字区間
設定範囲は0.3 ～ 4914600[pps]

> 加減速モードに直線加減速を指定した場合は0を設定してください

速度倍率により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

fSDecSpeed

減速S字区間
設定範囲は0.3 ～ 4914600[pps]

> 加減速モードに直線加減速を指定した場合は0を設定してください

速度倍率により設定範囲が変わります。
動作パラメータ計算方法を参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	FLOAT	float	float	Single	Single	float

lSlowdown

スローダウンポイント

設定値	内容
-1	スローダウンポイントは自動設定 減速開始点は内部で計算されます。
0 ～ 16777215	スローダウンポイントはマニュアル設定 残パルス数が設定値以下になると減速を開始します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

lStep

移動パルス数
設定範囲は-134217728 ～ +134217727

2軸の移動パルス数を設定します。
lStep[0] : X軸の移動パルス数

lStep[1] : Y軸の移動パルス数

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	LONG	long	int	Integer	Long	int32_t

bAbsolute

絶対座標指定
移動パルス数の設定値を絶対座標とすることができます。

設定値	内容
0	移動パルス数の設定値は移動量
1	移動パルス数の設定値は絶対座標

2軸の絶対座標指定を設定します。
bAbsolute[0] : X軸の絶対座標指定
bAbsolute[1] : Y軸の絶対座標指定

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	BOOL	long	int	Integer	Long	int32_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

> **Warning**
>
> 本関数を実行した後、PmcmStartMotionLine関数を実行するまでの間に各種設定関数（PmcmSetxxxxConfig関数）を実行しても、設定内容が反映されずに動作する場合があります。
> 各種設定をおこなう場合は本関数を実行する前におこなってください。

使用例

IDが0のボードへ、直線補間動作パラメータを設定します。

C/C++

```
int nResult;
WORD wAxis;
MOTIONLINEPMCM MotionLine;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
MotionLine.wStartMode = PMCM_CONST;
MotionLine.fSpeedRate = 1;
MotionLine.wAccDecMode = PMCM_ACC_LINEAR;
MotionLine.fLowSpeed = 100;
MotionLine.fSpeed = 100;
MotionLine.wAccTime = 0;
MotionLine.wDecTime = 0;
MotionLine.fSAccSpeed = 0;
MotionLine.fSDecSpeed = 0;
MotionLine.lSlowdown = -1;
MotionLine.lStep[0] = 1000;
MotionLine.lStep[1] = 500;
MotionLine.bAbsolute[0] = 0;
MotionLine.bAbsolute[1] = 0;
nResult = PmcmSetMotionLine(0, wAxis, &MotionLine);
```

C++/CLI

```
int result;
unsigned short axis;
MOTIONLINEPMCM motionLine;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
motionLine.wStartMode = PMCM_CONST;
motionLine.fSpeedRate = 1;
motionLine.wAccDecMode = PMCM_ACC_LINEAR;
motionLine.fLowSpeed = 100;
motionLine.fSpeed = 100;
motionLine.wAccTime = 0;
motionLine.wDecTime = 0;
motionLine.fSAccSpeed = 0;
motionLine.fSDecSpeed = 0;
motionLine.lSlowdown = -1;
motionLine.lStep[0] = 1000;
motionLine.lStep[1] = 500;
motionLine.bAbsolute[0] = 0;
motionLine.bAbsolute[1] = 0;
result = PmcmSetMotionLine(0, axis, &motionLine);
```

C#

```
int result;
ushort axis;
Pmcm.MOTIONLINEPMCM motionLine = new Pmcm.MOTIONLINEPMCM();
motionLine.Initialize();
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
motionLine.wStartMode = Pmcm.PMCM_CONST;
motionLine.fSpeedRate = 1;
motionLine.wAccDecMode = Pmcm.PMCM_ACC_LINEAR;
motionLine.fLowSpeed = 100;
motionLine.fSpeed = 100;
motionLine.wAccTime = 0;
motionLine.wDecTime = 0;
motionLine.fSAccSpeed = 0;
```

```
motionLine.fSDecSpeed = 0;
motionLine.lSlowdown = -1;
motionLine.lStep[0] = 1000;
motionLine.lStep[1] = 500;
motionLine.bAbsolute[0] = 0;
motionLine.bAbsolute[1] = 0;
result = Pmcm.SetMotionLine(0, axis, ref motionLine);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim motionLine As MOTIONLINEPMCM
motionLine.Initialize()
axis = PMCM_AXIS_X + PMCM_AXIS_Y
motionLine.wStartMode = PMCM_CONST
motionLine.fSpeedRate = 1
motionLine.wAccDecMode = PMCM_ACC_LINEAR
motionLine.fLowSpeed = 100
motionLine.fSpeed = 100
motionLine.wAccTime = 0
motionLine.wDecTime = 0
motionLine.fSAccSpeed = 0
motionLine.fSDecSpeed = 0
motionLine.lSlowdown = -1
motionLine.lStep(0) = 1000
motionLine.lStep(1) = 500
motionLine.bAbsolute(0) = 0
motionLine.bAbsolute(1) = 0
result = PmcmSetMotionLine(0, axis, motionLine)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim MotionLine As MOTIONLINEPMCM
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
MotionLine.wStartMode = PMCM_CONST
MotionLine.fSpeedRate = 1
MotionLine.wAccDecMode = PMCM_ACC_LINEAR
MotionLine.fLowSpeed = 100
MotionLine.fSpeed = 100
MotionLine.wAccTime = 0
MotionLine.wDecTime = 0
MotionLine.fSAccSpeed = 0
MotionLine.fSDecSpeed = 0
MotionLine.lSlowdown = -1
MotionLine.lStep(0) = 1000
MotionLine.lStep(1) = 500
MotionLine.bAbsolute(0) = 0
MotionLine.bAbsolute(1) = 0
lngResult = PmcmSetMotionLine(0, intAxis, MotionLine)
```

GCC

```
int32_t result;
uint16_t axis;
MOTIONLINEPMCM motion_line;
```

```
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
motion_line.wStartMode = PMCM_CONST;
motion_line.fSpeedRate = 1;
motion_line.wAccDecMode = PMCM_ACC_LINEAR;
motion_line.fLowSpeed = 100;
motion_line.fSpeed = 100;
motion_line.wAccTime = 0;
motion_line.wDecTime = 0;
motion_line.fSAccSpeed = 0;
motion_line.fSDecSpeed = 0;
motion_line.lSlowdown = -1;
motion_line.lStep[0] = 1000;
motion_line.lStep[1] = 500;
motion_line.bAbsolute[0] = 0;
motion_line.bAbsolute[1] = 0;
result = PmcmSetMotionLine(0, axis, &motion_line);
```

PmcmSetPulseConfig

機能

パルス出力の設定をします。

パルス出力について

書式

```
INT PmcmSetPulseConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    WORD wConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

設定する項目を指定します。

設定値	内容
PMCM_PULSE_OUT	パルス出力モード
PMCM_DIR_WAIT	方向変化時の遅延
PMCM_FIN_TIMING	動作完了タイミング
PMCM_FH_REVISE	移動速度補正（三角駆動回避）

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

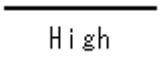
wConfig

設定値を指定します。
設定する項目（wMode）によって値が異なります。

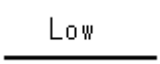
- wMode : PMCM_PULSE_OUT
  - 0

    共通パルスモード
    OUT：負論理
    DIR：（＋方向）High、（―方向）Low

    CW（＋）方向動作時

OUT出力	DIR出力
	

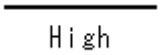
    CCW（―）方向動作時

OUT出力	DIR出力
	

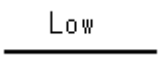
  - 1

    共通パルスモード
    OUT：正論理
    DIR：（＋方向）High、（―方向）Low

    CW（＋）方向動作時

OUT出力	DIR出力
	

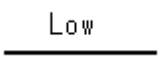
CCW（—）方向動作時

OUT出力	DIR出力
	

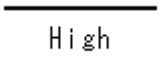
- 2

  共通パルスモード
  OUT：負論理
  DIR：（＋方向）Low、（—方向）High

  CW（＋）方向動作時

OUT出力	DIR出力
	

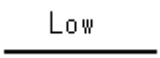
  CCW（—）方向動作時

OUT出力	DIR出力
	

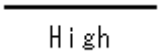
- 3

  共通パルスモード
  OUT：正論理
  DIR：（＋方向）Low、（—方向）High

  CW（＋）方向動作時

OUT出力	DIR出力
	

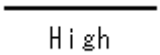
  CCW（—）方向動作時

OUT出力	DIR出力
	

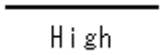
- 4

  2パルスモード
  負論理

  CW（＋）方向動作時

OUT出力	DIR出力
	

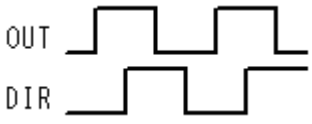
  CCW（―）方向動作時

OUT出力	DIR出力
	

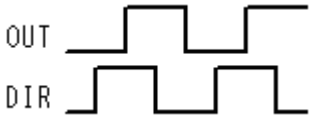
- 5

  90度位相差モード（4逓倍）
  OUT：進みパルス
  DIR：遅れパルス

  CW（＋）方向動作時

OUT/DIR出力


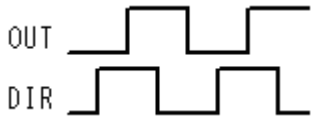
  CCW（―）方向動作時

OUT/DIR出力


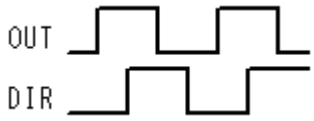
- 6

  90度位相差モード（4逓倍）
  OUT：遅れパルス
  DIR：進みパルス

  CW（＋）方向動作時

OUT/DIR出力


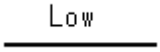
  CCW（―）方向動作時

OUT/DIR出力


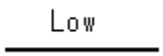
- 7

  2パルスモード
  正論理

  CW（＋）方向動作時

OUT出力	DIR出力
	

  CCW（―）方向動作時

OUT出力	DIR出力
	

初期値 : 7

- wMode : PMCM_DIR_WAIT

  モーターの回転方向が変わるときにパルス出力を遅延させることができます。

設定値	内容
0	方向変化時に200μsecの遅延あり
1	方向変化時に遅延なし

> PMCM_PULSE_OUTの設定が0～3の場合のみ有効です

初期値 : 0

- wMode : PMCM_FIN_TIMING

設定値	内容
0	最終パルスの周期完了時
1	最終パルスの出力OFF時
2	INP信号入力時

INP信号を使用しない場合は、0（周期完了時）を設定してください。
1（出力OFF時）を設定した場合は、CP動作時や絶対座標指定でのPTP動作時に、最終パルスと次動作のスタートパルスとの間隔が狭くなり、正しく動作しない場合があります。

初期値 : 0

- wMode : PMCM_FH_REVISE

移動速度補正のON/OFFを設定します（三角駆動回避）。
位置決め動作、CP動作時に有効となります。

> 直線加減速の時のみ設定が有効となり、S字加減速の時は自動的にONとなります

設定値	内容
0	移動速度補正OFF
1	移動速度補正ON

初期値 : 1

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸とY軸のパルス出力を0に設定します。

C/C++

```
int nResult;
WORD wAxis;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmSetPulseConfig(0, wAxis, PMCM_PULSE_OUT, 0);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetPulseConfig(0, axis, PMCM_PULSE_OUT, 0);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.SetPulseConfig(0, axis, Pmcm.PMCM_PULSE_OUT, 0);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmSetPulseConfig(0, axis, PMCM_PULSE_OUT, 0)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmSetPulseConfig(0, intAxis, PMCM_PULSE_OUT, 0)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetPulseConfig(0, axis, PMCM_PULSE_OUT, 0);
```

PmcmSetSensorConfig

機能

センサの設定をします。

書式

```
INT PmcmSetSensorConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    WORD wConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

設定する項目を指定します。

設定値	内容

設定値	内容
PMCM_LOGIC	入力論理（センサ極性）
PMCM_EL_FUNC	EL信号入力ON時の処理
PMCM_SD_FUNC	SD信号入力ON時の処理
PMCM_SD_ACTIVE	SD信号入力方法
PMCM_ORG_FUNC	原点復帰方法
PMCM_EZ_FUNC	Z信号入力論理
PMCM_EZ_COUNT	Z信号カウント数
PMCM_ALM_FUNC	ALM信号入力ON時の処理
PMCM_LTC_FUNC	LTC信号入力論理
PMCM_PCS_FUNC	PCS信号入力の処理
PMCM_SENS_FILTER	入力フィルタの設定

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wConfig

設定値を指定します。
設定する項目（wMode）によって値が異なります。

- wMode : PMCM_LOGIC

  各センサの入力論理

bit	15	14	13	12	11	10	9	8
信号	-	-	-	-	-	-	-	-

bit	7	6	5	4	3	2	1	0
信号	-	-	PCS ※1,2	INP ※1	ALM	ORG	SD	±EL

各ビットの設定値	内容
0	オフで検知（正論理）
1	オンで検知（負論理）

> ※1 汎用デジタル入力として使用する場合は1に設定してください
> ※2 PCSはX軸のみ

初期値 : H'0（X軸）、H'20（Y軸）

- wMode : PMCM_EL_FUNC

  +EL、-EL信号がONになった時の停止方法

設定値	内容
0	即停止
1	減速停止

初期値 : 0

- wMode : PMCM_SD_FUNC

  SD信号がONになった時の減速方法

設定値	内容
0	減速のみ
1	減速停止
2	SD無効

初期値 : 0

- wMode : PMCM_SD_ACTIVE

  SD信号の入力方法

設定値	内容
0	レベル入力
1	ラッチ入力

初期値 : 0

- wMode : PMCM_ORG_FUNC

  ORG、Z信号を使用した原点復帰方法

設定値	内容
0	ORG信号のみ使用
1	ORG信号とZ信号を使用

初期値 : 0

- wMode : PMCM_EZ_FUNC

  Z信号の入力論理

設定値	内容
0	立ち下がりエッジ
1	立ち上がりエッジ

初期値 : 0

- wMode : PMCM_EZ_COUNT

  原点復帰完了条件のZ信号カウント数
  設定範囲は1 ～ 16

  初期値 : 1

- wMode : PMCM_ALM_FUNC

  ALM信号がONになった時の停止方法

設定値	内容
0	即停止
1	減速停止

初期値 : 0

- wMode : PMCM_LTC_FUNC

  LTC信号の入力論理

設定値	内容

設定値	内容
0	LTC信号の立ち下がりエッジでラッチ
1	LTC信号の立ち上がりエッジでラッチ

> 汎用デジタル入力として使用する場合は0に設定してください

初期値 : 0

- wMode : PMCM_PCS_FUNC

  PCS信号の機能

設定値	内容
0	PCS信号として使用しない
1	PCS信号として使用する（位置のオーバーライド）
2	自軸のみに有効な外部スタート／同時スタート信号

初期値 : 0

- wMode : PMCM_SENS_FILTER

設定値	内容
0	フィルタなし
1	3.2μsec以下のパルス幅は無視
2	25μsec以下のパルス幅は無視
3	100μsec以下のパルス幅は無視
4	1.6msec以下のパルス幅は無視

+EL、-EL、SD、ORG、ALM、INPにフィルタを挿入します。
フィルタを挿入すると、指定パルス幅以下の信号は無視されます。

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

入力論理（センサ極性）の設定をします。
IDが0のボードの、X軸とY軸のPCS、INP、ALM、ORG、SD、-EL、+ELを「オンで検知」に設定します。

C/C++

```
int nResult;
WORD wAxis;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmSetSensorConfig(0, wAxis, PMCM_LOGIC, 0x3F);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetSensorConfig(0, axis, PMCM_LOGIC, 0x3F);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.SetSensorConfig(0, axis, Pmcm.PMCM_LOGIC, 0x3F);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmSetSensorConfig(0, axis, PMCM_LOGIC, &H3F)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmSetSensorConfig(0, intAxis, PMCM_LOGIC, &H3F)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetSensorConfig(0, axis, PMCM_LOGIC, 0x3F);
```

PmcmSetSynchroConfig

機能

起動・停止条件の設定をします。

書式

```
INT PmcmSetSynchroConfig(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    WORD wConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

設定する項目を指定します。

設定値	内容

設定値	内容
PMCM_SYNC_START_MODE	起動条件
PMCM_SYNC_START	他軸の停止によるスタート条件
PMCM_SYNC_SIG_START	内部同期信号によるスタート条件
PMCM_SYNC_STOP_MODE	停止条件

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wConfig

設定値を指定します。
設定する項目（wMode）によって値が異なります。

- wMode : PMCM_SYNC_START_MODE

設定値	内容
0	スタート条件なし（即スタート）
1	PCS入力によるスタート※（Y軸は設定不可）
2	内部同期信号によるスタート
3	他軸の停止によるスタート

> ※ PCS信号を**自軸のみに有効な外部スタート／同時スタート信号**に設定する必要があります

初期値 : 0

- wMode : PMCM_SYNC_START

  停止確認する軸の設定（複数軸の指定が可能）

設定値	内容
PMCM_AXIS_X	X軸の停止でスタート
PMCM_AXIS_Y	Y軸の停止でスタート

初期値 : 0（設定軸なし）

- wMode : PMCM_SYNC_SIG_START

  同期信号の設定（同期信号出力を設定する必要があります）

設定値	内容
0	X軸が出力した内部同期信号でスタート
1	Y軸が出力した内部同期信号でスタート

  初期値 : 0

- wMode : PMCM_SYNC_STOP_MODE

  他軸の異常停止による停止の設定

設定値	内容
0	他軸の異常停止で停止しない
1	他軸の異常停止で停止する

  初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸とY軸の起動条件を即スタートに設定します。

C/C++

```
int nResult;
WORD wAxis;
```

```
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmSetSynchroConfig(0, wAxis, PMCM_SYNC_START_MODE, 0);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetSynchroConfig(0, axis, PMCM_SYNC_START_MODE, 0);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.SetSynchroConfig(0, axis, Pmcm.PMCM_SYNC_START_MODE, 0);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmSetSynchroConfig(0, axis, PMCM_SYNC_START_MODE, 0)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmSetSynchroConfig(0, intAxis, PMCM_SYNC_START_MODE, 0)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetSynchroConfig(0, axis, PMCM_SYNC_START_MODE, 0);
```

PmcmSetSynchroOut

機能

内部同期信号出力タイミングの設定をします。

書式

```
INT PmcmSetSynchroOut(
    WORD wID,
    WORD wAxis,
    WORD wMode,
    WORD wConfig
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

設定する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wMode

設定する項目を指定します。

設定値	内容

設定値	内容
PMCM_SYNC_OUT_MODE	内部同期信号出力タイミング

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wConfig

設定値を指定します。

- wMode : PMCM_SYNC_OUT_MODE

設定値	内容
0	内部同期信号出力なし
1	出力パルスコンパレータ条件成立時
2	エンコーダコンパレータ条件成立時
3	加速開始時
4	加速完了時
5	減速開始時
6	減速完了時

初期値 : 0

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸とY軸の内部同期信号出力タイミングを出力なしに設定します。

C/C++

```
int nResult;
WORD wAxis;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmSetSynchroOut(0, wAxis, PMCM_SYNC_OUT_MODE, 0);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetSynchroOut(0, axis, PMCM_SYNC_OUT_MODE, 0);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.SetSynchroOut(0, axis, Pmcm.PMCM_SYNC_OUT_MODE, 0);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmSetSynchroOut(0, axis, PMCM_SYNC_OUT_MODE, 0)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmSetSynchroOut(0, intAxis, PMCM_SYNC_OUT_MODE, 0)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmSetSynchroOut(0, axis, PMCM_SYNC_OUT_MODE, 0);
```

PmcmStartMotion

機能

モーター動作を開始します。

書式

```
INT PmcmStartMotion(
    WORD wID,
    WORD wAxis
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

動作を開始する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

動作パラメータに初期設定値はありませんので、本関数を実行する前にPmcmSetMotion関数にて動作パラメータを設定してください。

使用例

IDが0のボードの、X軸とY軸の動作を開始します。

C/C++

```
int nResult;
WORD wAxis;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmStartMotion(0, wAxis);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmStartMotion(0, axis);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.StartMotion(0, axis);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmStartMotion(0, axis)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmStartMotion(0, intAxis)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmStartMotion(0, axis);
```

PmcmStartMotionCp

機能

CP動作を開始します。

書式

```
INT PmcmStartMotionCp(
    WORD wID,
    WORD wAxis
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

動作を開始する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

本関数を実行したときに、既に実行中のCP動作がある場合は、その後に続けて実行されます。

動作パラメータに初期設定値はありませんので、本関数を実行する前にPmcmSetMotionCp関数にて動作パラメータを設定してください。

使用例

IDが0のボードの、X軸とY軸のCP動作を開始します。

C/C++

```
int nResult;
WORD wAxis;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmStartMotionCp(0, wAxis);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmStartMotionCp(0, axis);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.StartMotionCp(0, axis);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmStartMotionCp(0, axis)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmStartMotionCp(0, intAxis)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmStartMotionCp(0, axis);
```

PmcmStartMotionLine

機能

直線補間動作を開始します。

書式

```
INT PmcmStartMotionLine(
    WORD wID
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

動作パラメータに初期設定値はありませんので、本関数を実行する前にPmcmSetMotionLine関数にて動作パラメータを設定してください。

使用例

IDが0のボードの、直線補間動作を開始します。

C/C++

```
int nResult;
nResult = PmcmStartMotionLine(0);
```

C++/CLI

```
int result;
result = PmcmStartMotionLine(0);
```

C#

```
int result;
result = Pmcm.StartMotionLine(0);
```

VB（.NET2002以降）

```
Dim result As Integer
result = PmcmStartMotionLine(0)
```

VB6.0/VBA

```
Dim lngResult As Long
lngResult = PmcmStartMotionLine(0)
```

GCC

```
int32_t result;
result = PmcmStartMotionLine(0);
```

PmcmStopMotion

機能

モーター動作を停止します。

書式

```
INT PmcmStopMotion(
    WORD wID,
    WORD wAxis,
    WORD wStopMode
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

動作を停止する軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wStopMode

停止モードを指定します。

設定値	内容

設定値	内容
PMCM_DEC_STOP	減速停止
PMCM_IMMEDIATE_STOP	即停止

減速停止を指定した場合でも、定速起動で動作中の場合は即停止します

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

備考

CP動作時に実行した場合は、待ち状態となっている動作もクリアされます

使用例

IDが0のボードの、X軸とY軸のモーターを即停止します。

C/C++

```
int nResult;
WORD wAxis;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmStopMotion(0, wAxis, PMCM_IMMEDIATE_STOP);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmStopMotion(0, axis, PMCM_IMMEDIATE_STOP);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.StopMotion(0, axis, Pmcm.PMCM_IMMEDIATE_STOP);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmStopMotion(0, axis, PMCM_IMMEDIATE_STOP)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmStopMotion(0, intAxis, PMCM_IMMEDIATE_STOP)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmStopMotion(0, axis, PMCM_IMMEDIATE_STOP);
```

PmcmWaitForEvent（Linux）

> この関数はLinux用です。WindowsではWin32 APIのWaitForSingleObjectを使用してください。

機能

イベントの発生を待ちます。

イベントについて

書式

```
int32_t PmcmWaitForEvent(
    uint16_t wID,
    uint32_t milliseconds
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	GCC
型	uint16_t

milliseconds

タイムアウト（msec）を指定します。
0xFFFFFFFFを指定した場合、イベントが発生するまで待ちます。

言語	GCC
型	uint32_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	GCC

言語	GCC
型	int32_t

使用例

IDが0のボードの、イベントが発生するまで待ちます。

GCC

```
int32_t result;
result = PmcmWaitForEvent(0, 0xFFFFFFFF);
```

関数一覧

モーターコントローラLSI（PCL6123）のコマンドやレジスタに対して、書き込み・読み込みする場合に使用するAPI群です。
（通常はモーター制御API群を使用しますが、モーター制御API群にて動作を実現できない場合に本API群を使用してください）
各コマンド・レジスタ・ステータスの詳細についてはコントローラLSIのマニュアルを参照してください。

関数	機能
PmcmWriteCmdBuf	PCL6123のコマンドバッファへコマンドを書き込みます
PmcmWriteDataBuf	PCL6123のデータバッファへデータを書き込みます
PmcmReadDataBuf	PCL6123のデータバッファからデータを読み込みます
PmcmReadMainStatus	PCL6123のメインステータスを読み込みます
PmcmReadSubStatus	PCL6123のサブステータスを読み込みます

PmcmWriteCmdBuf

機能

モーターコントローラLSI（PCL6123）へコマンドを書き込みます。

書式

```
INT PmcmWriteCmdBuf(
    WORD wID,
    WORD wAxis,
    BYTE byCmd
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

書き込む軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

byCmd

書き込むコマンド値を指定します。

> 設定値の詳細はPCL6123のマニュアルを参照してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	BYTE	unsigned char	byte	Byte	Byte	uint8_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸とY軸のコマンドバッファへH'C0を書き込みます。

C/C++

```
int nResult;
WORD wAxis;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmWriteCmdBuf(0, wAxis, 0xC0);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmWriteCmdBuf(0, axis, 0xC0);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.WriteCmdBuf(0, axis, 0xC0);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmWriteCmdBuf(0, axis, &HC0)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmWriteCmdBuf(0, intAxis, &HC0)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmWriteCmdBuf(0, axis, 0xC0);
```

PmcmWriteDataBuf

機能

モーターコントローラLSI（PCL6123）へデータを書き込みます。

書式

```
INT PmcmWriteDataBuf(
    WORD wID,
    WORD wAxis,
    DWORD dwData
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

書き込む軸を指定します。複数の軸を指定することができます。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

dwData

書き込むデータを指定します。

> データの詳細はPCL6123のマニュアルを参照してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	DWORD	unsigned long	uint	Integer	Long	uint32_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸とY軸のデータバッファへ0を書き込みます。

C/C++

```
int nResult;
WORD wAxis;
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
nResult = PmcmWriteDataBuf(0, wAxis, 0);
```

C++/CLI

```
int result;
unsigned short axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmWriteDataBuf(0, axis, 0);
```

C#

```
int result;
ushort axis;
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
result = Pmcm.WriteDataBuf(0, axis, 0);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
axis = PMCM_AXIS_X + PMCM_AXIS_Y
result = PmcmWriteDataBuf(0, axis, 0)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
lngResult = PmcmWriteDataBuf(0, intAxis, 0)
```

GCC

```
int32_t result;
uint16_t axis;
axis = PMCM_AXIS_X + PMCM_AXIS_Y;
result = PmcmWriteDataBuf(0, axis, 0);
```

PmcmReadDataBuf

機能

モーターコントローラLSI（PCL6123）のデータを読み込みます。

書式

```
INT PmcmReadDataBuf(
    WORD wID,
    WORD wAxis,
    PDWORD pdwData
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

読み込む軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pdwData

読み込んだデータを格納するバッファへのポインタを指定します。

> データの詳細はPCL6123のマニュアルを参照してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PDWORD	unsigned long*	out uint	Integer	Long	uint32_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸のデータバッファからデータを読み込みます。

C/C++

```
int nResult;
WORD wAxis;
DWORD dwData;
wAxis = PMCM_AXIS_X;
nResult = PmcmReadDataBuf(0, wAxis, &dwData);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned long data;
axis = PMCM_AXIS_X;
result = PmcmReadDataBuf(0, axis, &data);
```

C#

```
int result;
ushort axis;
uint data;
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.ReadDataBuf(0, axis, out data);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim data As Integer
axis = PMCM_AXIS_X
result = PmcmReadDataBuf(0, axis, data)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim lngData As Long
intAxis = PMCM_AXIS_X
lngResult = PmcmReadDataBuf(0, intAxis, lngData)
```

GCC

```
int32_t result;
uint16_t axis;
uint32_t data;
axis = PMCM_AXIS_X;
result = PmcmReadDataBuf(0, axis, &data);
```

PmcmReadMainStatus

機能

モーターコントローラLSI（PCL6123）のメインステータスを読み込みます。

書式

```
INT PmcmReadMainStatus(
    WORD wID,
    WORD wAxis,
    PWORD pwData
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

読み込む軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pwData

読み込んだメインステータスを格納するバッファへのポインタを指定します。

> メインステータスの詳細はPCL6123のマニュアルを参照してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PWORD	unsigned short*	out ushort	Short	Integer	uint16_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸のメインステータスを読み込みます。

C/C++

```
int nResult;
WORD wAxis;
WORD wData;
wAxis = PMCM_AXIS_X;
nResult = PmcmReadMainStatus(0, wAxis, &wData);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned short data;
axis = PMCM_AXIS_X;
result = PmcmReadMainStatus(0, axis, &data);
```

C#

```
int result;
ushort axis;
ushort data;
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.ReadMainStatus(0, axis, out data);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim data As Short
axis = PMCM_AXIS_X
result = PmcmReadMainStatus(0, axis, data)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intData As Integer
intAxis = PMCM_AXIS_X
lngResult = PmcmReadMainStatus(0, intAxis, intData)
```

GCC

```
int32_t result;
uint16_t axis;
uint16_t data;
axis = PMCM_AXIS_X;
result = PmcmReadMainStatus(0, axis, &data);
```

PmcmReadSubStatus

機能

モーターコントローラLSI（PCL6123）のサブステータスを読み込みます。

書式

```
INT PmcmReadSubStatus(
    WORD wID,
    WORD wAxis,
    PBYTE pbyData
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wAxis

読み込む軸を指定します。複数の軸を指定することはできません。

設定値	内容
PMCM_AXIS_X	X軸
PMCM_AXIS_Y	Y軸

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pbyData

読み込んだサブステータスを格納するバッファへのポインタを指定します。

> サブステータスの詳細はPCL6123のマニュアルを参照してください

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PBYTE	unsigned char*	out byte	Byte	Byte	uint8_t*

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、X軸のサブステータスを読み込みます。

C/C++

```
int nResult;
WORD wAxis;
BYTE byData;
wAxis = PMCM_AXIS_X;
nResult = PmcmReadSubStatus(0, wAxis, &byData);
```

C++/CLI

```
int result;
unsigned short axis;
unsigned char data;
axis = PMCM_AXIS_X;
result = PmcmReadSubStatus(0, axis, &data);
```

C#

```
int result;
ushort axis;
byte data;
axis = Pmcm.PMCM_AXIS_X;
result = Pmcm.ReadSubStatus(0, axis, out data);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim axis As Short
Dim data As Byte
axis = PMCM_AXIS_X
result = PmcmReadSubStatus(0, axis, data)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim intAxis As Integer
Dim bytData As Byte
intAxis = PMCM_AXIS_X
lngResult = PmcmReadSubStatus(0, intAxis, bytData)
```

GCC

```
int32_t result;
uint16_t axis;
uint8_t data;
axis = PMCM_AXIS_X;
result = PmcmReadSubStatus(0, axis, &data);
```

関数一覧

デジタル入力・デジタル出力を制御します。

関数	機能
PmcmDioInput	入力端子の読み込みをおこないます
PmcmDioOutput	出力端子の制御をおこないます

PmcmDioInput

機能

任意の点数の入力端子の状態を読み込みます。

書式

```
INT PmcmDioInput(
    WORD wID,
    PBYTE pbyData,
    WORD wStart,
    WORD wCount
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pbyData

入力データを格納するバッファへのポインタを指定します。

設定値	内容
0	OFF
1	ON

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PBYTE	unsigned char*	byte	Byte	Byte	uint8_t*

wStart

入力開始番号（0～）を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wCount

入力の読み込みをおこなう数を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードの、IN0からIN4の入力端子の状態を読み込みます。
データはIN0から順にデータバッファへ格納されます。

C/C++

```
int nResult;
BYTE byData[5];
nResult = PmcmDioInput(0, byData, 0, 5);
```

C++/CLI

```
int result;
unsigned char data[5];
result = PmcmDioInput(0, data, 0, 5);
```

C#

```
int result;
byte[] inputData = new byte[5];
result = Pmcm.DioInput(0, inputData, 0, 5);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim inputData(4) As Byte
result = PmcmDioInput(0, inputData, 0, 5)
```

VB6.0/VBA

```
Dim lngResult As Long
Dim bytData(4) As Byte
lngResult = PmcmDioInput(0, bytData(0), 0, 5)
```

GCC

```
int32_t result;
uint8_t input_data[5];
result = PmcmDioInput(0, input_data, 0, 5);
```

PmcmDioOutput

機能

任意の点数の出力端子を制御します。

書式

```
INT PmcmDioOutput(
    WORD wID,
    PBYTE pbyData,
    WORD wStart,
    WORD wCount
);
```

パラメータ

wID

ボードのID番号を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

pbyData

出力データが格納されているバッファへのポインタを指定します。

設定値	内容
0	OFF
1	ON
2	現在の状態を保持

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	PBYTE	unsigned char*	byte	Byte	Byte	uint8_t*

wStart

出力開始番号（0～）を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

wCount

出力を制御する数を指定します。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	WORD	unsigned short	ushort	Short	Integer	uint16_t

戻り値

関数が正常に終了した場合は0（PMCM_RESULT_SUCCESS）が返ります。
正常に終了しなかった場合は0以外が返りますので、その場合はエラーコードを参照してください。

言語	C/C++	C++/CLI	C#	VB（.NET2002以降）	VB6.0/VBA	GCC
型	INT	int	int	Integer	Long	int32_t

使用例

IDが0のボードへ、OUT0はOFF、OUT1はON、OUT2は現在の出力状態、OUT3はOFFを出力する。

C/C++

```
int nResult;
BYTE byData[4];
byData[0] = 0;
byData[1] = 1;
byData[2] = 2;
byData[3] = 0;
nResult = PmcmDioOutput(0, byData, 0, 4);
```

C++/CLI

```
int result;
unsigned char data[4];
data[0] = 0;
data[1] = 1;
data[2] = 2;
data[3] = 0;
result = PmcmDioOutput(0, data, 0, 4);
```

C#

```
int result;
byte[] outputData = new byte[4];
outputData[0] = 0;
outputData[1] = 1;
outputData[2] = 2;
outputData[3] = 0;
result = Pmcm.DioOutput(0, outputData, 0, 4);
```

VB（.NET2002以降）

```
Dim result As Integer
Dim outputData(3) As Byte
outputData(0) = 0
outputData(1) = 1
outputData(2) = 2
outputData(3) = 0
result = PmcmDioOutput(0, outputData, 0, 4)
```

VB6.0/VBA

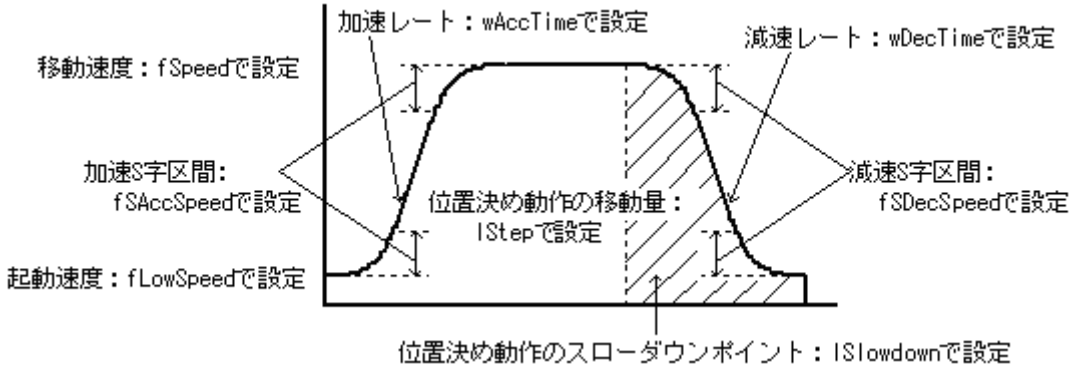
```
Dim lngResult As Long
Dim bytData(3) As Byte
bytData(0) = 0
bytData(1) = 1
bytData(2) = 2
bytData(3) = 0
lngResult = PmcmDioOutput(0, bytData(0), 0, 4)
```

GCC

```
int32_t result;
uint8_t output_data[4];
output_data[0] = 0;
output_data[1] = 1;
output_data[2] = 2;
output_data[3] = 0;
result = PmcmDioOutput(0, output_data, 0, 4);
```

動作パラメータ計算方法

動作パラメータにて設定された各値は下図の場所で使用されます。



速度倍率（fSpeedRate）

コントローラLSI内部で各軸ごとに速度倍率という値を持ち、それを元に出力パルスが作成されます。
速度倍率の設定値により、出力できる速度範囲及び速度設定間隔が決まります。

速度設定範囲最小値 = 速度倍率
速度設定範囲最大値 = 速度倍率 × 16383

高倍率になるほど設定できる速度間隔が粗くなりますので、できるだけ小さな倍率を設定してください。

動作パラメータに設定された値は速度倍率の値を元に再計算されて設定されます。
その際、近似値に設定されることがあります（速度設定値は速度倍率の倍数のみとなります）。

- 速度倍率と速度設定範囲の例

速度倍率	速度設定範囲	速度設定間隔
0.3	0.3～4914.9（0.3,0.6,0.9.....4914.6,4914.9）	0.3
0.5	0.5～8191.5（0.5,1,1.5.....8191,8191.5）	0.5
1	1～16383（1,2,3.....16382,16383）	1
2	2～32766（2,4,6.....32764,32766）	2
5	5～81915（5,10,15.....81910,81915）	5
10	10～163830（10,20,30.....163820,163830）	10
20	20～327660（20,40,60.....327640,327660）	20
50	50～819150（50,100,150.....819100,819150）	50
100	100～1638300（100,200,300.....1638200,1638300）	100

速度倍率	速度設定範囲	速度設定間隔
200	200～3276600（200,400,600.....3276400,3276600）	200
400	400～6553200（400,800,1200.....6552800,6553200）	400
600	600～9829800（600,1200,1800.....9829200,9829800）	600

起動速度（fLowSpeed）

1. 動作パラメータに設定された起動速度からコントローラの設定値を計算します

   起動速度設定レジスタ = fLowSpeed / 速度倍率

2. コントローラに設定された値から再計算された値が実際の起動速度になります

   起動速度 = 起動速度設定レジスタ × 速度倍率

- 計算例

  速度倍率 : 0.3
  起動速度 : 100

  コントローラの設定値
  100 / 0.3 = 333.333 ≒ 333

  実際の起動速度
  333 × 0.3 = 99.9[pps]

移動速度（fSpeed）

1. 動作パラメータに設定された移動速度からコントローラの設定値を計算します

   移動速度設定レジスタ = fSpeed / 速度倍率

2. コントローラに設定された値から再計算された値が実際の移動速度になります

   移動速度 = 移動速度設定レジスタ × 速度倍率

- 計算例

  速度倍率 : 0.3
  移動速度 : 1000

  コントローラの設定値
  1000 / 0.3 = 3333.333 ≒ 3333

  実際の移動速度
  3333 × 0.3 = 999.9[pps]

加速時間（wAccTime）、減速時間（wDecTime）

直線加減速
（加減速モードがPMCM_ACC_LINEAR）

1. 動作パラメータに設定された加速（減速）時間からコントローラの設定値を計算します

加速（減速）時間設定レジスタ = 加速（減速）時間 / 1000 × 19660800 / ((移動速度設定レジスタ - 起動速度設定レジスタ) × 2) - 1

> **Tip**
>
> 加速（減速）時間設定レジスタの設定範囲は1～16383です。
> 1未満になる場合は1に、16383を超える場合は16383に丸められます。

2. コントローラに設定された値から再計算された値が実際の加速（減速）時間になります

   加速（減速）時間 = (移動速度設定レジスタ - 起動速度設定レジスタ) × (加速（減速）時間設定レジスタ + 1) × 2 / 19660800 × 1000

- 計算例

  起動時速度レジスタ : 333
  移動速度レジスタ : 3333
  加速（減速）時間[msec] : 500

  コントローラの設定値
  500 / 1000 × 19660800 / ((3333 - 333) × 2) - 1 = 1637.4 ≒ 1637

  実際の加速（減速）時間
  (3333 - 333) × (1637 + 1) × 2 / 19660800 × 1000 ≒ 500[msec]

直線加減速部分のないS字加減速
（加減速モードがPMCM_ACC_SCURVEで加速（減速）S字区間が0）

1. 動作パラメータに設定された加速（減速）時間からコントローラの設定値を計算します

   加速（減速）時間設定レジスタ = 加速（減速）時間 / 1000 × 19660800 / ((移動速度設定レジスタ - 起動速度設定レジスタ) × 4) - 1

> **Tip**
>
> 加速（減速）時間設定レジスタの設定範囲は1～16383です。
> 1未満になる場合は1に、16383を超える場合は16383に丸められます。

2. コントローラに設定された値から再計算された値が実際の加速（減速）時間になります

   加速（減速）時間 = (移動速度設定レジスタ - 起動速度設定レジスタ) × (加速（減速）時間設定レジスタ + 1) × 4 / 19660800 × 1000

- 計算例

  起動時速度レジスタ : 333
  移動速度レジスタ : 3333
  加速（減速）時間[msec] : 500

  コントローラの設定値
  500 / 1000 × 19660800 / ((3333 - 333) × 4) - 1 = 818.2 ≒ 818

  実際の加速（減速）時間
  (3333 - 333) × (818 + 1) × 4 / 19660800 × 1000 ≒ 500[msec]

直線加減速部分のあるS字加減速
（加減速モードがPMCM_ACC_SCURVEで加速（減速）S字区間が0以外）

1. 動作パラメータに設定された加速（減速）時間からコントローラの設定値を計算します

   加速（減速）時間設定レジスタ = 加速（減速）時間 / 1000 × 19660800 / ((移動速度設定レジスタ - 起動速度設定レジスタ + 2 × 加速（減速）S字区間設定レジスタ) × 2) - 1

   > **Tip**
   >
   > 加速（減速）時間設定レジスタの設定範囲は1～16383です。
   > 1未満になる場合は1に、16383を超える場合は16383に丸められます。

2. コントローラに設定された値から再計算された値が実際の加速（減速）時間になります

   加速（減速）時間 = (移動速度設定レジスタ - 起動速度設定レジスタ + 2 × 加速（減速）S字区間設定レジスタ) × (加速（減速）時間設定レジスタ + 1) × 2 / 19660800 × 1000

- 計算例

  起動時速度レジスタ : 333
  移動速度レジスタ : 3333
  加速（減速）S字区間レジスタ : 1000
  加速（減速）時間[msec] : 500

  コントローラの設定値
  500 / 1000 × 19660800 / ((3333 - 333 + 2 × 1000) × 2) - 1 = 982.04 ≒ 982

  実際の加速（減速）時間
  (3333 - 333 + 2 × 1000) × (982 + 1) × 2 / 19660800 × 1000 ≒ 500[msec]

加速S字区間（fSAccSpeed）、減速S字区間（fSDecSpeed）

1. 動作パラメータに設定された加速（減速）S字区間からコントローラの設定値を計算します

   加速（減速）S字区間設定レジスタ = 1 / 速度倍率 × 加速（減速）S字区間

2. コントローラに設定された値から再計算された値が実際の加速（減速）S字区間になります

   加速（減速）S字区間 = 加速（減速）S字区間設定レジスタ × 速度倍率

起動速度～（起動速度 + 加速S字区間）までと、（移動速度 - 加速S字区間）～移動速度までがS字加速動作となり、中間部分は直線加速動作となります。
移動速度～（移動速度 - 減速S字区間）までと、（起動速度 + 減速S字区間）～起動速度までがS字減速動作となり、中間部分は直線減速動作となります。

- 計算例

  速度倍率 : 0.3
  加速（減速）S字区間 : 300

  コントローラの設定値
  1 / 0.3 × 300 = 999.99 ≒ 1000

  実際の加速（減速）S字区間
  1000 × 0.3 = 300[pps]

C/C++

Visual C++ .NET2002以降

1. 以下のファイルをプロジェクトフォルダにコピーします
   PmcMApi.h
   PmcM.lib
2. PmcMApi.hをプロジェクトに追加します
3. PmcM.libを以下の手順でプロジェクトに追加します
   メニューの[プロジェクト]-[プロパティ]を選択し、プロパティページのダイアログを開きます。
   ダイアログの左ペインで[構成プロパティ]-[リンカ]-[入力]を選択します。
   右ペインの[追加の依存ファイル]にPmcM.libと入力します。
4. ソースファイルにPmcMApi.hをインクルードします

Visual C++ 6.0

1. 以下のファイルをプロジェクトフォルダにコピーします
   PmcMApi.h
   PmcM.lib
2. PmcMApi.h、PmcM.libをプロジェクトに追加します
3. ソースファイルにPmcMApi.hをインクルードします

C++/CLI

1. 以下のファイルをプロジェクトフォルダにコピーします
   PmcMApiCLI.h
2. PmcMApiCLI.hをプロジェクトに追加します
3. ソースファイルにPmcMApiCLI.hをインクルードします
4. usingディレクティブを使ってPmcMCLIを宣言します（using namespace PmcMCLI;）

C#

1. 以下のファイルをプロジェクトフォルダにコピーします
   PmcM.cs
2. PmcM.csをプロジェクトに追加します
3. ソースファイルにusing ディレクティブを使ってPmcMCsを宣言します（using PmcMCs;）

Visual Basic（ .NET2002以降）

1. 以下のファイルをプロジェクトフォルダにコピーします
   PmcMApi.vb
2. PmcMApi.vbをプロジェクトに追加します

Visual Basic 6.0

1. 以下のファイルをプロジェクトフォルダにコピーします
   PmcMApi.bas
2. PmcMApi.basをプロジェクトに追加します

VBA

1. 以下のファイルをインポートします
   PmcMApi.bas

GCC

1. 以下のファイルをプロジェクトフォルダにコピーします（GCCサンプルに含まれています）
   PmcMApi.h
2. ソースファイルにPmcMApi.hをインクルードします
3. リンク時はライブラリとリンクするために以下を追加してください（GCCサンプルのmakefileも参考にしてください）

```
-L/usr/lib/y2c -lpmcm
```

サンプルプログラム

概要

Visual Basic 6.0、VBA、Visual C++ 6.0、Visual C# .NET2003、Visual Basic .NET2003、Visual C++/CLIのサンプルプログラムが付属しています。

サンプル内容

サンプル名	内容
DIO	汎用デジタル入力/出力
Motion1	連続動作
Motion2	条件スタート
MotionLine	直線補間動作
MotionCp	CP動作
Event	イベント発生

備考

付属しているサンプルを他のバージョンで使用する場合は以下のようにしてください。

- Visual C++ .NET2002以降
  Visual C++ 6.0のプロジェクトをVisual C++で開き、変換して使用してください。
- Visual C# 2005以降
  Visual C# .NET2003のプロジェクトをVisual C#で開き、変換して使用してください。
- Visual Basic 2005以降
  Visual Basic .NET2003のプロジェクトをVisual Basicで開き、変換して使用してください。

DIO

動作概要

汎用デジタル入力と汎用デジタル出力の制御をおこないます。

画面

1. オープン

   ボードのオープンをおこないます。

2. 設定

   デジタル入力は兼用端子です。
   デジタル入力として使用する際には設定をおこなう必要があります。（デジタル出力のみ使用する場合は実行する必要はありません）

3. 入力／出力

   入力及び出力をおこないます。

4. クローズ

   ボードのクローズをおこないます。
   オープンをおこなった場合は必ず実行する必要があります。

サンプルソース

- Visual C++
- Visual C++/CLI
- Visual C#
- Visual Basic（.NET2002以降）

- Visual Basic 6.0/VBA

オブジェクト

テキストボックス	ID	メンバ変数
ID番号	IDC_ID	m_wID
入力No.	IDC_INPUT_NO	m_wInputNo
出力No.	IDC_OUTPUT_NO	m_wOutputNo
出力データ	IDC_OUTPUT_DATA	m_byOutData

オープン

```
if( !UpdateData( TRUE ) ) {
    return;
}

int nResult;

nResult = PmcmOpen( m_wID, "PMC-M2C-U" );
if( nResult == PMCM_RESULT_SUCCESS ) {
    AfxMessageBox( "オープン成功", MB_OK | MB_ICONINFORMATION );
} else {
    AfxMessageBox( "オープン失敗", MB_OK | MB_ICONSTOP );
}
```

設定

```
int nResult;
WORD wAxis;
char szResult[64];

wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

// 汎用デジタル入力として使用する場合はINPとPCSをオンで検知(負論理)に設定
nResult = PmcmSetSensorConfig( m_wID, wAxis, PMCM_LOGIC, 0x30 );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetSensorConfig ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}

// 汎用デジタル入力として使用する場合はLTCを立ち下がりエッジに設定
nResult = PmcmSetSensorConfig( m_wID, wAxis, PMCM_LTC_FUNC, 0 );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetSensorConfig ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
```

```
    return;
}
```

入力

```
if( !UpdateData( TRUE ) ) {
    return;
}

int nResult;
BYTE byData;
char szInResult[64];

nResult = PmcmDioInput( m_wID, &byData, m_wInputNo, 1 );
if( nResult == PMCM_RESULT_SUCCESS ) {
    wsprintf( szInResult, "入力データ: %d", byData );
    AfxMessageBox( szInResult, MB_OK | MB_ICONINFORMATION );
} else {
    wsprintf( szInResult, "PmcmDioInput ERROR : 0x%X", nResult );
    AfxMessageBox( szInResult, MB_OK | MB_ICONSTOP );
}
```

出力

```
if( !UpdateData( TRUE ) ) {
    return;
}

int nResult;
char szOutResult[64];

nResult = PmcmDioOutput( m_wID, &m_byOutData, m_wOutputNo, 1 );
if( nResult == PMCM_RESULT_SUCCESS ) {
    AfxMessageBox( "出力正常終了", MB_OK | MB_ICONINFORMATION );
} else {
    wsprintf( szOutResult, "PmcmDioOutput ERROR : 0x%X", nResult );
    AfxMessageBox( szOutResult, MB_OK | MB_ICONSTOP );
}
```

クローズ

```
BOOL bResult;

bResult = PmcmClose( m_wID );
```

Visual C++/CLI

オブジェクト

テキストボックス	Name
ID番号	idTextBox
入力No.	inputNoTextBox
出力No.	outputNoTextBox
出力データ	outputDataTextBox

変数

```
unsigned short id;
```

オープン

```
int result;

id = Convert::ToUInt16(idTextBox->Text);
result = PmcmOpen(id, "PMC-M2C-U");
if (result == PMCM_RESULT_SUCCESS) {
    MessageBox::Show("オープン成功", "", MessageBoxButtons::OK, MessageBoxIcon::Information);
} else {
    MessageBox::Show("オープン失敗", "", MessageBoxButtons::OK, MessageBoxIcon::Error);
}
```

設定

```
int result;
unsigned short axis;
String^ messageText;

axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// 汎用デジタル入力として使用する場合はINPとPCSをオンで検知(負論理)に設定
result = PmcmSetSensorConfig(id, axis, PMCM_LOGIC, 0x30);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetSensorConfig ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}

// 汎用デジタル入力として使用する場合はLTCを立ち下がりエッジに設定
result = PmcmSetSensorConfig(id, axis, PMCM_LTC_FUNC, 0);
if (result != PMCM_RESULT_SUCCESS) {
```

```
    messageText = String::Format("PmcmSetSensorConfig ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

入力

```
int result;
unsigned char inputData;
unsigned short inputNo;
String^ messageText;

inputNo = Convert::ToUInt16(inputNoTextBox->Text);

result = PmcmDioInput(id, &inputData, inputNo, 1);
if (result == PMCM_RESULT_SUCCESS) {
    messageText = String::Format("入力データ : {0:D}", inputData);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Information);
} else {
    messageText = String::Format("PmcmDioInput ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
}
```

出力

```
int result;
unsigned char outputData;
unsigned short outputNo;
String^ messageText;

outputData = Convert::ToByte(outputDataTextBox->Text);
outputNo = Convert::ToUInt16(outputNoTextBox->Text);

result = PmcmDioOutput(id, &outputData, outputNo, 1);
if (result == PMCM_RESULT_SUCCESS) {
    MessageBox::Show("出力正常終了", "", MessageBoxButtons::OK, MessageBoxIcon::Information);
} else {
    messageText = String::Format("PmcmDioOutput ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
}
```

クローズ

```
bool result;

result = PmcmClose(id);
```

Visual C#

オブジェクト

テキストボックス	Name
ID番号	Id
入力No.	InputNo
出力No.	OutputNo
出力データ	OutputData

変数

```
ushort id;
```

オープン

```
int result;

id = Convert.ToUInt16(Id.Text);
result = Pmcm.Open( id, "PMC-M2C-U" );
if( result == Pmcm.PMCM_RESULT_SUCCESS )
{
    MessageBox.Show( "オープン成功", "", MessageBoxButtons.OK, MessageBoxIcon.Information );
}
else
{
    MessageBox.Show( "オープン失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
}
```

設定

```
int result;
ushort axis;
string outputString;

axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

// 汎用デジタル入力として使用する場合はINPとPCSをオンで検知(負論理)に設定
result = Pmcm.SetSensorConfig( id, axis, Pmcm.PMCM_LOGIC, 0x30 );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetSensorConfig ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

```
// 汎用デジタル入力として使用する場合はLTCを立ち下がりエッジに設定
result = Pmcm.SetSensorConfig( id, axis, Pmcm.PMCM_LTC_FUNC, 0 );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetSensorConfig ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

入力

```
int result;
byte[] inputData = new byte[1];
ushort inputNo;
string outputString;

inputNo = Convert.ToUInt16( InputNo.Text );

result = Pmcm.DioInput( id, inputData, inputNo, 1 );
if( result == Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "入力データ : {0:D}", inputData[0] );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Information );
}
else
{
    outputString = String.Format( "PmcmDioInput ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
}
```

出力

```
int result;
byte[] outputData = new byte[1];
ushort outputNo;
string outputString;

outputData[0] = Convert.ToByte( OutputData.Text );
outputNo = Convert.ToUInt16( OutputNo.Text );

result = Pmcm.DioOutput( id, outputData, outputNo, 1 );
if( result == Pmcm.PMCM_RESULT_SUCCESS )
{
    MessageBox.Show( "出力正常終了", "", MessageBoxButtons.OK, MessageBoxIcon.Information );
}
else
{
    outputString = String.Format( "PmcmDioOutput ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
}
```

クローズ

```
bool result;

result = Pmcm.Close( id );
```

Visual Basic（.NET2002以降）

オブジェクト

テキストボックス	Name
ID番号	idTextBox
入力No.	inputNoTextBox
出力No.	outputNoTextBox
出力データ	outputDataTextBox

変数

```
Dim id As Short
```

オープン

```
Dim result As Integer

id = Convert.ToInt16(idTextBox.Text)
result = PmcmOpen(id, "PMC-M2C-U")
If result = PMCM_RESULT_SUCCESS Then
    MessageBox.Show("オープン成功", "", MessageBoxButtons.OK, MessageBoxIcon.Information)
Else
    MessageBox.Show("オープン失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Error)
End If
```

設定

```
Dim result As Integer
Dim axis As Short

axis = PMCM_AXIS_X + PMCM_AXIS_Y

'汎用デジタル入力として使用する場合はINPとPCSをオンで検知(負論理)に設定
result = PmcmSetSensorConfig(id, axis, PMCM_LOGIC, &H30)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetSensorConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If

'汎用デジタル入力として使用する場合はLTCを立ち下がりエッジに設定
result = PmcmSetSensorConfig(id, axis, PMCM_LTC_FUNC, 0)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetSensorConfig ERROR : 0x" & result.ToString("X"), "",
```

```
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

入力

```
Dim result As Integer
Dim inputData(0) As Byte
Dim inputNo As Short

inputNo = Convert.ToInt16(inputNoTextBox.Text)

result = PmcmDioInput(id, inputData, inputNo, 1)
If result = PMCM_RESULT_SUCCESS Then
    MessageBox.Show("入力データ : " & inputData(0).ToString(), "", MessageBoxButtons.OK,
MessageBoxIcon.Information)
Else
    MessageBox.Show("PmcmDioInput ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
End If
```

出力

```
Dim result As Integer
Dim outputData(0) As Byte
Dim outputNo As Short

outputData(0) = Convert.ToByte(outputDataTextBox.Text)
outputNo = Convert.ToInt16(outputNoTextBox.Text)

result = PmcmDioOutput(id, outputData, outputNo, 1)
If result = PMCM_RESULT_SUCCESS Then
    MessageBox.Show("出力正常終了", "", MessageBoxButtons.OK, MessageBoxIcon.Information)
Else
    MessageBox.Show("PmcmDioOutput ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
End If
```

クローズ

```
Dim result As Boolean

result = PmcmClose(id)
```

Visual Basic 6.0/VBA

オブジェクト

テキストボックス	オブジェクト名
ID番号	txtID
入力No.	txtInputNo
出力No.	txtOutputNo
出力データ	txtOutputData

変数

```
Dim m_intID As Integer
```

オープン

```
Dim lngResult As Long
Dim strModelName As String

m_intID = Val(txtID.Text)
strModelName = "PMC-M2C-U"
lngResult = PmcmOpen(m_intID, strModelName)
If lngResult = PMCM_RESULT_SUCCESS Then
    MsgBox "オープン成功", vbInformation
Else
    MsgBox "オープン失敗", vbCritical
End If
```

設定

```
Dim lngResult As Long
Dim intAxis As Integer

intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

'汎用デジタル入力として使用する場合はINPとPCSをオンで検知(負論理)に設定
lngResult = PmcmSetSensorConfig(m_intID, intAxis, PMCM_LOGIC, &H30)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetSensorConfig ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If

'汎用デジタル入力として使用する場合はLTCを立ち下がりエッジに設定
lngResult = PmcmSetSensorConfig(m_intID, intAxis, PMCM_LTC_FUNC, 0)
If lngResult <> PMCM_RESULT_SUCCESS Then
```

```
    MsgBox "PmcmSetSensorConfig ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

入力

```
Dim lngResult As Long
Dim bytData As Byte
Dim intInputNo As Integer

intInputNo = Val(txtInputNo.Text)
lngResult = PmcmDioInput(m_intID, bytData, intInputNo, 1)
If lngResult = PMCM_RESULT_SUCCESS Then
    MsgBox "入力データ: " & bytData, vbInformation
Else
    MsgBox "PmcmDioInput ERROR : 0x" & Hex(lngResult), vbCritical
End If
```

出力

```
Dim lngResult As Long
Dim bytData As Byte
Dim intOutputNo As Integer

bytData = Val(txtOutputData.Text)
intOutputNo = Val(txtOutputNo.Text)
lngResult = PmcmDioOutput(m_intID, bytData, intOutputNo, 1)
If lngResult = PMCM_RESULT_SUCCESS Then
    MsgBox "正常終了", vbInformation
Else
    MsgBox "PmcmDioOutput ERROR : 0x" & Hex(lngResult), vbCritical
End If
```

クローズ

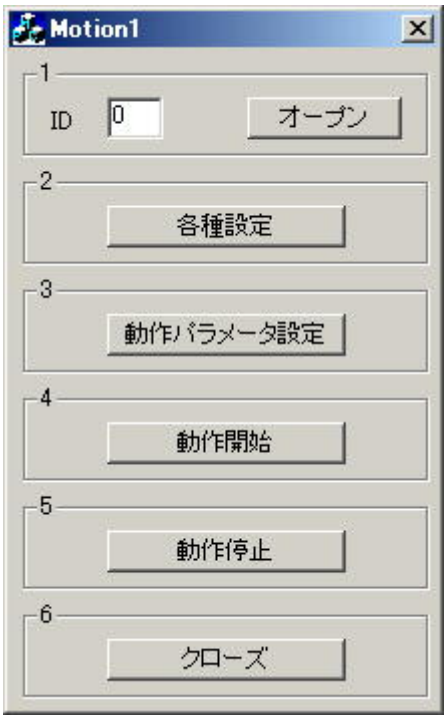
```
Dim blnResult As Boolean

blnResult = PmcmClose(m_intID)
```

Motion1

動作概要

X軸とY軸のモーターを連続動作させます。
X軸はCW（+）方向、Y軸はCCW（-）方向に動作させます。

画面



1. オープン

   ボードのオープンをおこないます。

2. 各種設定

   使用環境に応じて、センサ設定及びパルス出力設定をおこないます。

3. 動作パラメータ設定

   動作パラメータを設定します。

4. 動作開始

   モーター動作を開始します。

5. 動作停止

   モーター動作を停止します。

6. クローズ

   ボードのクローズをおこないます。
   オープンをおこなった場合は必ず実行する必要があります。

備考

⚠ 非常停止信号（EMG）・アラーム信号（ALM）・エンドリミット信号（+EL、-EL）がONの場合、モーターは動作しません

サンプルソース

- Visual C++
- Visual C++/CLI
- Visual C#
- Visual Basic（.NET2002以降）
- Visual Basic 6.0/VBA

オブジェクト

テキストボックス	ID	メンバ変数
ID番号	IDC_ID	m_wID

オープン

```
if( !UpdateData( TRUE ) ) {
    return;
}

int nResult;

nResult = PmcmOpen( m_wID, "PMC-M2C-U" );
if( nResult == PMCM_RESULT_SUCCESS ) {
    AfxMessageBox( "オープン成功", MB_OK | MB_ICONINFORMATION );
} else {
    AfxMessageBox( "オープン失敗", MB_OK | MB_ICONSTOP );
}
```

各種設定

```
int nResult;
WORD wAxis;
char szResult[64];

wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//nResult = PmcmSetSensorConfig( m_wID, wAxis, PMCM_LOGIC, 0x3F );
//if( nResult != PMCM_RESULT_SUCCESS ) {
//    wsprintf( szResult, "PmcmSetSensorConfig ERROR : 0x%X", nResult );
//    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
nResult = PmcmSetPulseConfig( m_wID, wAxis, PMCM_PULSE_OUT, 7 );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetPulseConfig ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

動作パラメータ設定

```
int nResult;
WORD wAxis;
MOTIONPMCM Motion[2];
char szResult[64];

wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

// X軸
Motion[0].wMoveMode = PMCM_JOG; // 動作モード
Motion[0].wStartMode = PMCM_CONST; // 起動モード
Motion[0].fSpeedRate = 1; // 速度倍率
Motion[0].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
Motion[0].fLowSpeed = 1000; // 起動時速度
Motion[0].fSpeed = 1000; // 移動速度
Motion[0].wAccTime = 0; // 加速時間
Motion[0].wDecTime = 0; // 減速時間
Motion[0].fSAccSpeed = 0; // 加速S字区間
Motion[0].fSDecSpeed = 0; // 減速S字区間
Motion[0].lSlowdown = -1; // スローダウンポイント
Motion[0].lStep = PMCM_DIR_CW; // 移動パルス数,移動方向
Motion[0].bAbsolutePtp = 0; // 絶対座標指定
// Y軸
Motion[1].wMoveMode = PMCM_JOG; // 動作モード
Motion[1].wStartMode = PMCM_CONST; // 起動モード
Motion[1].fSpeedRate = 1; // 速度倍率
Motion[1].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
Motion[1].fLowSpeed = 500; // 起動時速度
Motion[1].fSpeed = 500; // 移動速度
Motion[1].wAccTime = 0; // 加速時間
Motion[1].wDecTime = 0; // 減速時間
Motion[1].fSAccSpeed = 0; // 加速S字区間
Motion[1].fSDecSpeed = 0; // 減速S字区間
Motion[1].lSlowdown = -1; // スローダウンポイント
Motion[1].lStep = PMCM_DIR_CCW; // 移動パルス数,移動方向
Motion[1].bAbsolutePtp = 0; // 絶対座標指定
// 動作パラメータ設定
nResult = PmcmSetMotion( m_wID, wAxis, Motion );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetMotion ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

動作開始

```
int nResult;
WORD wAxis;
char szResult[64];

// 動作させる軸
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

// 動作開始
nResult = PmcmStartMotion( m_wID, wAxis );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmStartMotion ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

動作停止

```
int nResult;
WORD wAxis;
WORD wStopMode;
char szResult[64];

// 停止させる軸
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

// 停止モード
wStopMode = PMCM_IMMEDIATE_STOP;

// 動作停止
nResult = PmcmStopMotion( m_wID, wAxis, wStopMode );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmStopMotion ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

クローズ

```
BOOL bResult;

bResult = PmcmClose( m_wID );
```

Visual C++/CLI

オブジェクト

テキストボックス	Name
ID番号	idTextBox

変数

```
unsigned short id;
```

オープン

```
int result;

id = Convert::ToUInt16(idTextBox->Text);
result = PmcmOpen(id, "PMC-M2C-U");
if (result == PMCM_RESULT_SUCCESS) {
    MessageBox::Show("オープン成功", "", MessageBoxButtons::OK, MessageBoxIcon::Information);
} else {
    MessageBox::Show("オープン失敗", "", MessageBoxButtons::OK, MessageBoxIcon::Error);
}
```

各種設定

```
int result;
unsigned short axis;
String^ messageText;

axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//result = PmcmSetSensorConfig(id, axis, PMCM_LOGIC, 0x3F);
//if (result != PMCM_RESULT_SUCCESS) {
//    messageText = String::Format("PmcmSetSensorConfig ERROR : 0x{0:X}", result);
//    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
result = PmcmSetPulseConfig(id, axis, PMCM_PULSE_OUT, 7);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetPulseConfig ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

動作パラメータ設定

```
int result;
unsigned short axis;
MOTIONPMCM motion[2];
String^ messageText;

axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// X軸
motion[0].wMoveMode = PMCM_JOG; // 動作モード
motion[0].wStartMode = PMCM_CONST; // 起動モード
motion[0].fSpeedRate = 1; // 速度倍率
motion[0].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
motion[0].fLowSpeed = 1000; // 起動時速度
motion[0].fSpeed = 1000; // 移動速度
motion[0].wAccTime = 0; // 加速時間
motion[0].wDecTime = 0; // 減速時間
motion[0].fSAccSpeed = 0; // 加速S字区間
motion[0].fSDecSpeed = 0; // 減速S字区間
motion[0].lSlowdown = -1; // スローダウンポイント
motion[0].lStep = PMCM_DIR_CW; // 移動パルス数,移動方向
motion[0].bAbsolutePtp = 0; // 絶対座標指定
// Y軸
motion[1].wMoveMode = PMCM_JOG; // 動作モード
motion[1].wStartMode = PMCM_CONST; // 起動モード
motion[1].fSpeedRate = 1; // 速度倍率
motion[1].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
motion[1].fLowSpeed = 500; // 起動時速度
motion[1].fSpeed = 500; // 移動速度
motion[1].wAccTime = 0; // 加速時間
motion[1].wDecTime = 0; // 減速時間
motion[1].fSAccSpeed = 0; // 加速S字区間
motion[1].fSDecSpeed = 0; // 減速S字区間
motion[1].lSlowdown = -1; // スローダウンポイント
motion[1].lStep = PMCM_DIR_CCW; // 移動パルス数,移動方向
motion[1].bAbsolutePtp = 0; // 絶対座標指定
// 動作パラメータ設定
result = PmcmSetMotion(id, axis, motion);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetMotion ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

動作開始

```
int result;
unsigned short axis;
String^ messageText;

// 動作させる軸
axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// 動作開始
result = PmcmStartMotion(id, axis);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmStartMotion ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

動作停止

```
int result;
unsigned short axis;
unsigned short stopMode;
String^ messageText;

// 停止させる軸
axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// 停止モード
stopMode = PMCM_IMMEDIATE_STOP;

// 動作停止
result = PmcmStopMotion(id, axis, stopMode);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmStopMotion ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

クローズ

```
bool result;

result = PmcmClose(id);
```

Visual C#

オブジェクト

テキストボックス	Name
ID番号	Id

変数

```
ushort id;
```

オープン

```
int result;

id = Convert.ToUInt16(Id.Text);
result = Pmcm.Open( id, "PMC-M2C-U" );
if( result == Pmcm.PMCM_RESULT_SUCCESS )
{
    MessageBox.Show( "オープン成功", "", MessageBoxButtons.OK, MessageBoxIcon.Information );
}
else
{
    MessageBox.Show( "オープン失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
}
```

各種設定

```
int result;
ushort axis;
string outputString;

axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//result = Pmcm.SetSensorConfig( id, axis, Pmcm.PMCM_LOGIC, 0x3F );
//if( result != Pmcm.PMCM_RESULT_SUCCESS )
//{
//    outputString = String.Format( "PmcmSetSensorConfig ERROR : 0x{0:X}", result );
//    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
result = Pmcm.SetPulseConfig( id, axis, Pmcm.PMCM_PULSE_OUT, 7 );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetPulseConfig ERROR : 0x{0:X}", result );
```

```
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

動作パラメータ設定

```
int result;
ushort axis;
Pmcm.MOTIONPMCM[] motion = new Pmcm.MOTIONPMCM[2];
string outputString;

axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

// X軸
motion[0].wMoveMode = Pmcm.PMCM_JOG; // 動作モード
motion[0].wStartMode = Pmcm.PMCM_CONST; // 起動モード
motion[0].fSpeedRate = 1; // 速度倍率
motion[0].wAccDecMode = Pmcm.PMCM_ACC_LINEAR; // 加減速モード
motion[0].fLowSpeed = 1000; // 起動時速度
motion[0].fSpeed = 1000; // 移動速度
motion[0].wAccTime = 0; // 加速時間
motion[0].wDecTime = 0; // 減速時間
motion[0].fSAccSpeed = 0; // 加速S字区間
motion[0].fSDecSpeed = 0; // 減速S字区間
motion[0].lSlowdown = -1; // スローダウンポイント
motion[0].lStep = Pmcm.PMCM_DIR_CW; // 移動パルス数,移動方向
motion[0].bAbsolutePtp = 0; // 絶対座標指定
// Y軸
motion[1].wMoveMode = Pmcm.PMCM_JOG; // 動作モード
motion[1].wStartMode = Pmcm.PMCM_CONST; // 起動モード
motion[1].fSpeedRate = 1; // 速度倍率
motion[1].wAccDecMode = Pmcm.PMCM_ACC_LINEAR; // 加減速モード
motion[1].fLowSpeed = 500; // 起動時速度
motion[1].fSpeed = 500; // 移動速度
motion[1].wAccTime = 0; // 加速時間
motion[1].wDecTime = 0; // 減速時間
motion[1].fSAccSpeed = 0; // 加速S字区間
motion[1].fSDecSpeed = 0; // 減速S字区間
motion[1].lSlowdown = -1; // スローダウンポイント
motion[1].lStep = Pmcm.PMCM_DIR_CCW; // 移動パルス数,移動方向
motion[1].bAbsolutePtp = 0; // 絶対座標指定
// 動作パラメータ設定
result = Pmcm.SetMotion( id, axis, motion );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetMotion ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

動作開始

```
int result;
ushort axis;
string outputString;

// 動作させる軸
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
```

```
// 動作開始
result = Pmcm.StartMotion( id, axis );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmStartMotion ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

動作停止

```
int result;
ushort axis;
ushort stopMode;
string outputString;

// 停止させる軸
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

// 停止モード
stopMode = Pmcm.PMCM_IMMEDIATE_STOP;

// 動作停止
result = Pmcm.StopMotion( id, axis, stopMode );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmStopMotion ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

クローズ

```
bool result;

result = Pmcm.Close( id );
```

Visual Basic（.NET2002以降）

オブジェクト

テキストボックス	Name
ID番号	idTextBox

変数

```
Dim id As Short
```

オープン

```
Dim result As Integer

id = Convert.ToInt16(idTextBox.Text)
result = PmcmOpen(id, "PMC-M2C-U")
If result = PMCM_RESULT_SUCCESS Then
    MessageBox.Show("オープン成功", "", MessageBoxButtons.OK, MessageBoxIcon.Information)
Else
    MessageBox.Show("オープン失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Error)
End If
```

各種設定

```
Dim result As Integer
Dim axis As Short

axis = PMCM_AXIS_X + PMCM_AXIS_Y

'センサ設定
'オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
'その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
'result = PmcmSetSensorConfig(id, axis, PMCM_LOGIC, &H3F)
'If result <> PMCM_RESULT_SUCCESS Then
'    MessageBox.Show("PmcmSetSensorConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
'    Exit Sub
'End If

'パルス出力モード設定
'使用しているドライバに合致したパルス出力モードを選択してください
result = PmcmSetPulseConfig(id, axis, PMCM_PULSE_OUT, 7)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetPulseConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

動作パラメータ設定

```
Dim result As Integer
Dim axis As Short
Dim motion(1) As MOTIONPMCM

axis = PMCM_AXIS_X + PMCM_AXIS_Y

'X軸
motion(0).wMoveMode = PMCM_JOG '動作モード
motion(0).wStartMode = PMCM_CONST '起動モード
motion(0).fSpeedRate = 1 '速度倍率
motion(0).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
motion(0).fLowSpeed = 1000 '起動時速度
motion(0).fSpeed = 1000 '移動速度
motion(0).wAccTime = 0 '加速時間
motion(0).wDecTime = 0 '減速時間
motion(0).fSAccSpeed = 0 '加速S字区間
motion(0).fSDecSpeed = 0 '減速S字区間
motion(0).lSlowdown = -1 'スローダウンポイント
motion(0).lStep = PMCM_DIR_CW '移動パルス数,移動方向
motion(0).bAbsolutePtp = 0 '絶対座標指定
'Y軸
motion(1).wMoveMode = PMCM_JOG '動作モード
motion(1).wStartMode = PMCM_CONST '起動モード
motion(1).fSpeedRate = 1 '速度倍率
motion(1).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
motion(1).fLowSpeed = 500 '起動時速度
motion(1).fSpeed = 500 '移動速度
motion(1).wAccTime = 0 '加速時間
motion(1).wDecTime = 0 '減速時間
motion(1).fSAccSpeed = 0 '加速S字区間
motion(1).fSDecSpeed = 0 '減速S字区間
motion(1).lSlowdown = -1 'スローダウンポイント
motion(1).lStep = PMCM_DIR_CCW '移動パルス数,移動方向
motion(1).bAbsolutePtp = 0 '絶対座標指定
'動作パラメータ設定
result = PmcmSetMotion(id, axis, motion)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetMotion ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

動作開始

```
Dim result As Integer
Dim axis As Short

'動作させる軸
axis = PMCM_AXIS_X + PMCM_AXIS_Y

'動作開始
result = PmcmStartMotion(id, axis)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmStartMotion ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

動作停止

```
Dim result As Integer
Dim axis As Short
Dim stopMode As Short

'停止させる軸
axis = PMCM_AXIS_X + PMCM_AXIS_Y

'停止モード
stopMode = PMCM_IMMEDIATE_STOP

'動作停止
result = PmcmStopMotion(id, axis, stopMode)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmStopMotion ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

クローズ

```
Dim result As Boolean

result = PmcmClose(id)
```

Visual Basic 6.0/VBA

オブジェクト

テキストボックス	オブジェクト名
ID番号	txtID

変数

```
Dim m_intID As Integer
```

オープン

```
Dim lngResult As Long
Dim strModelName As String

m_intID = Val(txtID.Text)
strModelName = "PMC-M2C-U"
lngResult = PmcmOpen(m_intID, strModelName)
If lngResult = PMCM_RESULT_SUCCESS Then
    MsgBox "オープン成功", vbInformation
Else
    MsgBox "オープン失敗", vbCritical
End If
```

各種設定

```
Dim lngResult As Long
Dim intAxis As Integer

intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

'センサ設定
'オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
'その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
'lngResult = PmcmSetSensorConfig(m_intID, intAxis, PMCM_LOGIC, &H3F)
'If lngResult <> PMCM_RESULT_SUCCESS Then
'    MsgBox "PmcmSetSensorConfig ERROR : 0x" & Hex(lngResult), vbCritical
'    Exit Sub
'End If

'パルス出力モード設定
'使用しているドライバに合致したパルス出力モードを選択してください
lngResult = PmcmSetPulseConfig(m_intID, intAxis, PMCM_PULSE_OUT, 7)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetPulseConfig ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

動作パラメータ設定

```
Dim lngResult As Long
Dim intAxis As Integer
Dim Motion(1) As MOTIONPMCM

intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

'X軸
Motion(0).wMoveMode = PMCM_JOG '動作モード
Motion(0).wStartMode = PMCM_CONST '起動モード
Motion(0).fSpeedRate = 1 '速度倍率
Motion(0).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
Motion(0).fLowSpeed = 1000 '起動時速度
Motion(0).fSpeed = 1000 '移動速度
Motion(0).wAccTime = 0 '加速時間
Motion(0).wDecTime = 0 '減速時間
Motion(0).fSAccSpeed = 0 '加速S字区間
Motion(0).fSDecSpeed = 0 '減速S字区間
Motion(0).lSlowdown = -1 'スローダウンポイント
Motion(0).lStep = PMCM_DIR_CW '移動パルス数,移動方向
Motion(0).bAbsolutePtp = 0 '絶対座標指定
'Y軸
Motion(1).wMoveMode = PMCM_JOG '動作モード
Motion(1).wStartMode = PMCM_CONST '起動モード
Motion(1).fSpeedRate = 1 '速度倍率
Motion(1).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
Motion(1).fLowSpeed = 500 '起動時速度
Motion(1).fSpeed = 500 '移動速度
Motion(1).wAccTime = 0 '加速時間
Motion(1).wDecTime = 0 '減速時間
Motion(1).fSAccSpeed = 0 '加速S字区間
Motion(1).fSDecSpeed = 0 '減速S字区間
Motion(1).lSlowdown = -1 'スローダウンポイント
Motion(1).lStep = PMCM_DIR_CCW '移動パルス数,移動方向
Motion(1).bAbsolutePtp = 0 '絶対座標指定
'動作パラメータ設定
lngResult = PmcmSetMotion(m_intID, intAxis, Motion(0))
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetMotion ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

動作開始

```
Dim lngResult As Long
Dim intAxis As Integer

'動作させる軸
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

'動作開始
lngResult = PmcmStartMotion(m_intID, intAxis)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmStartMotion ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

動作停止

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intStopMode As Integer

'停止させる軸
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

'停止モード
intStopMode = PMCM_IMMEDIATE_STOP

'動作停止
lngResult = PmcmStopMotion(m_intID, intAxis, intStopMode)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmStopMotion ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

クローズ

```
Dim blnResult As Boolean

'ボードクローズ
blnResult = PmcmClose(m_intID)
```

Motion2

動作概要

X軸とY軸を条件スタートさせます。

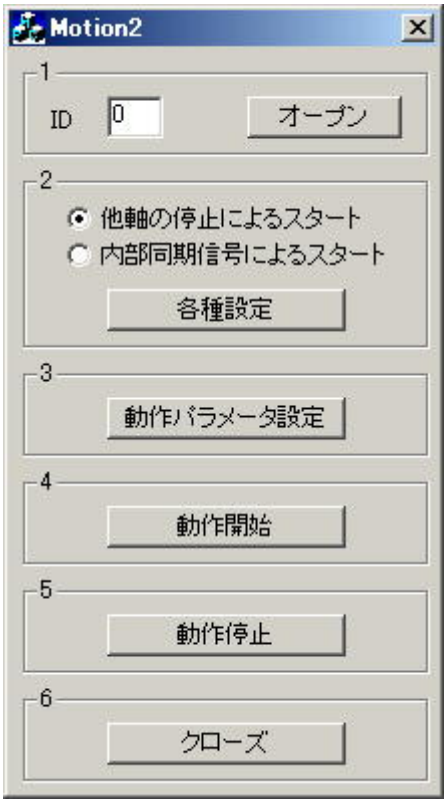
- 他軸の停止によるスタート

  Y軸の停止によりX軸が動作します。

- 内部同期信号によるスタート

  X軸の出力パルスカウンタが500になったらY軸が動作します。

画面



1. オープン

   ボードのオープンをおこないます。

2. 各種設定

   使用環境に応じて、センサ設定及びパルス出力設定をおこないます。
   [他軸の停止によるスタート]または[内部同期信号によるスタート]の設定をおこないます。

3. 動作パラメータ設定

   動作パラメータを設定します。

4. 動作開始

   モーター動作を開始します。

5. 動作停止

モーター動作を停止します。
既に停止している場合、実行する必要はありません。

6. クローズ
ボードのクローズをおこないます。
オープンをおこなった場合は必ず実行する必要があります。

備考

内部同期信号によるスタートを選択した場合、動作開始を複数回実行しても、最初の実行時のみ正常に動作します（2回目以降はX軸の出力パルスカウンタが500にならないため）。

> ⚠ 非常停止信号（EMG）・アラーム信号（ALM）・エンドリミット信号（+EL、-EL）がONの場合、モーターは動作しません

サンプルソース

- Visual C++
- Visual C++/CLI
- Visual C#
- Visual Basic（.NET2002以降）
- Visual Basic 6.0/VBA

オブジェクト

テキストボックス	ID	メンバ変数
ID番号	IDC_ID	m_wID

ラジオボタン	ID
他軸の停止によるスタート	IDC_STARTMODE_0
内部同期信号によるスタート	IDC_STARTMODE_1

オープン

```
if( !UpdateData( TRUE ) ) {
    return;
}

int nResult;

nResult = PmcmOpen( m_wID, "PMC-M2C-U" );
if( nResult == PMCM_RESULT_SUCCESS ) {
    AfxMessageBox( "オープン成功", MB_OK | MB_ICONINFORMATION );
} else {
    AfxMessageBox( "オープン失敗", MB_OK | MB_ICONSTOP );
}
```

各種設定

```
int nResult;
WORD wAxis;
char szResult[64];
COMPPMCM Comp[2];
CButton* pMode0 = NULL;
CButton* pMode1 = NULL;

wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//nResult = PmcmSetSensorConfig( m_wID, wAxis, PMCM_LOGIC, 0x3F );
//if( nResult != PMCM_RESULT_SUCCESS ) {
//    wsprintf( szResult, "PmcmSetSensorConfig ERROR : 0x%X", nResult );
//    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
//    return;
//}
```

```
// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
nResult = PmcmSetPulseConfig( m_wID, wAxis, PMCM_PULSE_OUT, 7 );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetPulseConfig ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}

pMode0 = (CButton*)GetDlgItem( IDC_STARTMODE_0 );
pMode1 = (CButton*)GetDlgItem( IDC_STARTMODE_1 );
if( pMode0->GetCheck() ) {
    // 他軸の停止によるスタート
    // Y軸の停止によりX軸がスタートします

    // X軸の起動条件を設定します(他軸の停止によるスタート)
    nResult = PmcmSetSynchroConfig( m_wID, PMCM_AXIS_X, PMCM_SYNC_START_MODE, 3 );
    if( nResult != PMCM_RESULT_SUCCESS ) {
        wsprintf( szResult, "PmcmSetSynchroConfig ERROR : 0x%X", nResult );
        AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
        return;
    }

    // X軸の他軸の停止によるスタート条件を設定します(Y軸の停止確認);
    nResult = PmcmSetSynchroConfig( m_wID, PMCM_AXIS_X, PMCM_SYNC_START, PMCM_AXIS_Y );
    if( nResult != PMCM_RESULT_SUCCESS ) {
        wsprintf( szResult, "PmcmSetSynchroConfig ERROR : 0x%X", nResult );
        AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
        return;
    }
} else if( pMode1->GetCheck() ) {
    // 内部同期信号によるスタート
    // X軸の出力パルスカウンタが500になったらY軸がスタートします

    // Y軸の起動条件を設定(内部同期信号によるスタート)
    nResult = PmcmSetSynchroConfig( m_wID, PMCM_AXIS_Y, PMCM_SYNC_START_MODE, 2);
    if( nResult != PMCM_RESULT_SUCCESS ) {
        wsprintf( szResult, "PmcmSetSynchroConfig ERROR : 0x%X", nResult );
        AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
        return;
    }

    // Y軸の内部同期信号によるスタート条件の設定(X軸の内部同期信号でスタート)
    nResult = PmcmSetSynchroConfig( m_wID, PMCM_AXIS_Y, PMCM_SYNC_SIG_START, 0);
    if( nResult != PMCM_RESULT_SUCCESS ) {
        wsprintf( szResult, "PmcmSetSynchroConfig ERROR : 0x%X", nResult );
        AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
        return;
    }

    // X軸の内部同期信号出力タイミングの設定(出力パルスコンパレータ条件成立時)
    nResult = PmcmSetSynchroOut( m_wID, PMCM_AXIS_X, PMCM_SYNC_OUT_MODE, 1);
    if( nResult != PMCM_RESULT_SUCCESS ) {
        wsprintf( szResult, "PmcmSetSynchroOut ERROR : 0x%X", nResult );
        AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
        return;
    }

    // X軸のコンパレータの設定
    Comp[0].wConfig = 1;
    Comp[0].lCount = 500;
    nResult = PmcmSetComparatorConfig( m_wID, PMCM_AXIS_X, PMCM_COMP_COUNTER, Comp );
    if( nResult != PMCM_RESULT_SUCCESS ) {
        wsprintf( szResult, "PmcmSetComparatorConfig ERROR : 0x%X", nResult );
```

```
        AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
        return;
    }
}
```

動作パラメータ設定

```
int nResult;
WORD wAxis;
MOTIONPMCM Motion[2];
char szResult[64];

wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

// X軸
Motion[0].wMoveMode = PMCM_PTP; // 動作モード
Motion[0].wStartMode = PMCM_CONST; // 起動モード
Motion[0].fSpeedRate = 1; // 速度倍率
Motion[0].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
Motion[0].fLowSpeed = 500; // 起動時速度
Motion[0].fSpeed = 500; // 移動速度
Motion[0].wAccTime = 0; // 加速時間
Motion[0].wDecTime = 0; // 減速時間
Motion[0].fSAccSpeed = 0; // 加速S字区間
Motion[0].fSDecSpeed = 0; // 減速S字区間
Motion[0].lSlowdown = -1; // スローダウンポイント
Motion[0].lStep = 1000; // 移動パルス数,移動方向
Motion[0].bAbsolutePtp = 0; // 絶対座標指定
// Y軸
Motion[1].wMoveMode = PMCM_PTP; // 動作モード
Motion[1].wStartMode = PMCM_CONST; // 起動モード
Motion[1].fSpeedRate = 1; // 速度倍率
Motion[1].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
Motion[1].fLowSpeed = 1000; // 起動時速度
Motion[1].fSpeed = 1000; // 移動速度
Motion[1].wAccTime = 0; // 加速時間
Motion[1].wDecTime = 0; // 減速時間
Motion[1].fSAccSpeed = 0; // 加速S字区間
Motion[1].fSDecSpeed = 0; // 減速S字区間
Motion[1].lSlowdown = -1; // スローダウンポイント
Motion[1].lStep = 1000; // 移動パルス数,移動方向
Motion[1].bAbsolutePtp = 0; // 絶対座標指定
// 動作パラメータ設定
nResult = PmcmSetMotion( m_wID, wAxis, Motion );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetMotion ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

動作開始

```
int nResult;
WORD wAxis;
char szResult[64];
CButton* pMode0 = NULL;
CButton* pMode1 = NULL;

pMode0 = (CButton*)GetDlgItem( IDC_STARTMODE_0 );
```

```
pMode1 = (CButton*)GetDlgItem( IDC_STARTMODE_1 );
if( pMode0->GetCheck() ) {
    // 他軸の停止によるスタート
    // X軸はY軸の停止を確認して動作しますので、Y軸を先に動作させます
    // 同時に動作させてしまうと、その時点でY軸がまだ動作開始前で停止中のため、X軸が動作してしまいます

    // 動作開始
    nResult = PmcmStartMotion( m_wID, PMCM_AXIS_Y );
    if( nResult != PMCM_RESULT_SUCCESS ) {
        wsprintf( szResult, "PmcmStartMotion ERROR : 0x%X", nResult );
        AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
        return;
    }
    // 動作開始
    nResult = PmcmStartMotion( m_wID, PMCM_AXIS_X );
    if( nResult != PMCM_RESULT_SUCCESS ) {
        wsprintf( szResult, "PmcmStartMotion ERROR : 0x%X", nResult );
        AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
        return;
    }
} else if( pMode1->GetCheck() ) {
    // 内部同期信号によるスタート

    // 動作開始
    wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;
    nResult = PmcmStartMotion( m_wID, wAxis );
    if( nResult != PMCM_RESULT_SUCCESS ) {
        wsprintf( szResult, "PmcmStartMotion ERROR : 0x%X", nResult );
        AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
        return;
    }
}
```

動作停止

```
int nResult;
WORD wAxis;
WORD wStopMode;
char szResult[64];

// 停止させる軸
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

// 停止モード
wStopMode = PMCM_IMMEDIATE_STOP;

// 動作停止
nResult = PmcmStopMotion( m_wID, wAxis, wStopMode );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmStopMotion ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

クローズ

```
BOOL bResult;

bResult = PmcmClose( m_wID );
```

Visual C++/CLI

オブジェクト

テキストボックス	Name
ID番号	idTextBox

ラジオボタン	Name
他軸の停止によるスタート	startMode0
内部同期信号によるスタート	startMode1

変数

```
unsigned short id;
```

オープン

```
int result;

id = Convert::ToUInt16(idTextBox->Text);
result = PmcmOpen(id, "PMC-M2C-U");
if (result == PMCM_RESULT_SUCCESS) {
    MessageBox::Show("オープン成功", "", MessageBoxButtons::OK, MessageBoxIcon::Information);
} else {
    MessageBox::Show("オープン失敗", "", MessageBoxButtons::OK, MessageBoxIcon::Error);
}
```

各種設定

```
int result;
unsigned short axis;
COMPPMCM comp[2];
String^ messageText;

axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//result = PmcmSetSensorConfig(id, axis, PMCM_LOGIC, 0x3F);
//if (result != PMCM_RESULT_SUCCESS) {
//    messageText = String::Format("PmcmSetSensorConfig ERROR : 0x{0:X}", result);
//    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
//    return;
//}
```

```
// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
result = PmcmSetPulseConfig(id, axis, PMCM_PULSE_OUT, 7);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetPulseConfig ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}

if (startMode0->Checked) {
    // 他軸の停止によるスタート
    // Y軸の停止によりX軸がスタートします

    // X軸の起動条件を設定します(他軸の停止によるスタート)
    result = PmcmSetSynchroConfig(id, PMCM_AXIS_X, PMCM_SYNC_START_MODE, 3);
    if (result != PMCM_RESULT_SUCCESS) {
        messageText = String::Format("PmcmSetSynchroConfig ERROR : 0x{0:X}", result);
        MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
        return;
    }

    // X軸の他軸の停止によるスタート条件を設定します(Y軸の停止確認)
    result = PmcmSetSynchroConfig(id, PMCM_AXIS_X, PMCM_SYNC_START, PMCM_AXIS_Y);
    if (result != PMCM_RESULT_SUCCESS) {
        messageText = String::Format("PmcmSetSynchroConfig ERROR : 0x{0:X}", result);
        MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
        return;
    }
} else if (startMode1->Checked) {
    // 内部同期信号によるスタート
    // X軸の出力パルスカウンタが500になったらY軸がスタートします

    // Y軸の起動条件を設定(内部同期信号によるスタート)
    result = PmcmSetSynchroConfig(id, PMCM_AXIS_Y, PMCM_SYNC_START_MODE, 2);
    if (result != PMCM_RESULT_SUCCESS) {
        messageText = String::Format("PmcmSetSynchroConfig ERROR : 0x{0:X}", result);
        MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
        return;
    }

    // Y軸の内部同期信号によるスタート条件の設定(X軸の内部同期信号でスタート)
    result = PmcmSetSynchroConfig(id, PMCM_AXIS_Y, PMCM_SYNC_SIG_START, 0);
    if (result != PMCM_RESULT_SUCCESS) {
        messageText = String::Format("PmcmSetSynchroConfig ERROR : 0x{0:X}", result);
        MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
        return;
    }

    // X軸の内部同期信号出力タイミングの設定(出力パルスコンパレータ条件成立時)
    result = PmcmSetSynchroOut(id, PMCM_AXIS_X, PMCM_SYNC_OUT_MODE, 1);
    if (result != PMCM_RESULT_SUCCESS) {
        messageText = String::Format("PmcmSetSynchroOut ERROR : 0x{0:X}", result);
        MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
        return;
    }

    // X軸のコンパレータの設定
    comp[0].wConfig = 1;
    comp[0].lCount = 500;
    result = PmcmSetComparatorConfig(id, PMCM_AXIS_X, PMCM_COMP_COUNTER, comp);
    if (result != PMCM_RESULT_SUCCESS) {
        messageText = String::Format("PmcmSetComparatorConfig ERROR : 0x{0:X}", result);
        MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
```

```
        return;
    }
}
```

動作パラメータ設定

```
int result;
unsigned short axis;
MOTIONPMCM motion[2];
String^ messageText;

axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// X軸
motion[0].wMoveMode = PMCM_PTP; // 動作モード
motion[0].wStartMode = PMCM_CONST; // 起動モード
motion[0].fSpeedRate = 1; // 速度倍率
motion[0].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
motion[0].fLowSpeed = 500; // 起動時速度
motion[0].fSpeed = 500; // 移動速度
motion[0].wAccTime = 0; // 加速時間
motion[0].wDecTime = 0; // 減速時間
motion[0].fSAccSpeed = 0; // 加速S字区間
motion[0].fSDecSpeed = 0; // 減速S字区間
motion[0].lSlowdown = -1; // スローダウンポイント
motion[0].lStep = 1000; // 移動パルス数,移動方向
motion[0].bAbsolutePtp = 0; // 絶対座標指定
// Y軸
motion[1].wMoveMode = PMCM_PTP; // 動作モード
motion[1].wStartMode = PMCM_CONST; // 起動モード
motion[1].fSpeedRate = 1; // 速度倍率
motion[1].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
motion[1].fLowSpeed = 1000; // 起動時速度
motion[1].fSpeed = 1000; // 移動速度
motion[1].wAccTime = 0; // 加速時間
motion[1].wDecTime = 0; // 減速時間
motion[1].fSAccSpeed = 0; // 加速S字区間
motion[1].fSDecSpeed = 0; // 減速S字区間
motion[1].lSlowdown = -1; // スローダウンポイント
motion[1].lStep = 1000; // 移動パルス数,移動方向
motion[1].bAbsolutePtp = 0; // 絶対座標指定
// 動作パラメータ設定
result = PmcmSetMotion(id, axis, motion);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetMotion ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

動作開始

```
int result;
unsigned short axis;
String^ messageText;

if (startMode0->Checked) {
    // 他軸の停止によるスタート
    // X軸はY軸の停止を確認して動作しますので、Y軸を先に動作させます
    // 同時に動作させてしまうと、その時点でY軸がまだ動作開始前で停止中のため、X軸が動作してしまいます
```

```
    // 動作開始
    result = PmcmStartMotion(id, PMCM_AXIS_Y);
    if (result != PMCM_RESULT_SUCCESS) {
        messageText = String::Format("PmcmStartMotion ERROR : 0x{0:X}", result);
        MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
        return;
    }
    // 動作開始
    result = PmcmStartMotion(id, PMCM_AXIS_X);
    if (result != PMCM_RESULT_SUCCESS) {
        messageText = String::Format("PmcmStartMotion ERROR : 0x{0:X}", result);
        MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
        return;
    }
} else if (startMode1->Checked) {
    // 内部同期信号によるスタート

    // 動作開始
    axis = PMCM_AXIS_X + PMCM_AXIS_Y;
    result = PmcmStartMotion(id, axis);
    if (result != PMCM_RESULT_SUCCESS) {
        messageText = String::Format("PmcmStartMotion ERROR : 0x{0:X}", result);
        MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
        return;
    }
}
```

動作停止

```
int result;
unsigned short axis;
unsigned short stopMode;
String^ messageText;

// 停止させる軸
axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// 停止モード
stopMode = PMCM_IMMEDIATE_STOP;

// 動作停止
result = PmcmStopMotion(id, axis, stopMode);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmStopMotion ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

クローズ

```
bool result;

result = PmcmClose(id);
```

Visual C#

オブジェクト

テキストボックス	Name
ID番号	Id

ラジオボタン	Name
他軸の停止によるスタート	StartMode0
内部同期信号によるスタート	StartMode1

変数

```
ushort id;
```

オープン

```
int result;

id = Convert.ToUInt16(Id.Text);
result = Pmcm.Open( id, "PMC-M2C-U" );
if( result == Pmcm.PMCM_RESULT_SUCCESS )
{
    MessageBox.Show( "オープン成功", "", MessageBoxButtons.OK, MessageBoxIcon.Information );
}
else
{
    MessageBox.Show( "オープン失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
}
```

各種設定

```
int result;
ushort axis;
Pmcm.COMPPMCM[] Comp = new Pmcm.COMPPMCM[2];
string outputString;

axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//result = Pmcm.SetSensorConfig( id, axis, Pmcm.PMCM_LOGIC, 0x3F );
//if( result != Pmcm.PMCM_RESULT_SUCCESS )
//{
```

```
//    outputString = String.Format( "PmcmSetSensorConfig ERROR : 0x{0:X}", result );
//    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
result = Pmcm.SetPulseConfig( id, axis, Pmcm.PMCM_PULSE_OUT, 7 );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetPulseConfig ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}

if( StartMode0.Checked )
{
    // 他軸の停止によるスタート
    // Y軸の停止によりX軸がスタートします

    // X軸の起動条件を設定します(他軸の停止によるスタート)
    result = Pmcm.SetSynchroConfig( id, Pmcm.PMCM_AXIS_X, Pmcm.PMCM_SYNC_START_MODE, 3 );
    if( result != Pmcm.PMCM_RESULT_SUCCESS )
    {
        outputString = String.Format( "PmcmSetSynchroConfig ERROR : 0x{0:X}", result );
        MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
        return;
    }

    // X軸の他軸の停止によるスタート条件を設定します(Y軸の停止確認);
    result = Pmcm.SetSynchroConfig( id, Pmcm.PMCM_AXIS_X, Pmcm.PMCM_SYNC_START,
Pmcm.PMCM_AXIS_Y );
    if( result != Pmcm.PMCM_RESULT_SUCCESS )
    {
        outputString = String.Format( "PmcmSetSynchroConfig ERROR : 0x{0:X}", result );
        MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
        return;
    }
}
else if( StartMode1.Checked )
{
    // 内部同期信号によるスタート
    // X軸の出力パルスカウンタが500になったらY軸がスタートします

    // Y軸の起動条件を設定(内部同期信号によるスタート)
    result = Pmcm.SetSynchroConfig( id, Pmcm.PMCM_AXIS_Y, Pmcm.PMCM_SYNC_START_MODE, 2);
    if( result != Pmcm.PMCM_RESULT_SUCCESS )
    {
        outputString = String.Format( "PmcmSetSynchroConfig ERROR : 0x{0:X}", result );
        MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
        return;
    }

    // Y軸の内部同期信号によるスタート条件の設定(X軸の内部同期信号でスタート)
    result = Pmcm.SetSynchroConfig( id, Pmcm.PMCM_AXIS_Y, Pmcm.PMCM_SYNC_SIG_START, 0);
    if( result != Pmcm.PMCM_RESULT_SUCCESS )
    {
        outputString = String.Format( "PmcmSetSynchroConfig ERROR : 0x{0:X}", result );
        MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
        return;
    }

    // X軸の内部同期信号出力タイミングの設定(出力パルスコンパレータ条件成立時)
    result = Pmcm.SetSynchroOut( id, Pmcm.PMCM_AXIS_X, Pmcm.PMCM_SYNC_OUT_MODE, 1);
```

```
    if( result != Pmcm.PMCM_RESULT_SUCCESS )
    {
        outputString = String.Format( "PmcmSetSynchroOut ERROR : 0x{0:X}", result );
        MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
        return;
    }

    // X軸のコンパレータの設定
    Comp[0].wConfig = 1;
    Comp[0].lCount = 500;
    result = Pmcm.SetComparatorConfig( id, Pmcm.PMCM_AXIS_X, Pmcm.PMCM_COMP_COUNTER, Comp );
    if( result != Pmcm.PMCM_RESULT_SUCCESS )
    {
        outputString = String.Format( "PmcmSetComparatorConfig ERROR : 0x{0:X}", result );
        MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
        return;
    }
}
```

動作パラメータ設定

```
int result;
ushort axis;
Pmcm.MOTIONPMCM[] motion = new Pmcm.MOTIONPMCM[2];
string outputString;

axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

// X軸
motion[0].wMoveMode = Pmcm.PMCM_PTP; // 動作モード
motion[0].wStartMode = Pmcm.PMCM_CONST; // 起動モード
motion[0].fSpeedRate = 1; // 速度倍率
motion[0].wAccDecMode = Pmcm.PMCM_ACC_LINEAR; // 加減速モード
motion[0].fLowSpeed = 500; // 起動時速度
motion[0].fSpeed = 500; // 移動速度
motion[0].wAccTime = 0; // 加速時間
motion[0].wDecTime = 0; // 減速時間
motion[0].fSAccSpeed = 0; // 加速S字区間
motion[0].fSDecSpeed = 0; // 減速S字区間
motion[0].lSlowdown = -1; // スローダウンポイント
motion[0].lStep = 1000; // 移動パルス数,移動方向
motion[0].bAbsolutePtp = 0; // 絶対座標指定
// Y軸
motion[1].wMoveMode = Pmcm.PMCM_PTP; // 動作モード
motion[1].wStartMode = Pmcm.PMCM_CONST; // 起動モード
motion[1].fSpeedRate = 1; // 速度倍率
motion[1].wAccDecMode = Pmcm.PMCM_ACC_LINEAR; // 加減速モード
motion[1].fLowSpeed = 1000; // 起動時速度
motion[1].fSpeed = 1000; // 移動速度
motion[1].wAccTime = 0; // 加速時間
motion[1].wDecTime = 0; // 減速時間
motion[1].fSAccSpeed = 0; // 加速S字区間
motion[1].fSDecSpeed = 0; // 減速S字区間
motion[1].lSlowdown = -1; // スローダウンポイント
motion[1].lStep = 1000; // 移動パルス数,移動方向
motion[1].bAbsolutePtp = 0; // 絶対座標指定
// 動作パラメータ設定
result = Pmcm.SetMotion( id, axis, motion );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetMotion ERROR : 0x{0:X}", result );
```

```
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

動作開始

```
int result;
ushort axis;
string outputString;

if( StartMode0.Checked )
{
    // 他軸の停止によるスタート
    // X軸はY軸の停止を確認して動作しますので、Y軸を先に動作させます
    // 同時に動作させてしまうと、その時点でY軸がまだ動作開始前で停止中のため、X軸が動作してしまいます

    // 動作開始
    result = Pmcm.StartMotion( id, Pmcm.PMCM_AXIS_Y );
    if( result != Pmcm.PMCM_RESULT_SUCCESS )
    {
        outputString = String.Format( "PmcmStartMotion ERROR : 0x{0:X}", result );
        MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
        return;
    }
    // 動作開始
    result = Pmcm.StartMotion( id, Pmcm.PMCM_AXIS_X );
    if( result != Pmcm.PMCM_RESULT_SUCCESS )
    {
        outputString = String.Format( "PmcmStartMotion ERROR : 0x{0:X}", result );
        MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
        return;
    }
}
else if( StartMode1.Checked )
{
    // 内部同期信号によるスタート

    // 動作開始
    axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;
    result = Pmcm.StartMotion( id, axis );
    if( result != Pmcm.PMCM_RESULT_SUCCESS )
    {
        outputString = String.Format( "PmcmStartMotion ERROR : 0x{0:X}", result );
        MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
        return;
    }
}
```

動作停止

```
int result;
ushort axis;
ushort stopMode;
string outputString;

// 停止させる軸
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

// 停止モード
```

```
stopMode = Pmcm.PMCM_IMMEDIATE_STOP;

// 動作停止
result = Pmcm.StopMotion( id, axis, stopMode );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmStopMotion ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

クローズ

```
bool result;

result = Pmcm.Close( id );
```

Visual Basic（.NET2002以降）

オブジェクト

テキストボックス	Name
ID番号	idTextBox

ラジオボタン	Name
他軸の停止によるスタート	startMode0
内部同期信号によるスタート	startMode1

変数

```
Dim id As Short
```

オープン

```
Dim result As Integer

id = Convert.ToInt16(idTextBox.Text)
result = PmcmOpen(id, "PMC-M2C-U")
If result = PMCM_RESULT_SUCCESS Then
    MessageBox.Show("オープン成功", "", MessageBoxButtons.OK, MessageBoxIcon.Information)
Else
    MessageBox.Show("オープン失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Error)
End If
```

各種設定

```
Dim result As Integer
Dim axis As Short
Dim comp(1) As COMPPMCM

axis = PMCM_AXIS_X + PMCM_AXIS_Y

'センサ設定
'オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
'その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
'result = PmcmSetSensorConfig(id, axis, PMCM_LOGIC, &H3F)
'If result <> PMCM_RESULT_SUCCESS Then
'    MessageBox.Show("PmcmSetSensorConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
'    Exit Sub
'End If
```

```
'パルス出力モード設定
'使用しているドライバに合致したパルス出力モードを選択してください
result = PmcmSetPulseConfig(id, axis, PMCM_PULSE_OUT, 7)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetPulseConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If

If startMode0.Checked = True Then
    '他軸の停止によるスタート
    'Y軸の停止によりX軸がスタートします

    'X軸の起動条件を設定します(他軸の停止によるスタート)
    result = PmcmSetSynchroConfig(id, PMCM_AXIS_X, PMCM_SYNC_START_MODE, 3)
    If result <> PMCM_RESULT_SUCCESS Then
        MessageBox.Show("PmcmSetSynchroConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        Exit Sub
    End If

    'X軸の他軸の停止によるスタート条件を設定します(Y軸の停止確認)
    result = PmcmSetSynchroConfig(id, PMCM_AXIS_X, PMCM_SYNC_START, PMCM_AXIS_Y)
    If result <> PMCM_RESULT_SUCCESS Then
        MessageBox.Show("PmcmSetSynchroConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        Exit Sub
    End If
ElseIf startMode1.Checked = True Then
    '内部同期信号によるスタート
    'X軸の出力パルスカウンタが500になったらY軸がスタートします

    'Y軸の起動条件を設定(内部同期信号によるスタート)
    result = PmcmSetSynchroConfig(id, PMCM_AXIS_Y, PMCM_SYNC_START_MODE, 2)
    If result <> PMCM_RESULT_SUCCESS Then
        MessageBox.Show("PmcmSetSynchroConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        Exit Sub
    End If

    'Y軸の内部同期信号によるスタート条件の設定(X軸の内部同期信号でスタート)
    result = PmcmSetSynchroConfig(id, PMCM_AXIS_Y, PMCM_SYNC_SIG_START, 0)
    If result <> PMCM_RESULT_SUCCESS Then
        MessageBox.Show("PmcmSetSynchroConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        Exit Sub
    End If

    'X軸の内部同期信号出力タイミングの設定(出力パルスコンパレータ条件成立時)
    result = PmcmSetSynchroOut(id, PMCM_AXIS_X, PMCM_SYNC_OUT_MODE, 1)
    If result <> PMCM_RESULT_SUCCESS Then
        MessageBox.Show("PmcmSetSynchroOut ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        Exit Sub
    End If

    'X軸のコンパレータの設定
    comp(0).wConfig = 1
    comp(0).lCount = 500
    result = PmcmSetComparatorConfig(id, PMCM_AXIS_X, PMCM_COMP_COUNTER, comp)
    If result <> PMCM_RESULT_SUCCESS Then
        MessageBox.Show("PmcmSetComparatorConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        Exit Sub
```

```
    End If
End If
```

動作パラメータ設定

```
Dim result As Integer
Dim axis As Short
Dim motion(1) As MOTIONPMCM

axis = PMCM_AXIS_X + PMCM_AXIS_Y

'X軸
motion(0).wMoveMode = PMCM_PTP '動作モード
motion(0).wStartMode = PMCM_CONST '起動モード
motion(0).fSpeedRate = 1 '速度倍率
motion(0).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
motion(0).fLowSpeed = 500 '起動時速度
motion(0).fSpeed = 500 '移動速度
motion(0).wAccTime = 0 '加速時間
motion(0).wDecTime = 0 '減速時間
motion(0).fSAccSpeed = 0 '加速S字区間
motion(0).fSDecSpeed = 0 '減速S字区間
motion(0).lSlowdown = -1 'スローダウンポイント
motion(0).lStep = 1000 '移動パルス数,移動方向
motion(0).bAbsolutePtp = 0 '絶対座標指定
'Y軸
motion(1).wMoveMode = PMCM_PTP '動作モード
motion(1).wStartMode = PMCM_CONST '起動モード
motion(1).fSpeedRate = 1 '速度倍率
motion(1).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
motion(1).fLowSpeed = 1000 '起動時速度
motion(1).fSpeed = 1000 '移動速度
motion(1).wAccTime = 0 '加速時間
motion(1).wDecTime = 0 '減速時間
motion(1).fSAccSpeed = 0 '加速S字区間
motion(1).fSDecSpeed = 0 '減速S字区間
motion(1).lSlowdown = -1 'スローダウンポイント
motion(1).lStep = 1000 '移動パルス数,移動方向
motion(1).bAbsolutePtp = 0 '絶対座標指定
'動作パラメータ設定
result = PmcmSetMotion(id, axis, motion)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetMotion ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

動作開始

```
Dim result As Integer
Dim axis As Short

If startMode0.Checked = True Then
    '他軸の停止によるスタート
    'X軸はY軸の停止を確認して動作しますので、Y軸を先に動作させます
    '同時に動作させてしまうと、その時点でY軸がまだ動作開始前で停止中のため、X軸が動作してしまいます

    'Y軸の動作開始
    result = PmcmStartMotion(id, PMCM_AXIS_Y)
```

```
    If result <> PMCM_RESULT_SUCCESS Then
        MessageBox.Show("PmcmStartMotion ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        Exit Sub
    End If

    'X軸の動作開始
    result = PmcmStartMotion(id, PMCM_AXIS_X)
    If result <> PMCM_RESULT_SUCCESS Then
        MessageBox.Show("PmcmStartMotion ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        Exit Sub
    End If
ElseIf startMode1.Checked = True Then
    '内部同期信号によるスタート

    '動作開始
    axis = PMCM_AXIS_X + PMCM_AXIS_Y
    result = PmcmStartMotion(id, axis)
    If result <> PMCM_RESULT_SUCCESS Then
        MessageBox.Show("PmcmStartMotion ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        Exit Sub
    End If
End If
```

動作停止

```
Dim result As Integer
Dim axis As Short
Dim stopMode As Short

'停止させる軸
axis = PMCM_AXIS_X + PMCM_AXIS_Y

'停止モード
stopMode = PMCM_IMMEDIATE_STOP

'動作停止
result = PmcmStopMotion(id, axis, stopMode)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmStopMotion ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

クローズ

```
Dim result As Boolean

result = PmcmClose(id)
```

Visual Basic 6.0/VBA

オブジェクト

テキストボックス	オブジェクト名
ID番号	txtID

ラジオボタン	オブジェクト名
他軸の停止によるスタート	optStartMode0
内部同期信号によるスタート	optStartMode1

変数

```
Dim m_intID As Integer
```

オープン

```
Dim lngResult As Long
Dim strModelName As String

m_intID = Val(txtID.Text)
strModelName = "PMC-M2C-U"
lngResult = PmcmOpen(m_intID, strModelName)
If lngResult = PMCM_RESULT_SUCCESS Then
    MsgBox "オープン成功", vbInformation
Else
    MsgBox "オープン失敗", vbCritical
End If
```

各種設定

```
Dim lngResult As Long
Dim intAxis As Integer
Dim Comp(1) As COMPPMCM

intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

'センサ設定
'オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
'その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
'lngResult = PmcmSetSensorConfig(m_intID, intAxis, PMCM_LOGIC, &H3F)
'If lngResult <> PMCM_RESULT_SUCCESS Then
'    MsgBox "PmcmSetSensorConfig ERROR : 0x" & Hex(lngResult), vbCritical
'    Exit Sub
'End If
```

```
'パルス出力モード設定
'使用しているドライバに合致したパルス出力モードを選択してください
lngResult = PmcmSetPulseConfig(m_intID, intAxis, PMCM_PULSE_OUT, 7)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetPulseConfig ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If

If optStartMode0.Value = True Then
    '他軸の停止によるスタート
    'Y軸の停止によりX軸がスタートします

    'X軸の起動条件を設定します(他軸の停止によるスタート)
    lngResult = PmcmSetSynchroConfig(m_intID, PMCM_AXIS_X, PMCM_SYNC_START_MODE, 3)
    If lngResult <> PMCM_RESULT_SUCCESS Then
        MsgBox "PmcmSetSynchroConfig ERROR : 0x" & Hex(lngResult), vbCritical
        Exit Sub
    End If

    'X軸の他軸の停止によるスタート条件を設定します(Y軸の停止確認)
    lngResult = PmcmSetSynchroConfig(m_intID, PMCM_AXIS_X, PMCM_SYNC_START, PMCM_AXIS_Y)
    If lngResult <> PMCM_RESULT_SUCCESS Then
        MsgBox "PmcmSetSynchroConfig ERROR : 0x" & Hex(lngResult), vbCritical
        Exit Sub
    End If
ElseIf optStartMode1.Value = True Then
    '内部同期信号によるスタート
    'X軸の出力パルスカウンタが500になったらY軸がスタートします

    'Y軸の起動条件を設定(内部同期信号によるスタート)
    lngResult = PmcmSetSynchroConfig(m_intID, PMCM_AXIS_Y, PMCM_SYNC_START_MODE, 2)
    If lngResult <> PMCM_RESULT_SUCCESS Then
        MsgBox "PmcmSetSynchroConfig ERROR : 0x" & Hex(lngResult), vbCritical
        Exit Sub
    End If

    'Y軸の内部同期信号によるスタート条件の設定(X軸の内部同期信号でスタート)
    lngResult = PmcmSetSynchroConfig(m_intID, PMCM_AXIS_Y, PMCM_SYNC_SIG_START, 0)
    If lngResult <> PMCM_RESULT_SUCCESS Then
        MsgBox "PmcmSetSynchroConfig ERROR : 0x" & Hex(lngResult), vbCritical
        Exit Sub
    End If

    'X軸の内部同期信号出力タイミングの設定(出力パルスコンパレータ条件成立時)
    lngResult = PmcmSetSynchroOut(m_intID, PMCM_AXIS_X, PMCM_SYNC_OUT_MODE, 1)
    If lngResult <> PMCM_RESULT_SUCCESS Then
        MsgBox "PmcmSetSynchroOut ERROR : 0x" & Hex(lngResult), vbCritical
        Exit Sub
    End If

    'X軸のコンパレータの設定
    Comp(0).wConfig = 1
    Comp(0).lCount = 500
    lngResult = PmcmSetComparatorConfig(m_intID, PMCM_AXIS_X, PMCM_COMP_COUNTER, Comp(0))
    If lngResult <> PMCM_RESULT_SUCCESS Then
        MsgBox "PmcmSetComparatorConfig ERROR : 0x" & Hex(lngResult), vbCritical
        Exit Sub
    End If
End If
```

動作パラメータ設定

```
Dim lngResult As Long
Dim intAxis As Integer
Dim Motion(1) As MOTIONPMCM

intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

'X軸
Motion(0).wMoveMode = PMCM_PTP '動作モード
Motion(0).wStartMode = PMCM_CONST '起動モード
Motion(0).fSpeedRate = 1 '速度倍率
Motion(0).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
Motion(0).fLowSpeed = 500 '起動時速度
Motion(0).fSpeed = 500 '移動速度
Motion(0).wAccTime = 0 '加速時間
Motion(0).wDecTime = 0 '減速時間
Motion(0).fSAccSpeed = 0 '加速S字区間
Motion(0).fSDecSpeed = 0 '減速S字区間
Motion(0).lSlowdown = -1 'スローダウンポイント
Motion(0).lStep = 1000 '移動パルス数,移動方向
Motion(0).bAbsolutePtp = 0 '絶対座標指定
'Y軸
Motion(1).wMoveMode = PMCM_PTP '動作モード
Motion(1).wStartMode = PMCM_CONST '起動モード
Motion(1).fSpeedRate = 1 '速度倍率
Motion(1).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
Motion(1).fLowSpeed = 1000 '起動時速度
Motion(1).fSpeed = 1000 '移動速度
Motion(1).wAccTime = 0 '加速時間
Motion(1).wDecTime = 0 '減速時間
Motion(1).fSAccSpeed = 0 '加速S字区間
Motion(1).fSDecSpeed = 0 '減速S字区間
Motion(1).lSlowdown = -1 'スローダウンポイント
Motion(1).lStep = 1000 '移動パルス数,移動方向
Motion(1).bAbsolutePtp = 0 '絶対座標指定
'動作パラメータ設定
lngResult = PmcmSetMotion(m_intID, intAxis, Motion(0))
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetMotion ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

動作開始

```
Dim lngResult As Long
Dim intAxis As Integer

If optStartMode0.Value = True Then
    '他軸の停止によるスタート
    'X軸はY軸の停止を確認して動作しますので、Y軸を先に動作させます
    '同時に動作させてしまうと、その時点でY軸がまだ動作開始前で停止中のため、X軸が動作してしまいます

    'Y軸の動作開始
    lngResult = PmcmStartMotion(m_intID, PMCM_AXIS_Y)
    If lngResult <> PMCM_RESULT_SUCCESS Then
        MsgBox "PmcmStartMotion ERROR : 0x" & Hex(lngResult), vbCritical
        Exit Sub
    End If

    'X軸の動作開始
    lngResult = PmcmStartMotion(m_intID, PMCM_AXIS_X)
    If lngResult <> PMCM_RESULT_SUCCESS Then
```

```
        MsgBox "PmcmStartMotion ERROR : 0x" & Hex(lngResult), vbCritical
        Exit Sub
    End If
ElseIf optStartMode1.Value = True Then
    '内部同期信号によるスタート

    '動作開始
    intAxis = PMCM_AXIS_X + PMCM_AXIS_Y
    lngResult = PmcmStartMotion(m_intID, intAxis)
    If lngResult <> PMCM_RESULT_SUCCESS Then
        MsgBox "PmcmStartMotion ERROR : 0x" & Hex(lngResult), vbCritical
        Exit Sub
    End If
End If
```

動作停止

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intStopMode As Integer

'停止させる軸
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

'停止モード
intStopMode = PMCM_IMMEDIATE_STOP

'動作停止
lngResult = PmcmStopMotion(m_intID, intAxis, intStopMode)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmStopMotion ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

クローズ

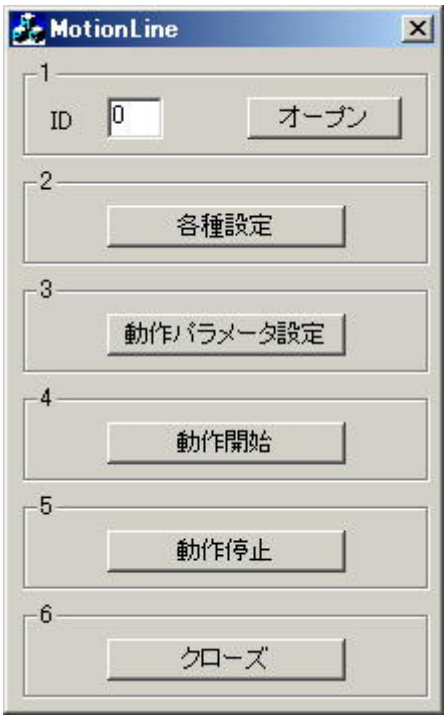
```
Dim blnResult As Boolean

'ボードクローズ
blnResult = PmcmClose(m_intID)
```

MotionLine

動作概要

直線補間動作をおこないます。
X軸をCW方向に500パルス、Y軸をCW方向に1000パルス動作させます。

画面



1. オープン

   ボードのオープンをおこないます。

2. 各種設定

   使用環境に応じて、センサ設定及びパルス出力設定をおこないます。

3. 動作パラメータ設定

   動作パラメータを設定します。

4. 動作開始

   モーター動作を開始します。

5. 動作停止

   モーター動作を停止します。
   既に停止している場合、実行する必要はありません。

6. クローズ

   ボードのクローズをおこないます。
   オープンをおこなった場合は必ず実行する必要があります。

備考

⚠ 非常停止信号（EMG）・アラーム信号（ALM）・エンドリミット信号（+EL、-EL）がONの場合、モーターは動作しません

サンプルソース

- Visual C++
- Visual C++/CLI
- Visual C#
- Visual Basic（.NET2002以降）
- Visual Basic 6.0/VBA

オブジェクト

テキストボックス	ID	メンバ変数
ID番号	IDC_ID	m_wID

オープン

```
if( !UpdateData( TRUE ) ) {
    return;
}

int nResult;

nResult = PmcmOpen( m_wID, "PMC-M2C-U" );
if( nResult == PMCM_RESULT_SUCCESS ) {
    AfxMessageBox( "オープン成功", MB_OK | MB_ICONINFORMATION );
} else {
    AfxMessageBox( "オープン失敗", MB_OK | MB_ICONSTOP );
}
```

各種設定

```
int nResult;
WORD wAxis;
char szResult[64];

wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//nResult = PmcmSetSensorConfig( m_wID, wAxis, PMCM_LOGIC, 0x3F );
//if( nResult != PMCM_RESULT_SUCCESS ) {
//    wsprintf( szResult, "PmcmSetSensorConfig ERROR : 0x%X", nResult );
//    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
nResult = PmcmSetPulseConfig( m_wID, wAxis, PMCM_PULSE_OUT, 7 );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetPulseConfig ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

動作パラメータ設定

```
int nResult;
WORD wAxis;
MOTIONLINEPMCM MotionLine;
char szResult[64];

wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

MotionLine.wStartMode = PMCM_CONST; // 起動モード
MotionLine.fSpeedRate = 1; // 速度倍率
MotionLine.wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
MotionLine.fLowSpeed = 1000; // 起動時速度
MotionLine.fSpeed = 1000; // 移動速度
MotionLine.wAccTime = 0; // 加速時間
MotionLine.wDecTime = 0; // 減速時間
MotionLine.fSAccSpeed = 0; // 加速S字区間
MotionLine.fSDecSpeed = 0; // 減速S字区間
MotionLine.lSlowdown = -1; // スローダウンポイント
MotionLine.lStep[0] = 500; // 移動パルス数,移動方向
MotionLine.lStep[1] = 1000; // 移動パルス数,移動方向
MotionLine.bAbsolute[0] = 0; // 絶対座標指定
MotionLine.bAbsolute[1] = 0; // 絶対座標指定
// 動作パラメータ設定
nResult = PmcmSetMotionLine( m_wID, wAxis, &MotionLine );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetMotionLine ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

動作開始

```
int nResult;
char szResult[64];

// 動作開始
nResult = PmcmStartMotionLine( m_wID );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmStartMotionLine ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

動作停止

```
int nResult;
WORD wAxis;
WORD wStopMode;
char szResult[64];

// 停止させる軸
wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

// 停止モード
wStopMode = PMCM_IMMEDIATE_STOP;

// 動作停止
nResult = PmcmStopMotion( m_wID, wAxis, wStopMode );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmStopMotion ERROR : 0x%X", nResult );
```

```
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

クローズ

```
BOOL bResult;

bResult = PmcmClose( m_wID );
```

オブジェクト

テキストボックス	Name
ID番号	idTextBox

変数

```
unsigned short id;
```

オープン

```
int result;

id = Convert::ToUInt16(idTextBox->Text);
result = PmcmOpen(id, "PMC-M2C-U");
if (result == PMCM_RESULT_SUCCESS) {
    MessageBox::Show("オープン成功", "", MessageBoxButtons::OK, MessageBoxIcon::Information);
} else {
    MessageBox::Show("オープン失敗", "", MessageBoxButtons::OK, MessageBoxIcon::Error);
}
```

各種設定

```
int result;
unsigned short axis;
String^ messageText;

axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//result = PmcmSetSensorConfig(id, axis, PMCM_LOGIC, 0x3F);
//if (result != PMCM_RESULT_SUCCESS) {
//    messageText = String::Format("PmcmSetSensorConfig ERROR : 0x{0:X}", result);
//    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
result = PmcmSetPulseConfig(id, axis, PMCM_PULSE_OUT, 7);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetPulseConfig ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

動作パラメータ設定

```
int result;
unsigned short axis;
MOTIONLINEPMCM motionLine;
String^ messageText;

axis = PMCM_AXIS_X + PMCM_AXIS_Y;

motionLine.wStartMode = PMCM_CONST; // 起動モード
motionLine.fSpeedRate = 1; // 速度倍率
motionLine.wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
motionLine.fLowSpeed = 1000; // 起動時速度
motionLine.fSpeed = 1000; // 移動速度
motionLine.wAccTime = 0; // 加速時間
motionLine.wDecTime = 0; // 減速時間
motionLine.fSAccSpeed = 0; // 加速S字区間
motionLine.fSDecSpeed = 0; // 減速S字区間
motionLine.lSlowdown = -1; // スローダウンポイント
motionLine.lStep[0] = 500; // 移動パルス数,移動方向
motionLine.lStep[1] = 1000; // 移動パルス数,移動方向
motionLine.bAbsolute[0] = 0; // 絶対座標指定
motionLine.bAbsolute[1] = 0; // 絶対座標指定
// 動作パラメータ設定
result = PmcmSetMotionLine(id, axis, &motionLine);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetMotionLine ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

動作開始

```
int result;
String^ messageText;

// 動作開始
result = PmcmStartMotionLine(id);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmStartMotionLine ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

動作停止

```
int result;
unsigned short axis;
unsigned short stopMode;
String^ messageText;

// 停止させる軸
axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// 停止モード
stopMode = PMCM_IMMEDIATE_STOP;

// 動作停止
```

```
result = PmcmStopMotion(id, axis, stopMode);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmStopMotion ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

クローズ

```
bool result;

result = PmcmClose(id);
```

オブジェクト

テキストボックス	Name
ID番号	Id

変数

```
ushort id;
```

オープン

```
int result;

id = Convert.ToUInt16(Id.Text);
result = Pmcm.Open( id, "PMC-M2C-U" );
if( result == Pmcm.PMCM_RESULT_SUCCESS )
{
    MessageBox.Show( "オープン成功", "", MessageBoxButtons.OK, MessageBoxIcon.Information );
}
else
{
    MessageBox.Show( "オープン失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
}
```

各種設定

```
int result;
ushort axis;
string outputString;

axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//result = Pmcm.SetSensorConfig( id, axis, Pmcm.PMCM_LOGIC, 0x3F );
//if( result != Pmcm.PMCM_RESULT_SUCCESS )
//{
//    outputString = String.Format( "PmcmSetSensorConfig ERROR : 0x{0:X}", result );
//    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
result = Pmcm.SetPulseConfig( id, axis, Pmcm.PMCM_PULSE_OUT, 7 );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetPulseConfig ERROR : 0x{0:X}", result );
```

```
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

動作パラメータ設定

```
int result;
ushort axis;
Pmcm.MOTIONLINEPMCM motionLine = new Pmcm.MOTIONLINEPMCM();
motionLine.Initialize();
string outputString;

axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

motionLine.wStartMode = Pmcm.PMCM_CONST; // 起動モード
motionLine.fSpeedRate = 1; // 速度倍率
motionLine.wAccDecMode = Pmcm.PMCM_ACC_LINEAR; // 加減速モード
motionLine.fLowSpeed = 1000; // 起動時速度
motionLine.fSpeed = 1000; // 移動速度
motionLine.wAccTime = 0; // 加速時間
motionLine.wDecTime = 0; // 減速時間
motionLine.fSAccSpeed = 0; // 加速S字区間
motionLine.fSDecSpeed = 0; // 減速S字区間
motionLine.lSlowdown = -1; // スローダウンポイント
motionLine.lStep[0] = 500; // 移動パルス数,移動方向
motionLine.lStep[1] = 1000; // 移動パルス数,移動方向
motionLine.bAbsolute[0] = 0; // 絶対座標指定
motionLine.bAbsolute[1] = 0; // 絶対座標指定
// 動作パラメータ設定
result = Pmcm.SetMotionLine( id, axis, ref motionLine );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetMotionLine ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

動作開始

```
int result;
string outputString;

// 動作開始
result = Pmcm.StartMotionLine( id );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmStartMotionLine ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

動作停止

```
int result;
ushort axis;
ushort stopMode;
string outputString;
```

```
// 停止させる軸
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

// 停止モード
stopMode = Pmcm.PMCM_IMMEDIATE_STOP;

// 動作停止
result = Pmcm.StopMotion( id, axis, stopMode );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmStopMotion ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

クローズ

```
bool result;

result = Pmcm.Close( id );
```

オブジェクト

テキストボックス	Name
ID番号	idTextBox

変数

```
Dim id As Short
```

オープン

```
Dim result As Integer

id = Convert.ToInt16(idTextBox.Text)
result = PmcmOpen(id, "PMC-M2C-U")
If result = PMCM_RESULT_SUCCESS Then
    MessageBox.Show("オープン成功", "", MessageBoxButtons.OK, MessageBoxIcon.Information)
Else
    MessageBox.Show("オープン失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Error)
End If
```

各種設定

```
Dim result As Integer
Dim axis As Short

axis = PMCM_AXIS_X + PMCM_AXIS_Y

'センサ設定
'オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
'その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
'result = PmcmSetSensorConfig(id, axis, PMCM_LOGIC, &H3F)
'If result <> PMCM_RESULT_SUCCESS Then
'    MessageBox.Show("PmcmSetSensorConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
'    Exit Sub
'End If

'パルス出力モード設定
'使用しているドライバに合致したパルス出力モードを選択してください
result = PmcmSetPulseConfig(id, axis, PMCM_PULSE_OUT, 7)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetPulseConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

動作パラメータ設定

```
Dim result As Integer
Dim axis As Short
Dim motionLine As MOTIONLINEPMCM
motionLine.Initialize()

axis = PMCM_AXIS_X + PMCM_AXIS_Y

motionLine.wStartMode = PMCM_CONST '起動モード
motionLine.fSpeedRate = 1 '速度倍率
motionLine.wAccDecMode = PMCM_ACC_LINEAR '加減速モード
motionLine.fLowSpeed = 1000 '起動時速度
motionLine.fSpeed = 1000 '移動速度
motionLine.wAccTime = 0 '加速時間
motionLine.wDecTime = 0 '減速時間
motionLine.fSAccSpeed = 0 '加速S字区間
motionLine.fSDecSpeed = 0 '減速S字区間
motionLine.lSlowdown = -1 'スローダウンポイント
motionLine.lStep(0) = 500 '移動パルス数,移動方向
motionLine.lStep(1) = 1000 '移動パルス数,移動方向
motionLine.bAbsolute(0) = 0 '絶対座標指定
motionLine.bAbsolute(1) = 0 '絶対座標指定
'動作パラメータ設定
result = PmcmSetMotionLine(id, axis, motionLine)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetMotionLine ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

動作開始

```
Dim result As Integer

'動作開始
result = PmcmStartMotionLine(id)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmStartMotionLine ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

動作停止

```
Dim result As Integer
Dim axis As Short
Dim stopMode As Short

'停止させる軸
axis = PMCM_AXIS_X + PMCM_AXIS_Y

'停止モード
stopMode = PMCM_IMMEDIATE_STOP

'動作停止
result = PmcmStopMotion(id, axis, stopMode)
If result <> PMCM_RESULT_SUCCESS Then
```

```
    MessageBox.Show("PmcmStopMotion ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

クローズ

```
Dim result As Boolean

result = PmcmClose(id)
```

Visual Basic 6.0/VBA

オブジェクト

テキストボックス	オブジェクト名
ID番号	txtID

変数

```
Dim m_intID As Integer
```

オープン

```
Dim lngResult As Long
Dim strModelName As String

m_intID = Val(txtID.Text)
strModelName = "PMC-M2C-U"
lngResult = PmcmOpen(m_intID, strModelName)
If lngResult = PMCM_RESULT_SUCCESS Then
    MsgBox "オープン成功", vbInformation
Else
    MsgBox "オープン失敗", vbCritical
End If
```

各種設定

```
Dim lngResult As Long
Dim intAxis As Integer

intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

'センサ設定
'オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
'その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
'lngResult = PmcmSetSensorConfig(m_intID, intAxis, PMCM_LOGIC, &H3F)
'If lngResult <> PMCM_RESULT_SUCCESS Then
'    MsgBox "PmcmSetSensorConfig ERROR : 0x" & Hex(lngResult), vbCritical
'    Exit Sub
'End If

'パルス出力モード設定
'使用しているドライバに合致したパルス出力モードを選択してください
lngResult = PmcmSetPulseConfig(m_intID, intAxis, PMCM_PULSE_OUT, 7)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetPulseConfig ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

動作パラメータ設定

```
Dim lngResult As Long
Dim intAxis As Integer
Dim MotionLine As MOTIONLINEPMCM

intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

MotionLine.wStartMode = PMCM_CONST '起動モード
MotionLine.fSpeedRate = 1 '速度倍率
MotionLine.wAccDecMode = PMCM_ACC_LINEAR '加減速モード
MotionLine.fLowSpeed = 1000 '起動時速度
MotionLine.fSpeed = 1000 '移動速度
MotionLine.wAccTime = 0 '加速時間
MotionLine.wDecTime = 0 '減速時間
MotionLine.fSAccSpeed = 0 '加速S字区間
MotionLine.fSDecSpeed = 0 '減速S字区間
MotionLine.lSlowdown = -1 'スローダウンポイント
MotionLine.lStep(0) = 500 '移動パルス数,移動方向
MotionLine.lStep(1) = 1000 '移動パルス数,移動方向
MotionLine.bAbsolute(0) = 0 '絶対座標指定
MotionLine.bAbsolute(1) = 0 '絶対座標指定
'動作パラメータ設定
lngResult = PmcmSetMotionLine(m_intID, intAxis, MotionLine)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetMotionLine ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

動作開始

```
Dim lngResult As Long

'動作開始
lngResult = PmcmStartMotionLine(m_intID)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmStartMotionLine ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

動作停止

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intStopMode As Integer

'停止させる軸
intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

'停止モード
intStopMode = PMCM_IMMEDIATE_STOP

'動作停止
lngResult = PmcmStopMotion(m_intID, intAxis, intStopMode)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmStopMotion ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

クローズ

```
Dim blnResult As Boolean

'ボードクローズ
blnResult = PmcmClose(m_intID)
```

MotionCp

動作概要

CP動作をおこないます。

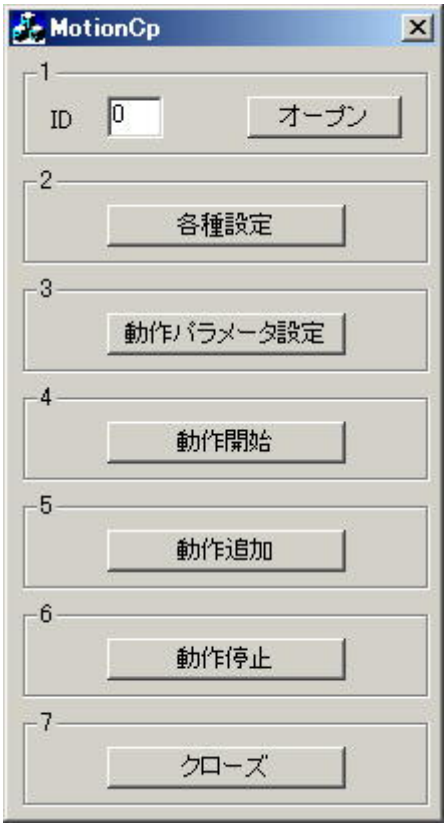
- [動作パラメータ設定]→[動作開始]での動作

  動作説明に記載されている動作をおこないます。

- [動作追加]での動作

  CW方向に1000パルス移動する動作を100回おこないます。

画面



1. オープン

   ボードのオープンをおこないます。

2. 各種設定

   使用環境に応じて、センサ設定及びパルス出力設定をおこないます。

3. 動作パラメータ設定

   動作パラメータを設定します。

4. 動作開始

   3で設定したCP動作を開始します。

5. 動作追加

設定できる最大CP動作数（96）を超えるCP動作をおこないます。
3→4の後に実行した場合は続けて実行されます。
3→4を実行しなくても動作します。

6. 動作停止

   モーター動作を停止します。 既に停止している場合、実行する必要はありません。

7. クローズ

   ボードのクローズをおこないます。
   オープンをおこなった場合は必ず実行する必要があります。

備考

> ⚠ 非常停止信号（EMG）・アラーム信号（ALM）・エンドリミット信号（+EL、-EL）がONの場合、モーターは動作しません

サンプルソース

- Visual C++
- Visual C++/CLI
- Visual C#
- Visual Basic（.NET2002以降）
- Visual Basic 6.0/VBA

オブジェクト

テキストボックス	ID	メンバ変数
ID番号	IDC_ID	m_wID

オープン

```
if( !UpdateData( TRUE ) ) {
    return;
}

int nResult;

nResult = PmcmOpen( m_wID, "PMC-M2C-U" );
if( nResult == PMCM_RESULT_SUCCESS ) {
    AfxMessageBox( "オープン成功", MB_OK | MB_ICONINFORMATION );
} else {
    AfxMessageBox( "オープン失敗", MB_OK | MB_ICONSTOP );
}
```

各種設定

```
int nResult;
WORD wAxis;
char szResult[64];

wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//nResult = PmcmSetSensorConfig( m_wID, wAxis, PMCM_LOGIC, 0x3F );
//if( nResult != PMCM_RESULT_SUCCESS ) {
//    wsprintf( szResult, "PmcmSetSensorConfig ERROR : 0x%X", nResult );
//    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
nResult = PmcmSetPulseConfig( m_wID, wAxis, PMCM_PULSE_OUT, 7 );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetPulseConfig ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

動作パラメータ設定

```
int nResult;
WORD wAxis;
MOTIONPMCM Motion[3];
WORD wEmpty[2];
WORD wCount;
char szResult[64];

wAxis = PMCM_AXIS_X;

// CP動作パラメータ数
wCount = 3;

// 1回目
Motion[0].wMoveMode = PMCM_PTP; // 動作モード
Motion[0].wStartMode = PMCM_ACC_DEC; // 起動モード
Motion[0].fSpeedRate = 1; // 速度倍率
Motion[0].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
Motion[0].fLowSpeed = 100; // 起動時速度
Motion[0].fSpeed = 500; // 移動速度
Motion[0].wAccTime = 500; // 加速時間
Motion[0].wDecTime = 500; // 減速時間
Motion[0].fSAccSpeed = 0; // 加速S字区間
Motion[0].fSDecSpeed = 0; // 減速S字区間
Motion[0].lSlowdown = 0; // スローダウンポイント
Motion[0].lStep = 1000; // 移動パルス数,移動方向
Motion[0].bAbsolutePtp = 0; // 絶対座標指定
// 2回目
Motion[1].wMoveMode = PMCM_PTP; // 動作モード
Motion[1].wStartMode = PMCM_ACC_DEC; // 起動モード
Motion[1].fSpeedRate = 1; // 速度倍率
Motion[1].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
Motion[1].fLowSpeed = 500; // 起動時速度
Motion[1].fSpeed = 1000; // 移動速度
Motion[1].wAccTime = 500; // 加速時間
Motion[1].wDecTime = 500; // 減速時間
Motion[1].fSAccSpeed = 0; // 加速S字区間
Motion[1].fSDecSpeed = 0; // 減速S字区間
Motion[1].lSlowdown = -1; // スローダウンポイント
Motion[1].lStep = 2000; // 移動パルス数,移動方向
Motion[1].bAbsolutePtp = 0; // 絶対座標指定
// 3回目
Motion[2].wMoveMode = PMCM_PTP; // 動作モード
Motion[2].wStartMode = PMCM_CONST_DEC; // 起動モード
Motion[2].fSpeedRate = 1; // 速度倍率
Motion[2].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
Motion[2].fLowSpeed = 100; // 起動時速度
Motion[2].fSpeed = 500; // 移動速度
Motion[2].wAccTime = 0; // 加速時間
Motion[2].wDecTime = 500; // 減速時間
Motion[2].fSAccSpeed = 0; // 加速S字区間
Motion[2].fSDecSpeed = 0; // 減速S字区間
Motion[2].lSlowdown = -1; // スローダウンポイント
Motion[2].lStep = 1500; // 移動パルス数,移動方向
Motion[2].bAbsolutePtp = 0; // 絶対座標指定

// CPの空き取得
nResult = PmcmGetEmptyCp( m_wID, wAxis, wEmpty );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmGetEmptyCp ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
// CPの空き確認
```

```
if( wEmpty[0] < wCount ) {
    AfxMessageBox( "CP空きなし", MB_OK | MB_ICONSTOP );
    return;
}

// 動作パラメータ設定
nResult = PmcmSetMotionCp( m_wID, wAxis, Motion, wCount );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetMotionCp ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

動作開始

```
int nResult;
WORD wAxis;
char szResult[64];

// 動作させる軸
wAxis = PMCM_AXIS_X;

// 動作開始
nResult = PmcmStartMotionCp( m_wID, wAxis );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmStartMotionCp ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

動作追加

```
// CP動作は96動作までしか設定できません
// それ以上の動作を設定したい場合は、動作中にCPの空きを確認し、空き数分の動作を開始します
int nResult;
WORD wAxis;
MOTIONPMCM Motion[100];
WORD wEmpty[2];
WORD wCount;
WORD wRemainCount;
WORD wSetCount;
int i;
char szResult[64];

wAxis = PMCM_AXIS_X;

// CP動作パラメータ数
wCount = 100;

for( i = 0; i < wCount; i++ ) {
    Motion[i].wMoveMode = PMCM_PTP; // 動作モード
    Motion[i].wStartMode = PMCM_CONST; // 起動モード
    Motion[i].fSpeedRate = 1; // 速度倍率
    Motion[i].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
    Motion[i].fLowSpeed = 1000; // 起動時速度
    Motion[i].fSpeed = 1000; // 移動速度
    Motion[i].wAccTime = 0; // 加速時間
    Motion[i].wDecTime = 0; // 減速時間
    Motion[i].fSAccSpeed = 0; // 加速S字区間
```

```
    Motion[i].fSDecSpeed = 0; // 減速S字区間
    Motion[i].lSlowdown = -1; // スローダウンポイント
    Motion[i].lStep = 1000; // 移動パルス数,移動方向
    Motion[i].bAbsolutePtp = 0; // 絶対座標指定
}

// 残り動作数
wRemainCount= wCount;

while( wRemainCount > 0 ) {
    // CPの空き取得
    nResult = PmcmGetEmptyCp( m_wID, wAxis, wEmpty );
    if( nResult != PMCM_RESULT_SUCCESS ) {
        wsprintf( szResult, "PmcmGetEmptyCp ERROR : 0x%X", nResult );
        AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
        return;
    }
    // CPの空きを確認し、設定動作数を決定する
    if( wEmpty[0] == 0 ) { // CP空きなし
        wSetCount = 0;
    } else if( wEmpty[0] < wRemainCount ) { // CP空き < 残り動作数
        wSetCount = wEmpty[0]; // CP空き分だけ設定する
    } else if( wEmpty[0] >= wRemainCount ) { // CP空き >= 残り動作数
        wSetCount = wRemainCount; // 残り動作数を設定する
    }

    if( wSetCount > 0 ) {
        // 動作パラメータ設定
        nResult = PmcmSetMotionCp( m_wID, wAxis, &Motion[wCount - wRemainCount], wSetCount
);
        if( nResult != PMCM_RESULT_SUCCESS ) {
            wsprintf( szResult, "PmcmSetMotionCp ERROR : 0x%X", nResult );
            AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
            return;
        }
        // 動作開始
        nResult = PmcmStartMotionCp( m_wID, wAxis );
        if( nResult != PMCM_RESULT_SUCCESS ) {
            wsprintf( szResult, "PmcmStartMotionCp ERROR : 0x%X", nResult );
            AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
            return;
        }
    }
    // 残り動作数更新
    wRemainCount -= wSetCount;
}
```

動作停止

```
int nResult;
WORD wAxis;
WORD wStopMode;
char szResult[64];

// 停止させる軸
wAxis = PMCM_AXIS_X;

// 停止モード
wStopMode = PMCM_IMMEDIATE_STOP;

// 動作停止
```

```
nResult = PmcmStopMotion( m_wID, wAxis, wStopMode );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmStopMotion ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

クローズ

```
BOOL bResult;

bResult = PmcmClose( m_wID );
```

オブジェクト

テキストボックス	Name
ID番号	idTextBox

変数

```
unsigned short id;
```

オープン

```
int result;

id = Convert::ToUInt16(idTextBox->Text);
result = PmcmOpen(id, "PMC-M2C-U");
if (result == PMCM_RESULT_SUCCESS) {
    MessageBox::Show("オープン成功", "", MessageBoxButtons::OK, MessageBoxIcon::Information);
} else {
    MessageBox::Show("オープン失敗", "", MessageBoxButtons::OK, MessageBoxIcon::Error);
}
```

各種設定

```
int result;
unsigned short axis;
String^ messageText;

axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//result = PmcmSetSensorConfig(id, axis, PMCM_LOGIC, 0x3F);
//if (result != PMCM_RESULT_SUCCESS) {
//    messageText = String::Format("PmcmSetSensorConfig ERROR : 0x{0:X}", result);
//    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
result = PmcmSetPulseConfig(id, axis, PMCM_PULSE_OUT, 7);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetPulseConfig ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

動作パラメータ設定

```
int result;
unsigned short axis;
MOTIONPMCM motion[3];
unsigned short empty[2];
unsigned short count;
String^ messageText;

axis = PMCM_AXIS_X;

// CP動作パラメータ数
count = 3;

// 1回目
motion[0].wMoveMode = PMCM_PTP; // 動作モード
motion[0].wStartMode = PMCM_ACC_DEC; // 起動モード
motion[0].fSpeedRate = 1; // 速度倍率
motion[0].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
motion[0].fLowSpeed = 100; // 起動時速度
motion[0].fSpeed = 500; // 移動速度
motion[0].wAccTime = 500; // 加速時間
motion[0].wDecTime = 500; // 減速時間
motion[0].fSAccSpeed = 0; // 加速S字区間
motion[0].fSDecSpeed = 0; // 減速S字区間
motion[0].lSlowdown = 0; // スローダウンポイント
motion[0].lStep = 1000; // 移動パルス数,移動方向
motion[0].bAbsolutePtp = 0; // 絶対座標指定
// 2回目
motion[1].wMoveMode = PMCM_PTP; // 動作モード
motion[1].wStartMode = PMCM_ACC_DEC; // 起動モード
motion[1].fSpeedRate = 1; // 速度倍率
motion[1].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
motion[1].fLowSpeed = 500; // 起動時速度
motion[1].fSpeed = 1000; // 移動速度
motion[1].wAccTime = 500; // 加速時間
motion[1].wDecTime = 500; // 減速時間
motion[1].fSAccSpeed = 0; // 加速S字区間
motion[1].fSDecSpeed = 0; // 減速S字区間
motion[1].lSlowdown = -1; // スローダウンポイント
motion[1].lStep = 2000; // 移動パルス数,移動方向
motion[1].bAbsolutePtp = 0; // 絶対座標指定
// 3回目
motion[2].wMoveMode = PMCM_PTP; // 動作モード
motion[2].wStartMode = PMCM_CONST_DEC; // 起動モード
motion[2].fSpeedRate = 1; // 速度倍率
motion[2].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
motion[2].fLowSpeed = 100; // 起動時速度
motion[2].fSpeed = 500; // 移動速度
motion[2].wAccTime = 0; // 加速時間
motion[2].wDecTime = 500; // 減速時間
motion[2].fSAccSpeed = 0; // 加速S字区間
motion[2].fSDecSpeed = 0; // 減速S字区間
motion[2].lSlowdown = -1; // スローダウンポイント
motion[2].lStep = 1500; // 移動パルス数,移動方向
motion[2].bAbsolutePtp = 0; // 絶対座標指定

// CPの空き取得
result = PmcmGetEmptyCp(id, axis, empty);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmGetEmptyCp ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
```

```
}
// CPの空き確認
if (empty[0] < count) {
    MessageBox::Show("CP空きなし", "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}

// 動作パラメータ設定
result = PmcmSetMotionCp(id, axis, motion, count);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetMotionCp ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

動作開始

```
int result;
unsigned short axis;
String^ messageText;

// 動作させる軸
axis = PMCM_AXIS_X;

// 動作開始
result = PmcmStartMotionCp(id, axis);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmStartMotionCp ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

動作追加

```
// CP動作は96動作までしか設定できません
// それ以上の動作を設定したい場合は、動作中にCPの空きを確認し、空き数分の動作を開始します
int result;
unsigned short axis;
MOTIONPMCM motion[100];
unsigned short empty[2];
unsigned short count;
unsigned short remainCount;
unsigned short setCount;
int i;
String^ messageText;

axis = PMCM_AXIS_X;

// CP動作パラメータ数
count = 100;

for (i = 0; i < count; i++) {
    motion[i].wMoveMode = PMCM_PTP; // 動作モード
    motion[i].wStartMode = PMCM_CONST; // 起動モード
    motion[i].fSpeedRate = 1; // 速度倍率
    motion[i].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
    motion[i].fLowSpeed = 1000; // 起動時速度
    motion[i].fSpeed = 1000; // 移動速度
    motion[i].wAccTime = 0; // 加速時間
```

```
    motion[i].wDecTime = 0; // 減速時間
    motion[i].fSAccSpeed = 0; // 加速S字区間
    motion[i].fSDecSpeed = 0; // 減速S字区間
    motion[i].lSlowdown = -1; // スローダウンポイント
    motion[i].lStep = 1000; // 移動パルス数,移動方向
    motion[i].bAbsolutePtp = 0; // 絶対座標指定
}

// 残り動作数
remainCount= count;

while (remainCount > 0) {
    // CPの空き取得
    result = PmcmGetEmptyCp(id, axis, empty);
    if (result != PMCM_RESULT_SUCCESS) {
        messageText = String::Format("PmcmGetEmptyCp ERROR : 0x{0:X}", result);
        MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
        return;
    }
    // CPの空きを確認し、設定動作数を決定する
    if (empty[0] == 0) { // CP空きなし
        setCount = 0;
    } else if (empty[0] < remainCount) { // CP空き < 残り動作数
        setCount = empty[0]; // CP空き分だけ設定する
    } else if (empty[0] >= remainCount) { // CP空き >= 残り動作数
        setCount = remainCount; // 残り動作数を設定する
    }

    if (setCount > 0) {
        // 動作パラメータ設定
        result = PmcmSetMotionCp(id, axis, &motion[count - remainCount], setCount);
        if (result != PMCM_RESULT_SUCCESS) {
            messageText = String::Format("PmcmSetMotionCp ERROR : 0x{0:X}", result);
            MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
            return;
        }
        // 動作開始
        result = PmcmStartMotionCp(id, axis);
        if (result != PMCM_RESULT_SUCCESS) {
            messageText = String::Format("PmcmStartMotionCp ERROR : 0x{0:X}", result);
            MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
            return;
        }
    }
    // 残り動作数更新
    remainCount -= setCount;
}
```

動作停止

```
int result;
unsigned short axis;
unsigned short stopMode;
String^ messageText;

// 停止させる軸
axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// 停止モード
stopMode = PMCM_IMMEDIATE_STOP;
```

```
// 動作停止
result = PmcmStopMotion(id, axis, stopMode);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmStopMotion ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

クローズ

```
bool result;

result = PmcmClose(id);
```

オブジェクト

テキストボックス	Name
ID番号	Id

変数

```
ushort id;
```

オープン

```
int result;

id = Convert.ToUInt16(Id.Text);
result = Pmcm.Open( id, "PMC-M2C-U" );
if( result == Pmcm.PMCM_RESULT_SUCCESS )
{
    MessageBox.Show( "オープン成功", "", MessageBoxButtons.OK, MessageBoxIcon.Information );
}
else
{
    MessageBox.Show( "オープン失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
}
```

各種設定

```
int result;
ushort axis;
string outputString;

axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//result = Pmcm.SetSensorConfig( id, axis, Pmcm.PMCM_LOGIC, 0x3F );
//if( result != Pmcm.PMCM_RESULT_SUCCESS )
//{
//    outputString = String.Format( "PmcmSetSensorConfig ERROR : 0x{0:X}", result );
//    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
result = Pmcm.SetPulseConfig( id, axis, Pmcm.PMCM_PULSE_OUT, 7 );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetPulseConfig ERROR : 0x{0:X}", result );
```

```
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

動作パラメータ設定

```
int result;
ushort axis;
Pmcm.MOTIONPMCM[] motion = new Pmcm.MOTIONPMCM[3];
ushort[] empty = new ushort[2];
ushort count;
string outputString;

axis = Pmcm.PMCM_AXIS_X;

// CP動作パラメータ数
count = 3;

// 1回目
motion[0].wMoveMode = Pmcm.PMCM_PTP; // 動作モード
motion[0].wStartMode = Pmcm.PMCM_ACC_DEC; // 起動モード
motion[0].fSpeedRate = 1; // 速度倍率
motion[0].wAccDecMode = Pmcm.PMCM_ACC_LINEAR; // 加減速モード
motion[0].fLowSpeed = 100; // 起動時速度
motion[0].fSpeed = 500; // 移動速度
motion[0].wAccTime = 500; // 加速時間
motion[0].wDecTime = 500; // 減速時間
motion[0].fSAccSpeed = 0; // 加速S字区間
motion[0].fSDecSpeed = 0; // 減速S字区間
motion[0].lSlowdown = 0; // スローダウンポイント
motion[0].lStep = 1000; // 移動パルス数,移動方向
motion[0].bAbsolutePtp = 0; // 絶対座標指定
// 2回目
motion[1].wMoveMode = Pmcm.PMCM_PTP; // 動作モード
motion[1].wStartMode = Pmcm.PMCM_ACC_DEC; // 起動モード
motion[1].fSpeedRate = 1; // 速度倍率
motion[1].wAccDecMode = Pmcm.PMCM_ACC_LINEAR; // 加減速モード
motion[1].fLowSpeed = 500; // 起動時速度
motion[1].fSpeed = 1000; // 移動速度
motion[1].wAccTime = 500; // 加速時間
motion[1].wDecTime = 500; // 減速時間
motion[1].fSAccSpeed = 0; // 加速S字区間
motion[1].fSDecSpeed = 0; // 減速S字区間
motion[1].lSlowdown = -1; // スローダウンポイント
motion[1].lStep = 2000; // 移動パルス数,移動方向
motion[1].bAbsolutePtp = 0; // 絶対座標指定
// 3回目
motion[2].wMoveMode = Pmcm.PMCM_PTP; // 動作モード
motion[2].wStartMode = Pmcm.PMCM_CONST_DEC; // 起動モード
motion[2].fSpeedRate = 1; // 速度倍率
motion[2].wAccDecMode = Pmcm.PMCM_ACC_LINEAR; // 加減速モード
motion[2].fLowSpeed = 100; // 起動時速度
motion[2].fSpeed = 500; // 移動速度
motion[2].wAccTime = 0; // 加速時間
motion[2].wDecTime = 500; // 減速時間
motion[2].fSAccSpeed = 0; // 加速S字区間
motion[2].fSDecSpeed = 0; // 減速S字区間
motion[2].lSlowdown = -1; // スローダウンポイント
motion[2].lStep = 1500; // 移動パルス数,移動方向
motion[2].bAbsolutePtp = 0; // 絶対座標指定
```

```
// CPの空き取得
result = Pmcm.GetEmptyCp( id, axis, empty );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmGetEmptyCp ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
// CPの空き確認
if( empty[0] < count )
{
    MessageBox.Show( "CP空きなし", "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}

// 動作パラメータ設定
result = Pmcm.SetMotionCp( id, axis, motion, count );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetMotionCp ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

動作開始

```
int result;
ushort axis;
string outputString;

// 動作させる軸
axis = Pmcm.PMCM_AXIS_X;

// 動作開始
result = Pmcm.StartMotionCp( id, axis );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmStartMotionCp ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

動作追加

```
// CP動作は96動作までしか設定できません
// それ以上の動作を設定したい場合は、動作中にCPの空きを確認し、空き数分の動作を開始します
int result;
ushort axis;
Pmcm.MOTIONPMCM[] motion = new Pmcm.MOTIONPMCM[100];
Pmcm.MOTIONPMCM[] setMotion;
ushort[] empty = new ushort[2];
ushort count;
ushort remainCount;
ushort setCount = 0;
int i;
string outputString;

axis = Pmcm.PMCM_AXIS_X;
```

```
// CP動作パラメータ数
count = 100;

for( i = 0; i < count; i++ )
{
    motion[i].wMoveMode = Pmcm.PMCM_PTP; // 動作モード
    motion[i].wStartMode = Pmcm.PMCM_CONST; // 起動モード
    motion[i].fSpeedRate = 1; // 速度倍率
    motion[i].wAccDecMode = Pmcm.PMCM_ACC_LINEAR; // 加減速モード
    motion[i].fLowSpeed = 1000; // 起動時速度
    motion[i].fSpeed = 1000; // 移動速度
    motion[i].wAccTime = 0; // 加速時間
    motion[i].wDecTime = 0; // 減速時間
    motion[i].fSAccSpeed = 0; // 加速S字区間
    motion[i].fSDecSpeed = 0; // 減速S字区間
    motion[i].lSlowdown = -1; // スローダウンポイント
    motion[i].lStep = 1000; // 移動パルス数,移動方向
    motion[i].bAbsolutePtp = 0; // 絶対座標指定
}

// 残り動作数
remainCount= count;

while( remainCount > 0 )
{
    // CPの空き取得
    result = Pmcm.GetEmptyCp( id, axis, empty );
    if( result != Pmcm.PMCM_RESULT_SUCCESS )
    {
        outputString = String.Format( "PmcmGetEmptyCp ERROR : 0x{0:X}", result );
        MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
        return;
    }
    // CPの空きを確認し、設定動作数を決定する
    if( empty[0] == 0 )
    { // CP空きなし
        setCount = 0;
    }
    else if( empty[0] < remainCount )
    { // CP空き < 残り動作数
        setCount = empty[0]; // CP空き分だけ設定する
    }
    else if( empty[0] >= remainCount )
    { // CP空き >= 残り動作数
        setCount = remainCount; // 残り動作数を設定する
    }

    if( setCount > 0 )
    {
        // 動作パラメータ
        setMotion = new Pmcm.MOTIONPMCM[setCount];
        Array.Copy( motion, count - remainCount, setMotion, 0, setCount );

        // 動作パラメータ設定
        result = Pmcm.SetMotionCp( id, axis, setMotion, setCount );
        if( result != Pmcm.PMCM_RESULT_SUCCESS )
        {
            outputString = String.Format( "PmcmSetMotionCp ERROR : 0x{0:X}", result );
            MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
            return;
        }
        // 動作開始
        result = Pmcm.StartMotionCp( id, axis );
        if( result != Pmcm.PMCM_RESULT_SUCCESS )
```

```
        {
            outputString = String.Format( "PmcmStartMotionCp ERROR : 0x{0:X}", result );
            MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
            return;
        }
    }
    // 残り動作数更新
    remainCount -= setCount;
}
```

動作停止

```
int result;
ushort axis;
ushort stopMode;
string outputString;

// 停止させる軸
axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

// 停止モード
stopMode = Pmcm.PMCM_IMMEDIATE_STOP;

// 動作停止
result = Pmcm.StopMotion( id, axis, stopMode );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmStopMotion ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

クローズ

```
bool result;

result = Pmcm.Close( id );
```

Visual Basic（.NET2002以降）

オブジェクト

テキストボックス	Name
ID番号	idTextBox

変数

```
Dim id As Short
```

オープン

```
Dim result As Integer

id = Convert.ToInt16(idTextBox.Text)
result = PmcmOpen(id, "PMC-M2C-U")
If result = PMCM_RESULT_SUCCESS Then
    MessageBox.Show("オープン成功", "", MessageBoxButtons.OK, MessageBoxIcon.Information)
Else
    MessageBox.Show("オープン失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Error)
End If
```

各種設定

```
Dim result As Integer
Dim axis As Short

axis = PMCM_AXIS_X + PMCM_AXIS_Y

'センサ設定
'オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
'その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
'result = PmcmSetSensorConfig(id, axis, PMCM_LOGIC, &H3F)
'If result <> PMCM_RESULT_SUCCESS Then
'    MessageBox.Show("PmcmSetSensorConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
'    Exit Sub
'End If

'パルス出力モード設定
'使用しているドライバに合致したパルス出力モードを選択してください
result = PmcmSetPulseConfig(id, axis, PMCM_PULSE_OUT, 7)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetPulseConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

動作パラメータ設定

```
Dim result As Integer
Dim axis As Short
Dim motion(2) As MOTIONPMCM
Dim empty(1) As Short
Dim count As Short

axis = PMCM_AXIS_X

'CP動作パラメータ数
count = 3

'1回目
motion(0).wMoveMode = PMCM_PTP '動作モード
motion(0).wStartMode = PMCM_ACC_DEC '起動モード
motion(0).fSpeedRate = 1 '速度倍率
motion(0).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
motion(0).fLowSpeed = 100 '起動時速度
motion(0).fSpeed = 500 '移動速度
motion(0).wAccTime = 500 '加速時間
motion(0).wDecTime = 500 '減速時間
motion(0).fSAccSpeed = 0 '加速S字区間
motion(0).fSDecSpeed = 0 '減速S字区間
motion(0).lSlowdown = 0 'スローダウンポイント
motion(0).lStep = 1000 '移動パルス数,移動方向
motion(0).bAbsolutePtp = 0 '絶対座標指定
'2回目
motion(1).wMoveMode = PMCM_PTP '動作モード
motion(1).wStartMode = PMCM_ACC_DEC '起動モード
motion(1).fSpeedRate = 1 '速度倍率
motion(1).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
motion(1).fLowSpeed = 500 '起動時速度
motion(1).fSpeed = 1000 '移動速度
motion(1).wAccTime = 500 '加速時間
motion(1).wDecTime = 500 '減速時間
motion(1).fSAccSpeed = 0 '加速S字区間
motion(1).fSDecSpeed = 0 '減速S字区間
motion(1).lSlowdown = -1 'スローダウンポイント
motion(1).lStep = 2000 '移動パルス数,移動方向
motion(1).bAbsolutePtp = 0 '絶対座標指定
'3回目
motion(2).wMoveMode = PMCM_PTP '動作モード
motion(2).wStartMode = PMCM_CONST_DEC '起動モード
motion(2).fSpeedRate = 1 '速度倍率
motion(2).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
motion(2).fLowSpeed = 100 '起動時速度
motion(2).fSpeed = 500 '移動速度
motion(2).wAccTime = 0 '加速時間
motion(2).wDecTime = 500 '減速時間
motion(2).fSAccSpeed = 0 '加速S字区間
motion(2).fSDecSpeed = 0 '減速S字区間
motion(2).lSlowdown = -1 'スローダウンポイント
motion(2).lStep = 1500 '移動パルス数,移動方向
motion(2).bAbsolutePtp = 0 '絶対座標指定

'CPの空き取得
result = PmcmGetEmptyCp(id, axis, empty)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmGetEmptyCp ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

```
'CPの空き確認
If empty(0) < count Then
    MessageBox.Show("CP空きなし", "", MessageBoxButtons.OK, MessageBoxIcon.Stop)
    Exit Sub
End If

'動作パラメータ設定
result = PmcmSetMotionCp(id, axis, motion, count)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetMotionCp ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

動作開始

```
Dim result As Integer
Dim axis As Short

'動作させる軸
axis = PMCM_AXIS_X

'動作開始
result = PmcmStartMotionCp(id, axis)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmStartMotionCp ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

動作追加

```
'CP動作は96動作までしか設定できません
'それ以上の動作を設定したい場合は、動作中にCPの空きを確認し、空き数分の動作を開始します
Dim result As Integer
Dim axis As Short
Dim motion(99) As MOTIONPMCM
Dim setMotion() As MOTIONPMCM
Dim empty(1) As Short
Dim count As Short
Dim remainCount As Short
Dim setCount As Short
Dim i As Integer

axis = PMCM_AXIS_X

'CP動作パラメータ数
count = 100

For i = 0 To count - 1
    motion(i).wMoveMode = PMCM_PTP '動作モード
    motion(i).wStartMode = PMCM_CONST '起動モード
    motion(i).fSpeedRate = 1 '速度倍率
    motion(i).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
    motion(i).fLowSpeed = 1000 '起動時速度
    motion(i).fSpeed = 1000 '移動速度
    motion(i).wAccTime = 0 '加速時間
    motion(i).wDecTime = 0 '減速時間
    motion(i).fSAccSpeed = 0 '加速S字区間
```

```
    motion(i).fSDecSpeed = 0 '減速S字区間
    motion(i).lSlowdown = -1 'スローダウンポイント
    motion(i).lStep = 1000 '移動パルス数,移動方向
    motion(i).bAbsolutePtp = 0 '絶対座標指定
Next

'残り動作数
remainCount = count

Do While remainCount > 0
    'CPの空き取得
    result = PmcmGetEmptyCp(id, axis, empty)
    If result <> PMCM_RESULT_SUCCESS Then
        MessageBox.Show("PmcmGetEmptyCp ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        Exit Sub
    End If
    'CPの空きを確認し、設定動作数を決定する
    If empty(0) = 0 Then 'CP空きなし
        setCount = 0
    ElseIf empty(0) < remainCount Then 'CP空き < 残り動作数
        setCount = empty(0) 'CP空き分だけ設定する
    ElseIf empty(0) >= remainCount Then 'CP空き >= 残り動作数
        setCount = remainCount '残り動作数を設定する
    End If

    If setCount > 0 Then
        '動作パラメータ
        ReDim setMotion(setCount - 1)
        Array.Copy(motion, count - remainCount, setMotion, 0, setCount)

        '動作パラメータ設定
        result = PmcmSetMotionCp(id, axis, setMotion, setCount)
        If result <> PMCM_RESULT_SUCCESS Then
            MessageBox.Show("PmcmSetMotionCp ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
            Exit Sub
        End If
        '動作開始
        result = PmcmStartMotionCp(id, axis)
        If result <> PMCM_RESULT_SUCCESS Then
            MessageBox.Show("PmcmStartMotionCp ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
            Exit Sub
        End If
    End If
    '残り動作数更新
    remainCount = remainCount - setCount
Loop
```

動作停止

```
Dim result As Integer
Dim axis As Short
Dim stopMode As Short

'停止させる軸
axis = PMCM_AXIS_X

'停止モード
stopMode = PMCM_IMMEDIATE_STOP
```

```
'動作停止
result = PmcmStopMotion(id, axis, stopMode)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmStopMotion ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

クローズ

```
Dim result As Boolean

result = PmcmClose(id)
```

Visual Basic 6.0/VBA

オブジェクト

テキストボックス	オブジェクト名
ID番号	txtID

変数

```
Dim m_intID As Integer
```

オープン

```
Dim lngResult As Long
Dim strModelName As String

m_intID = Val(txtID.Text)
strModelName = "PMC-M2C-U"
lngResult = PmcmOpen(m_intID, strModelName)
If lngResult = PMCM_RESULT_SUCCESS Then
    MsgBox "オープン成功", vbInformation
Else
    MsgBox "オープン失敗", vbCritical
End If
```

各種設定

```
Dim lngResult As Long
Dim intAxis As Integer

intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

'センサ設定
'オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
'その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
'lngResult = PmcmSetSensorConfig(m_intID, intAxis, PMCM_LOGIC, &H3F)
'If lngResult <> PMCM_RESULT_SUCCESS Then
'    MsgBox "PmcmSetSensorConfig ERROR : 0x" & Hex(lngResult), vbCritical
'    Exit Sub
'End If

'パルス出力モード設定
'使用しているドライバに合致したパルス出力モードを選択してください
lngResult = PmcmSetPulseConfig(m_intID, intAxis, PMCM_PULSE_OUT, 7)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetPulseConfig ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

動作パラメータ設定

```
Dim lngResult As Long
Dim intAxis As Integer
Dim Motion(2) As MOTIONPMCM
Dim intEmpty(1) As Integer
Dim intCount As Integer

intAxis = PMCM_AXIS_X

'CP動作パラメータ数
intCount = 3

'1回目
Motion(0).wMoveMode = PMCM_PTP '動作モード
Motion(0).wStartMode = PMCM_ACC_DEC '起動モード
Motion(0).fSpeedRate = 1 '速度倍率
Motion(0).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
Motion(0).fLowSpeed = 100 '起動時速度
Motion(0).fSpeed = 500 '移動速度
Motion(0).wAccTime = 500 '加速時間
Motion(0).wDecTime = 500 '減速時間
Motion(0).fSAccSpeed = 0 '加速S字区間
Motion(0).fSDecSpeed = 0 '減速S字区間
Motion(0).lSlowdown = 0 'スローダウンポイント
Motion(0).lStep = 1000 '移動パルス数,移動方向
Motion(0).bAbsolutePtp = 0 '絶対座標指定
'2回目
Motion(1).wMoveMode = PMCM_PTP '動作モード
Motion(1).wStartMode = PMCM_ACC_DEC '起動モード
Motion(1).fSpeedRate = 1 '速度倍率
Motion(1).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
Motion(1).fLowSpeed = 500 '起動時速度
Motion(1).fSpeed = 1000 '移動速度
Motion(1).wAccTime = 500 '加速時間
Motion(1).wDecTime = 500 '減速時間
Motion(1).fSAccSpeed = 0 '加速S字区間
Motion(1).fSDecSpeed = 0 '減速S字区間
Motion(1).lSlowdown = -1 'スローダウンポイント
Motion(1).lStep = 2000 '移動パルス数,移動方向
Motion(1).bAbsolutePtp = 0 '絶対座標指定
'3回目
Motion(2).wMoveMode = PMCM_PTP '動作モード
Motion(2).wStartMode = PMCM_CONST_DEC '起動モード
Motion(2).fSpeedRate = 1 '速度倍率
Motion(2).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
Motion(2).fLowSpeed = 100 '起動時速度
Motion(2).fSpeed = 500 '移動速度
Motion(2).wAccTime = 0 '加速時間
Motion(2).wDecTime = 500 '減速時間
Motion(2).fSAccSpeed = 0 '加速S字区間
Motion(2).fSDecSpeed = 0 '減速S字区間
Motion(2).lSlowdown = -1 'スローダウンポイント
Motion(2).lStep = 1500 '移動パルス数,移動方向
Motion(2).bAbsolutePtp = 0 '絶対座標指定

'CPの空き取得
lngResult = PmcmGetEmptyCp(m_intID, intAxis, intEmpty(0))
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmGetEmptyCp ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
'CPの空き確認
```

```
If intEmpty(0) < intCount Then
    MsgBox "CP空きなし", vbCritical
    Exit Sub
End If

'動作パラメータ設定
lngResult = PmcmSetMotionCp(m_intID, intAxis, Motion(0), intCount)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetMotionCp ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

動作開始

```
Dim lngResult As Long
Dim intAxis As Integer

'動作させる軸
intAxis = PMCM_AXIS_X

'動作開始
lngResult = PmcmStartMotionCp(m_intID, intAxis)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmStartMotionCp ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

動作追加

```
'CP動作は96動作までしか設定できません
'それ以上の動作を設定したい場合は、動作中にCPの空きを確認し、空き数分の動作を開始します
Dim lngResult As Long
Dim intAxis As Integer
Dim Motion(99) As MOTIONPMCM
Dim intEmpty(1) As Integer
Dim intCount As Integer
Dim intRemainCount As Integer
Dim intSetCount As Integer
Dim i As Integer

intAxis = PMCM_AXIS_X

'CP動作パラメータ数
intCount = 100

For i = 0 To intCount - 1
    Motion(i).wMoveMode = PMCM_PTP '動作モード
    Motion(i).wStartMode = PMCM_CONST '起動モード
    Motion(i).fSpeedRate = 1 '速度倍率
    Motion(i).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
    Motion(i).fLowSpeed = 1000 '起動時速度
    Motion(i).fSpeed = 1000 '移動速度
    Motion(i).wAccTime = 0 '加速時間
    Motion(i).wDecTime = 0 '減速時間
    Motion(i).fSAccSpeed = 0 '加速S字区間
    Motion(i).fSDecSpeed = 0 '減速S字区間
    Motion(i).lSlowdown = -1 'スローダウンポイント
    Motion(i).lStep = 1000 '移動パルス数,移動方向
    Motion(i).bAbsolutePtp = 0 '絶対座標指定
```

```
Next

'残り動作数
intRemainCount = intCount

Do While intRemainCount > 0
    'CPの空き取得
    lngResult = PmcmGetEmptyCp(m_intID, intAxis, intEmpty(0))
    If lngResult <> PMCM_RESULT_SUCCESS Then
        MsgBox "PmcmGetEmptyCp ERROR : 0x" & Hex(lngResult), vbCritical
        Exit Sub
    End If
    'CPの空きを確認し、設定動作数を決定する
    If intEmpty(0) = 0 Then 'CP空きなし
        intSetCount = 0
    ElseIf intEmpty(0) < intRemainCount Then 'CP空き < 残り動作数
        intSetCount = intEmpty(0) 'CP空き分だけ設定する
    ElseIf intEmpty(0) >= intRemainCount Then 'CP空き >= 残り動作数
        intSetCount = intRemainCount '残り動作数を設定する
    End If

    If intSetCount > 0 Then
        '動作パラメータ設定
        lngResult = PmcmSetMotionCp(m_intID, intAxis, Motion(intCount - intRemainCount),
intSetCount)
        If lngResult <> PMCM_RESULT_SUCCESS Then
            MsgBox "PmcmSetMotionCp ERROR : 0x" & Hex(lngResult), vbCritical
            Exit Sub
        End If
        '動作開始
        lngResult = PmcmStartMotionCp(m_intID, intAxis)
        If lngResult <> PMCM_RESULT_SUCCESS Then
            MsgBox "PmcmStartMotionCp ERROR : 0x" & Hex(lngResult), vbCritical
            Exit Sub
        End If
    End If
    '残り動作数更新
    intRemainCount = intRemainCount - intSetCount
Loop
```

動作停止

```
Dim lngResult As Long
Dim intAxis As Integer
Dim intStopMode As Integer

'停止させる軸
intAxis = PMCM_AXIS_X

'停止モード
intStopMode = PMCM_IMMEDIATE_STOP

'動作停止
lngResult = PmcmStopMotion(m_intID, intAxis, intStopMode)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmStopMotion ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

クローズ

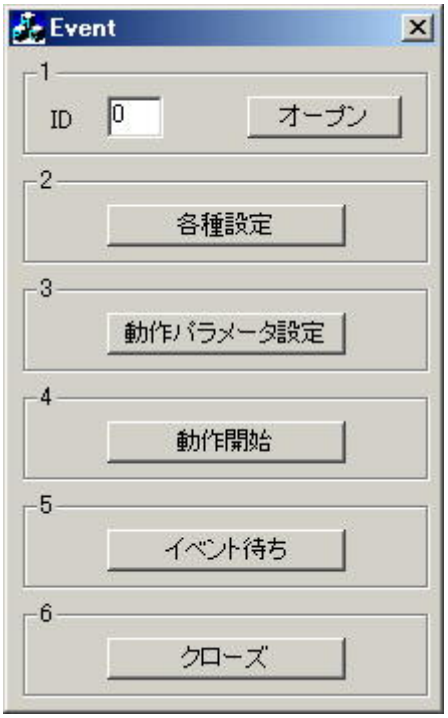
```
Dim blnResult As Boolean

'ボードクローズ
blnResult = PmcmClose(m_intID)
```

Event

動作概要

モーターの正常停止時にイベントを発生させます。

画面



1. オープン

   ボードのオープンをおこないます。

2. 各種設定

   使用環境に応じて、センサ設定及びパルス出力設定をおこないます。

3. 動作パラメータ設定

   動作パラメータを設定します。

4. 動作開始

   モーター動作を開始します。

5. イベント待ち

   イベント待ちをおこないます。モーターが正常停止するまで終了しません。

6. クローズ

   ボードのクローズをおこないます。
   オープンをおこなった場合は必ず実行する必要があります。

備考

⚠ 非常停止信号（EMG）・アラーム信号（ALM）・エンドリミット信号（+EL、-EL）がONの場合、モーターは動作しません

サンプルソース

- Visual C++
- Visual C++/CLI
- Visual C#
- Visual Basic（.NET2002以降）
- Visual Basic 6.0/VBA

オブジェクト

テキストボックス	ID	メンバ変数
ID番号	IDC_ID	m_wID

変数

```
HANDLE m_hEvent;
```

オープン

```
if( !UpdateData( TRUE ) ) {
    return;
}

int nResult;

nResult = PmcmOpen( m_wID, "PMC-M2C-U" );
if( nResult == PMCM_RESULT_SUCCESS ) {
    AfxMessageBox( "オープン成功", MB_OK | MB_ICONINFORMATION );
} else {
    AfxMessageBox( "オープン失敗", MB_OK | MB_ICONSTOP );
}
```

各種設定

```
int nResult;
WORD wAxis;
char szResult[64];

wAxis = PMCM_AXIS_X + PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//nResult = PmcmSetSensorConfig( m_wID, wAxis, PMCM_LOGIC, 0x3F );
//if( nResult != PMCM_RESULT_SUCCESS ) {
//    wsprintf( szResult, "PmcmSetSensorConfig ERROR : 0x%X", nResult );
//    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
nResult = PmcmSetPulseConfig( m_wID, wAxis, PMCM_PULSE_OUT, 7 );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetPulseConfig ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
```

```
}

// イベントマスク設定
// 自動停止時のイベント発生を許可に設定します
nResult = PmcmSetEventMask( m_wID, wAxis, PMCM_EVENT_STOP, 0x01 );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetEventMask ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}

// イベントオブジェクト作成
m_hEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
if( m_hEvent == NULL ) {
    AfxMessageBox( "イベントオブジェクト作成失敗", MB_OK | MB_ICONSTOP );
    return;
}

// イベント設定
nResult = PmcmSetEvent( m_wID, m_hEvent );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetEvent ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

動作パラメータ設定

```
int nResult;
WORD wAxis;
MOTIONPMCM Motion[2];
char szResult[64];

wAxis = PMCM_AXIS_X;

// X軸
Motion[0].wMoveMode = PMCM_PTP; // 動作モード
Motion[0].wStartMode = PMCM_CONST; // 起動モード
Motion[0].fSpeedRate = 1; // 速度倍率
Motion[0].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
Motion[0].fLowSpeed = 1000; // 起動時速度
Motion[0].fSpeed = 1000; // 移動速度
Motion[0].wAccTime = 0; // 加速時間
Motion[0].wDecTime = 0; // 減速時間
Motion[0].fSAccSpeed = 0; // 加速S字区間
Motion[0].fSDecSpeed = 0; // 減速S字区間
Motion[0].lSlowdown = -1; // スローダウンポイント
Motion[0].lStep = 10000; // 移動パルス数,移動方向
Motion[0].bAbsolutePtp = 0; // 絶対座標指定
// 動作パラメータ設定
nResult = PmcmSetMotion( m_wID, wAxis, Motion );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmSetMotion ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

動作開始

```
int nResult;
WORD wAxis;
char szResult[64];

// 動作させる軸
wAxis = PMCM_AXIS_X;

// 動作開始
nResult = PmcmStartMotion( m_wID, wAxis );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmStartMotion ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
```

イベント待ち

```
int nResult;
EVENTFACTORPMCM EventFactor;
char szResult[64];

// イベント発生待ち
WaitForSingleObject( m_hEvent, INFINITE );

// イベントクリア
ResetEvent( m_hEvent );

// イベント要因取得
nResult = PmcmGetEventFactor( m_wID, &EventFactor );
if( nResult != PMCM_RESULT_SUCCESS ) {
    wsprintf( szResult, "PmcmGetEventFactor ERROR : 0x%X", nResult );
    AfxMessageBox( szResult, MB_OK | MB_ICONSTOP );
    return;
}
if( EventFactor.nResult != 0 ) {
    AfxMessageBox( "イベント失敗", MB_OK | MB_ICONSTOP );
    return;
}
wsprintf( szResult, "停止イベント要因 : H'%X\n状態イベント要因 : H'%X", EventFactor.wStop[0],
EventFactor.wState[0] );
AfxMessageBox( szResult, MB_OK | MB_ICONINFORMATION );
```

クローズ

```
BOOL bResult;

CloseHandle( m_hEvent );
m_hEvent = NULL;

bResult = PmcmClose( m_wID );
```

オブジェクト

テキストボックス	Name
ID番号	idTextBox

変数

```
unsigned short id;
ManualResetEvent^ eventHandle;
```

オープン

```
int result;

id = Convert::ToUInt16(idTextBox->Text);
result = PmcmOpen(id, "PMC-M2C-U");
if (result == PMCM_RESULT_SUCCESS) {
    MessageBox::Show("オープン成功", "", MessageBoxButtons::OK, MessageBoxIcon::Information);
} else {
    MessageBox::Show("オープン失敗", "", MessageBoxButtons::OK, MessageBoxIcon::Error);
}
```

各種設定

```
int result;
unsigned short axis;
String^ messageText;

axis = PMCM_AXIS_X + PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//result = PmcmSetSensorConfig(id, axis, PMCM_LOGIC, 0x3F);
//if (result != PMCM_RESULT_SUCCESS) {
//    messageText = String::Format("PmcmSetSensorConfig ERROR : 0x{0:X}", result);
//    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
result = PmcmSetPulseConfig(id, axis, PMCM_PULSE_OUT, 7);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetPulseConfig ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

```
// イベントマスク設定
// 自動停止時のイベント発生を許可に設定します
result = PmcmSetEventMask(id, axis, PMCM_EVENT_STOP, 0x01);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetEventMask ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}

// イベントオブジェクト作成
eventHandle = gcnew ManualResetEvent(false);

// イベント設定
result = PmcmSetEvent(id, eventHandle->Handle);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetEvent ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

動作パラメータ設定

```
int result;
unsigned short axis;
MOTIONPMCM motion[2];
String^ messageText;

axis = PMCM_AXIS_X;

// X軸
motion[0].wMoveMode = PMCM_PTP; // 動作モード
motion[0].wStartMode = PMCM_CONST; // 起動モード
motion[0].fSpeedRate = 1; // 速度倍率
motion[0].wAccDecMode = PMCM_ACC_LINEAR; // 加減速モード
motion[0].fLowSpeed = 1000; // 起動時速度
motion[0].fSpeed = 1000; // 移動速度
motion[0].wAccTime = 0; // 加速時間
motion[0].wDecTime = 0; // 減速時間
motion[0].fSAccSpeed = 0; // 加速S字区間
motion[0].fSDecSpeed = 0; // 減速S字区間
motion[0].lSlowdown = -1; // スローダウンポイント
motion[0].lStep = 10000; // 移動パルス数,移動方向
motion[0].bAbsolutePtp = 0; // 絶対座標指定
// 動作パラメータ設定
result = PmcmSetMotion(id, axis, motion);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmSetMotion ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

動作開始

```
int result;
unsigned short axis;
String^ messageText;

// 動作させる軸
axis = PMCM_AXIS_X;
```

```
// 動作開始
result = PmcmStartMotion(id, axis);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmStartMotion ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
```

イベント待ち

```
int result;
EVENTFACTORPMCM eventFactor;
String^ messageText;

// イベント発生待ち
eventHandle->WaitOne();

// イベントクリア
eventHandle->Reset();

// イベント要因取得
result = PmcmGetEventFactor(id, &eventFactor);
if (result != PMCM_RESULT_SUCCESS) {
    messageText = String::Format("PmcmGetEventFactor ERROR : 0x{0:X}", result);
    MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
if (eventFactor.nResult != 0) {
    MessageBox::Show("イベント失敗", "", MessageBoxButtons::OK, MessageBoxIcon::Error);
    return;
}
messageText = String::Format("停止イベント要因 : H'{0:X}\n状態イベント要因 : H'{1:X}",
eventFactor.wStop[0], eventFactor.wState[0]);
MessageBox::Show(messageText, "", MessageBoxButtons::OK, MessageBoxIcon::Information);
```

クローズ

```
bool result;

eventHandle->Close();

result = PmcmClose(id);
```

オブジェクト

テキストボックス	Name
ID番号	Id

変数

```
ushort id;
ManualResetEvent eventHandle;
```

オープン

```
int result;

id = Convert.ToUInt16(Id.Text);
result = Pmcm.Open( id, "PMC-M2C-U" );
if( result == Pmcm.PMCM_RESULT_SUCCESS )
{
    MessageBox.Show( "オープン成功", "", MessageBoxButtons.OK, MessageBoxIcon.Information );
}
else
{
    MessageBox.Show( "オープン失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
}
```

各種設定

```
int result;
ushort axis;
string outputString;

axis = Pmcm.PMCM_AXIS_X + Pmcm.PMCM_AXIS_Y;

// センサ設定
// オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
// その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
//result = Pmcm.SetSensorConfig( id, axis, Pmcm.PMCM_LOGIC, 0x3F );
//if( result != Pmcm.PMCM_RESULT_SUCCESS )
//{
//    outputString = String.Format( "PmcmSetSensorConfig ERROR : 0x{0:X}", result );
//    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
//    return;
//}

// パルス出力モード設定
// 使用しているドライバに合致したパルス出力モードを選択してください
result = Pmcm.SetPulseConfig( id, axis, Pmcm.PMCM_PULSE_OUT, 7 );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
```

```
    outputString = String.Format( "PmcmSetPulseConfig ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}

// イベントマスク設定
// 自動停止時のイベント発生を許可に設定します
result = Pmcm.SetEventMask( id, axis, Pmcm.PMCM_EVENT_STOP, 0x01 );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetEventMask ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}

// イベントオブジェクト作成
eventHandle = new ManualResetEvent(false);

// イベント設定
result = Pmcm.SetEvent( id, eventHandle.Handle );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetEvent ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

動作パラメータ設定

```
int result;
ushort axis;
Pmcm.MOTIONPMCM[] motion = new Pmcm.MOTIONPMCM[2];
string outputString;

axis = Pmcm.PMCM_AXIS_X;

// X軸
motion[0].wMoveMode = Pmcm.PMCM_PTP; // 動作モード
motion[0].wStartMode = Pmcm.PMCM_CONST; // 起動モード
motion[0].fSpeedRate = 1; // 速度倍率
motion[0].wAccDecMode = Pmcm.PMCM_ACC_LINEAR; // 加減速モード
motion[0].fLowSpeed = 1000; // 起動時速度
motion[0].fSpeed = 1000; // 移動速度
motion[0].wAccTime = 0; // 加速時間
motion[0].wDecTime = 0; // 減速時間
motion[0].fSAccSpeed = 0; // 加速S字区間
motion[0].fSDecSpeed = 0; // 減速S字区間
motion[0].lSlowdown = -1; // スローダウンポイント
motion[0].lStep = 10000; // 移動パルス数,移動方向
motion[0].bAbsolutePtp = 0; // 絶対座標指定
// 動作パラメータ設定
result = Pmcm.SetMotion( id, axis, motion );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmSetMotion ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

動作開始

```
int result;
ushort axis;
string outputString;

// 動作させる軸
axis = Pmcm.PMCM_AXIS_X;

// 動作開始
result = Pmcm.StartMotion( id, axis );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmStartMotion ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
```

イベント待ち

```
int result;
Pmcm.EVENTFACTORPMCM eventFactor = new Pmcm.EVENTFACTORPMCM();
eventFactor.Initialize();
string outputString;

// イベント発生待ち
eventHandle.WaitOne();

// イベントクリア
eventHandle.Reset();

// イベント要因取得
result = Pmcm.GetEventFactor( id, out eventFactor );
if( result != Pmcm.PMCM_RESULT_SUCCESS )
{
    outputString = String.Format( "PmcmGetEventFactor ERROR : 0x{0:X}", result );
    MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
if( eventFactor.nResult != 0 )
{
    MessageBox.Show( "イベント失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Stop );
    return;
}
outputString = String.Format( "停止イベント要因 : H'{0:X}\n状態イベント要因 : H'{1:X}",
eventFactor.wStop[0], eventFactor.wState[0] );
MessageBox.Show( outputString, "", MessageBoxButtons.OK, MessageBoxIcon.Information );
```

クローズ

```
bool result;

eventHandle.Close();

result = Pmcm.Close( id );
```

Visual Basic（.NET2002以降）

オブジェクト

テキストボックス	Name
ID番号	idTextBox

変数

```
Dim id As Short
Dim eventHandle As ManualResetEvent
```

オープン

```
Dim result As Integer

id = Convert.ToInt16(idTextBox.Text)
result = PmcmOpen(id, "PMC-M2C-U")
If result = PMCM_RESULT_SUCCESS Then
    MessageBox.Show("オープン成功", "", MessageBoxButtons.OK, MessageBoxIcon.Information)
Else
    MessageBox.Show("オープン失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Error)
End If
```

各種設定

```
Dim result As Integer
Dim axis As Short

axis = PMCM_AXIS_X + PMCM_AXIS_Y

'センサ設定
'オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
'その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
'result = PmcmSetSensorConfig(id, axis, PMCM_LOGIC, &H3F)
'If result <> PMCM_RESULT_SUCCESS Then
'    MessageBox.Show("PmcmSetSensorConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
'    Exit Sub
'End If

'パルス出力モード設定
'使用しているドライバに合致したパルス出力モードを選択してください
result = PmcmSetPulseConfig(id, axis, PMCM_PULSE_OUT, 7)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetPulseConfig ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If

'イベントマスク設定
```

```
'自動停止時のイベント発生を許可に設定します
result = PmcmSetEventMask(id, axis, PMCM_EVENT_STOP, &H1)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetEventMask ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If

'イベントオブジェクト作成
eventHandle = New ManualResetEvent(False)

'イベント設定
result = PmcmSetEvent(id, eventHandle.Handle)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetEvent ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

動作パラメータ設定

```
Dim result As Integer
Dim axis As Short
Dim motion(1) As MOTIONPMCM

axis = PMCM_AXIS_X

'X軸
motion(0).wMoveMode = PMCM_PTP '動作モード
motion(0).wStartMode = PMCM_CONST '起動モード
motion(0).fSpeedRate = 1 '速度倍率
motion(0).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
motion(0).fLowSpeed = 1000 '起動時速度
motion(0).fSpeed = 1000 '移動速度
motion(0).wAccTime = 0 '加速時間
motion(0).wDecTime = 0 '減速時間
motion(0).fSAccSpeed = 0 '加速S字区間
motion(0).fSDecSpeed = 0 '減速S字区間
motion(0).lSlowdown = -1 'スローダウンポイント
motion(0).lStep = 10000 '移動パルス数,移動方向
motion(0).bAbsolutePtp = 0 '絶対座標指定
'動作パラメータ設定
result = PmcmSetMotion(id, axis, motion)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmSetMotion ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

動作開始

```
Dim result As Integer
Dim axis As Short

'動作させる軸
axis = PMCM_AXIS_X

'動作開始
result = PmcmStartMotion(id, axis)
```

```
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmStartMotion ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
```

イベント待ち

```
Dim result As Integer
Dim eventFactor As EVENTFACTORPMCM
eventFactor.Initialize()

'イベント発生待ち
eventHandle.WaitOne()

'イベントクリア
eventHandle.Reset()

'イベント要因取得
result = PmcmGetEventFactor(id, eventFactor)
If result <> PMCM_RESULT_SUCCESS Then
    MessageBox.Show("PmcmGetEventFactor ERROR : 0x" & result.ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
If eventFactor.nResult <> 0 Then
    MessageBox.Show("イベント失敗", "", MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
End If
MessageBox.Show("停止イベント要因 : 0x" & eventFactor.wStop(0).ToString("X") &
ControlChars.CrLf & "状態イベント要因 : 0x" & eventFactor.wState(0).ToString("X"), "",
MessageBoxButtons.OK, MessageBoxIcon.Information)
```

クローズ

```
Dim result As Boolean

eventHandle.Close()

result = PmcmClose(id)
```

オブジェクト

テキストボックス	オブジェクト名
ID番号	txtID

変数

```
Dim m_intID As Integer
#If VBA7 Then
Dim m_lngEvent As LongPtr
#Else
Dim m_lngEvent As Long
#End If
```

オープン

```
Dim lngResult As Long
Dim strModelName As String

m_intID = Val(txtID.Text)
strModelName = "PMC-M2C-U"
lngResult = PmcmOpen(m_intID, strModelName)
If lngResult = PMCM_RESULT_SUCCESS Then
    MsgBox "オープン成功", vbInformation
Else
    MsgBox "オープン失敗", vbCritical
End If
```

各種設定

```
Dim lngResult As Long
Dim intAxis As Integer

intAxis = PMCM_AXIS_X + PMCM_AXIS_Y

'センサ設定
'オンで検知するセンサを接続している場合や、リミットスイッチを接続していない場合はモーターが動作しません
'その場合は以下の関数を実行してセンサ設定を"オンで検知"に変更してください
'lngResult = PmcmSetSensorConfig(m_intID, intAxis, PMCM_LOGIC, &H3F)
'If lngResult <> PMCM_RESULT_SUCCESS Then
'    MsgBox "PmcmSetSensorConfig ERROR : 0x" & Hex(lngResult), vbCritical
'    Exit Sub
'End If

'パルス出力モード設定
'使用しているドライバに合致したパルス出力モードを選択してください
lngResult = PmcmSetPulseConfig(m_intID, intAxis, PMCM_PULSE_OUT, 7)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetPulseConfig ERROR : 0x" & Hex(lngResult), vbCritical
```

```
    Exit Sub
End If

'イベントマスク設定
'自動停止時のイベント発生を許可に設定します
lngResult = PmcmSetEventMask(m_intID, intAxis, PMCM_EVENT_STOP, &H1)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetEventMask ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If

'イベントオブジェクト作成
m_lngEvent = CreateEvent(0, True, False, 0)
If m_lngEvent = 0 Then
    MsgBox "イベントオブジェクト作成失敗", vbCritical
    Exit Sub
End If

'イベント設定
lngResult = PmcmSetEvent(m_intID, m_lngEvent)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetEvent ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

動作パラメータ設定

```
Dim lngResult As Long
Dim intAxis As Integer
Dim Motion(1) As MOTIONPMCM

intAxis = PMCM_AXIS_X

'X軸
Motion(0).wMoveMode = PMCM_PTP '動作モード
Motion(0).wStartMode = PMCM_CONST '起動モード
Motion(0).fSpeedRate = 1 '速度倍率
Motion(0).wAccDecMode = PMCM_ACC_LINEAR '加減速モード
Motion(0).fLowSpeed = 1000 '起動時速度
Motion(0).fSpeed = 1000 '移動速度
Motion(0).wAccTime = 0 '加速時間
Motion(0).wDecTime = 0 '減速時間
Motion(0).fSAccSpeed = 0 '加速S字区間
Motion(0).fSDecSpeed = 0 '減速S字区間
Motion(0).lSlowdown = -1 'スローダウンポイント
Motion(0).lStep = 10000 '移動パルス数,移動方向
Motion(0).bAbsolutePtp = 0 '絶対座標指定
'動作パラメータ設定
lngResult = PmcmSetMotion(m_intID, intAxis, Motion(0))
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmSetMotion ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

動作開始

```
Dim lngResult As Long
Dim intAxis As Integer
```

```
'動作させる軸
intAxis = PMCM_AXIS_X

'動作開始
lngResult = PmcmStartMotion(m_intID, intAxis)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmStartMotion ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
```

イベント待ち

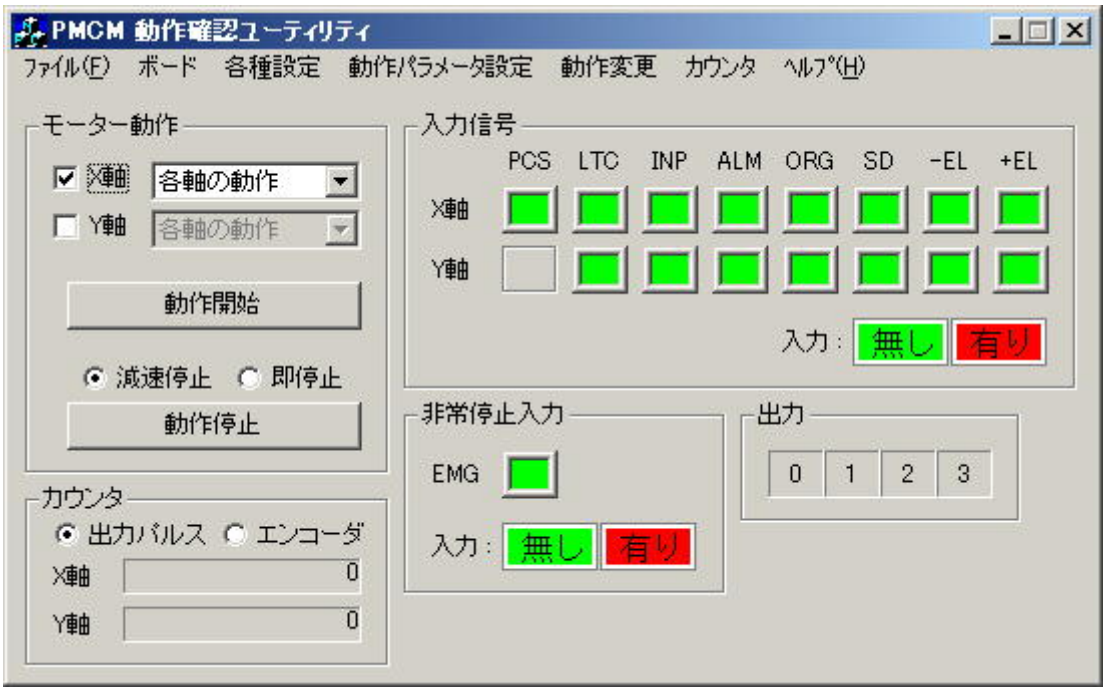
```
Dim lngResult As Long
Dim EventFactor As EVENTFACTORPMCM

'イベント発生待ち
lngResult = WaitForSingleObject(m_lngEvent, INFINITE)

'イベントクリア
ResetEvent m_lngEvent

'イベント要因取得
lngResult = PmcmGetEventFactor(m_intID, EventFactor)
If lngResult <> PMCM_RESULT_SUCCESS Then
    MsgBox "PmcmGetEventFactor ERROR : 0x" & Hex(lngResult), vbCritical
    Exit Sub
End If
If EventFactor.nResult <> 0 Then
    MsgBox "イベント失敗", vbCritical
    Exit Sub
End If
MsgBox "停止イベント要因 : H'" & Hex(EventFactor.wStop(0)) & vbCrLf & "状態イベント要因 : H'" &
Hex(EventFactor.wState(0)), vbInformation
```

クローズ

```
Dim blnResult As Boolean

'イベントオブジェクトクローズ
CloseHandle m_lngEvent
m_lngEvent = 0

'ボードクローズ
blnResult = PmcmClose(m_intID)
```

動作確認ユーティリティ（Windowsのみ）

動作確認ユーティリティを使用し、各動作の確認ができます。



モーター動作

モーター動作に関する項目です。
動作開始、動作停止を参照してください。

> ⚠ 動作パラメータに初期設定値はありませんので、動作パラメータを設定しないと動作開始をしても動作しません

カウンタ

出力パルスカウンタまたはエンコーダカウンタの値を表示します。
表示は100msec間隔で更新されます。

入力信号

100msec間隔で各入力信号の入力状態を監視しています。
入力が無い場合は緑、入力が有る場合は赤で表示されます。

> ⚠ Warning
>
> +ELに入力が有る場合、モーターはCW（+）方向に動作しません。
> -ELに入力が有る場合、モーターはCCW（-）方向に動作しません。
> ALMに入力が有る場合、モーターは動作しません。

非常停止入力

100msec間隔で非常停止入力（EMG）の入力状態を監視しています。
入力が無い場合は緑、入力が有る場合は赤で表示されます。

> ⚠ 非常停止入力に入力が有る場合、モーターは動作しません

出力

汎用デジタル出力端子（OUT0～3）を制御します。
出力番号の一覧が表示されていますので、出力ON/OFFを切り替えたい出力番号をクリックしてください。
ONの時には緑、OFFの時は灰色で表示されます。

メニュー

ファイル

メニュー名	説明
アプリケーションの終了	動作確認ユーティリティを終了します

ボード

メニュー名	説明
オープン	ボードのオープンをおこないます
クローズ	ボードのクローズをおこないます

各種設定

メニュー名	説明
センサ	センサの設定・取得をおこないます
パルス出力	パルス出力の設定・取得をおこないます
起動・停止条件	起動・停止条件の設定・取得をおこないます
内部同期信号出力	内部同期信号出力の設定・取得をおこないます
コンパレータ	コンパレータの設定・取得をおこないます

メニュー名	説明
カウンタラッチ	カウンタラッチの設定・取得をおこないます
エンコーダ	エンコーダの設定・取得をおこないます
ERC信号	ERC信号の設定・取得をおこないます

動作パラメータ設定

メニュー名	説明
各軸の動作	各軸の動作パラメータの設定・取得をおこないます
直線補間動作	直線補間動作パラメータの設定・取得をおこないます
CP動作	CP動作パラメータの設定・取得をおこないます

動作変更

メニュー名	説明
速度変更	動作中に速度の変更をおこないます
目標位置変更	PTP動作中に目標位置の変更をおこないます

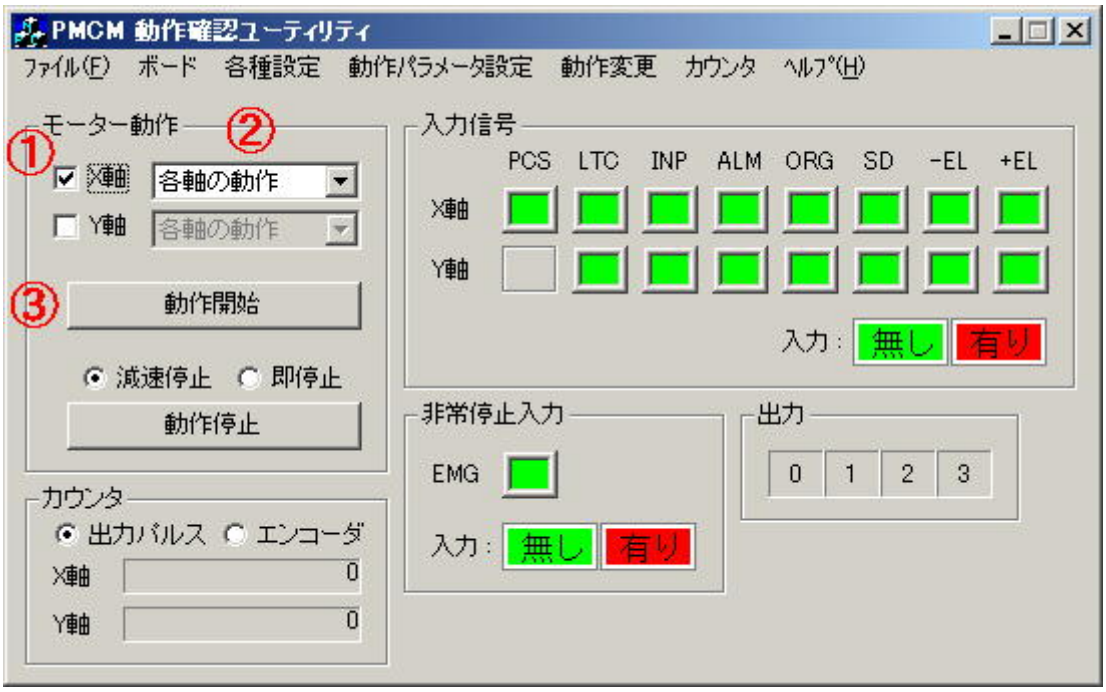
カウンタ

メニュー名	説明
カウンタ値設定	カウンタ値の設定をおこないます
ラッチ値取得	ラッチ値の取得をおこないます

動作開始

動作を開始する前に動作パラメータの設定をおこなってください。

> ⚠ 動作パラメータに初期設定値はありませんので、動作パラメータを設定しないと動作開始をしても動作しません



1. 動作させる軸をチェック状態にしてください
   直線補間動作をおこなう場合はX、Y軸ともにチェック状態にしてください。
2. 動作を選択してください
   直線補間動作をおこなう場合はX、Y軸ともに直線補間動作を選択してください。
3. 動作開始ボタンをクリックしてください

モーターが動作しない場合は以下の点を確認してください。

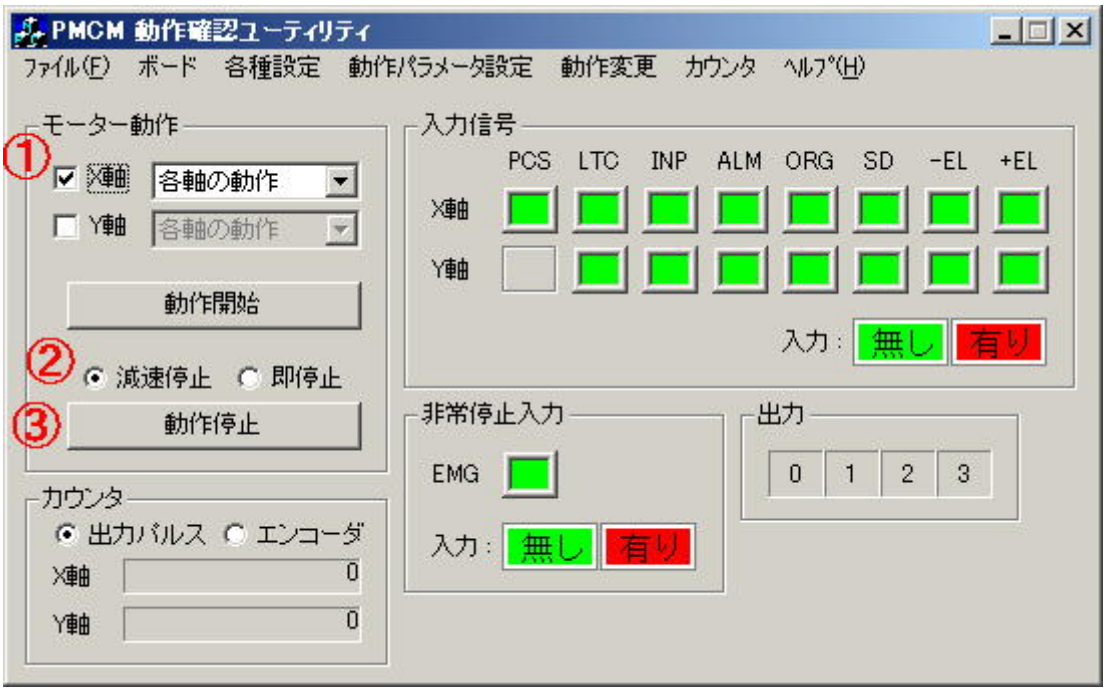
状態：非常停止入力の表示が赤
原因：非常停止入力が有る場合は動作しません
対処：非常停止入力をOFFにしてください

状態：ALM入力の表示が赤
原因：ALM入力が有る場合は動作しません
対処：ALM入力をOFFにするか、入力論理（センサ極性）を反転してください

状態 : 移動方向のEL入力の表示が赤
原因 : 移動方向のEL入力が有る場合は動作しません
対処 : 移動方向のEL入力をOFFにするか、入力論理（センサ極性）を反転してください

動作停止

動作を停止します。
設定の詳細についてはPmcmStopMotion関数の説明を参照してください。



1. 停止させる軸をチェック状態にしてください
2. 停止モードを選択してください
3. 動作停止ボタンをクリックしてください

オープン

メニューの[ボード]－[オープン]をクリックするとボードオープンの画面が表示されます。

1. ボードのID番号を選択してください
2. OKボタンをクリックしてください。オープン成功と表示されるとオープン完了です

クローズ

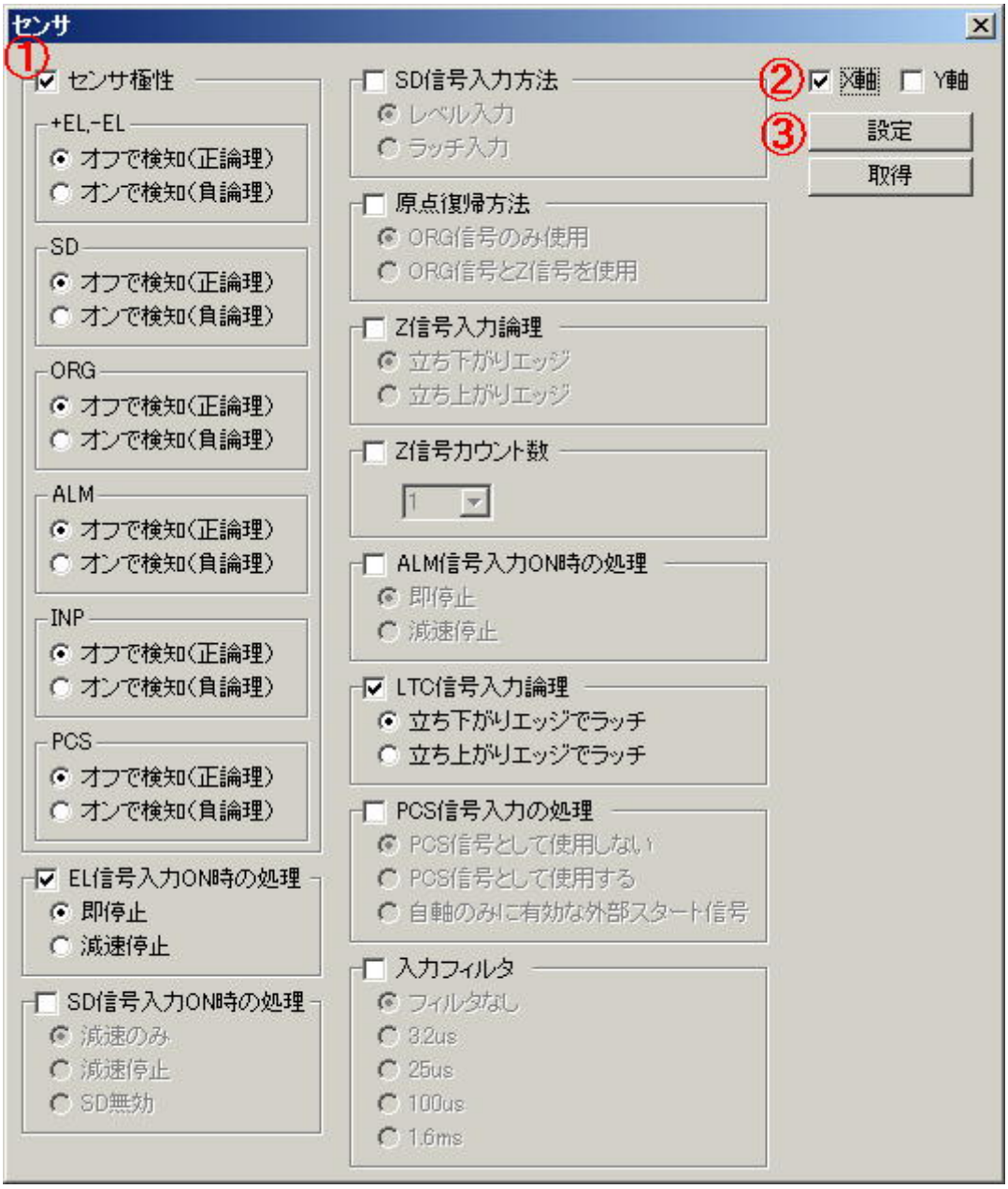
メニューの[ボード]−[クローズ]をクリックすると、現在オープンされているボードがクローズされます。
（ユーティリティ終了時にクローズ処理をおこなっていますので、クローズを実行せずにユーティリティを終了させても問題ありません）

センサの設定・取得

メニューの[各種設定]－[センサ]をクリックするとセンサの設定・取得画面が表示されます。

設定

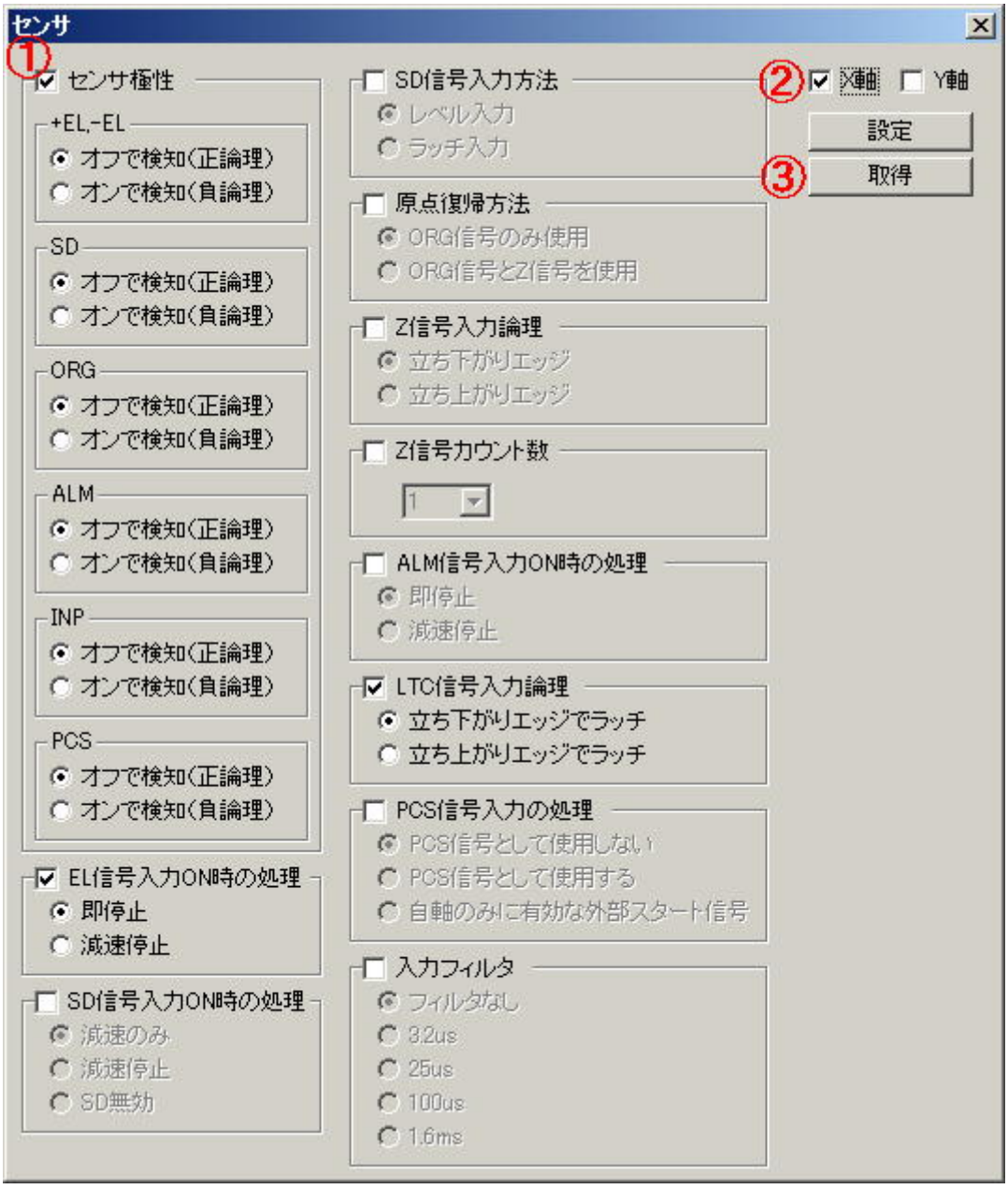
センサの設定をおこないます。
設定の詳細についてはPmcmSetSensorConfig関数の説明を参照してください。



1. 設定をおこなう項目をチェック状態にし、設定内容を選択してください（複数の設定項目を選択可能です）
2. 設定をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
3. 設定ボタンをクリックしてください。正常終了と表示されると設定完了です

取得

センサ設定の取得をおこないます。
設定の詳細についてはPmcmGetSensorConfig関数の説明を参照してください。



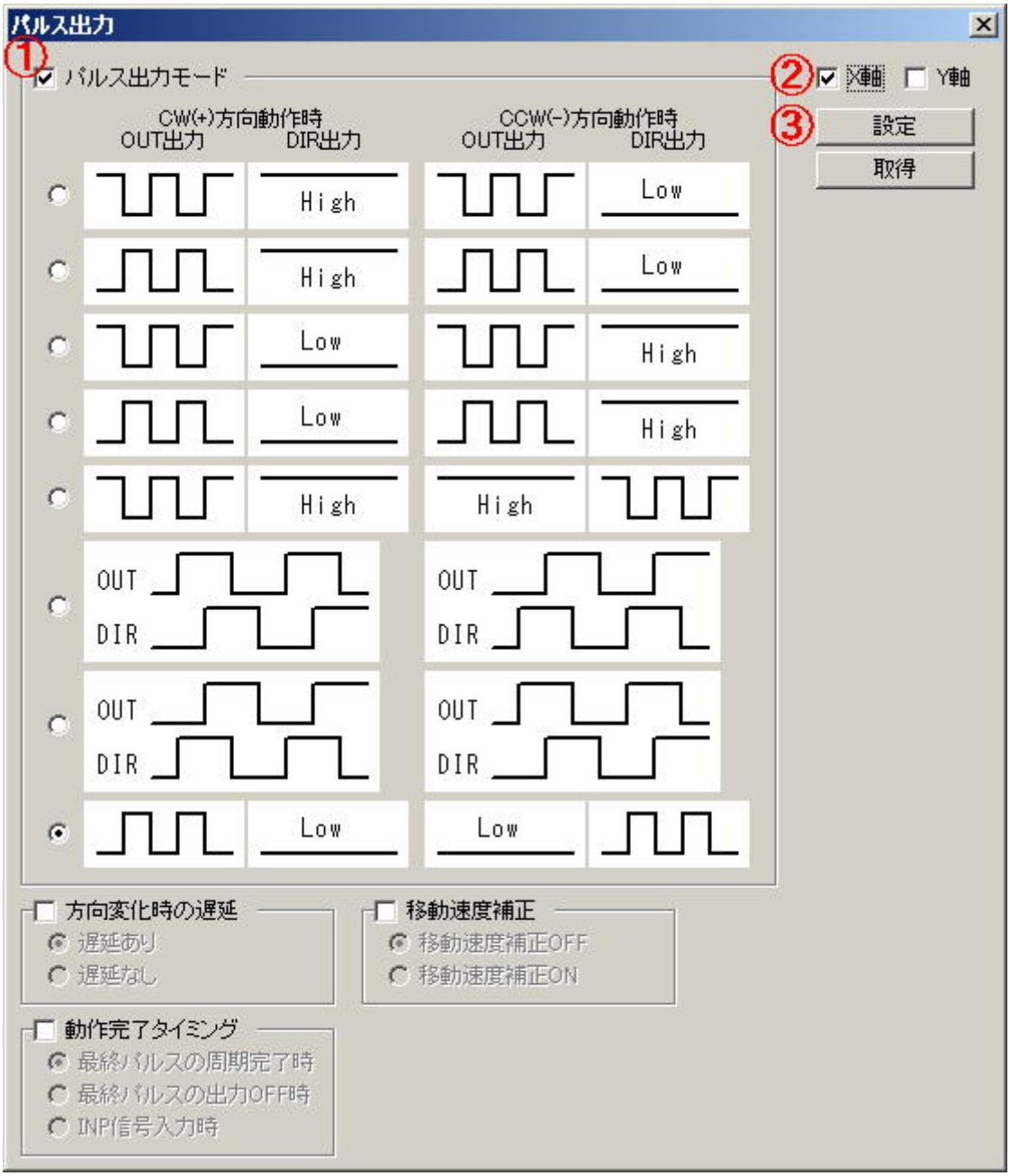
1. 設定を取得する項目をチェック状態にしてください（複数の取得項目を選択可能です）
2. 設定を取得する軸をチェック状態にしてください（複数の軸は選択できません）
3. 取得ボタンをクリックしてください。正常終了と表示されると取得完了です

パルス出力の設定・取得

メニューの[各種設定]－[パルス出力]をクリックするとパルス出力の設定・取得画面が表示されます。

設定

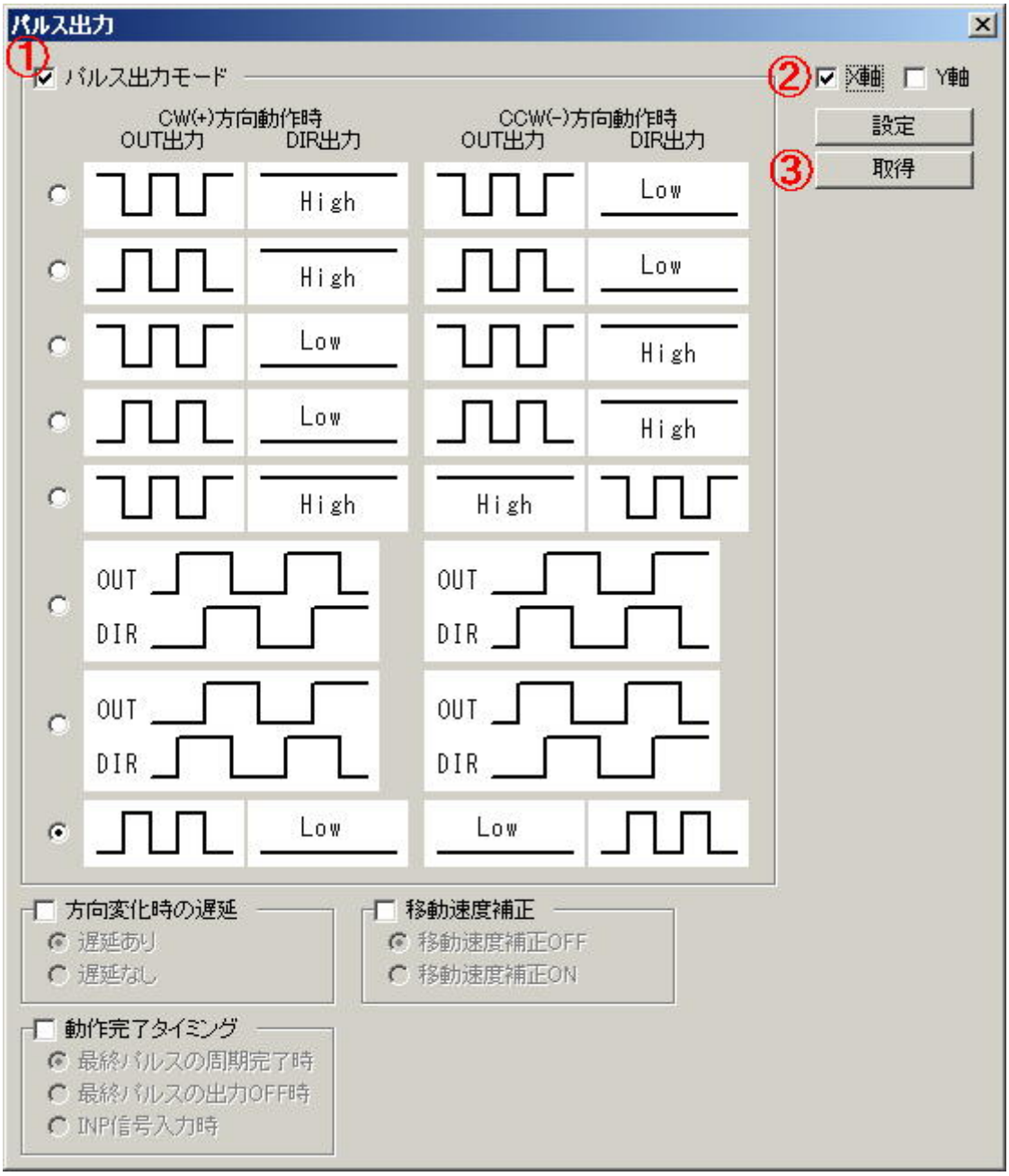
パルス出力の設定をおこないます。
設定の詳細についてはPmcmSetPulseConfig関数の説明を参照してください。



1. 設定をおこなう項目をチェック状態にし、設定内容を選択してください（複数の設定項目を選択可能です）
2. 設定をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
3. 設定ボタンをクリックしてください。正常終了と表示されると設定完了です

取得

パルス出力設定の取得をおこないます。
設定の詳細については[PmcmGetPulseConfig関数]の説明を参照してください。



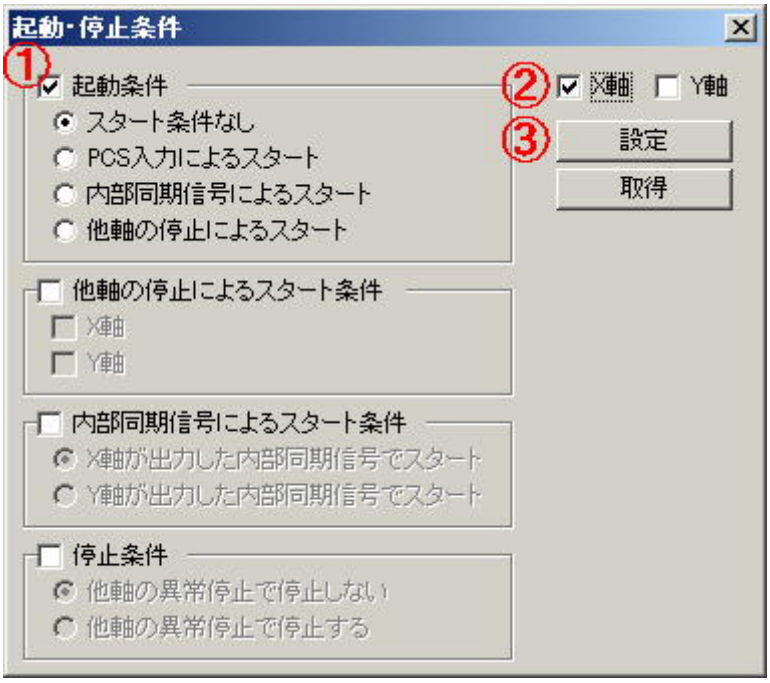
1. 設定を取得する項目をチェック状態にしてください（複数の取得項目を選択可能です）
2. 設定を取得する軸をチェック状態にしてください（複数の軸は選択できません）
3. 取得ボタンをクリックしてください。正常終了と表示されると取得完了です

起動・停止条件の設定・取得

メニューの[各種設定]－[起動・停止条件]をクリックすると起動・停止条件の設定・取得画面が表示されます。

設定

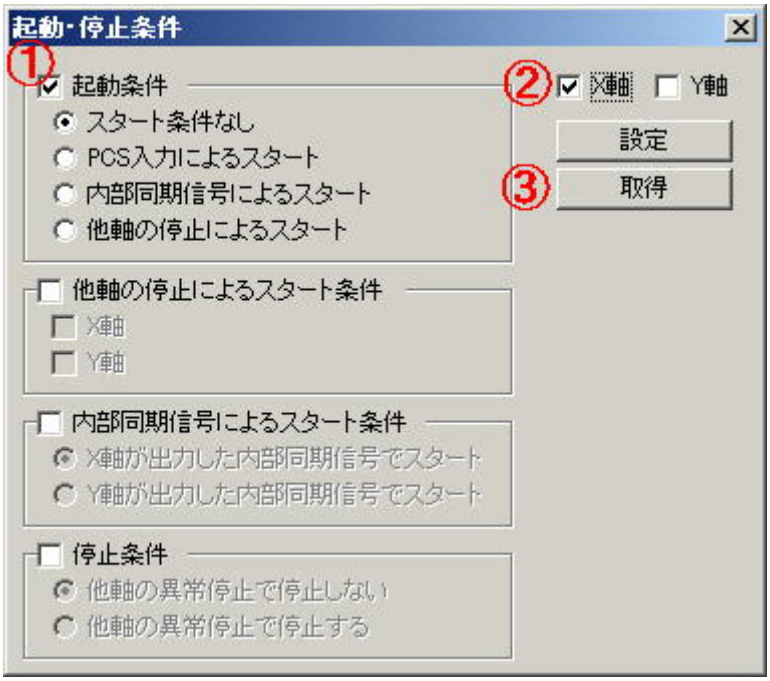
起動・停止条件の設定をおこないます。
設定の詳細についてはPmcmSetSynchroConfig関数の説明を参照してください。



1. 設定をおこなう項目をチェック状態にし、設定内容を選択してください（複数の設定項目を選択可能です）
2. 設定をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
3. 設定ボタンをクリックしてください。正常終了と表示されると設定完了です

取得

起動・停止条件設定の取得をおこないます。
設定の詳細についてはPmcmGetSynchroConfig関数の説明を参照してください。



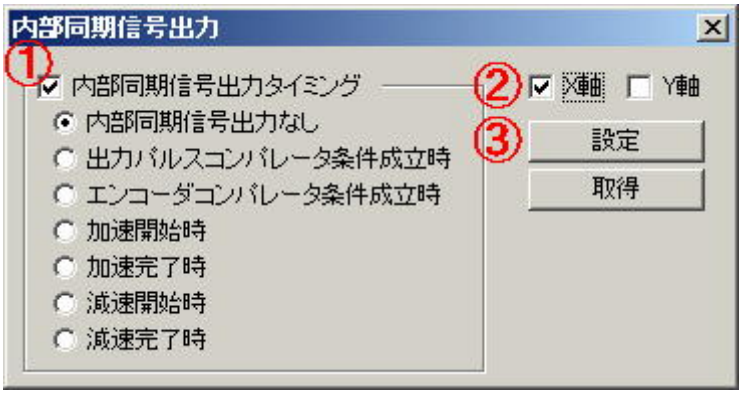
1. 設定を取得する項目をチェック状態にしてください（複数の取得項目を選択可能です）
2. 設定を取得する軸をチェック状態にしてください（複数の軸は選択できません）
3. 取得ボタンをクリックしてください。正常終了と表示されると取得完了です

内部同期信号出力の設定・取得

メニューの[各種設定]－[内部同期信号出力]をクリックすると内部同期信号出力の設定・取得画面が表示されます。

設定

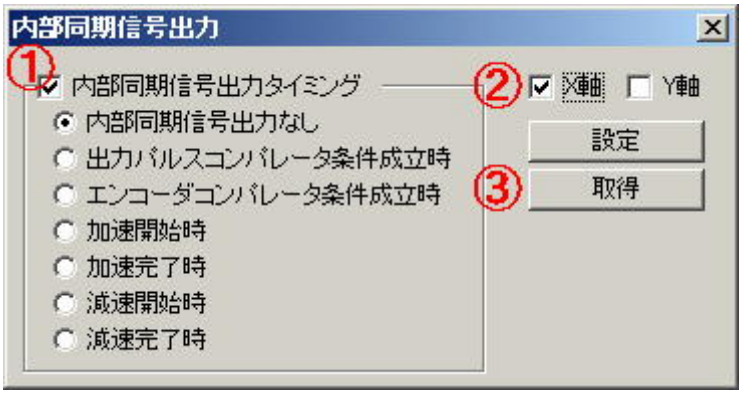
内部同期信号出力の設定をおこないます。
設定の詳細についてはPmcmSetSynchroOut関数の説明を参照してください。



1. 設定をおこなう項目をチェック状態にし、設定内容を選択してください
2. 設定をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
3. 設定ボタンをクリックしてください。正常終了と表示されると設定完了です

取得

内部同期信号出力設定の取得をおこないます。
設定の詳細についてはPmcmGetSynchroOut関数の説明を参照してください。



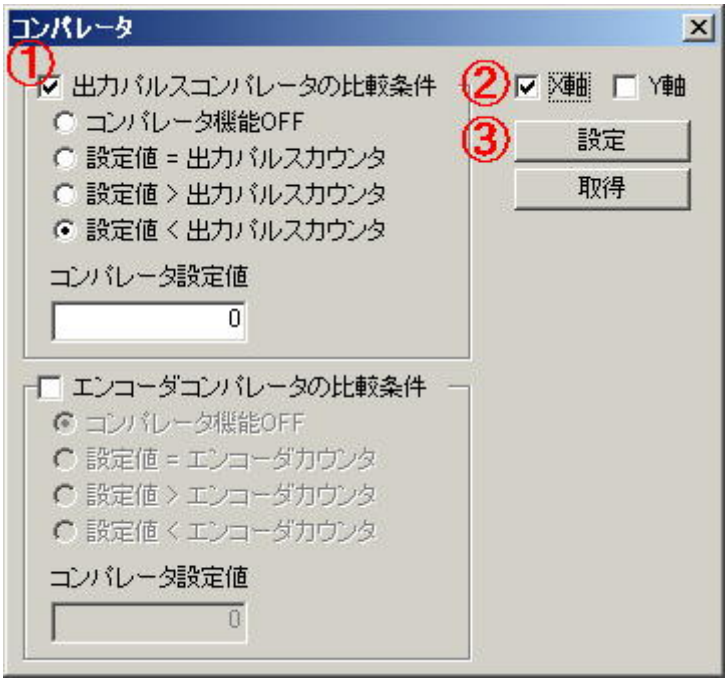
1. 設定を取得する項目をチェック状態にしてください
2. 設定を取得する軸をチェック状態にしてください（複数の軸は選択できません）
3. 取得ボタンをクリックしてください。正常終了と表示されると取得完了です

コンパレータの設定・取得

メニューの[各種設定]－[コンパレータ]をクリックするとコンパレータの設定・取得画面が表示されます。

設定

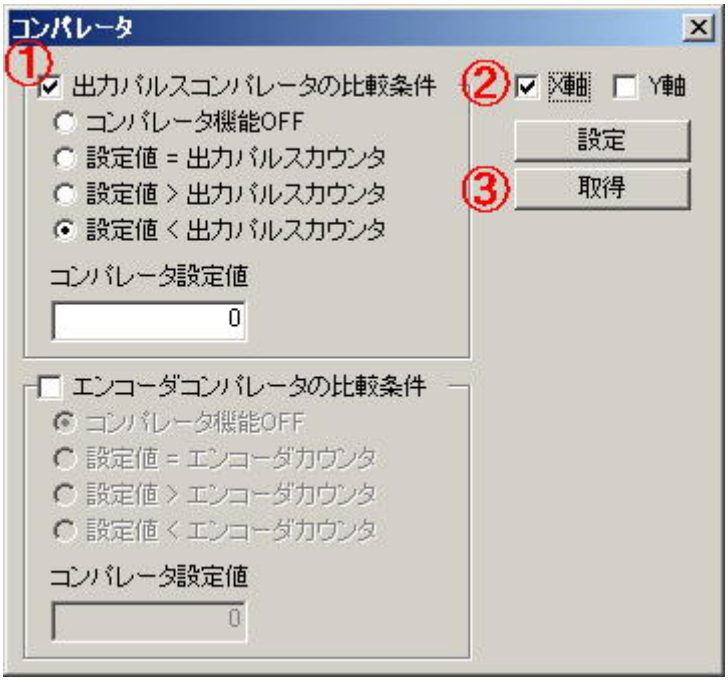
コンパレータの設定をおこないます。
設定の詳細についてはPmcmSetComparatorConfig関数の説明を参照してください。



1. 設定をおこなう項目をチェック状態にし、設定内容を選択・入力してください（複数の設定項目を選択可能です）
2. 設定をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
3. 設定ボタンをクリックしてください。正常終了と表示されると設定完了です

取得

コンパレータ設定の取得をおこないます。
設定の詳細についてはPmcmGetComparatorConfig関数の説明を参照してください。



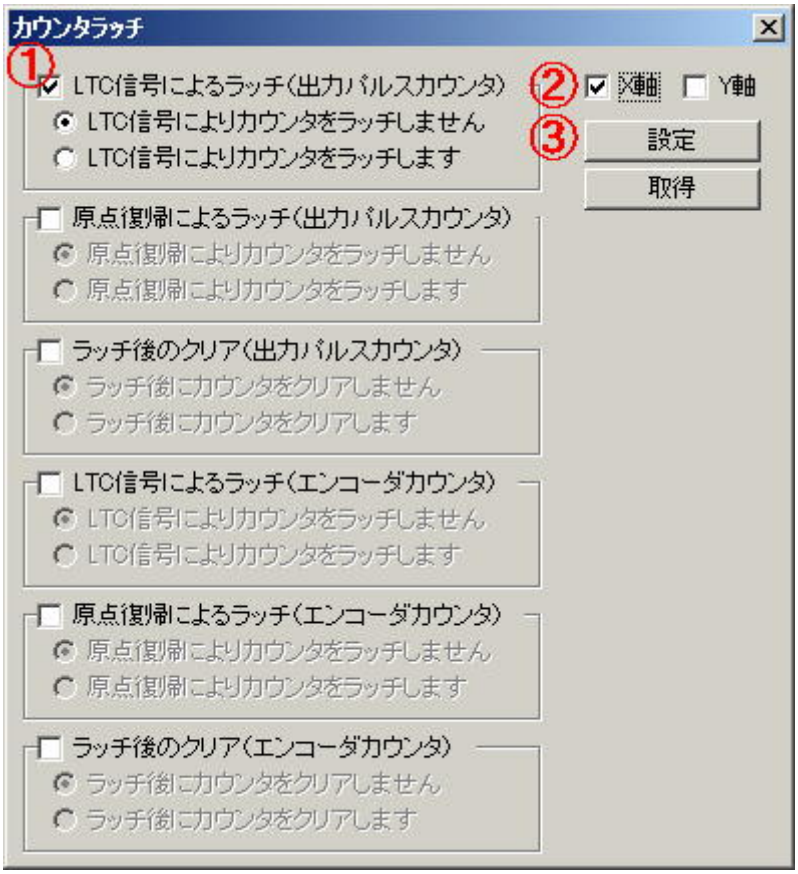
1. 設定を取得する項目をチェック状態にしてください（複数の取得項目を選択可能です）
2. 設定を取得する軸をチェック状態にしてください（複数の軸は選択できません）
3. 取得ボタンをクリックしてください。正常終了と表示されると取得完了です

カウンタラッチの設定・取得

メニューの[各種設定]－[カウンタラッチ]をクリックするとカウンタラッチの設定・取得画面が表示されます。

設定

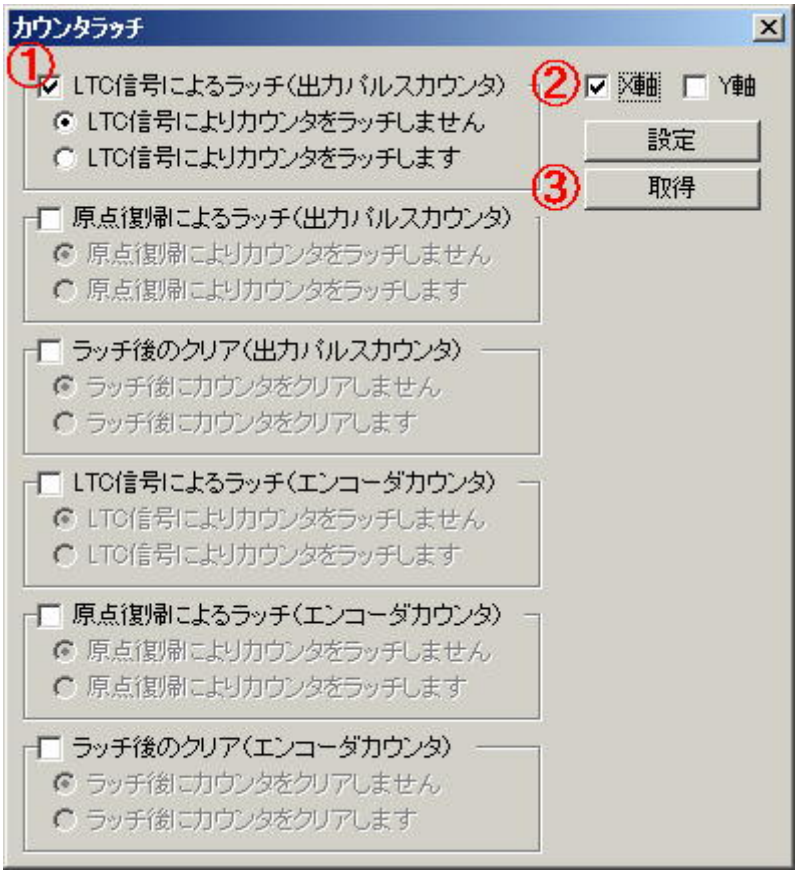
カウンタラッチの設定をおこないます。
設定の詳細についてはPmcmSetLatchConfig関数の説明を参照してください。



1. 設定をおこなう項目をチェック状態にし、設定内容を選択してください（複数の設定項目を選択可能です）
2. 設定をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
3. 設定ボタンをクリックしてください。正常終了と表示されると設定完了です

取得

カウンタラッチ設定の取得をおこないます。
設定の詳細についてはPmcmGetLatchConfig関数の説明を参照してください。



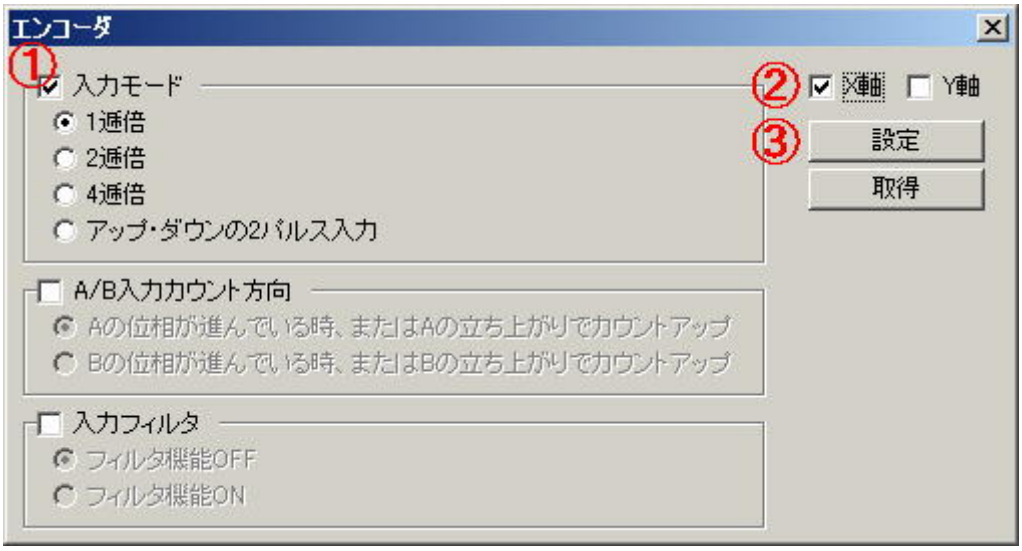
1. 設定を取得する項目をチェック状態にしてください（複数の取得項目を選択可能です）
2. 設定を取得する軸をチェック状態にしてください（複数の軸は選択できません）
3. 取得ボタンをクリックしてください。正常終了と表示されると取得完了です

エンコーダの設定・取得

メニューの[各種設定]－[エンコーダ]をクリックするとエンコーダの設定・取得画面が表示されます。

設定

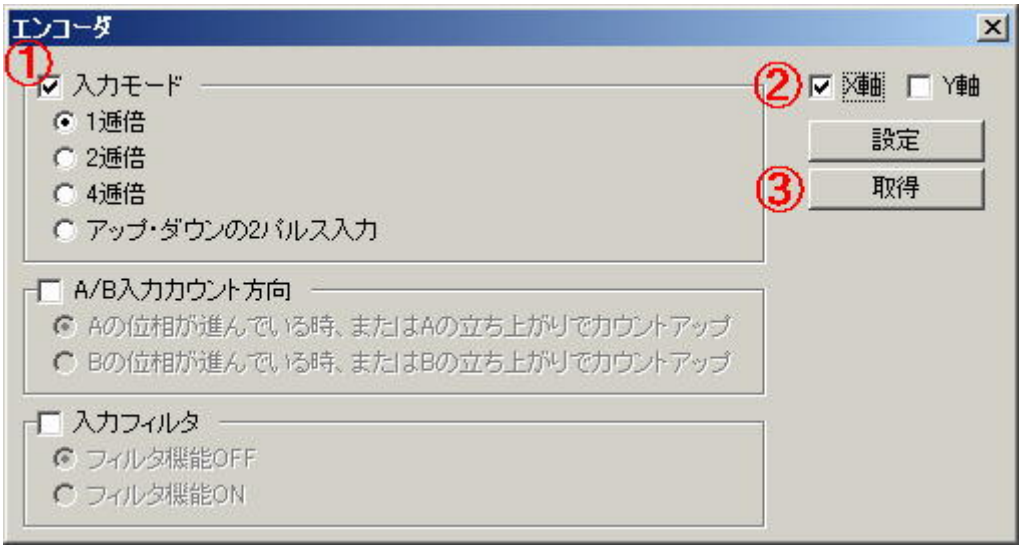
エンコーダの設定をおこないます。
設定の詳細についてはPmcmSetEncoderConfig関数の説明を参照してください。



1. 設定をおこなう項目をチェック状態にし、設定内容を選択してください（複数の設定項目を選択可能です）
2. 設定をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
3. 設定ボタンをクリックしてください。正常終了と表示されると設定完了です

取得

エンコーダ設定の取得をおこないます。
設定の詳細についてはPmcmGetEncoderConfig関数の説明を参照してください。



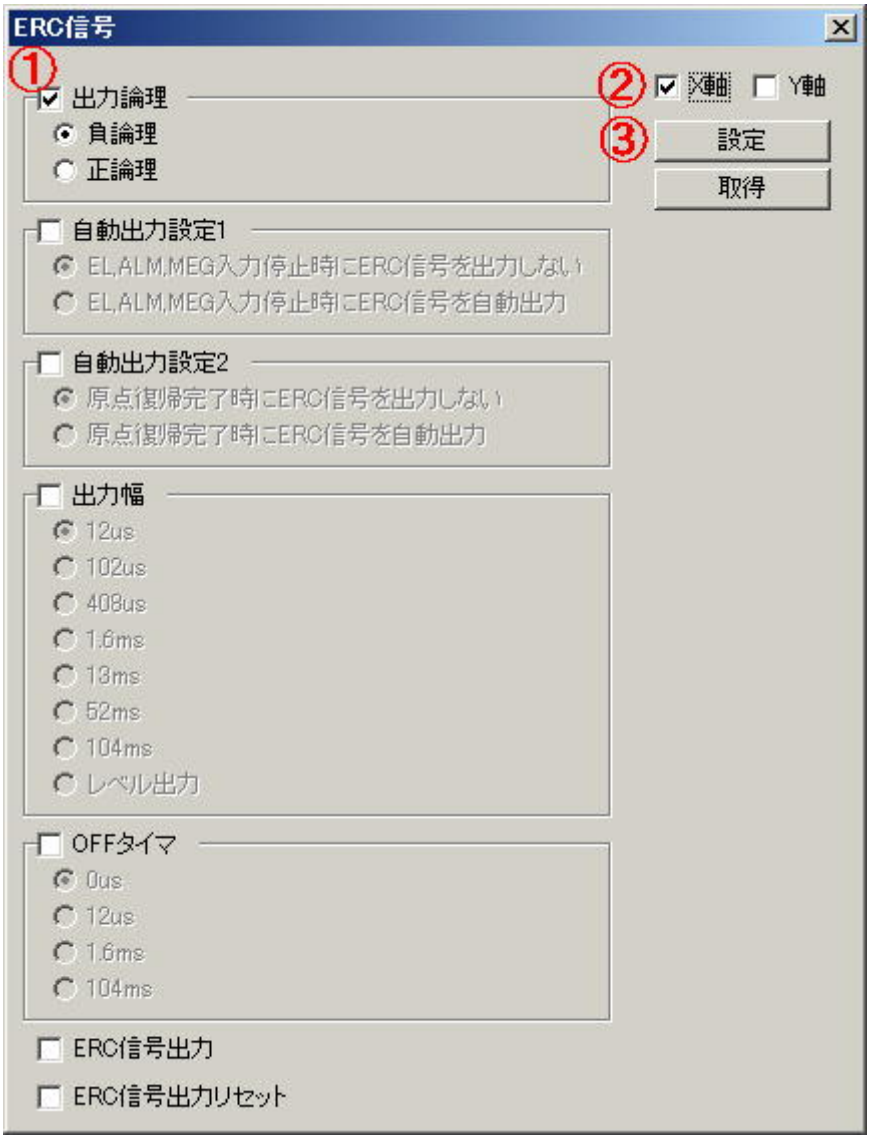
1. 設定を取得する項目をチェック状態にしてください（複数の取得項目を選択可能です）
2. 設定を取得する軸をチェック状態にしてください（複数の軸は選択できません）
3. 取得ボタンをクリックしてください。正常終了と表示されると取得完了です

ERC信号の設定・取得

メニューの[各種設定]－[ERC信号]をクリックするとERC信号の設定・取得画面が表示されます。

設定

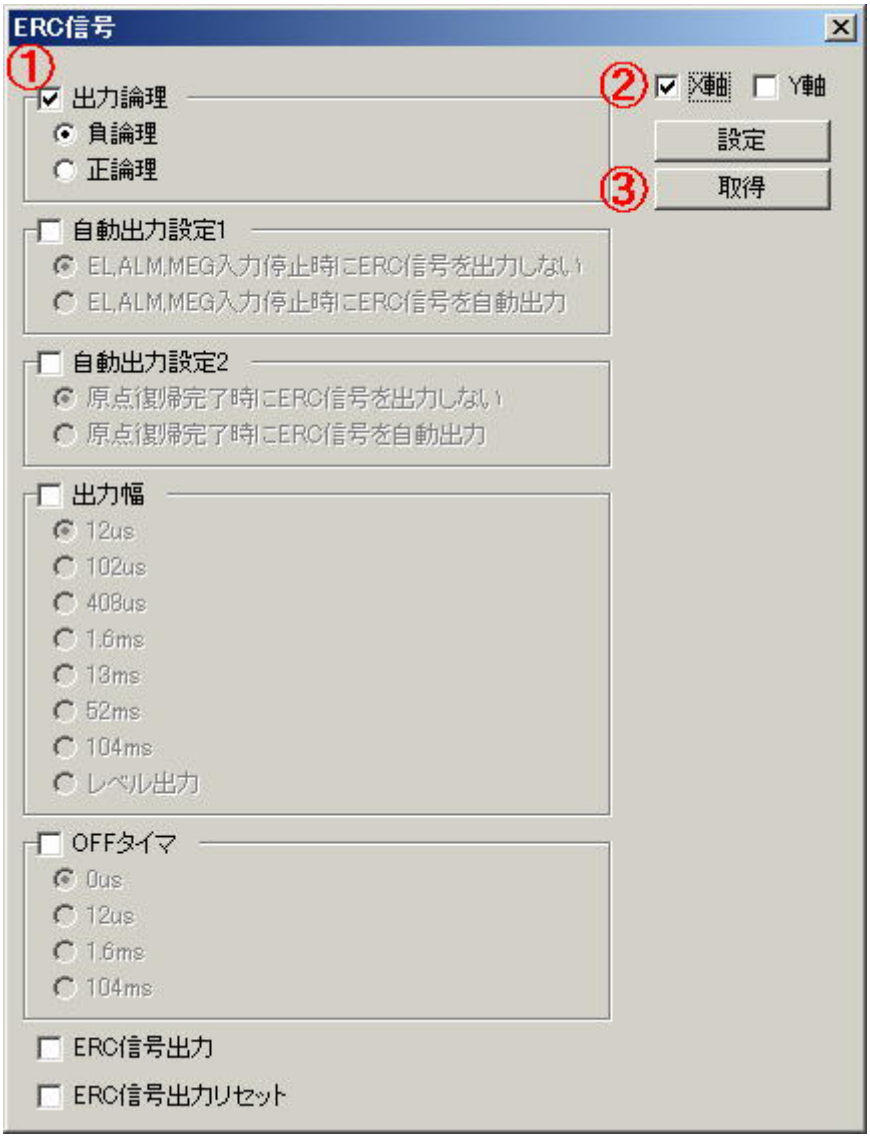
ERC信号の設定をおこないます。
設定の詳細についてはPmcmSetErcConfig関数の説明を参照してください。



1. 設定をおこなう項目をチェック状態にし、設定内容を選択してください（複数の設定項目を選択可能です）
2. 設定をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
3. 設定ボタンをクリックしてください。正常終了と表示されると設定完了です

取得

ERC信号設定の取得をおこないます。
設定の詳細についてはPmcmGetErcConfig関数の説明を参照してください。



1. 設定を取得する項目をチェック状態にしてください（複数の取得項目を選択可能です）
2. 設定を取得する軸をチェック状態にしてください（複数の軸は選択できません）
3. 取得ボタンをクリックしてください。正常終了と表示されると取得完了です

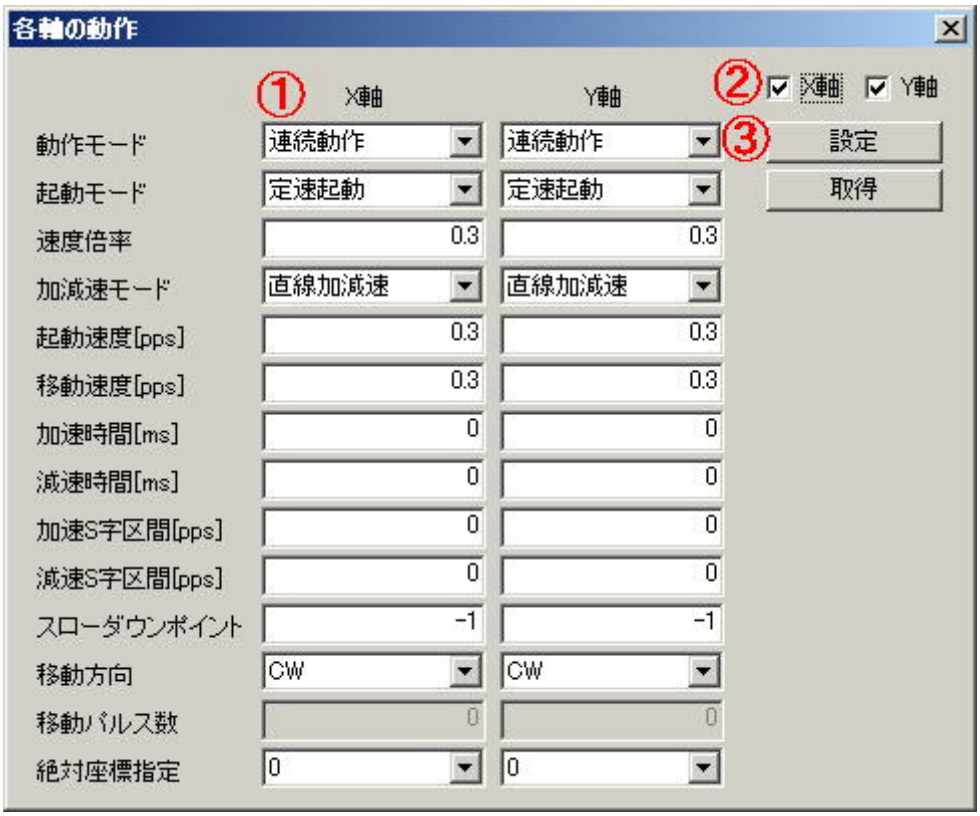
各軸の動作の設定・取得

メニューの[動作パラメータ設定]－[各軸の動作]をクリックすると各軸の動作の設定・取得画面が表示されます。

> ⚠ 動作パラメータに初期設定値はありませんので、動作開始前に動作パラメータを設定してください

設定

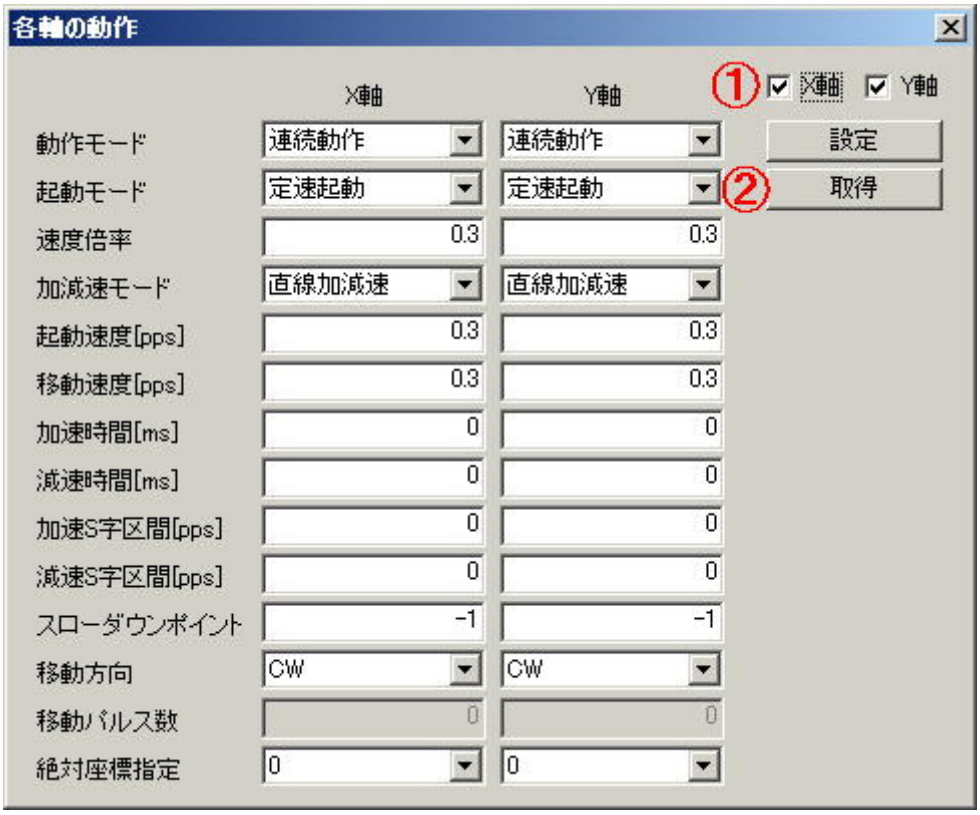
各軸の動作の設定をおこないます。
設定の詳細についてはPmcmSetMotion関数の説明を参照してください。



1. 各軸の動作パラメータの設定をしてください
   （設定をおこなわない軸の設定値も設定範囲内の数値にしてください）
2. 設定をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
3. 設定ボタンをクリックしてください。正常終了と表示されると設定完了です

取得

各軸の動作設定の取得をおこないます。
設定の詳細についてはPmcmGetMotion関数の説明を参照してください。



1. 設定を取得する軸をチェック状態にしてください（複数の軸を選択可能です）
2. 取得ボタンをクリックしてください。正常終了と表示されると取得完了です

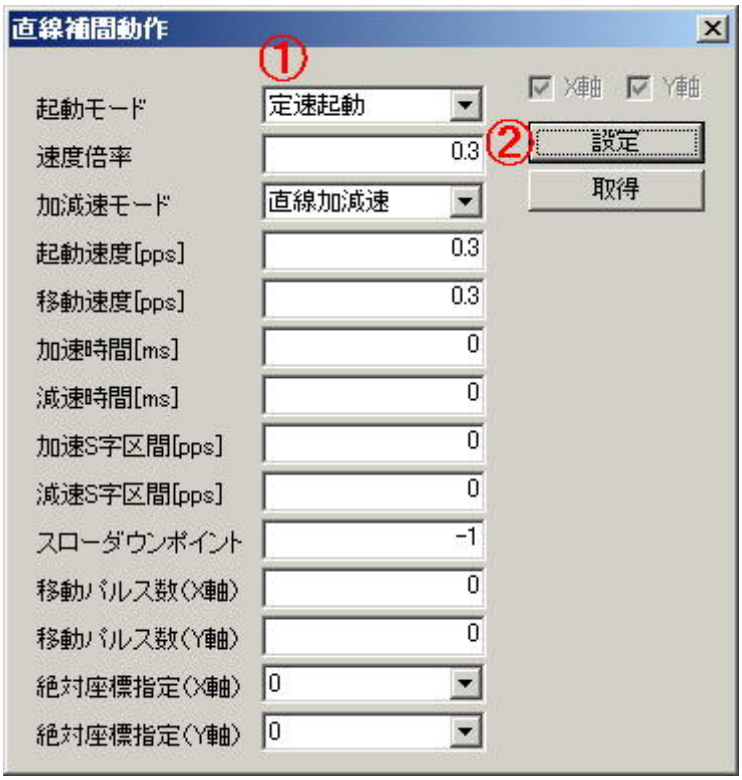
直線補間動作の設定・取得

メニューの[動作パラメータ設定]－[直線補間動作]をクリックすると直線補間動作の設定・取得画面が表示されます。

> ⚠ 動作パラメータに初期設定値はありませんので、動作開始前に動作パラメータを設定してください

設定

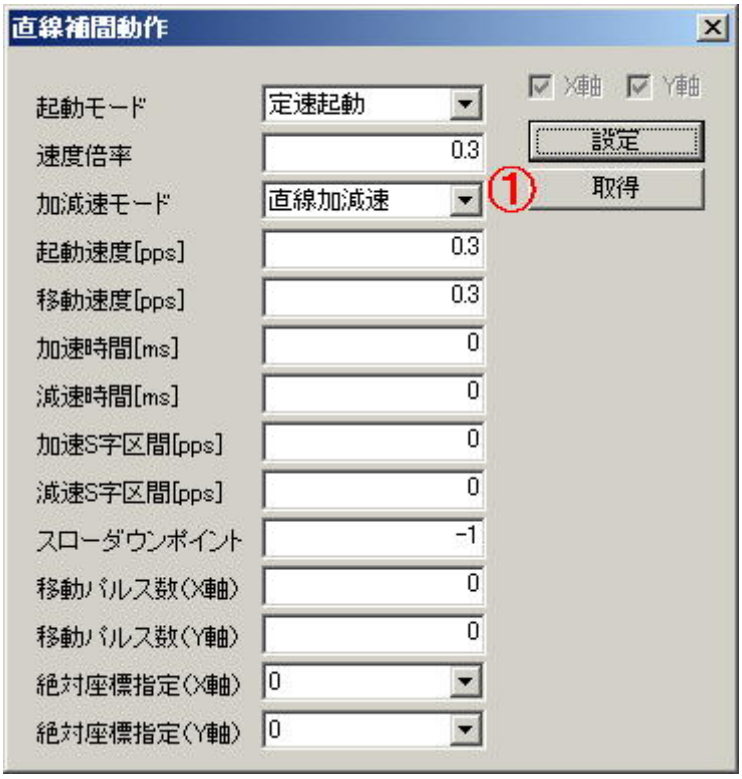
直線補間動作の設定をおこないます。
設定の詳細についてはPmcmSetMotionLine関数の説明を参照してください。



1. 動作パラメータの設定をしてください
2. 設定ボタンをクリックしてください。正常終了と表示されると設定完了です

取得

直線補間動作設定の取得をおこないます。
設定の詳細についてはPmcmGetMotionLine関数の説明を参照してください。



1. 取得ボタンをクリックしてください。正常終了と表示されると取得完了です

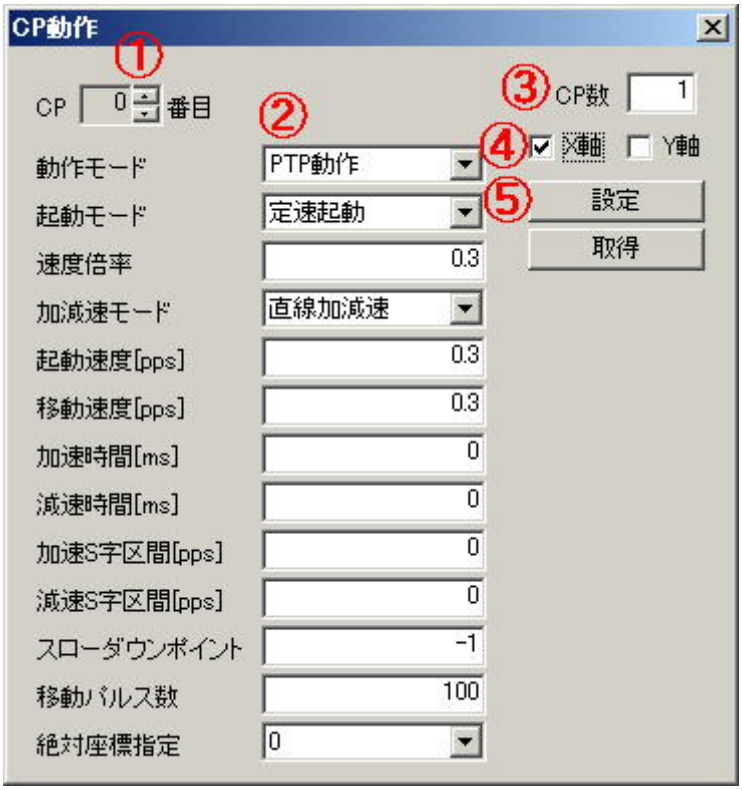
CP動作の設定・取得

メニューの[動作パラメータ設定]－[CP動作]をクリックするとCP動作の設定・取得画面が表示されます。

> ⚠ 動作パラメータに初期設定値はありませんので、動作開始前に動作パラメータを設定してください

設定

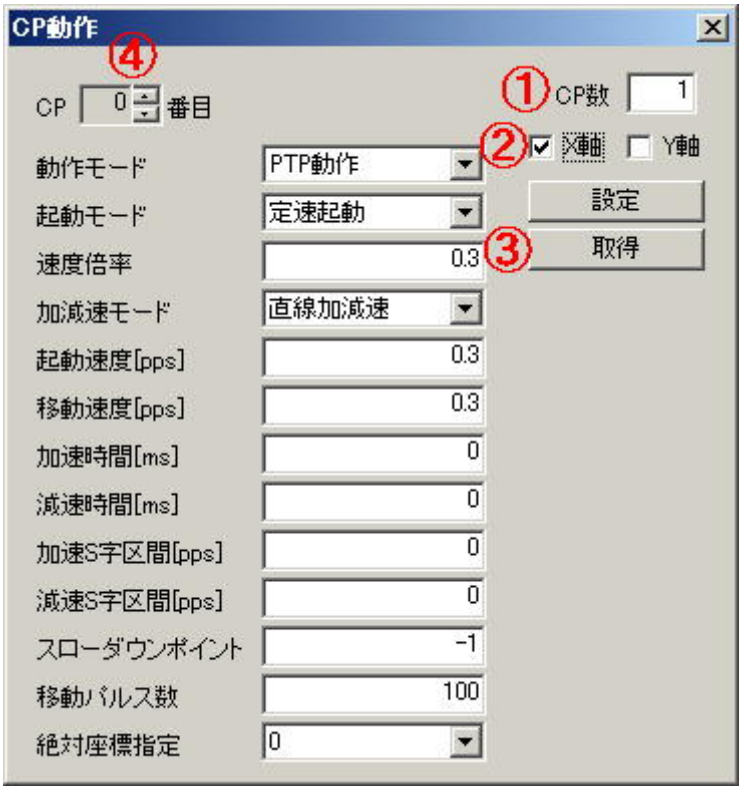
CP動作の設定をおこないます。
設定の詳細についてはPmcmSetMotionCp関数の説明を参照してください。



1. 動作パラメータを0番目から順に設定します
   5つのCP動作を設定する場合は0～4番目の動作パラメータを設定します。
2. 動作パラメータの設定をしてください
3. 動作パラメータ数を設定します
   1で0～4番目の動作パラメータを設定した場合は5を設定します。
4. 設定をおこなう軸をチェック状態にしてください（複数の軸は選択できません）
5. 設定ボタンをクリックしてください。正常終了と表示されると設定完了です

取得

CP動作設定の取得をおこないます。
設定の詳細についてはPmcmGetMotionCp関数の説明を参照してください。

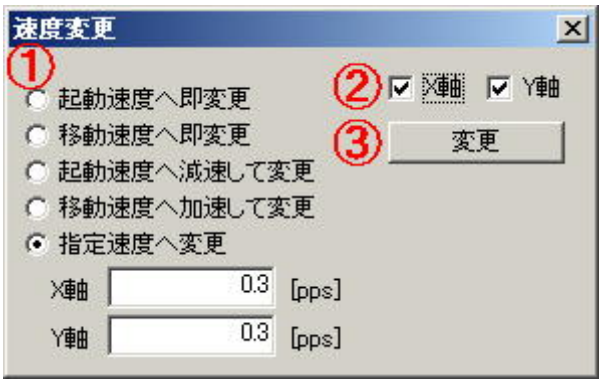


1. 設定を取得する動作パラメータ数を設定してください
   取得完了後には実際に取得した動作パラメータ数が表示されます。
2. 設定を取得する軸をチェック状態にしてください（複数の軸は選択できません）
3. 取得ボタンをクリックしてください。正常終了と表示されると取得完了です
4. 取得した動作パラメータは0番目から順に格納されています
   5つの動作パラメータを取得した場合は0～4番目に取得したデータが表示されます。

速度変更

メニューの[動作変更]－[速度変更]をクリックすると速度変更画面が表示されます。

速度の変更をおこないます。動作中のみ変更できます。
設定の詳細についてはPmcmChangeSpeed関数の説明を参照してください。



1. 変更内容を選択してください
   [指定速度へ変更]を選択した場合は変更する速度を入力してください（変更をおこなわない軸の速度も設定範囲内の値にしてください）。
2. 変更をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
3. 変更ボタンをクリックしてください

目標位置変更

メニューの[動作変更]－[目標位置変更]をクリックすると目標位置変更画面が表示されます。

目標位置の変更をおこないます。PTP動作中のみ変更できます。
設定の詳細についてはPmcmChangeStep関数の説明を参照してください。

1. 目標位置を入力してください（変更をおこなわない軸の目標位置も設定範囲内の値にしてください）
2. 変更をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
3. 変更ボタンをクリックしてください

カウンタ値設定

メニューの[カウンタ]−[カウンタ値設定]をクリックするとカウンタ値設定画面が表示されます。

カウンタ値の設定をおこないます。
設定の詳細についてはPmcmSetCounter関数（出力パルス）、PmcmSetEncoder関数（エンコーダ）の説明を参照してください。

1. 設定をおこなうカウンタを選択してください
2. 設定をおこなうカウンタ値を入力してください（設定をおこなわない軸も設定範囲内の値にしてください）
3. 設定をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
4. 設定ボタンをクリックしてください。正常終了と表示されると設定完了です

ラッチ値取得

メニューの[カウンタ]－[ラッチ値取得]をクリックするとラッチ値取得画面が表示されます。

ラッチ値の取得をおこないます。
設定の詳細についてはPmcmGetLatchCounter関数（出力パルス）、PmcmGetLatchEncoder関数（エンコーダ）の説明を参照してください。

1. 取得をおこなうカウンタを選択してください
2. 取得をおこなう軸をチェック状態にしてください（複数の軸を選択可能です）
3. 取得ボタンをクリックしてください。正常終了と表示されると取得完了です

注意事項

本製品及び本書の内容については、改良のために予告なく変更することがあります。

本製品を運用した結果の他への影響については、上記にかかわらず責任を負いかねますのでご了承ください。

本製品は人命にかかわる設備や機器、及び高度な信頼性を必要とする設備や機器としての使用またはこれらに組み込んでの使用は意図されておりません。
これら、設備や機器、制御システムなどに本製品を使用され、本製品の故障により人身事故、火災事故、損害などが生じても、弊社ではいかなる責任も負いかねます。
設備や機器、制御システムなどにおいて、安全設計に万全を期されるようご注意願います。

本製品は「外国為替及び外国貿易法」の規定により戦略物資等輸出規制製品に該当する場合があります。
国外に持ち出す際には、日本国政府の輸出許可申請などの手続きが必要になる場合があります。

本製品は日本国内仕様です。本製品を日本国外で使用された場合、弊社は一切の責任を負いかねます。また、弊社は本製品に関し、日本国外への技術サポート等をおこなっておりませんので、予めご了承ください。