TMBSCOM.DLL
Termi-BUS SIO Control Library
For Win32 PC Based Controller
機能説明書

Document No. DEE-00054H

Ver. 4.20 26th / Oct. / 2004

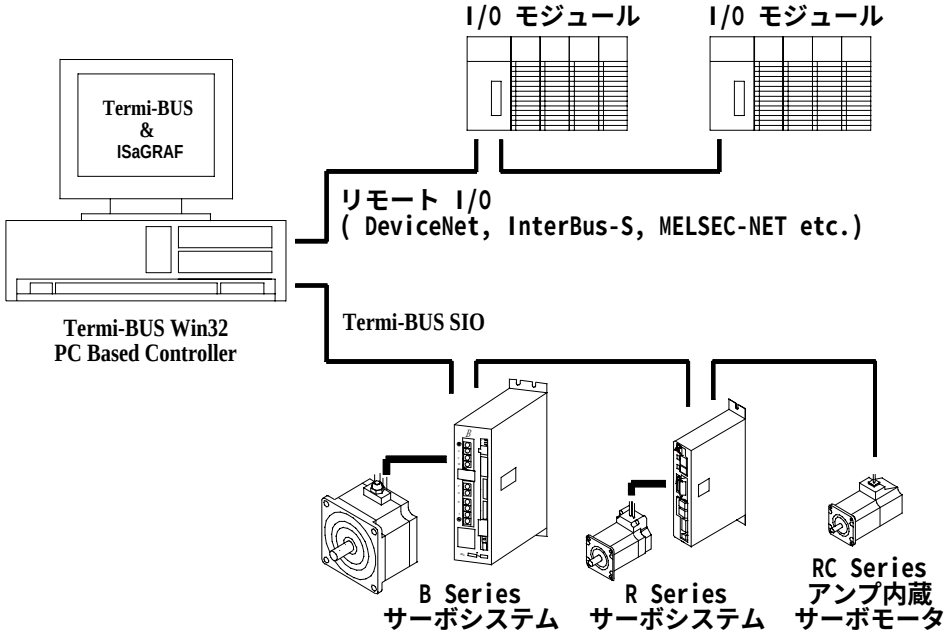
Dyadic Systems Co.,Ltd.

目　次

このドキュメントは、Version 4.20 以降のバージョンの TMBSCOM.DLL を対象に記載されています。

1. Termi-BUS for PC Based Controller

メカニカルシステムに用いられるアクチュエータが、従来からその種類によって異なるインターフェイスを有していることは、メカニカルシステムとそのコントローラの設計を徒に繁雑化し、結果として使用されるアクチュエータの種類限定と言う形で最適応用を疎外する大きな要因となっています。
Termi-BUS は、多様なアクチュエータを統一的に制御するためのシステムコントローラ～サーボアンプ間のインターフェイスで、種々のサーボアクチュエータを同一のデジタル情報を用いて制御することができるため、過剰性能を排した最適なアクチュエータ選択を実現します。
パーソナルコンピュータ上にＦＡコントローラを構築する PC Based Controller は、市場に大量に出回っているハードウェア／ソフトウェア資源を活用しながら柔軟性に富んだコントローラを短期間で安価に構築できる有力な方法であり、Termi-BUS SIO は、このPC Based Controller に効率的なサーボアクチュエータ制御機能を与えます。
Termi-BUS SIO Control Library を用いて Termi-BUS SIO 対応サーボシステムとソフトウェアＰＬＣを核とした PC Based Controller を容易に構築することができます。これを用いて構築されたコントローラは、実行環境としてパーソナルコンピュータの基本ハードウェア以外を必要としないため、ハードウェアコストを最小化できるとともに、タスク切り替え時間短縮のために特殊なリアルタイムオペレーティングシステムを使用したり、業界標準オペレーティングシステムに特殊なリアルタイム拡張を行う必要もないため、業界標準に適合した市販ハードウェアや市販ソフトウェアを最大限に活用でき、高い拡張性と可搬性を実現することができます。



2. Termi-BUS 対応サーボシステムと TMBSCOM.DLL

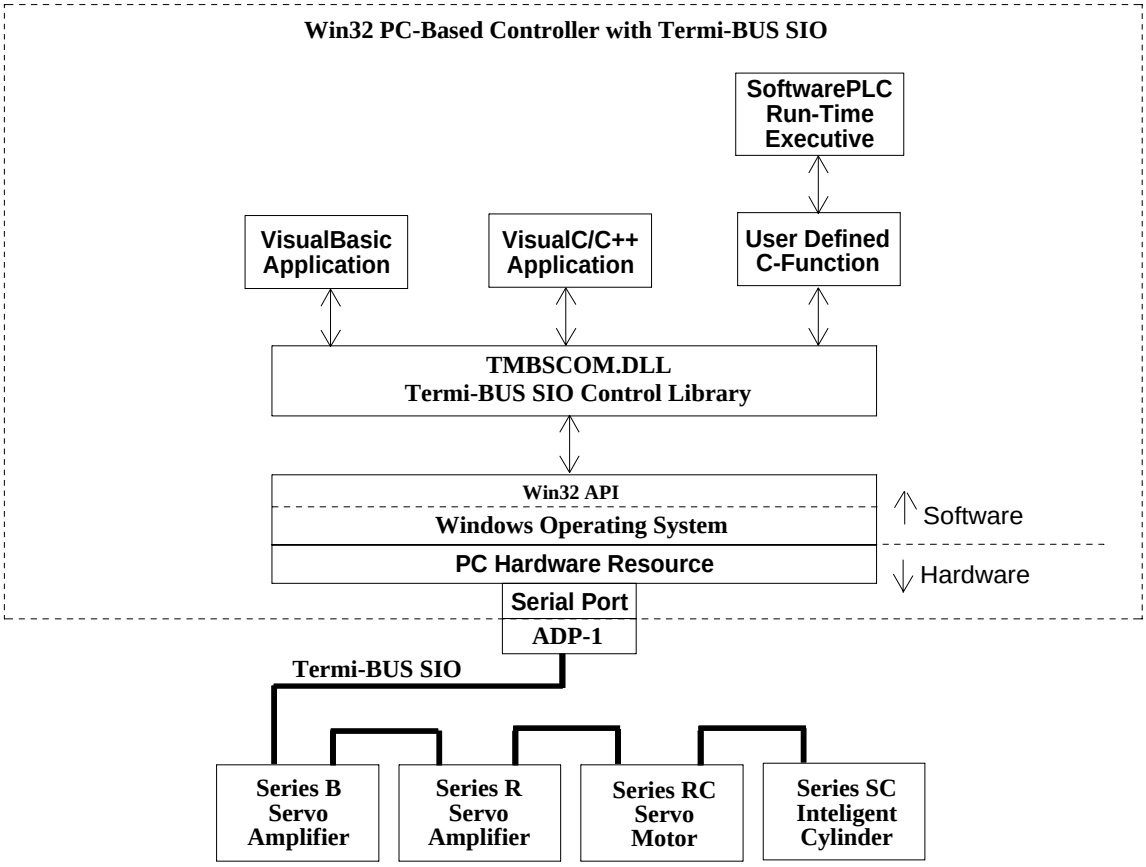
Termi-BUS 対応サーボアンプは、PTP（PointToPoint）制御時の動作プロフィル生成機能を内蔵しているため時間的に連続した位置変分指令を常時送り続ける従来方式のインターフェイスに比べて、上位コントローラとのインターフェイスのトラフィックが極端に小さくなっています。このため **Termi-BUS** を用いたシステムでは、サーボアクチュエータを制御するインターフェイスに専用設計ハードウェアを準備する必要がなく、コントローラのハードウェアコストを圧迫することがありません。**Termi-BUS** 対応サーボシステムにはパラレルI/O（DC24V系）によって構成される **Termi-BUS PIO** とシリアル通信による **Termi-BUS SIO**という二種類の上位コントローラインターフェイスが用意されています。**TMBSCOM.DLL**は、32ビット**Windows** 環境で動作するパーソナルコンピュータ上のアプリケーションプログラムから **Termi-BUS SIO** を介してこれらのサーボシステムを制御するためのライブラリソフトウェアであり、**Termi-BUS SIO** が持つ柔軟な機能をパーソナルコンピュータ上でのプログラミングが容易な形で提供致します。

２．１. Termi-BUS SIO の概要

Termi-BUS SIO は、サーボアクチュエータ制御専用インターフェイスとして、柔軟性に富んだ豊富なコマンド体系と効率的なプロトコルを持っており、リモートI/0やＰＬＣの I/0モジュールを介して軸コントローラに指令を与える場合に比べて、はるかに柔軟で高機能な指令を行うことができます。Termi-BUS SIO は、ＥＩＡ ＲＳ４８５に準拠した調歩同期式のシリアルバスインターフェイスで、ADP-1 ＲＳ２３２→ＲＳ４８５変換器を用いてパーソナルコンピュータの標準ＣＯＭ（シリアル）ポートに直結することができます。
Termi-BUS SIO では、モジュラケーブルによってサーボアンプ相互がディジーチェインに接続され、１つのＣＯＭポートで最大１６軸までの Termi-BUS 対応サーボシステムを制御することができます。Termi-BUS SIO インターフェイスは、Termi-BUS 対応サーボシステムに標準機能として実装されています。

２．２. TMBSCOM.DLL の概要

TMBSCOM.DLL は、Termi-BUS SIO で提供されるサーボ制御機能を３２ビットWindowsアプリケーションで有効に利用するためのダイナミックリンクライブラリで、位置決めや位置決め完了確認等の制御機能単位に対応した関数が用意されていますので、プログラマはこれを用いることにより通信プロトコルやハードウェアリソースの詳細な制御を一切考慮する必要がなくなります。
TMBSCOM.DLL は、通常のダイナミックリンクライブラリと同様に VisualC/C++ 又は、VisualBasic 等で作成される一般的な Windowsアプリケーションから自由に呼び出すことができ、また３２ビットWindows環境上で動作するソフトウェアPLCであれば、ユーザ定義C言語ファンクション等からこれを呼び出すことにより、高度なサーボ制御機能をこれらのPLCプログラム上で簡単に利用できるようになります。



3. TMBSCOM.DLL の特長

TMBSCOM.DLL の関数を用いて**Termi-BUS SIO** の全ての機能を利用することができます。これらの関数は、その内部でコマンドの送信からレスポンスの受信による送達確認、及び誤り回復までの伝送制御手順を自動的に実行するため、ユーザが **Termi-BUS SIO** の伝送制御手順を意識する必要はありません。通信確立手順実行関数によって、一旦 **Termi-BUS SIO** の通信が確立すると、伝送制御上の異常等を表示する通信状態変数と、通信確立時に認識された全軸に対応したサーボシステム動作状態モニタ変数がダイナミックリンクライブラリ内部に作成されます。これらの状態変数は、**Termi-BUS SIO** の通信実行の度に更新され、ユーザは、これらの状態変数の内容を取得する関数を使用して随時読み出すことができます。

3．1. VisualC/C++ アプリケーションからの呼び出し

TMBSCOM.DLL は、関数プロトタイプが宣言されているヘッダファイル **TBUSSIO.H**、及び各関数のエントリをアプリケーションにリンケージするための **TMBSCOM.LIB** ファイルとともに提供されます。ユーザは、これらのファイルを用いて任意の **VisualC/C++**アプリケーションから **TMBSCOM.DLL** の各関数を自由に呼び出すことができます。
TBUSSIO.H ではこれらの関数は全て WINAPI として宣言されていますので、ユーザはこれらを通常の **Win32API** 関数と同じ呼び出し規約で呼び出すことができます。また**TBUSSIO.H** には各関数からの返り値、及びフラグとして使用される変数から各ビット情報を抽出するための定数が定義されており、ユーザはこれらを自分のプログラムの中で自由に使用することができます。

3．2. VisualBasic アプリケーションからの呼び出し

TMBSCOM.DLL は、関数がパブリックのプロシジャーとして宣言されている **TBUSSIO.BAS**、ファイルとともに提供されます。ユーザは、このファイルを標準モジュールとしてプロジェクトに挿入することによって、任意の **VisualBasic**アプリケーションから **TMBSCOM.DLL** の各関数を自由に呼び出すことができます。**TBUSSIO.BAS** には各関数からの返り値、及びフラグとして使用される変数から各ビット情報を抽出するための定数もパブリック定義されており、ユーザはこれらを自分のプログラムの中で自由に使用することができます。

TMBSCOM.DLL は、RS-232C/RS-485 変換器として **ADP-1** を使用することを前提としております。従いまして **ADP-1** 以外の変換器を使用した場合は正常に動作致しませんのでご注意下さい。

4. TMBSCOM.DLL 関数リファレンス

4．1. 既定義の各関数共通定数

以下に示す **TMBSCOM.DLL** で使用される定数は、**TMBSCOM.H** の中で既定義であり、このファイルをインクルードしているモジュールの中では自由に使用することができます。

```
int WINAPI get_axes( unsigned short *axes );
int WINAPI get_sio_error( void );
int WINAPI get_tmbs_state( void );
int WINAPI get_current_baud( void );
int WINAPI get_com_errlog( void );
```

上記の関数を除く、**TMBSCOM.DLL** のすべての関数は次の返り値を返します。

```
SIO_DONE        1       // コマンド正常終了、又は条件成立
SIO_ERROR       0       // コマンドエラー、又は条件非成立
```

４．２. 通信確立手順実行及び通信状態確認関数

４．２．１． init_tmbs_config 関数

```
int WINAPI init_tmbs_config(
    LPCTSTR port,      // 通信ポートファイル名 ( "COM1","COM2"etc. )
    int baud,          // ボーレイト指定コード
    int nrt,           // コマンド再送最大回数
    BOOL reset,        // TRUE で再送回数オーバのとき通信初期化に戻る
    BOOL automatic,    // TRUE でボーレイト自動設定
    int *axes_info     // 軸構成指定情報
);
```

init_tmbs_config 関数によって **Termi-BUS SIO** の通信確立手順と接続軸の検出又は確認を自動的に行うことができます。

入力パラメータ

port:

Termi-BUS SIO として使用するシリアル通信ポート名が入った文字列へのポインタ。
この文字列は、シリアル通信ポート名として有効なファイル名でなければなりません。

baud:

Termi-BUS SIO で使用するシリアル通信のボーレイトを指定します。
このパラメータは、以下の **TBUSSIO.H** で既定義の定数のいずれかでなければなりません。

```
TMBS_BAUD_9600      0x04    //   9600 bps
TMBS_BAUD_19200     0x05    //  19200 bps
TMBS_BAUD_38400     0x06    //  38400 bps
TMBS_BAUD_14400     0x11    //  14400 bps
TMBS_BAUD_57600     0x13    //  57600 bps
TMBS_BAUD_115200    0x14    // 115200 bps
```

パラメータ automatic に TRUE が指定されている場合は、このパラメータの設定値jは無視され上記の選択肢のうちで、御使用のコンピュータで使用可能な最高速のものが選択されます。

nrt:

通信確立後に通信エラーが生じた時の再送による回復手順の最大再送回数を指定します。
このパラメータに指定された回数以内の再送によって誤りが回復された場合は、通信異常とはなりません。0を指定すると誤り回復手順を行いません。

reset:

TRUE が指定されると、誤り回復手順によっても回復できない通信異常が生じた場合に、自動的に通信初期化手順を再度実行します。この様な状態は、アプリケーションによって管理される必要がありますので、通常このパラメータには FALSE を設定します。

aurtomatic:

TRUE が指定されると、パラメータ baud の指定に拘わりなく御使用のコンピュータで使用可能な最大のボーレイトが選択されます。FALSE が指定されている場合は、ボーレイトはパラメータ baud の指定に従って設定されます。

axes_info:

軸構成情報を指定する 16 要素の int 配列へのポインタ。
この配列のインデックスは軸番号に相当し、各要素の値が 0 で対応する軸が存在し、-1 は対応する軸が存在しないことを示します。配列要素のうちで1つでも -1 でない要素が存在する場合は、通信初期化手順において-1 でない値が設定されている軸に対してのみ、存在確認を行います。
例えば#0 軸、#2 軸、#3 軸のみが存在するシステムにおいては、このパラメータが指す配列要素の設定値は下記のようにします。

```
axes_info[0] = 0;
axes_info[1] = -1;
axes_info[2] = 0;
axes_info[3] = 0;
axes_info[4] = -1;
axes_info[5] = -1;
axes_info[6] = -1;
axes_info[7] = -1;
axes_info[8] = -1;
axes_info[9] = -1;
axes_info[10] = -1;
axes_info[11] = -1;
axes_info[12] = -1;
axes_info[13] = -1;
axes_info[14] = -1;
axes_info[15] = -1;
```

-1 以外の値が設定されている配列要素に対応する軸が存在しなかった場合は、通信初期化エラーとなります。この配列の全ての要素の設定値が-1 であった場合は、通信確立手順において#0 軸から#15 軸までの全軸に対して存在確認を行い、存在が確認された軸に対応する配列要素の値を 0 に書き換えます。従ってこの場合は、通信確立後に各配列要素の値を調べることにより、現実のシステムの軸構成情報を取得することができます。1つの軸の存在も確認できなかった場合は、通信初期化エラーとなります。

返り値

正常に通信確立の状態になると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。SIO_ERROR が返される状態は必ずしも通信初期化エラーではなく、通信初期化手順実行中にも SIO_ERROR が返されます。

4．2．2． init_tmbs 関数

int WINAPI init_tmbs(void);

init_tmbs 関数の機能は、基本的に init_tmbs_config 関数の機能と同じですが init_tmbs_config 関数の各パラメータに対応する設定値は全て **Windows** のディレクトリ（**Windows95/98** の場合は¥Windows、**WindowsNT** の場合は¥WinNT）の **TBUSSIO.INI** ファイルによって定義されます。
以下に init_tmbs_config 関数のパラメータ port="COM1"、baud= TMBS_BAUD_115200、nrt=2、reset=FALSE、automatic=TRUE、axes_info=#0 軸、#1 軸、#2 軸、#3 軸存在指定の場合に相当する **TBUSSIO.INI** ファイルの定義は下記の様になります。

```
[SYSTEM]
PORT=COM1
BRSL=14
NRT=2
RESET=0
AUTOMATIC=1
AXIS=00010203
```

AXIS は、軸番号を1軸当たり10進数二桁で指定し、axes_info の要素が全て-1 に設定されている場合に相当する全軸ポーリングによる軸構成情報取得の指定はできません。

返り値

正常に通信確立の状態になると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。SIO_ERROR が返される状態は必ずしも通信初期化エラーではなく、通信初期化手順実行中にも SIO_ERROR が返されます。

4．2．3． close_tmbs 関数

int WINAPI close_tmbs(void);

この関数は、TMBSCOM.DLL が作成した全ての資源を閉じて DLL の実行を終了します。
TMBSCOM.DLL は、お客様のアプリケーションプログラムの開始に伴うロード時に、スレッドや同期オブジェクト等の専用の資源を作成／使用しますが、お客様のアプリケーションプログラム終了時に、DLL に対するプロセス終了イベントが来る前に呼び出し側のプロセスが見かけ上終了してしまうことがあり、この場合、これらの専用資源の解放ができなくなって、プログラム終了後もお客様のアプリケーションプログラムのプロセスの一部がメモリ空間上に取り残されてしまうことがあります。この問題を避けるため、
お客様のアプリケーションプログラムの終了時に、必ずこの関数を呼び出して、TMBSCOM.DLL が作成した全ての資源を閉じるようにして下さい。

4．2．4． reopen_tmbs 関数

int WINAPI reopen_tmbs(void);

この関数は、close_tmbs 関数によって閉じられた DLL の内部資源を再び作成し、通信処理を再開できるようにします。お客様のアプリケーションプログラムの中で close_tmbs と reopen_tmbs を順番に呼び出すことにより、通信の再初期化を行うことができ、通信異常等の復旧を行うことができます。

4．2．5． init_sio_tbus 関数

この関数は、旧バージョンとの互換性のために残されています。
init_tmbs_config 関数を御使用下さい。

4．2．6． init_sio 関数

この関数は、旧バージョンとの互換性のために残されています。
init_tmbs 関数を御使用下さい。

4．2．7． get_tmbs_state 関数

int WINAPI get_tmbs_state(void);

get_tmbs_state 関数によって **Termi-BUS SIO** の現在の通信の状態を取得することができます。

返り値

この関数からは下記の **TBUSSIO.H** で既定義の定数のいずれかの値が返されます。

TMBS_NO_EXIST	0	// 通信処理は開始されていません
TMBS_INITIAL	1	// 通信初期化要求待ち状態
TMBS_INIT_ERROR	2	// 通信初期化エラー状態
TMBS_OPENING	3	// 通信初期化実行状態
TMBS_RUNNING	4	// 通信確立状態

4．2．8． get_current_baud 関数

int WINAPI get_current_baud(void);

get_current_baud 関数によって実際に使用されているボーレイト設定値を取得することができます。

返り値

この関数からは下記の **TBUSSIO.H** で既定義の定数のいずれかの値が返されます。

TMBS_BAUD_9600	0x04	// 9600 bps
TMBS_BAUD_19200	0x05	// 19200 bps
TMBS_BAUD_38400	0x06	// 38400 bps
TMBS_BAUD_14400	0x11	// 14400 bps
TMBS_BAUD_57600	0x13	// 57600 bps
TMBS_BAUD_115200	0x14	// 115200 bps

４．２．９． get_sio_error 関数

int WINAPI get_sio_error(void);

get_sio_error 関数によってこの関数の呼び出し前に起こった最も新しい通信エラーコードを取得することができます。

返り値

この関数からは下記の **TBUSSIO.H** で既定義の定数のいずれかの値が返されます。

SIO_COMUSED	-1 // COM ポートがすでに使われています
SIO_TIMEOUT	-2 // 誤り回復の再送回数がオーバしました
SIO_NOINIT	-3 // 通信初期化前に通信を行う指令が実行されました
SIO_INVALID_PARAM	-5 // 正しくないパラメータが渡されました
SIO_NOTSUPORT_TO	-6 // 通信タイムアウトがサポートされていません
SIO_NOTSUPORT_BAUD	-8 // サポートされていないボーレイトが指定されました
SIO_NOTSUPORT_PARA	-9 // 通信パラメータ変更がサポートされていません
SIO_NO_CONFIGFILE	-10 // TBUSSIO.INI 環境設定ファイルが見つかりません
SIO_COMFAILED	-12 // シリアル通信ポートのオープンに失敗しました

４．２．１０．通信確立の実際と注意事項

通信確立手順実行関数（init_tmbs_config 関数、init_tmbs 関数、init_sio_tbus 関数、init_sio 関数）を用いて Termi-BUS SIO の通信が確立するまで（通信確立手順実行関数が返り値として SIO_DONE を返すまで）は、通信確立手順実行関数と get_tmbs_state 関数以外の TMBSCOM.DLL 中の他の関数を呼び出してはなりません。
下記に通信確立手順実行関数及び通信状態確認関数を用いた実際の通信確立実行プログラムの C 言語でのイメージを示します。

```
while ( init_tmbs( ) != SIO_DONE )
{
    Sleep( 5 );   // Anothor thread may need the time slice.

    if ( get_tmbs_state( ) == TMBS_INIT_ERROR )
    {
        // Hard Luck!! some axes may not exist.
        break;
    }
}

if ( get_tmbs_state( ) == TMBS_RUNNING )
    // OK!! the communication is established.
```

４．３．モータ移動動作指令関数と動作完了確認関数

４．３．１． move_point（ポイント番号指定間接 PTP 動作指令）関数

```
int WINAPI move_point(
    int axis,        // 実行対象軸番号
    int point        // PTP動作指令ポイントデータ番号
);
```

サーボアンプ／モータ内部の不揮発性メモリ領域に記憶されているPTP動作指令データを実行領域にロードし、PTP動作を実行します。各ポイントのPTP動作指令データは、速度、加速度等の動作プロファイルパラメータを含めてすべて実行領域に一括ロードされますので、これらのデータは、予め **TBVST** 又は **CTA** によって不揮発性メモリ領域に全て書き込んでおく必要があります。
R、RC、SCシリーズで指令可能な突き当て動作はこの関数を用いてのみ指令可能です。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．３．２． hmove_point（ポイント番号指定間接 PTP 動作バッファリング指令）関数

```
int WINAPI hmove_point(
    int axis,        // 実行対象軸番号
    int point        // PTP動作指令ポイントデータ番号
);
```

多軸のシステムでの move_point 関数による指令において、複数回のコマンド発行に伴うオーバヘッドが問題となる場合、各軸に対して、このコマンドによって、**前回の軸動作実行中に事前に指令しておいて、これを１回の** start_preload_command **関数の実行によって各軸同時に起動するバッファリング指令を行うことができます。**

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．３．３．move_abs（絶対位置指定 PTP 動作指令）関数

```
int WINAPI move_abs(
    int axis,        // 軸番号
    long position    // 絶対座標系での目標位置
);
```

絶対位置座標系での目標位置 position に位置決めを行います。この関数の実行時の速度／加速度等の動作プロフィルパラメータは、実行領域に残っている値がそのまま使用されます。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．３．４．hmove_abs（絶対位置指定 PTP 動作バッファリング指令）関数

```
int WINAPI hmove_abs(
    int axis,        // 軸番号
    long position    // 絶対座標系での目標位置
);
```

多軸のシステムでの move_abs 関数による指令において、複数回のコマンド発行に伴うオーバヘッドが問題となる場合、各軸に対して、このコマンドによって、**前回の軸動作実行中に事前に指令しておいて、これを１回の** start_preload_command **関数の実行によって各軸同時に起動するバッファリング指令を行うことができます。**

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．３．５．move_inc（相対移動量指定 PTP 動作指令）関数

```
int WINAPI move_inc(
    int axis,        // 軸番号
    long distance    // 相対移動量
);
```

現在位置に移動量 distance を加算した位置を目標位置として位置決めを行います。この関数の実行時の速度／加速度等の動作プロフィルパラメータは、実行領域に残っている値がそのまま使用されます。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．３．６．hmove_inc（相対移動量指定 PTP 動作指令）関数

```
int WINAPI hmove_inc(
    int axis,        // 軸番号
    long distance    // 相対移動量
);
```

多軸のシステムでの move_inc 関数による指令において、複数回のコマンド発行に伴うオーバヘッドが問題となる場合、各軸に対して、このコマンドによって、**前回の軸動作実行中に事前に指令しておいて、これを１回の** start_preload_command **関数の実行によって各軸同時に起動するバッファリング指令を行うことができます。**

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．３．７．move_org（原点復帰動作指令）関数

```
void move_org(
    int axis        // 実行対象軸番号
    int mode        // 原点復帰動作のパターン指定コード
);
```

パラメータ mode に基づいて指定されたパターンの原点復帰動作を行います。modeで指定される原点復帰動作パターンは下記となります。

mode=00H

サーボモータは移動せず、その位置の座標値を０にします。
アブソリュートエンコーダではこのパターンがデフォルト値ですが、他のパターン選択コードに設定し直した場合は、そのパターンでの原点復帰動作を実行します。

mode=01H**（エンコーダ2chタイプのRCシリーズでは選択不可）**

サーボモータは正転方向に低速で移動しエンコーダマーカ（Cch）位置で停止、その位置で座標値を０にします。

mode=02H**（エンコーダ2chタイプのRCシリーズでは選択不可）**

サーボモータは逆転方向に低速で移動しエンコーダマーカ（Cch）位置で停止、その位置で座標値を０にします。

mode=03H**（ＲＣシリーズでは選択不可）**

サーボモータは図１に示す順序で正転方向のストロークリミット信号（Termi-BUS PIO ＊ＩＮＨ＋）のエッジを検出した後、逆転方向に低速で移動しエンコーダマーカ（Cch）位置で停止、その位置で座標値を０にします。

mode=04H**（ＲＣシリーズでは選択不可）**

サーボモータは図６に示す順序で逆転方向のストロークリミット信号（Termi-BUS PIO ＊ＩＮＨ－）のエッジを検出した後、正転方向に低速で移動しエンコーダマーカ（Cch）位置で停止、その位置で座標値を０にします。

mode=05H**（ＲＣシリーズでは選択不可）**

サーボモータは図５に示す順序で正転方向のストロークリミット信号（Termi-BUS PIO ＊ＩＮＨ＋）のエッジを検出した位置で停止、その位置で座標値を０にします。

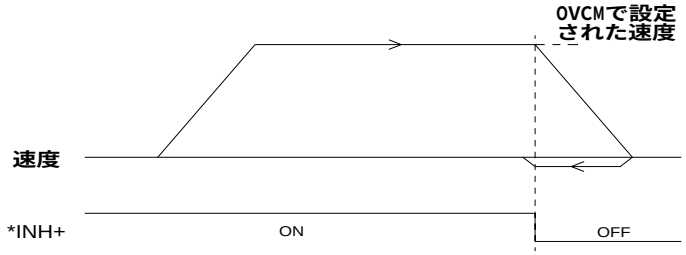


図１．正転方向＊ＩＮＨ＋エッジ検出シーケンス

mode=06H（ＲＣシリーズでは選択不可）

サーボモータは図６に示す順序で逆転方向のストロークリミット信号（Termi-BUS PIO ＊ＩＮＨ－）のエッジを検出した位置で停止、その位置で座標値を０にします。

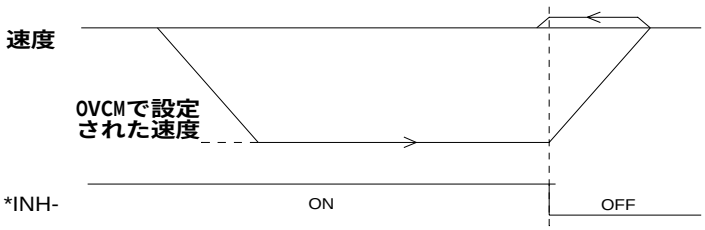


図２．逆転方向＊ＩＮＨ－エッジ検出シーケンス

mode=07H（Ｒ／ＲＣシリーズでのみ選択可）

サーボモータをパラメータＯＤＰＷに設定された値に電流制限された状態として、パラメータＯＶＣＭで指定された速度で正転方向に移動してメカニカルストッパに突き当て、その位置からエンコーダフィードバック２カウント分戻った位置で座標値を０にします。動作開始からパラメータＯＴＩＭに設定された時間経過しても突き当て状態を検出できなかったときには、アラーム（ BEH ）となります。

mode=08H（Ｒ／ＲＣシリーズでのみ選択可）

サーボモータをパラメータＯＤＰＷで設定された値に電流制限された状態として、パラメータＯＶＣＭで指定された速度で逆転方向に移動してメカニカルストッパに突き当て、その位置からエンコーダフィードバック２カウント分戻った位置で座標値を０にします。動作開始からパラメータＯＴＩＭに設定された時間経過しても突き当て状態を検出できなかったときには、アラーム（ BEH ）となります。

mode=09H（Ｒ／ＲＣシリーズでのみ選択可）

サーボモータをパラメータＯＤＰＷに設定された値に電流制限された状態として、パラメータＯＶＣＭで指定された速度で正転方向に移動してメカニカルストッパに突き当て、その位置からサーボモータは逆転方向に低速で移動しエンコーダマーカ（Cch）位置で停止、その位置で座標値を０にします。動作開始からパラメータＯＴＩＭに設定された時間経過しても突き当て状態を検出できなかったときには、アラーム（ BEH ）となります。

mode=0AH（Ｒ／ＲＣシリーズでのみ選択可）

サーボモータをパラメータＯＤＰＷで設定された値に電流制限された状態として、パラメータＯＶＣＭで指定された速度で逆転方向に移動してメカニカルストッパに突き当て、その位置からサーボモータは正転方向に低速で移動しエンコーダマーカ（Cch）位置で停止、その位置で座標値を０にします。動作開始からパラメータＯＴＩＭに設定された時間経過しても突き当て状態を検出できなかったときには、アラーム（ BEH ）となります。

ODPW、OVCM、OTIMは、サーボアンプ／モータ内部の不揮発性メモリ領域上の共通パラメータで、これらのデータは、予め **TBVST** 又は **CTA** によって不揮発性メモリ領域に書き込んでおく必要があります。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．3．8．hmove_org（原点復帰動作指令）関数

```
void hmove_org(
    int axis        // 実行対象軸番号
    int mode        // 原点復帰動作のパターン指定コード
);
```

多軸のシステムでの move_org 関数による指令において、複数回のコマンド発行に伴うオーバヘッドが問題となる場合、各軸に対して、このコマンドによって、**前回の軸動作実行中に事前に指令しておいて、これを１回の** start_preload_command **関数の実行によって各軸同時に起動するバッファリング指令を行うことができます。**

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．3．9．move_rotate（無限回転動作指令）関数

```
int WINAPI move_rotate(
    int axis,         // 実行対象軸軸番号
    int dir,          // 回転方向 0=正転, 1=逆転
    int vcmd,         // 速度指令値
    int acmd          // 加速度指令値
);
```

この関数は、対象軸を指令した速度／加速度で指令した方向に回転させたままにします。主軸等の回転指令に用います。この関数は、実行データ領域の動作プロファイルパラメータを vcmd 及びacmd の値で書き換えます。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．3．10．move_jog（JOG移動動作指令）関数

```
int WINAPI move_jog(
    int axis,         // 実行対象軸番号
    long distance     // JOG指令繰り返し周期当たりの移動量
);
```

サーボアンプ／モータに対してJOG動作の実行を指令します。JOG指令の移動速度は、内部で下記の計算に基づいて自動的に決定されます。

((distance * 60) / (delta_T * motor_1rev)) [r/min]

但し、 delta_T　　:JOG移動指令の前回受信と今回受信の間隔 [sec]
　　motor_1rev :モータ軸1回転当たりの移動量

上記の delta_T の最大値は、0.1 [sec] であり連続したJOG移動を行うためには、この値より短い周期でJOG移動指令関数を実行する必要があります。また移動量指令値distance は、対象となるサーボシステムの最大回転数において 0.1 [sec] の時間のなかで移動できる値を上限とします。従って最後のJOG動作指令関数の実行後 0.1 秒以内にモータは停止します。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．3．11．hmove_jog（JOG移動動作指令）関数

```
int WINAPI hmove_jog(
    int axis,        // 実行対象軸番号
    long distance    // JOG指令繰り返し周期当たりの移動量
);
```

多軸のシステムでの move_jog 関数による指令において、複数回のコマンド発行に伴うオーバヘッドが問題となる場合、各軸に対して、このコマンドによって、**前回の軸動作実行中に事前に指令しておいて、これを１回の** start_preload_command **関数の実行によって各軸同時に起動するバッファリング指令を行うことができます。**

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．3．12．check_pfin（位置決め完了状態取得）関数

```
int WINAPI check_pfin(
    int axis        // 対象軸番号
);
```

対象軸の位置決め完了状態（**Termi-BUS PIOのPFIN**)を取得します。この関数は、ダイナミックリンクライブラリ内のサーボシステム動作状態モニタ変数の内容で完了状態を判別しますので、繰り返しチェックの場合はcheck_status関数を併用する必要が有ります。

返り値

位置決め完了状態であれば SIO_DONE が返り、それ以外では SIO_ERROR が返ります。

4．3．13．check_status（サーボシステム内部状態ポーリング）関数

```
int WINAPI check_status(
    int axis        // 対象軸番号
);
```

対象軸の内部状態を取得し、ダイナミックリンクライブラリ内のサーボシステム動作状態モニタ変数に格納します。この関数は、サーボアンプ／モータの内部状態を取得するためのもので、何の実動作も指令しません。

返り値

対象軸から正常に内部状態が取得できると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．3．14．check_status（サーボシステム内部状態ポーリング）関数

```
int WINAPI check_status(
    int axis        // 対象軸番号
);
```

対象軸の内部状態を取得し、ダイナミックリンクライブラリ内のサーボシステム動作状態モニタ変数に格納します。この関数は、サーボアンプ／モータの内部状態を取得するためのもので、何の実動作も指令しません。

返り値

対象軸から正常に内部状態が取得できると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．3．15．follow_position（現在位置フォローアップ／即時停止）関数

```
int WINAPI follow_position(
    int axis        // 対象軸番号
);
```

絶対座標系での目標位置を関数実行時の現在位置の値にフォローアップします。対象軸が移動中であった場合は、即時停止の後に上記の位置に位置決めします。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．3．16．start_preload_command（バッファリングコマンド実行）関数

```
int WINAPI start_preload_comand(
    int axis        // 対象軸番号
);
```

予めバッファリングされている指令を起動します。このコマンドは、全ての軸にブロードキャストされますので、1回の関数呼び出しで複数軸を同時起動できます。対称軸番号は、存在する軸であれば、どの軸でも構いません。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．3．17．clear_preload_command（バッファリングコマンド取り消し）関数

```
int WINAPI clear_preload_comand(
    int axis        // 対象軸番号
);
```

バッファリングされている指令を取り消します。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．3．18．軸移動動作指令の実際

下記に絶対位置指定 PTP 動作指令関数を用いた2軸の実際の往復移動動作実行プログラムの C 言語でのイメージを示します。

```
int ax0_err , ax1_err;

while( FOREVER )
{
    ax0_err = move_abs( 0, 2000 )    // Ax#0 start to 2000;
    ax1_err = move_abs( 1, 1500 )    // Ax#1 start to 1500;

    if ( ( ax0_err == SIO_ERROR) | | ( ax1_err == SIO_ERROR) )
        break;    // Hard Luck!! failed.

    while ( FOREVER )
    {
        if ( ( check_status( 0 ) == SIO_DONE ) && ( check_status( 1 ) == SIO_DONE ) )
        {
            if ( ( check_pfin( 0 ) == SIO_DONE ) && ( check_pfin( 1 ) == SIO_DONE ) )
                break;   // Move complete
        }
        Sleep( 5 );   // Anothor thread may need the time slice.
    }

    ax0_err = move_abs( 0, 0 )    // Ax#0 return to 0;
    ax1_err = move_abs( 1, 0 )    // Ax#1 return to 0;

    if ( ( ax0_err == SIO_ERROR) | | ( ax1_err == SIO_ERROR) )
        break;    // Hard Luck!! failed.

    while ( FOREVER )
    {
        if ( ( check_status( 0 ) == SIO_DONE ) && ( check_status( 1 ) == SIO_DONE ) )
        {
            if ( ( get_pfin( 0 ) == SIO_DONE ) && ( get_pfin( 1 ) == SIO_DONE ) )
                break;
        }
        Sleep( 5 );   // Anothor thread may need the time slice.
    }
}
```

４．４．サーボシステム状態変更関数

４．４．１．write_position（現在位置直接設定）関数

```
int WINAPI write_position(
    int axis,        // 対象軸番号
    long position    // 絶対座標系での現在位置
);
```

絶対座標系での現在位置の値を position の値に変更します。この関数は、絶対位置座標系をシフトする目的で使用するもので、これによるモータの移動動作はありません。

返り値

対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．４．２．set_son（サーボON指令）関数

```
int WINAPI set_son(
    int axis         // 対象軸番号
);
```

サーボアンプの状態をサーボON状態にします。但しサーボON状態へは、対象軸のサーボアンプの **Termi-BUS PIO** のSON入力がON（RC、SCシリーズでは主電源ON）になっていなければ移行できません。

返り値

対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．４．３．set_soff（サーボOFF指令）関数

```
int WINAPI set_soff(
    int axis         // 対象軸番号
);
```

サーボアンプの状態をサーボOFF状態にします。

返り値

対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．４．４．reset_alarm（サーボアンプアラームリセット）関数

```
int WINAPI reset_alarm(
    int axis         // 軸番号
);
```

サーボアンプのアラーム状態をリセットします。

返り値

対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．５．モータ移動動作パラメータ変更関数

４．５．１．write_velocity（速度／加速度指令設定）関数

```
int WINAPI write_velocity(
    int axis,          // 対象軸番号
    int vcmd,          // 速度指令値
    int acmd           // 加速度指令値
);
```

PTP動作時の速度及び加速度指令値を設定します。この関数は、実行データ領域の動作プロファイルパラメータを vcmd 及びacmdの値で書き換えますので以後の軸動作指令ではこの動作プロファイルパラメータが使用されます。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．５．２．select_svparm（サーボゲインパラメータ選択）関数

```
int WINAPI select_svparm(
    int axis,          // 軸番号
    int gain_sel,      // 停止時移動時の選択 0=移動時, 1=停止時
    int svparm         // サーボゲインパラメータ
);
```

対象軸の移動時又は停止時のサーボゲインパラメータを svparm の値とします。この関数は、実行データ領域のサーボゲインパラメータを書き換えますので以後の軸動作指令ではこのサーボゲインパラメータが使用されます。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．５．３．write_trqlim（電流制限値設定）関数

```
int WINAPI write_trqlim(
    int axis,          // 対象軸番号
    int l_dynamic,     // 停止時電流制限値
    int l_static       // 移動時電流制限値
);
```

サーボモータの移動中と停止中の電流制限値を設定します。この関数は、実行データ領域の電流制限パラメータを書き換えますので以後の軸動作指令ではこの電流制限値が使用されます。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．5．4．write_inpos（位置決め完了検出幅設定）関数

```
int WINAPI write_inpos(
    int axis,          // 対象軸番号
    long width         // 位置決め完了検出幅
);
```

位置決め時の動作完了を検出する時に用いる目標位置と現在位置の差の許容値を width の値とします。この関数は、実行データ領域の位置決め完了検出幅パラメータを書き換えますので以後の軸動作指令ではこの位置決め完了検出幅が使用されます。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．5．5．write_fzone（ゾーン出力＋側境界値設定）関数

```
int WINAPI write_fzone(
    int axis,          // 対象軸番号
    long zone          // 正転側ゾーン境界値
);
```

ゾーン出力の＋側の境界値をzoneの値に変更します。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．5．6．write_rzone（ゾーン出力－側境界値設定）関数

```
int WINAPI write_rzone(
    int axis,          // 対象軸番号
    long zone          // 逆転側ゾーン境界値
);
```

ゾーン出力の－側の境界値を変更します。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

4．6．サーボシステム内部状態確認関数

4．6．1．check_run（サーボシステム RUN 状態取得）関数

```
int WINAPI check_run(
   int axis        // 対象軸番号
);
```

返り値

サーボシステム RUN 状態であれば SIO_DONE が返り、それ以外では SIO_ERROR が返ります。

４．６．２．check_son（サーボ ON/OFF 状態取得）関数

```
int WINAPI check_son(
    int axis        // 対象軸番号
);
```

返り値

サーボ ON 状態であれば SIO_DONE が返り、サーボ OFF 状態では SIO_ERROR が返ります。

４．６．３．check_alrm（アラーム状態取得）関数

```
int WINAPI check_alrm(
    int axis        // 対象軸番号
);
```

返り値

アラーム状態であれば SIO_DONE が返り、正常状態では SIO_ERROR が返ります。

４．６．４．check_org（原点復帰完了状態取得）関数

```
int WINAPI check_org(
    int axis        // 対象軸番号
);
```

返り値

原点復帰完了状態であれば SIO_DONE が返り、それ以外では SIO_ERROR が返ります。

４．６．５．get_status（サーボシステム内部状態取得）関数

```
int WINAPI get_status(
    int axis,           // 対象軸番号
    char *param         // サーボシステム内部状態データ格納領域へのポインタ
);
```

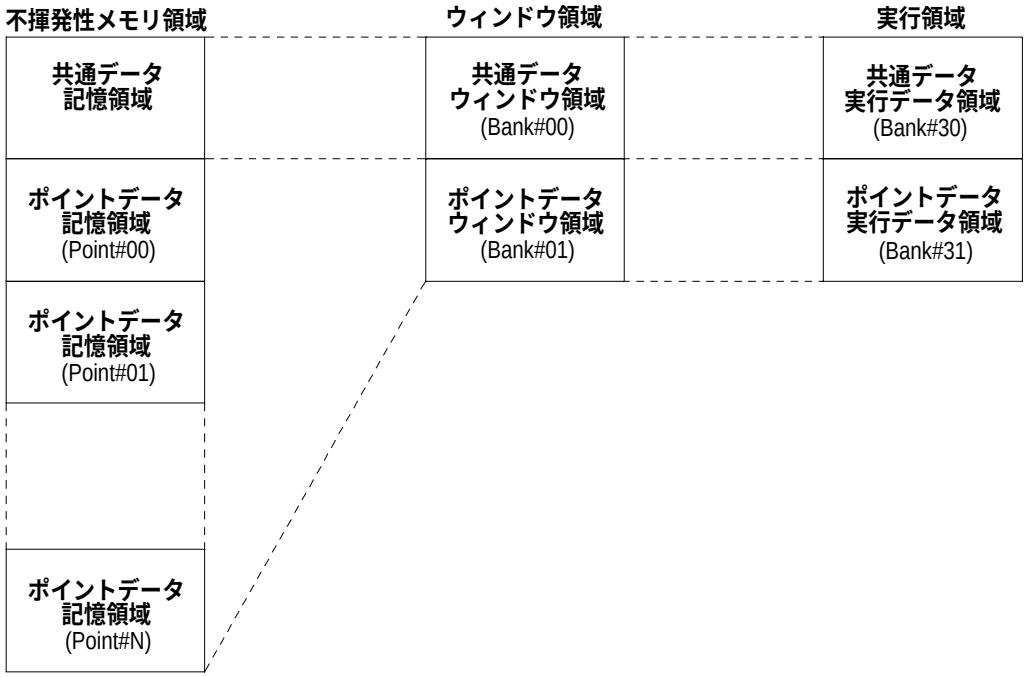
check_status関数または、其の他の通信を伴う関数の実行によって取得されたダイナミックリンクライブラリ内のサーボシステム動作状態モニタ変数から、対象軸のサーボアンプ／モータの全ての内部状態を取得しパラメータparamの指すchar配列にこれを格納します。

返り値

実行対象軸に正常に指令が受け入れられると SIO_DONE が返り、それ以外の状態では SIO_ERROR が返ります。

４．７．サーボアンプ／モータ内仮想メモリ空間アクセス関数

Termi-BUS SIO 対応サーボアンプ／モータ内部には、予め軸制御指令データを記憶できる不揮発性メモリ領域があり、この領域は、自由に読み書きができるウィンドウ領域を介してサーボアンプ／モータ内部の軸制御実行データ領域に結合されています。不揮発性メモリ領域アクセス関数によって、不揮発性メモリ領域に対して書き込み、及び読みだしを行うことができます。これらの操作は、全てウィンドウ領域を介して行われます。



通常は、不揮発性メモリ領域のデータ編集と記憶は、**Termi-BUS Tools** 又は **CTA** を用いて行いますので、不揮発性メモリ領域アクセス関数をシーケンスプログラムの中で用いてデータの転送を行う必要はありません。以降に示す仮想メモリ空間アクセス関数を使用する場合には、仮想メモリ空間の構造とその使用法についての情報を得るために **４．８．COMPACK構造体と仮想メモリ空間の実際** を参照下さい。

４．７．１．仮想メモリ空間からのランダムアクセス読み出し関数関数

```
int WINAPI read_svmem(
    int axis,       // 対象軸番号
    int address,    // 読み出しデータの仮想メモリ空間上アドレス
    long *dst       // 読み出しデータ格納領域へのポインタ
);
```

サーボアンプ内部の仮想メモリ空間上からaddressで指定されたデータを読み出し、ポインタ dst が指している領域に書き込みます。

返り値

読み出しが成功すれば SIO_DONE が返り、それ以外では SIO_ERROR が返ります。

4．7．2．仮想メモリ空間からのランダムアクセス書き込み関数

```
int WINAPI write_svmem(
    int axis,              // 対象軸番号
    int address,           // 仮想メモリ空間の書き込み先アドレス
    const long *src        // 書き込みデータ格納領域へのポインタ
);
```

サーボアンプ内部の仮想メモリ空間上のaddressで指定された領域に、ポインタ src が指しているデータを書き込みます。

返り値

書き込みが成功すれば SIO_DONE が返り、それ以外では SIO_ERROR が返ります。

4．7．3．サーボシステム動作パラメータ書き込み関数

```
int WINAPI write_param(
    int axis,                  // 対象軸番号
    const COMPACK *src   // 書き込みデータのある COMPACK 構造体へのポインタ
);
```

サーボアンプ内部の不揮発性メモリ領域の共通データ記憶領域に、ポインタ src が指している COMPACK 構造体に書き込まれた動作パラメータを一括して書き込みます。
COMPACK 構造体は、下記に示す構造体でその使用法については **4．8．COMPACK構造体と仮想メモリ空間の実際** を参照下さい。

```
typedef struct {
    int address[32];    // 仮想アドレスのバンク内オフセット値の配列( 値-1は処理をスキップ )
    long data[32];      // データの配列
} COMPACK;
```

返り値

書き込みが成功すれば SIO_DONE が返り、それ以外では SIO_ERROR が返ります。

4．7．4．サーボシステム動作パラメータ読みだし関数

```
int WINAPI read_param(
    int axis,              // 対象軸番号
    COMPACK *dst     // 読み出し先 COMPACK 構造体へのポインタ
);
```

サーボアンプ内部の不揮発性メモリ領域の共通データ記憶領域に記憶されている動作パラメータを読みだし、ポインタ dst が指している COMPACK 構造体に一括して書き込みます。
COMPACK 構造体は、下記に示す構造体でその使用法については **4．8．COMPACK構造体と仮想メモリ空間の実際** を参照下さい。

```
typedef struct {
    int address[32];    // 仮想アドレスのバンク内オフセット値の配列( 値-1は処理をスキップ )
    long data[32];      // データの配列
} COMPACK;
```

返り値

読み出しが成功すれば SIO_DONE が返り、それ以外では SIO_ERROR が返ります。

4. 7. 5. PTP動作指令データ書き込み関数

```
int WINAPI write_point(
    int axis,                   // 対象軸番号
    int point,                  // ポイント番号
    const COMPACK *src   // 書き込みデータのある COMPACK 構造体へのポインタ
);
```

サーボアンプ内部の不揮発性メモリ領域の、point で指定されたポイント番号に対応するポイントデータ記憶領域に、ポインタ src が指している COMPACK 構造体に書き込まれたPTP動作指令データを一括して書き込みます。
COMPACK 構造体は、下記に示す構造体でその使用法については **4. 8. COMPACK構造体と仮想メモリ空間の実際** を参照下さい。

```
typedef struct {
    int address[32];     // 仮想アドレスのバンク内オフセット値の配列( 値-1は処理をスキップ )
    long data[32];       // データの配列
} COMPACK;
```

返り値

書き込みが成功すれば SIO_DONE が返り、それ以外では SIO_ERROR が返ります。

4. 7. 6. PTP動作指令データ読みだし関数

```
int WINAPI read_point(
    int axis,               // 対象軸番号
    point,                  // ポイント番号
    COMPACK *dst     // 読み出し先 COMPACK 構造体へのポインタ
);
```

サーボアンプ内部の不揮発性メモリ領域の、point で指定されたポイント番号に対応するポイントデータ記憶領域に記憶されているPTP動作指令データを読みだし、ポインタ dst が指している COMPACK 構造体に一括して書き込みます。
COMPACK 構造体は、下記に示す構造体でその使用法については **4. 8. COMPACK構造体と仮想メモリ空間の実際** を参照下さい。

```
typedef struct {
    int address[32];     // 仮想アドレスのバンク内オフセット値の配列( 値-1は処理をスキップ )
    long data[32];       // データの配列
} COMPACK;
```

返り値

読み出しが成功すれば SIO_DONE が返り、それ以外では SIO_ERROR が返ります。

4．7．7．サーボシステム動作パラメータロード関数

```
int WINAPI load_param(
    int axis        // 対象軸番号
);
```

サーボアンプ内部の不揮発性メモリ領域の共通データ記憶領域に記憶されている動作パラメータを一括してウィンドウ領域を経由して実行領域にロードします。この関数実行後は、不揮発性メモリ領域に記憶されていた動作パラメータに従って動作します。ウィンドウ領域及び実行領域の各動作パラメータは、電源投入時には不揮発性メモリ領域に記憶されていた値が自動的にロードされます。従ってシステム起動時にこの関数を用いて動作パラメータをロードする必要はなく、再初期化時又は、不揮発性メモリ領域の動作パラメータを変更した時にのみ用います。動作パラメータは、予め **Termi-BUS Tools** 又は **CTA** によって不揮発性メモリ領域に書き込んでおく必要があります。

返り値

ロードが成功すれば SIO_DONE が返り、それ以外では SIO_ERROR が返ります。

4．7．8．サーボシステム動作パラメータセーブ関数

```
int WINAPI save_param(
    int axis        // 対象軸番号
);
```

実行領域の共通データ記憶領域に記憶されている動作パラメータを一括してウィンドウ領域を経由してサーボアンプ内部の不揮発性メモリ領域にセーブします。この関数は、実行時に変更された動作パラメータをそのまま不揮発性メモリ領域に書き込む必要がある場合に用います。

返り値

セーブが成功すればSIO_DONEが返り、それ以外ではSIO_ERRORが返ります。

4．7．9．PTP指令データセーブ関数

```
int WINAPI save_point(
    int axis,       // 対象軸番号
    int point       // ポイント番号
);
```

実行領域のポイントデータ記憶領域に記憶されている、現在のPTP指令データを一括してウィンドウ領域を経由してサーボアンプ内部の不揮発性メモリ領域にセーブします。この関数は、実行時に変更されたPTP指令データをそのまま不揮発性メモリ領域に書き込む必要がある場合に用います。

返り値

セーブが成功すればSIO_DONEが返り、それ以外ではSIO_ERRORが返ります。

4．7．10．サーボアンプ内部メモリ初期化関数

```
int WINAPI reset_memory(
    int axis        // 対象軸番号
);
```

ウィンドウ領域及び実行領域のサーボシステム動作パラメータとポイントデータをデフォルト値とします。設定されるデフォルト値は、機種によって決まった固定値で、不揮発性メモリ領域に書き込まれている値ではありません。reset_memory 関数の実行によってグローバルメモリ上の対象軸の状態変数は新しい値に更新されます。

4．8. COMPACK 構造体と仮想メモリ空間の実際

サーボアンプ／モータ内部の仮想メモリ空間は、全てバンクと呼ばれる32要素のDWORD配列に分けられています。COMPACK構造体は、このバンクを一括して取り扱うことができる様にするためのもので、下記に示すような2つの32要素の配列からなります。

```
typedef struct {
    int address[32];    // 仮想アドレスのバンク内オフセット値の配列( 値-1は処理をスキップ )
    long data[32];      // データの配列
} COMPACK;
```

COMPACK構造体では address と data の2つの要素は1組として取り扱われ、ある数 i に対する address[i] と data[i] はあるバンク内のある1つの要素に対するバンク内オフセットアドレスとデータを意味します。address に-1の値が書かれている項目は無視されますので、これによりバンク内の任意の部分データの抽出や部分書き込みを実現することができます。

以下にバンク0（サーボシステム動作パラメータ）とバンク1（PTP動作指令データ）、及び状態モニタ領域を含む全仮想メモリ空間のデータ構造を以下に示します。なお COMPACK構造体の address フィールドに指定するバンク内オフセットアドレスは、下記の表中の仮想アドレスから、そのバンクのベースアドレス（そのバンクの先頭の項目のアドレス）を減算したものになります。

共通パラメータ　ウィンドウ領域　バンク０（ＣＯＭ０）

仮想アドレス (HEX)	略号	項目	機種
００００００００	ＣＮＴＭ	絶対位置座標範囲＋側最大値	
００００００００１	ＣＮＴＬ	絶対位置座標範囲－側最大値	
００００００００２	ＬＩＭＭ	ソフトウエアストロークリミット値＋側	
００００００００３	ＬＩＭＬ	ソフトウエアストロークリミット値－側	
００００００００４	ＺＯＮＭ	ゾーン境界値＋側	
００００００００５	ＺＯＮＬ	ゾーン境界値－側	
００００００００６	ＯＲＧ	原点復帰パターン選択コード ビット０～３：原点復帰パターン選択コード ビット７：近回り制御有効指定ビット（１＝有効）	
００００００００７	ＰＨＳＰ	モータ励磁相信号検出動作パラメータ ビット０～６　：励磁相信号検出動作起動遅延時間指定コード 単位：１ｍｓ ビット７：励磁相信号検出動作移動方向指定ビット ０／１＝正転／逆転	Ｒ／ＲＣ
００００００００８	ＦＰＩＯ	ＰＩＯ機能設定フラグ ビット０：０／１＝ＰＦＩＮ／ＩＮP ビット４：１＝ＣＳＴＲ無効 ビット５：１＝ＩＮＨ＋無効 ビット６：１＝ＩＮＨ－無効 ビット７：１＝ＩＬＫ無効	
００００００００９	ＢＲＳＬ	ＳＩＯ通信速度選択コード	
００００００００Ａ	ＯＶＣＭ	原点復帰時の速度指令　単位：０．２ｒ／ｍｉｎ	
００００００００Ｂ	ＯＡＣＣ	原点復帰時の加速度指令　単位：０．１ｒ／ｍｉｎ／ｍｓ	
００００００００Ｃ	ＲＴＩＭ	従局トランスミッタ活性化最小遅延時間パラメータ 単位：１ｍｓ	
００００００００Ｄ	ＩＮＰ	インポジション幅デフォルト値	

０００００００Ｅ	ＶＣＭＤ	速度指令デフォルト値　単位：０．２ｒ／ｍｉｎ	
０００００００Ｆ	ＡＣＭＤ	加速度指令デフォルト値　単位：０．１ｒ／ｍｉｎ／ｍｓ	
００００００１０	ＳＰＯＷ	位置決め停止時の電流制限デフォルト値	
００００００１１	ＤＰＯＷ	移動時の電流制限デフォルト値	
００００００１２	ＰＬＧ０	サーボゲイン番号デフォルト値	
００００００１３	ＭＸＡＣ	加速時最大加速度指定フラグデフォルト値	Ｒ／ＲＣ
００００００１４	ＣＰＡＣ	ＣＰ制御モード加減速時定数（将来の拡張のための予約）	Ｂ／Ｒ
００００００１５	ＰＳＷＴ	特殊仕様（予約）	
		将来の拡張のための予約	
００００００１８	ＺＲＭＫ	原点復帰禁止フラグ(Ｂシリーズアブソリュートのみ)	Ｂ
００００００１９	ＯＤＰＷ	原点復帰時の電流制限値	Ｒ／ＲＣ
００００００１Ａ	ＯＴＩＭ	原点復帰時のタイムアウト値　単位：１ｍｓ	Ｒ／ＲＣ
００００００１Ｂ	ＰＬＧ１	位置決め停止時のサーボゲイン番号デフォルト値	Ｂ
００００００１Ｃ	ＰＬＪＬ	負荷イナーシャによるサーボゲインテーブル選択スイッチ ０＝軽負荷イナーシャ １＝中負荷イナーシャ ２＝重負荷イナーシャ	Ｂ
００００００１Ｄ	ＦＬＳＬ	電流指令フィルタの種別選択フラグ ０＝１次ローパスフィルタ １＝バンドエリミネーションフィルタ	Ｂ
００００００１Ｅ	ＦＬＦＣ	電流指令のＬＰカットオフ周波数／ＢＥＦ中心周波数	Ｂ
００００００１Ｆ		不揮発性記憶領域（領域Ａ）通算書き込み回数	

ポイントデータ ウインドウ領域　　バンク１（ＰＮＴ１）

仮想アドレス(HEX)	略号	項目	機種
００００４００	ＰＣＭＤ	絶対位置座標位置決め停止目標位置	
００００４０１	ＦＬＧＰ	軸動作パラメータデフォルト／ポイントデータ選択フラグ ポイントデータ有効 ビット７：インポジション幅 ビット６：速度、加速度 、加速時加速度最大 ビット５：電流制限値 ビット４：サーボゲイン番号	
００００４０２		将来拡張のための予約	
００００４０３	ＩＮＰ	インポジション幅	
００００４０４	ＶＣＭＤ	速度指令　単位：０．２ｒ／ｍｉｎ	
００００４０５	ＡＣＭＤ	加速度指令　単位：０．１ｒ／ｍｉｎ／ｍｓ	
００００４０６	ＳＰＯＷ	位置決め停止時の電流制限値	
００００４０７	ＤＰＯＷ	移動時の電流制限値	
００００４０８	ＰＬＧ０	サーボゲイン番号値	
００００４０９	ＭＸＡＣ	加速時最大加速度指定フラグ ビット０：１＝加速時最大加速度	Ｒ／ＲＣ
００００４１１	ＰＬＧ１	位置決め停止時のサーボゲイン番号	Ｂ
００００４１Ｆ		不揮発性記憶領域（領域Ａ）通算書き込み回数	

アンプ／モータ機種モニタ　　バンク２６（ＴＹＰＥ）

仮想アドレス (HEX)	略号	項目	機種
００００６８００	ＲＯＭ	機種コード及びＲＯＭバージョンコード	
００００６８０１	Ｓ／Ｎ	シリアル番号	ＲＣ
００００６８０２	ＡＭＰ１	サーボアンプ型名文字列 第１組４文字	
００００６８０３	ＡＭＰ２	サーボアンプ型名文字列 第２組４文字	
００００６８０４	ＡＭＰ３	サーボアンプ型名文字列 第３組４文字	
００００６８０５		予約	
００００６８０６	ＭＯＴ１	サーボモータ型名文字列 第１組４文字	
００００６８０７	ＭＯＴ２	サーボモータ型名文字列 第２組４文字	
００００６８０８	ＭＯＴ３	サーボモータ型名文字列 第３組４文字	
００００６８０９		予約	

モニタ関連データ　　バンク２７（ＭＯＮＩ）

仮想アドレス (HEX)	略号	項目	機種
００００６Ｃ００	Ａ＿ＦＬ	アナログモニタのフラグ　　（注１）	Ｒ
００００６Ｃ０１	Ａ＿ＡＤ	アナログモニタのアドレス （固定値７４０１）（注１）	Ｒ
００００６Ｃ０２	Ｈ＿ＤＴ	トレースデータ指定アドレス	Ｂ／ＲＣ
００００６Ｃ０３	Ｈ＿ＳＣ	トレースデータのサンプリング間隔 設定値ｎ：（ｎ＋１）＊５００μｓ毎	Ｂ／ＲＣ
００００６Ｃ０４	Ｈ＿ＷＲ	トレースデータの書き込み済みの最大アドレス 最上位ビットが１の時全領域書き込み完了	Ｂ／ＲＣ
００００６Ｃ０５	Ｈ＿ＢＹ	トレースするデータ型 ＢＹＴＥ＝１，ＷＯＲＤ＝２，ＬＷＯＲＤ＝４	Ｂ／ＲＣ

（注１）弊社検査用のため設定しないでください。

トレースデータの格納領域

仮想アドレス (HEX)	略号	項目	機種
１００００００		最初のデータ	
１００００００１		２番目のデータ	
．		．	
＊＊＊＊＊＊＊＊		最後のデータ	

アラームモニタ領域　　バンク２８（ＡＬＲＭ）

仮想アドレス (HEX)	略号	項目	機種
０００７０００	ＷＡＲＮ	最終検出ワーニングコード	
０００７００１	ＨＹＳ０	最終検出アラームコード	
０００７００２	ＨＹＳ１	１回前の検出アラームコード	
０００７００３	ＨＹＳ２	２回前の検出アラームコード	
０００７００４	ＨＹＳ３	３回前の検出アラームコード	
０００７００５	ＨＹＳ４	４回前の検出アラームコード	
０００７００６	ＨＹＳ５	５回前の検出アラームコード	
０００７００７	ＨＹＳ６	６回前の検出アラームコード	
０００７００８	ＨＹＳ７	７回前の検出アラームコード	
０００７００９	ＡＲＭＡ	実行時データ異常があったデータのアドレス	

内部状態モニタ領域　　バンク２９（ＳＴＡＴ）

仮想アドレス (HEX)	略号	項目	機種
００００７４００	ＰＮＯＷ	絶対位置カウンタ現在位置	
００００７４０１	ＶＮＯＷ	現在速度モニタ	
００００７４０２		将来拡張のための予約	
００００７４０３	ＳＴＡＴ	内部ステータスフラグ	
００００７４０４	ＡＬＲＭ	現在のアラーム／ワーニングコード	
００００７４０５	ＰＩ	ＰＩＯ入力ポートモニタ	Ｂ／Ｒ
００００７４０６	ＰＯ	ＰＩＯ出力ポートモニタ	
００００７４０７	ＳＷ	ＳＷ１（ロータリ）、ＳＷ２（ＤＩＰ）の状態モニタ ビット４～７：ＳＷ１の４ビットコードの状態（軸番号） ビット３：ＳＷ２の６（１／０＝オン／オフ） ビット２：ＳＷ２の５（１／０＝オン／オフ） ビット１：ＳＷ２の４（１／０＝オン／オフ） ビット０：ＳＷ２の３（１／０＝オン／オフ）	Ｂ／Ｒ
００００７４０８	ＳＴＡ２	ビット０：原点復帰実行中フラグ １＝原点復帰実行中	
００００７４０９	ＷＡＤＲ	Ｗ４コマンドでの書き込み先アドレスカウンタ	
００００７４０Ａ	ＲＯＭ	機種コード及びＲＯＭバージョン	
００００７４０Ｂ	Ａ／Ｄ０	検査用アナログデータ値	Ｒ／ＲＣ
００００７４０Ｃ	Ａ／Ｄ１	検査用アナログデータ値	Ｒ／ＲＣ
００００７４０Ｄ	Ａ／Ｄ２	検査用アナログデータ値	Ｒ／ＲＣ
００００７４１２	ＯＬＬＶ	過負荷検出現在値　７８Ｈ以上でアラーム	
００００７４１３	ＬＶＰＫ	過負荷検出レベルのピークホールド値	
００００７４１４	ＩＣＭＤ	内部電流指令値（トルク指令値）１００／定格電流	Ｂ
００００７４１５	ＰＮＴＭ	現在位置番号モニタ（０～２５５）	Ｂ

. 変 更 履 歴 .

Ver. 1.00 初版 3rd/Jun./2001

Ver. 2.00 第２版 14th/Feb./2003

４. ２. ３. close_tmbs **関数** を追加