ＩＴ－３２０５

ＤＬＬ

コマンドマニュアル

株式会社　アイ･ティー･ティー

資料番号　BS20002CM08

注意

１． ご使用の前に本ユーザーズマニュアルとその他の関連資料を熟読し正しくお使いください。

２． 本製品は一般産業用です。本製品の故障や誤作動により直接人命に関わる装置（原子力制御、航空宇宙機器、交通機器、医療機器、各種安全装置）に使用する場合、その都度検討が必要になります。弊社までご相談ください。

目 次

1. はじめに

1.1. 概要

ＩＴ－３２０５ＤＬＬは、VisualC++や VisualBasic 等で作成したアプリケーションから関数を動的に呼び出して使用することができるダイナミックリンクライブラリです。ＩＴ－３２０５ＤＬＬには共通の軸動作やＩ/O制御への対応が組み込まれているため、Ｗｉｎｄｏｗｓ上の開発から難しい作業の多くを省くことができます。また、別途配布されているＣ言語ソースファイルを自由にカスタマイズしてアプリケーションに組み込むことで開発時間を大幅に短縮することができます。

本書はこのＩＴ－３２０５ＤＬＬを使用するためのマニュアルです。このＤＬＬに定義されている関数の使用方法について説明しています。

＊C 言語ソースファイルは IT-3205 ご購入の方のみに配布させていただいております。
ご希望の方は下記のメールアドレスまでその旨お伝えください。
また、基本的にソースファイルの内容に関する御質問にはお答えできません。

電子メール　cnc@itt.co.jp

2. コマンドの説明

ＤＬＬに定義されている関数の内容について説明します。

2.1. コマンド一覧表

2.2. コマンドの詳細

各コマンドの詳細について説明します

2.2.1. データ通信コマンドグループ

一次局と二次局の間のデータ交換に使用するコマンドについて説明します。

1.1 パラメータ読込みコマンド	PrmGet
コマンド分類	データ通信コマンド

ＶＣ

```
int PrmGet(
        int     axis_no,
        int     prm_no,
        long    ＊data_h,
        long    ＊data_l
       );
```

ＶＢ

```
Private Declare Function prmGet Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal prm_no As Long, data_h As Long, data_l As Long) As Long
```

引数

axis_no　：制御する軸番号
prm_no　：パラメータ番号
「IT－3205DLL ユーザーズマニュアル」の
見出し番号 2.1 パラメータデータ一覧表を参照ください。
data_h　：パラメータデータ（上位４バイト）
data_l　：パラメータデータ（下位４バイト）

戻り値

正常終了　　０
異常終了　－１

解説

パラメータの番号を指定してパラメータの読出しを行います。
なお、このコマンドはモーションコマンド*1 動作中は実行しないで下さい。

＊１ AxHome，AxHomeOp，AxFeed，AxFeedOp，AxNc，AxNcOp，AxNcEx，AxNcExOp，LkStart，IpStrt，IpStrtRp

1.2 パラメータ書込みコマンド	PrmPut
コマンド分類	データ通信コマンド

ＶＣ

```
int PrmPut(
        int     axis_no,
        int     prm_no,
        long    data_h,
        long    data_l
       );
```

ＶＢ

```
Private Declare Function prmPut Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal prm_no As Long, ByVal data_h As Long, ByVal data_l As Long) As Long
```

引数

axis_no　：制御する軸番号
prm_no　：パラメータ番号
「IT－3205DLL ユーザーズマニュアル」の
見出し番号 2.1 パラメータデータ一覧表を参照ください。
data_h　：書込みを行うパラメータのデータ内容（上位４バイト）
パラメータサイズが８バイトの場合にのみ使用されます
それ以外の場合は０を入れてください。
data_l　：書込みを行うパラメータのデータ内容（下位４バイト）

戻り値

正常終了　０
異常終了　－１

解説

パラメータの番号を指定してパラメータの書込みをおこないます。
なお、このコマンドはモーションコマンド*1 動作中は実行しないで下さい。

＊１ AxHome, AxHomeOp, AxFeed, AxFeedOp, AxNc, AxNcOp, AxNcEx, AxNcExOp, LkStart, IpStrt, IpStrtRp

Σｎシリーズの場合の注意点

下記の３定数はについてはコマンド終了後、機器のセットアップコマンド(Config)を発行して下さい。
Pn-0001：メモリスイッチ１
Pn-0002：メモリスイッチ２
Pn-0017：エンコーダパルス数

1.3 不揮発パラメータ読込みコマンド	PprmGet
コマンド分類	データ通信コマンド

ＶＣ

```
int PprmGet(
        int     axis_no,
        int     prm_no,
        long    data_h,
        long    data_l
    );
```

ＶＢ

```
Private Declare Function PprmGet Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal prm_no As Long, data_h As Long, data_l As Long) As Long
```

引数

axis_no	：制御する軸番号
prm_no	：パラメータ番号 「IT－3205DLL ユーザーズマニュアル」の 見出し番号 2.1 パラメータデータ一覧表を参照ください。
data_h	：パラメータデータ（上位４バイト）
data_l	：パラメータデータ（下位４バイト）

戻り値

正常終了　０

異常終了　－１

解説

パラメータの番号を指定して不揮発メモリ上のパラメータを読出します。

なお、このコマンドはモーションコマンド*1 動作中は実行しないで下さい。

＊１ AxHome，AxHomeOp，AxFeed，AxFeedOp，AxNc，AxNcOp，
AxNcEx，AxNcExOp，LkStart，IpStrt，IpStrtRp

注意事項

このコマンドはΣｎシリーズでサポートされていますが、ΣⅡシリーズでは
未サポートコマンドとなっています。

1.4 不揮発パラメータ書込みコマンド	PprmPut
コマンド分類	データ通信コマンド

ＶＣ

```
int PprmPut(
        int     axis_no,
        int     prm_no,
        long    data_h,
        long    data_l
    );
```

ＶＢ

```
Private Declare Function PprmPut Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal prm_no As Long, ByVal data_h As Long, ByVal data_l As Long) As Long
```

引数

axis_no ：制御する軸番号
prm_no ：パラメータ番号
「IT－3205DLL ユーザーズマニュアル」の
見出し番号 2.1 パラメータデータ一覧表を参照ください。
data_h ：書込みを行うパラメータのデータ内容（上位４バイト）
パラメータサイズが８バイトの場合にのみ使用されます
それ以外の場合は０を入れてください。
data_l ：書込みを行うパラメータのデータ内容（下位４バイト）

戻り値

正常終了 ０
異常終了 －１

解説

パラメータの番号とデータサイズを指定して不揮発メモリ上にパラメータを書込みます。なお、このコマンドはモーションコマンド*1動作中は実行しないで下さい。

＊１ AxHome, AxHomeOp, AxFeed, AxFeedOp, AxNc, AxNcOp, AxNcEx, AxNcExOp, LkStart, IpStrt, IpStrtRp

Σｎシリーズの場合の注意点

不揮発メモリ上に格納されたデータは電源を切った後もデータが保持されます。
下記の３定数はについてはコマンド終了後、機器のセットアップコマンド(Config)を発行して下さい。

Pn-0001：メモリスイッチ１
Pn-0002：メモリスイッチ２
Pn-0017：エンコーダパルス数

1.5 ＩD情報読込みコマンド	IdGet
コマンド分類	データ通信コマンド

ＶC

```
int IdGet(
        int             axis_no,
        unsigned char   device_no,
        unsigned char   idtbl[17]
       );
```

ＶB

```
Private Declare Function IdGet Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal device_no As Long, idtbl As Long) As Long
```

引数

axis_no ：制御する軸番号
device_no ：デバイスコード
idtbl[17] ：読み出したＩD内容を格納する

戻り値

正常終了 ０
異常終了 －１

解説

デバイスコードで指定したＩDデータを読込みます。読み出されるＩDの種類は機器ごとに異なります。読み出されるＩDの詳細については各機器のマニュアルにて確認してください。

サーボパックがＳＧＤ－□□□Ｎの場合の例

読み出されるＩD内容

デバイスコード	ID 内容															
	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00H	S	G	D	－	＊	＊	＊	N	00	不定						
02H	ソフトVer		不定													

（注）１．００～０７までの文字はアスキーコード／０８は００Hです。
２．ソフトＶｅｒはバイナリデータです。

1.6 ステータスデータ読込みコマンド	StdtGet
コマンド分類	データ通信コマンド

VC

```
int StdtGet(
        int     axis_no,
        int     stdt_no ,
        long    ＊stdt
       );
```

VB

```
Private Declare Function StdtGet Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal stdt_no As Long, stdt As Long) As Long
```

引数

axis_no ：制御する軸番号
stdt_no ：ステータス選択コード
「IT－3205DLL ユーザーズマニュアル」の
見出し番号 3.1 ステータスコード一覧表を参照ください。
stdt ：ステータスデータ

戻り値

正常終了 0
異常終了 －1

解説

ステータス選択コードで指定したデータを読込みます。

1.7 ステータスデータ読込みコマンド	StdtstGet
コマンド分類	データ通信コマンド

ＶＣ

```
int StdtstGet(
        int     axis_no,
        long    ＊st_state
       );
```

ＶＢ

```
Private Declare Function StdtstGet Lib "mechad00.dll" (ByVal axis_no As Long,
st_state As Long) As Long
```

引数

Axis_no　：制御する軸番号
st_state　：ステータスデータ
ステータスデータの詳細については「IT－3205DLL ユーザーズマニュアル」の 見出し番号 3.1 ステータスコード一覧表を参照ください。

戻り値

正常終了　０
異常終了　－１

解説

ステータスデータを１６ビットで一括して読込みます。

1.8 入出力信号読込みコマンド	IodtGet
コマンド分類	データ通信コマンド

ＶＣ

```
int IodtGet(
        int     axis_no,
        long    ＊Iodt
       );
```

ＶＢ

```
Private Declare Function IodtGet Lib "mechad00.dll" (ByVal axis_no As Long,
Iodt As Long) As Long
```

引数

axis_no　：制御する軸番号
Iodt　：入出力信号モニタデータ

戻り値

正常終了　０
異常終了　－１

解説

入出力信号モニタ情報を１６ビットで一括して読込みます。

入出力信号モニタデータビット並び

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						BRK	EXT3	EXT2	EXT1	PC	PB	PA	DEC	N-OT	P-OT

各入出力信号の説明

ビット番号	名称	内容
0	Ｐ－ＯＴ	正転側リミットスイッチ入力
1	Ｎ－ＯＴ	逆転側リミットスイッチ入力
2	ＤＥＣ	減速用リミットスイッチ（減速ＬＳ）入力
3	ＰＡ	エンコーダA相信号入力
4	ＰＢ	エンコーダB相信号入力
5	ＰＣ	エンコーダC相信号入力
6	ＥＸＴ１	第１外部（ラッチ）信号入力
7	ＥＸＴ２	第２外部（ラッチ）信号入力
8	ＥＸＴ３	第３外部（ラッチ）信号入力
9	ＢＲＫ	ブレーキ状態出力

1.9 モニタ情報読込みコマンド	MonGet
コマンド分類	データ通信コマンド

ＶＣ

```
int MonGet(
        int     axis_no,
        int     mcode,
        long    ＊mdata
       );
```

ＶＢ

```
Private Declare Function MonGet Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal mcode As Long, mdata As Long) As Long
```

引数

axis_no　：制御する軸番号
mcode　：モニタ選択コード
「IT－3205DLL ユーザーズマニュアル」の
見出し番号 4 モニタ選択コード一覧表を参照ください。
mdata　：モニタデータ

戻り値

正常終了　０
異常終了　－１

解説

サーボドライブの位置や速度情報を読込みます。
引数で指定したモニタ選択コードが、McodePut コマンドによりモニタコードとして設定されていない場合はエラーを返します。

1.10 モニタ選択コードの設定コマンド	McodePut
コマンド分類	データ通信コマンド

ＶＣ

```
int McodePut(
        int     axis_no,
        int     m1,
        int     m2
      );
```

ＶＢ

```
Private Declare Function McodePut Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal m1 As Long, ByVal m2 As Long) As Long
```

引数

axis_no　：制御する軸番号
m1　：モニタ１で読込む情報のモニタコード。
「IT－3205DLL ユーザーズマニュアル」の
見出し番号 4 モニタ選択コード一覧表を参照ください。
m2　：モニタ２で読込む情報のモニタコード。
「IT－3205DLL ユーザーズマニュアル」の
見出し番号 4 モニタ選択コード一覧表を参照ください。

戻り値

正常終了　０
異常終了　－１

解説

MECHATROLINK-Ⅰのコマンドではモニタ情報として２個まで読込める仕様なのでモニタ番号１とモニタ番号２で読込む情報を設定します。
このコマンドで設定されたモニタコードは MonGet コマンドで読込み可能なモニタコードとして利用されます。

使用例

```
McodePut( 1，1，８ );①
MonGet( 1，1,＊X );②
MonGet( 1，8,＊Y);③
```

①軸番号１のモニタ番号１へモニタ選択コード１（位置情報）を設定します。
　モニタ番号２へモニタ選択コード８（速度情報）を設定します。
②軸番号１のモニタ選択コード１（位置情報）の読込み
　変数 X へ軸番号１の位置情報が格納されます。
③軸番号１のモニタ選択コード１（位置情報）の読込み
　変数 Y へ軸番号１の速度情報が格納されます。

またデフォルト値としてモニタ番号１に位置情報（モニタ選択コード１：機械座標系における指令位置)、モニタ番号２に速度情報（コード８：フィードバックから求めた機械速度）が設定されています。

1.11 オプションコードの設定コマンド	OcodePut
コマンド分類	データ通信コマンド

ＶＣ

```
int OcodePut(
        int     axis_no,
        int     ocodeno,
        int     ocode
      );
```

ＶＢ

```
Private Declare Function OcodePut Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal ocodeno As Long, ByVal ocode As Long) As Long
```

引数

axis_no　：制御する軸番号
ocodeno　：オプションコード番号
ocode　　：オプションコード

戻り値

正常終了　０
異常終了　－１

解説

加減速モードと速度ループ制御を設定します。また、オプションコードについては表 1.1 を参照ください。
ここで設定された情報が、AxHome，AxFeed，AxNc，AxNcEx，AxDown，SvOn，Ipd1Set，Ipd2Set，Ipd3Set，Ipd4Set，Ipd5Set，Ipd6Set コマンドの設定として利用されます。また、これらのコマンド以外で使用される加減速モードや速度ループは、OcodePut コマンドの設定には影響ありません。
なお、軸移動中にこのコマンドを発行しましても、次の軸移動コマンドが発行されるまで値は反映されません。

表 1.1 オプションコード番号ごとの選択項目対応表

オプションコード番号	オプションコード		
０：加減速モード	０：直線加減速	１：指数関数加減速	２：S 字加減速
１：速度ループＰ／ＰＩ切替え	０：ＰＩ制御	１：Ｐ制御	

1.12 アラームコード読込みコマンド	AlmGet
コマンド分類	データ通信コマンド

ＶＣ

```
int AlmGet(
        int             axis_no ,
        unsigned char   ＊almdt
       );
```

ＶＢ

```
Private Declare Function AlmGet Lib "mechad00.dll" (ByVal axis_no As Long,
almdt As Integer) As Long
```

引数

axis_no　：制御する軸番号
almdt　：アラームコード

戻り値

正常終了　０
異常終了　－１

解説

アラームコードの読込みをおこないます。
アラームコードについては「IT－3205DLL ユーザーズマニュアル」
の見出し番号 5 アラームコード一覧表を参照ください。

2.2.2. モーションコマンドグループ

モーションコントロールに使用するコマンドについて説明します。

2.1 原点復帰コマンド	AxHome
コマンド分類	モーションコマンド

ＶＣ

```
int AxHome(
        int     axis_no,
        long    posi,
        int     vel
       );
```

ＶＢ

```
Private Declare Function AxHome Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal vel As Long, ByVal latch As Long) As Long
```

引数

axis_no　：制御する軸番号
posi　：目標位置［指令単位］
vel　：送り速度［指令単位／秒］

戻り値

正常終了　０
異常終了　－１

解説

減速 LS とラッチ信号を使った原点復帰を行います。
動作シーケンス
１．パラメータで指定された方向に、コマンドで指定した速度で早送りを開始します。
２．減速 LS が Close になるとアプローチ速度まで減速し、定速送りを行います。
３．減速 LS が 0pen になり、指定の外部信号が入力されたら、パラメータで設定された最終走行距離をクリープ速度で移動し、減速停止します。

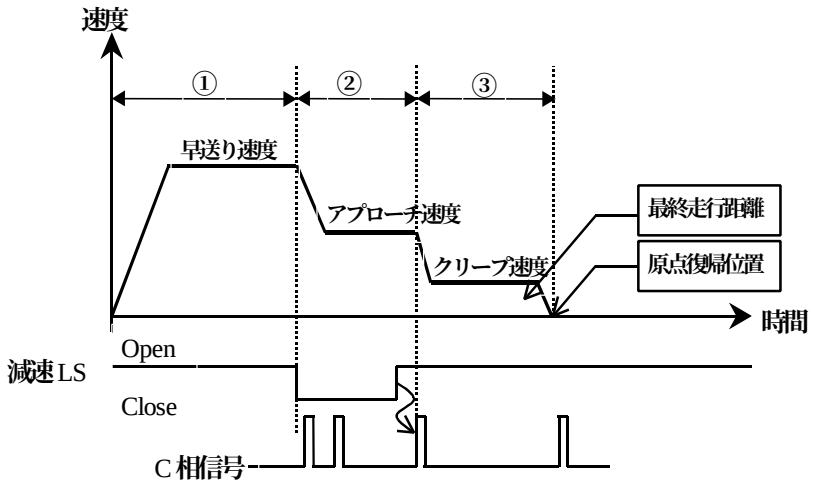

図2.1 ドライブ側原点復帰シーケンス（C 相の場合）

2.2 原点復帰コマンド（オプション設定有り）	AxHomeOp
コマンド分類	モーションコマンド

ＶＣ

```
int AxHomeOp(
        int     axis_no,
        int     vel,
        int     slope,
        int     velloop,
        int     latch
       );
```

ＶＢ

```
Private Declare Function AxHomeOp Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal vel As Long, ByVal slope As Long, ByVal velloop As Long, ByVal latch
As Long) As Long
```

引数

axis_no	：制御する軸番号
vel	：早送り速度［指令単位／秒］
slope	：加減速モード（０：直線加減速，１：指数関数加減速，２：S 字加減速）
velloop	：速度ループＰ／ＰＩ切替え（０：ＰＩ制御，１：Ｐ制御）
latch	：ラッチ信号選択 （０：エンコーダＣ相信号，１：第１外部信号，２：第２外部信号， ３：第３外部信号）

戻り値

正常終了　０

異常終了　－１

解説

減速 LS とラッチ信号を使った原点復帰を行います。

動作シーケンス

１．パラメータで指定された方向に、コマンドで指定した速度で早送りを開始します。

２．減速 LS が Close になるとアプローチ速度まで減速し、定速送りを行います。

３．減速 LS が Open になり、指定の外部信号が入力されたら、パラメータで設定された最終走行距離をクリープ速度で移動し、減速停止します。

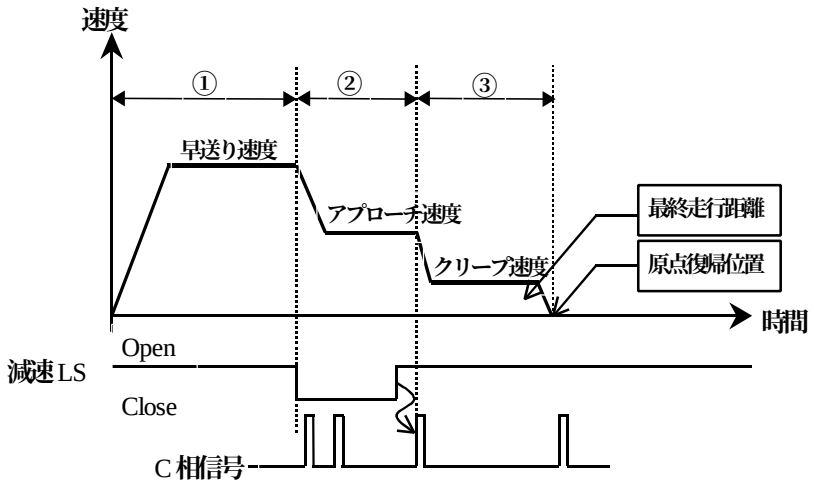

図2.2 ドライブ側原点復帰シーケンス（C相の場合）

2.3 定速送りコマンド	AxFeed
コマンド分類	モーションコマンド

ＶＣ

```
int AxFeed(
        int    axis_no,
        int    vel
       );
```

ＶＢ

Private Declare Function AxFeed Lib "mechad00.dll" (ByVal axis_no As Long, ByVal vel As Long) As Long

引数

axis_no　：制御する軸番号
vel　：早送り速度［指令単位／秒］

戻り値

正常終了　０
異常終了　－１

解説

指定送り速度による定速送りを行います。
図 2.3 で示すように、指令された送り速度で定速送りを実行します。

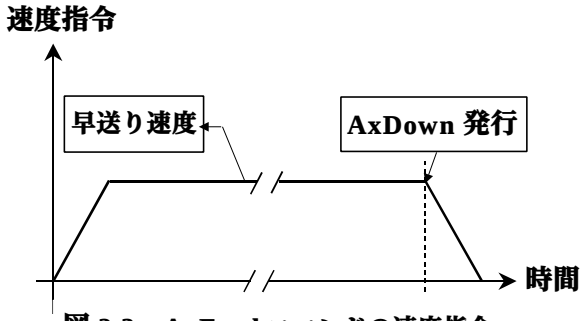

図 2.3　AxFeed コマンドの速度指令

2.4 定速送りコマンド（オプション設定有り）	AxFeedOp
コマンド分類	モーションコマンド

ＶＣ

```
int AxFeedOp(
        int     axis_no,
        int     vel,
        int     slope,
        int     velloop
       );
```

ＶＢ

```
Private Declare Function AxFeedOp Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal vel As Long, ByVal slope As Long, ByVal velloop As Long, ByVal velloop
As Long) As Long
```

引数

axis_no	：制御する軸番号
vel	：早送り速度［指令単位／秒］
slope	：加減速モード（０：直線加減速，１：指数関数加減速，２：S 字加減速）
velloop	：速度ループＰ／ＰＩ切替え（０：ＰＩ制御，１：Ｐ制御）

戻り値

正常終了　０

異常終了　－１

解説

指定送り速度による定速送りを行います。

図 2.4 で示すように、指令された送り速度で定速送りを実行します。

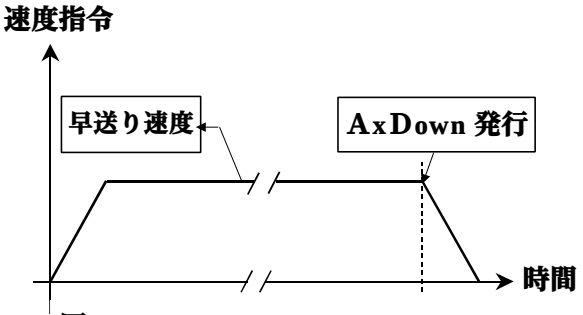

図 2.4　AxFeedOp コマンドの速度

2.5 早送り位置決めコマンド	AxNc
コマンド分類	モーションコマンド

ＶＣ

```
int AxNc(
        int     axis_no,
        long    posi,
        int     vel
       );
```

ＶＢ

```
Private Declare Function AxNc Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal posi As Long, ByVal vel As Long) As Long
```

引数

axis_no	：制御する軸番号
posi	：目標位置［指令単位］
vel	：早送り速度［指令単位／秒］

戻り値

正常終了　　０
異常終了　－１

解説

引数で指定した速度による目標位置への早送り位置決めを行います。
図 2.5 に示すように、指定した早送り速度で目標位置（P1）に位置決めします。

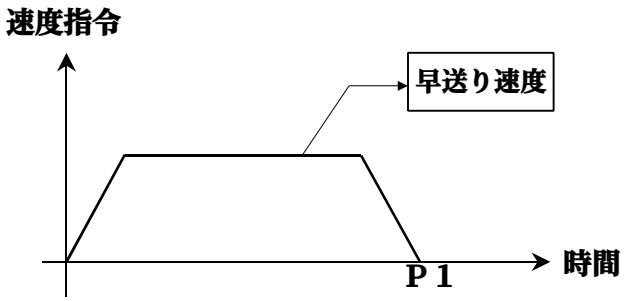

図 2.5 AxNc コマンドの速度指令

2.6 早送り位置決めコマンド（オプション設定有り）	AxNcOp
コマンド分類	モーションコマンド

ＶＣ

```
int AxNcOp(
        int     axis_no,
        long    posi,
        int     vel,
        int     slope,
        int     velloop
      );
```

ＶＢ

Private Declare Function AxNcExOp Lib "mechad00.dll" (ByVal axis_no As Long, ByVal posi As Long, ByVal vel As Long, ByVal slope As Long, ByVal velloop As Long, ByVal latch As Long) As Long

引数

axis_no	：制御する軸番号
posi	：目標位置［指令単位］
vel	：早送り速度［指令単位／秒］
slope	：加減速モード（０：直線加減速，１：指数関数加減速，２：S 字加減速）
velloop	：速度ループＰ／ＰＩ切替え（０：ＰＩ制御，１：Ｐ制御）

戻り値

正常終了　０

異常終了　－１

解説

引数で指定した速度による目標位置への早送り位置決めを行います。

図 2.6 に示すように、指定した早送り速度で目標位置（P1）に位置決めします。

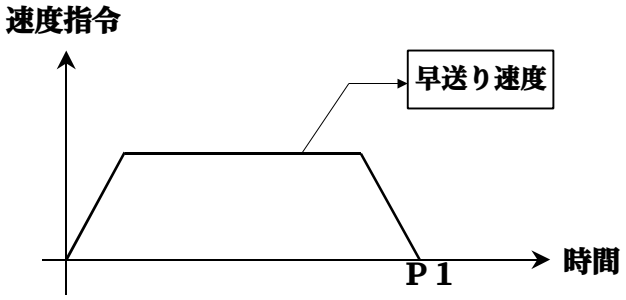

図 2.6 AxNcOp コマンドの速度指令

2.7 外部入力位置決めコマンド	AxNcEx
コマンド分類	モーションコマンド

ＶＣ

```
int AxNcEx(
        int     axis_no,
        long    posi,
        int     vel,
        int     latch
    );
```

ＶＢ

```
Private Declare Function AxNcEx Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal posi As Long, ByVal vel As Long, ByVal latch As Long) As Long
```

引数

axis_no	：制御する軸番号
posi	：目標位置［指令単位］
vel	：早送り速度［指令単位／秒］
latch	：ラッチ信号選択 （０：エンコーダＣ相信号，１：第１外部信号，２：第２外部信号， ３：第３外部信号）

戻り値

正常終了　０
異常終了　－１

解説

早送りを行い、外部位置決め信号の入力によって位置決めを行います。
位置決め動作実行中に指定された外部信号（外部位置決め信号）が入力されると、現在位置カウンタのラッチを行い、その位置からパラメータで設定された外部位置決め走行距離だけ進んだ位置に減速停止して、位置決めを行います。

動作シーケンス

１．一次局は、外部入力位置決め実行中、AxNcEx コマンドを連続して送信します。
２．二次局が AxNcEx コマンドを受信すると、指令された目標位置 P1 に向かって、指令された送り速度で移動を開始します。同時にラッチモードに入ります。
３．一次局は二次局からの AxNcEx の応答でコマンドが受け付けられたことを確認すると、払戻し完了フラグ（DEN）が１となってコマンドの実行が完了するのを監視します。
４．二次局は、外部位置決め信号が入力されると、ラッチ信号データ P2 を作成し、ラッチ完了ステータス（L_CMP）を１として外部位置決め信号が入力されたことを一次局に通知します。

５．ドライブは（目標位置 P3）＝（外部位置決め信号ラッチ位置 P2）＋
（外部位置決め走行距離）を計算し、目標位置 P3 に向けて、位置決めを行います。

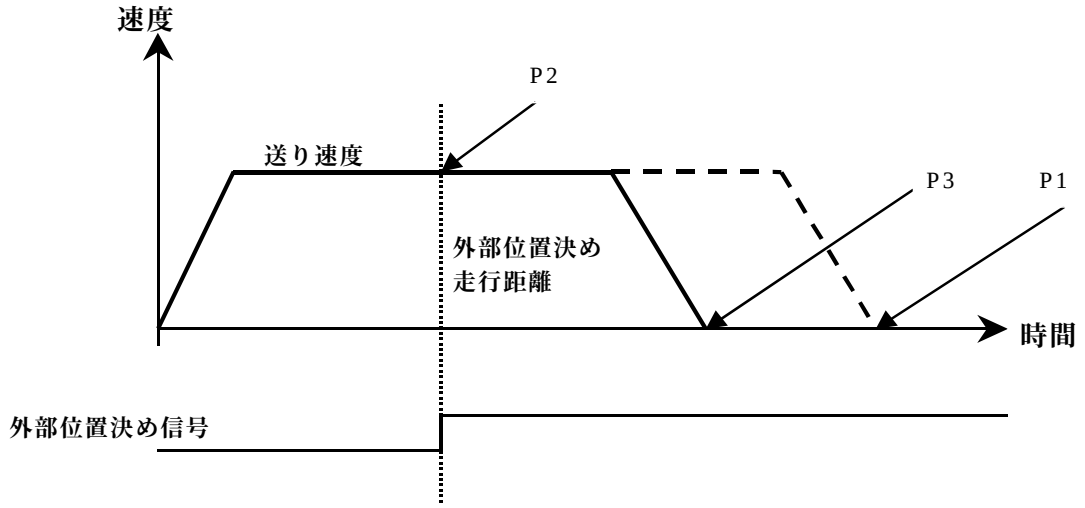

図 2.7　AxNcEx コマンドの速度指令

外部位置決め走行距離で減速できない場合

外部位置決め走行距離が減速停止に必要な距離よりも短い場合、減速パターンに従って目標位置 P3 を越えた位置 P4 に一度減速停止します。

減速停止の払出し完了後、改めて二次局は目標位置 P3 へ位置決め動作を開始します。

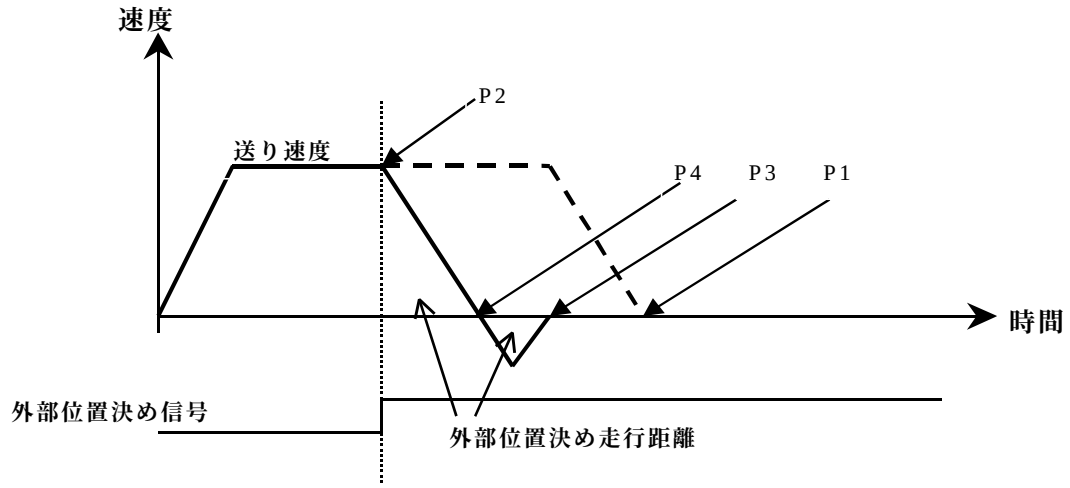

図 2.8 外部位置決め走行距離で減速できない場合の速度指令

補足.

1．コマンドで指定された目標位置 P1 に到達するまでに外部入力位置決め信号が入力されなかった場合、そのまま位置決めを完了します。

2．目標位置 P1 が同一の AxNcEx コマンドを連続送信する場合、新たな AxNcEx コマンドであることを二次局に認識させるため、間に StdtGet などの別のコマンドを送信する必要があります。

2.8 外部入力位置決めコマンド（オプション設定有り）	AxNcExOp
コマンド分類	モーションコマンド

ＶＣ

```
AxNcExOp(
        int     axis_no,
        long    posi,
        int     vel,
        int     slope,
        int     velloop,
        int     latch
       );
```

ＶＢ

```
Private Declare Function AxNcOp Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal posi As Long, ByVal vel As Long, ByVal slope As Long, ByVal velloop
As Long) As Long
```

引数

axis_no	：制御する軸番号
posi	：目標位置［指令単位］
vel	：早送り速度［指令単位／秒］
slope	：加減速モード（０：直線加減速，１：指数関数加減速，２：S 字加減速）
velloop	：速度ループＰ／ＰＩ切替え（０：ＰＩ制御，１：Ｐ制御）
latch	：ラッチ信号選択 （０：エンコーダＣ相信号，１：第１外部信号，２：第２外部信号， ３：第３外部信号）

戻り値

正常終了　０

異常終了　－１

解説

早送りを行い、外部位置決め信号の入力によって位置決めを行います。
位置決め動作実行中に指定された外部信号（外部位置決め信号）が入力されると、現在位置カウンタのラッチを行い、その位置からパラメータで設定された外部位置決め走行距離だけ進んだ位置に減速停止して、位置決めを行います。

動作シーケンス

１．一次局は、外部入力位置決め実行中、AxNcExOp コマンドを連続して送信します。
２．二次局が AxNcExOp コマンドを受信すると、指令された目標位置 P1 に向かって、指令された送り速度で移動を開始します。同時にラッチモードに入ります。

3．一次局は二次局からの AxNcExOp の応答でコマンドが受け付けられたことを確認すると、払戻し完了フラグ（DEN）が 1 となってコマンドの実行が完了するのを監視します。

4．二次局は、外部位置決め信号が入力されると、ラッチ信号データ P2 を作成し、ラッチ完了ステータス（L_CMP）を 1 として外部位置決め信号が入力されたことを一次局に通知します。

5．ドライブは（目標位置 P3）＝（外部位置決め信号ラッチ位置 P2）＋（外部位置決め走行距離）を計算し、目標位置 P3 に向けて、位置決めを行います。

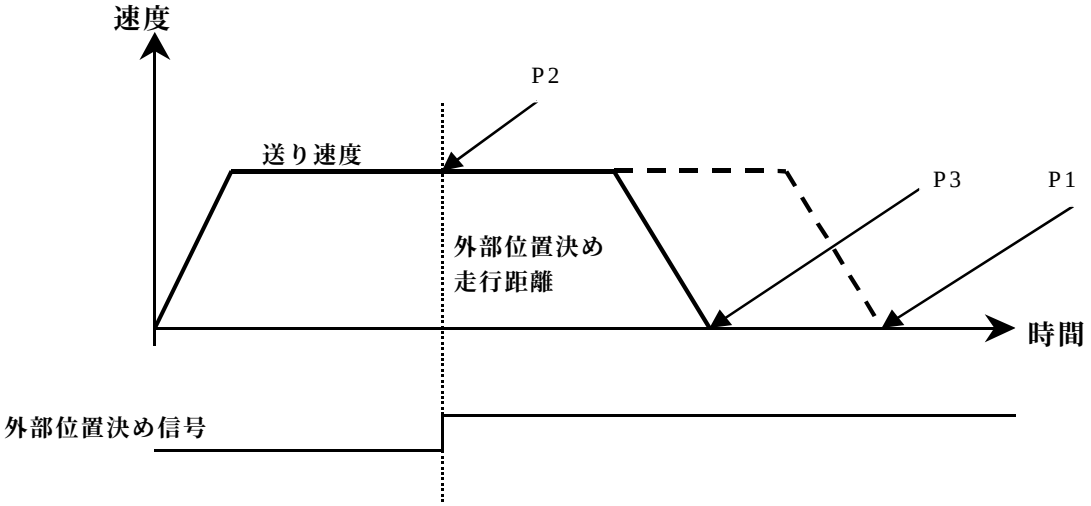

図 2.9　AxNcExOp コマンドの速度指令

外部位置決め走行距離で減速できない場合

外部位置決め走行距離が減速停止に必要な距離よりも短い場合、減速パターンに従って目標位置 P3 を越えた位置 P4 に一度減速停止します。

減速停止の払出し完了後、改めて二次局は目標位置 P3 へ位置決め動作を開始します。

補足.

1．コマンドで指定された目標位置 P1 に到達するまでに外部入力位置決め信号が入力されなかった場合、そのまま位置決めを完了します。

2．目標位置 P1 が同一の AxNcExOp コマンドを連続送信する場合、新たな AxNcExOp コマンドであることを二次局に認識させるため、間に StdtGet などの別のコマンドを送信する必要があります。

2.9 減速停止コマンド	AxDown
コマンド分類	モーションコマンド

ＶＣ

```
int AxDown(
        int     axis_no
);
```

ＶＢ

```
Private Declare Function AxDown Lib "mechad00.dll" (ByVal axis_no As Long)
As Long
```

引数

axis_no　：制御する軸番号
（－１を指定すると現在接続されている局すべてに対してこのコマンドが実行されます。）

戻り値

正常終了　０
異常終了　－１

解説

減速停止を行います。また減速パターンや速度ループ制御は、OcodePut コマンドで設定したものを使用します。

2.10 減速停止コマンド（オプション設定有り）	AxDownOp
コマンド分類	モーションコマンド

ＶＣ

```
int AxDownOp(
        int     axis_no,
        int     slope,
        int     velloop
       );
```

ＶＢ

```
Private Declare Function AxDownOp Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal slope As Long, ByVal velloop As Long) As Long
```

引数

axis_no　：制御する軸番号
（－１を指定すると現在接続されている局すべてに対してこのコマンドが実行されます。）
slope　：加減速モード（０：直線加減速，１：指数関数加減速，２：S 字加減速）
velloop　：速度ループＰ／ＰＩ切替え（０：ＰＩ制御，１：Ｐ制御）

戻り値

正常終了　０
異常終了　－１

解説

指定された減速パターンと速度ループ制御に従って停止する。

2.11 早送り位置決め運転データセットコマンド	AxdtSt
コマンド分類	モーションコマンド

ＶＣ

```
int AxdtSt(
        int     axis_no,
        long    posi,
        int     vel
       );
```

ＶＢ

```
Private Declare Function AxdtSt Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal posi As Long, ByVal vel As Long) As Long
```

引数

axis_no	：制御する軸番号
posi	：目標位置［指令単位］
vel	：早送り速度［指令単位／秒］

戻り値

正常終了　０

異常終了　－１

解説

位置決めを行うために必要なデータをメモリに格納します。

使用例

```
AxdtSt( 0x0001，1000，100 );①
AxdtSt( 0x0002，2000，200 );②
AxdtSt( 0x0003，3000，300 );③
LkStart( 0x0000，0x0007 );④
```

①軸番号 1、目標位置 1000、速度 100 の運転データをメモリ内に格納

②軸番号 2、目標位置 2000、速度 200 の運転データをメモリ内に格納

③軸番号 3、目標位置 3000、速度 300 の運転データをメモリ内に格納

④１，２，３の３軸を同時に位置決め運転開始します。

2.12 早送り位置決め運転データセットコマンド（オプション設定有り）	AxdtStOp
コマンド分類	モーションコマンド

ＶＣ

```
int AxdtStOp(
        int     axis_no,
        long    posi,
        int     vel,
        int     slope,
        int     velloop
       );
```

ＶＢ

Private Declare Function AxdtStOp Lib "mechad00.dll" (ByVal axis_no As Long, ByVal posi As Long, ByVal vel As Long, ByVal slope As Long, ByVal velloop As Long) As Long

引数

axis_no	：制御する軸番号
posi	：目標位置［指令単位］
vel	：早送り速度［指令単位／秒］
slope	：加減速モード（０：直線加減速，１：指数関数加減速，２：S 字加減速）
velloop	：速度ループＰ／ＰＩ切替え（０：ＰＩ制御，１：Ｐ制御）

戻り値

正常終了　０

異常終了　－１

解説

位置決めを行うために必要なデータをメモリに格納します。

使用例

```
AxdtStOp( 0x0001，1000，100，0，0 );①
AxdtStOp( 0x0002，2000，200，0，0 );②
LkStart( 0x0000，0x0003 );③
```

①軸番号 1、目標位置 1000、速度 100、直線加減速、ＰＩ制御の運転データをメモリ内に格納

②軸番号 2、目標位置 2000、速度 200、直線加減速、ＰＩ制御の運転データをメモリ内に格納

③１，２の２軸を同時に位置決め運転開始します。

2.13 早送り位置決め運転データクリアコマンド	AxdtClr
コマンド分類	モーションコマンド

ＶＣ

```
int AxdtClr(
        long    axis_h,
        long    axis_l
      );
```

ＶＢ

```
Private Declare Function AxdtClr Lib "mechad00.dll" (ByVal axis_h As Long,
ByVal axis_l As Long) As Long
```

引数

axis_h　：同時にデータクリアする軸番号（上位４バイト）
axis_l　：同時にデータクリアする軸番号（下位４バイト）
axis_h＋axis_l で同時に処理する軸を指定します。（８バイト分：１～６４軸）
例：軸番号１を指定するとき２進数で表した場合　：0001
軸番号２を指定するとき２進数で表した場合　：0010
軸番号１、２、３を指定するとき２進数で表した場合：0111

戻り値

正常終了　０
異常終了　－１

解説

AxdtSt コマンドによって格納された位置決めデータをメモリから解放します。

使用例

```
AxdtSt( 0x0001，1000，100 );①
AxdtSt( 0x0002，2000，200 );②
AxdtClr( 0x0000，0x0001 );③
AxdtSt( 0x0003，3000，300 );④
LkStart( 0x0000，0x0006 );⑤
```

①軸番号 1、目標位置 1000、速度 100 の運転データをメモリ内に格納
②軸番号 2、目標位置 2000、速度 200 の運転データをメモリ内に格納
③格納されている軸番号１の運転データをクリアします。
④軸番号 3、目標位置 3000、速度 300 の運転データをメモリ内に格納
⑤軸番号２と軸番号 3 を同時に位置決め運転開始します。

2.14 早送り位置決め運転複数軸同時開始コマンド	LkStart
コマンド分類	モーションコマンド

ＶＣ

```
int LkStart(
        long    axis_h,
        long    axis_l
        );
```

ＶＢ

```
Private Declare Function LkStart Lib "mechad00.dll" (ByVal axis_h As Long,
ByVal axis_l As Long) As Long
```

引数

axis_h　：同時スタートする軸番号（上位４バイト）
axis_l　：同時スタートする軸番号（下位４バイト）
axis_h＋axis_l で同時に処理する軸を指定します。（８バイト分：１～６４軸）
例：軸番号１を指定するとき２進数で表した場合　：0001
軸番号２を指定するとき２進数で表した場合　：0010
軸番号１、２、３を指定するとき２進数で表した場合：0111

戻り値

正常終了　０
異常終了　－１

解説

AxdtSt コマンドによって格納された複数個の位置決め運転データを同時に運転開始させます。また、コマンド実行後はメモリ内に格納されたデータが解放されます。

使用例

```
AxdtSt( 0x0001，1000，100 );①
AxdtSt( 0x0002，2000，200 );②
AxdtSt( 0x0003，3000，300 );③
LkStart( 0x0000，0x0007 );④
```

①軸番号 1、目標位置 1000、速度 100 の運転データをメモリ内に格納
②軸番号 2、目標位置 2000、速度 200 の運転データをメモリ内に格納
③軸番号 3、目標位置 3000、速度 300 の運転データをメモリ内に格納
④１，２，３の３軸を同時に位置決め運転開始します。

2.15 補間送り運転データセットコマンド	IpdtSet
コマンド分類	モーションコマンド

ＶＣ

```
int IpdtSet(
        int     axis_no[15],
        long    posi[15]
       );
```

ＶＢ

```
Private Declare Function IpdtSet Lib "mechad00.dll" (axis_no As Long,
posi_no As Long) As Long
```

引数

axis_no[15]　：制御する軸番号の配列
有効にする局に軸番号を入力します。
（複数軸指定するため、１軸目を１として軸番号順に１ビットづつ左シフトしていった値を軸番号に使用します。）

posi[15]　：補間位置データの配列［指令単位］

戻り値

正常終了　０
異常終了　－１

解説

補間送りを行うために必要なデータをデバイスドライバ内でメモリに格納します。
（格納データ：１５軸分の軸番号，補間位置）

使用例

例１

```
int         axis[15] =  { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                         ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                         ,0x1000,0x2000,0x4000};①
long        posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpdtSet( axis,posi );③
for(int i=0; i<15; i++ )posi[i] = 20;④
IpdtSet( axis,posi );⑤
IpStrt( 0x0000，0x7fff );⑥
IpStop( 0x0000，0x7fff );⑦
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［指令単位］まで移動させるためのデータ）
③１～１５軸を通信周期内に 10［指令単位］まで移動させるためのデータを
メモリ内にセット。
④１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
配列内にセット。
⑤１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
メモリ内にセット。
⑥補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、
次の通信周期で１～１５軸を同時に 20 へ移動します。
⑦補間運転を停止します。

例２

```
int          axis[15] =   { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                           ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                           ,0x1000,0x2000,0x4000};①
long         posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpdtSet( axis,posi );③
for(int i=0; i<15; i++ )posi[i] = 20;④
IpdtSet( axis,posi );⑤
IpStrt( 0x0000，0x7fff );⑥
for(int i=0; i<15; i++ )posi[i] = 30;⑦
IpdtSet( axis,posi );⑧
IpStop( 0x0000，0x7fff );⑨
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［指令単位］まで移動させるためのデータ）
③１～１５軸を通信周期内に 10［指令単位］まで移動させるためのデータをメモリ内にセット。
④１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを配列内にセット。
⑤１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータをメモリ内にセット。
⑥補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、次の通信周期で１～１５軸を同時に 20 へ移動します。
⑦１～１５軸を通信周期内に 30［指令単位］まで移動させるためのデータを配列内にセット。
⑧１～１５軸を通信周期内に 30［指令単位］まで移動させるためのデータをメモリ内にセット。
⑨補間運転を停止します。

例３

```
int             axis[15] =    { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                              ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                              ,0x1000,0x2000,0x4000};①
long            posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpStrt( 0x0000，0x7fff );③
IpdtSet( axis,posi );④
for(int i=0; i<15; i++ )posi[i] = 20;⑤
IpdtSet( axis,posi );⑥
for(int i=0; i<15; i++ )posi[i] = 30;⑦
IpdtSet( axis,posi );⑧
IpStop( 0x0000，0x7fff );⑨
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［指令単位］まで移動させるためのデータ）
③補間運転を開始します。
④１～１５軸を通信周期内に 10［指令単位］まで移動します。
⑤１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを配列内にセット。
⑥１～１５軸を通信周期内に 20［指令単位］まで移動します。
⑦１～１５軸を通信周期内に 30［指令単位］まで移動させるためのデータを配列内にセット。
⑧１～１５軸を通信周期内に 30［指令単位］まで移動します。
⑨補間運転を停止します。

2.16 補間送り運転開始コマンド	IpStrt
コマンド分類	モーションコマンド

ＶＣ

```
int IpStrt(
        long    axis_h,
        long    axis_l
      );
```

ＶＢ

```
Private Declare Function IpStrt Lib "mechad00.dll" (ByVal axis_h As Long,
ByVal axis_l As Long) As Long
```

引数

axis_h　：同時スタートする軸番号（上位４バイト）
axis_l　：同時スタートする軸番号（下位４バイト）
axis_h＋axis_l で同時に処理する軸を指定します。（８バイト分：１～６４軸）
例：軸番号１を指定するとき２進数で表した場合　：0001
　　軸番号２を指定するとき２進数で表した場合　：0010
　　軸番号１、２、３を指定するとき２進数で表した場合：0111

戻り値

正常終了　０
異常終了　－１

解説

補間送りを開始します。
運転中使用されたデータはメモリ上から解放されていきます。

使用例

例１

```
int         axis[15] =  { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                         ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                         ,0x1000,0x2000,0x4000};①
long        posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpdtSet( axis,posi );③
for(int i=0; i<15; i++ )posi[i] = 20;④
IpdtSet( axis,posi );⑤
IpStrt( 0x0000，0x7fff );⑥
IpStop( 0x0000，0x7fff );⑦
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［指令単位］まで移動させるためのデータ）
③１～１５軸を通信周期内に 10［指令単位］まで移動させるためのデータを
メモリ内にセット。
④１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
配列内にセット。
⑤１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
メモリ内にセット。
⑥補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、
次の通信周期で１～１５軸を同時に 20 へ移動します。
⑦補間運転を停止します。

例２

```
int          axis[15] =   { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                           ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                           ,0x1000,0x2000,0x4000};①
long         posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpdtSet( axis,posi );③
for(int i=0; i<15; i++ )posi[i] = 20;④
IpdtSet( axis,posi );⑤
IpStrt( 0x0000，0x7fff );⑥
for(int i=0; i<15; i++ )posi[i] = 30;⑦
IpdtSet( axis,posi );⑧
IpStop( 0x0000，0x7fff );⑨
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［指令単位］まで移動させるためのデータ）
③１～１５軸を通信周期内に 10［指令単位］まで移動させるためのデータをメモリ内にセット。
④１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを配列内にセット。
⑤１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータをメモリ内にセット。
⑥補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、次の通信周期で１～１５軸を同時に 20 へ移動します。
⑦１～１５軸を通信周期内に 30［指令単位］まで移動させるためのデータを配列内にセット。
⑧１～１５軸を通信周期内に 30［指令単位］まで移動させるためのデータをメモリ内にセット。
⑨補間運転を停止します。

例３

```
int             axis[15] =    { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                              ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                              ,0x1000,0x2000,0x4000};①
long            posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpStrt( 0x0000，0x7fff );③
IpdtSet( axis,posi );④
for(int i=0; i<15; i++ )posi[i] = 20;⑤
IpdtSet( axis,posi );⑥
for(int i=0; i<15; i++ )posi[i] = 30;⑦
IpdtSet( axis,posi );⑧
IpStop( 0x0000，0x7fff );⑨
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［指令単位］まで移動させるためのデータ）
③補間運転を開始します。
④１～１５軸を通信周期内に 10［指令単位］まで移動します。
⑤１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを配列内にセット。
⑥１～１５軸を通信周期内に 20［指令単位］まで移動します。
⑦１～１５軸を通信周期内に 30［指令単位］まで移動させるためのデータを配列内にセット。
⑧１～１５軸を通信周期内に 30［指令単位］まで移動します。
⑨補間運転を停止します。

2.17 補間送り運転開始コマンド（データ保持）	IpStrtRp
コマンド分類	モーションコマンド

ＶＣ

```
int IpStrtRp(
        long    axis_h,
        long    axis_l
      );
```

ＶＢ

```
Private Declare Function IpStrtRp Lib "mechad00.dll" (ByVal axis_h As Long,
ByVal axis_l As Long) As Long
```

引数

axis_h　：同時スタートする軸番号（上位４バイト）
axis_l　：同時スタートする軸番号（下位４バイト）
axis_h＋axis_l で同時に処理する軸を指定します。（８バイト分：１～６４軸）
例：軸番号１を指定するとき２進数で表した場合　：0001
　　軸番号２を指定するとき２進数で表した場合　：0010
　　軸番号１、２、３を指定するとき２進数で表した場合：0111

戻り値

正常終了　０
異常終了　－１

解説

補間送りを開始します。運転中使用されたデータはメモリ上から解放されません。そのため、運転終了した後にこのコマンドを再発行すると同じ内容の補間運転がおこなえます。
メモリ上に格納された運転データの解放は IpdtClr コマンド（見出し番号 2.19）でおこないます。

使用例　　例１

```
int          axis[15] =   { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                           ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                           ,0x1000,0x2000,0x4000};①
long         posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpdtSet( axis,posi );③
for(int i=0; i<15; i++ )posi[i] = 20;④
IpdtSet( axis,posi );⑤
IpStrtRp( 0x0000，0x7fff );⑥
IpStop( 0x0000，0x7fff );⑦
IpStrt( 0x0000，0x7fff );⑧
IpStop( 0x0000，0x7fff );⑨
IpdtClr( 0x0000，0x7fff );⑩
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［指令単位］まで移動させるためのデータ）
③１～１５軸を通信周期内に 10［指令単位］まで移動させるためのデータを
メモリ内にセット。
④１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
配列内にセット。
⑤１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
メモリ内にセット。
⑥補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、
次の通信周期で１～１５軸を同時に 20 へ移動します。
⑦補間運転を停止します。
⑧補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、
次の通信周期で１～１５軸を同時に 20 へ移動します。
⑨補間運転を停止します。
⑩IpdtSet コマンドでデバイスドライバ内メモリ上に格納された補間運転データを
解放します。

例２

```
int             axis[15] =    { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                              ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                              ,0x1000,0x2000,0x4000};①
long            posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpdtSet( axis,posi );③
for(int i=0; i<15; i++ )posi[i] = 20;④
IpdtSet( axis,posi );⑤
IpStrtRp( 0x0000，0x7fff );⑥
for(int i=0; i<15; i++ )posi[i] = 30;⑦
IpdtSet( axis,posi );⑧
IpStop( 0x0000，0x7fff );⑨
IpStrtRp( 0x0000，0x7fff );⑩
IpStop( 0x0000，0x7fff );⑪
IpdtClr( 0x0000，0x7fff );⑫
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［指令単位］まで移動させるためのデータ）
③１～１５軸を通信周期内に 10［指令単位］まで移動させるためのデータを
メモリ内にセット。
④１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
配列内にセット。
⑤１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
メモリ内にセット。
⑥補間運転を開始します。
通信周期内に１～１５軸を同時に 10 へ移動し、
次の通信周期で１～１５軸を同時に 20 へ移動します。
⑦１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
配列内にセット。
⑧１～１５軸を通信周期内に 30［指令単位］まで移動します。
⑨補間運転を停止します。
⑩補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、
次の通信周期に 20 へ移動し、その次の通信周期で 30 へ移動します。
⑪補間運転を停止します。
⑫IpdtSet コマンドでデバイスドライバ内メモリ上に格納された補間運転データを
解放します。

例３

```
int             axis[15] =    { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                              ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                              ,0x1000,0x2000,0x4000};①
long            posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpStrtRp( 0x0000，0x7fff );③
IpdtSet( axis,posi );④
for(int i=0; i<15; i++ )posi[i] = 20;⑤
IpdtSet( axis,posi );⑥
for(int i=0; i<15; i++ )posi[i] = 30;⑦
IpdtSet( axis,posi );⑧
IpStop( 0x0000，0x7fff );⑨
IpStrtRp( 0x0000，0x7fff );③
IpStop( 0x0000，0x7fff );⑨
IpdtClr( 0x0000，0x7fff );⑩
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［**指令単位**］**まで移動させるためのデータ）**
③補間運転を開始します。
④１～１５軸を通信周期内に 10［**指令単位**］**まで移動します。**
⑤１～１５軸を通信周期内に 20［**指令単位**］**まで移動させるためのデータを配列内にセット。**
⑥１～１５軸を通信周期内に 20［**指令単位**］**まで移動します。**
⑦１～１５軸を通信周期内に 30［**指令単位**］**まで移動させるためのデータを配列内にセット。**
⑧１～１５軸を通信周期内に 30［**指令単位**］**まで移動します。**
⑨補間運転を停止します。
⑩IpdtSet **コマンドでデバイスドライバ内メモリ上に格納された補間運転データを解放します。**

2.18 補間送り運転停止コマンド	IpStop
コマンド分類	モーションコマンド

ＶＣ

```
int IpStop(
        long   axis_h,
        long   axis_l
       );
```

ＶＢ

```
Private Declare Function IpStop Lib "mechad00.dll" (ByVal axis_h As Long,
ByVal axis_l As Long) As Long
```

引数

axis_h　：同時に停止する軸番号（上位４バイト）
axis_l　：同時に停止する軸番号（下位４バイト）
axis_h＋axis_l で同時に処理する軸を指定します。（８バイト分：１～６４軸）
例：軸番号１を指定するとき２進数で表した場合　：0001
軸番号２を指定するとき２進数で表した場合　：0010
軸番号１、２、３を指定するとき２進数で表した場合：0111

戻り値

正常終了　０
異常終了　－１

解説

このコマンドを発行した時点でメモリ上に格納されているデータを補間送りした後、IpdtSet コマンドによるデータセットをおこなっても補間移動が実行されないようになります。
再度、補間送りをおこなうときは IpStrt コマンドか IpStrtRp を発行することでおこないます。

使用例　　例１

```
int          axis[15] =   { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                           ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                           ,0x1000,0x2000,0x4000};①
long         posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpdtSet( axis,posi );③
for(int i=0; i<15; i++ )posi[i] = 20;④
IpdtSet( axis,posi );⑤
IpStrtRp( 0x0000，0x7fff );⑥
IpStop( 0x0000，0x7fff );⑦
IpStrt( 0x0000，0x7fff );⑧
IpStop( 0x0000，0x7fff );⑨
IpdtClr( 0x0000，0x7fff );⑩
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［指令単位］まで移動させるためのデータ）
③１～１５軸を通信周期内に 10［指令単位］まで移動させるためのデータをメモリ内にセット。
④１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを配列内にセット。
⑤１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータをメモリ内にセット。
⑥補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、次の通信周期で１～１５軸を同時に 20 へ移動します。
⑦補間運転を停止します。
⑧補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、次の通信周期で１～１５軸を同時に 20 へ移動します。
⑨補間運転を停止します。
⑩IpdtSet コマンドでデバイスドライバ内メモリ上に格納された補間運転データを解放します。

例２

```
int             axis[15] =    { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                              ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                              ,0x1000,0x2000,0x4000};①
long            posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpdtSet( axis,posi );③
for(int i=0; i<15; i++ )posi[i] = 20;④
IpdtSet( axis,posi );⑤
IpStrtRp( 0x0000，0x7fff );⑥
for(int i=0; i<15; i++ )posi[i] = 30;⑦
IpdtSet( axis,posi );⑧
IpStop( 0x0000，0x7fff );⑨
IpStrtRp( 0x0000，0x7fff );⑩
IpStop( 0x0000，0x7fff );⑪
IpdtClr( 0x0000，0x7fff );⑫
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［**指令単位**］**まで移動させるためのデータ）**
③１～１５軸を通信周期内に 10［**指令単位**］**まで移動させるためのデータをメモリ内にセット。**
④１～１５軸を通信周期内に 20［**指令単位**］**まで移動させるためのデータを配列内にセット。**
⑤１～１５軸を通信周期内に 20［**指令単位**］**まで移動させるためのデータをメモリ内にセット。**
⑥補間運転を開始します。
通信周期内に１～１５軸を同時に 10 **へ移動し、次の通信周期で１～１５軸を同時に** 20 **へ移動します。**
⑦１～１５軸を通信周期内に 20［**指令単位**］**まで移動させるためのデータを配列内にセット。**
⑧１～１５軸を通信周期内に 30［**指令単位**］**まで移動します。**
⑨補間運転を停止します。
⑩補間運転を開始します。通信周期内に１～１５軸を同時に 10 **へ移動し、次の通信周期に** 20 **へ移動し、その次の通信周期で** 30 **へ移動します。**
⑪補間運転を停止します。
⑫IpdtSet **コマンドでデバイスドライバ内メモリ上に格納された補間運転データを解放します。**

例３

```
int             axis[15] =    { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                              ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                              ,0x1000,0x2000,0x4000};①
long            posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpStrtRp( 0x0000，0x7fff );③
IpdtSet( axis,posi );④
for(int i=0; i<15; i++ )posi[i] = 20;⑤
IpdtSet( axis,posi );⑥
for(int i=0; i<15; i++ )posi[i] = 30;⑦
IpdtSet( axis,posi );⑧
IpStop( 0x0000，0x7fff );⑨
IpStrtRp( 0x0000，0x7fff );③
IpStop( 0x0000，0x7fff );⑨
IpdtClr( 0x0000，0x7fff );⑩
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［**指令単位**］**まで移動させるためのデータ）**
③補間運転を開始します。
④１～１５軸を通信周期内に 10［**指令単位**］**まで移動します。**
⑤１～１５軸を通信周期内に 20［**指令単位**］**まで移動させるためのデータを配列内にセット。**
⑥１～１５軸を通信周期内に 20［**指令単位**］**まで移動します。**
⑦１～１５軸を通信周期内に 30［**指令単位**］**まで移動させるためのデータを配列内にセット。**
⑧１～１５軸を通信周期内に 30［**指令単位**］**まで移動します。**
⑨補間運転を停止します。
⑩IpdtSet **コマンドでデバイスドライバ内メモリ上に格納された補間運転データを解放します。**

2.19 補間送り運転データ解放コマンド	IpdtClr
コマンド分類	モーションコマンド

ＶＣ

```
int IpdtClr(
        long    axis_h,
        long    axis_l
      );
```

ＶＢ

```
Private Declare Function IpdtClr Lib "mechad00.dll" (ByVal axis_h As Long,
ByVal axis_l As Long) As Long
```

引数

axis_h　：同時にデータクリアする軸番号（上位４バイト）
axis_l　：同時にデータクリアする軸番号（下位４バイト）
axix_h＋axix_l で同時に処理する軸を指定します。（８バイト分：１～６４軸）
例：軸番号１を指定するとき２進数で表した場合　：0001
　　軸番号２を指定するとき２進数で表した場合　：0010
　　軸番号１、２、３を指定するとき２進数で表した場合：0111

戻り値

正常終了　０
異常終了　－１

解説

IpdtSet コマンドでデバイスドライバ内でメモリ上に格納された補間運転データを解放します。

使用例

例１

```
int         axis[15] =  { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                         ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                         ,0x1000,0x2000,0x4000};①
long        posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpdtSet( axis,posi );③
for(int i=0; i<15; i++ )posi[i] = 20;④
IpdtSet( axis,posi );⑤
IpStrtRp( 0x0000，0x7fff );⑥
IpStop( 0x0000，0x7fff );⑦
IpStrt( 0x0000，0x7fff );⑧
IpStop( 0x0000，0x7fff );⑨
IpdtClr( 0x0000，0x7fff );⑩
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［指令単位］まで移動させるためのデータ）
③１～１５軸を通信周期内に 10［指令単位］まで移動させるためのデータをメモリ内にセット。
④１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを配列内にセット。
⑤１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータをメモリ内にセット。
⑥補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、次の通信周期で１～１５軸を同時に 20 へ移動します。
⑦補間運転を停止します。
⑧補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、次の通信周期で１～１５軸を同時に 20 へ移動します。
⑨補間運転を停止します。
⑩IpdtSet コマンドでデバイスドライバ内メモリ上に格納された補間運転データを解放します。

例２

```
int             axis[15] =    { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                              ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                              ,0x1000,0x2000,0x4000};①
long            posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpdtSet( axis,posi );③
for(int i=0; i<15; i++ )posi[i] = 20;④
IpdtSet( axis,posi );⑤
IpStrtRp( 0x0000，0x7fff );⑥
for(int i=0; i<15; i++ )posi[i] = 30;⑦
IpdtSet( axis,posi );⑧
IpStop( 0x0000，0x7fff );⑨
IpStrtRp( 0x0000，0x7fff );⑩
IpStop( 0x0000，0x7fff );⑪
IpdtClr( 0x0000，0x7fff );⑫
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［指令単位］まで移動させるためのデータ）
③１～１５軸を通信周期内に 10［指令単位］まで移動させるためのデータを
メモリ内にセット。
④１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
配列内にセット。
⑤１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
メモリ内にセット。
⑥補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、
次の通信周期で１～１５軸を同時に 20 へ移動します。
⑦１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
配列内にセット。
⑧１～１５軸を通信周期内に 30［指令単位］まで移動します。
⑨補間運転を停止します。
⑩補間運転を開始します。通信周期内に１～１５軸を同時に 10 へ移動し、
次の通信周期に 20 へ移動し、その次の通信周期で 30 へ移動します。
⑪補間運転を停止します。
⑫IpdtSet コマンドでデバイスドライバ内メモリ上に格納された補間運転データを
解放します。

例３

```
int             axis[15] =    { 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020
                              ,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800
                              ,0x1000,0x2000,0x4000};①
long            posi[15]={10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,};②
IpStrtRp( 0x0000，0x7fff );③
IpdtSet( axis,posi );④
for(int i=0; i<15; i++ )posi[i] = 20;⑤
IpdtSet( axis,posi );⑥
for(int i=0; i<15; i++ )posi[i] = 30;⑦
IpdtSet( axis,posi );⑧
IpStop( 0x0000，0x7fff );⑨
IpStrtRp( 0x0000，0x7fff );③
IpStop( 0x0000，0x7fff );⑨
IpdtClr( 0x0000，0x7fff );⑩
```

①制御する軸番号のテーブル宣言と初期データ設定（１－１５軸有効）
②補間位置のテーブル宣言と初期データ設定
（１－１５軸を通信周期内に 10［指令単位］まで移動させるためのデータ）
③補間運転を開始します。
④１～１５軸を通信周期内に 10［指令単位］まで移動します。
⑤１～１５軸を通信周期内に 20［指令単位］まで移動させるためのデータを
配列内にセット。
⑥１～１５軸を通信周期内に 20［指令単位］まで移動します。
⑦１～１５軸を通信周期内に 30［指令単位］まで移動させるためのデータを
配列内にセット。
⑧１～１５軸を通信周期内に 30［指令単位］まで移動します。
⑨補間運転を停止します。
⑩IpdtSet コマンドでデバイスドライバ内メモリ上に格納された補間運転データを
解放します。

2.2.3. 制御コマンドグループ

機器の状態制御に使用するコマンドについて説明。

3.1 非常停止コマンド	AxEmg
コマンド分類	制御コマンド

ＶＣ

```
int AxEmg(
        int    axis_no
      );
```

ＶＢ

```
Private Declare Function AxEmg Lib "mechad00.dll" (ByVal axis_no As Long)
As Long
```

引数

axis_no　：制御する軸番号
（－１を指定すると現在接続されている局すべてに対してこのコマンドが実行されます。）

戻り値

正常終了　０
異常終了　－１

解説

非常停止コマンドです。ブレーキ作動信号を出力します。

3.2 非常停止解除コマンド	AxEmgOff
コマンド分類	制御コマンド

ＶＣ

```
int AxEmgOff(
                int     axis_no
              );
```

ＶＢ

```
Private Declare Function AxEmgOff Lib "mechad00.dll" (ByVal axis_no As Long)
As Long
```

引数　axis_no　：制御する軸番号

戻り値　正常終了　０
異常終了　－１

解説　非常停止解除コマンドです。ブレーキ解除信号を出力します。

3.3 ボード選択コマンド	InitDll
コマンド分類	制御コマンド

ＶＣ

```
int InitDll(
        int     Kind_of_Board
    );
```

ＶＢ

```
Private Declare Function InitDll Lib "mechad00.dll" (ByVal Kind_of_Board As Long
) As Long
```

引数　Kind_of_Board　：デバイス選択コード

デバイス選択コード

デバイス選択コード	ボード名
1	ＩＴ－３２０５

戻り値

正常終了　０
異常終了　－１

解説

Ｄｌｌの動作が、どの種類のボードに対応させるかを設定します。
本コマンドは BdOpen コマンドの前に必ず行うようにしてください。

使用例

```
InitDll( 1 );                   //ＩＴ－３２０５を使用するよう設定
BdOpen( 1, 20, 0x00, add );
```

3.4 ボード初期設定コマンド	BdOpen
コマンド分類	制御コマンド

ＶＣ

```
int BdOpen(
        int     bd_no,
        int     cc,
        int     int_no,
        int     add[15]
    );
```

ＶＢ

```
Private Declare Function BdOpen Lib "mechad00.dll" (ByVal bdno As Long,
ByVal cc As Long, ByVal intno As Long, add As Long) As Long
```

引数

bd_no　：利用するボードに割り当てたボード番号
cc　：通信周期分解能（２０：2ｍｓ，１０：1ｍｓ，５：０．５ｍｓ）
Int_no　：通信周期定数（０～１５） 通信周期＝(通信周期定数 + 1)× 2ｍｓ
通信周期の概要ついては「IT－3205DLL ユーザーズマニュアル」の
見出し番号 7 用語解説の通信周期を参照ください。
add[15]　：配列 add の添字０～１４が１～１５局に対応しています。それぞれの局に対応したサーボパックのロータリスイッチの値を設定します。（サーボパックを接続しない局には０を設定）サーボパックの局設定用ロータリスイッチの値（１局目なら 65 等）を設定することで、そのサーボパックが１局として設定されます。

戻り値

正常終了　０
異常終了　－１

解説

デバイスドライバのオープン、レジスタの初期化、通信領域のクリア、通信周期の設定、通信の開始を行います。

使用例

```
int         add[15] = {65,66,0,0,0,0,0,0,0,0,0,0,0,0,0};
BdOpen( 1, 20, 0x00, add );
```

通信周期を２ｍｓに設定
1 局目 接続 ロータリスイッチの値 65（１６進数：41h）
2 局目 接続 ロータリスイッチの値 66（１６進数：42h）
3 局目～１５局目 未接続

3.5 終了処理コマンド	BdClose
コマンド分類	制御コマンド

ＶＣ

```
int BdClose(
        int     bd_no
);
```

ＶＢ

```
Private Declare Function BdClose Lib "mechad00.dll" (ByVal bd_no As Long)
As Long
```

引数　bd_no　：利用するボードに割り当てたボード番号

戻り値　正常終了　０
異常終了　－１

解説　対応する制御ボードのデバイスドライバのクローズ、通信の停止を行います。

3.6 サーボオンコマンド	SvOn
コマンド分類	制御コマンド

ＶＣ

```
int SvOn(
        int     axis_no
       );
```

ＶＢ

```
Private Declare Function SvOn Lib "mechad00.dll" (ByVal axis_no As Long)
As Long
```

引数　axis_no　：制御する軸番号

戻り値　正常終了　０
異常終了　－１

解説　サーボオンをおこないます。

3.7 サーボオフコマンド	SvOff
コマンド分類	制御コマンド

ＶＣ

```
int SvOff(
        int     axis_no
      );
```

ＶＢ

```
Private Declare Function SvOff Lib "mechad00.dll" (ByVal axis_no As Long)
As Long
```

引数　axis_no　：制御する軸番号

戻り値　正常終了　０
異常終了　－１

解説　サーボオフを行います。

3.8 座標系設定コマンド	PosiPut
コマンド分類	制御コマンド

ＶＣ

```
int PosiPut(
        int     axis_no,
        int     posi_no,
        long    posi
       );
```

ＶＢ

```
Private Declare Function PosiPut Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal prm_no As Long, ByVal posi As Long) As Long
```

引数

axis_no　：制御する軸番号
posi_no　：座標系選択コード
「IT－3205DLL ユーザーズマニュアル」の
見出し番号 4 モニタ選択コード一覧表を参照ください。
posi　：位置データ［指令単位］

戻り値

正常終了　０
異常終了　－１

解説

座標系設定を行います。座標値の種類は座標系選択コードで指定します。

3.9 アラームクリアコマンド	AlmClr
コマンド分類	制御コマンド

ＶＣ

```
int AlmClr(
        int    axis_no
);
```

ＶＢ

```
Private Declare Function AlmClr Lib "mechad00.dll" (ByVal axis_no As Long)
As Long
```

引数　axis_no　：制御する軸番号

戻り値　正常終了　０
異常終了　－１

解説　異常状態，警告状態を解除します。

3.10 手動パルサの設定コマンド	AxHndPls
コマンド分類	制御コマンド

VC

```
int AxHndPls(
        int     axis_no,
        int     mode,
        int     magnification
       );
```

VB

```
Private Declare Function AxHndPls Lib "mechad00.dll" (ByVal axis_no As Long,
ByVal mode As Long, ByVal magnification As Long) As Long
```

引数

axis_no	：制御する軸番号
mode	：手動パルサ無効 0，手動パルサ有効 1
magnification	：移動倍率［倍］

戻り値

正常終了　0
異常終了　－1

解説

手動パルサの有効・無効と移動倍率を設定します。

3.11 エンコーダオンコマンド	EncdrOn
コマンド分類	制御コマンド

ＶＣ

```
int EncdrOn(
        int     axis_no
);
```

ＶＢ

```
Private Declare Function EncdrOn Lib "mechad00.dll" (ByVal axis_no As Long)
As Long
```

引数　axis_no　：制御する軸番号

戻り値　正常終了　０
異常終了　－１

解説　エンコーダの情報初期化要求コマンドです。エンコーダの電源オンと初期化を行います。

3.12 エンコーダオフコマンド	EncdrOff
コマンド分類	制御コマンド

ＶＣ

```
int EncdrOff(
        int     axis_no
);
```

ＶＢ

```
Private Declare Function EncdrOff Lib "mechad00.dll" (ByVal axis_no As Long)
As Long
```

引数　axis_no　：制御する軸番号

戻り値　正常終了　０
異常終了　－１

解説　エンコーダの電源オフ要求コマンドです。エンコーダの電源をオフします。

3.13 マシンロック要求コマンド	MlockOn
コマンド分類	制御コマンド

ＶＣ

```
int MlockOn(
        int     axis_no
);
```

ＶＢ

```
Private Declare Function MlockOn Lib "mechad00.dll" (ByVal axis_no As Long)
As Long
```

引数　axis_no　：制御する軸番号

戻り値　正常終了　０
異常終了　－１

解説　機器をマシンロックモードに設定します。マシンロックとは、機会の駆動系と移動指令系統を切り放した状態を意味します。マシンロック中、移動指令系統上の位置は更新されますが、機械の物理位置は変化しません。

3.14 マシンロック解除要求コマンド	MlockOff
コマンド分類	制御コマンド

ＶＣ

```
int MlockOff(
        int     axis_no
);
```

ＶＢ

```
Private Declare Function MlockOff Lib "mechad00.dll" (ByVal axis_no As Long)
As Long
```

引数　axis_no　：制御する軸番号

戻り値　正常終了　０
異常終了　－１

解説　機器のマシンロックモードを解除します。マシンロック解除は、クラッチの接続イメージで、移動指令系統と駆動系の接続が行われます。マシンロック中に発生した指令系座標値と機械の物理位置のずれはオフセットとして残ります。オフセットの解消には座標系設定が必要となります。

3.15 機器セットアップコマンド	Config
コマンド分類	制御コマンド

ＶＣ

```
int Config(
        int     axis_no
       );
```

ＶＢ

```
Private Declare Function Config Lib "mechad00.dll" (ByVal axis_no As Long)
As Long
```

引数

axis_no　：制御する軸番号

戻り値

正常終了　０

異常終了　－１

解説

ΣｎシリーズＳＶＲ

現在の設定での機器のセットアップを行います。

このコマンドで再設定されるパラメータは以下の３定数です。

Pn-0001：メモリスイッチ１

Pn-0002：メモリスイッチ２

Pn-0017：エンコーダパルス数

なお、この３定数以外のパラメータは書き込みコマンド（PrmPut，PPrmPut）の実行のみで値が反映されます。

ΣⅡシリーズＳＶＲ

現在設定されているすべてのパラメータを再計算し、位置、信号などを初期化します。実行終了までに約４ｓｅｃかかります。

3.16 ＡＢＳエンコーダ初期化コマンド	AbsIni
コマンド分類	制御コマンド

ＶＣ

```
int AbsIni(
        int         axis_no
    );
```

ＶＢ

```
Private Declare Function AbsIni Lib "mechad00.dll" (ByVal axis_no As Long
) As Long
```

引数

axis_no　：制御する軸番号

戻り値

正常終了　０
異常終了　－１

解説

ＡＢＳエンコーダの初期化を行います。本コマンド実行後にサーボパックの電源を再起動することでＡＢＳエンコーダが使用可能となります。

3.17 ＤＩデータ読込みコマンド	Din
コマンド分類	制御コマンド

ＶＣ

```
int Din(
        int     iounit_no,
        int     bitblock_no,
        char    bit[16]
       );
```

ＶＢ

```
Private Declare Function Din Lib "mechad00.dll" (ByVal iounit_no As Long,
ByVal bitblock_no As Long, bit As Long) As Long
```

引数

iounit_no	：読込む局の局番号
bitblock_no	：読込むビットの区画番号（１～８） １区画につき１６ビット幅の値を読込みます。 区画番号のビット配置詳細については、付録 A.1 Ｉ／Ｏコマンドの区画配置図を参照ください。
bit[0]	：指定したビット領域の先頭ビットの値
bit[1]	：指定したビット領域の先頭ビット＋1 ビットの値
bit[2]	：指定したビット領域の先頭ビット＋2 ビットの値
bit[3]	：指定したビット領域の先頭ビット＋3 ビットの値
bit[4]	：指定したビット領域の先頭ビット＋4 ビットの値
bit[5]	：指定したビット領域の先頭ビット＋5 ビットの値
bit[6]	：指定したビット領域の先頭ビット＋6 ビットの値
bit[7]	：指定したビット領域の先頭ビット＋7 ビットの値
bit[8]	：指定したビット領域の先頭ビット＋8 ビットの値
bit[9]	：指定したビット領域の先頭ビット＋9 ビットの値
bit[10]	：指定したビット領域の先頭ビット＋10 ビットの値
bit[11]	：指定したビット領域の先頭ビット＋11 ビットの値
bit[12]	：指定したビット領域の先頭ビット＋12 ビットの値
bit[13]	：指定したビット領域の先頭ビット＋13 ビットの値
bit[14]	：指定したビット領域の先頭ビット＋14 ビットの値
bit[15]	：指定したビット領域の先頭ビット＋15 ビットの値

戻り値

正常終了　　０

異常終了　－１

解説

指定した区画のビット情報を読込みます。

3.18 DO データ書込みコマンド	Dout
コマンド分類	制御コマンド

VC

```
Dout(
        int     iounit_no,
        int     bitblock_no,
        char    bit[16]
    );
```

VB

```
Private Declare Function Dout Lib "mechad00.dll" (ByVal iounit_no As Long,
ByVal bitblock_no As Long, bit As Long) As Long
```

引数

iounit_no	：書込む局の局番号
bitblock_no	：読込むビットの区画番号（１～８） １区画につき１６ビット幅の値を読込みます。 区画番号のビット配置詳細については、付録 A.1 Ｉ／Ｏコマンドの区画配置図を参照ください。
bit[0]	：指定したビット領域の先頭ビットへ書込む値
bit[1]	：指定したビット領域の先頭ビット＋1 ビットへ書込む値
bit[2]	：指定したビット領域の先頭ビット＋2 ビットへ書込む値
bit[3]	：指定したビット領域の先頭ビット＋3 ビットへ書込む値
bit[4]	：指定したビット領域の先頭ビット＋4 ビットへ書込む値
bit[5]	：指定したビット領域の先頭ビット＋5 ビットへ書込む値
bit[6]	：指定したビット領域の先頭ビット＋6 ビットへ書込む値
bit[7]	：指定したビット領域の先頭ビット＋7 ビットへ書込む値
bit[8]	：指定したビット領域の先頭ビット＋8 ビットへ書込む値
bit[9]	：指定したビット領域の先頭ビット＋9 ビットへ書込む値
bit[10]	：指定したビット領域の先頭ビット＋10 ビットへ書込む値
bit[11]	：指定したビット領域の先頭ビット＋11 ビットへ書込む値
bit[12]	：指定したビット領域の先頭ビット＋12 ビットへ書込む値
bit[13]	：指定したビット領域の先頭ビット＋13 ビットへ書込む値
bit[14]	：指定したビット領域の先頭ビット＋14 ビットへ書込む値
bit[15]	：指定したビット領域の先頭ビット＋15 ビットへ書込む値

戻り値

正常終了　　0

異常終了　－1

解説

指定した区画のビットへデータを書込みます。

3.19 DＩ８ビット整数型データ読込みコマンド	D8in
コマンド分類	制御コマンド

ＶＣ

```
int D8in (
        int             iounit_no,
        int             bitblock_no,
        int             inverse,
        int             bitnot ,
        unsigned char   ＊bit8
       );
```

ＶＢ

Private Declare Function D8in Lib "mechad00.dll" (ByVal iounit_no As Long, ByVal bitblock_no As Long, ByVal inverse As Long, ByVal bitnot As Long, bit8 As Long) As Long

引数

iounit_no	：読込む局の局番号
bitblock_no	：読込むビットの区画番号（１～１６） １区画につき８ビット幅の値を読込みます。 区画番号のビット配置詳細については、付録 A.1 Ｉ／Ｏコマンドの区画配置図を参照ください。
inverse	：ビット並びの変更, ０：並び換え無し １：ＭＳＢとＬＳＢが逆になるようビットを並び換える
bitnot	：ビット反転（０：反転しない，１：反転する）
bit	：指定した領域のビット情報をビット幅８の整数値として読込んだ値

戻り値

正常終了　０

異常終了　－１

解説

指定した領域のビット情報を、ビット幅８の整数値として読込みます。

使用例

例１（ビット番号１から 8 ビット分のデータが２進数 00010011 の場合）

unsigned char bit8; ①

D8in(1，1，0，0,&bit8);②

①変数 bit8 を定義

②ビット番号１のデータを並び換えなし、ビット反転なしで読込みます。

変数 bit8 にビット番号１から 8 ビット分の値(00010011 $)_2$ **が格納されます。**

例２（ビット番号１から 8 ビット分のデータが２進数 00010011 の場合）

unsigned char bit8; ①

D8in(1，1，1，0,&bit8);②

①変数 bit8 を定義

②ビット番号１のデータを並び換え有り、ビット反転なしで読込みます。

変数 bit8 にビット番号１から 8 ビット分の値(11001000 $)_2$ **が格納されます。**

例３（ビット番号１から 8 ビット分のデータが２進数 00010011 の場合）

unsigned char bit8; ①

D8in(1，1，0，1);②

①変数 bit8 を定義

②ビット番号１のデータをビット反転ありで読込み

変数 bit8 にビット番号１から 8 ビット分の値(11101100 $)_2$ **が格納されます。**

3.20 DＩ16 ビット整数型データ読込みコマンド	D16in
コマンド分類	制御コマンド

ＶＣ

```
int D16in(
        int             iounit_no,
        int             bitblock_no,
        int             inverse,
        int             bitnot,
        unsigned short  ＊bit16
       );
```

ＶＢ

```
Private Declare Function D16in Lib "mechad00.dll" (ByVal iounit_no As Long,
ByVal bitblock_no As Long, ByVal inverse As Long, ByVal bitnot As Long,
bit16 As Long) As Long
```

引数

iounit_no	：読込む局の局番号
bitblock_no	：読込むビットの区画番号（１～８） １区画につき１６ビット幅の値を読込みます。 区画番号のビット配置詳細については、付録 A.1 Ｉ／Ｏコマンドの区画配置図を参照ください。
inverse	：ビット並びの変更, ０：並び換え無し １：ＭＳＢとＬＳＢが逆になるようビットを並び換える
bitnot	：ビット反転（０：反転しない，１：反転する）
bit16	指定した領域のビット情報をビット幅１６の整数値として読込んだ値

戻り値

正常終了　０

異常終了　－１

解説

指定した領域のビット情報を、ビット幅１６の整数値として読込みます。

使用例

例１（ビット番号１から１６ビット分のデータが２進数 0001000100010011 **の場合）**

unsigned short bit16;①

D16in(1，1，0，0 ,&bit16);②

①変数 bit16 **を定義**

②ビット番号１のデータを並び換えなし、ビット反転なしで読込みます。

変数 bit16 **にビット番号１から１６ビット分の値(** 0001000100010011 $)_2$ **が格納されます。**

例２（ビット番号１から１６ビット分のデータが２進数 0001000100010011 **の場合）**

unsigned short bit16;①

D16in(1，1，1，0 ,&bit16);②

①変数 bit16 **を定義**

②ビット番号１のデータを並び換え有り、ビット反転なしで読込みます。

変数 bit16 **にビット番号１から１６ビット分の値(** 1100100010001000 $)_2$ **が格納されます。**

例３（ビット番号１から１６ビット分のデータが２進数 0001000100010011 **の場合）**

unsigned short bit16;①

D16in(1，1，0，1 ,&bit16);②

①変数 bit16 **を定義**

②ビット番号１のデータをビット反転ありで読込み

変数 bit16 **にビット番号１から１６ビット分の値(** 1110111011101100 $)_2$ **が格納されます。**

3.21 DＩ32 ビット整数型データ読込みコマンド	D32in
コマンド分類	制御コマンド

ＶＣ

```
int D32in(
        int             iounit_no,
        int             bitblock_no,
        int             inverse,
        int             bitnot,
        unsigned long   ＊bit32
       );
```

ＶＢ

```
Private Declare Function D32in Lib "mechad00.dll" (ByVal iounit_no As Long,
ByVal bitblock_no As Long, ByVal inverse As Long, ByVal bitnot As Long,
bit32 As Long) As Long
```

引数

iounit_no	：読込む局の局番号
bitblock_no	：読込むビットの区画番号（１～４） １区画につき３２ビット幅の値を読込みます。 区画番号のビット配置詳細については、付録 A.1 Ｉ／Ｏコマンドの区画配置図を参照ください。
inverse	：ビット並びの変更, ０：並び換え無し １：ＭＳＢとＬＳＢが逆になるようビットを並び換える
bitnot	：ビット反転（０：反転しない，１：反転する）
bit32	指定した領域のビット情報をビット幅３２の整数値として読込んだ値

戻り値

正常終了　０

異常終了　－１

解説

指定した領域のビット情報を、ビット幅３２の整数値として読込みます。

使用例

例１（ビット番号１から３２ビット分のデータが
２進数 00010001000100010001000100010011 の場合）

unsigned long bit32①

D32in(1，1，0，0 ,&bit32);②

①変数 bit32 を定義します。

②ビット番号１のデータを並び換えなし、ビット反転なしで読込みます。
変数 bit32 にビット番号１から３２ビット分の値
(00010001000100010001000100010011)$_2$ が格納されます。

例２（ビット番号１から３２ビット分のデータが
２進数 00010001000100010001000100010011 の場合）

unsigned long bit32①

D32in(1，1，1，0 ,&bit32);②

①変数 bit32 を定義します。

②ビット番号１のデータを並び換え有り、ビット反転なしで読込みます。
変数 bit32 にビット番号１から３２ビット分の値
(11001000100010001000100010001000)$_2$ が格納されます。

例３（ビット番号１から３２ビット分のデータが
２進数 00010001000100010001000100010011 の場合）

unsigned long bit32①

D32in(1，1，0，1);②

①変数 bit32 を定義します。

②ビット番号１のデータをビット反転ありで読込み
変数 bit32 にビット番号１から３２ビット分の値
(11101110111011101110111011101100)$_2$ が格納されます。

3.22 DO8 ビット整数型データ書込みコマンド	D8out
コマンド分類	制御コマンド

VC

```
int D8out(
        int             iounit_no,
        int             bitblock_no,
        unsigned char   bit8,
        int             inverse,
        int             bitnot
       );
```

VB

```
Private Declare Function D8out Lib "mechad00.dll" (ByVal iounit_no As Long,
ByVal bitblock_no As Long, ByVal bit8 As Long, ByVal inverse As Long,
ByVal bitnot As Long) As Long
```

引数

iounit_no	：出力する局の局番号
bitblock_no	：書込むビットの区画番号（１～１６） １区画につき８ビット幅の値を書込みます。 区画番号のビット配置詳細については、付録 A.1 Ｉ／Ｏコマンドの区画配置図を参照ください。
bit8	：出力するビット幅 8 の整数型データ
Inverse	：ビット並びの変更, ０：並び換え無し １：ＭＳＢとＬＳＢが逆になるようビットを並び換える
bitnot	：ビット反転（０：反転しない，１：反転する）

戻り値

正常終了　０

異常終了　－１

解説

指定した８ビット領域へデータを書込みます。

使用例

例 1

D8out(1，1，0x0013，0，0);

指定したビットデータ(000100011)$_2$ を、並び換え無し、ビット反転なしで

ビット１から 8 へ書込みます。書込まれる値：(000100011)$_2$

例２

D8out(1，1，0x0013，1，0);

指定したビットデータ(000100011)$_2$**を、並び換え有り、ビット反転なしで**
ビット１から 8 **へ書込みます。書込まれる値：**(110001000)$_2$

例３

D8out(1，1，0x0013，0，1);

指定したビットデータ(000100011)$_2$**を、並び換え無し、ビット反転ありで**
ビット１から 8 **へ書込みます。書込まれる値：**(111011100)$_2$

3.23 DO１６ビット整数型データ書込みコマンド	D16out
コマンド分類	制御コマンド

ＶＣ

```
int D16out (
        int             iounit_no,
        int             bitblock_no,
        unsigned short  bit16,
        int             inverse,
        int             bitnot
       );
```

ＶＢ

```
Private Declare Function D16out Lib "mechad00.dll" (ByVal iounit_no As Long,
ByVal bitblock_no As Long, ByVal bit16 As Long, ByVal inverse As Long,
ByVal bitnot As Long) As Long
```

引数

iounit_no	：出力する局の局番号
bitblock_no	：書込むビットの区画番号（１～８） １区画につき１６ビット幅の値を書込みます。 区画番号のビット配置詳細については、付録 A.1 Ｉ／Ｏコマンドの区画配置図を参照ください。
bit16	：出力するビット幅１６の整数型データ
inverse	：ビット並びの変更, ０：並び換え無し １：ＭＳＢとＬＳＢが逆になるようビットを並び換える
bitnot	：ビット反転（０：反転しない，１：反転する）

戻り値

正常終了　０

異常終了　－１

解説

指定した１６ビット領域へデータを書込みます。

使用例

例 1

D16out(1，1，0x1113，0，0);

指定したビットデータ(0001000100010011)$_2$ を、並び換え無し、ビット反転なしで

ビット１から１６へ書込みます。書込まれる値：(0001000100010011)$_2$

例２

D16out(1，1，0x1113，1，0);

指定したビットデータ(0001000100010011)$_2$**を、並び換え有り、ビット反転なしで**

ビット１から１６へ書込みます。書込まれる値：(1100100010001000)$_2$

例３

D16out(1，1，0x1113，0，1);

指定したビットデータ(0001000100010011)$_2$**を、並び換え無し、ビット反転ありで**

ビット１から１６へ書込みます。書込まれる値：(1110111011101100)$_2$

3.24 ＤＯ３２ビット整数型データ書込みコマンド	D32out
コマンド分類	制御コマンド

ＶＣ

```
int D32out(
        int             iounit_no,
        int             bitblock_no,
        unsigned long   bit32,
        int             inverse,
        int             bitnot
       );
```

ＶＢ

```
Private Declare Function D32out Lib "mechad00.dll" (ByVal iounit_no As Long,
ByVal bitblock_no As Long, ByVal bit32 As Long, ByVal inverse As Long,
ByVal bitnot As Long) As Long
```

引数

iounit_no	：出力する局の局番号
bitblock_no	：書込むビットの区画番号（１～４） １区画につき３２ビット幅の値を書込みます。 区画番号のビット配置詳細については、付録 A.1 Ｉ／Ｏコマンドの区画配置図を参照ください。
bit32	：出力するビット幅１６の整数型データ
inverse	：ビット並びの変更, ０：並び換え無し １：ＭＳＢとＬＳＢが逆になるようビットを並び換える
bitnot	：ビット反転（０：反転しない，１：反転する）

戻り値

正常終了　０

異常終了　－１

解説

指定した３２ビット領域へデータを書込みます。

使用例

例 1

```
D32out( 1，1，0x11111113，0，0 );
```

指定したビットデータ(00010001000100010001000100010011)$_2$ を、並び換え無し、ビット反転無しでビット１から３２へ書込みます。

書込まれる値：(00010001000100010001000100010011)$_2$

例２

D32out(1，1，0x11111113，1，0);

指定したビットデータ(00010001000100010001000100010011)$_2$**を、並び換え有り、ビット反転なしでビット１から３２へ書込みます。**

書込まれる値：(11001000100010001000100010001000)$_2$

例３

D32out(1，1，0x11111113，0，1);

指定したビットデータ(00010001000100010001000100010011)$_2$**を、並び換え無し、ビット反転ありでビット１から３２へ書込みます。**

書込まれる値：(11101110111011101110111011101100)$_2$

A. 付録

A.1. Ｉ／Ｏコマンドの区画配置図

下図は MECHATROLINK-Ⅰ通信領域のコマンド領域、またはレスポンス領域の１バイト目から１６バイト目とします。

データ幅が８ビットの場合

８ビット

バイト	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
区画番号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ビット	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0	7-0

データ幅が１６ビットの場合

１６ビット

バイト	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
区画番号	1		2		3		4		5		6		7		8	
ビット	7-0	15-8	7-0	15-8	7-0	15-8	7-0	15-8	7-0	15-8	7-0	15-8	7-0	15-8	7-0	15-8

データ幅が３２ビットの場合

３２ビット

バイト	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
区画番号	1				2				3				4			
ビット	7-0	15-8	23-16	31-24	7-0	15-8	23-16	31-24	7-0	15-8	23-16	31-24	7-0	15-8	23-16	31-24

A.2. サポート

製品のサポートは下記で行っています。

・電子メール

cnc@itt.co.jp

・ftp サーバ

ftp.itt.co.jp

最新のソフトウェアモジュールや、評価用ソフトウェアを提供します。

A.3. 製品案内

・ITT ホームページ

http://www.itt.co.jp/

当社のホームページです。

製品とサービスに関する情報をご覧いただけます。

さらに、下記宛てのメールにてご質問をお受けします。

query@itt.co.jp

・ITT FAX インフォメーションサービス（053-466-5555）

当社の製品とサービスに関する最新情報が 24 時間いつでも取り出すことができます。

0010001#で総合目次を取り出すことができます。

株式会社　アイ・ティー・ティー
静岡県浜松市向宿 1-14-7
TEL　（053）462-6111　　FAX　（053）462-2557
FAX Information　　（053）466-5555

E-Mail　query@itt.co.jp　　WWW　http://www.itt.co.jp/