管理番号	C06621A-Z082H

技術資料
（マクロプログラミングマニュアル）

品　名　　GraphicsOperationPanel

シリーズ名 GOP-4000/5000Series

本資料は GOP-4000/5000 シリーズのマクロプログラミングに関する技術資料です。記載内容は予告無く変更する場合がありますことをご了承下さい。

初版作成日	本書作成日	デバイス製造１課			品質保証	
		承認	確認	担当	承認	確認
2009/8/17	2014/10/21	藤岡	藤本	島田		

備考
本書は TP-DesignerV4 Ver4.3.0.10 以降の環境を対象にしています。
対応機種が限定されるコマンドを対象機種以外で実行しないで下さい。

管理番号	C06621A-Z082H

改定履歴表

改定番号	改定年月日	改定内容	担当	承認
―	2009/8/17	・初版。	島田	藤岡
A	2009/9/24	・GOP-4043WQTA 対応。（下位互換のため使用できない機能コメント追加。） ・GOP-4043WQTA および GOP-4***VT*ver1.0.9 より 12 ドットフォント対応したため、文字描画コマンドで 12 ドットフォント指定用パラメータ追加。	島田	藤岡
B	2010/2/4	・新ｺﾏﾝﾄﾞに関する記述を追加 LOAD_WAV LOAD_JPEG LOAD_BMP HTTP_GET CONVARET_BMP ALIAS_NAME SCREEN_CAPT ・誤記修正 RND	島田	藤岡
C	2010/6/10	・マクロ描画に関する記述を、キャンバスオブジェ　クト対応に修正	島田	藤岡
D	2012/8/3	・GOP-5000 シリーズ対応に伴う新コマンド追加。 ・注記追加	島田	藤岡
E	2013/6/18	・CP、APD コマンドの記述内容修正。	島田	藤岡
F	2013/8/21	・DOEVT 注意事項追加。	島田	藤岡
G	2014/8/12	・TREBDBUF_FILL 誤記修正。 ・FILEOPERATION 誤記修正。 ・表記揺らぎなど全面見直し。 ・GOP 動作の流れの記述の見直し。 ・プログラム制御コマンド関係の非推奨扱いを見直し。 ・イメージキャッシュ関連コマンドの非推奨扱いを見直し。	島田	藤岡
H	2014/10/21	・ファイルアクセスコマンド使用時の注意事項追記。	島田	藤岡

備考

目次

1. 概要

GOP のマクロは、GOP の持つ内部コマンドを直接記述することで、標準オブジェクトにない動作を実現することが出来ます。(TPD V4 で作画した標準のオブジェクトも画面転送時にあらかじめ指定された形式でマクロと同様のコマンドに変換されます。)

GOP のマクロは独自の仕様に基づいており、本書にて文法・コマンドおよびサンプルの説明を行います。

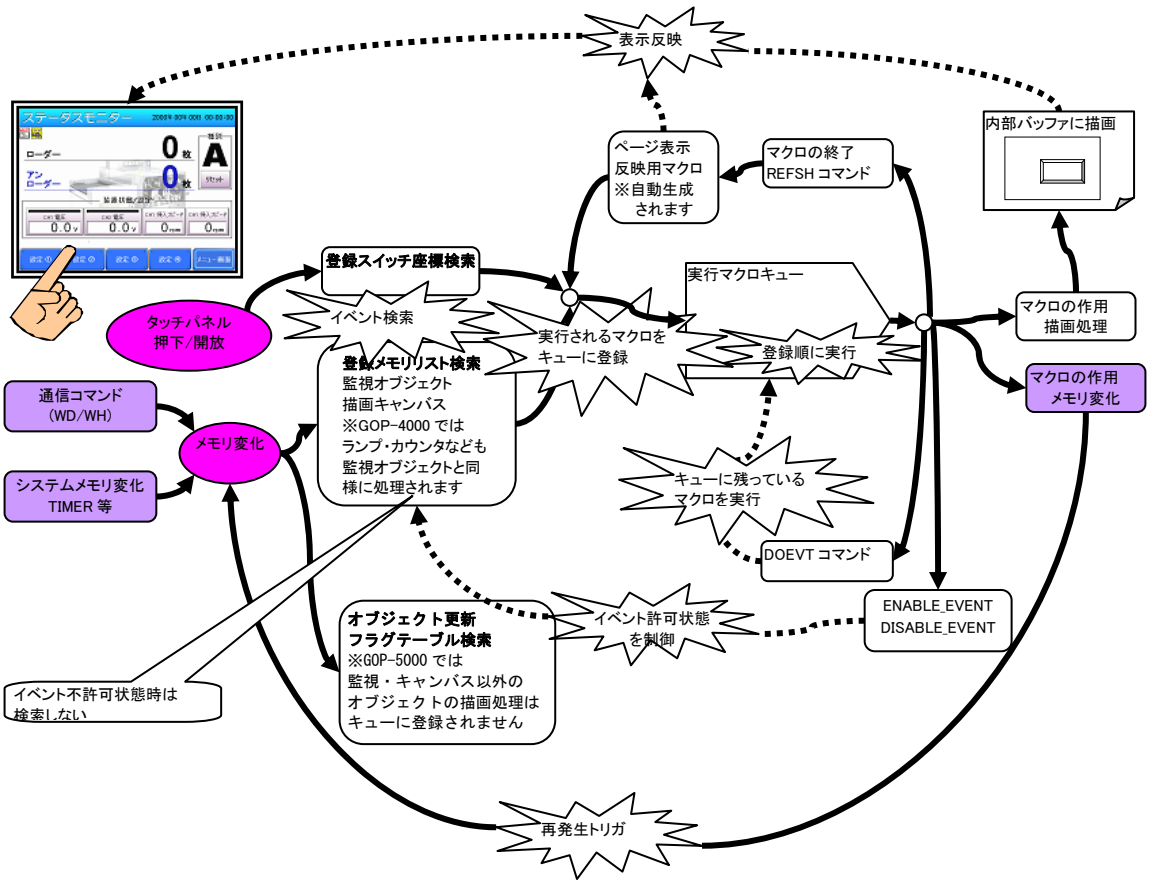
本書を読むにあたって、TPD V4 でマクロを使用しない画面データの構築は出来る程度の経験を持っていることを前提とします。

2. マクロの実行タイミングとGOP動作の流れ

2. 1 GOP動作の流れ

マクロの実行は以下の図のような流れとなり、実際の開始のトリガとしては以下がきっかけでマクロが起動します。

・タッチパネルの押下/開放
・メモリの変化
　通信によるメモリの変化
　システム動作によるメモリの変化(タイマやシリアル受信フラグなど)
　マクロ自身によるメモリの変化



マクロの登録は 2. 2 項にて説明します。
マクロ以外のランプやカウンターなどのオブジェクト類も内部動作としてはマクロと同様用の仕組みとして実現されておりトリガは共通です。

マクロのトリガとなる事象が発生すると登録されているマクロのリストを検索します。(タッチパネル押下/開放の場合は登録スイッチリスト、メモリ変化の場合は登録メモリリスト。)
この検索で発生した事象に一致するアイテムがあった場合、それに関連づいたマクロが実行マクロキューに実行予約されます。この登録マクロのリスト検索作業をイベント検索といいます。
(GOP-4000 ではランプなどの更新描画処理もマクロと同様に実行マクロキューに予約されますが、GOP-5000 においてはランプなどのオブジェクト類の更新動作は実行マクロキューではなく別管理されます。)

実行マクロキューに格納されたマクロは、GOP の動作がアイドル(マクロや描画が行われていないタイミング)になった時点で取り出され実行されます。また DOEVT コマンド(詳細は後述)を実行す

るとアイドルにならなくても実行マクロキューに残ったマクロを取り出し実行します。
マクロで描画を行う場合、マクロ実行時点では内部バッファに対してのみ描画され、表示には反映されません。マクロが終了したときおよびREFSHコマンド(詳細は後述)実行時に内部バッファから表示プレーンに転送するマクロ(このマクロはTPD V4によりページごとに自動生成されます)が実行マクロキューに登録されます。

マクロの実行結果によりメモリが変化する場合、その変化をトリガとしてイベント検索が再発生します。それにより他のマクロやオブジェクトの表示変化を発生させることができますが、逆に単にメモリ変化のみを行いたい場合は、DISABLE_EVENTコマンド(詳細は後述)でイベント検索を抑止することができます。イベント検索を再開する場合はENABLE_EVENTコマンド(詳細は後述)を使用します。イベント検索が抑止されるのはメモリ変化に伴うもののみデータッチパネルの押下/開放によるイベント検索は抑止されません。

実行マクロキューに予約されるものはどのマクロを実行するか？という情報のみでそのときのメモリの値などは格納されません。また実行マクロキューの輻輳防止のため、すでに実行マクロキューに登録されているマクロと同じマクロを格納使用とした場合、すでに登録済みのマクロは処理されず後から登録したもののみが実行されます。例えば高速にメモリ変化するようなマクロを作成し、そのメモリにリンクした監視オブジェクトなどを作成した場合、メモリ変化回数とマクロ実行回数が一致しないという結果になります。また実行マクロキューはメモリの値を保持しないため最終のメモリの値でマクロは必ず一回は実行されることになります。

マクロの実行は上記モデルのようになっているため以下の点に注意して作成して下さい。
- ・マクロの中でメモリ変化した場合、それに伴いマクロが再発生します。例えば監視オブジェクトのリンクメモリをマクロの中で更新した場合、それにより連鎖的にマクロが発生し無限ループと同じような状況になります。
- ・マクロはイベント発生した時点では実行を予約されるのみで、実際の実行のタイミングはその時点の実行マクロキューの状態によります。そのため最悪応答時間については一切の保証はありません。リアルタイム性が要求される動作につきましてはホスト側にて実装下さい。

2. 2 マクロの登録方法

マクロは次のイベントをトリガとして起動させることが出来ます。

(1) カスタムボタン押下

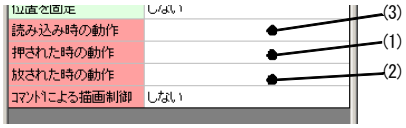
カスタム動作定義ボタンが押された時に動作します。カスタム動作定義ボタンのプロパティシートの「押された時の動作」で指定します。

(2) カスタムボタン開放

カスタム動作定義ボタンが放された時に動作します。カスタム動作定義ボタンのプロパティシートの「離された時の動作」で指定します。

(3) カスタムボタン読み込み

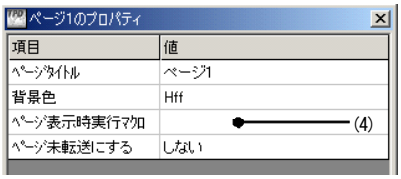
カスタム動作定義ボタンを GOP が表示する際に読み込んだ時に動作します。カスタム動作定義ボタンのプロパティシートの「読み込み時の動作」で指定します。
ボタンの初期化などに使用します。
この動作は、後述の(5)のマクロ挿入オブジェクトの動作と同じものです。
※ボタン＋マクロ挿入オブジェクトをボタンだけで記述できるようにしています。



【カスタムボタンのプロパティシート】

(4) ページ表示時：その１

ページ表示時実行マクロはページが読み込まれたときに動作します。ページのプロパティシートの「ページ表示実行マクロ」で指定します。
ページ表示時の初期化動作などを記述します。
ページ表示時実行マクロに記述の動作は、ページに登録されているオブジェクトの描画前に実行されます。



【ページのプロパティシート】

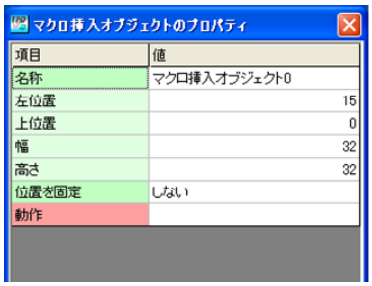
(5) ページ表示時：その２(マクロ挿入オブジェクト)

マクロ挿入オブジェクトはページが読み込まれたときに動作します。マクロ挿入オブジェクトのプロパティシートの「動作」で指定します。本オブジェクトは作図エリアには表示されますが、GOP へは表示されません。

本オブジェクトに記述されている動作は、本オブジェクトが配置されている位置にて実行されます。例えば監視オブジェクトのリンクメモリに初期値を設定する場合、監視オブジェクトの背面にマクロ挿入オブジェクトを配置している場合、マクロ挿入オブジェクト実行時点では監視オブジェクトは登録されていないため、監視オブジェクトのイベントは発生しません。

監視オブジェクトの前面にマクロ挿入オブジェクトを配置した場合は、マクロ挿入オブジェクト実行時点で監視オブジェクトは登録されているため、監視オブジェクトのイベントを起動することが出来ます。

※オブジェクトは配置の最背面のものから描画(TPD V4 が作成した描画マクロの実行)されていきます。



【アイコン】　　【プロパティシート】

(6) メモリ更新(監視オブジェクト)

監視オブジェクトで指定したリンクメモリの値が条件に合致した時に動作します。監視オブジェクトのプロパティシートの「動作」で指定します。

また、監視オブジェクトの監視動作範囲は適用範囲で指定し、オブジェクトの配置ページを表示中のみ有効(表示中)または、ページを変更しても監視を継続(継続)の 2 パターンを設定できます。継続を指定時は、一度その監視オブジェクトが配置してあるページを表示すると、GOP の電源を切るまでその監視は有効になります。

監視オブジェクトは、その配置ページを表示しないと有効にならないため、画面の動作の全体にわたり監視が必要な場合は、継続オブジェクトをページ１に配置します。

※ページ１は必ず表示されるため。

本オブジェクトは TPD V4 の作図エリアには表示されますが、GOP へは表示されません。

インターバル動作、遅延動作などは本オブジェクトのリンクメモリにシステムメモリのタイマーメモリ(TIMER100ms_1 など)をリンクメモリとして設定することで実現できます。

アイコン】　　【プロパティシート】

(7) サブルーチン

(1)～(6)のイベントにより起動したマクロからサブルーチンとして呼び出すことが出来ます。
メニューの[設定]-[マクロ]-[共通サブルーチンの設定]で指定します。
ここで定義したマクロは、特定のトリガと結びつかないため、サブルーチンとして呼ばれない限り実行されることはありません。

(8) キャンバスオブジェクト

マクロによる任意描画を行うためのオブジェクトとしてキャンバスオブジェクトがあります。
このオブジェクトの描画内容(マクロ)をクリックするとマクロを記述できます。
ここで記述したマクロはリンクメモリが指定の値に変化したときに実行されます。
また他のオブジェクトからもキャンバスに対し描画することも可能です。
詳しくは後述の 4.2.16 描画コマンドを参照下さい。

【アイコン】

【プロパティシート】

3. 基本的な文法

※使用例等において未説明のコマンドが出現する場合がありますがご了承下さい。それらのコマンドの内容については本章および 4 章を参照下さい。

3. 1 表記

(1)メモリ指定

マクロ中でメモリを指定する場合、通常の直接指定[w0000etc]・間接指定形式[b(w0010)etc]の他に、メモリリスト、名前指定メモリリストおよびマクロ内での ADIM,DIM,EDIM コマンドでの定義がある場合メモリの指定を型+アドレス記述でなく、定義名で指定できます。

・メモリ名称での指定。

```
MOV 照明表示 1 #1
```

・間接参照のインデックスメモリのメモリ名称での指定。

```
MOV B(メモリ番号+10) #1
```

(2)値の表記

マクロ中でメモリではなく、具体的な値(数値、文字列)を表記する場合、以下のように記述します。

・整数 先頭に#をつけ、数値を記述する。

```
#123 #555
MOV w0000 #1234
EXPR w0000=#123*w0002
```

・浮動小数点値

先頭に@をつけ、数値を記述する。

```
@123.45  @100
MOV F0000 @-0.1
EXPR F0000=@123.45*w0002
```

・文字列

先頭に$をつけ、文字列を記述する。

```
$ABCDEFG
CP 10 T0000 $12345
```

※文字列では行の末尾までが文字列として認識されます。

例えば以下の場合 T000 には ABC の後ろにスペースが 3 つ入ります。

```
CP 10 T0000 $ABC    CR LF
```

・整数の値について、先頭に&H とつけることで 16 進数表記が出来ます。

```
#&HFF00
MOV w0000 #&HFF00
```

(3)コメント

行頭に;をつけるとその行はコメントとなります。

※行末に;をつけてもコメントとはなりません。コマンドにより末尾までの文法解釈が行われない場合、末尾に;をつけ以降にコメントを記述してもエラーとならない場合がありますが、これは正しい動作ではありませんのでコメントは行頭から記述して下さい。

(4)字下げ

ループや IF の適用範囲が見やすいよう字下げを行うことが出来ます。字下げは半角スペース及び TAB を使用できます。全角スペースはエラーになります。

例

```
IF w0000=#0 THEN
  MOV w0002 #0
ELSE
  FOR w0004=#0 TO #10
      EXPR w(0004)=w0004*#10
  NEXT
ENDIF
```

(5)メモリアドレス取得

メモリ名の先頭に&を付けることでそのメモリのアドレスを取得することが可能です。

例

```
MOV w0000 &PAGE
```

とすると

```
MOV w0000 #&Hf000
```

と置き換えられます。

間接参照時は

```
MOV w0000 &B(w2000+15)
```

は

```
MOV w0000 w2000
ADD w0000 #15
```

と置き換わります。

(6)メモリタイプ取得

メモリ名の先頭に:(コロン)を付けることでメモリの型を数値で取得できます。

型毎の番号は以下のとおりです。

'b'=1
'w'=2
'l'=3
'B'=4
'W'=5
'L'=6
'F'=7
'T'=8

例

```
ADIM XVAL F
MOV w0000 :XVAL
```

とすると w0000 に 7 がセットされます。

(7)文字列のコード指定

文字列の中で以下の表記をすることで、キーボードから入力できないコードを入力できます。

¥&h[文字コード(16 進数表記)]

例

```
CP 10 T0000 $¥&h02UV¥&h03
```

上記で T0000 に<STX>UV<ETX>がセットされます。

3. 2 構文マクロ

マクロの制御構造などを記述するためのコマンドです。構文コマンドは GOP 自身の内部コマンドではなく、いくつかの内部コマンドを組み合わせたもので TPD V4 にてマクロで記述したものを変換して出力します。(どのように変換されるかは個々のコマンドを参照下さい。)

(1)ルーチン定義　　:FUNC～END_FUNCコマンド

マクロの一連の処理を記述する範囲を定義します。

FUNC [ルーチン名](処理の内容に応じた名前。重複した名前を定義することは出来ません)

:

(処理を記述)

EXIT_FUNC(処理の途中終了)<無くても可>

:

END_FUNC

全てのマクロは FUNC～END_FUNC コマンドの中に記述します。

ただしメモリの定義については外に出してもかまいません。

FUNC コマンドで定義したルーチンは SUB コマンド、BSUB コマンドで呼び出します。

なおオブジェクトでマクロを設定時自動で_(オブジェクト名)_(動作名)をルーチン名とした空のルーチンが作成されます。この中に全て記述することも出来ますが、任意にサブルーチンを作成し、自動作成されたルーチンからサブルーチンを呼び出すように記述することも出来ます。

FUNC コマンドで定義したルーチン名は、マクロ上では LBL コマンドで指定するラベル名として使用できます。

但し、FUNC で定義したルーチン名を JP コマンドで使用することはしないで下さい。

例

ルーチン定義

```
FUNC _ボタン1_押したときの動作(自動作成されたルーチン)
     SUB ユーザールーチン
END_FUNC
FUNC ユーザールーチン              (任意で作成したルーチン)
     (何か処理を記述)
END_FUNC
```

FUNC,END_FUNC コマンドは必ず対にして下さい。

また途中でルーチンを終了する場合は EXIT_FUNC コマンドをご使用下さい。

例

ルーチン定義

```
FUNC _ボタン1_押したときの動作(自動作成されたルーチン)
   IF w0000=#1 THEN
        EXIT_FUNC (w0000 が 1 のときは何もせずマクロ終了)
   ENDIF
   SUB ユーザールーチン
END_FUNC
```

※FUNC～END_FUNC コマンドは、実際には

・FUNC コマンドは LBL コマンド

・END_FUNC,EXIT_FUNC コマンドは END コマンド

に置き換えられます。(このため上の説明のラベル名として使用可能になります。)

FUNC～END_FUNC コマンドを使用して記述した場合、ルーチンの始まりと終わりが明確となり構造がはっきりする、閉じられていない場合などのエラーチェックが行われるなどのメリットがあります。

(2)メモリ別名定義　　:DIMコマンド

DIM［別名］［本体］と記述することにより別名が定義できます。

例

```
DIM 温度情報 W0010
```

DIMコマンドで定義した名称は(1)項のメモリ指定で使用することが出来ます。

DIMコマンドは単純な置き変え処理ですので、定数定義や式置換等にも利用できます。

DIMコマンドで定義した名称はマクロ中どこからでも使用できます。

・定数定義

円周率定義の例

```
DIM PI @3.141592
DIM 円面積 F0000
DIM 半径 F0004
EXPR 円面積=半径*(PI*PI)
```

・式置換

レコードデータのアクセス

下のような構造のデータクセスの表記簡素化の例を以下に記述します。

NO	1バイト
月	1バイト
日	1バイト
値	2バイト

上記のような構造のデータが0100番地から並んで配置されている場合、以下のようにメモリ上に配置されます。

b0100	データ0	NO
b0101		月
b0102		日
w0103		値
b0105	データ1	NO
b0106		月
b0107		日
w0108		値
:		
b(<データ先頭>+0)	データn データ先頭は以下式 0100h+n*5	NO
b(<データ先頭>+1)		月
b(<データ先頭>+2)		日
w(<データ先頭>+3)		値

このようなデータをアクセスする場合、間接指定表記で使用しますが、間接指定表記をDIMコマンドで置き換えることで、わかりやすい記述を行うことが出来ます。

```
DIM REC_TOP w1000
DIM REC_NUM b1002
DIM RECSIZE #5
DIM DATA.NO b(REC_TOP+0)
DIM DATA.MONTH b(REC_TOP+1)
DIM DATA.DAY b(REC_TOP+2)
DIM DATA.VALUE w(REC_TOP+3)
```

上記のように定義すると、読みたいデータ番号をREC_NUMにセットし以下の記述でそれぞれのデータにアクセスできます。

```
EXPR REC_TOP=#&H0100+( REC_NUM * RECSIZE)
MOV b0000 DATA.NO
MOV b0001 DATA.MONTH
MOV b0002 DATA.DAY
MOV w0003 DATA.VALUE
```

上記の置き換えを行わない場合以下のように記述しますが、それぞれのデータが何を意味して

いるか見た目がわかりにくいです。

```
MOV b0000 b(REC_TOP+0)
MOV b0001 b(REC_TOP+1)
MOV b0002 b(REC_TOP+2)
MOV w0003 w(REC_TOP+3)
```

※GOP-5000 シリーズのみ

ADIM コマンドで取得したメモリからオフセットを指定した別名定義が可能です。

```
DIM 別名 型'&ADIM メモリ+オフセット

ADIM str T
DIM chr1 b'¥&str+0
DIM chr2 b'¥&str+1
chr1・・・str の一文字目
chr2・・・str の二文字目
```

(3) 自動領域取得　　：ADIMコマンド

ADIM［名称］［型］と指定することで TPD V4 使用メモリ空間内に指定した型のメモリを取得することが出来ます。ADIM コマンドで取得したメモリはメモリ名称でのアクセスしか出来ません。（アドレスはページ転送時に設定されるため、画面作成時には取得することは出来ません。）

ADIM コマンドで定義した名称はマクロ中どこからでも使用できます。

例

```
ADIM 作業領域 L
```

T 型メモリを ADIM コマンドで取得する場合以下の記述で取得文字長を指定できます。

ADIM［名称］［型］［取得文字長］

例

```
ADIM 作業用文字 T 30
```

取得文字長を省略時は 41 文字分確保されます。

取得文字長は終端文字も含めた値を指定して下さい。（最大 20 文字であれば終端文字用含めて 21 を指定します。）

※DIM,ADIM コマンド及びメモリリストで設定されるメモリ名称は重複定義できません。

※メモリ名称定義に使用できない名称

?から始まる名称	:?付きメモリ指定と解釈されます。
_から始まる名称	:_は TPD V4 が生成する名称として予約されています。
#、@、$から始まる名称	:数値、文字として解釈されます

(4) 重複メモリ名領域取得　　：EDIMコマンド

EDIM［名称］［型］と指定することで、ADIM コマンドと同様にメモリを確保できます。

EDIM コマンドは同名のメモリを複数の箇所で宣言できますが、メモリは 1 箇所分しか確保されません。

EDIM コマンドで定義した名称はマクロ中どこからでも使用できます。

EDIM コマンドでも ADIM コマンド同様、文字型メモリの確保が可能です。EDIM コマンドごとに取得文字長が異なる場合（このような使用方法は推奨しません）、最大のものではなく、一番最初に宣言されたものになりますのでご注意下さい。

(5)計算式　　：EXPR,EXPDコマンド

以下の形式で四則演算および論理演算を数式の形で記述できます。

EXPR［計算式］

例

```
EXPR 温度情報=温度入力生値*#100+#273
EXPR IO=(IO&#&HFE)|#&H02
```

動作

計算式の内容を実行します。

上記の例の場合､温度情報のメモリに温度入力生値×100+273 を計算した結果が入力されます。

備考

計算式で使用可能な演算子は以下の通りです。

種別	演算順位	動作
(...)	1	(～)でくくられた範囲を先に計算します。
~	2	~の次の項の論理反転を行います
*	3	*の左右の項の乗算を行います
/	3	/の左項を右項で除算します。
%	3	%の左項を右項で除算した余りを求めます
&	3	&の左右の項の論理積を求めます
^	3	^の左右の項の排他的論理和を求めます
+	4	+の左右の項の加算を行います
-	4	-の左項かた右項を減算します。
|	4	|の左右の項の論理和を求めます
=	5	左辺に右辺の計算結果を代入します。 =の左辺には、代入先のメモリのみ記述します。 =の右辺には、計算式を記述します。

演算は演算順位に従い、順位の高いものから演算されます。

同順位の場合、左側から演算されます。

また、式の項はメモリ、数値の指定規則に従い記述して下さい。

※EXPR コマンドは計算式をそのまま実行するのではなく、与えられた計算式を GOP の内部コマンドに展開し実行します。また展開されたマクロは DISABLE_EVENT～ENABLE_EVENT コマンドでくくられ途中演算でのメモリ変化イベント検索は行われません。最終結果を左項に代入するときのみイベント検索が行われます。

展開例として以下のように展開されます。

```
EXPR A=B*C/(D-E)
      ↓
SCOPE _EXPR97
  DISABLE_EVENT  中間計算を行うためイベント発生を禁止
  LDIM t116 D E  (D-E)計算のワークメモリ確保
  MOV t116 D          D をワークメモリにセット
  DEL t116 E          D-E を実行
  LDIM t117 B C  B*C のワークメモリを確保
  MOV t117 B          B をセット
  MUL t117 C          B*C を実行
  DEV t117 t116  <B*C のワークメモリ>/<B*C のワークメモリ>を実行
  ENABLE_EVENT        最終値を左項にセットするためイベント発生を許可
  MOV A t117          計算結果を A にセット
END_SCOPE
```

展開例中ので SCOPE､LDIM､END_SCOPE は式展開用の内部コマンドです。(通常使用は出来ないため本書で説明はありません。)

上記のとおり必ず展開後のコードに ENABLE_EVENT が入るため、EXPR の副作用として、式実行後は常に ENABLE_EVENT 状態になります。

この副作用を避ける場合

EXPD［計算式］

を使用します。
EXPD を使用した場合、以下のように展開されます。

```
EXPD A=B*C/(D-E)
     ↓
SCOPE _EXPR97
  LDIM t116 D E  (D-E)計算のワークメモリ確保
  MOV t116 D          D をワークメモリにセット
  DEL t116 E          D-E を実行
  LDIM t117 B C  B*C のワークメモリを確保
  MOV t117 B          B をセット
  MUL t117 C          B*C を実行
  DEV t117 t116  <B*C のワークメモリ>/<B*C のワークメモリ>を実行
  MOV A t117          計算結果を A にセット
END_SCOPE
```

但しこの場合、ENEABLE_EVENT 状態で実行すると途中の展開式でもイベント検索が行われるため、動的オブジェクトの数が多い場合パフォーマンスに影響がある場合があります。

・EXPR コマンドを使用する場合

```
DISABLE_EVENT
MOV w0000 #1
EXPR w0002=w0000*#2    計算結果の w0002 へのセットはイベント検索が行われてしまう(パフォーマンス低下要因)
EXPR w0004=w0002*#2    計算結果の w0004 へのセットはイベント検索が行われてしまう(パフォーマンス低下要因)
EXPR w0006=w0004*#2    計算結果の w0006 へのセットはイベント検索が行われてしまう(パフォーマンス低下要因)
                  上の EXPR で ENABLE_EVENT 状態になっているため、ENABLE_EVENT は不要
```

・EXPD コマンドを使用する場合

```
DISABLE_EVENT
MOV w0000 #1
EXPD w0002=w0000*#2    計算結果の w0002 へのセットはイベント検索が行われない
EXPD w0004=w0002*#2    計算結果の w0004 へのセットはイベント検索が行われない
EXPD w0006=w0004*#2    計算結果の w0006 へのセットはイベント検索が行われない
ENABLE_EVENT      DISAVLE_EVENT 状態解除のため、ENABLE_EVENT が必要
```

(6)条件分岐 ：IF～ENDIFコマンド

以下の形式で条件文を作ることが出来ます。

IF［条件式］THEN
　　（条件成立時の動作）
ELSEIF［条件式 2］THEN 〈省略可〉
　　（条件 2 成立時の動作） 〈省略可〉
：
ELSEIF［条件式 n］THEN 〈省略可〉
　　（条件 n 成立時の動作）〈省略可〉
ELSE〈省略可〉
　　（条件不成立時の動作）〈省略可〉
ENDIF

例

```
IF w0000<#10 AND w0000>#5 THEN
  MOV 範囲内管理 #1
ELSE
  MOV 範囲内管理 #0
ENDIF
```

動作

条件式の内容により動作を切り替えます。上記の例の場合 w0000 が 10 より小さく 5 より大きい場合は範囲内管理メモリに 1 が入力されます。
そうでない場合は範囲内管理メモリに 0 が入力されます。

備考

IF コマンドは最大 15 階層まで入れ子構造にすることが出来ます。

```
IF w0000<#10 THEN
  IF w0000<#5 THEN
       ;5 より小さいときの処理
  ELSE
      ;5 以上 10 未満のときの処理
  ENDIF
ELSE
   ;10 以上のときの処理
ENDIF
```

条件式は比較子として以下のものが使用できます。

=	=の左辺と右辺が等しい場合、成立
!=	!=の左辺と右辺が等しくない場合、成立
>	>の左辺が右辺より大きい場合、成立
<	<の左辺が右辺より小さい場合、成立
>=	>=の左辺が右辺以上の場合、成立
<=	<=の左辺が右辺以下の場合、成立

ひとつの条件式は、"左辺 比較子 右辺"の形式となりこれらを、論理演算子でつなぐことにより条件を複合させることも可能です。
論理演算子は以下が使用できます。

記述	動作	順位
AND	左式と右式ともに成立時、成立します。	1
OR	左式と右式どちらかが成立すれば、成立します。	2

例

```
IF w0000=0 OR w0002>10 THEN
```

また括弧を使用し、より複雑な条件を記述できます。

```
IF w0000=#1 AND (w0002=#2 OR w0004=#2) THEN
```

条件式中で数値計算を行うことも出来ます。
条件式中で記述できる内容は(9)計算式の右辺と同様です。
計算式は、条件式の右辺、左辺いずれにも記述できます。

```
IF (w0000-#2)< w0002&#&H00ff THEN
IF ((w0000-#2)<w0002) AND (w0020 & #&H0100 =#0) THEN
```

※IF コマンドは式をそのまま実行するのではなく、与えられた式を GOP の内部コマンドに展開し実行します。また展開されたマクロは DISABLE_EVENT～ENABLE_EVENT コマンドでくくられ、途中演算でのメモリ変化によるイベント検索は行われません。また IF コマンドの副作用として、式実行後は常に ENABLE_EVENT 状態になります。

(7)回数指定ループ　　：FOR～NEXTコマンド

以下の書式で繰り返しループを作成します。

FOR [計数メモリ]=[初期値] TO [終了値] STEP [増加幅] <STEP 省略可>
　　(処理を記述)
NEXT

※STEP 以降は省略可。省略時は増加幅は 1 になります。

例

```
FOR w0100=#0 TO #10
  MOV b(0100) #0
NEXT
```

動作

上記の場合、w0100 が 0 から 10(16 進数で a)になるまで MOV b(0100) #0 の動作を繰り返します。
結果として b0000 から b000a のメモリに 0 が入力されます。

備考

・STEP に負の数を指定すると減算カウントします。
・EXIT コマンド(後述)でループを抜けることが出来ます。

※FOR～NEXT コマンドは、与えられた条件に基づき GOP 内部コマンドに展開し実行します。

```
FOR w0000 =#10 TO #20
      (処理)
NEXT
```

は以下のように置き換えられます。

```
DISABLE_EVENT     ･･･初期値セット
MOV w0000 #10
ENABLE_EVENT
R_LBL  _LOOPTOP_n ･･･ループ先頭ラベル
      (処理)
ADD w0000 #1      ･･･増分処理
CMP <= w0000 #20  ･･･条件判定
R_BR _LOOPTOP_n   ･･･ループ先頭ラベルにジャンプ
R_LBL  _LOOPEND_n ･･･ループ終了ラベル
```

ループの終了判定は末尾判定のため処理は必ず 1 回は実行されます。

(8)無限ループ　　：DO～LOOPコマンド

以下の書式でループを作成します。

DO

（処理を記述）

LOOP

例

```
DO
  DOEVT
  IF w0000!=#0 THEN
      EXIT
  ENDIF
LOOP
```

動作

w0000 が 0 でなくなるまでループします。

備考

DO コマンドは無限ループコマンドです。必ずループ中に終了条件を判定する処理と EXIT コマンドを記述して下さい。

※DO～LOOP コマンドは、与えられた条件に基づき GOP 内部コマンドに展開し実行します。

```
DO
  （処理）
LOOP
```

は以下のように置き換えられます。

```
R_LBL  _LOOPTOP_n ･･･ループ先頭ラベル
（処理）
R_JP _LOOPTOP_n  ･･･ループ先頭へジャンプ
R_LBL  _LOOPEN_n ･･･ループ終了ラベル
```

(9)ループから抜ける　：EXITコマンド

FOR または DO コマンドのループから抜けます。

EXIT

動作

FOR～NEXT コマンドまたは DO～LOOP のループから抜けます。

※EXIT コマンドはコマンド直後のループ終了ラベルへの R_JP コマンドに置き換えられます。

(10)引数、戻り値を持ったルーチンの作成　　：FUNC,ADIM応用

TPD V4 はサブルーチン呼び出しに引数、戻り値をとることは出来ませんが、以下の方法で擬似的に処理することが出来ます。

引数、戻り値を使用する場合、それ用のメモリをサブルーチン側で取得します。

例

例えば 2 つのメモリの平均値を求めるサブルーチンを作る場合の例。

```
ADIM Param1 w        一つ目の引数
ADIM Param2 w        二つ目の引数
ADIM RetAvarage w 戻り値。これに平均値がセットされます。
FUNC AVAREGE2
   EXPR RetAvarage=(Param1+Param2)/#2
END_FUNC
```

呼び出す場合は以下のようにします。(w0000 と w0002 の平均を w0004 にセット。)

```
VALSET Param1 w0000
VALSET Param2 w0002
SUB AVAREGE2
MOV w0004 RetAvarage イベント発生が不要時は VALSET で可
```

(11)外部テキストファイル取り込み　　：　;!Import

マクロを外部エディタで作成したファイルを共通サブルーチンに読み込むことが可能です。

書式は以下の通りです。

;!Import [外部ファイル名]

外部ファイルは画面データと同じフォルダにおいて下さい。

外部ファイル化することで、共通的な動作をライブラリ化することが可能です。

例

```
;!Import 外部マクロ.txt
LBL AAA
END
:
```

4. 内部コマンドリファレンス

4. 1 本書での表記について

本書でのコマンドの表記は以下のフォーマットになります。

コマンド名 [パラメーター1] [パラメーター2]･･･

コマンド名は英大文字です。
セパレータは半角スペースです。
パラメーターについては英大文字で記述してある箇所はメモリ指定の箇所です。この箇所を定数と指定することも可能です。(出来ない場合もあります。)定数値を設定するには先頭に#,@等識別符号を付加する必要があります。
小文字表記の箇所は定数値のみ指定可です。数値先頭に識別符号は不要です。
またパラメーターは各コマンドごとに名前を付けていますが、共通で使用している名前として DEST は主にコマンド実行結果の格納先を示すパラメーターとして使用しています。
なお、DEST と表記がある箇所についてはメモリしか指定できません。メモリの型については各々のコマンドごとに制限があります。
SRC は主にコマンドに与える引数として使用する箇所に使用しています。
例えば MOV コマンドの場合、パラメーターは以下のようになります。

```
MOV DEST SRC
```

DEST,SRC とも大文字表記なので、ここにはメモリまたは数値(識別符号付)を指定します。

```
MOV W0000 #123
MOV W0000 F0000
```

以下のような表記はエラーとなります。

```
MOV W0000 123
```

同様に CP の場合

```
CP length DEST SRC
```

ですので、

```
CP 10 T0000 $ABC
CP 5 T0000 T1000
```

といった表記が可能です。

```
CP #10 T0000 $ABC
```

は数値のみ指定の箇所に識別符号をつけたためエラーとなります。

4. 2 内部コマンド

GOP で使用できるマクロコマンドの説明です。
なお以下説明で GOP-H クラスとは GOP-4065/84/104VTA/B 及び、GOP-5065/84/104HVTA/B をいいます。

4. 2. 1 プログラム制御コマンド

プログラム制御コマンドは、マクロの実行順の制御を行います。

(1) END コマンド終了

コマンド書式

END

動作

コマンドの終了
サブルーチンからの復帰

※本コマンドについては構文マクロコマンド(END_FUNC,EXIT_FUNC コマンド)にて代替可能なため原則構文マクロの使用を推奨します。

(2) LBL ラベル

コマンド書式

LBL lbl

動作

指定した箇所に lbl で指定したラベル名でラベルを挿入します。
ラベル名に同じ名称は使用できません。
LBL コマンドを使用することにより、lbl 名称を ID とした番号が割り当てられます。
JP コマンドや BR コマンドなどの LBL コマンドで指定したラベル名を使用するコマンドでも、lbl 名称⇒番号変換は自動で行われるので、マクロ記述上は lbl は数値指定ですが、ラベル名称を指定して下さい。

使用例

```
FUNC _ボタン1_押されたときの動作
  LBL LOOPTOP
  IF w0000=#0 THEN
      MOV w0000 #1
      JP LOOPTOP
  ENDIF
END_FUNC
```

(3) JP 無条件ジャンプ

コマンド書式

JP lbl

動作

指定のラベルにジャンプします。

(4) CMP 比較

コマンド書式

CMP cmp_optor DEST SRC

動作

SRC と DEST を比較します。

比較結果をシステムメモリ CMPFLG に格納します。

cmp_optor は比較演算子を指定します。以下のものが指定できます。

CMP コマンドは BR,BSUB コマンドと組み合わせて使用します。

=:if DEST 値==SRC 値 then CMPFLG=1 else CMPFLG =0 (等しい場合)

>:if DEST 値>SRC 値 then 同上 (より大きい場合)

<:if DEST 値<SRC 値 then 同上 (より小さい場合)

>=,=> : if DEST 値>=SRC 値 then 同上 (以上)

<=,=< : if DEST 値<=SRC 値 then 同上 (以下)

!=:if DEST 値！=SRC 値 then 同上 (等しくない)

SRC,DEST がテキストメモリの場合は=のみ使用できます。

使用例

```
FUNC _ボタン1_押されたときの動作
  CMP != w0000 #0
  BR STEP2
  MOV w0000 #1
  LBL STEP2
END_FUNC
```

(5) BR 条件付ジャンプ

コマンド書式

BR lbl

動作

CMPFLG が 1 のとき指定のラベルにジャンプします。

(6) SUB サブルーチンコール

コマンド書式

SUB lbl

動作

FUNC～END_FUNC コマンドで作成したサブルーチンを呼び出します。

呼出し先で END_FUNC,EXIT_FUNC コマンドを実行すると、呼び出し元に戻ります。

使用例

```
FUNC AAA
  MOV b0000 #0
END_FUNC

SUB AAA
```

(7) BSUB 条件付サブルーチンコール

コマンド書式

BSUB lbl

動作

CMPFLG が 1 のとき FUNC～END_FUNC コマンドで作成したサブルーチンを呼び出します。

呼出し先で END_FUNC,EXIT_FUNC コマンドを実行すると、呼び出し元に戻ります。

使用例

```
FUNC AAA
  MOV b0000 #0
END_FUNC

CMP != b0000 #1
BSUB AAA
```

4. 2. 2 アクション実行制御コマンド

イベントの登録、実行の制御を行います。
メモリ操作を行うと、基本的にイベント検索が行われます。DEST で指定されているメモリがイベントテーブルに存在する場合、それに対応するマクロ(LWH,WHN コマンドで指定した lbl からのマクロ)が実行マクロキューにエントリーされます。
イベントテーブルが大きな場合(画面上にある動的オブジェクト数が多い場合)、このイベント検索が処理上のオーバーヘッドになります。
その場合、DISABLE_EVENT コマンドや VALSET コマンドを使用してイベント検索を禁止することでこのオーバーヘッドを排除することが可能です。

(1) WHN グルーバルイベント登録

コマンド書式

WHN lbl DEST

動作

グローバルイベント定義テーブルに DEST のメモリを登録します。DEST のメモリ変化時 lbl の位置のマクロをサブルーチンコールします。
グローバルイベントは 1 度登録すると、再起動するまでその情報は保持されます。

使用例

本コマンドは監視オブジェクト(継続)を配置した場合に TPD V4 により使用されます。TPD V4 は監視オブジェクト(継続)を配置すると以下のようなマクロを登録し、配置ページ表示時に実行されます。

```
ADIM _監視0_ONCE b
IF _監視0_ONCE=#0 THEN
    WHN _監視0_動作 リンクメモリ
    VALSET _監視0_ONCE #1
ENDIF
```

通常本コマンドをユーザーで記述する必要はありません。

※WHN コマンドは重複登録すると、内部のテーブルを必要以上に消費するため重複登録防止のフラグを設けて登録します。

(2) LWH ページ内イベント登録

コマンド書式

LWH lbl DEST

動作

ページ内イベント定義テーブルに DEST のメモリを登録します。DEST のメモリ変化時 lbl の位置のマクロをサブルーチンコールします。
ページ内イベントはページを変更すると登録された情報はクリアされます。

使用例

本コマンドは監視オブジェクト(表示中)を配置した場合に TPD V4 により使用されます。TPD V4 は監視オブジェクト(表示中)を配置すると以下のようなマクロを登録し、配置ページ表示時に実行されます。

```
LWH _監視0_動作 リンクメモリ
```

通常本コマンドをユーザーで記述する必要はありません。

※本コマンドは監視オブジェクトだけではなく、カウンター、ランプなどの標準のオブジェクトも使用しています。

(3) DOEVT キューのイベント実行

コマンド書式

DOEVT

動作

マクロ中に発生したイベントによる動作を実行します。
マクロ中でループ等を行うと、ループ中に発生したイベントは実行中のマクロが終了するまで実行されないため、画面の表示等も変化しませんが、ループ中に本コマンドと次の REFSH コマンドを実行することによりループ中も他の動作を割り込ませることが可能になります。

使用例

ボタン1の動作として以下のようなマクロを記述し画面上に w0002 にリンクしたカウンタを配置します。

```
FUNC _ボタン1_押したときの動作
FOR w0000=#0 TO #10
     MOV w0002 w0000･･･①
NEXT
END_FUNC
```

上記の動作としてはボタンを押すと、カウンタの値は 10 が表示されループ途中の加算過程は表示されません。
これは、①の行でリンクメモリの変化によりカウンタの描画イベントがキューに登録されますが、登録されたイベントの実行タイミングは、ボタン 1 を押したイベントの終了後(上記マクロが終了後)であり、その時点での w0002 の値を元にカウンタの描画を行うため、途中値が表示されません。途中値を表示したい場合、実行後すぐにカウンタの描画イベントを発生させる必要があります。その為に使用するのが DOEVT コマンドです。
上のマクロを下のように修正することで途中値を表示できます。

```
FUNC _ボタン1_押したときの動作
FOR w0000=#0 TO #10
     MOV w0002 w0000･･･
     DOEVT････
     REFSH････
NEXT
END_FUNC
```

このようにすることで登録されたイベントをタイミングで実行させることが出来ます。但し描画に関する処理は、全て非表示プレーンに対して行われますのでこれを表示プレーンに反映するための REFSH コマンドが必要となります。発生するイベントがカウンタ等表示に反映されるものでない場合(監視オブジェクトなど)REFSH コマンドはなくてもかまいません。

本コマンドを使用する場合、以下に注意してご使用下さい。

- DOEVT コマンドをサブルーチン中に記述し、その呼び出しが多重となる場合意図しない実行順となる可能性がありますのでサブルーチン内のループには記述しないようにして下さい。
- 周期動作を行う監視オブジェクトのマクロ中にループをおき、その中で DOEVT コマンドを実行した場合マクロが多重となり、意図しない実行順となる可能性がありますので周期動作を行うマクロ内で DOEVT コマンドを含むループを実行しないで下さい。
- DO～LOOP コマンドなどで無限ループをつくりその中で DOEVT コマンドを実行した場合、オブジェクトなどの表示反映などは正常に動作いますが、ページ遷移などアイドル状態に戻る必要がある動作は正しく行えません。

(4) REFSH バックプレーン描画の表画面への反映

コマンド書式

REFSH

動作

画面表示を更新します。

使用例

(3)項 参照

※GOP は画面表示を内部バッファに作成しマクロ終了時のタイミングで表示プレーンに転送しています。そのため画面変化があるような動作(カウンタにリンクしたメモリの変更等)を実行してもマクロが終了するまで表示に反映されません。これをマクロ途中で表示に反映するため本コマンドを使用します。

(5) REDRAW ページ再描画

コマンド書式

REDRAW

動作

表示中のページを再読み込みします。

使用例

```
FUNC _ボタン1_押したときの動作
  REDRAW
END_FUNC
```

※ページ表示の動作、マクロ挿入オブジェクトのマクロに使用すると無限連鎖発生の要因となりますので使用しないで下さい。

(6) DISABLE_EVENT イベントの禁止

コマンド書式

DISABLE_EVENT

動作

メモリ変化に伴うイベント検索を禁止します。
イベント検索禁止状態は ENABLE_EVENT コマンドおよび END コマンド(END_FUNC,EXIT_FUNC コマンド)実行まで継続します。
イベント検索禁止状態中は、メモリ変化に伴う、イベント検索が禁止されるため新たに実行マクロキューへの登録がされないため、描画の変更(ランプ等)やマクロ実行(監視オブジェクト)は行われません。またイベント検索が行われないことおよび連鎖的なマクロが起動しないことからマクロの処理は高速化されます。

使用例

```
FUNC _ボタン1_押したときの動作
  DISABLE_EVENT
  MOV w0000 #1
  ENABLE__EVENT
  MOV w0002 #2
END_FUNC
```

上記で w0000 にリンクしているオブジェクトの表示は変化しませんが、w0002 にリンクしているオブジェクトは表示変化します。

(7) ENABLE_EVENT イベントの許可

コマンド書式

ENABLE_EVENT

動作

イベント発生禁止状態を解除します。

4. 2. 3 整数演算系コマンド

※整数演算系のコマンドはほとんどの場合構文マクロコマンド(EXPR コマンド)でより簡潔な表記が可能ですので構文マクロコマンドの使用を推奨します。

(1) MOV 値の代入

コマンド書式

MOV DEST SRC

動作

DEST←SRC

SRC で指定された値を DEST にコピーします。

SRC が文字列または文字列型時は、数値として読めるデータの場合数値に変換され DEST にコピーされます。

DEST が文字列型の場合はエラーとなります。

使用例

```
MOV w0000 #123
```

(2) ADD 加算

コマンド書式

ADD DEST SRC

動作

DEST←DEST+SRC

SRC で指定された値とコマンド実行前の DEST の値を加算した値を DEST にセットします。

DEST が文字列型の場合はエラーとなります。

使用例

```
ADD w0000 #1
```

(3) DEL 減算

コマンド書式

DEL DEST SRC

動作

DEST←DEST-SRC

コマンド実行前の DEST の値から SRC で指定された値を減算した値を DEST にセットします。

DEST が文字列型の場合はエラーとなります。

使用例

```
DEL w0000 #1
```

(4) MUL 掛算

コマンド書式

MUL DEST SRC

動作

DEST←DEST*SRC

SRC で指定された値とコマンド実行前の DEST の値を乗算した値を DEST にセットします。

DEST が文字列型の場合はエラーとなります。

使用例

```
DEL w0000 w0000
```

(5) DEV 割算

コマンド書式

DEV DEST SRC

動作

DEST←DEST/SRC

コマンド実行前の DEST の値から SRC で指定された値を除算した値を DEST にセットします。

DEST が文字列型の場合はエラーとなります。

SRC が 0 のときはエラーとなります。

※SRC に#0 が指定されていた場合、画面転送時にエラーチェックが行われますが、SRC にメモリを指定し、実行時にメモリ値が 0 になっていた場合などは事前チェックでは検出できません。その場合 ERR メモリに 0 除算検出(1)がセットされます。なお 0 除算検出された場合 DEST のメモリに変化はありません。

使用例

```
DEV w0000 w0002
```

(6) MOD 剰余算

コマンド書式

MOD DEST SRC

動作

DEST←DEST%SRC

コマンド実行前の DEST の値から SRC で指定された値を除算した余りを DEST にセットします。

DEST が文字列型の場合はエラーとなります。

SRC が 0 のときはエラーとなります。

※0 除算時の動作は(5)項を参照。

使用例

```
MOD w0000 #3
```

(7) VALSET 値の代入(イベント発生なし)

コマンド書式

VALSET DEST SRC

動作

DEST←SRC

MOV と同等動作です。但しイベント検索が発生しません。

以下動作と等価です。

```
DISABLE_EVENT
MOV DEST SRC
ENABLE_EVENT
```

※本コマンド使用により DISABLE_EVENT,ENABLE_EVENT コマンドにより設定されたイベント検索/許可/禁止の状態を変化させることはありません。

使用例

イベント検索を発生させたくない値セット動作に使用します。

上記のように、DISABLE_EVENT 状態にして MOV を行っても同様の動きです。

```
VALSET w0000 #0
```

(8) NOZERODEC 非ゼロ条件付減算

コマンド書式

NOZERODEC DEST SRC

動作

DEST が 0 でなければ DEL と同等動作です。但しイベント検索が発生しません。
DEST が 0 の時は何も動作しません。
以下動作と等価です。

```
CMP = DEST #0
BR _DUMMYLBL
DISABLE_EVENT
DEL DEST SRC
ENABLE_EVENT
LBL _DUMMYLBL
```

使用例

タイマの数を拡張したい場合などを想定しています。
タイマー系のシステムメモリ(SYSCOUNT など)にリンクした監視オブジェクトの動作で以下のように記述すればそれぞれをタイマと同様な動きにすることが可能です。(但しそれぞれのメモリにリンクしたオブジェクトにイベントは発生しません。)

```
NOZERODEC w0000
NOZERODEC w0002
NOZERODEC w0004
NOZERODEC w0006
```

4. 2. 4 実数演算系コマンド

数学関数を使用するためのコマンドです。
但し実数演算系のコマンドは EXPR コマンドの式に含めることは出来ません。実数演算を式に含める場合、次のように式を分ける必要があります。

$$y = \mathrm{r} \times \sin\theta$$

↓

```
ADIM temp F
ADIM R F
ADIM θ F
ADIM Y F
SIN temp θ
EXPR Y=temp*R
```

(1) SIN　正弦

コマンド書式
SIN DEST SRC
動作
DEST←sin(SRC)
SRC は度で指定します。
DEST に指定可能なのは F 型メモリのみです。
使用例

```
SIN F0000 #45
```

(2) COS　余弦

コマンド書式
COS DEST SRC
動作
DEST←cos(SRC)
SRC は度で指定します。
DEST に指定可能なのは F 型メモリのみです。
使用例

```
COS F0000 #45
```

(3) TAN　正接

コマンド書式
TAN DEST SRC
動作
DEST←tan(SRC)
SRC は度で指定します。
DEST に指定可能なのは F 型メモリのみです。
使用例

```
TAN F0000 #45
```

(4) ASIN　逆正弦

コマンド書式
ASIN DEST SRC
動作
DEST←arcsin(SRC)
DEST は度で格納されます。
SRC は-1～1 の範囲で指定して下さい。
DEST に指定可能なのは F 型メモリのみです。
使用例

```
ASIN F0000 @0.5
```

(5) ACOS　逆余弦

コマンド書式

ACOS DEST SRC

動作

DEST←arccos(SRC)

DEST は度で格納されます。

SRC は-1～1 の範囲で指定して下さい。

DEST に指定可能なのはＦ型メモリのみです。

使用例

```
ACOS F0000 @0.5
```

(6) ATAN　逆正接

コマンド書式

ATAN DEST SRC

動作

DEST←atan(SRC)

DEST は度で格納されます。

DEST に指定可能なのはＦ型メモリのみです。

使用例

```
ATAN F0000 @0.5
```

(7) SQR　平方根

コマンド書式

SQR DEST SRC

動作

DEST に SRC の平方根をセットします。

DEST に指定可能なのはＦ型メモリのみです。

使用例

```
ATAN F0000 @0.5
```

(8) ABS　浮動小数点絶対値

コマンド書式

ABS DEST SRC

動作

DEST←|SRC|(絶対値)

DEST に SRC の絶対値をセットします。

使用例

```
ABS F0000 @-123.4
```

(9) RND　乱数生成

コマンド書式

RND DEST

動作

DEST←rand()

DEST には w,W,I,L,F 型を指定して下さい。

乱数は 0 から 2,147,483,648 まで範囲で発生します。

DEST が I､L､F の場合は上記の範囲、それ以外の型は型毎の指定可能範囲の値が DEST に格納されます。

使用例

```
RND w0000
```

4. 2. 5 論理演算コマンド

整数型メモリをビットの集合とみなし、個々のビットごとに論理演算を行います。
※論理演算コマンドはほとんどの場合構文マクロコマンド(EXPR コマンド)でより簡潔な表記が可能ですので構文マクロコマンドの使用を推奨します。

(1) OR 論理和

コマンド書式

OR DEST SRC

動作

DEST←DEST|SRC

コマンド実行前の DEST の値と SRC で指定された値の論理和を DEST にセットします。
DEST,SRC は整数型のみ使用できます。

使用例

```
OR bOOOO #&H01
```

(2) AND 論理積

コマンド書式

AND DEST SRC

動作

DEST←DEST&SRC

コマンド実行前の DEST の値と SRC で指定された値の論理積を DEST にセットします。
DEST,SRC は整数型のみ使用できます。

使用例

```
AND bOOOO #&HFE
```

(3) XOR 排他的論理和

コマンド書式

XOR DEST SRC

動作

DEST←DEST^SRC

コマンド実行前の DEST の値と SRC で指定された値の排他的論理和を DEST にセットします。
DEST,SRC は整数型のみ使用できます。

使用例

```
XOR bOOOO #&HFE
```

(4) NOT ビット反転

コマンド書式

NOT DEST

動作

DEST←~DEST

コマンド実行前の DEST の値をビット反転した値が DEST にセットされます。DEST は整数型のみ使用できます。

使用例

```
NOT bOOOO
```

4. 2. 6 文字列操作コマンド

文字列を組み立てるために使用するコマンドです。

(1) CP　文字列コピー

コマンド書式

CP length DEST SRC

動作

SRC で指定された文字列を DEST にコピーします。
その際 length で指定された文字以上はコピーされません。
SRC が文字型,文字列以外の場合エラーとなります。

使用例

```
CP 10 T0000 $あいうえお
CP 10 T0000 T0010
```

(2) APD 文字列追加

コマンド書式

APD length DEST SRC

動作

SRC で指定された文字列を DEST の末尾に追加します。その際 DEST が length で指定された文字以上になる場合、オーバーする範囲は追加されません。
SRC が文字型,文字列以外の場合エラーとなります。

使用例

```
APD 10 T0000 $/
APD 10 T0000 T0010
```

(3) FMT　数値の文字列化

コマンド書式

FMT alllength floatlength fmt DEST SRC

動作

SRC で指定した数値データを指定書式に基づき文字列化し DEST にセットします。

書式指定

alllength　変換後の文字列の総桁数。 桁数には小数点、符号領域も含みます。
最大値は 10 です。

floatlength　変換後の文字列の小数点以下の桁数。
最大値は alllength-2 までです。
SRC が F 型の場合 小数点位置による四捨五入を行います。
floatlength=2 の場合
12.3456 → 12.36
SRC が F 型以外の場合桁固定の擬似小数点
(指定値文の下位桁を少数点以下として表示します。)
floatlength=2 の場合
123456 → 123.46

fmt(alllength=5,SRC=123 のとき)
0：0 サプレスあり符号領域あり(″　123″)
1：0 サプレスなし符号領域あり(″ 00123″)
2：0 サプレスあり符号領域なし(″　123″)
3：0 サプレスなし符号領域なし(″00123″)

表示オーバー時の処置
表示範囲に収まらないほど絶対値が大きい値　″E″
表示範囲に収まらないほど絶対値が小さい値　″0″

使用例

```
FMT 5 0 0 T0000 w0000
```

(4) BS　一文字消去

コマンド書式

BS DEST

動作

DEST の末尾 1 文字を削除します。

※BS コマンドは日本語モードでのみご使用下さい。

使用例

BS T0000

4. 2. 7 ホスト通信関連コマンド

GOP プロトコルを使用して、GOP に接続されているホスト機器に対し GOP 側からの割り込みコマンドを出力します。

(1) OUTPUT　ホストポート出力

コマンド書式

OUTPUT SRC

動作

SRC の内容をホストポートに出力します。

SRC が T 型メモリまたは文字列の場合は、文字列をそれ以外の場合は数値を文字列化し出力します。出力の形式は、メモリの中身の出力の場合は、メモリ読み出しコマンドの書式("[メモリ名]=[値]")、その他の返信に関しては、TPD V4 にて指定する識別コードを付加します。

TPD V4 での識別コード　　# 数値
$ 文字列
@ 浮動小数点

コマンド列の先頭に識別子の " A " を付加します。

出力例

①wf000 の値を出力する場合

マクロ

```
OUTPUT wf000
```

出力

```
<STX>Awf000=1<ETX><CSM><CRLF>
```

②文字列 " abc " を出力する場合

マクロ

```
OUTPUT $abc
```

出力

```
<STX>A$abc<ETX><CSM><CRLF>
```

使用例

OUTPUT コマンドは GOP で発生した動作をホストに通知する場合などに使用します。

例えば、ボタンに OUTPUT コマンドを記述していた場合、そのボタンが押されるとホスト側にメッセージが通知されるので、ホスト側はボタンが押されたことを知ることが出来ます。

```
FUNC _ボタン 1_押されたときの動作
  OUTPUT $BTN1_ONPRESS
END_FUNC
```

ボタンごとに異なるメッセージにしておくことで、ホスト側はボタンごとに処理を分けることが出来ます。

また、マクロのデバッグ中に、特定のメモリを出力したり、特定の経路を通ったかを確認することが出来ます。

(2) OUTPUTH　16 進数でのホストポート出力

コマンド書式

OUTPUTH SRC

動作

データ構造、動作は OUTPUT コマンドと同様ですが、出力形式が異なり 16 進数状の文字列として出力します。

出力例

①w0000=100(10 進数)のとき

マクロ

```
OUTPUTH w0000
```

出力

```
<STX>Aw0000=H0064<ETX><CSM><CRLF>
```

使用例

OUTPUT コマンドと同様ですが、出力値を 16 進数表記で得たい場合に使用します。

(3) OUTPUT_VAL　値のみのホストポート出力

コマンド書式

OUTPUTH_VAL SRC

動作

データ構造、動作は OUTPUT コマンドと同様ですが、SRC にメモリを指定時、メモリの情報は出力せず値のみを出力します。(OUTPUT コマンドで SRC にメモリ以外を指定したときと同じ出力書式。)

SRC にメモリ以外を指定時は OUTPUT と同じ動作です。

出力例

①w0000=100(10 進数)のとき

マクロ

```
OUTPUT_VAL w0000
```

出力

```
<STX>A#100<ETX><CSM><CRLF>
```

②T0010="ABCDEFG"のとき

マクロ

```
OUTPUT_VAL T0010
```

出力

```
<STX>A$ABCDEFG<ETX><CSM><CRLF>
```

③文字列"ABCDEFG"を直接指定のとき

マクロ

```
OUTPUT_VAL $ABCDEFG
```

出力

```
<STX>A$ABCDEFG<ETX><CSM><CRLF>          ·····②と同じ出力になります。
```

使用例

OUTPUT コマンドと同様ですが、メモリ情報を出力したくない場合に使用します。

4. 2. 8 サウンド再生機能

TPD V4 で登録された WAV データの再生制御を行います。
※本機能は GOP-H クラスのみの機能です。

(1) PLAYSOUND　WAVデータの再生

コマンド書式
PLAYSOUND op NO

動作
指定された NO の WAV データを再生します。
op=0：1 回再生 1：ループ再生

使用例
```
PLAYSOUND 0 #1
```

(2) STOPSOUND　WAV再生の停止

コマンド書式
STOPSOUND

動作
WAV 再生を停止します。

使用例
```
STOPSOUND
```

(3) LOAD_WAV　ファイル名指定でのWAVデータ再生

コマンド書式
LOADWAV op FILENAME

動作
指定された FILENAME の WAV データを再生します。
op=0：1 回再生 1：ループ再生

使用例
USB メモリの WAV データや実行時のファイル転送で取得した WAV データを再生するために使用します。
```
LOADWAV 0 $0:BELL.WAV
```

(4) RESERVE_SOUND 次再生ファイルの予約

※GOP-5000H クラスのみの機能です。

コマンド書式

RESERVE_SOUND STATEMEM WAV_FILE

動作

現在再生中の WAV データの終了後に続けて再生する WAV データを設定します。
再生する WAV データはすべてメモリにロードされるため、大きなサイズの WAV データはメモリ不足で再生できない、または GOP の動作に影響がある場合があります。
本コマンドで小さい WAV データを連続して再生することで GOP へのメモリ負荷の減少が図れます。
予約した WAV データの再生が開始すると STATEMEM に 1 をセットします。
STATEMEM は b 型メモリを指定して下さい。
STATEMEM を監視オブジェクトにリンクすることで連続して再生することができます。
WAV_FILE は予約する WAV データのファイル名またはファイル名が保存されているテキストメモリを指定します。

使用例

DATA0.WAV, DATA1.WAV, DATA2.WAV, DATA3.WAV を連続して再生する場合
再生開始のボタンのマクロ

```
LOADWAV 0 $2:DATA0.WAV
RESERVE_SOUND b0100 $2:DATA1.WAV
```

監視オブジェクト(b0100 にリンク)

```
RESERVE_SOUND b0101 $2:DATA2.WAV
```

監視オブジェクト(b0101 にリンク)

```
RESERVE_SOUND b0102 $2:DATA3.WAV
```

4. 2. 9 マクロポート(無手順シリアル)通信関連コマンド

マクロポートは、無手順で 1 バイト単位で送受信可能です。以下はマクロポートから読み書きするためのコマンドです。

(1) PUTC　マクロポートへの 1 バイト出力

コマンド書式

PUTC SRC

動作

SRC で指定される値をマクロポートへ出力します。
指定された値が b 型でない場合、b 型に変換されて出力します。

使用例

```
PUTC #&H41
```

(2) GETC　マクロポートからの 1 バイト入力

コマンド書式

GETC DEST

動作

DEST で指定されるメモリへマクロポートから 1 バイト入力します。
マクロポートへデータがない場合-1 が格納されます。ある場合は受信したバイトデータ(値範囲 0～255)がセットされます。
DEST には W 型メモリを指定して下さい。

使用例

```
GETC W0000
```

4. 2. 10 高精度タイマ制御関連

GOP ではタイマメモリの他に、1ms 単位で時間を計測できる高精度タイマを使用することが出来ます。高精度タイマはタイマカウント開始時刻からカウント値取得までの経過時間を ms 単位で取得することが出来ます。
タイマ計測が不要になればカウントを停止します。

例

```
;10000 回ループの時間を計測
;タイマを開始
HRTSTA
;10000 回のループ実行
FOR I0000=#1 TO #10000
(繰り返し処理を記述)
NEXT
;経過時間を取得
HRTGET I0004
;ホストに結果を通知
OUTPUT I0004
;タイマを停止
HRTSTP
```

(1) HRTSTA　高精度タイマのカウント開始

コマンド書式

HRTSTA

動作

高精度タイマのカウントを 0 にリセットし 1ms 毎のカウントアップを開始します。

(2) HRTSTP　高精度タイマのカウント停止

コマンド書式

HRTSTP

動作

高精度タイマのカウントを停止します。

(3) HRTGET　高精度タイマのカウント値取得

コマンド書式

HRTGET DEST

動作

高精度タイマの HRTSTA コマンド実行時からの経過時間を ms 単位で取得します。

4. 2. 11 SENDMAIL メール送信機能

※本機能は GOP-H クラスのみの機能です。

GOP にオプションのイーサネット拡張ユニットが組み込まれている場合、GOP からメール送信を行うことが出来ます。(外部に GOP がアクセスできる SMTP サーバーの用意が必要です。)

(1) SENDMAIL　メール送信

コマンド書式

SENDMAIL MSG

動作

MSG の内容に従いメールを送信します。

MSG は以下のフォーマットの文字列となります。

送信者アドレス:宛先[;(複数ある場合はセミコロン区切り)宛先 2]:件名:本文(宛先は TO:で CC:BCC:は指定できません。また同時指定件数は 16 件までです。)

未達確認はできません。

使用例

以下で d-support@ishiihyoki.co.jp 宛に以下の内容でメール送信されます。

送信者　GOP4000@ishiihyoki.co.jp

件名 メールのテスト

本文　このメールは GOP から送信されました。

```
SENDMAIL $GOP4000@ishiihyoki.co.jp:d-support@ishiihyoki.co.jp:メールのテスト:このメールは GOP から送信されました。
```

4. 2. 12 リセット

GOP を再起動させます。

(1) RESET　リセット

コマンド書式

RESET

動作

GOP をハードリセットします。

4. 2. 13 トレンドログバッファ制御

トレンドログバッファはシステムメモリの LOGPUT_*,LOGGET_*を介してアクセスしますが、これを初期化するためのコマンドを用意しています。

(1) TRENDBUF_FILL　トレンドログバッファ領域のデータフィル

コマンド書式

TRENDBUF_FILL CH DATA

動作

CH で指定されたトレンドログバッファを DATA で指定された値で埋めます。

CH の値が 1～8 以外のときは何もしません。

使用例

トレンドログバッファの Ch1 を全て 0 で埋めます。

```
TRENDBUF_FILL #1 #0
```

4. 2. 14 ファイル操作コマンド

・ファイル操作概要

GOP に挿した USB メモリや内部の RAM ディスクに対して、ファイルの読み込みまたは書き込み操作を行うことが出来ます。

GOP が扱えるドライブは以下の通りです。

・ドライブ 0　FAT16　10MB(揮発性)
・ドライブ 1　FAT12　1MB(不揮発性)※
・ドライブ 2　接続された USB メモリに依る
・ドライブ 3　フラッシュ(書き込み不可)

※ドライブ 1 は GOP-H クラスのみ使用できます。

またディレクトリによる階層化が出来ます。ディレクトリ区切りは'/'を使用します。

ファイル名は 8.3 形式ファイル名に対応しています。(大文字小文字区別しません。)

例えば、USB メモリの LOG ディレクトリの LOG.TXT のファイルは以下のように表現します。

ファイル指定文字列　2:/LOG/LOG.TXT

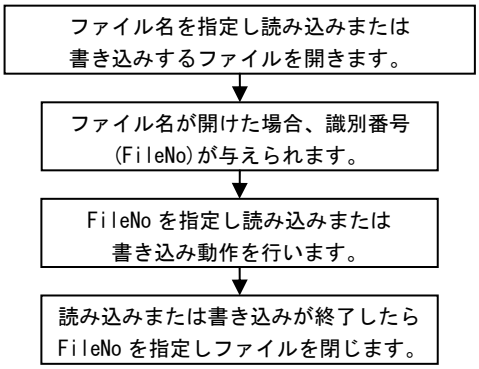
またドライブのみを指定する場合は以下のようにします。(USB メモリの場合。)

ドライブ指定文字列　2:

またディレクトリを指定する場合、ディレクトリ名の末に"/"をつけます。

ディレクトリ指定文字列　2:/LOG/

ファイル操作を行う場合、流れは以下のようになります。



識別番号(FileNo)は数値メモリで管理します。ファイルオープンの成否はファイルオープンコマンド実行後の FileNo のメモリの状態で知ることが出来ます。

またファイルは同時に 4 つまで開くことが出来ます。

・ファイル操作コマンド使用時の注意事項

ファイル操作関連コマンドはブロッキングコマンドであり、その処理が終わるまで GOP の動作を拘束し、画面操作や通信応答などが行なえなくなります。RAM ディスクに対しての操作は瞬時に終わりますが、操作対象が USB メモリの場合、USB メモリの種類によっては極端に時間がかかる場合があります。

そのためマクロ等で USB メモリを扱う場合、USB メモリアクセス中はホストとの通信を行なわない(開始・終了時に通信出力を行い、終了まで通信を抑制する等。)設計をして下さい。

(1) OPENFILE　ファイルオープン

コマンド書式

OPENFILE FILENO mode FILENAME

動作

ファイルを開きます。同時にファイル操作に使用する識別番号を FILENO に受け取ります。
FILENO は整数型メモリを指定します。
FILENAME は文字列メモリまたは文字列を指定します。
FILENAME にはファイル指定文字列を指定して下さい。
ファイルオープンについては、mode で３パターンから選択します。mode は数値のみ指定です。(#は不要です。)

mode=　0：読み込みオープン
　　1：書き込みオープン
　　3：追記オープン

書き込みオープン時、ファイルが存在しない場合は自動ファイル作成を行います。また、ディレクトリが存在しない場合は、ディレクトリも自動作成します。
ファイルオープンに成功した場合は、FILENO に 1～4 が入ります。
その後のファイル操作に FILENO を使用します。
(FILENO は複数ファイルを開いた際の管理番号になります。 ファイルクローズまで保持して下さい。)
以下の場合は、ファイルオープンエラーが発生します。オープンエラーが発生すると FILENO のメモリに 0 が入ります。

・読み込みオープン時、存在しないファイルを開く。
・追記オープン時、存在しないファイルを開く。
・既に開かれているファイルを開く。
・最大同時オープン数(4 ファイル)を越えてファイルオープンを行う。

また、ファイルオープンした際のファイルポインタについては読み込みオープン・書き込みオープンの時はファイルの先頭、追記オープン時はファイル末尾となります。

使用例

```
ADIM FNO w
OPENFILE FNO 1 $0:/log/log.txt
```

(2) CLOSEFILE　ファイルクローズ

コマンド書式

CLOSEFILE FILENO

動作

OPENFILE コマンドで開いたファイルを閉じます。
FILENO は OPENFILE コマンドで使用したものと同じメモリを使用します。

使用例

```
ADIM FNO w
OPENFILE FNO 1 $0:/log/log.txt
  :(ファイル読み書きに関する処理)
  :  この間 FNO のメモリを書き換えることは不可
CLOSEFILE FNO
```

(3) WRITEFILE　ファイルに指定バイト書き込み

コマンド書式

WRITEFILE FILENO ADDR SIZE EXCUTESIZE

動作

指定されたファイル(FILENO)に ADDR の位置からのデータを指定バイト数(SIZE)書き込みます。
コマンド実行後、実際に書き込まれたサイズが EXCUTESIZE にセットされます。
書き込み後、ファイルポインタは書き込んだ数だけ進みます。
ADDR､SIZE は整数値または整数型メモリを､EXCUTESIZE は整数型メモリを指定して下さい。(SIZE に応じた型を指定して下さい。)

使用例

WRITEFILE コマンドはデータをバイナリで扱う場合に適します。文字列として扱う場合は後述の WRITELINE コマンドの方が容易です。

```
OPENFILE FNO 1 $2:/log/log.txt
WRITEFILE FNO #&H0100 #&H20 w0000
CLOSEFILE FNO
```

GOP のメモリの 0100 番地から 32(20H)byte(0120 番地まで)をファイルに出力します。

(4) READFILE　ファイルから指定バイト読み込み

コマンド書式

READFILE FILENO ADDR SIZE EXCUTESIZE

動作

指定されたファイル(FILENO)から ADDR の位置にデータを指定バイト数(SIZE)を読み込みます。
コマンド実行後、実際に読み込まれたサイズが EXCUTESIZE にセットされます。
読み込み後、ファイルポインタは読み込んだ数だけ進みます。
EXCUTESIZE と SIZE が異なる場合、ファイルを最後まで読み込んだと判断できます。
ADDR､SIZE は整数値または整数型メモリを､EXCUTESIZE は整数型メモリを指定して下さい。(SIZE に応じた型を指定して下さい。)

使用例

READFILE コマンドはデータをバイナリで扱う場合に適します。文字列として扱う場合は後述の READLINE コマンドの方が容易です。

```
OPENFILE FNO 0 $2:/log/log.txt
READFILE FNO #&H0100 #&H20 w0000
CLOSEFILE FNO
```

GOP のメモリの 0100 番地から 32(20H)byte(0120 番地まで)にファイルの内容を書き込みます。

(5) SEEKFILE　ファイルの読み書き位置を指定

コマンド書式

SEEKFILE FILENO LOCATE mode

動作

指定ファイル(FILENO)のファイルポインタを mode に応じて操作します。
ファイルサイズを超えた LOCATE 値が指定された場合、ファイル末尾に移動し LOCATE に末尾位置をセットします。(mode=3 と同じ動作)
LOCATE は整数値または整数型メモリ(mode=2 のときは整数型メモリのみ)を指定して下さい。
mode=2,3 の場合には、LOCATE は 0 固定として下さい。
mode は数値のみなので#は不要です。
mode=　0：指定位置：LOCATE に移動
　　1：現在位置をメモリ：LOCATE に取得
　　2：ファイルの先頭に移動
　　3：ファイルの末尾に移動

使用例

SEEKFILE コマンドはファイルのデータを読み飛ばしたい場合や読み直し、書き直しを行いたい場合に使用します。
以下は読み飛ばす例です。

```
OPENFILE FNO 0 $2:/log/log.txt
READFILE FNO #&H0100 #&H20 w0000
SEEKFILE FNO #&H100 0    ･･･log.txt の 100 バイト目に移動
READFILE FNO #&H0200 #&H20 w0000
CLOSEFILE FNO
```

上記で log.txt の 0～32(20H)byte 目のデータが 0100 番地に 256(100H)～288(120H)byte 目のデータが 0200 番地から書き込まれます。

(6) WRITELINE　ファイルに 1 行書き込み

コマンド書式

WRITELINE FILENO TEXT

動作

FILENO で指定されたファイルに TEXT で指定される文字列を(終端文字'¥0'まで)書き込みます。
TEXT は文字列または T 型メモリを指定します。
また TEXT で指定される文字列を書き込んだ後、改行コード(CR+LF)を自動的に書き込みます。

使用例

```
OPENFILE FNO 1 $2:/log/log.txt
WRITELINE FNO $ABCD
WRITELINE FNO $EFG
CLOSEFILE FNO
```

上記のようにすると,USB メモリの/log/log.txt のファイルに
　　ABCD[CRLF]
　　EFG[CRLF]
と書き込まれます。

(7) READLINE　ファイルから 1 行読み込み

コマンド書式

READLINE FILENO SIZE TEXT

動作

FILENO で指定されたファイルから一行読み込み、TEXT で指定されたテキストメモリに文字列を書き込みます。

TEXT は T 型メモリを指定します。

SIZE は整数または整数型メモリを指定します。SIZE には TEXT で用意されている領域のサイズを指定して下さい。(TEXT 以降のメモリを破壊しないため。)

文字列は改行コード(CR+LF または CR 単独または LF 単独自動認識。)まで、または SIZE で指定された長さまで読み込み、終端文字列を付加します。また改行コードは削除されます。

なお SIZE を超えたときの処理の例を示します。

ファイル中の文字列

ABCD[改行]
EFGHIJK[改行]
LMN[改行]

読み込み動作

READLINE FNO #5 T0000 →T0000=ABCD
READLINE FNO #5 T0000 →T0000=EFGH
READLINE FNO #5 T0000 →T0000=IJK
READLINE FNO #5 T0000 →T0000=LMN

となります。

またファイルを最後まで読んだ場合、テキストメモリの先頭に FFh が格納されます。

使用例

ファイルを一行ずつ読み、それを OUTPUT コマンドで出力します。

```
OPENFILE FNO 0 $2:/log/log.txt
DO
  READLINE FNO #40 T0000
  IF b0000=#&HFF THEN ･･･テキストの先頭番地を確認
      EXIT
  ENDIF
  OUTPUT T0000
LOOP
CLOSEFILE FNO
```

(8) FILEOPERATION　ファイル操作

コマンド書式

FILEOPERATION mode DESTFILE SRCFILE

動作

ファイルのコピー、移動、削除を行います。ドライブを跨ぐ移動も可能です。DESTFILE・SRCFILE はドライブ名からパス指定して下さい。

※DESTFILE は文字列としての指定も可能ですが、文字列表記の例外として FILEOPERATION での使用に限り、スペースが文字区切りとみなされます。

（通常は文字列は$～行末まで。）

DESTFILE,SRCFILE は文字列または T 型メモリを指定します。

DESTFILE,SRCFILE にはファイル指定文字列を指定して下さい。

mode は数値のみのため#は不要です。

mode=　0:SRCFILE を DESTFILE にコピーします。

1：SRCFILE を DESTFILE にコピーし SRCFILE を削除します。

2：SRCFILE を削除します。(DESTFILE は無視されます。)

使用例

USB メモリの/test/test2.txt を SDRAM ドライブに test.txt の名前でコピーします。

```
FILEOPERATION 0 $0:/test.txt $2:/test/test2.txt
```

USB メモリの/test/test2.txt を削除します。

```
FILEOPERATION 2 $ $2:/test/test2.txt
```

※DESTFILE は無視しますが、指定は必要なため空を指定します。

(9) CHECK_REMAIN　ファイルシステム容量確認

コマンド書式

CHECK_REMAIN DEST SRC

動作

指定したドライブ：SRC の残容量を byte 単位の表示で DEST に取得します。

DEST は I 型メモリを指定して下さい。

SRC は数値または、整数型メモリを指定します。

SRC はドライブ番号を指定して下さい。

使用例

```
CHECK_REMAIN I0000 #2
```

(10) CHECK_REMAIN2　ファイルシステム容量確認(大容量対応)

コマンド書式

CHECK_REMAIN2 DEST SRC

動作

指定したドライブ：SRC の残容量を **Kbyte 単位**の表示で DEST に取得します。

DEST は I 型メモリを指定して下さい。

SRC は数値または、整数型メモリを指定します。

SRC はドライブ番号を指定して下さい。

※4Gbyte を超える USB メモリの残容量は CHECK_REMAIN コマンドの byte 単位の値では I 型メモリで表現することができないため Kbyte 単位の本コマンドを使用してください。

使用例

```
CHECK_REMAIN2 I0000 #2
```

(11) DIR_INFO　ディレクトリ情報の取得

コマンド書式

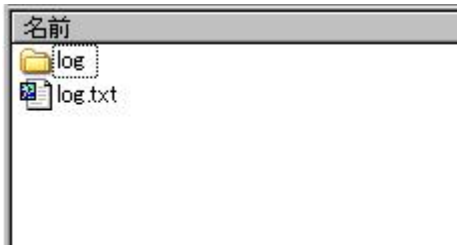
DIR_INFO DEST SRC

動作

指定したパス(SRC)のディレクトリに含まれるファイルまたはサブディレクトリ(メンバー)名を列挙します。
DIR_INFO コマンドは直前に実行した DIR_INFO コマンドと SRC が同じ場合は、次のメンバー名をそうでない場合は先頭(返す順番はメンバーの作成順です)のメンバー名を DEST に格納します。
DEST は T 型メモリを指定します。
SRC は文字列または T 型メモリを指定します。
SRC にはディレクトリ指定文字列またはドライブ指定文字列を指定して下さい。
渡したテキストメモリにメンバー名がコマンド実行毎に上書きされます。
メンバーがサブディレクトリの場合は先頭に'/'がつきます。
SRC 指定のディレクトリ中のファイルを最後まで返すと空文字'¥0'を返します。

使用例

ディレクトリ中に含まれるファイル一覧を取得する場合などに使用します。



上記ファイル構成の時

```
;コマンド              実行後の DEST メモリ状態
DIR_INFO T0000 $0:    (T0000=log.txt)
DIR_INFO T0000 $0:    (T0000=/log)
DIR_INFO T0000 $0:    (T0000='¥0' 終端文字)
```

となります。

4. 2. 15 レイヤー関連コマンド

(1) GETFRAME　現在のレイヤー番号

コマンド書式

GETFRAME DEST

動作

マクロ実行中のレイヤーを DEST に書き込みます。
DEST には整数型メモリを指定します。

使用例

同じページを異なるレイヤーに表示し、表示レイヤーごとに動作を切り替えたい場合に、現在のレイヤーを取得するために使用します。

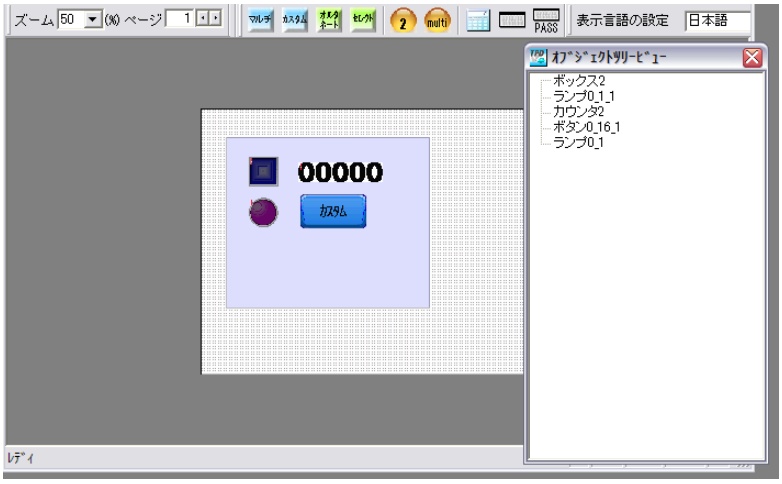
```
GETFRAME w0000
IF w0000=#1 THEN
   :
ELSE
   :
ENDIF
```

4. 2. 16 描画コマンド

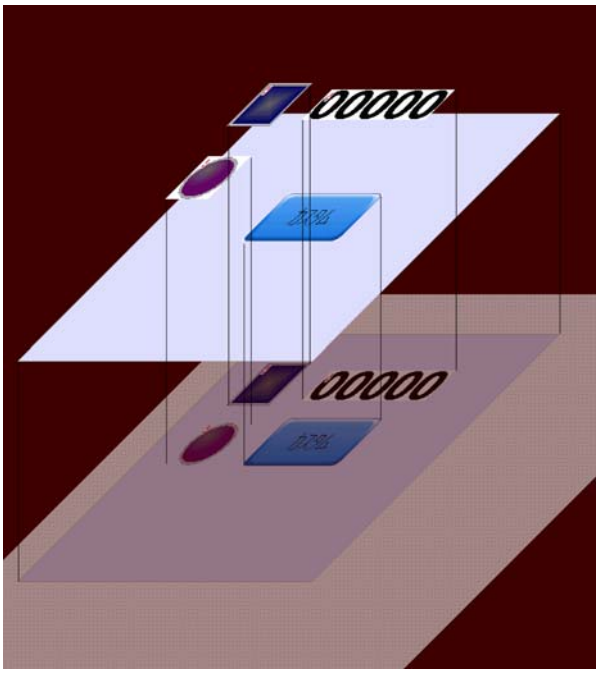
マクロで任意の描画を行うためのコマンドです。
マクロで描画する場合、GOP の描画の仕組みに関する知識が必要になりますので以下に若干の説明を行います。

・GOPの描画の仕組み

GOP の描画処理は以下のような仕組みとなっています。
TPD V4 で以下<図 1>のように設計した画面は、GOP で実行時<図 2>のように各オブジェクトのイメージ画像の重ねあわせで表現されます。

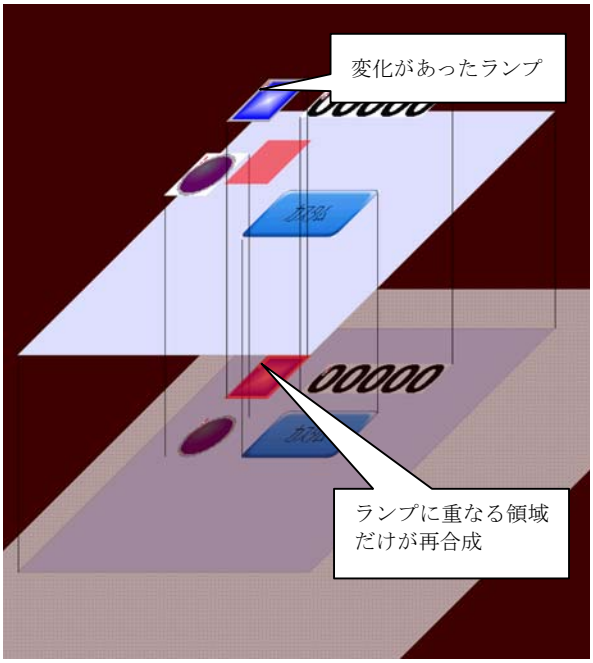


〈図 1〉



〈図 2〉

例えば、ランプの表示が変わった場合はランプのイメージ画像を書き換え再重ねあわせで表示イメージを再合成します。
但し、このとき再合成する必要があるのはランプの領域に重なる部分だけでよく、その他の部分まで再合成すると実行速度が低下するため、この部分だけを再合成します。



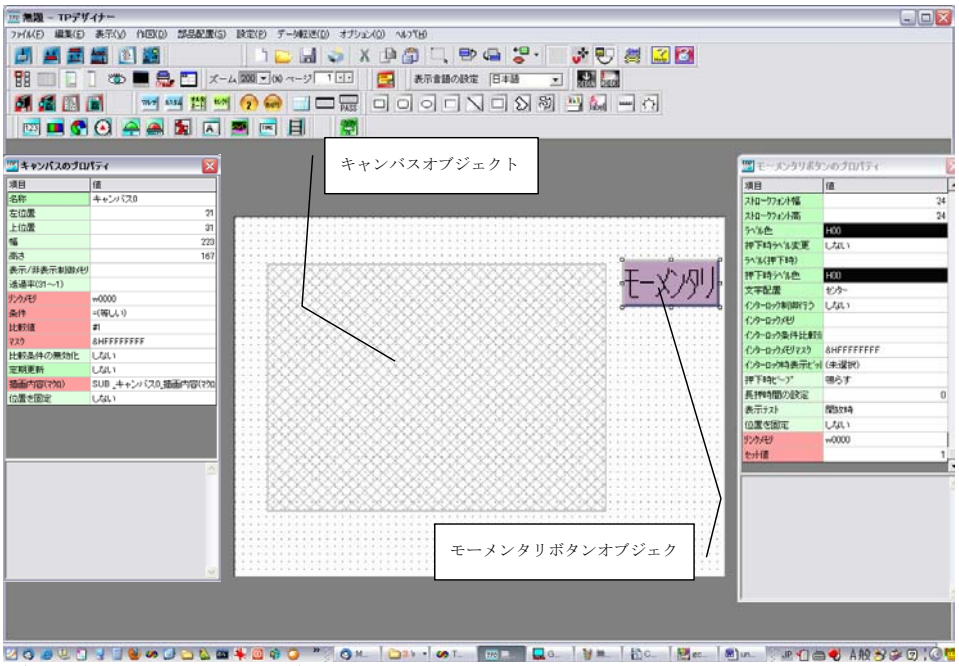
〈図 3〉

このように GOP の描画は、オブジェクトのイメージ描画のための領域が必要になります。この領域をイメージキャッシュといいます。
マクロで描画する場合も、イメージキャッシュが必要になります.
このイメージキャッシュを提供するオブジェクトがキャンバスオブジェクトになります。
マクロを使った描画は、キャンバスオブジェクトに描画を行う、といった手法になります。
また、描画が終わった場合、そのオブジェクトの更新通知を行います。
キャンバスオブジェクトには、描画開始トリガとなるリンクメモリを設定します。そのリンクメモリが指定の条件になった場合とページ読み込み時、キャンバスオブジェクトで記述したマクロ描画が行われます。(定期更新指定時は指定の周期で描画が行なわれます。)
描画マクロが呼ばれた段階で、既にイメージキャッシュは選択された状態となっていますので描画マクロでは、描画コマンドを記述するだけでキャンバスのイメージキャッシュに描画されます。
また、座標系ですが、キャンバス領域の左上が(0,0)となります(画面上の絶対位置ではありません)。
更新描画領域の管理もキャンバスオブジェクト側で行います。描画マクロで行う必要はありません。

使用例

ボタンを押すと円を描画します。

ボタンでキャンバスオブジェクトのリンクメモリを操作します。

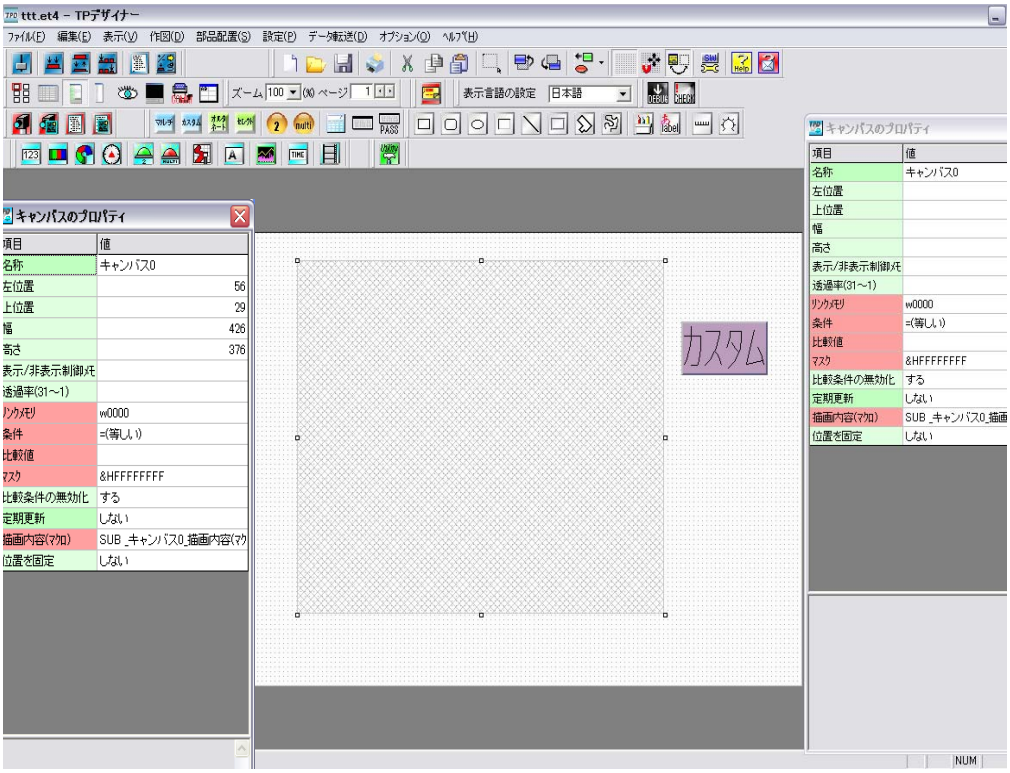


ボタン0(モーメンタリボタン)を押されたときに、キャンバス0に円を書く場合、以下のようなマクロとなります。

キャンバス0のリンクメモリはボタン0のリンクメモリと同じメモリを設定します。

```
;*****************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*****************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
;ここにマクロを記述して下さい。
  ELLI_EX #0 #0 #100 #100 #&0 #&H7fff
END_FUNC
```

また、キャンバスオブジェクト自身の描画マクロではなく、他のボタンから任意のキャンバスオブジェクトに描画したい場合は、キャンバスのイメージキャッシュの選択⇒描画⇒キャンバスのイメージキャッシュの開放といった操作が必要になります。上記の例をキャンバス自身の描画マクロではなく、カスタムボタンのマクロから描画する使用例を以下に示します。



キャンバスオブジェクトではなくカスタムボタンオブジェクトに以下のように記述します。

```
;*****************************
;!ボタン 0_押された時の動作のマクロ記述
;*****************************
FUNC _ボタン 0_押された時の動作
;ここにマクロを記述して下さい。
   SUB _キャンバス 0SELECT
   ELLI_EX #0 #0 #100 #100 #&0 #&H7fff
   SUB _キャンバス 0UNSELECT
END_FUNC
```

_[キャンバスオブジェクト名称]SELECT で指定キャンバスを選択し描画可能状態にします。
_[キャンバスオブジェクト名称]UNSELECT で指定キャンバスを開放します。
[キャンバスオブジェクト名称]SELECT を呼び出したあとは必ず[キャンバスオブジェクト名称]UNSELECT を呼び出して下さい。(画面表示が崩れます。最悪暴走します。)

なお、マクロで描画する場合、キャンバスオブジェクトの大きさを超える描画を行うことは出来ません。キャンバスオブジェクトの大きさを超える描画を行った場合、意図しない領域に書き込みを行い、最悪の場合 GOP が暴走いたします。

・GOPの色コードについて

GOP で使用できる色は RGB 各 32 階調です。

値設定値は各色の階調を 2 進数(5bit)で表現しそれらをつなげて 15bit+1bit(この 1bit は透明色フラグになります)の 16bit を 16 進数表記で表現します。

透明色ビット	赤 0～31 ビットパタン 00000～11111	緑 0～31 ビットパタン 00000～11111	青 0～31 ビットパタン 00000～11111

例 赤 16 緑 16 青 31

2 進表現

0	1	0	0	0	0	1	0	0	0	0	1	1	1	1	1

16 進数表現

4	2	1	F

となります。(&H421F)

また透明色は&H8000 で表します。

・TPD V4 によって定義される値

TPD V4 は、その描画処理のため定数を定義します。

マクロからもそれらの定数を参照・利用することで効率的にマクロを組んでいくことが可能です。

以下に TPD V4 で定義される定数の説明を行います。

定義値	書式	使用例
キャンバス配置左位置	_[キャンバス名称].X	;タッチ位置をイメージキャッシュ上の ;位置に変換 EXPR X=TOUCH_X-_キャンバス 0.X EXPR Y=TOUCH_Y-_キャンバス 0.Y
キャンバス配置上位置	_[キャンバス名称].Y	
キャンバス配置右位置	_[キャンバス名称].X2	
キャンバス配置下位置	_[キャンバス名称].Y2	

キャンバス幅	_[キャンバス名称].W	;領域判定
		IF X>#0 AND X<=_キャンバス 0.W THEN
		;X 位置がイメージキャッシュ内なら描画
		DOT_EX X #10 #0
		ENDIF

キャンバス高さ	_[キャンバス名称].H	
キャンバス X 方向最大	_[キャンバス名称].XMAX	;キャンバス塗りつぶし RECT_EX #0 #0 _キャンバス 0.XMAX _キャンバス 0.YMAX #&H8000 #&H8000 ;キャンバスをクリア
キャンバス Y 方向最大	_[キャンバス名称].YMAX	

また、キャンバスエリア選択のためのサブルーチンも以下が定義されます。

キャンバスエリアを描画可能にする	_[キャンバス名称]SELECT	;該当キャンバス以外のオブジェクトのマクロから SUB _キャンバス 0SELECT LINE_EX #0 #0 #100 #100 #0 SUB _キャンバス 0UNSELECT
キャンバスエリアの描画後処理	_[キャンバス名称]SELECT	

なお、単一グループ化されたキャンバスでは上記ルーチンは作成されないため、外部からの描画は出来ません。

外部描画を行う場合、単一グループ化は行わないで下さい。

(1) DOT_EX　点描画

コマンド書式

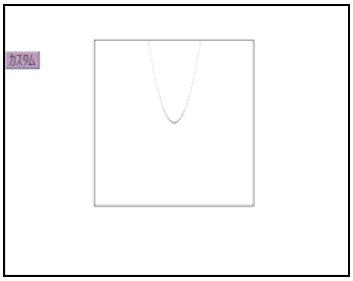
DOT_EX X Y COLOR

動作

(X,Y)に COLOR の点を描画します。

使用例

二次曲線をプロットします。



```
;*****************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*****************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
;作業用のメモリを確保
   ;計算上の X,Y
   ADIM X F
   ADIM Y F
   ;描画中心
   ADIM CX W
   ADIM CY W
   ;実際の描画座標
   ADIM PTX W
   ADIM PTY W
   ;プロットエリアの中心を求める
   EXPR CX=_キャンバス 0.W/#2
   EXPR CY=_キャンバス 0.H/#2
   ;X をプロットエリアの左端から右端に移動したときの Y を求め点を描画
   VALSET PTX #0
   DO
        EXPR X=PTX-CX
        EXPR Y=(X*X)/@16
        EXPR PTY=CY-Y
        IF PTY>=#0 AND PTY<_キャンバス 0.H THEN
             ;プロット点がキャンバス 0 の範囲に収まるならば描画
             DOT_EX PTX PTY #0
        ENDIF
        ADD PTX #1
        IF PTX=_キャンバス 0.W THEN
             EXIT
        ENDIF
   LOOP

   END_FUNC
```

(2) LINE_EX　線描画

コマンド書式

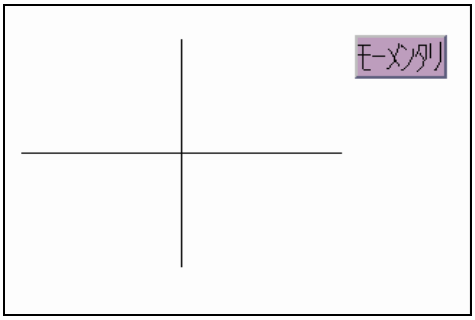
LINE_EX X1 Y1 X2 Y2 COLOR

動作

(X1,Y1)-(X2,Y2)に COLOR の線を描画します。

使用例

X 軸、Y 軸を描画します。




```
;*****************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*****************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
   ;作業用のメモリを確保
   ;描画中心
   ADIM CX W
   ADIM CY W
   ;プロットエリアの中心を求める
   EXPR CX=_キャンバス 0.W/#2
   EXPR CY=_キャンバス 0.H/#2
   LINE_EX #0 CY _キャンバス 0.XMAX CY #0
   LINE_EX CX #0 CX _キャンバス 0.YMAX #0

END_FUNC
```

(3) RECT_EX　矩形描画

コマンド書式

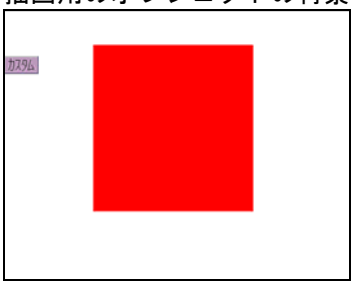
RECT_EX X1 Y1 X2 Y2 FCOLOR BCOLOR

動作

(X1,Y1)-(X2,Y2)の矩形範囲を枠線色(FCOLOR)、塗りつぶし色(BCOLOR)で描画します。

使用例

描画用のオブジェクトの背景を赤で塗りつぶします。




```
;*****************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*****************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
   RECT_EX #0 #0 _キャンバス 0.XMAX _キャンバス 0.YMAX #&H7c00 #&H7c00
END_FUNC
```

(4) RECTG_EX　矩形描画(グラデーション)

コマンド書式

RECTG_EX X1 Y1 X2 Y2 dir COLOR1 COLOR2

動作

(X1,Y1)-(X2,Y2)の矩形範囲をグラデーションで塗りつぶします。

dir　1：上から下
　　2：下から上
　　3：右から左
　　4：左から右
　　5：上下中央線から上下
　　6：左右中央線から左右

※dir はメモリ指定は出来ないため、#なしの数値で指定します。

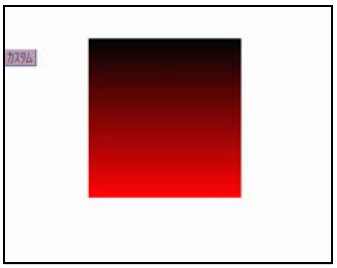
※グラデーションの色は以下の式で計算され、リニアに変化します。

基準点の色=COLOR1

終端点の色= COLOR2

任意の描画点の色=COLOR1+(COLOR1-COLOR2/(基準点～終端点の距離)*描画点の位置)

使用例



```
;******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
   RECTG_EX #0 #0 _キャンバス 0.XMAX _キャンバス 0.YMAX 1 #&H0 #&H7c00
END_FUNC
```

(5) ELLI_EX　楕円描画

コマンド書式

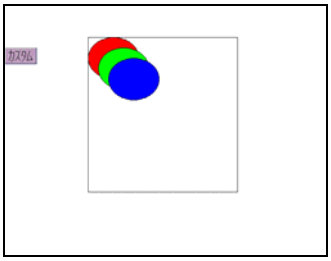
ELLI_EX X1 Y1 X2 Y2 FCOLOR BCOLOR

動作

(X1,Y1)-(X2,Y2)の矩形に内接する楕円領域を枠線色(FCOLOR)、塗りつぶし色(BCOLOR)で描画します。

使用例

下のような描画を行います



```
;******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
   ELLI_EX #0 #0 #100 #80 #0 #&H7C00
   ELLI_EX #20 #20 #120 #100 #0 #&H03e0
   ELLI_EX #40 #40 #140 #120 #0 #&H001F
END_FUNC
```

(6) ELLIG_EX 楕円描画(グラデーション)

コマンド書式

ELLIG_EX X1 Y1 X2 Y2 dir COLOR1 COLOR2

動作

(X1,Y1)-(X2,Y2)の矩形に内接する楕円領域をグラデーションで塗りつぶします。

dir　1：上から下
　　2：下から上
　　3：右から左
　　4：左から右
　　5：上下中央線から上下
　　6：左右中央線から左右

※dir はメモリ指定は出来ないため、#なしの数値で指定します。

※グラデーションの色は以下の式で計算され、リニアに変化します。

基準点の色=COLOR1

終端点の色= COLOR2

任意の描画点の色=COLOR1+(COLOR1-COLOR2/(基準点～終端点の距離)*描画点の位置

使用例

下のような図を表示します。

```
;*******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
  ELLIG_EX #0 #0 #200 #150 5 #&H7C00 #&H7FFF
  ELLIG_EX #40 #50 #240 #200 5 #&H03e0 #&H7FFF
  ELLIG_EX #80 #100 #280 #250 5 #&H001F #&H7FFF
END_FUNC
```

(7) R_RECT_EX　角R付矩形

コマンド書式

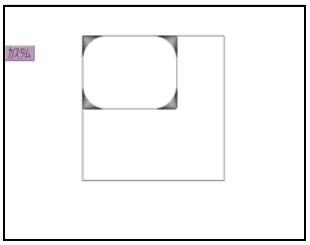
R_RECT_EX X1 Y1 X2 Y2 FCOLOR BCOLOR R

動作

(X1,Y1)-(X2,Y2)の矩形範囲をRで面取りし、枠線色(FCOLOR)、塗りつぶし色(BCOLOR)で描画します。

使用例

下のような図を表示します。



```
;******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
   ;作業用のメモリを確保
   ;角 R 部の半径指定用
   ADIM R W
   FOR R=#5 TO #50 STEP #5
        R_RECT_EX #0 #0 #200 #150 #&H0 #&H7FFF R
   NEXT
END_FUNC
```

(8) R_RECTG_EX 角R付矩形(グラデーション)

コマンド書式

R_RECTG_EX X1 Y1 X2 Y2 dir FCOLOR BCOLOR R

動作

(X1,Y1)-(X2,Y2)の矩形に内接する角 R 矩形領域をグラデーションで塗りつぶします。

dir　1：上から下
　　2：下から上
　　3：右から左
　　4：左から右
　　5：上下中央線から上下
　　6：左右中央線から左右

※dir はメモリ指定は出来ないため、#なしの数値で指定します。

※グラデーションの色は以下の式で計算され、リニアに変化します。

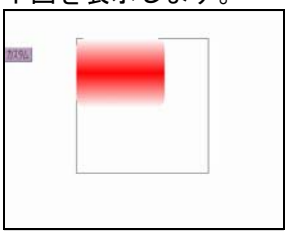
基準点の色=COLOR1

終端点の色= COLOR2

任意の描画点の色=COLOR1+(COLOR1-COLOR2/(基準点～終端点の距離)*描画点の位置

使用例

下図を表示します。



```
;*******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
  R_RECTG_EX #0 #0 #200 #150 5 #&H7c00 #&H7FFF #20
END_FUNC
```

(9) F_RECT_EX　立体枠

コマンド書式

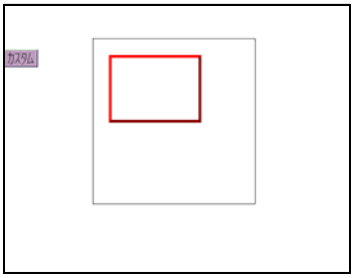
F_RECT_EX X1 Y1 X2 Y2 FCOLOR BCOLOR H

動作

(X1,Y1)-(X2,Y2)の矩形範囲に左上線色：FCOLOR、右下線色：BCOLOR で指定の領域で浮き出し高さ H で立体枠を描画します。(枠だけの描画です。)

使用例

下図を表示します。



```
;*******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
  F_RECT_EX #30 #30 #200 #150 #&H7c00 #&H3C00 #5
END_FUNC
```

(10) TXT_EX　文字

コマンド書式

TXT_EX X Y size pitch FCOLOR BCOLOR SRCTEXT

動作

(X,Y)の位置から文字色 FCOLOR、背景色 BCOLOR で文字サイズは size、文字間隔は pitch で SRCTEXT で指定される文字を描画します。

size

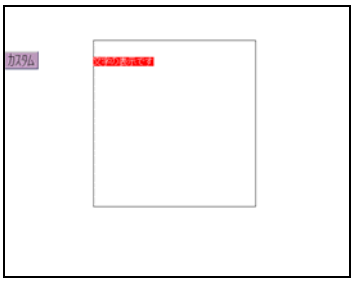
0：標準　1：縦倍角　2：4 倍角　3：9 倍角　4：横 1/2 倍　5：縦 1/2 倍　6：1/4 倍

7：12 ドット標準　8:12 ドット縦倍角　9:12 ドット 4 倍角　10:12 ドット 9 倍角

※size,pitch はメモリ指定できませんので#なしで指定して下さい。

使用例

下図を表示します。



```
;*******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
   TXT_EX #0 #30 0 0 #&H7FFF #&H7C00 $文字の表示です
END_FUNC
```

(11) STXT_EX　文字(ストロークフォント)

コマンド書式

STXT_EX X Y w h pitch borld FCOLOR BCOLOR SRCTEXT

動作

(X,Y)の位置から文字色 FCOLOR、背景色 BCOLOR で文字間隔は pitch で SRCTEXT で指定される文字を描画します。

ストロークフォントは w,h で指定される大きさになります。

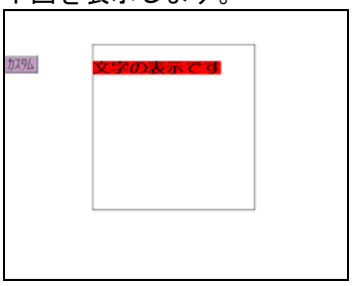
borld

0:細字　1：太字　2：極太字

※pitch､borld､w､h はメモリ指定できませんので#なしで指定して下さい。

使用例

下図を表示します。



```
;*******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
   STXT_EX #0 #30 32 24 0 2 #0 #&H7C00 $文字の表示です
END_FUNC
```

(12) ARC_EX　扇形

コマンド書式

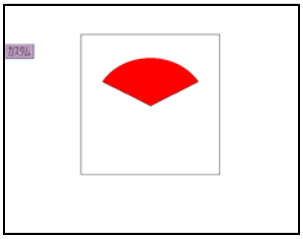
ARC_EX CX CY RX RY START_ANGLE END_ANGLE FCOLOR BCOLOR

動作

前景色：FCOLOR、背景色：BCOLOR、中心点：CX,CY、X 軸半径：RX、Y 軸半径：RY、
開始角：START_ANGLE、終了角：END_ANGLE で扇形を描画します。

使用例

下図を表示します。



```
;*******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
   ARC_EX #150 #150 #120 #100 #30 #150 #0 #&H7c00
END_FUNC
```

(13) BMP_FONT_EX　BMPフォント描画

コマンド書式

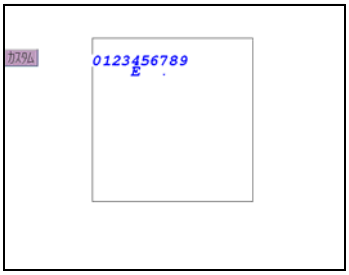
BMP_FONT_EX X Y size pitch FCOLOR BCOLOR fontno SRCTEXT

動作

X,Y の位置から SRC の文字列を fontno の BMP フォントで SRCTEXT を文字色：FCOLOR、背景色：BCOLOR で描画します。
SRCTEXT に指定される文字列の中に 0～9,E,-以外の文字がある場合はスペースとなります。
size の指定は以下のとおりです。
　0：標準 1：縦倍角 2：4 倍角 3：9 倍角
fontno は TPD V4 のビットマップフォント登録ダイアログで登録したフォントセット番号を指定します。但し、TPD V4 のビットマップフォント登録ではフォントセット番号 1～3 となっていますが、fontno は値範囲は 0～2 です。表示させたいフォントセットの番号-1 を設定して下さい。
※size,pitch,fontno はメモリ指定できないため,#なしで指定して下さい。

使用例

以下を表示します。
※TPD V4 のビットマップフォント登録ダイアログでフォントセット 1 に"1822BookV3.ibf"を登録しておいて下さい。
ビットマップフォントにない文字は表示されません。




```
;*******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
   BMP_FONT_EX #0 #30 0 0 #&H001F #&H7FFF 0 $0123456789
   BMP_FONT_EX #0 #50 0 0 #&H001F #&H7FFF 0 $ABCDEFG.
END_FUNC
```

(14) XDOT_EX　XOR点描画

コマンド書式

XDOT_EX X Y COLOR

動作

(X,Y)に COLOR の点を XOR オペレーションで描画します。XOR オペレーションの場合同じ位置に同じ色で 2 回描画すると元の表示に戻ります。

(15) XLINE_EX　XOR線描画

コマンド書式

XLINE_EX X1 Y1 X2 Y2 COLOR

動作

(X1,Y1)-(X2,Y2)に COLOR の線を XOR オペレーションで描画します。XOR オペレーションの場合同じ位置に同じ色で 2 回描画すると元の表示に戻ります。

(16) XRECT_EX XOR矩形描画

コマンド書式

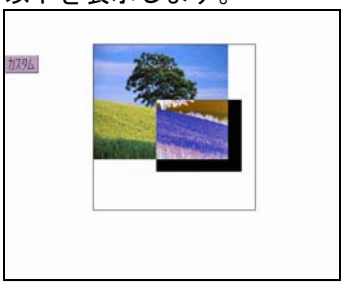
XRECT_EX X1 Y1 X2 Y2 FCOLOR BCOLOR

動作

前景色：FCOLOR 背景色：BCOLOR で指定の領域で矩形を XOR オペレーションで描画します。XOR オペレーションの場合同じ位置に同じ色で 2 回描画すると元の表示に戻ります。

使用例

以下を表示します。



```
;*****************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*****************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
  ;背景を描画します。
  STRETCHBMPT_EX #0 #0 #250 #200 #&H8000 #0
  ;指定矩形をネガポジ反転します。
  XRECT_EX #100 #100 #280 #220 #&H7FFF #&H7FFF
END_FUNC
```

(17) STRETCHBMPT　透過対応リサイズブルビットマップ描画

コマンド書式

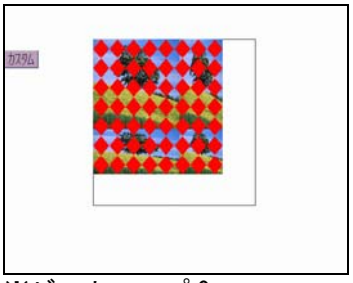
STRETCHBMPT_EX X Y W H C NO

動作

X,Y の位置から NO で指定されたビットマップを W,H の大きさに拡大・縮小して描画します。W,H に 0 が指定された場合、ビットマップの原寸で描画します。C で指定された色番は透明色となります。透明色を指定しない場合 C に&h8000 を指定します。

使用例

以下を表示します。



※ビットマップ 0

※ビットマップ 1

```
;******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
   ;木のビットマップを拡縮して描画
   STRETCHBMPT_EX #0 #0 #160 #160 #&H8000 #0
   STRETCHBMPT_EX #160 #0 #80 #160 #&H8000 #0
   STRETCHBMPT_EX #0 #160 #160 #80 #&H8000 #0
   STRETCHBMPT_EX #160 #160 #80 #80 #&H8000 #0
   ;チェッカーのビットマップを白を透明色として描画
   STRETCHBMPT_EX #0 #0 #240 #240 #&H7FFF #1
END_FUNC
```

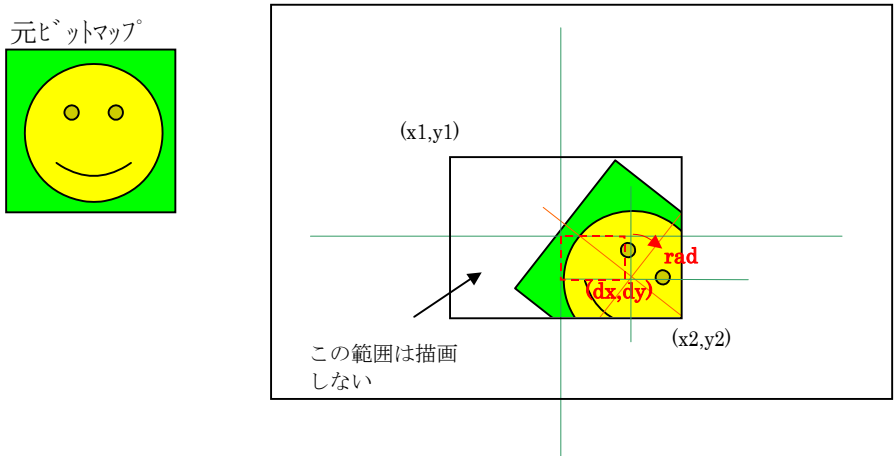
(18) ROTBMP_EX　回転対応ビットマップ

コマンド書式

ROTBMPT_EX X1 Y1 X2 Y2 DX DY RAD C NO

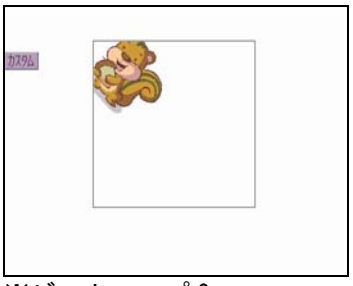
動作

(X1,Y1)-(X2,Y2)の矩形範囲内にNOで指定されたビットマップをRAD度回転させDX,DYオフセットさせて描画します。Cで指定された色番は透明色となります。透明色を指定しない場合Cに&h8000を指定します。



使用例

以下を表示します。



※ビットマップ0

```
;*******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
   ;リスのビットマップを回転して描画
   ROTBMPT_EX #0 #0 #130 #130 #0 #0 #45 #&H8000 #0
END_FUNC
```

(19) BOXTXT2_EX　矩形領域内に文字描画

コマンド書式

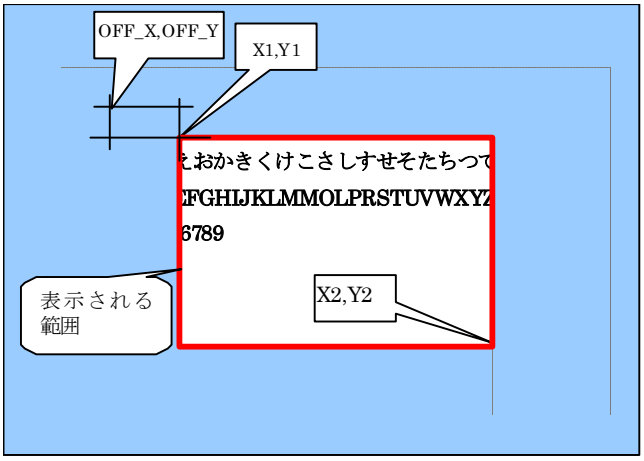
BOXTXT2_EX X1 Y1 X2 Y2 OFF_X OFF_Y size pitch hali FCOLOR BCOLOR SRCTEXT

動作

(X1,Y1)-(X2,Y2)の矩形内に収まるよう、SRCTEXT で指定される文字列をトリミングして描画します。文字列は(OFF_X,OFF_Y)にて開始座標を設定できるためスクロール等、途中状態からを描画することが可能です。

矩形の内部は BCOLOR、文字は FCOLOR で描画します。

SRCTEXT で指定される文字列中に¥n があった場合、改行処理を行います。行高さは、文字の高さ分です。



size

0：標準　1：縦倍角　2：4 倍角　3：9 倍角　4：横 1/2 倍　5：縦 1/2 倍　6：1/4 倍

7：12 ドット標準　8:12 ドット縦倍角　9:12 ドット 4 倍角　10:12 ドット 9 倍角

pitch

文字間隔

hali

0:左寄せ 1:右寄せ 2:中央寄せ

※size,pitch,hali はメモリ指定できないため,#なしで指定します。

使用例

```
;*******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
  ;文字列は実際は改行して入力されていません。
  BOXTXT2_EX #10 #10 #220 #70 #-24 #-8 0 0 0 #&H7c00 #0 $SRCTEXT で指定される文字列¥n あいうえおかきくけこさしすせそたちつてと¥nABCDEFGHIJKLMMOLPRSTUVWXYZ¥n0123456789¥n あ¥n い¥n う¥n え¥n お
END_FUNC
```

(20) SBOXTXT2_EX　矩形領域内に文字描画(ストロークフォント)

コマンド書式

SBOXTXT2_EX X1 Y1 X2 Y2 OFF_X OFF_Y w h pitch borld hali FCOLOR BCOLOR SRCTEXT

動作

BOXTXT2_EX のストロークフォントバージョンです。

w h

　ストロークフォント幅、高さ

pitch

　文字間隔

borld

　文字太さを指定します。0(標準)、1(太字)、2(極太)のいずれかを指定します。

hali

　0:左寄せ 1:右寄せ 2:中央寄せ

※w、h、pitch、borld、hali はメモリ指定できないため,#なしで指定します。

(21) BOXINTXT_EX　矩形内縮小文字列描画

コマンド書式

BOXINTXT_EX X1 Y1 X2 Y2 orgsize orgpith hali vali　FCOLOR BCOLOR SRCTEXT

動作

SRCTEXT で指定される文字列を (X1,Y1)-(X2,Y2)矩形内に収まるように縮小し表示します。矩形の内部は BCOLOR、文字は FCOLOR で描画します。

※縮小しなくても矩形範囲に収まる場合は等倍で描画します。

SRCTEXT で指定される文字列中に¥n があった場合、改行処理を行います。行高さは、文字の高さ分です。

orgsize

　0：標準　1：縦倍角　2：4 倍角　3：9 倍角　4：横 1/2 倍　5：縦 1/2 倍　6：1/4 倍

　7：12 ドット標準　8:12 ドット縦倍角　9:12 ドット 4 倍角　10:12 ドット 9 倍角

orgpitch

　文字間隔

hali

　0:左寄せ 1:右寄せ 2:中央寄せ

vali

　0:上寄せ 1:下寄せ 2:中央寄せ

※orgsize、origpitch、hali、vali はメモリ指定できないため、#なしで指定して下さい。

使用例


```
;*******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
  BOXINTXT_EX #0 #0 #250 #70 0 0 2 2 #&H6f7F #0 $矩形内縮小描画の¥n サンプルです。¥n 指定の大きさに収まるよう¥n 縮小されます
  BOXINTXT_EX #0 #80 #200 #130 0 0 2 2 #&H6f7F #0 $矩形内縮小描画の¥n サンプルです。¥n 指定の大きさに収まるよう¥n 縮小されます
  BOXINTXT_EX #0 #140 #150 #170 0 0 2 2 #&H6f7F #0 $矩形内縮小描画の¥n サンプルです。¥n 指定の大きさに収まるよう¥n 縮小されます
END_FUNC
```

(22) SBOXINTXT_EX矩形内縮小文字列描画(ストロークフォント)

コマンド書式

SBOXINTXT_EX X1 Y1 X2 Y2 orgw orgh orgpith borld hali vali FCOLOR BCOLOR SRCTEXT

動作

BOXINTXT_EX のスクロールフォントバージョンです。

orgw orgh

　ストロークフォント幅、高さ

orgpitch

　文字間隔

borld

　文字太さを指定します。0(標準)、1(太字)、2(極太)のいずれかを指定します。

hali

　0:左寄せ 1:右寄せ 2:中央寄せ

vali

　0:上寄せ 1:下寄せ 2:中央寄せ

※orgw、orgh、origpitch、borld、hali、vali はメモリ指定できないため#なしで指定して下さい。

(23) SELECT_IMAGECACHE　イメージキャッシュの選択

コマンド書式

SELECT_IMAGECACHE id x y

動作

既存の画像キャッシュを書き込み対象として選択します。

※GOP シリーズでマクロにより描画を行う場合、内部にイメージキャッシュというバッファ領域を確保しそれに対して描画するようになります。イメージキャッシュはボックス等のオブジェクトの領域を使用します。

id

選択するイメージキャッシュの ID を指定します。

イメージキャッシュは TPD V4 で定義され、以下のような様式になります。

_[オブジェクト名称]_ID

オブジェクト名称が"ボックス 0"の場合"_ボックス 0_ID"となります。

※id は数値ですが、上記フォーマットの ID を TPD V4 が番号に置き換えますので上記以外のデータを指定しないで下さい。

x,y

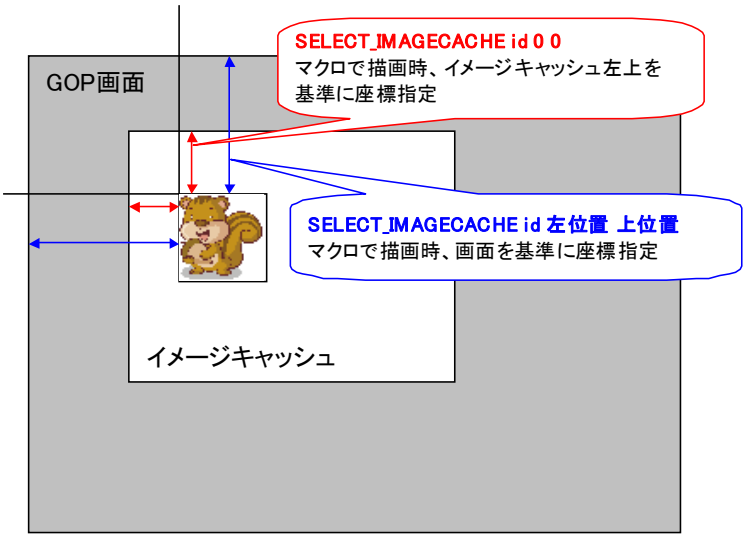
イメージキャッシュにマクロ描画コマンドで描画する際に、イメージキャッシュの左上座標のオフセット値を指定します。

通常ここには,(0,0)またはオブジェクトの配置左上座標を指定します。

(0,0)を指定時は、後のマクロコマンドの描画でイメージキャッシュ上の位置で記述することが可能です。

オブジェクト配置位置を指定した場合、GOP の画面上の位置を基準に座標を記述します。

※x、y は数値のみですので#なしで指定して下さい。



※本コマンドはキャンバスオブジェクトを使用する場合不要です。

他のオブジェクトから任意のキャンバスのイメージキャッシュを選択する場合

SUB _[キャンバスオブジェクト名称]SELECT

のモジュールコールで取得して下さい。

(24) INIT_IMAGECACHE　イメージキャッシュのクリア

コマンド書式

INIT_IMAGECACHE

動作

選択されているイメージキャッシュをクリアします。

クリア後は透明になります。

(25) UNSELECT_IMAGECACHE　画像キャッシュの選択解除

コマンド書式

UNSELECT_IMAGECACHE

動作

画像キャッシュの選択を解除します。
マクロで描画後は必ず選択解除して下さい。
※本コマンドはキャンバスオブジェクトを使用する場合不要です。
　SUB _[キャンバスオブジェクト名称]SELECT
で取得したイメージキャッシュは
　SUB _[キャンバスオブジェクト名称]UNSELECT
で開放して下さい。

(26) LOAD_IMAGE　JPEG及びBMP形式の画像データの表示

コマンド書式

LOAD_IMAGE X Y W H FILENAME

動作

X,Y で指定した箇所に W,H で指定した大きさで、FILENAME で指定した JPEG ﾌｧｲﾙまたは BMP ファイルの画像を描画します。

X,Y
　描画する場所を数値または数値型メモリで指定します。

W,H
　描画サイズを数値または数値型メモリで指定します。
　元サイズが W,H より大きい場合は縮小し描画します。
　元サイズが W,H より小さい場合は拡大し描画します。
　なお拡大･縮小を伴う場合補間処理は行わないため、画像品位は低下します。

FILENAME
　ロードする画像ファイル名を指定します。テキストまたはテキストメモリを指定して下さい。

※GOP-5000 シリーズのみの機能としてハードウェア JPEG デコードユニットが搭載されており以下の形式の JPEG ファイルを高速にデコードすることができます。
　ベースライン YCrCb422 または YCrCb420
　画像サイズ　　幅:16 で割り切れる
　　高さ:YCrCb422 の場合 8 で割り切れるサイズ
　　YCrCb420 の場合 16 で割り切れるサイズ
　上記に合致しない場合ソフトウェアデコーダーを使用します。

使用例

USB メモリにある、P0001.JPG を描画します。

P0001.JPG

実行イメージ

```
;*****************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*****************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
  LOAD_IMAGE #0 #0 #280 #185 $2:P0001.JPG
END_FUNC
```

(27) LOAD_BMP　GOP Bitmap(GB)形式ファイルの描画

コマンド書式

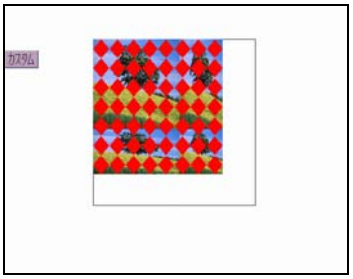
LOAD_BMP X Y W H C BMPFILE_NAME

動作

X,Y の位置から BMPFILE_NAME で指定されたビットマップを W,H の大きさに拡大・縮小して描画します。W,H に 0 が指定された場合、ビットマップの原寸で描画します。C で指定された色番は透明色となります。透明色を指定しない場合 C に&h8000 を指定します。

※ビットマップの指定方法が番号からファイル名に変更されているだけで他の使い方は STRETCHBMPT_EX コマンドと同様です。

使用例



※BMP1.GB

※BMP2.GB

```
;******************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;******************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
  ;木のビットマップを拡縮して描画
  LOAD_BMP #0 #0 #160 #160 #&H8000 $2:BMP1.GB
  LOAD_BMP #160 #0 #80 #160 #&H8000 $2:BMP1.GB
  LOAD_BMP #0 #160 #160 #80 #&H8000 $2:BMP1.GB
  LOAD_BMP #160 #160 #80 #80 #&H8000 $2:BMP1.GB
  ;チェッカーのビットマップを白を透明色として描画
  LOAD_BMP #0 #0 #240 #240 #&H7FFF $2:BMP2.GB
END_FUNC
```

4. 2. 17 HTTP クライアント機能

※本機能は GOP-H クラスのみの機能です。

GOP にオプションのイーサネット拡張ユニットが組み込まれている場合、任意の Web サーバーからデータを取得することが出来ます。

(1) HTTP_GET Webサーバーからのファイル取得

コマンド書式

HTTP_GET STATUS URL DEST_FILENAME AUTH

動作

HTTP プロトコルの GET メソッドを指定のサーバーに対し送信し、受け取ったデータをファイルとして DEST_FILENAME で指定される場所・名称で保存します。

本コマンドは非ブロッキングモード(サーバーからの応答をまたずコマンドを終了する)で動作します。その為、本コマンドが終了した場合でも DEST_FILENAME は作成・更新されていません。

Web サーバーから応答が完了した場合,STATUS で指定されるメモリに 200(HTTP プロトコルで正常終了の意)がセットされます。

AUTH は現時点未実装のため$のみを指定して下さい。

STATUS	GET メソッドの応答結果を格納するメモリ
URL	サーバー名&GET メソッドのクエリーストリングス
DEST_FILENAME	Web サーバーから送られてくるデータを保存するファイル名を指定
AUTH	認証用文字列(現時点機能未実装)

使用例

Web サーバー(http://testpc)から画像(/pic/gazo1.jpeg)を取得し、RAM ドライブ(0:)に PIC.JPG の名称で保存します。

なお動作環境として GOP からアクセス可能な場所にサーバーが用意され、データとしてサーバーの web 公開フォルダの中の pic のフォルダ中に gazo1.jpeg が存在していることとします。

```
HTTP_GET w0000 $testpc/pic/gazo1.jpeg $0:PIC.JPG $
```

4. 2. 18 その他

(1) CONVERT_BMP　JPEG及びBMP形式の画像データをGOP Bitmap(GB)形式に変換

コマンド書式

CONVERT_BMP W H DEST_NAME SRC_NAME

動作

JPEG 画像は画像の表示に時間がかるため、表示速度が必要な場合 JPEG データを GB 形式に変換しておくことで表示速度を上げることが出来ます。

W,H

変換後の GOP bitmap のサイズを指定します。
#0 #0 を指定すると原寸のまま変換します。

DEST_NAME

変換されたデータが保存される GOP Bitmap のファイル名を指定

SRC_NAME

変換元となる画像ファイル名を指定

使用例

```
CONVERT_BMP #128 #64 $0:BMP.GB $2:PIC001.JPG
```

(2) ALIAS_NAME　任意のリソース番号(WAVやビットマップ指定用)の割付を変更

コマンド書式

ALIAS_NAME op NO BASE_NAME

動作

既存のリソース(ビットマップ、WAV、イメージフォント)の番号による呼び出しを、BASE_NAME に指定したファイル名置き換えます。
本コマンドを使用することにより、リソースを番号で指定する以下のコマンドの動作に影響を与えます。
ビットマップ描画系コマンド

STRETCHBMPT
STRETCHBMPT_EX
ROTBMPT_EX

WAV 再生コマンド

PLAYSOUND

また、上記のコマンドを使用している各種オブジェクトにも影響します。

※TPD V4 で番号でビットマップ、サウンドを指定している箇所全てに影響します。
リンクで設定したファイルが存在しなくなった場合、本コマンドで設定した内容はクリアされ元のリソースが呼び出されます。

使用例

アニメーションでビットマップの 0～9 を使用しているとします。
ファイル転送等で RAMDISK 上に JPG0 から JPG9 をダウンロードした後、CONVERT_BMP で JPG0.GB-JPG9.GB に変換してあるものとします。
この状態で以下のようなマクロを実行します。
コマンド実行後からアニメーションの絵柄が置き換わります。

```
ALIAS_NAME 0 #0 $0:JPG0.GB
ALIAS_NAME 0 #1 $0:JPG1.GB
   :
ALIAS_NAME 0 #9 $0:JPG9.GB
```

(3) SCREEN_CAPT　画面イメージをBMP形式で保存

コマンド書式

SCREEN_CAPT X Y W H FNAME

動作

現在、GOP で表示している画面を BMP 形式で保存します。

X,Y

保存する範囲の左上座標位置。

W,H

保存する範囲の幅と高さ。

DEST_NAME

データ保存するファイル名。

使用例

```
SCREEN_CAPT #0 #0 #640 #480 $2:SCREEN.BMP
```

5. マクロ使用例

5. 1 文字列処理

(1)文字の仕組み

GOP の文字の取り扱いは、内部的には文字列自体を扱うのではなく、文字コードが格納されたｂ型メモリが連続して配置されているものとして扱います。

例：T0100 に"ABCDEF"と格納する場合
メモリ上には

b0100	&h41（A の文字コード）
b0101	&h42（B の文字コード）
b0102	&h43（C の文字コード）
b0103	&h44（D の文字コード）
b0104	&h45（E の文字コード）
b0105	&h46（F の文字コード）
b0106	0 ：文字列の終わりを示す終端文字

と格納されます。
末尾の 0 は文字列の終わりを示す文字コードです。このｺｰﾄﾞがあると以降のメモリにデータがあったとしても、文字列としては扱われません。

例
以下の場合 b0103 に 0 があるため文字列としては T0100=ABC となります。

b0100	&h41（A の文字コード）
b0101	&h42（B の文字コード）
b0102	&h43（C の文字コード）
b0103	0 ：文字列の終わりを示す終端文字
b0104	&h45（E の文字コード）
b0105	&h46（F の文字コード）
b0106	0 ：文字列の終わりを示す終端文字

文字列の先頭のアドレスがわかっている場合、それぞれの位置の文字をｂ型で直接指定することで文字列を変更することが出来ます。

例：4 文字目を H に置き換える場合。

```
MOV b0103 #&H48（H の文字コード）
```

とすることでメモリの内容が以下のようになります。

b0100	&h41（A の文字コード）
b0101	&h42（B の文字コード）
b0102	&h43（C の文字コード）
b0103	&H48（H の文字コード）
b0104	&h45（E の文字コード）
b0105	&h46（F の文字コード）
b0106	0 ：文字列の終わりを示す終端文字

ただしこの方法でメモリを書き換えた場合、テキストボックス等の表示変化は発生しません。(オブジェクトの描画更新はリンクメモリの値書き込みがトリガとなるため。)
描画更新を発生させたい場合、自身の値を再書き込みします。

```
CP 10 T0100 T0100
```

(これにより内容は変化していませんが、書き込むという動作が発生するため描画更新トリガとなります。)
テキストメモリの先頭のアドレスがわからない場合、間接指定で取り出すことが出来ます。

例

```
ADIM testTEXT T   ←このメモリを操作したい
ADIM addr w  ←間接指定のアドレス格納用メモリ
MOV addr & testTEXT    ←メモリ名の先頭に&を付けるとアドレスを取得できます
CP 10 testTEXT $ABCDEF
MOV b(addr+3) #&H48    ←これで testTEXT=ABCHEF となります
CP 10 testTEXT testTEXT ←描画更新必要な場合はこれを実行
```

(2) 文字列の組み立て

文字列の仕組みとしては(1)のような構造となっていますが、文字列を書式化された文字列を作成するといった用途には、文字列操作コマンドを使用して簡単に作成することが出来ます。
書式化された文字列とは、以下のように文字列の中にメモリの値を含んだ文字列を言います。
"身長:[w0000 の値]cm,体重:[w0002 の値]kg"
例えば w0000 が 172 w0002 が 70 の場合、以下のようになります。
"身長:172cm,体重:72kg"

マクロを使用し書式化された文字列を作成するには以下の手順で行います。

1. 文字列の可変部分(数値メモリの値)を FMT コマンドを使用し作業用のテキストメモリに作成する。
2. 最終結果格納先のメモリに固定部分("身長:"等)や(1)で作成した可変部分の文字列を APD コマンドを使用して、順番に追加していく。

マクロでの記述例は以下のとおりです。
完成された文字列は T0100 に格納されるものとします。
また作業用のテキストメモリを T0120 とします。

例

```
;例としての値をセット
MOV w0000 #172
MOV w0002 #70
;文字列組み立てのコード
CP 24 T0100 $          ←格納先のテキストメモリを初期化
APD 24 T0100 $身長:
FMT 3 0 2 T0120 w0000  ←w0000 の値を 3 桁 0ｻﾌﾟﾚｽで文字列化します。
APD 24 T0100 T0120
APD 24 T0100 $cm,体重:
FMT 3 0 2 T0120 w0002  ←w0002 の値を 3 桁 0ｻﾌﾟﾚｽで文字列化します。
APD 24 T0100 T0120
APD 24 T0100 $kg
;結果を通信で出力
OUTPUT T0100
```

(3) 文字列の解析処理

文字列の解析を行う場合は、(1)文字列の仕組みで説明したように、文字列を文字コードが格納された b 型の配列として考え、1 バイトずつ文字コードを確認して処理するようにマクロを組む必要があります。
例として 16 進数状の文字列を整数値に変換および整数値を 16 進数状文字列に変換するマクロを以下に示します。

例：16 進数状文字列⇒10 進数値サブルーチン

```
;SRC はテキスト型で 16 進数状の文字列が格納されているものとします。
;また文字列長は変換対象の整数長に応じた長さで 0ｻﾌﾟありで記述されているとします。
;DEST は整数型(b,B,w,W,l,L)のうちいずれか
;SIZE は DEST の型に応じた値が入っていること(b,B=1,w,W=2,l,L=4)
FUNC HEXTONUM
;変換元データ格納アドレス
ADIM srcAddr w
;変化後データ格納アドレス
ADIM destAddr w
```

```
ADIM dataSize w
ADIM loopct b
  ; GOP の内部データがリトルエンディアンのため
  ;末尾からつめていく必要があるためデータサイズ(w,W の場合 1、I,L,場合 4)-1 分進める
  EXPR destAddr= dataSize - #1
  FOR loopct=#1 TO dataSize
       ;上位 4bit を変換
       IF b(srcAddr)>=#&H30 AND b(srcAddr)<=#&H39 THEN
            EXPR b(destAddr)= b(srcAddr)-#&H30
       ELSE
            ;数字、英大文字以外は来ない前提での決めうちです。
            ;そうでない場合は上と同じ用に範囲判定しｴﾗｰチェックが必要です
            EXPR b(destAddr)= (b(srcAddr)-#&H41)+#10
       ENDIF
       ADD srcAddr #1
       MUL b(destAddr) #&H10
       ;下位 4bit を変換
       IF b(srcAddr)>=#&H30 AND b(srcAddr)<=#&H39 THEN
            EXPR b(destAddr)= b(destAddr)+(b(srcAddr)-#&H30)
       ELSE
            EXPR b(destAddr)= b(destAddr)+((b(srcAddr)-#&H41)+#10)
       ENDIF
       ADD srcAddr #1
       ;1ﾊﾞｲﾄ進める
       ; GOP の内部データがリトルエンディアンのため
       ;加算でなく減算し書き込みアドレスを前に進める
       DEL destAddr #1
  LOOP
END_FUNC
```

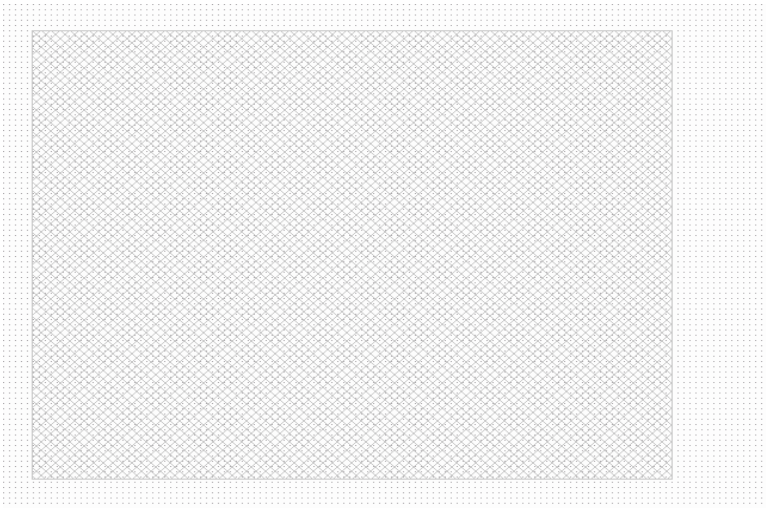
上記サブルーチン使用例
変換文字列格納メモリ T0000=01FF
変換結果格納メモリ　w0100
のとき以下のように使用します。

```
VALSET destAddr &w0100
VALSET srcAddr &T0000
VALSET dataSize #2
SUB HEXTONUM
OUTPUTH w0100‥‥Aw0100=H01FF と出力されます。
```

5. 2 マクロでの描画

(1)タッチのストロークを描画

タッチパネルの押下座標の変化を検出し、その軌跡を描画する画面を作成します。
タッチパネルの押下座標はシステムメモリの TOUCH_X,TOUCH_Y で取得できます。
タッチ座標の変化を検出するため、キャンバスオブジェクトのリンクメモリに TOUCH_X を指定します。
※TOUCH_X,Y は同時に変化するため、どちらかひとつの変化を検出します。
また、比較条件は無効化しておきます。



キャンバスの描画マクロは以下のようになります。

```
;*****************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*****************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
;ここにマクロを記述して下さい。
;軌跡描画では、現座標とその時点での直前の座標を結ぶ線を描画していきます。
;直前座標を記憶しておくためのメモリを取得します。
ADIM OLDX W
ADIM OLDY W
ADIM X W
ADIM Y W

  ;押下位置がキャンバス 0 の範囲内か確認
  IF TOUCH_X>=_キャンバス 0.X AND TOUCH_X<=_キャンバス 0.X2 AND TOUCH_Y>=_キャンバス 0.Y AND TOUCH_Y<=_
キャンバス 0.Y2 THEN
      ;押下座標をキャンバス内の座標に変換
      EXPR X=TOUCH_X-_キャンバス 0.X
      EXPR Y=TOUCH_Y-_キャンバス 0.Y
      ;範囲内のとき、直前の座標が開放(-1,-1)でない場合、直前の座標と現座標を結ぶ直線を描画
      IF OLDX!=#-1 AND TOUCH_X!=#-1 THEN
          LINE_EX X Y OLDX OLDY #0
      ENDIF
      ;直前の座標として現座標をセット
      VALSET OLDX X
      VALSET OLDY Y
  ELSEIF TOUCH_X=#-1 THEN
      ;開放状態のときも直前の座標として現座標をセット
      VALSET OLDX #-1
      VALSET OLDY #-1
  ENDIF
END_FUNC
```

また、直前値の初期化のため、ページ表示時実行マクロに以下を記述します。

```
;*****************************
;!Page1_ﾍﾟｰｼﾞ表示時実行ﾏｸﾛのマクロ記述
;*****************************
FUNC _Page1_ﾍﾟｰｼﾞ表示時実行ﾏｸﾛ
;直前値を開放状態に初期化します。
   VALSET OLDX #-1
   VALSET OLDY #-1
END_FUNC
```

動作イメージ

タッチで押下した軌跡を描画します。

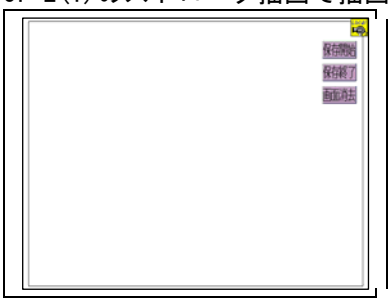
※タッチパネル感度の設定で初期設定では誤認識を防止するため追従性を抑えた設定としているため、この設定で上記のマクロを動作すると速く動かすと追従できません。追従性を向上するには wF262(TP_PERMIT_ALLAERA)の値を 50 程度に設定してください。なお wF262 は GOP-4000 シリーズでは機能仕様として公開はしておりませんが隠しメモリとして機能しています。

5. 3 ファイル処理

マクロを使用して、任意形式のデータファイルの作成や、読み込みを行うサンプルの説明です。

(1)ストローク軌跡のログ出力

5. 2(1)のストローク描画で描画した点座標をファイルに出力するサンプルを作成します。



上図のように、保存開始･保存終了・画面消去のボタンを追加します。
保存開始を押してストローク描画すると、その座標情報が X,Y[CRLF]の形式でファイルにテキスト形式で出力されます。
なお、最終的には USB 出力を行いますが、USB 出力は書き込み速度が遅いため軌跡描画中は SDRAM ディスクに書き込みを行い、保存終了で USB にコピーする仕組みとします。
保存開始のボタンではファイルのオープン処理を行います。
メモリ FNO は正常に開ければ 0 以外の値が入りますので以降の動作で、FNO をファイルナンバー以外にオープン済みのフラグとしても使用します。

```
;*****************************
;!ボタン保存開始_押された時の動作のマクロ記述
;*****************************
;ファイル NO 保存メモリ
ADIM FNO W
FUNC _ボタン保存開始_押された時の動作
   VALSET FNO #0
   ;軌跡データ保存用のファイル(PLOT.CSV)を書き込みモードで開きます
   OPENFILE FNO 1 $0:/PLOT.CSV
   ;開放状態に初期化します。
   VALSET OLDX #-1
   VALSET OLDY #-1
END_FUNC
```

また,キャンバスオブジェクトの描画マクロは以下のように修正し、ファイルが開かれていれば座標をファイルに出力するようにします。

```
;*****************************
;!キャンバス 0_描画内容(ﾏｸﾛ)のマクロ記述
;*****************************
FUNC _キャンバス 0_描画内容(ﾏｸﾛ)
;ここにマクロを記述して下さい。
;軌跡描画では、現座標とその時点での直前の座標を結ぶ線を描画していきます。
;直前座標を記憶しておくためのメモリを取得します。
ADIM OLDX W
ADIM OLDY W
ADIM X W
ADIM Y W

   ;押下位置がキャンバス 0 の範囲内か確認
   IF TOUCH_X>=_キャンバス 0.X AND TOUCH_X<=_キャンバス 0.X2 AND TOUCH_Y>=_キャンバス 0.Y AND TOUCH_Y<=_
キャンバス 0.Y2 THEN
         ;押下座標をキャンバス内の座標に変換
         EXPR X=TOUCH_X-_キャンバス 0.X
         EXPR Y=TOUCH_Y-_キャンバス 0.Y
```

```
        ;範囲内のとき、直前の座標が開放(-1,-1)でない場合、直前の座標と現座標を結ぶ直線を描画
        IF OLDX!=#-1 AND TOUCH_X!=#-1 THEN
            LINE_EX X Y OLDX OLDY #0
        ENDIF
        ;直前の座標として現座標をセット
        VALSET OLDX X
        VALSET OLDY Y
    ;ファイルが開かれているときその座標をファイルに書き出します。
        ;(2).1のマクロから追記
        IF FNO!=#0 THEN
            ;文字列変換用の作業メモリを用意
            ADIM temp T 10
            ;書き込み用の作業メモリを用意
            ADIM wbuf T 20
            ;X,Yの文字列に変換
            FMT 4 0 0 temp X
            CP 20 wbuf temp
            APD 20 wbuf $,
            FMT 4 0 0 temp Y
            APD 20 wbuf temp
            WRITELINE FNO wbuf
        ENDIF

    ELSEIF TOUCH_X=#-1 THEN
        ;開放状態のときも直前の座標として現座標をセット
        VALSET OLDX #-1
        VALSET OLDY #-1
        ;ファイルが開かれているとき開放を示す空行を出力します。
        ;(2).1のマクロから追記
        IF FNO!=#0 THEN
            ;開放時は区切りのため空行を入れます
            WRITELINE FNO $
        ENDIF
    ENDIF
END_FUNC
```

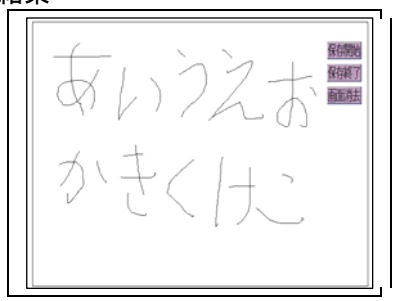
保存終了のボタンでは、ファイルのクローズ処理とUSBメモリへの転送処理を行います。

```
;*****************************
;!ボタン保存終了_押された時の動作のマクロ記述
;*****************************
FUNC _ボタン保存終了_押された時の動作
    ;ファイルを閉じます
    CLOSEFILE FNO
    ;メモリに0をセットします。
    VALSET FNO #0
    ;PLOT.CSVをUSBメモリにコピーします。
    FILEOPERATION 0 $2:/PLOT.CSV $0:/PLOT.CSV
END_FUNC
```

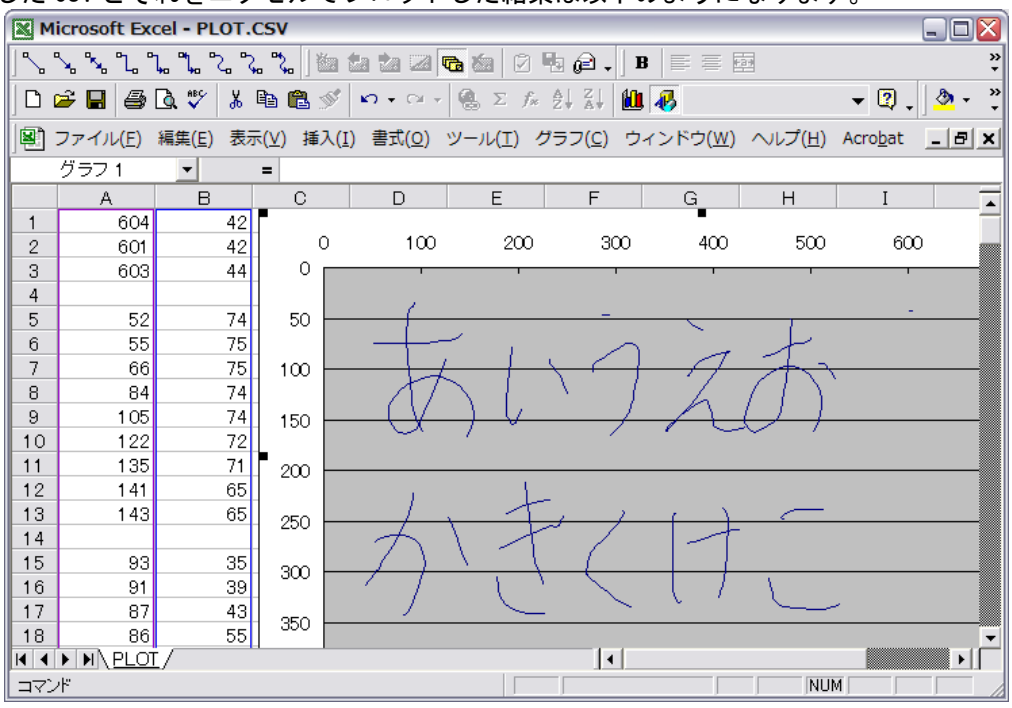
画面消去ボタンで、書かれた軌跡をクリアします。
消去ボタンのマクロでキャンバス0のイメージキャッシュを取得し、透明色で描画後イメージキャッシュを開放します。

```
;*****************************
;!ボタン画面消去_押された時の動作のマクロ記述
;*****************************
FUNC _ボタン画面消去_押された時の動作
    ;消去動作用のフラグ
    SUB _キャンバス0SELECT
    RECT_EX #0 #0 _キャンバス0.XMAX _キャンバス0.YMAX #&H8000 #&H8000
    ;フラグセットして、キャンバス描画を呼び出す
    SUB _キャンバス0UNSELECT
END_FUNC
```

動作結果



出力した CSV とそれをエクセルでプロットした結果は以下のようになります。



	A	B
1	604	42
2	601	42
3	603	44
4		
5	52	74
6	55	75
7	66	75
8	84	74
9	105	74
10	122	72
11	135	71
12	141	65
13	143	65
14		
15	93	35
16	91	39
17	87	43
18	86	55

(2)ストローク軌跡の読み込みと表示

(1)で作成した CSV ファイルを読み込み、描画するマクロを作成します。
データ読み込みのボタンを(1)の画面に追加し、これを押すと USB メモリ上の PLOT.CSV を読み込み
保存されている軌跡を描画します。
データ読み込みボタンを押すと、USB メモリの PLOT.CSV を開き、1 行ずつ読み込んで文字列を数値に変換し、直線を描画します。
変換処理は、1 文字ずつ文字コードで判断し、文字コードが'0'～'9'のときに変換処理、','と NULL(0)のときに値の確定処理を行っています。

```
;******************************
;!ボタン_データ読み込み_押された時の動作のマクロ記述
;******************************
ADIM RFNO W
FUNC _ボタン_データ読み込み_押された時の動作
  ;ファイルを読み込みで開きます
  OPENFILE RFNO 0 $2:/PLOT.CSV
  IF RFNO!=#0 THEN
      ;ファイルが開けたら描画処理開始
      SUB _キャンバス OSELECT
      DO
          ;作業用のメモリ確保
```

```
        ADIM rbuf T 20
        ADIM addr w
        VALSET addr &rbuf
        ;ファイルから１行読み込みます
        READLINE RFNO #20 rbuf
        IF b(addr)=#&HFF THEN
            ;ファイル最後まで読んだらループ抜けます
            EXIT
        ENDIF
        IF b(addr)=#0 THEN
            ;先頭が０の場合空行なので開放処理します。
            VALSET OLDX #-1
            VALSET OLDY #-1
        ELSE
            ;それ以外の場合読んだデータを数値変換します。
            ;変換作業用のメモリを用意します。
            ADIM num W
            VALSET num #0
            DO
                IF b(addr)=#0 THEN
                  ;行末になれば変換用メモリの値をＹにセットしてループ抜けます
                  VALSET Y num
                  EXIT
                ELSEIF b(addr)=#&H2C THEN
                  ;&h2C[,]であれば変換用メモリの値をＸにセットします。
                  VALSET X num
                  VALSET num #0
                ELSEIF b(addr)>=#&H30 AND b(addr)<=#&H39 THEN
                  ;'0'～'9'の時、変換用の値を処理します。
                  ;現在の値を10倍し、読み込んだ文字コード-'0'(のコード)を加算します。
                  EXPR num=num*#10+(b(addr)-#&H30)
                ENDIF
                ;アドレスを進めます
                ADD addr #1
            LOOP
            ;行のデータを数値に変換し、直前値が開放でない場合、線を描画します。
            IF OLDX!=#-1 THEN
                LINE_EX OLDX OLDY X Y #0
            ENDIF
            VALSET OLDX X
            VALSET OLDY Y
        ENDIF
    LOOP
    ;ファイルを閉じます
    CLOSEFILE RFNO
    SUB _キャンバス OUNSELECT
  ENDIF
END_FUNC
```

5. 4 マクロポートを使用した通信処理

マクロポートと他の機器を接続し、その通信コマンドを処理するサンプルです。

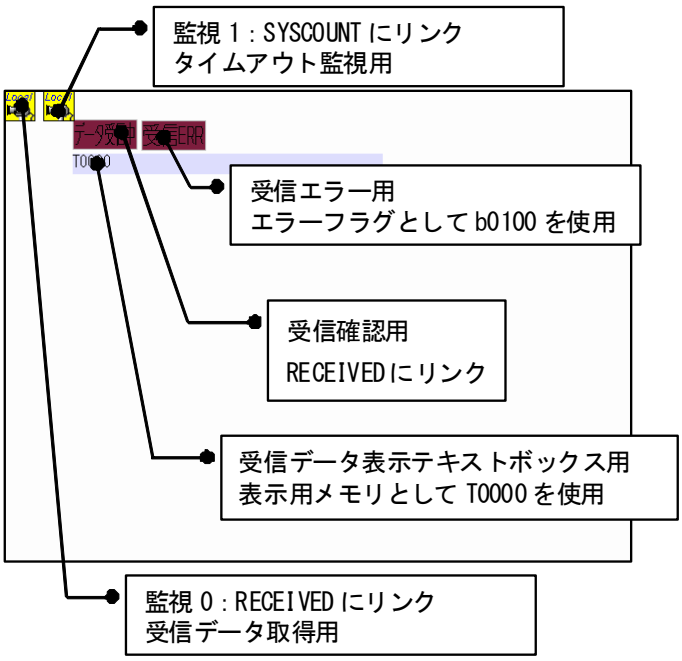
(1)無手順データの取得(バーコードリーダー等)

バーコードリーダー等、読み取った文字列を、そのまま出力するタイプの機器から送られてくるデータを読み取るためのサンプルを以下に示します。

機器の仕様として以下を想定します。
・データの読み取りに成功すると 10 バイトのデータを伝送。
・データの開始・終了コード等は特になし。
・データの送信は送り始めてから 500ms 以内には終了。
※上記は特定の機器を想定したものではありません。

上記のような機器からデータを受け取る場合、読み込み開始のトリガとしてはシステムメモリの RECEIVED(bF09A)を使用します。このメモリはマクロポートにデータが格納されれば 1 がセットされますのでこのメモリを監視メモリにリンクしておけば、読み込みをスタートすることが出来ます。

サンプルの画面は以下のようになります。



監視 0 で受信があったときの、データ取得動作をマクロで記述します。
監視 1 で受信が正常に取れないときのタイムアウト確認をします。
※データ受信開始から 1 秒以内に 10 バイト取得できない場合、エラーとします。
取得したデータは T0000 に格納しその結果をテキストボックスに表示します。
RECEIVED の状態と受信エラーの状態はランプで表示します。受信エラーのフラグは b0100 を使用します。
上記の仕様の時、監視 0 には以下のようなマクロを記述します。
監視 0 は動作条件を RECEIVED=1 のときと設定します。

```
;*****************************
;!監視0_動作のマクロ記述
;*****************************
;受信タイムアウト判定用予期込み開始時刻記憶メモリ
;0で取り込みなし
;0以外で開始時刻が保存されます
ADIM STARTTIME l
;読み込みバイト数カウント用メモリ
ADIM RCVCOUNT b
;読み込みデータ保存メモリ
ADIM RCVDATA T 11
;保存作業用アドレス指定用メモリ
ADIM addr w

FUNC _監視0_動作
  ;受信用のメモリ確保
  ADIM rcv W
  ;連続して取り出せるようループを回します。
  DO
      ;データ受信
      GETC rcv
      IF rcv!=#-1 THEN
          ;データがある場合
          IF STARTTIME=#0 THEN
              ;取り込み開始されていないならば、開始時刻をセット
              VALSET STARTTIME SYSCOUNT
              ;格納用のポインタ、カウントの初期化
              VALSET addr &RCVDATA
              VALSET RCVCOUNT #0
              ;エラー表示を解除
              MOV b0100 #0
          ENDIF
          ;取り込んだ値をバッファに格納
          VALSET b(addr) rcv
          ;ポインタとカウントを1進める
          ADD addr #1
          ADD RCVCOUNT #1
          IF RCVCOUNT=#10 THEN
              ;10バイト取得できた場合はデータをT0000にコピーします。
              ;受信データに終端をつける
              VALSET b(addr) #0
              ;表示用メモリにコピー
              CP 10 T0000 RCVDATA
              ;次のデータ読み込み様に、タイマをクリア
              VALSET STARTTIME #0
          ENDIF
      ELSE
          ;データがない場合RECIVEDをクリアしてルーチン終了
          MOV RECEIVED #0
          EXIT_FUNC
      ENDIF
      DOEVT
      REFSH
  LOOP
END_FUNC
```

監視１のタイムアウト確認マクロは以下となります。

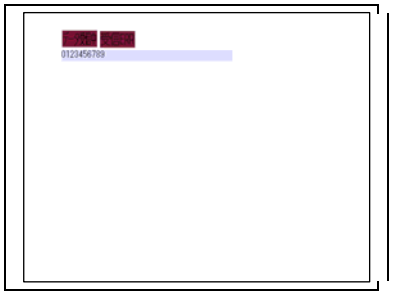
```
;*****************************
;!監視1_動作のマクロ記述
;*****************************
FUNC _監視1_動作
  IF STARTTIME!=#0 THEN
```

```
        ;取り込み中データタイムアウト確認(SYSCOUNT は 50ms 単位なので 50×20 で 1000ms 経過したか？)
        IF SYSCOUNT>(STARTTIME+#20) THEN
            ;エラー通知のためブザー
            MOV BUZER #10
            ;エラーランプのメモリをセット
            MOV b0100 #1
            ;開始時刻カウンタをクリア
            VALSET STARTTIME #0
        ENDIF
    ENDIF
END_FUNC
```

また、メモリ状態の初期化のためページ表示時の動作を指定します。

```
;*****************************
;!Page1_ﾍﾟｰｼﾞ表示時実行ﾏｸﾛのマクロ記述
;*****************************
FUNC _Page1_ﾍﾟｰｼﾞ表示時実行ﾏｸﾛ
  ;制御に使用するメモリを初期化
  VALSET RECEIVED #0
  VALSET b0100 #0
  VALSET STARTTIME #0
END_FUNC
```

動作イメージ



シリアルで 10 バイトまとまったデータ(ASCII 文字列)を受信するとテキストボックスに表示します。受信中は RECEIVED にリンクしたメモリが点滅します。
受信データが 10 バイトに満たない場合、受信エラーのランプが点灯します。
受信エラーは新しくデータが入ってくるとクリアされます。

6. マクロエラー

マクロを作成する過程で様々なエラーが発生します。文法的なエラーについては TPD V4 で画面転送時や、画面データチェック時にエラーの有無とその発生箇所を確認することが出来ます。以下にマクロエラーの内容について説明します。

6. 1 画面書き込みの流れ

TPD V4 が GOP に書き込むデータを作成する場合、以下の手順で行っています。

①TPD V4 が中間ファイルとして画面データ(IBC 形式)を出力。
②IBC ファイルの構文マクロを展開し IBA 形式のファイルを出力。
③IBA ファイルのラベル、メモリ名を実行番地、実メモリアドレスに変換し IBX ファイルを作成。
④IBX ファイルを GOP で実行可能なバイナリインタプリタ形式に変換(OUT 形式)。
⑤OUT 形式をバイナリ形式のテキストファイルからバイナリファイル(pagedata.dat)に変換。

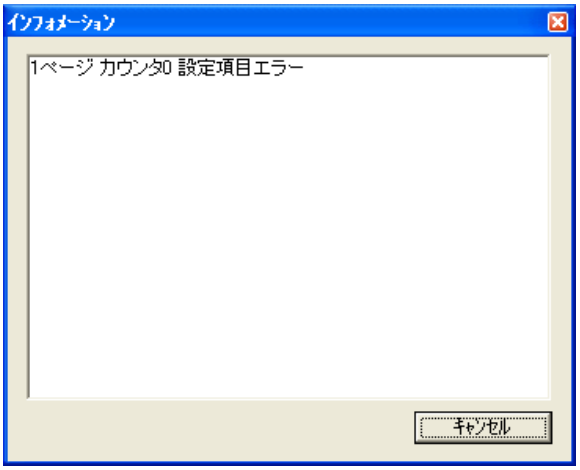
上記で作成した中間ファイルは一旦 TPD V4 を終了した後、次回の TPD V4 起動時に削除されます。
中間ファイルは、TPD V4 のメニューの「ファイル」→「ワークフォルダを開く」で表示されるフォルダの Temp¥tp????のフォルダ中に作成されます。

6. 2 各々のステップでのエラーメッセージ

上記の各ステップごとに以下のようなエラーを表示します。

(1) IBCファイル作成時

オブジェクトに必要なプロパティ設定項目がない場合以下のようなダイアログを表示します。
本内容はマクロの記述で発生することはありません。



必須プロパティが設定されていないオブジェクトが上記ダイアログ内に列記されます。

(2)構文マクロ展開時のエラー

IF,FOR,EXPR,FUNC 文を展開時のエラーです

メッセージ	エラー内容
?が開けません	作業用一時ファイルが開いたままの状態になっています。
閉じていない FUNC 文があります	FUNC に対応していない END_FUNC があります。
IF・FOR・DO の階層が深すぎるか、対応する ENDIF・NEXT・LOOP がありません	IF に対応する ENDIF がない。 FOR に対応する NEXT がない。 DO に対応する LOOP がない。 IF,FOR,DO 文の入れ子構造が深すぎる。
対応する FOR がありません	FOR がないのに NEXT 文があった
FUNC はネスト出来ません	FUNC のあと END_FUNC が呼ばれる前に別の FUNC 文があった。
式に誤りがあります	EXPR,IF 文で誤りがあった。 ()が閉じていない。 演算符号の間違い。
FOR 構文エラー	FOR 文の書式間違い。

(3)ラベル置き換え時のエラー

メモリ名称の実アドレス表記置き換えや、ラベルのインデックス変換時に発生するエラーです。
定義していないメモリやラベルを使用したした場合に発生します。

メッセージ	エラー内容
メモリ指定表記が正しくありません	メモリの間接参照表記方法間違い。 例 b(0000+#1) ⇒b(0000+1)
指定のメモリはありません	DIM,ADIM,EDIM,メモリリストで定義していないメモリ名を使用した。 例 ADIM ABC w MOV ABD #1 ;タイプミス
w 型または l 型でないメモリをインデックスに使用しています	間接参照のインデックスに w,l 以外の型で定義しているメモリ名を使用した。 ADIM index b MOV b(index+1) ;b 型をインデックスとして使用。
スコープのネストが深すぎます	SCOPE 文の入れ子が深すぎる。(16 以上。)
文法エラー	ADIM,EDIM,DIM 文の書式間違い。
EDIM 以外のコマンドで同名を使用しています	ADIM,DIM で同名のメモリ名を使用。 例 ADIM ABC w : EDIM ABC W ;これは NG (正) EDIM ABC W : EDIM ABC W ;これは OK
ADIM 使用個数が多すぎます	ADIM,EDIM で割り当て可能領域をオーバー。
ハッシュテーブルオーバー	TPD V4 内部のエラー メモリ、ラベル重複定義が多いと、内部変化用テーブルをオーバフローします。 (約 139000 件)
同名のメモリ名称は定義できません	メモリ名定義が重複しています。 例 ADIM AAA w : ADIM AAA b
同名のスイッチ名称は定義できません	TPD V4 内部エラー SW 登録した場合、重複したスイッチ名を使用しています。
LDIM 型指定がありません	LDIM 文で型指定がありません。
LDIM 使用個数が多すぎます	LDIM 用エリアを越えて割り当てようとしています。 同一スコープ内の LDIM 使用数が多すぎます。
同名のイメージキャッシュ ID は定義できません	内部エラー。 CREATE_IMAGECACHE 文で重複した ID を定義。

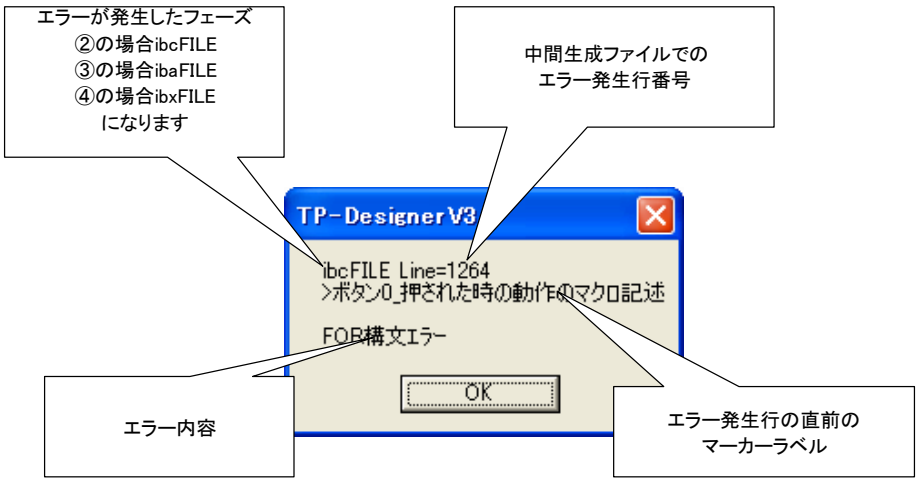
(4)GOPコマンド変換時のエラー

コマンドを最終的なバイナリ形式に変換する際に発生するエラーです。
存在しないコマンドや、コマンドのフォーマットに合致しないパラメータを指定した場合、このステップでエラーが発生します。

メッセージ	エラー内容
オペランド指定エラー 文法エラー コマンド指定エラー データ型指定エラー	コマンドのフォーマットに合致したパラメータの与え方になっていません。 ・与えられる型が違う。 ・パラメータの数が足りない。 ・存在しないコマンドを使用した。
ラベル数上限エラー	TPD V4 内部エラー(前工程でトラップされるます。) ラベル番号記述間違い。
ページ数上限エラー	TPD V4 内部エラー ページ登録最大数を超えたページ宣言をした。
0 除算エラー	DEV,MOD で除数に 0 を指定した。 (動的に 0 除算になるものは検出できません)
スイッチ登録数エラー	スイッチ数上限(8192)をオーバーした。
ラベル未定義 スイッチ未定義 ページ未定義	TPD V4 内部エラー 存在しないインデックスを指定した。
値範囲オーバー	パラメータとして指定できる範囲をオーバーした。

(5) エラーメッセージの表示

(2), (3), (4) ステップのエラーは以下のような表示になります。



マーカラベルとは、ラベルを ";!" に続けて書くことで、エラー箇所を特定しやすくします。
マーカーラベルは TPD V4 が自動でオブジェクトごとのマクロ等の先頭に出力しますがマクロエディタ中で任意の箇所に追加することも可能です。

マーカーラベルの記述例

```
;!ファイル書き込みルーチン
OPENFILE FNO 1 $2:/log/log.txt
WRITEFILEE FNO #&H0100 #&H20 w0000
CLOSEFILE FNO
```

上記の場合 WRITEFILE コマンドの表記が"WRITEFILEE"と誤っているため、エラーが発生します。
そのとき、エラーダイアログのマーカーラベルには"ファイル書き込みルーチン"と表示されます。

7. マクロ記述時の注意事項

(1) 無限ループ/無限連鎖

マクロで無限ループとなる動作を組み込むと、GOP がフリーズします。無限ループとならないように記述して下さい。

また、無限ループではないですが、繰り返しイベントが発生することで、無限ループと同じような症状になることがあります。以下のような場合に発生します。

- ・監視オブジェクト内のマクロで、自身のリンクメモリを操作した場合。
- ・マクロ挿入オブジェクトやページ表示時のマクロで REDARW コマンドを実行した場合。
- ・マクロ挿入オブジェクトやページ表示時のマクロで同じページ間のページ移動を行った場合。たとえば 1 ページ目で 2 ページに移動し、2 ページ目で 1 ページに移動するようにマクロを記述した場合。

(2) ページ移動

システムメモリの PAGE 制御関連メモリ(PAGE,PAGE2)に値をセットすると、ページ移動を行いますが、ページ移動実施のタイミングは実行中のマクロが終了した段階で行われます。

(REDRAW コマンドも同様です。)

ページ制御系の動作は、記述順と実行順が異なりますので、注意して記述して下さい。

※上記の防止のため、ページ制御系の動作はマクロの最終行に書くことをお勧めします。

※上記の仕様のためループ中でページ制御動作を行った場合、ループを抜けるまでページ移動は実行されません。このような場合は、ページ制御系のコマンドと合わせてループを抜ける処理もあわせて記述して下さい。