作成日 7/3/2014 版

TP-DesignerV4 FAQ集

～よくある質問と回答について～

TP デザイナーFAQ　質問一覧

Q.1 つのメモリで複数のランプを制御させたい

ランプのマスクを設定することで可能です。マスクについて以下に説明します。

マスクとは、リンクメモリの特定のビットについてのみ評価する場合、そのビットを指定するためのものです。

整数値は内部的に 2 進数で処理されています。

整数値(10 進数)	内部(2 進数)
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
6	00000110
:	:

2 進数で表した値の各桁を見るとOか1いずれかの状態になるので
1 桁目をランプ1、2 桁目をランプ2といった様に割り付けることが出来ます。
マスク値は評価したい桁(ビット)が 1 になるように整数値を指定します
例えば b0000 のメモリで 2 進数で 1 桁(0 ビット)目をランプ1、2 桁(1 ビット)目をランプ 2、
3 桁(2 ビット)目をランプ3にする場合、
ランプ1は1桁目だけが 1 の値なので、マスクには 1 を指定します。
ランプ2は 2 桁目だけが 1 の値なので、マスクには 2 を指定します。
ランプ3は 3 桁目だけが 1 の値なので、マスクには 4 を指定します。
比較値はマスクと同じ値を指定します。

また、メモリの型によるビットの数は異なっており、型毎のビット数は以下の通りです。

b,B 型で 8 ビット
w,W 型で 16 ビット
l,L 型で 32 ビット

尚、マスク値、比較値をセットする場合 4 ビットを 1 まとめにした 16 進表記で指定することも可能です

2 進	16 進	2 進	16 進
0000	0	1000	8
0001	1	1001	9
0010	2	1010	a
0011	3	1011	b
0100	4	1100	c
0101	5	1101	d
0110	6	1110	e
0111	7	1111	f

16 進表記で指定する場合が先頭に&H を付加します。

例

0100000000000000 ->&H4000 (10 進で 16384)

Q.DOEVTの使い方

DOEVT とは、マクロ実行中にマクロの動作により発生したイベントを行うためのコマンドです。
イベントとは監視やランプ、カウンタ等メモリにリンクしているオブジェクトがあるとき、リンクしてあるメモリに値変化があり、それに伴う動作が発生することです。
イベント発生時の動作ですが、例えば画面が以下のような構成だったとき

```
カウンタ  w0000 にリンク
ボタン
        ;マクロ1
        FOR w0000=#0 TO #10
        NEXT
        END
```

ボタンを押下するとボタンのマクロ 1 が実行されます。
マクロを実行するとカウンタのリンク条件に合致するため
カウンタの描画行進用の処理が呼ばれますが、その実行は現在実行中のマクロが終了後になります。マクロ終了時後にカウンタ描画処理を行いますがこの時点で w0000 は 10 になっているため、カウンタは 0～9 までの経過は表示せずに最終結果の 10 だけを表示します。
内部的な動作として、マクロ内で w0000 の値変化があるたびに w0000 にリンクしてある動作を検索します。カウンタがリンクしてあるためカウンタの描画処理が、実行動作としてイベント実行キュー(待ち行列)に格納されます。
イベント実行キューに開始情報が格納されると、ｲﾍﾞﾝﾄ実行が可能な状態(実行中の動作が終了した時等)に格納された動作を実行していきます。
したがって、上記の例だとマクロが終了した時点で、キューに格納されたカウンタの描画処理を行うため途中経過が表示されません。

DOEVT ですが、上記のようなときに途中経過を表示させたい場合に使用します。
DOEVT コマンドは DOEVT 実行時点でキューに格納されているイベントがあればその時点でそのイベントを実行します。
上記の例の場合

```
ボタン
        ;マクロ1改
        FOR w0000=#0 TO #10
                DOEVT   ;w0000 の変化で発生したイベントを実行
        NEXT
        END
```

とすると w0000 に変化によりキューに格納されたカウンタの描画処理を行います。
但し、カウンタやランプと言った画面変更を伴うイベント動作の場合これだけでは表示しません。

GOP の描画はちらつき防止のため内部フレームバッファに描画されます。ページ描画終了やマクロの終了等、描画すべき物がすべて描画し終わった段階で内部フレームバッファから表示用画面に転送されます。上記のマクロの例だとマクロ 1 改の END 実行時点で転送されます。
内部フレームバッファから表示用画面の転送を任意時点で行うためのコマンドとして REFSH があります。
したがって下のようにマクロを修正すると途中経過が表示されるようになります。

ボタン

```
;マクロ1改
FOR w0000=#0 TO #10
        DOEVT
        REFSH   ;イベントの実行により変化した画面の変化を表示面に反映
NEXT
END
```

※DOEVT が必要になる動作ではほとんどの場合 REFSH もペアで使用します。

Q.カウンタを押し数値入力させたい

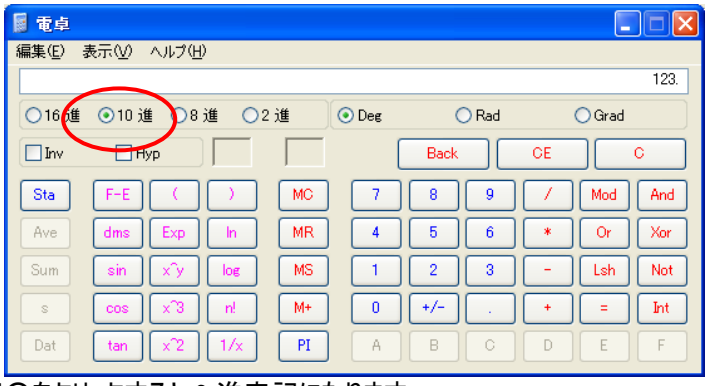
カウンタにキーパッド呼び出し機能を設定することが出来ます。
プロパティのキーパッド設定を"する"に設定すると、カウンタ上に透明のキーパッド呼び出しボタンが配置されます。

Q.マスクや 2 進数について

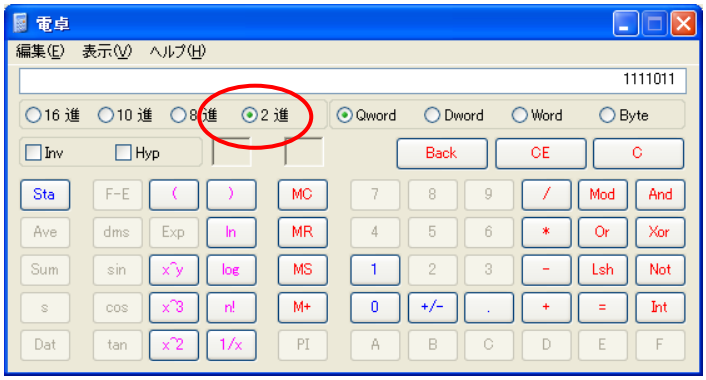
TP デザイナーの設定項目でマスク機能(ランプ、ボタンインターロック、グループ非表示設定等)を使用する場合、2 進数を使用した論理演算の知識を必要とします。
これらはソフト/ハードウェア開発者でなければなじみの薄い知識のため以下に簡単に説明いたします。

<2 進数について>
2 進数とは桁が上がる毎に 2 倍になっていく数値の表現方法です。
通常、私たちは 10 進数を使用しています。
10 進数だと
1 と 10 では 10 倍違います。
そしてひとつの桁には 0～9 の 10 個の数字が入ります。
同様に 2 進数では
1 と 10 では 2 倍違います
そしてひとつの桁には 0～1 の 2 個の数字が入ります。
(以下数値の表記で 2 進数を表す場合はわかりにくいので[]ではさんで表記します。)
2 進数で 0～10 を表現すると
0:[0]
1:[1]
2:[10]　ひとつの桁では表現できなくなるため桁が上がります
3:[11]
4:[100]　2 つの桁では表現できなくなるため桁が上がります
5:[101]
6:[110]
7:[111]
8:[1000]　3 つの桁では表現できなくなるため桁が上がります
9:[1001]
10:[1010]
となります。
10 進⇔2 進の変換については計算方法もありますが
windows のアクセサリの電卓を使用すると簡単に計算できます。
以下の方法で計算できます。
①電卓を立ち上げる。
関数電卓になっていない場合はメニュー−表示で関数電卓にします。

②下記○の 10 進が選択されているのを確認し、数値を入力します



③下記○をクリックすると 2 進表記になります



<なぜ 2 進数を使用するの？>

2 進数は各桁が0か1の 2 通りしかないため、それぞれを電気回路の L レベル(0V)、H レベル(5V)で表現できます。コンピュータは内部でこのようにして数値を保持しています。(GOP は内部にマイクロコンピュータを搭載しています)

コンピュータ内部では便宜上 8 桁を一まとめのデータ単位としています。この一まとめのデータ単位をバイト(byte)と言います。またそれぞれの桁(0か 1 の情報)をビットといいます。

GOP でも処理はバイトを基準にデータを読み書きします。(b,B 型1バイトずつ、w,W 型2バイトずつ、l,L 型 4 バイト)

しかしマスクを使う場合、バイトの中のビット単位で評価いたします。そのためマスクを指定する場合2進数の考え方が必要になります。

<マスク指定の仕方は>

マスクとは"マスク値として指定している値でビットが 1 になっているビットだけを評価する"と言うことです。

例

マスク値が 4[100]の場合メモリの値が 5 でも 6 でも同じ動きになります

5 の場合[101]となりますがマスクが[100]なので[1**(*は評価しない)]

6 の場合[110]となりますがマスクが[100]なので[1**(*は評価しない)]

となり動作としては同じ動きになります。

複数のオブジェクトをマスクを使用してひとつのメモリで使用する場合それぞれのオブジェクトで評価するビットが重ならないよう、2 のn乗の値を指定することが基本になります。

オブジェクト 1 では 2^0=1(^はべき乗記号)

オブジェクト 2 では 2^1=2

オブジェクト 3 では 2^2=4

:
オブジェクト n では 2^(n-1)
になります。
※リンクしてるメモリが b,B 型の場合 n は 8 まで
w,W 型の場合 n は 16 まで
l,L 型の場合 n は 32 まで 指定できます
このようにマスクを指定することでそれぞれのオブジェクトで他のオブジェクトに影響しないビットを指定することが出来ます。

<どうやってビット単位の値をセットするの？>
マスクを使用して複数のオブジェクトにリンクする場合、リンクしたメモリの値をビット単位で操作する必要があります。
特定のビットを操作するためには論理演算を行います。

特定のビットを 1 にするには OR という演算を行います。
OR 演算とは各桁のビットでどちらかが 1 ならば結果が 1 になる演算です。
GOP のマクロでは以下のように記述します
OR (結果格納メモリ) (OR する値)
で動作は
(結果格納メモリ) ←(結果格納メモリ)の現在値 OR (OR する値)
となります。
具体的な動作として
結果格納メモリが b0000 で現在値が 1 の場合に 4 と OR する場合
OR b0000 #4
で
b0000←1[00000001] OR 4[00000100]
となりますので各桁を比較しどちらかが 1 ならばその桁は 1 になります
[00000001]
[00000100]
↓
[00000101](10 進で 5)
となり、実行結果 b0000 の値は 5 になります。
したがってある桁のビットを 1 にしたい場合、2^(桁位置-1)の値を OR すればよいことになります。これはマスクで指定している値と同じ値ですので、下のようになります
OR (リンクメモリ) #(マスク値)

特定のビットを 0 にするためには AND という演算を行います。
AND 演算とは各桁のビットでどちらかが 0 ならば結果が 0 になる演算です。
GOP のマクロでは以下のように記述します
AND (結果格納メモリ) (AND する値)
で動作は
(結果格納メモリ) ←(結果格納メモリ)の現在値 AND (AND する値)
となります。
具体的な動作として
結果格納メモリが b0000 で現在値が7の場合に 251 と AND する場合
AND b0000 #251
で
b0000←7[00000111] AND 251[11111011]
となりますので各桁を比較しどちらかが 0 ならばその桁は 0 になります

[00000111]
[11111011]
↓
[00000011](10 進で 3)
となり、実行結果 b0000 の値は 3 になります。

AND 演算の場合、0 にしたいビットを 0、それ以外を 1 に指定します。これはマスク値の 0 と 1 を入れ替えた値になります。

※例として 3 桁目を使っている場合

マスク値は 2^(3-1)で 4 になっています。
これを 2 進数で表現すると[00000100]です。
AND を使って 3 桁目を 0 にする場合[11111011]を AND します。
これらの値を並べてみると
[00000100]
[11111011]となっており 0 と 1 が反転しています。

0 と 1 を入れ替えるのはあらかじめ計算してその値をセットしてもよいですが GOP に反転用の命令がありますので、その命令を使うことも出来ます。

反転用の命令は

NOT (反転したい値が格納されたメモリ)

です。

一連の流れをマクロで記述すると

MOV (リンクメモリと同じ型の作業メモリ) #(マスク値)
NOT (リンクメモリと同じ型の作業メモリ)
AND (リンクメモリ) (リンクメモリと同じ型の作業メモリ)

となります。

具体例として

リンクメモリ b0000
マスク 8
作業用メモリ b0001
上記の場合マクロは以下のようになります

```
MOV b0001 #8
NOT b0001
AND b0000 b0001
```

Q.メモリー括書込で文字の書き込みについて詳しく説明してほしい。

GOP のメモリに一括書き込みするために WB コマンドがあります。

WB コマンドはメモリの型にとらわれず、任意のアドレスから任意の数だけデータを転送するコマンドです。

メモリリストから

T0060

T008A

T00B4

と 3 つのメモリが確保されている場合、GOP 内部のメモリ配置は以下のようになります。

アドレス	(16進数表記)															
	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+a	+b	+c	+d	+e	+f
0060	T0060として確保された領域															
0070																
0080											T008aとして確保された					
0090	領域															
00a0																
00b0					T00b4として確保された領域											
00c0																
00d0																

ここで

T0060 に"abcdefg"

T008A に"hijklmn"

T00B4 に"あいうえお"

とセットする場合、それぞれのメモリの先頭から文字のコードをセットしていきますので

メモリの中は以下のようになります。

アドレス	(16進数表記)															
	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+a	+b	+c	+d	+e	+f
0060	'a'	'b'	'c'	'd'	'e'	'f'	'g'	0	*	*	*	*	*	*	*	*
0070	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0080	*	*	*	*	*	*	*	*	*	*	'h'	'i'	'j'	'k'	'l'	'm'
0090	'n'	0	*	*	*	*	*	*	*	*	*	*	*	*	*	*
00a0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
00b0	*	*	*	*	'あ'		'い'		'う'		'え'		'お'		0	*
00c0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
00d0	*	*	*	*	*	*	*	*	*	*	*	*	*	*		
"でくくってある箇所は、"内の文字のコードが入る																
*は値が何であってもGOPの動作に影響しない。書き込む場合は0を書く。																

文字部をコードに置き換えると以下のようになります

アドレス	(16進数表記)															
	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+a	+b	+c	+d	+e	+f
0060	61	62	63	64	65	66	67	00	00	00	00	00	00	00	00	00
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0080	00	00	00	00	00	00	00	00	00	00	68	69	6a	6b	6c	6d
0090	6e	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00b0	00	00	00	00	82	a0	82	a2	82	a4	82	a6	82	a8	00	00
00c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00		

したがってコマンド送信する場合

WB 0060

6162636465666700000000000000000000

00000000000000000000000000000000

00000000000000000000000068696a6b6c6d

6e00000000000000000000000000000000

00000000000000000000000000000000

0000000082a082a282a482a682a80000

00000000000000000000000000000000

00000000000000000000000000000000

WBE

と送信します。

ただ、WB コマンドで文字を送る場合、WD コマンドを送るよりデータ量が増えてしまいます。

動作に支障がない場合 WD コマンドでの送信をお勧めします。

WB コマンドは C のメモリブロックと同期を取ることを想定しています。

例えば、

GOP 側

w0000

W0002

F0004

l0008

L000c

ホスト側

```
struct _goplink{
        unsigned short w0000;
        short w0002;
```

```
        float F0004;
        unsigned long l0008;
        long l000c;
} goplinkdata;
```

とした場合,(※GOP のメモリ配置とホスト側のリンクブロックの構成を合わす)

```
unsigned char *senddata;
unsigned char line[34],buf[3];

senddata= goplinkdata;
GopSendCommand("WB 0000");
strcpy(line,"");
for(i=0;i<sizeof(struct _goplink);i++){
        sprintf(buf,"%02x",*senddata++);
        strcat(line,buf);
}
GopSendCommand(line);
GopSendCommand("WBE");
```

といった風に型やメモリの数を気にせずに転送することが出来ます。
※実際に使用時は、CPU のアライメント規制やエンディアンを十分理解したうえで
　ブロック転送ルーチンを作成して下さい。

Q.ランプのラベルを変化させたい

ランプのプロパティのラベル制御を"する"に設定し、ラベル制御メモリに任意のテキスト型メモリを指定することでラベルを動的に変化させることが出来ます

ラベル制御メモリの内容を変化させると、ランプのラベルが変化します。

Q.ランプを点滅させたい。

以下の手順で設定します。

<ランプ>

①ランプの ON 時点滅条件に値をセットします。

ON 時点滅条件は比較値と同じでも異なっていてもどちらでもよいです。同じ場合は、

点灯時は常に点滅いたします。異なる値の場合は、点灯と点滅を別々に制御できます。

ランプの点滅は ON 時色(ON 時ビットマップ)と OFF 時色(OFF 時ビットマップ)を交互に繰り返します。

<マルチランプ>

①状態設定で点滅を行いたい状態の行で点滅の設定を"する"にします。

マルチランプの点滅は、状態で指定された表示色(表示ビットマップ)と点滅消灯時色(点滅消灯時ビットマップ)を交互に繰り返します。

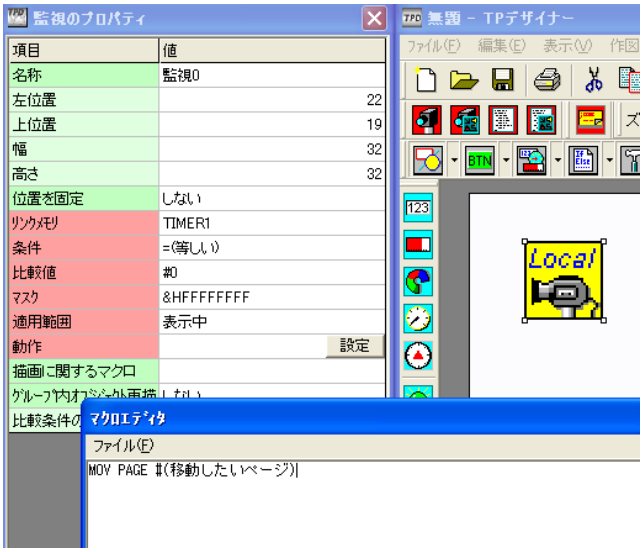
Q.時間が経つと自動的にページジャンプする方法(無操作時の自動復帰等)

時間がたつとページジャンプさせるには、監視オブジェクトを TIMER メモリをリンクさせて使用します。

①TIMERn にリンクした監視オブジェクトを作成します。

比較値は 0 として、動作の中でページ移動用のマクロを記述します。

MOV PAGE #(移動したいページ)

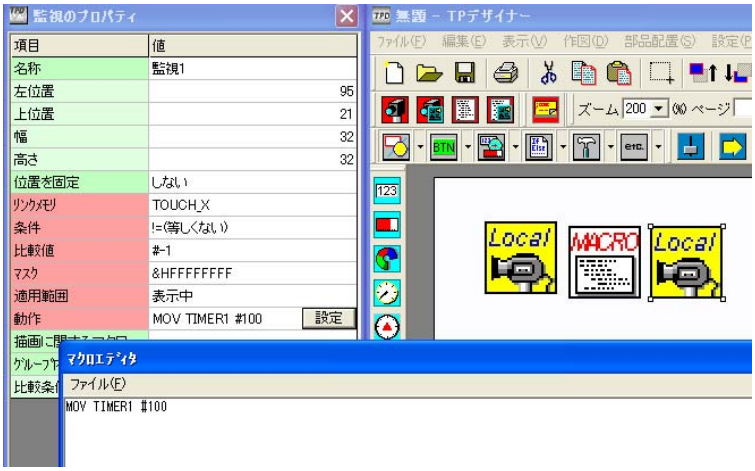


②タイマをスタートさせるためのマクロ挿入オブジェクトを配置します。

動作で TIMER メモリに待ち時間をセットするマクロを記述します。

MOV TIMER1 #(待ち時間)

③画面の操作を確認するため TOUCH_X(押下点の座標)にリンクした
監視オブジェクトを作成します。
値が-1 以外になったときに TIMER に待ち時間を再セットするよう
動作にマクロを記述します
MOV TIMER1 #(待ち時間)・・・②と同じもの



以上で一定時間画面操作が無い場合に、ページジャンプすることが出来ます。

Q.小数の表現方法は？

小数をカウンタで表示する場合、小数点以下桁数に任意の桁数を指定することで
小数点以下の表示が可能になります。
但し、リンクされるメモリの型によりメモリ格納値と表示値の対応に違いがあります。

<リンクメモリが整数型(b,B,w,W,l,L)>

メモリ格納値を 10^(小数点以下桁数)で割った値が表示されます。

例

小数点以下桁数　2
リンクメモリの値　　123
カウンタ表示値　　123÷10^2=1.23

<リンクメモリが浮動小数点型(F)>

メモリ格納値を小数点以下桁数まで表示します。

例

小数点以下桁数　2
リンクメモリの値　　123.456
カウンタ表示値　　123.45

※浮動小数点形について

浮動小数点型とは GOP 内部で値を保持するのに
符号×仮数×2^指数と行った形式で保持しており
値範囲が非常に小さな値から大きな値まで表現可能な反面
誤差が発生するといった問題もあります。
尚、浮動小数点型は IEEE754 単精度(C 言語の float 型)形式で
処理されています。

※整数型と浮動小数点型の使い分けについて

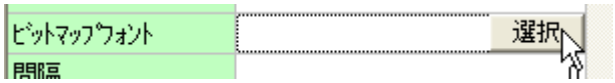
扱う値範囲が限られている場合(0.00001～1.0000 等)、整数型を使用した方が精度、速度(GOP としてはほとんど差はありませんが、ホスト側とリンクする場合、ホストのマイコンによっては浮動小数点の使用は大きな CPU パワーを消費することがあります)の面で有利です。
GOP 内部で浮動小数点コマンド(SIN 等)を使用する場合や、値範囲が大きい場合(0.000001～10000000 等)は浮動小数点を使用する必要があります。

Q.カウンタの数字のデザインを見栄えよくしたい

ビットマップフォントとして 0～9、.(小数点)、-(マイナス)、E(表示不可)を
ビットマップ画像から作成することが出来ます。

作成手順は以下の通りです。

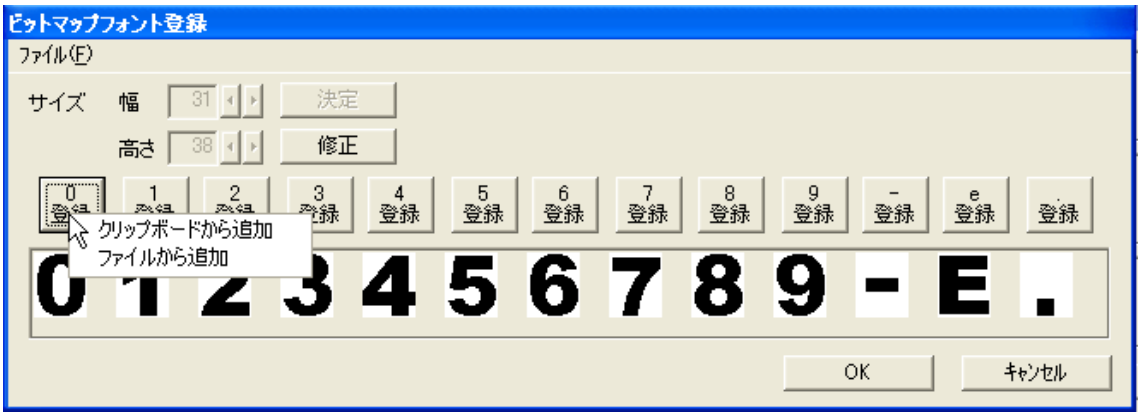
①プロパティのﾋﾞｯﾄﾏｯﾌﾟﾌｫﾝﾄの選択をクリックします。



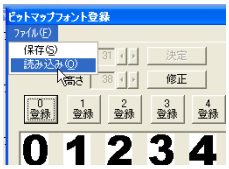
②ビットマップフォントの選択画面が表示されます。

登録されているﾋﾞｯﾄﾏｯﾌﾟﾌｫﾝﾄがない場合フォント表示欄をダブルクリックすると登録画面に
なります。

③それぞれの文字に対応するボタンでファイルまたはクリップボードよりビットマップを登録します。



また、ファイルの読み込みで登録済みのフォントセットを読み込むことも出来ます。

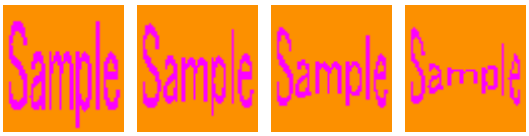


Q.アニメーションの作り方

アニメーションオブジェクトは複数のビットマップを一定間隔で切り替えることで動きを表現しています。

作成するには

①アニメーションの各コマのビットマップを作成・登録する。



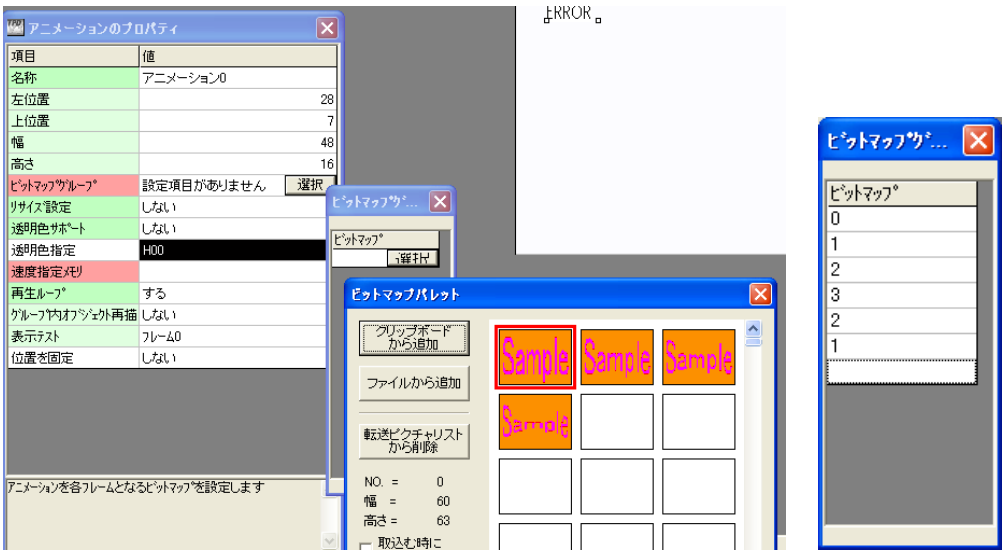
②アニメーションのアイコン  をクリックしアニメオブジェクトを配置

配置後の初期表示は登録されているビットマップが無いため ERROR と表示されます

③アニメーションのプロパティでビットマップグループを選択します。

ここで①のリストに表示するコマ順にビットマップを選択していきます。

同じビットマップを複数使用しても問題ありません。



④速度指定メモリを選択します。ここで選択したメモリにコマ送り間隔(100ms 単位)を指定することでアニメーションの再生が始まります。

※ボタン等でここで指定したメモリに値をセットしてください。

⑤オプションとして再生ループとして、再生終了後繰り返すか、一回限りで終了(再生ループしない)するかを指定できます。

再生ループをしない場合、再生終了後速度指定メモリに 0 がセットされます。ここを再度 1 以上の数字に設定すると再び再生が始まります。

Q.GOP-4000 で使用できる色のカラーコード

GOP-4000 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 進表現

4	2	1	F

となります。(421F)

TP デザイナーのカラーパレット右下に表示される値は色カラーコードの 16 進となります。

Q.文字列の取り扱いについて

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。
例えば、T0100 に"ABCDEF"と格納する場合
ﾒﾓﾘ的には

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。
例えば

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とトリガとなります。)

テキストメモリの先頭のアドレスがわからない場合は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
```

Q.GOP-4000 で多国語を扱うには

GOP-4000 で多国語で画面設計を行う場合、TP デザイナーの表示言語の設定のツールバーから言語を設定します。

選択可能な言語は以下のとおりです。

言語	フォントセット	言語指定番号	文字コード
日本語	JIS 第一,第二水準	0	SHIFT-JIS
英語	ISO-8859-1	96	ASCII
ドイツ語	ISO-8859-1	97	ISO-8859-1(Latin-1)
イタリア語	ISO-8859-1	98	ISO-8859-1(Latin-1)
スペイン語	ISO-8859-1	99	ISO-8859-1(Latin-1)
フランス語*1	ISO-8859-1	100	ISO-8859-1(Latin-1)
ロシア語*2	JIS 第一,第二水準	128	UTF-8

*1 フランス語には一部ISO-8859-1に含まれない文字があります。これを使用する場合代替文字を使用してください。

*2 ロシア語は日本語フォントセットに含まれるキリル文字(全角)を使用して表示します。半角での表示は出来ません。

なお、日本語以外の言語を使用する場合ストロークフォントは使用できません。
また、文字オブジェクトは日本語で設定した内容によりオブジェクトサイズが決定され他の言語では、この範囲に収まるように縮小表示されます。

日本語での文字の大きさ
Size of character in English

日本語以外で文字つぶれが好ましくない場合、日本語の表示内容の設定でスペース等を入れサイズを調整してください。

日本語での文字の大きさ
Size of character in English

多国語で作成したデータを GOP-4000 に転送後 GOP のシステムメモリの ENCODE(bF088)に言語ごとの指定番号を書き込むことで GOP の表示言語モードが変化します。
なお表示に付いては表示言語モードを切り替えた以降の描画から反映しますので、表示を切り替えるには ENCODE に値をセットした後、ページを移動するか REDRAW コマンドを実行する必要があります。

テキストボックス等にリンクしたテキストメモリへの書き込み時、書き込む文字列の文字コードは言語モードに応じた文字コードで書き込む必要があります。

Q.16 進数状文字列と整数値の相互変換を行う

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について説明します。

<16 進数状の文字列を整数値に変換>

```
;SRC はテキスト型で 16 進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)
;変換元データ格納アドレス
EDIM srcAddr w
;変化後データ格納アドレス
EDIM destAddr w
EDIM loopct b
;アドレスを取得
MOV     srcAddr &SRC
;アドレスを取得
MOV     destAddr &DEST
;対象機種が GOP-32V,GOP-4000 の場合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、l,L,場合 4)-1 分進める
;EXPR    destAddr= SIZE- #1(この行をコメントからは外す)
FOR loopct=#1 TO SIZE
        ;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
        ADD destAddr #1
        ;対象機種が GOP-32V,GOP-4000 の場合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 (この行をコメントから外し上の行をコメントアウト)
LOOP
END
```

<整数値を 16 進数状の文字列に変換>

```
;SRC は整数型(b,B,w,W,l,L)のうちいずれか
;DEST はテキスト型
;SIZE は SRC の型に応じた値が入っていること(b,B=1,w,W=2,l,L=4)
;変換元データ格納アドレス
EDIM srcAddr w
;変化後データ格納アドレス
EDIM destAddr w
EDIM loopct b
;アドレスを取得
MOV     srcAddr &SRC
;アドレスを取得
MOV     destAddr &DEST
;対象機種が GOP-32V,GOP-4000 の場合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、l,L,場合 4)-1 分進める
;EXPR    srcAddr = SIZE- #1(この行をコメントからは外す)
FOR loopct=#1 TO SIZE
        ;上位 4bit を変換(10H で割った商を変換)
        IF b(srcAddr)/#&10 <#10 THEN
                EXPR b(destAddr)= b(srcAddr)/#&H10+#&h30
        ELSE
                EXPR b(destAddr)= b(srcAddr)/#&h10+#&h37
                ; &H37 は&h41('A'の ASC-&H0a)
        ENDIF
        ADD destAddr #1
        ;下位 4bit を変換(10H で割った余りを変換)
        IF b(srcAddr)%#&10 <#10 THEN
                EXPR b(destAddr)= b(srcAddr)%#&H10+#&h30
        ELSE
                EXPR b(destAddr)= b(srcAddr)%#&h10+#&h37
        ENDIF
        ADD srcAddr #1
        ;1 ﾊﾞｲﾄ進める
        ADD destAddr #1
        ;対象機種が GOP-32V,GOP-4000 の場合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 (この行をコメントから外し上の行をコメントアウト)
LOOP
END
```

<使用例>

・整数値を 16 進数状の文字列に変換マクロを組み込んだ監視オブジェクト+テキストボックスで
  16 進表示カウンタ
・マクロポートでの通信での 16 進表示のデータの伝文作成、解析

Q.マクロで文字列を組み立てる

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"(_はスペースとします。)

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。

```
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
```

Q.日時を記憶したログをCSVファイルで保存するには

トレンドバッファに格納されたログを CSV 形式で出力することは標準機能で可能ですがデータのみしか出力できません。
ログの測定時刻などの付加データを一緒に行いたい場合、マクロを使用しファイルに書き込む必要があります。
マクロを使用し CSV ファイルを作成する場合以下のような手順で行います。
①ファイルをオープンします。
　ログですので基本的には追記モードでオープンしますが、初回データ書き込み等まだファイルが存在しない場合、ファイルオープンが失敗しますのでその場合、書き込みモードでオープンします。
②CSV に書き込むレコードを作製します。
　レコードは文字列を組み立てて作成します。
　FAQ の"マクロで文字列を組み立てる"も参考ください。
③②で作成した文字列をファイルに書き出します。
④ログファイルを閉じます。
　上記の手順をサンプリングにあわせて行います。

なお、CSV の出力先に付いてですが、サンプリング周期が高速(1 秒以下)の場合やオートリピートを使用時に出力する可能性がある場合、USB メモリに直接出力すると書き込み速度が遅いため操作感が低下する場合があります。
そのような場合、サンプリング時の書き出しは RAMDISK に対して行い、取り出しに RAMDISK から USB メモリに転送する等の運用上の工夫が必要になる可能性があります。
※上記の方法の場合 GOP の電源断で USB に書き込まれていないログは消失いたします。

マクロでの記述例は以下のようになります
[サンプリングのタイミングで動作する監視オブジェクトのマクロに記述してください]

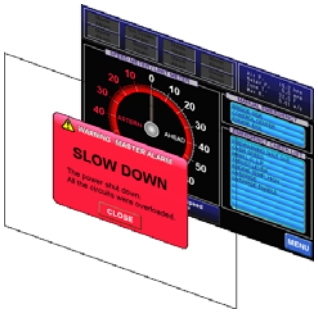
```
ADIM FNAME T 40          ;ファイル名指定用のテキストメモリ
ADIM FNO W               ;ファイル番号指定用の W メモリ
ADIM LINESTR T 40        ;書き込み行の文字列用のテキストメモリ
ADIM TEMPSTR T 40        ;書き込み行作成作業用のテキストメモリ
FUNC _LOG 出力監視_動作
                ;書き込み先によりドライブを設定
                ;ファイル開く場合"<ドライブ>:<ファイル名>"の形でファイルを指定します
                ;USB のときは"2:<ファイル名>"、RAMDISK の場合は"0:<ファイル名>"で
                ;ファイル名指定文字列を作成します
                CP 40 FNAME $2:DATALOG.CSV
                ;CP 40 FNAME $0:DATALOG.CSV
                ;追記モードで上記で指定したファイルを開きます
                ;          +--追記モード(3:追記  1:書き込み)
                ;          |
                OPENFILE FNO 3 FNAME
                ;ファイルオープンに成功すると FNO に 0 以外の値が入ります
                IF FNO=#0 THEN
                        ;ファイルが開けない場合書き込みモードでオープン
                        OPENFILE FNO 1 FNAME
                        ;ファイルオープンに成功すると FNO に 0 以外の値が入ります
                        IF FNO=#0 THEN
                        ;書き込みモードでも開けない場合、失敗とし書き込み処理は
                        ;行いません
```

```
                                        EXIT_FUNC
                                    ENDIF
                                ENDIF
                                ;20YY/MM/DD HH:MM:SS,L0003,L000B,L000F となるように書式化します
                                ;"20"
                                CP 80 LINESTR $20
                                ;YY の部分を作成
                                FMT 2 0 3 TMPSTR YEAR
                                ;"20YY"
                                APD 80 LINESTR TMPSTR
                                ;"20YY/"
                                APD 80 LINESTR $/
                                ;MM の部分を作成
                                FMT 2 0 3 TMPSTR MONTH
                                ;"20YY/MM"
                                APD 80 LINESTR TMPSTR
                                ;"20YY/MM/"
                                APD 80 LINESTR $/
                                ;DD の部分を作成
                                FMT 2 0 3 TMPSTR DAY
                                ;以下同様
                                APD 80 LINESTR TMPSTR
                                APD 80 LINESTR $
                                FMT 2 0 3 TMPSTR HOUR
                                APD 80 LINESTR TMPSTR
                                APD 80 LINESTR $:
                                FMT 2 0 3 TMPSTR MINUTE
                                APD 80 LINESTR TMPSTR
                                APD 80 LINESTR $:
                                FMT 2 0 3 TMPSTR SECOND
                                APD 80 LINESTR TMPSTR
                                APD 80 LINESTR $,
                                FMT 9 0 0 TMPSTR L0003
                                APD 80 LINESTR TMPSTR
                                APD 80 LINESTR $,
                                FMT 9 0 0 TMPSTR L000B
                                APD 80 LINESTR TMPSTR
                                APD 80 LINESTR $,
                                FMT 9 0 0 TMPSTR L000F
                                APD 80 LINESTR TMPSTR
                                ;書き込みデータの作成ここまで

                                ;開いたファイルに上で作った文字列を書き込みます
                                WRITELINE FNO LINESTR
                                ;ファイルを閉じます
                                CLOSEFILE FNO
                        ENDIF
                END_FUNC
```

Q.レイヤーの使い方

GOP-4000 シリーズは重ね合わせ表示用のレイヤーを持っています。
レイヤーを使用すると以下のような表示が可能になります。



このような表示は以下の手順で行います。
①TP デザイナーで背面となるページを作成します。
②TP デザイナーで前面となるページを作成します。
　このページは背景色を透明に設定してください。
③背面レイヤーは PAGE(wF000)のメモリに表示したいページ番号を入れることで表示します。
　マルチアクションのページジャンプも同じ動作を行います。
④前面レイヤーは PAGE2(wF002)のメモリに、前面に表示させたいページ番号を入れます。
⑤前面レイヤーの表示を消すには PAGE2(wF002)のメモリに 0 をセットします。
⑥背面レイヤーは消すことは出来ません。必ずどこかのページを表示いたします。

レイヤーを重ねて表示する場合、背面/前面とも画面に配置しているオブジェクトは動作いたします。但し背面のレイヤーのボタン認識範囲は、前面レイヤーに配置されるオブジェクトが全て含まれる矩形範囲外となります。

前面レイヤーを半透過表示にするには ALPHA2(bF008)に透過率(0～31)を設定した状態で PAGE2(wF002)に表示するページをセットします。

※TP デザイナーで用意してある各種キーパッドもレイヤーを使用しています。これの表示レイヤーは前面レイヤーのさらに前面に表示いたします。このレイヤーはキーパッド専用としているので制御用メモリは非公開としています。

Q.エフェクトの使い方

ページを切り替え時に色々な表示効果を行うことで見た目にインパクトのある画面を作ることが出来ます。

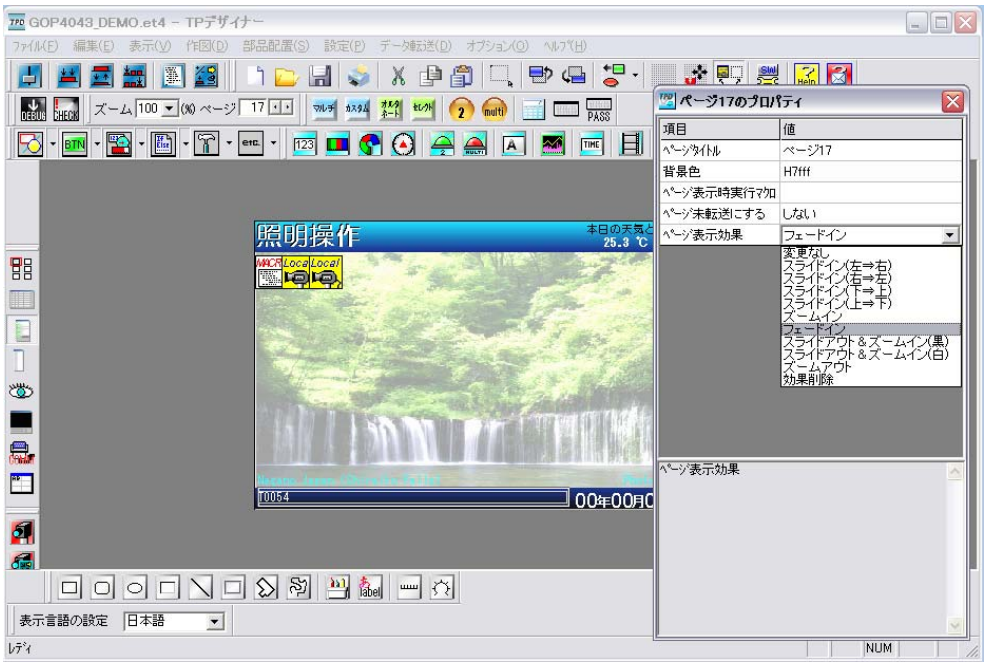
・エフェクトの種類

フェード

スライド

・エフェクトの設定方法

エフェクトは TP デザイナーの画面設計時にページのプロパティを設定することで簡単に設定できます。



また、システムメモリの EFFECT1(bF00B)[レイヤー1に表示する場合のエフェクトを指定]及び EFFECT2(bF00C)[レイヤー2に表示する場合のエフェクトを指定]に値を書込むことで GOP の動作時にもエフェクトを追加・変更することが出来ます。

Q.マクロを使用して、キーパッドを呼び出したい。

キーパッド呼出には、TPD 側で用意しているマクロルーチンを呼び出しことで使用できます。
具体的な呼び出し方法は以下のとおりです。

①テンキーの場合

```
;初期表示を TP デザイナー側で作成
FMT <最大文字数> <小数点以下桁数> <書式> _NumkeyDispText <リンクしたいメモリ>
;テンキー表示時にエフェクトをかけたい場合以下をセット
;番号はページのプロパティの並び順
VALSET _NumkeyEffect #0
;テンキーを半透過設定したい場合以下を設定
VALSET _NumkeyAlpha #0
;リンクメモリアドレス
VALSET _NumkeyLinkMemAddr &<リンクしたいメモリ>
;リンクメモリタイプ
VALSET _NumkeyLinkMemType : <リンクしたいメモリ>
;キーパッド配置ページ
VALSET _NumkeyKeyPage <ページ番号>
;元画面ロック有無
;キーパッドーを表示後、呼び出しもとのページを操作可能にする場合は以下を 0 に変更
VALSET _NumkeyIsBaseLock #1
;フォーカス消失時の処理
;_NumkeyIsBaseLock 0 を指定時,別のキーパッド呼び出し操作を行った場合、
;現在編集中の値の扱いを指定
;0 キャンセル扱い 1 確定扱い
; _NumkeyIsBaseLock が 1 のときは意味を持ちません
VALSET _NumkeyIsLostFoucs #1
;範囲チェック有無
;確定時の範囲チェック有無の設定
;1 のとき範囲チェックします。
VALSET _NumkeyRangeCheck #1
;範囲上限
VALSET _NumkeyRangeCheckMax <上限値>
;範囲下限
VALSET _NumkeyRangeCheckMin <下限値>
;範囲オーバー時の処理
;0 キャンセル 1:最大または最小に丸め
VALSET _NumkeyRangeOver #0
;整数桁数(整数部[小数部が 0 以外のときは+1 する(バグのため)])
VALSET _NumkeyIntLength <整数部桁数>
;小数部桁数
VALSET _NumkeyFloatLength <小数部桁数>
;ﾀｲﾄﾙ
CP 10 _NumkeyTitle <キーパッド>
;上記値セット後
SUB _ProcNumkeyCall
```

サンプル

```
;*****************************
;!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ください
  ;初期表示をTPデザイナー側で作成
  FMT 6 2 0 _NumkeyDispText I0000
  ;リンクメモリアドレス
  VALSET _NumkeyLinkMemAddr &I0000
  ;リンクメモリタイプ
  VALSET _NumkeyLinkMemType :I0000
  ;キーパッド配置ページ
  VALSET _NumkeyKeyPage #2
  ;元画面ロック有無
  VALSET _NumkeyIsBaseLock #1
  ;範囲チェック有無
  VALSET _NumkeyRangeCheck #1
  ;範囲上限
  VALSET _NumkeyRangeCheckMax @250
  ;範囲下限
  VALSET _NumkeyRangeCheckMin @0
  ;範囲オーバー時の処理
  VALSET _NumkeyRangeOver #0
  ;整数桁数
  VALSET _NumkeyIntLength #4
  ;小数部桁数
  VALSET _NumkeyFloatLength #2
  ;ﾀｲﾄﾙ
  CP 10 _NumkeyTitle $
  ;上記値セット後
  SUB _ProcNumkeyCall
END_FUNC
```

②文字キー

```
;TP デザイナーI/F
;エフェクト(テンキー参照)
VALSEt _MojikeyEffect #0
;半透過(テンキー参照)
VALSET _MojikeyAlpha #0
;リンクメモリアドレス
VALSET _MojikeyLinkMemAddr &<リンクしたいメモリ>
;キーパッド配置ページ
VALSET _MojikeyKeyPage <キーパッド配置ページ>
;元画面ロック有無
;キーパッドーを表示後、呼び出しもとのページを操作可能にする場合は以下を 0 に変更
VALSEt _MojikeyIsBaseLock #1
;文字数
VALSET _MojikeyLength #40
;ﾀｲﾄﾙ
CP 10 _MojikeyTitle $(タイトル)
;上記値セット後
SUB _ProcMojikeyCall
```

サンプル

```
;*****************************
;!ボタン 121_押された時の動作のマクロ記述
;*****************************
FUNC _ボタン 121_押された時の動作
    ;ここにマクロを記述してください
    ;リンクメモリアドレス
    VALSET _MojikeyLinkMemAddr &T0100
    ;キーパッド配置ページ
    VALSET _MojikeyKeyPage #3
    ;元画面ロック有無
    VALSET _MojikeyIsBaseLock #1
    ;文字数
    VALSET _MojikeyLength #40
    ;ﾀｲﾄﾙ
    CP 10 _MojikeyTitle $
    ;上記値セット後
    SUB _ProcMojikeyCall
END_FUNC
```

③パスワードキーパッド

```
;TP デザイナーI/F
;パスワード配置アドレスの先頭
;パスワード格納のテキストメモリのアドレス
VALSET _PassAddrTop &<パスワード格納のテキストメモリ>
;パスワードサイズ
;1ﾊﾟｽﾜｰﾄﾞあたりの最大文字数+1(Null 終端)
VALSET _PassSize <サイズ>
;上記位置に連続して配置されるパスワード数
;複数のパスワードがある場合
;パスワード個数
VALSET _PassNum <個数>
;動作モード
;0 照合 1 パスワード変更
VALSET _PassMode #0
;エフェクト(テンキー参照)
VALSET _PassEffect #0
;透過率(テンキー参照)
VALSET _PassAlpha #0
;キーパッド配置ページ
VALSET _PassKeyPage <パスキー配置ページ番号>
;照合 OK 時移動先ページ
VALSET _PassKeyOKPage <OK 時の移動先>
;上記値セット後
SUB _ProcPassCall
```

サンプル

```
;*****************************
;!ボタン 1177_押された時の動作のマクロ記述
;*****************************
FUNC _ボタン 1177_押された時の動作
  ;ここにマクロを記述してください
  ;パスワード配置アドレスの先頭
  VALSET _PassAddrTop &T1000
  ;パスワードサイズ
  VALSET _PassSize #9
  ;パスワード個数
  VALSET _PassNum #1
  ;動作モード
  VALSET _PassMode #1
  ;キーパッド配置ページ
  VALSET _PassKeyPage #4
  ;照合 OK 時移動先ページ
  VALSET _PassKeyOKPage #6
  ;上記値セット後
  SUB _ProcPassCall
END_FUNC
```

※

TPD 側で用意しているマクロモジュールは以下のフォルダにあります。
C:¥Program Files¥Ishiihyoki¥TPDesignerV4¥MacroModule¥VGA
テンキー　Tenkey.ibm
文字キー Mojikey.ibm
パスワードキー　　Passkey.ibm
上記をエディタ等で開くと、ソース参照できます。変更はしないようにしてください。

Q.マクロで使用できるオブジェクトのプロパティ

マクロを使用時にオブジェクトの設定情報を取得するため、オブジェクトごとに以下の定数が定義されます。

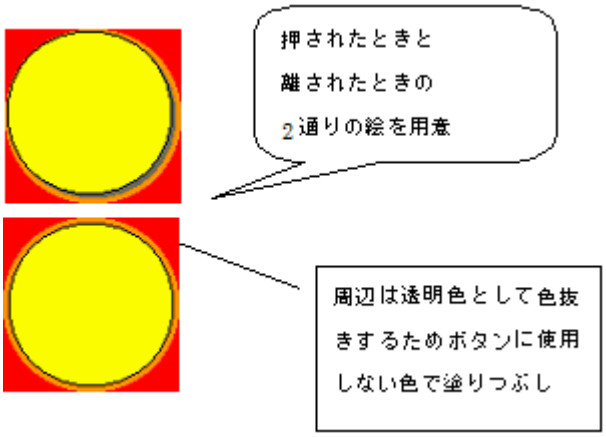
オブジェクト左位置　　_[オブジェクト名].X
オブジェクト上位置　　_[オブジェクト名].Y
オブジェクト幅　　_[オブジェクト名].W
オブジェクト高さ　　_[オブジェクト名].H

また表示非表示のメモリを指定時、指定したメモリに以下の別名が定義されます。
表示非表示制御　　_[オブジェクト名].UNVISIBLE

また透過率をメモリで指定時、指定したメモリに以下の別名が定義されます。
表示非表示制御　　_[オブジェクト名].ALPHA

Q.丸型など四角でないボタンを作成する

丸型のボタンはビットマップのボタンとして作成する必要があります。



ビットマップを用意し

ボタンのプロパティを設定します

スタイル　ビットマップまたはリサイズブルビットマップ

枠の表示　しない

押下時表示変更　する

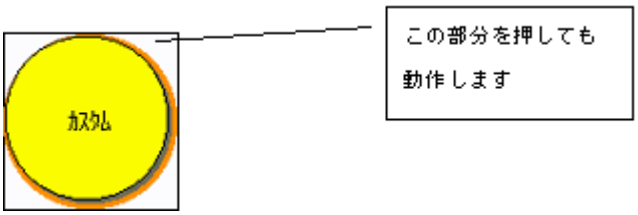
枠内ビットマップ　　作成したビットマップを指定

押下時枠内ビットマップ　↑

ビットマップ透明色サポート　する

ビットマップ透明色　ビットマップで周辺に塗りつぶした色

なおスイッチエリアは矩形なため、ボタンの外側の範囲を押してもボタンは動作します。



Q.オブジェクトに！マークが表示され、GOPへの転送が出来ない

オブジェクトの ！ マークはプロパティの設定で必須設定項目が設定されていない場合や、設定された値に問題がある場合表示されます。

オブジェクトごとに、！ マークのチェック対象となるプロパティ項目に付いて説明します。

①アニメーション
　リンクメモリ未指定

②リングメーター
　リンクメモリ未指定

③ボタン(共通)
　・インタロック指定時
　　インターロックメモリ未指定
　　インターロック比較値未指定
　・メモリによる背景制御指定時
　　背景制御メモリ未指定
　・メモリによるラベル制御指定時
　　ラベルメモリ未指定
　・メモリによるラベル色制御指定時
　　ラベル色メモリ未指定

④ボタン(オルタネート、モーメンタリ、セレクト)
　リンクメモリ未指定

⑤ボタン(セレクト)
　マスク、セット値未指定

⑦ボタン(カスタム、マルチアクション)
　メモリによる表示制御指定、表示制御メモリ未指定

⑧ボタン(マルチアクション)
　マルチアクション指定の中に不正がある場合(コンバート時、変換できないものがあると発生)

⑨カウンター
　リンクメモリ未指定
　キーパッド指定でキーパッド配置先が 0
　ビットマップフォント指定でビットマップフォント種類未指定

⑩ビットマップ
　表示ビットマップ番号未指定

⑪ランプ
　リンクメモリ未指定
　比較値未指定
　枠線幅が角 R 以上

⑫メーター
　リンクメモリ未指定

⑬マルチランプ
　リンクメモリ未指定
　状態未登録
　枠線幅が角 R 以上

⑭監視オブジェクト
　リンクメモリ未指定
　比較条件無効化しないときに比較値未指定

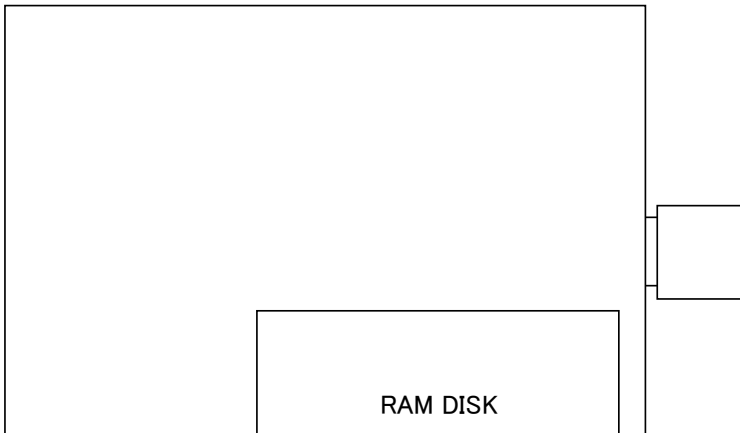
⑮ビットマップメーター
ビットマップ未指定
回転確度値未指定
X オフセット未指定
Y オフセット未指定
⑯角丸ボックス
枠線幅が角 R 以上
⑰スライダー
リンクメモリ未指定
最大、最小値未指定
⑱テキストボックス
リンクメモリ未指定
⑲トレンドエリア
開始、終了位置指定メモリ未指定

Q.GOP動作中に画像データのビットマップを入替えたい

GOP のビットマップデータはファイルとして管理されています。
またビットマップ番号とデータファイルは以下のようなルールでリンクされています。
ビットマップ番号:XXX　⇒GBXXX.GB
例えばビットマップ 0 を指定していた場合、GB000.GB というファイルが読込まれます。

GOP の動作中にビットマップを入替える(データ書き込み時にあらかじめ登録していないビットマップ)に入替える場合、以下の 2 つの動作が必要になります。
①動作中の GOP にビットマップファイルを送信する。
②入替えたいビットマップ番号のリンクを、送信したビットマップファイルに変更する。

①について、GOP にファイルを送信する方法として以下があります。
a)USB メモリを使用する。
b)シリアルポート経由のファイル転送コマンド(UPLOAD)を使用する。
c)イーサネットユニットを使用し TFTP を使用する。



②については AILIAS_NAME マクロコマンドを使用することでビットマップ番号とビットマップファイルのリンクを切替えることが出来ます。
また、動的に表示ビットマップを切替えることが前提の場合、キャンバスオブジェクトに LOAD_IMAGE マクロコマンドで描画を行うことも可能です。LOAD_IMAGE を使用する場合 JPEG 形式や WindowsBitmap(BMP)形式の画像データを使用することが出来ます。
※AILIAS_NAME では置き換え先のファイル形式はもとのデータと同じ形式である必要があるため GOP のビットマップ(GB)形式である必要があります。
※JPEG 形式や BMP 形式から GB 形式に変換を行う場合、CONVERT_BMP マクロコマンドを使用し、GOP 内部で変換を行うことが出来ます。(ホスト側であらかじめ変換しておいてもよいです)

Q.GOPのUSBメモリ機能を使用しホスト側のデータをUSBメモリに書き込む

1. 概要

GOP に直接上記の機能はないため、画面設計時ホストから送られてきたデータを USB メモリに書き込むマクロを作成します。
まずコマンドを受け取るメモリ(コマンドメモリ)とデータを受け取るメモリ領域(転送領域)を用意します。
コマンドメモリを監視オブジェクトにリンクし書き込まれた値により、ファイルのオープン・クローズおよび転送領域のデータのファイルへの書き込みを行うようにします。

ホスト側は上記のメモリに以下のようなシーケンスで値を書き込みます。
①コマンドメモリにファイルオープンのコマンドを送信
②転送領域に書き込みデータを書き込み
③コマンドメモリにファイル書き込みのコマンドを送信
④必要な数だけ②③を繰り返す
⑤コマンドメモリにファイルクローズのコマンドを送信

2. 実装例

書き込むデータをテキスト形式とした場合の実装例を以下に示します。

GOP 側メモリ

コマンドメモリ	b0010
転送領域	T0100 80 文字

監視オブジェクト

リンクメモリ b0010
比較条件 無効化

マクロ

```
FUNC _監視 0_動作
        ;ファイル番号記憶
        ADIM fno W
        ;USB メモリ状態の確認
        IF ENABLE_USBMEM=#1 THEN
                ;コマンドごとに処理を振り分け
                ;CMD=1    書き込みモードでオープン
                ;                 (ファイル名 TEST.TXT)
                ;CMD=4    DATA の内容をファイルに書き出し
                ;CMD=0    ファイルのクローズ
                IF CMD=#1 THEN
                        IF fno=#0 THEN
                                OPENFILE fno 1 $2:TEST.TXT
                        ENDIF
```

```
                ELSEIF CMD=#4 THEN
                        IF fno!=#0 THEN
                                WRITELINE fno DATA
                        ENDIF
                ELSEIF CMD=#0 THEN
                        CLOSEFILE fno
                        MOV fno #0
                ENDIF
        ENDIF
END_FUNC
```

３．使用例

ホストから以下のようなコマンドを送信します。(stx,etx の表記は省略)

```
WD b0010 1
WD T0100 123,456,789
WD b0010 4
WD T0100 あいうえおかきくけこ
WD b0010 4
WD b0010 0
```

(※赤文字部がデータ)

すると USB メモリ上に"TEST.TXT"の名称でファイルが作成され以下のような内容が記録されます。

```
123,456,789
あいうえおかきくけこ
```