METALコマンドリファレンス

Version 0.00

はじめに

このドキュメントは当初METAL v1.3のオンラインドキュメントの日本語訳として書き始められましたが、途中で個人的に気がついたことを書き込んでいくうちに現在の形になったものです。関数とコマンドの区別、また、それらの分類もオンラインドキュメントのものとは異なります。あくまでも原典は付属のオンラインドキュメントであるので、本ドキュメントの記述に不信を感じられた場合は、オンラインドキュメントを参照して下さい。そして、このドキュメントに誤りがあるのを発見した場合は saito@dep.sme.co.jp まで知らせていただければ幸いです。

齋藤宏治（2000 年 5 月 28 日）

METALとは

METAL とは Galactic Dreams Software（GDS）により作成された Macintosh 用の BASIC 言語です。いくつかの拡張が施されており、グラフィックス、サウンド、スピーチ、数学関係の豊富なコマンドが存在します。また、コンパイラも内蔵しており、スタンドアローンなアプリケーションも作成可能ですが、いまのところ正しく動作してくれません（英語環境ではきちんと動作するようです)。

METAL v1.3 コマンドリファレンス

注:大部分の例は Metal Help のものを用いました。

コメント

rem 文を用いると、プログラムに注釈（コメント）を書き込むことができる。

書式: rem comments...
省略形は'（シングルクォート）。

例:
```
rem this is comment.
' shorter version.
```

コメントは実行されない（プログラムの動作に影響を与えない）。

変数

数値変数
数値を格納する変数名は、アルファベットで始まる英数字。

例:
```
age
radius2
```

文字列変数
文字列を格納する変数名は最後に$を付ける。

例:
```
name$
address2$
```

配列変数

dim 文にて宣言する。dim 文で宣言した配列名に丸括弧をつけてアクセスする。

例:

```
dim a(3) :' Allocate array called "a" of three elements
input "Input two numbers";a(1);a(2)
' Exchange a(1) and a(2)
a(3)=a(1)
a(1)=a(2)
a(2)=a(3)
? a(1),a(2)
```

演算子

mod 演算子

書式: number1 mod number2

mod 演算子は number1 を number2 で割った余りを返す。

例:

```
? 17 mod 5 :' Outputs 2
? 4 mod 3:' Outputs 1
? 2 mod 2:' Outputs 0
```

注:

負の数を扱うときは注意が必要。いわゆる数学的な「余り」とは異なった結果を返す。

例:

```
? -10 mod 4 :'Outputs -2
? -10 mod -4 :' Outputs -2
? 10 mod -4 :' Outputs 2
```

and 演算子

書式: number1 and number2

number1 と number2 のビット毎の AND をとる。整数で無く実数を与えた場合は整数に変換後 AND をとる。

not 演算子

書式:not number

number のビット毎の NOT をとる。

or 演算子

書式: number1 or number2

number1 と number2 のビット毎の OR をとる。整数で無く実数を与えた場合は整数に変換してから OR をとる。

xor 演算子

書式: number1 xor number2

number1 と number2 のビット毎の XOR をとる。整数で無く実数を与えた場合は整数に変換してから XOR をとる。

流れ制御

if 文

書式: if condition then expressions
または
if condition then
    expressions
end if
または、
if condition then
    expressions
else
    expressions
end if

condition が成立するならば、then 以降を実行する。then 以降の文が複数行になる場合は endif が必要。条件が成立しない場合、else がある場合は else から end if までの文を実行する。条件が成立せず、かつ else が無い場合は何もしない。

例:

```
input "How old are you?";age
if age<5 then
       ? "It's almost impossible!"
       input "Are you sure you are less than 5 (Y/N)?";YN$
       if yn$="Y" then ? "It's very very nice to meet you."
else
       ? "Nice to meet you."
end if
```

for 文

書式: for variable=expression1 to expression2 step expression3
　　expressions
　　next variable

注: step expression3 はあっても無くても良い。

variable を expression1 から expression2 まで変化させつつ、next までに書かれた文を実行する。step が無い場合 variable は１つづつ増加する。step がある場合は step で指定された分だけ増加する。

例:

```
for i=1 to 10 step 2
　　? i
　　a=a+i
next i
? "Sum of all odd numbers between 1 to 10 is ";a
```

end 文

プログラムを終了するときに用いる。

例:

```
input "Select direction (N,S,W,E):";way$
if way$="N" then ? "You die!": end
```

参考: stop

gosub 文

書式: gosub label

label から始まる行をサブルーチンとして実行する。サブルーチンからは return で戻る。

例:

```
? "I'm main program"
? "...Jumping to subprogram..."
gosub subprog
? "We are in main program."
end

subprog:
? "I'm subprogram.."
? "Going back..."
return
```

goto 文

書式: goto label

label の部分へジャンプする。gosub と異なるのは、return で戻ってこれないところ。

例:

```
again:
? "Select one:"
? "1 - go to Ol' Creek."
? "2 - go to Scott's mine."
input n
if n=1 then goto creek
if n=2 then goto mine
goto again
end

creek:
? "creek..."
end

mine:
? "mine..."
end
```

pop 文

？？？

stop 文

？？？止まるけど、使い道が良く分からない。

文字列処理関数

注:METAL の文字列処理関数が日本語に対応しているかどうかは未確認。

chr$関数

書式: chr$(number)

chr$関数は与えられた値をアスキーコードとする文字を返す。

例:

```
? chr$(65) : 'Outputs A
? chr$(66) : 'Outputs B
? chr$(49) : 'Outputs 1
```

asc 関数

書式: asc(string)

asc 関数は与えられた文字列の最初の１文字のアスキーコードを返す。

例:

```
? asc("A"): 'Outputs 65
? asc("Aaron"): 'Outputs 65
? asc("B"): Outputs 66
? asc("1"): Outputs 49
```

len 関数

書式: len(string)

len 関数は与えられた文字列の長さを返す。

例:

```
? len("METAL"): 'Outputs 5
```

left$関数

書式: left$(string,number)

left$関数は与えられた文字列の左から、与えられた数値分だけの文字列を返す。

例:

```
? left$("METALanguage",4): 'Outputs "META"
```

right$関数

書式: right$(string,number)

right$関数は与えられた文字列の右側から、与えられた数値分だけの文字列を返す。

例:

```
? right$("METALanguage",3): 'Outputs "age"
```

mid$関数

書式: mid$(string,number1,number2)

mid$関数は与えられた文字列の左から、number1 文字目より number2 文字を取り出して返す。

例:

```
? mid$("METALanguage",5,8):'Outputs "Language"
```

str$関数

書式: str$(number)

str$関数は与えられた数値を文字列化して返す。

例:

```
a$=str$(2.2)
? a$+"22": 'Outputs 2.222
```

space$関数

書式: space$(number)

space$関数は与えられた数値分のスペース文字を返す。

例:

```
for x=1 to 10
       ? space$(x);"x"
next x
```

string$関数

書式: string$(number,string)

string$関数は与えられた文字列の先頭文字を、与えられた数値分繰り替えした文字列を返す。

例:

```
? "Five starts..."
? string$(5,"*")
```

val 関数

書式: val(string)

val 関数は与えられた文字列を数値に変換して、その値を返す。数値に変換できない文字列の場合は 0 を返す。

例:

```
a$="3.2"
b=val(a$)
? b+2 :'Outputs 5.2
```

数学関数

abs 関数

書式: abs(number)

abs 関数は、与えられた数値の絶対値を返す。

例:

```
? abs(-3.7) :'Outputs 3.7
? abs(3.5) :'Outputs 3.5
```

atn 関数

書式: atn(number)

atn 関数は、与えられた数値のアークタンジェントを返す。

例:

```
? atn(1) :' Outputs 0.785398=pi/4
```

cos 関数

書式: cos(number)

cos 関数は、与えられた数値（ラジアン）のコサインを返す。

exp 関数

書式: exp(number)

exp 関数は自然対数のベキを計算する。

int 関数

書式: int(number)

int 関数は、与えられた数値の整数部分を返す。

例:

```
? int(2.3456) :' Outputs 2
? int(-1.234) :' Outputs -1
```

log 関数

書式: log(number)

log 関数は与えられた数値のログを返す。

rnd 関数

書式: rnd(number)　または　rnd

乱数を発生する。rnd 関数に何も数値を与えないで用いると、０から１までの実数を返す。

例:

```
? rnd :' Output random number 0-1
```

　rnd 関数にマイナスの値を与えると、その与えた値により決まる実数を返す。例えば、最初の rnd(-6)と２度目の rnd(-6)は同じ値を返す。

　プラスの値を与えるた場合、その値は無視される。つまり、なにも数値を与えない場合と同じく、０から１までの実数を返す。

　実際にプログラム中で用いる場合、０から n までの整数が必要な時が良くあるので、念のため、ここにプログラム例を書いておく。

例:0から5までの乱数(整数)を求めるとき。

```
? int(rnd*6)
```

rnd*6 は、0から5までの実数を返すことに注意。

sgn 関数

書式: sng(number)

sng 関数は与えられた数値が負なら-1 を、正なら+1 を、0 なら 0 を返す。

例:

```
? sng(-5) :'Outputs -1
? sgn(0) :' Outputs 0
? sgn(8.3) :' Outputs 1
```

sin 関数

書式: sin(number)

sin 関数は与えられた数値のサインを返す。

sqr 関数

書式: sqr(number)

sqr 関数は与えられた数値の平方根を返す。

例:

```
? sqr(16) :' Outputs 4
? sqr(2) :' Outputs 1.414214
```

tan 関数

書式: tan(number)

tan 関数は与えられた数値のタンジェントを返す。

データの読み込み

date 文でデータを記述し、read 文で読む。

例:

```
read numDays : 'Read no. of days
? "One week has ";numDays;" days." :'Print it
? "There are:"
for x=1 to 7
       read day$:? day$;" "; :'Read days and print
next x
?
data 7,sun,mon,tue,wen,thu,fri,sat
```

read 文でデータを読み出す度に、次に読み出すべきデータの位置が進む（上の例だと、初めは 7、次は sun というように）。

状況によっては、一度読んでしまったデータを再び読み出したい場合もある。そういう場合に使うのが restore 文である。

書式: restore　または　restore label

ラベルを指定しない場合は、ソースコードに書かれた最初の data 文から、ラベルが指定されている場合は、ラベル以降の最初の data 文よりデータが読み出される。

ラベルを指定して、かつ、そのラベル以降に data 文が存在しない場合は、ラベル無しの restore 文と同じくソースコードに書かれた最初の data 文より読み出されるようになる。

例:

```
read a$,b$
print a$,b$
restore
read a$
print a$

animals:
print "---end---"
data cat,dog,bird
```

実行結果

```
cat     dog
cat
---end---
```

ユーザー定義関数

def fn func_name(parameter1,parameter2,...)
で定義し、
fn func_name で使用する。

例:

```
def fn sq(v)=v*v
? "Square of 4 is ";fn sq(4)
```

注意：def fn で定義される関数で用いられている変数（上の例だと v）は局所的なものである。

例:

```
def fn sq(v)=v*v

v=2
? fn sq(4)
? v
```

実行結果

```
16
2
```

つまり、fn sq を実行しても v の値は変化していない。

システム関連命令

button 文

button 文は、マウスボタンが押されていたら true(=-1)を返し、押されていなかったら、false(=0)を返す。

calloc 文

書式: calloc(size)

calloc 文は、与えられた数値バイト分のメモリブロックを確保し、そのアドレスを返す。確保した領域は0で埋められる。確保した領域が不要になったら、free 文を用いて領域解放しなければならない。

例:

```
start=calloc(1000) :' Allocates 1000 byte of memory
poke start,15 :' Writes number 15 in it
poke start+1,6 :' Writes number 6 in it
? peek (start) :' Outputs 15
? peek (start+1) :' Outputs 6
free start :' Releases memory
```

clear 文

clear 文は全ての変数を初期化する。文字列変数は何も代入されていない状態になり、数値変数には0が代入される。配列も初期化される。

例:

```
a=3
b$="hello"
? a,b$ :' Outputs 3   hello
clear
? a,b$ :' Outputs 0
```

cls 文

cls 文はカレントスクリーンを消去する。

deek 文

書式: deek(address)

deek はダブル peek の意味で、指定されたアドレスよりワード（２バイト分読み出す）

注:deek で読み出すアドレスに奇数バイト目を指定していけない。

例:

```
ptr=calloc(10)
poke ptr,1
poke ptr+1,2
poke ptr+2,3
poke ptr+3,4
? deek(ptr),1*256+2
? deek(ptr+2),3*256+4
free ptr
```

例:画面の縦方向の解像度を求める（8100/80AV では正しく表示された）

```
? "Your screen has:"; deek(3106); " lines."
```

doke 文

書式: doke address,data

doke 文はダブル poke の意味らしく指定したアドレスに指定したデータを２バイトのデータとして書き込みます。

例:

```
ptr=calloc(10)
doke ptr,258 :' 258=0x0102
? peek(ptr)
? peek(ptr+1)
free ptr
```

free 文

書式: free address

指定したアドレスで始まるメモリブロックの領域解放を行う。

注:使用例は、doke の例を参照。

fre 文

未実装。

fread 文

書式: fread file-pointer,variable1 [,variable2...]

fread 文は指定されたファイルポインタからデータを読み出し、指定された変数に代入する。

例:

```
? "Writing data..."
out=open file ("mu_data")
fwrite out, "Hi!", "1", "2", "3"
close file out
? "Reading data..."
in=open file ("my_data")
fread in, text$, n1,n2,n3
close file in
? "This is what I've read: ";text$,n1,n2,n3
```

fwrite 文

書式: fwrite file-pointer, variable1 [,variable2...]

fwrite 文は指定されたファイルポインタに指定された変数（または定数）に代入されているデータを書き出す。

例:→fread の例を見よ。

getmousex 関数

getmouse 関数はマウスの x 座標を返す。マニュアルでは述べていないが、この座標は指定した出力画面の座標系となるので、注意が必要。オフスクリーンの場合は、デスクトップの座標系となる。

例:

```
? "Your mouse coordinates are: X:";getmousex;
? " Y:"; getmousey
```

getmousey 関数

y 座標を返す以外は、getmousex 関数と同じ。

home 文

home 文はカレントスクリーンをクリアする。これは AppleSoft BASIC とのコンパチビリティの為に実装されている。

inkey$ 関数

inkey$関数はそのとき押されているキーの文字を返す。何も押されていないときは、空の文字列を返す。

例:

```
? "Please press key 'B'"
again: if inkey$<>"B" then goto again
? "Thank you..."
```

input 文

書式: input string-constant,variable1 [,variable2...]

変数にユーザーからの入力を代入する。

例:

```
input "give me a number";x
input "give me a name:";name$
? "type anything:"
input trash$
? x,name$
```

leek 文

書式: leek(address)

long peek。与えられたアドレスより連続する４バイトを読み、それを返す。

例:

```
ptr=calloc(10)
poke ptr,1
poke ptr+1,2
poke ptr+2,3
poke ptr+3,4
poke ptr+4,5
? leek(ptr), (((1*256)+2)*256+3)*256+4
loke ptr+4,1234567890
? leek(ptr+4)
free ptr
```

locate 文

書式: locate y,x

(x、y)にカーソルを設定する。x および y は 1 から 25 の間でなければならない。

例:

```
locate 5,6 :' Set cursor to row 5, column 6
locate 10 :' Set cursor to row 10
```

loke 文

書式: loke address,data

ロング poke。指定されたアドレスを先頭アドレスとして、data を４バイト分書き込む

例:

```
ptr=calloc(10)
loke ptr, (((1*256)+2)*256+3)*256+4
? peek(ptr),peek(ptr+1),peek(ptr+2),peek(ptr+3)
free ptr
```

malloc 文

書式: malloc(size)

malloc 文は、指定されたバイト数分のメモリ領域を確保し、そのアドレスを返す。calloc 文と異なるのは、確保したメモリ領域を０でクリアしない点（確保したメモリ領域の中には、すでに０が入っている場合もある事に注意)。

例:

```
ptr=malloc(10)
?peek(ptr)
loke ptr, (((1*256)+2)*256+3)*256+4
? peek(ptr),peek(ptr+1),peek(ptr+2),peek(ptr+3)
free ptr
```

open file 文

書式: open file (file_name)

open file 文は、指定されたファイル名のファイルを読み込み、書き込み用にオープンし、そのファイルポインタを返す（ファイルポインタとファイルを関連付けする)。ファイルを読み書きする場合は、このファイルポインタを指定する。

例:→fread の例を見よ。

close file 文

書式: close file file-pointer

close file 文は､指定されたファイルポインタに関連つけられたファイルをクローズする。fwrite 文などで、ファイルに書き込んだ場合は、close file 文でクローズしないと、結果が反映しない。

例:→fread の例を見よ。

peek 文

書式: peek(number)

peek 文は指定されたアドレスに格納されている１バイトを読み出す。

例:

```
? "Global volume is set to: ";peek(608)
```

poke 文

書式: poke address,data

poke 文はアドレスに、data の下位８ビットを書き込む。

例:

```
ptr=malloc(10)
poke ptr,255
? peek(ptr)
poke ptr,256
? peek(ptr)
free ptr
```

pos 文

未実装

print 文

書式: print {expression...}

print 文は、指定された文字列、数値をコンソールに出力する。省略形は"?"（ダブルクォーテーション不要）。何も指定しないと改行のみを行う。出力するものを、カンマでつなげると、いくつか空白をおいて出力される。セミコロンでつなげると、つめて出力される。最後にセミコロンをつけると改行を行わない。

例:

```
? "I'm shorter..."
print "I'm longer..."
?
? "Tabbed output:",1,2,3
? "Close output:";1;2;3
? "expressions:",1+4*3+2
```

timer 関数

timer 関数は 1970 年 1 月 1 日の午前 0 時 0 分 0 秒よりの、経過時間を秒数で返す。この秒数をそのまま使うことはほとんど無く、実際は、測定開始時に timer 関数を呼び、それとの差を取る事によって、経過時間を求める事が多い。

例:

```
x=timer
for y=3 to 0 step -1
       ? y
              again: if timer-x<=1 then goto again
       x=timer
next y
? "Go!!!"
```

wait 文

未実装

wait button 文

wait button 文はマウスボタンが押されるまで待つ。

例:

```
? "Please click mouse button to proceed..."
wait button
? "OK."
```

wait button up 文

wait button up 文はマウスボタンが離されるまで待つ。

例:

```
? "Please click mouse button"
wait button
? "Please release mouse button"
wait button up
? "Thank you!"
```

wait key 文

wait key 文は何かキーが押されるまで待つ。

注:コマンドキー、コントロールキー、オプションキー、シフトキー、キャップスロックキーにも反応する。

例:

```
? "Press any key to continue..."
wait key
? "OK."
```

wait key up 文

wait key up 文は全てのキーが押されなくなるまで待つ。

例:

```
? "Hold any key..."
wait key
? "Please release the key..."
wait key up
? "Thank you!"
```

グラフィック関連命令

backcolor 文

書式: backcolor red,green,blue

カレントスクリーンの背景色を指定する。それぞれの色の成分は、0 から 65535 まで。0,0,0 が黒で、65535,65535,65535 が白。

例:

```
backcolor 65535,0,0 :' Sets background color to red
backcolor 0,0,65535 :' Sets background color to blue
backcolor 65535,65535,0 :' Sets background color to yellow
' Sets background color to sky-blue
backcolor 30000,30000,65535
```

circle 文

書式: circle x,y,radius

(x,y)を中心とする半径 radius の円をカレントスクリーンに描く。色は forecolor で指定された色になる。

例:

```
circle 100,100,20 :' Draws circle at (100,100), radius=20
```

copyrect 文

書式: copyrect x0,y0,x1,y1, x2,y2,x3,y3, mode, source,dest

また、マスク付きの場合は。

copyrect x0,y0,x1,y1,mx0,my0,mx1,my1,x2,y2,x3,y3,source,mask,dest

注１:x0,y0…等の引き数は全て数値である。

注２:mode,source,dest は無くてもよい

copyrect 関数は、(x0,y0)-(x1,y1)を対角線とする長方形の内部を、(x2,y2)-(x3,y3)を対角線とする長方形の内部へとコピーする。

mode を指定することによって、コピー仕方を指定できる。mode に指定可能な値を後で述べる TransferMode 一覧に載せておく。

異なるスクリーン間での転送を行う場合には、source と dest を指定する。コンソールスクリーンを指定する場合は、0 を指定すればよい。

例:

```
' Copies and resizes part of screen
copyrect 10,10,100,100,80,80,190,190
```

また、マスク付きの場合は、(mx0,my0)-(mx1,my1)を対角線とする長方形の内部によりマスクされる。mask オプションが付いている場合は、mask で指定されたスクリーンの情報がマスク情報として使われる。

注:コピー元とコピー先のサイズが異なる場合は、コピー先のサイズに合うようにリサイズされる。

例:

```
' Allocate temporary screen
temp=init screen (0,0,100,100)
set screen to temp :' Set screen to temporary screen
cls : forecolor 655354,65535,65535 :' Cklear, white color
rect 50,50,100,100 :' Draw white square
set screen to console
' Now copy par of console using mask
copyrect masked 10,10,100,100, 0,0,100,100, 80,80,180,180, 0,temp,0
```

注:フォントの関係上 copyrect 文の引数が途中で改行しているが、実際入力する場合は改行してはならない。

TransferMode 一覧:

0 = srcCopy(normal)
1 = srcOr
2 = srcXor
3 = srcBic
4 = notScrCopy
5 = notSrcOr
6 = notSrcXor
7 = notSrcBic
8 = patCopy
9 = patOr
10=patXor
11=patBic
12=notPatCopy
13=notPatOr
14=notPatXor
15=notPatBic
32=blend
33=addPin
34=addOver
35=subPin
37=addMax
38=subOver
39=adMin
64=ditherCopy

ellipse 文

書式: ellipse x,y,radius1,radius2

ellipse 文は(x,y)を中心として、radius1 と radius2 を持つ楕円を描く。

例:

```
ellipse 100,100,20,10
```

fcircle 文

書式: fcircle x,y,radius

fcircle 文は(x,y)を中心として、radius を半径とする円を、forecolor で指定された色で塗りつぶす。

例:

```
fcircle 100,100,20 :' Fills circle at (100,100), radius=20
```

fellipse 文

書式: fellipse x,.y,radius1,radius2

fellipse 文は、(x,y)を中心として、radius1,radius2 を半径とする楕円を forecolor で指定された色で塗りつぶす。

forecolor 文

書式: forecolor red,green,blue

注:red,green,blue は 0 から 65535 までの数値

forecolor 文はカレントスクリーンのフォアカラーを red,green,blue で指定された色に設定する。0,0,0 が黒で 65535,65535,65535 が白。

例:→backcolor の例を見よ。

get pict size 文

書式: get pict size string,var1,var2

get pict size 文は指定された PICT ファイルの横幅と高さを、var1、var2 に代入する。

例:

```
get pict size "myPictureFile",picWidth,picHeight
? "The width of the picture is: ";picWidth
? "The height of the picture is: ";picHeight
```

注:この例を実行するためには、カレントディレクトリに、myPictureFile という名の PICT ファイルが必要。

hidecursor 文

hidecursor 文はマウスカーソルを非表示にする。

init screen 文

書式: init screen (x0,y0,x1,y1)

init screen 文は、(x0,y0)-(x1,y1)を対角線とする仮想スクリーンを作成し、そのアドレスを返す。使用しなくなった仮想スクリーンは、kill screen で
解放しなければならない。また、仮想スクリーンを使用する前に、cls を実行して仮想画面をクリアすべきである。

例:

```
virtu=init screen (0,0,100,100)
set screen to virtu
cls
text 20,20, "virtual..."
set screen to console
for x=1 to 100
      copyrect 0,0,100,100, 0+x,0,100+x,100, 0,virtu,0
next x
kill screen virtu
```

kill screen 文

書式: kill screen screen-address

注:screen-address に指定する値は、init screen が返した値のみ。

kill screen は仮想スクリーンを解放する。

例:→init screen の例をを見よ。

line 文

書式: line x0,y0,x1,y1

line 文はカレントスクリーンに(x0,y0)から(x1,y1)の直線をフォアカラーで描く。line 文を実行後、カレントポジションは(x1,y1)になる。

例:

```
line 10,10,100,100 :' Draws a line from 10,10 to 100,100
```

lineto 文

書式: lineto x,y

lineto 文は、カレントポジションから(x,y)までの直線をフォアカラーで描く。lineto 文を実行後、カレントポジションは(x,y)になる。

例:

```
moveto 10,10 :' Set current position 10,10
lineto 100,100 :' Draws line to 100,100
```

loadpict 文

書式: loadpict string,x0,y0,x1,y1

注:x0,y0,x1,y1 はオプショナルなので、あっても無くても良い。

loadpict 文は string で指定された PICT フォーマットの画像ファイルをカレントスクリーンに読み込む。x0,y0,x1,x2 が指定されているならば、(x0,y0)-(x1,y1)を対角線とする長方形の内部に画像を読み込む。画像ファイルと、指定した長方形の大きさが異なる場合は、指定した長方形に合うように拡大縮小される。

例:

```
loadpict "myPictFile",10,10,100,100
```

注:この例を正しく動作させるためには、カレントディレクトリに myPictFile という PICT ファイルが存在しなければならない。

moveto 文

書式: moveto x,y

moveto 文はカレントポジションを(x,y)に設定する。

例:→lineto の例を見よ。

plot 文

書式: plot x,y

plot 文はカレントスクリーンの(x,y)にフォアカラーで点を描く。

例:

```
cls
for x=0 to 100 step 3
   plot x,10
next x
```

rect 文

書式: rect x0,y0,x1,y1

rect 文は、カレントスクリーン上に(x0,y0)-(x1,y1)を対角線とする、長方形を描き、その内部をフォアカラーで塗りつぶす。

例:

```
rect 10,10,200,100 :' Fills rectangle
```

resize console 文

書式: resize console x0,y0,x1,y1

resize console 文はコンソールウィンドーをリサイズする。画面上の(x0,y0)で指定された位置がコンソールウィンドーの左上となり、(x1,y1)で指定された位置が右下になる。

例:

```
resize console 40,30,400,245
```

scroll 文

書式: scroll x0,y0,x1,y1,dirX,dirY

scroll 文はカレントスクリーン上の(x0,y0)-(x1,y1)を対角線とする長方形を(dirX,dirY)方向にスクロールする。

例:

```
'Smooth scroll, one by oen point...
for i=1 to 100
       scroll 0,0,200,200,-1,0 :' scrolls part of console to the left
next i
```

set screen to 文

書式: set screen to address

注:address は init screen が返した値のみ使用可能。

set screen to 文は指定した仮想画面をカレントのスクリーンとする。

例:→init screen の例を見よ。

set screen to console 文

書式: set screen to console

set screen to console 文はコンソールをカレントスクリーンに設定する。

例:→init screen の例を見よ。

showcursor 文

書式: showcursor

showcursor 文はマウスカーソルを表示するよう設定する。

text 文

書式: text x,y,string

text 文はカレントスクリーンの(x,y)に指定した文字列を描く。

例:

```
text 20,200,"Text!"
```

textface 文

書式: textface number
textface 文は text 文の描画スタイルを設定する。

描画スタイルは次の通りである：

0=normal
1=bold
2=italic
4=underline
8=outline
16=shadow
32=condense
64=extend

例:

```
textface 2+4 :' Set text face italic and underline
text 20,200,"italic and underline"
```

textfont 文

書式: textfont number

textfont 文は text 文で用いるフォントを指定する。

各フォントに割り当てられている数値は次の通りである：

0=system font
1=application font
2=New York
3=Geneva
4=Monaco
5=Venice
6=London
7=Athens
8=San Francisco
9=Toronto
11=Cairo
12=Los Angeles
20=Times
21=Helvetica
22=Courier
23=Symbol
24=Mobile

例:

```
textfont 22
text 20,200,"Courier"
```

textmode 文

書式: textmode number

textmode 文は、text 文が描画するさいに、どの転送モードを用いるかを指定する。転送モードの数値一覧は、copyrect の TransferMode 一覧を見よ。

例:

```
textmode 0
text 20,20,"This text overwrites other text..."
```

textsize 文

書式: textsize number

textsize 文は text 文で描画するときの文字のサイズを設定する。

例:

```
textsize 30
text 20,200,"This is BIG!"
```

サウンド関連命令

beep 文

書式: beep

beep 文はシステムの警告音を鳴らす。

play aiff 文

書式: play aiff string

play aiff 文は指定された aiff ファイルを再生する。再生は非同期に行われる。これは、サウンドを再生しながら、プログラムが実行しつづけるという意味である。

例:

```
play aiff "myAiffFile"
```

say 文

書式: say string

say 文は与えられた文字列を読み上げる。

注:スピーチマネージャが必要。

例:

```
say "Dave, I'm afraid!"
```

リファレンス

'→rem 文
?→print 文
end if → if 文
else → if 文
next → for 文
return → gosub 文
step → for 文
to → for 文