

高機能マルチファイルスクリーンエディタ

MIFES®

for *Windows*
Ver. 7.0

マクロマニュアル

MEGASOFT®

おことわり

- お客様が作成されたマクロについて、当社に起因すると認められる不具合以外のサポートはしていません。
お客様がマクロを作成される上で、製品の不具合に起因しない動作不良の修正作業や、お客様がマクロを作成する上でのアイデアなどの提供および助言などには、当社サポートセンターは対応いたしかねますので、ご了承ください。
- 本書の内容の全部または一部を、当社に無断で転載あるいは複製することは、法令に別段の定めがある場合を除き、固く禁じられています。
- 本書の内容は、本製品の改良のため、将来予告なく変更することがあります。
- 本書の内容については万全を期して制作しておりますが、万一わかりにくい点や記載漏れなどお気付きの点がございましたら、当社サポートセンターまでご連絡ください（本書の内容と現実が異なるときは、現実が本書に優先します）。
- 本製品を使用したことによるお客様の損害、逸失利益、または第三者のいかなる請求につきましても、当社は一切その責任を負いかねますので、あらかじめご了承ください。
- 本製品をご使用になるには、別掲の「ソフトウェア使用許諾条項」にご同意いただくことが必要です。パッケージを開封されたときに、同条項へのご同意があったものとさせていただきますので、ご了承ください。

商標について

MIFES はメガソフト株式会社の登録商標です。
その他、製品名などは一般に各社の商標または登録商標です。

はじめに

このたびは、高機能マルチスクリーンエディタ『MIFES for Windows Ver.7.0』をお買い上げいただき、まことにありがとうございます。

本書は、MIFES for Windows Ver.7.0のマクロ言語『MIL/W 言語』でマクロを作成するためのマニュアルです。

プログラミングの初心者や、MIL/W 言語をはじめてご使用になる方にも、マクロを作成していただけるように心がけて制作しました。

マクロ作成の際にご参照ください。

本書の構成は次のとおりです。

第1章 マクロの概要

MIL/W 言語の概要をまとめました。

MIL/W 言語をはじめてお使いの方はここからお読みください。

第2章 チュートリアル

5つのマクロを例に、作成方法を記載しました。

はじめてMIL/W 言語をお使いの方は、チュートリアル1をご参照ください。

チュートリアル2～5は、実践的なマクロの作成方法を記載しています。必要に応じてご参照ください。

第3章 マクロの実行手順

マクロのコンパイルやライブラリ登録など、作成したマクロを実行するための機能を説明しています。

第4章 リファレンス

システム変数、システム関数を解説しています。

本書が、皆さまのマクロ作成のお役に立てれば幸いです。

も く じ

● はじめに	iii
● 目次	iv
製品名の表現について	vi
アイコンについて	vi

第1章

マクロの概要

● マクロ言語MIL/W について	2
● DOS 版マクロ言語MIL との違い	3
● 旧バージョンからの変更点と互換性	4
Ver.6.0 のMIL/W 言語からの変更点	4
Ver.5.0 以前との互換性	5
● マクロの作成手順	6
● MIL/W 言語の書式	7
ソースファイルの形式	7
MIL/W 言語の書式	7
構文の形式	8
実行文	12
式	16
演算式	17
● 変数と定数について	20
変数	20
定数	22
● 関数や変数の調べ方	25
処理内容から関数や変数を調べる	25
ヘルプを使う	27

第2章

チュートリアル

● チュートリアル 1 右寄せ	30
● チュートリアル 2 改行だけの行を削除	35
● チュートリアル 3 指定した行範囲の印刷	41
● チュートリアル 4 範囲指定内の文字列の置換	51
● チュートリアル 5 C 言語関数内の文字列の置換	60

第3章

マクロコマンドを実行する

● コンパイルする	66
1つのコマンドずつコンパイルする	66
ファイル全体をコンパイルする	67
● マクロコマンドを実行する	68
カレントマクロの実行	68
ライブラリを使う	68
ボタンやキー操作に割り当てて実行する	70
MIW.MAC について	71
マクロコマンドの中止	71
● その他	72
マクロモードについて	72
1行ずつマクロコマンドを実行する	73
ソースコードの取り出し	74
ユーザー変数の確認と変更	74
他のライブラリからの登録(マクロライブラリのマージ)	76

第4章

リファレンス

● システム変数	78
1. カーソル位置の情報	78
2. カレントウィンドウの情報	80
3. 以降に開くウィンドウに関する情報	84
4. ユーザー入力情報	86
5. その他の情報	89
● システム関数	94
1. 文字列操作関数	94
2. 文字列の挿入, 削除関数	106
3. ジャンプ, 移動, 検索, 置換関数	113
4. ユーザー入力関数	122
5. ファイル操作関数	129
6. その他の関数	143
● エラーメッセージ一覧	164
● 索引	165

製品名の表現について

MIFES と記載するときは、MIFES for Windows Ver. 7.0 を表します。

MS-DOS 版のMIFES はDOS 版MIFES と記載しています。MS-DOS 版のMIFES とはMIFES Ver.5.5 をさします。

アイコンについて

本書では以下のアイコンを使って説明しています。



意味：**メモ**

説明：使用上の補足事項を説明しています。



意味：**参照**

説明：参照するページを記載しています。



意味：**注意**

説明：使用にあたっての注意点を記載しています。



意味：**ワンポイントアドバイス**

説明：使用にあたってのワンポイントアドバイスを記載しています。



意味：**用語**

説明：使用上の用語について説明しています。

第1章 マクロの概要

この章ではマクロ言語 MIL/W の概要を記載しています。
マクロの作成手順や MIL/W 言語の書式を記載していますので、
マクロを作成する前に必ずお読みください。

目次

マクロ言語 MIL/W について	2	変数と定数について	20
DOS 版マクロ言語 MIL との違い ...	3	変数	20
旧バージョンからの変更点と互換性	4	定数	22
Ver.6.0 の MIL/W 言語からの変更点 ...	4	関数や変数の調べ方	25
Ver.5.0 以前との互換性	5	処理内容から関数や変数を調べる	25
マクロの作成手順	6	ヘルプを使う	27
MIL/W 言語の書式	7		
ソースファイルの形式	7		
MIL/W 言語の書式	7		
構文の形式	8		
実行文	12		
式	16		
演算式	17		

マクロ言語MIL/W について

マクロとキーボードマクロ

MIFESには、「マクロ」と「キーボードマクロ」があります。

マクロは、MIFES専用のマクロ言語MIL/W（以下MIL/W言語とします）を使って作成するプログラムのことをいいます。

本書は、マクロを作成するためのマニュアルです。

キーボードマクロは、MIFESが持っている機能を記憶し、記憶した順序どおりに繰り返し実行することができる機能です。

キーボードマクロについては、別冊子ユーザーズマニュアルまたはヘルプを参照してください。

マクロでできること

マクロでは汎用性のある専用の関数を使います。

関数の組み合わせにより、いろいろな条件を設定したり、MIFESにはない独自の機能を作成することができます。

MIFESの機能を組み合わせて実行するキーボードマクロでは実現できない機能も、マクロでは実現できるようになります。

MIL/W 言語について

MIL/W言語はコンパイル言語です。

MIL/W言語で作成したソースプログラムは、「コンパイル」することで実行可能になります。

MIL/W言語で作成した機能をマクロコマンドと呼び、キー操作やメニューに割り当てて実行することもできます。



「コンパイル」すると、MIL/W言語で作成したソースプログラムは中間コードに変換されず、MIFESは中間コードを内部のインタプリタで実行します。

旧バージョンのマクロについて

DOS版MIFESでマクロを作成されていた方は、次の「DOS版マクロ言語MILとの違い」をお読みください。

また、MIFES for Windows Ver.6.0でMIL/W言語を使ったマクロを作成されていた方は、「Ver.6.0からの変更点と互換性」をお読みください。

DOS 版マクロ言語 MIL との違い

MIL/W 言語は、DOS 版 MIFES のマクロ言語 MIL (以下 MIL 言語とします) とは互換性がありません。

制御構造と演算子はほぼ同じですが、システム関数などには異なる部分がたくさんあります。MIL 言語はエディタが本来持っている機能を組み合わせてマクロコマンドを作成することを基本としていました。それに対して、MIL/W 言語はマクロコマンド用に用意している専用のシステム関数を組み合わせてマクロコマンドを作成します。

MIL 言語から MIL/W 言語へ機能アップした点は次のとおりです。

- ・ 中間コードの最大サイズが 32 K バイト (MIL 言語は最大サイズ 4 K バイト)
- ・ 32 ビット化された変数
- ・ 1024 要素に拡張された配列変数
- ・ グローバル変数とローカル変数
- ・ ユーザー定義変数
- ・ システム変数への代入
- ・ マクロ定数
- ・ 文字列定数のメタ文字処理
- ・ 代入演算子、単項演算子 (- ! など)
- ・ 汎用性のあるシステム関数 (引数がより柔軟に)
- ・ 内蔵コンパイラ
- ・ 起動時の自動コンパイル
- ・ 中間コードへのソースプログラムの埋め込み
- ・ マクロコマンドのライブラリへの自動格納

旧バージョンからの変更点と互換性

MIL/W 言語は、どのバージョンで作成されたマクロコマンドでも動作するように、互換性を考慮したバージョンアップを行っています。

しかし、MIFES 自身の機能アップなどに伴い、作成時とは動作が変わることがあります。

以下の変更箇所をご確認の上、ご対応ください。

Ver.6.0 の MIL/W 言語からの変更点

MIL/W 言語は、以下の点が変わりました。

追加になったシステム関数

- exeurl() 指定した URL をブラウザで表示
- setenv() 環境変数の追加/変更/削除

削除されたシステム関数

- speak() 使用できなくなりました。
記述してあってもコンパイルエラーにはなりませんが、実行しても何も行きません。

拡張されたシステム関数

- breplace() 複数置換 置換範囲の指定を追加
- gsearch() グローバル検索
 - 検索文字列を 3 つに拡張
 - 検索の条件の変更
 - 一斉表示機能の追加
 - タイムスタンプ条件の追加
- itemlist() リストウィンドウの表示
リストウィンドウ機能追加に伴い引数、機能を拡張
- replace() 文字列の置換 置換範囲の指定を追加
- ribbon() ツールバー/ユーザー定義バー/多目的バーの表示
 - ガイドリボン関連ビット無効
 - 多目的バーの上部配置設定の追加
 - 名称変更によるマクロ定数名の変更
- readprofile() カスタマイズファイルの読み込み 読み込む情報の設定の変更
- search() 文字列の検索 / リストウィンドウの検索 検索する方向の拡張
- sprintf() 書式文字列の取得 書式制御文字列の追加
- tagjump() タグジャンプ / バックタグジャンプ リストウィンドウに対応

使用できなくなったマクロ定数

ASTAT_BACKUPFILE	ASTAT_READER	ASTAT_DBLREADER
SYS_WFILE	SYS_DRAGSEL	SYS_SCROLL
SYS_RMENU	SYS_CKEY	SYS_CIFDEF
SYS_ROTTRIBON		

拡張 / 変更になったシステム変数

@astat	@dstat
@ribbon	@sys_astat
@sys_dstat	@sys_stat

Ver.5.0 以前との互換性

切り貼り操作を含むマクロコマンドの互換性

Ver.6.0 から、範囲選択時にカーソル位置を含むかどうかを、環境設定で指定できるようになりました。そのため、@selmode を使って切り貼りを行うマクロコマンドに関しては、以下の記述を用いて「方向範囲選択」の状態を考慮したプログラミングが必要になります。

```
@sys_stat &= SYS_NEWSEL ; 「カーソル位置含まず」に設定
@sys_stat != SYS_NEWSEL ; 「カーソル位置も含む」に設定
```

なお、Ver.5.0 以前にコンパイルされたマクロコマンドを実行する際は、互換性を保つために、自動的にマクロ作成時と同じ「カーソル位置含まず」の状態で行われます（マクロコマンドを実行中の間だけ一時的に「カーソル位置含まず」の状態になります）。Ver.7.0 用のコンパイラでコンパイルされたマクロコマンドを実行する際にはこのような処置は行われません。そのため、「方向範囲選択」の状態を意識する必要があります。

また、旧バージョンで作成したマクロコマンドを変更するなどして、Ver.7.0 のマクロ言語コンパイラでコンパイルした場合にも、@sys_stat の値を操作する必要があります。

switch 構文の実行制御の拡張

switch ~ case ~ endsw 構文の実行制御には Ver.5.0 までと互換性のあるタイプと、C 言語の switch と互換性のあるタイプの、2 とおりのタイプがあります。

どちらのタイプでマクロコマンドを実行するかは、【マクロ(M)】-【マクロモード設定/コンパイル(M)】で指定します。

・旧タイプ

case 文以下の実行は、最初に break 文、case 文、または endsw 文が見つかるまで行います。つまり、case 文以下の実行途中で次の case 文を見つければ、そこで switch 構文を抜け出る仕様です。

・新タイプ

case 文以下の実行は最初に break 文または endsw 文が見つかるまで行います。つまり case 文以下の実行途中で次の case 文を見つけても、switch 構文を抜け出すことはありません。

Ver.5.0 から Ver.6.0 へのバージョンアップ時には次のシステム関数を追加 / 拡張しました。Ver.6.0 からの変更点とあわせてご確認ください。

●追加

calc()	findfile()	getfile()	getftime()	getstring()
gettime()	getwinpos()	readprofile()	reform()	sendmail()
setftime()	setstring()	setwinpos()		

●拡張

breplace()	chgcolor()	gsearch()	itemlist()	macro()
open()	outprinter()	replace()	ribbon()	search()

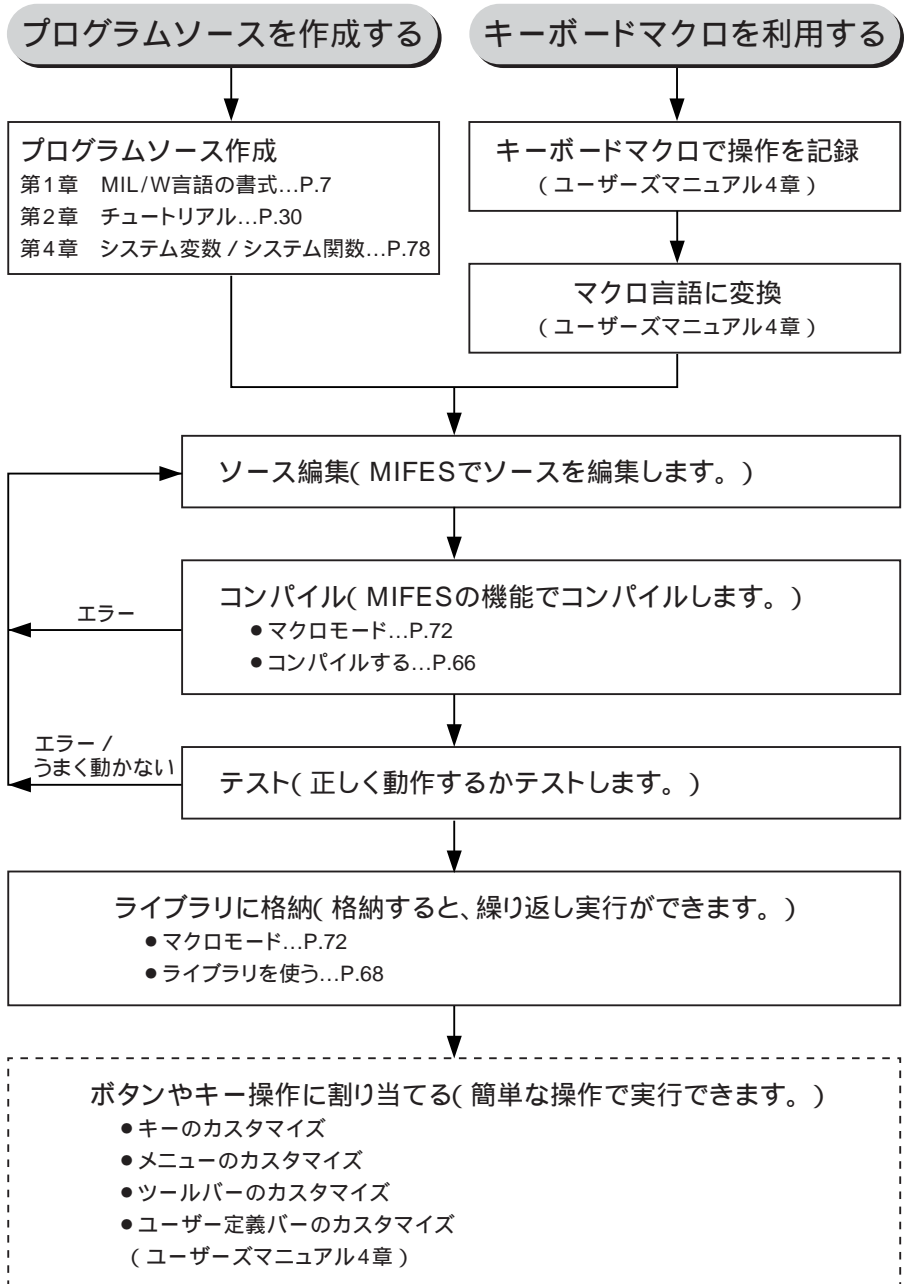


マクロの作成手順

マクロの作成手順は次のようになります。

キーボードマクロで操作を記録し、内容に少し手を加えるなどすると、初心者にも比較的簡単に作成することができます。

また、作成したマクロコマンドを頻繁に実行する場合は、メニューやキー操作などに登録しておく便利です。



MIL/W 言語の書式

ソースファイルの形式

ソースプログラムはテキストファイルに記述します。MIL/W 言語にはソースファイルの拡張子に決まりはありません。そのため、独自の拡張子を付けることもできます。ソースプログラム共通の拡張子を付けると、そのほかのファイルと区別しやすくなります。

(例 .src、.mac、.mil など)

MIL/W 言語ではマクロ定義行の半角アスタリスク * から、次の定義行のアスタリスクの直前、またはファイルの最後までを 1 つのマクロコマンドとみなします。

1 つのソースファイルに 1 つのマクロコマンドを記述することも、複数のマクロコマンドを記述することもできます。

MIL/W 言語の書式

MIL/W 言語の書式は以下のとおりです。

キーボードマクロから【マクロ言語に変換】機能でソースを作成したときも以下のルールで記述されています。

マクロ定義行

- ・マクロコマンドには必ず名前をつけてください。マクロコマンドの名前を定義する行をマクロ定義行といいます。
- ・マクロ定義行は次の書式で記述します。
 - * マクロコマンド名 コメント
- ・マクロコマンド名は最大15バイトまで定義できます。
- ・マクロコマンド名には全角文字も使用できます。
- ・マクロコマンド名とコメントは、タブまたは半角スペースで区切ります。
- ・コメントには、マクロコマンドの概要を最大59バイトまで記述できます。
- ・コメントには全角文字も使用できます。



メモ

キーボードマクロからソースを作成すると、マクロ定義行は次のようになっています。

必ずマクロコマンド名とコメントを書き換えてください。

* マクロコマンド名 マクロコマンドの内容のコメント

ラベル

- ・ラベルには先頭に半角コロン (:) を付け、31バイト以内で記述します。
- ・ラベルの半角英字は大文字と小文字の区別をしません。
- ・ラベルには全角文字も使用できます。

その他

- ・1行には1つの文のみ記述します。
(文とは、実行文、if 構文、while 構文、switch 構文のことを言います。詳しくは次ページ「構文の形式」を参照してください。)
- ・1行(1論理行)には、最大1000バイトまで記述できます。
1行が1000バイト以上になるとコンパイル時にエラーになります。
- ・すべての文は半角スペースまたはタブで段下げ(インデント)などをつけることができます。
ダブルクォーテーション (") で囲まれた文字列定数とコメント以外の場所に入力した全角スペースはエラーになります。
- ・行中で半角のセミコロン (;) 以降の文字列はコメントと見なします。
行の先頭に半角のセミコロン (;) を記述すると、その行全体はコメント行となります。

構文の形式

MIL/W 言語では次の4つの構文が使えます。

1. 実行文
2. if 構文
3. while 構文
4. switch 構文

各構文の説明で使用している表現や単語には以下の意味があります。

- ・構文中の [および] でくくっているものは省略できます。
- ・ < と > でくくっているものは / で区切っている中の一つを選択して記述します。
- ・ラベルとは、goto 文や gosub 文で分岐先にジャンプする場合に、位置を示すためにつける名前です。
- ・文とは、実行文、if 構文、while 構文、および switch 構文のことをさします。



スペースやタブを使って段落をつけて、見やすいマクロの記述を心がけてください。また、コメントをこまめに記述していると、一度作ったマクロを修正したり、変更したりするときに便利です。

1. 実行文

[:ラベル] 実行文]

式、goto文、gosub文、return文、break文、およびcontinue文を総称して、実行文といいます。

式は、変数、定数、関数、およびそれらを演算子や括弧でくくったもののいずれかです。実行文、式については次項で詳しく説明します。

```
例 1 : 式           @6=@str3[@7++]+' A '
      2 : 関数       insstr(" mifes ")
      3 : ラベルと関数 :sub1  move(@@str3)
```

2. if 構文

if 構文には、if ~ then 構文と if ~ else ~ endif 構文の 2 種類があります。

if ~ then 構文

[:ラベル] if 式 then 実行文 / goto :ラベル / gosub :ラベル

if 構文の条件 (if の直後の式の値) が真 (0 でない) のときに、実行する実行文または分岐先を記述します。if から実行文、goto 文、または gosub 文までを 1 行で記述します。

goto 文または gosub 文については「実行文」(P.12) を参照してください。

if 構文の then の次には実行文のみが記述できます。構文 (if 構文、while 構文、switch 構文) を記述することはできません。

例 1 : if ~ then 実行文

```
      if input(@str3, "入力してください") == 0 then exit()
      2 : goto 文
          if @code == 0x0d0a || @code == 0x000a goto :ret
      3 : gosub 文
          if @scol<@6 || @scol>=(@margin-3) || @line<@7 gosub :label
```

if ~ else ~ endif 構文

[:ラベル] if 式
 [文]
 [else if 式 / else]
 [文]
 endif

if 構文の条件 (if の直後の式の値 が真(0でない))のときの記述が 1 行で終わらない場合や、偽のときの記述が必要な場合に使います。偽のときは、else 以降に記述します。if 構文の終わりには endif を記述します。

```
例: if @command == COMMAND_RET
      insstr(0x0d0a)
    else if @command == COMMAND_BS
      move("l ")
      delchar(1)
    endif
```



- endif を忘れずに記述してください。endif がないと、コンパイル時にエラーになります。
- MIFES に同梱されているマクロライブラリ MIW.LIB には、if ~ else ~ endif 構文を入力するコマンドがあり、メニューなどから実行できます。

3. while 構文

```
[ :ラベル ] while 式
              [ 文 ]
              wend
```

while 構文の式の値が真(0でない)の間、wend までのすべての文を繰り返し実行します。wend は while 構文の最後に記述します。

```
例: while @col >= 1
      if ctype(@code) > 10
        move("r ")
        break
      endif
      move("l ")
    wend
```



- wend を忘れずに記述してください。wend がないと、コンパイル時にエラーになります。
- MIFES に同梱されているマクロライブラリ MIW.LIB には、while ~ wend 構文を入力するコマンドがあり、メニューなどから実行できます。

4.switch構文

```
[ :ラベル] switch 式
           case 定数
           [ case 定数]
             [ 文]
             [ break]
           [ default]
             [ 文]
             [ break]
        endsw
```

switch 構文の式の値が case 文の定数の値と同じときに、その case 文の直後から endsw 文の直前までの文を実行します。

ただし途中に break 文があると、switch 構文から処理が抜けます。

break 文については次項の「実行文」を参照してください。

switch 構文の式の値が case 文の定数の値と一致しないときは、default 文の直後から endsw 文の直前までの文を実行します。

default 文がないときは switch 構文を終了します(endsw 文にジャンプします)。

case 文の定数の値を複数設定して、同じ処理を実行したいときは、case 文を続けて記述します。

以上のように、C 言語の switch と同じ制御です。

```
例: switch @5
     case 0
       insstr("¥n")
       break
     case 1
       insstr("¥s")
       break
     case 2
     case 3
     case 4
       insstr("¥t")
       break
     default
       delchar(2)
       break
endsw
```



ポイント

- endsw を忘れずに記述してください。endsw がないと、コンパイル時にエラーになります。
- MIFES に同梱されているマクロライブラリ MIW.LIB には、switch ~ endsw 構文を入力するコマンドがあり、メニューなどから実行できます。

実行文

実行文には次の6つがあります。

1. 式
2. goto 文
3. gosub 文
4. return 文
5. break 文
6. continue 文

1. 式

式も実行文の1つです。式とは、変数、定数、関数、およびそれらを演算子や括弧でくくったもののいずれかです。

式については次の節で詳しく説明しています。

- 例1 : @str4[0] = VK_ESC (グローバル配列変数にマクロ定数を代入)
例2 : insstr(@3+2) (システム関数を実行)
例3 : @byte = @10 (システム変数にグローバル単純変数を代入)
例4 : @5++ (グローバル単純変数をインクリメント)

2. goto 文

goto :ラベル

ラベルで指定した文へジャンプします。

ジャンプ先の処理を実行し終えても、処理はgoto文の前の処理には戻りません。

また、1つのマクロコマンドに使用できるgoto文の数は最大100個です。

```
例: if @1 < 0 goto :error
    .
    .
    .
    :error clsmess()
    exit()
```

3. gosub 文

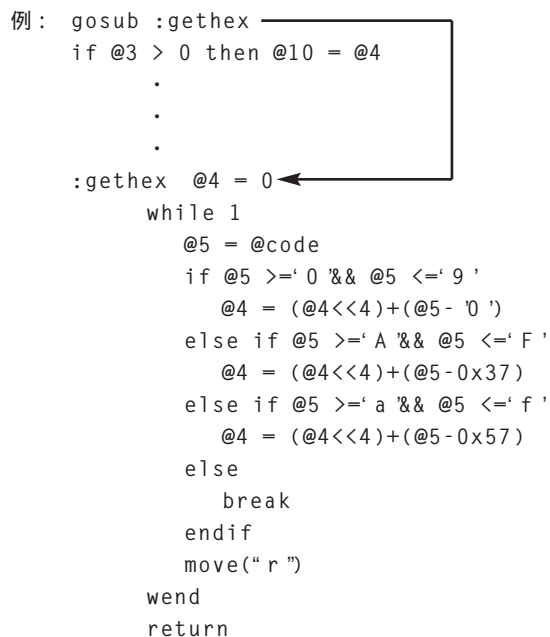
gosub :ラベル

ラベルで指定したサブルーチンへジャンプします。

このとき、gosub 文の次の文の位置を専用のスタックに記録します。そのため、サブルーチンの中の return 文で、記憶している gosub 文の次の文に処理が戻ります。

また、1つのマクロコマンドで使用できる gosub 文の数は最大100個です。return 文については、次項を参照してください。

```
例: gosub :gethex
    if @3 > 0 then @10 = @4
        .
        .
        .
:gethex @4 = 0
    while 1
        @5 = @code
        if @5 >=' 0' && @5 <=' 9'
            @4 = (@4<<4)+(@5-' 0')
        else if @5 >=' A' && @5 <=' F'
            @4 = (@4<<4)+(@5-0x37)
        else if @5 >=' a' && @5 <=' f'
            @4 = (@4<<4)+(@5-0x57)
        else
            break
        endif
        move(" r ")
    wend
return
```

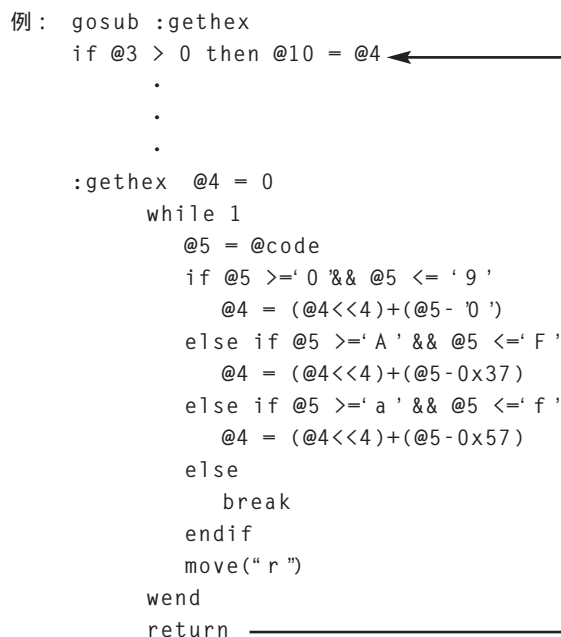
A diagram consisting of a rectangular box with a black border. The box starts at the line 'gosub :gethex' and extends to the right. From the right side of the box, a horizontal arrow points left to the line ':gethex @4 = 0', indicating the jump target.

4. return 文

gosub 文と対で使用します。return 文はサブルーチンを終了し、gosub 文の次の文の位置に戻ります。

gosub 文については、前項を参照してください。

```
例: gosub :gethex
     if @3 > 0 then @10 = @4 ←
     .
     .
     .
     :gethex @4 = 0
           while 1
             @5 = @code
             if @5 >='0' && @5 <='9'
               @4 = (@4<<4)+(@5-'0')
             else if @5 >='A' && @5 <='F'
               @4 = (@4<<4)+(@5-0x37)
             else if @5 >='a' && @5 <='f'
               @4 = (@4<<4)+(@5-0x57)
             else
               break
             endif
             move(" r ")
           wend
     return
```



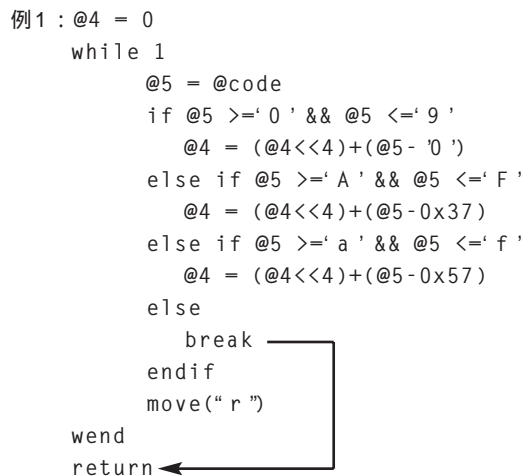
5. break 文

break 文はwhile ~ wend間、およびswitch ~ endsw間で使用します。

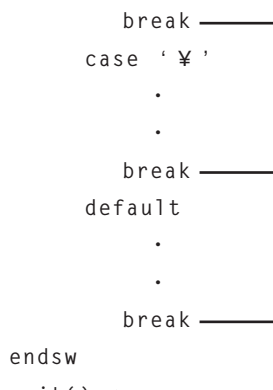
while ~ wend間にあるときには、同じレベルのwhile ~ wend構文を抜け出して、wendの次の文にジャンプします。

switch ~ endsw間にあるときには、同じレベルのswitch ~ endsw構文を抜け出して、endswの次の文にジャンプします。

```
例1: @4 = 0
      while 1
        @5 = @code
        if @5 >='0' && @5 <='9'
          @4 = (@4<<4)+(@5-'0')
        else if @5 >='A' && @5 <='F'
          @4 = (@4<<4)+(@5-0x37)
        else if @5 >='a' && @5 <='f'
          @4 = (@4<<4)+(@5-0x57)
        else
          break
        endif
        move(" r ")
      wend
      return ←
```



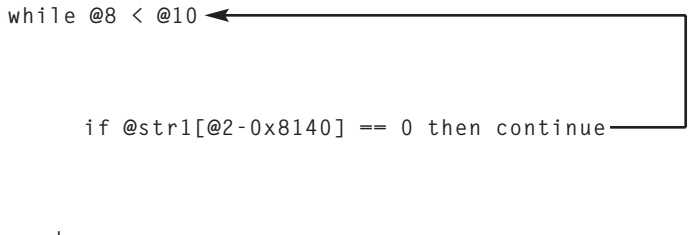
```
例2 : switch @code
      case ' ? '
        .
        .
        break
      case ' ¥ '
        .
        .
        break
      default
        .
        .
        break
    endsw
    exit()
```



6. continue 文

continue 文は while ~ wend 間だけで使用します。continue 文の後から wend までの間をスキップして、while 構文の式の評価に移ります。

```
例: while @8 < @10 ←
      .
      .
      if @str1[@2-0x8140] == 0 then continue
      .
      .
    wend
```



式とは、変数、定数、関数、およびそれらを演算子や括弧でくくったもののいずれかです。具体的には以下の6つです。

1. 変数（単純変数、配列変数、システム変数）
2. 定数（10進定数、16進定数、文字定数、文字列定数、マクロ定数）
3. 関数（システム関数）
4. 単項演算子を含んだもの
5. 2項演算子を含んだもの
6. 上記1～5を括弧でくくったもの

MIL/W 言語では、変数、定数、関数、および式はすべて同じように扱います。これはC言語と同じ仕様です。すべての式には必ずその演算結果の値があります。この値が0のとき、その条件式は偽であると見なします。反対に0以外の値のときにはその条件式は真であると見なします。

条件式とは、if 構文やwhile 構文などのすぐ後ろにあり、その条件式の結果によってなんらかの処理の分岐が生じる場合の式を指します。基本的には通常の式と同じものです。

代入文も式のひとつです(例4)。演算子 = は、右辺の式の値を左辺の変数に代入し、演算結果として左辺の変数の値を返します。

変数や定数だけの式もあります(例1、例2)。また、条件式に定数1とすると、演算結果は常に真になります。たとえば、while 構文の条件式に定数1とすると、while 構文の処理を無条件で繰り返します(例7)。

```
例1 : @12                (単純変数)
例2 : 0x834b             (16進定数)
例3 : insstr(0x0d0a)    (システム関数)
例4 : @str2[@1++] = 0
例5 : if(@1=2) && (@2=0)
例6 : if(@2=@code-0x20) <= 64 then insstr(@2+0x20)
例7 : while 1
        @1++
    wend
```

演算式

MIL/W 言語では、以下の演算子が使えます。

演算は符号付き32ビット整数で実行します。符号なし16ビット整数(配列変数)は、上位に16ビット分の 0 を付加して32ビットに変換後、演算します。

単項演算子

単項演算子は2項演算子よりも演算の優先順位は高くなります。

++	プリインクリメント	右側の単純変数を使用する前に1を加えます。
++	ポストインクリメント	左側の単純変数を使用した後に1を加えます。
--	プリデクリメント	右側の単純変数を使用する前に1を引きます。
--	ポストデクリメント	左側の単純変数を使用した後に1を引きます。
&	アドレス演算子	右側の配列変数要素の位置を返します。 この演算子は配列変数にのみ使えます。 単純変数やシステム変数には使えません。
-	2の補数(負の数)	右側の式の2の補数を返します。
	1の補数(全ビットの反転)	右側の式の1の補数を返します。
!	論理否定	右側の式の値が0ならば1を返します。 0でなければ0を返します。



プリインクリメント、ポストインクリメント、プリデクリメント、およびポストデクリメントは、ユーザー単純変数にのみ使用できます。配列変数やシステム変数には使えません。

2項演算子

括弧内の数値は演算の優先順位を表しています。数値が大きいほど優先順位が高くなります。

代入演算子		
=	代入(2)	右側の式の値を左側の変数に代入します。 左側には変数を記述します。
&=	ビット論理積と代入(2)	左側の変数の値と右側の式の値のビット論理積を計算し、その結果を左側の変数に代入します。 左側には変数を記述します。
=	ビット論理和と代入(2)	左側の変数の値と右側の式の値のビット論理和を計算し、その結果を左側の変数に代入します。 左側には変数を記述します。
^=	ビット排他的論理和と代入(2)	左側の変数の値と右側の式の値のビット排他的論理和を計算し、その結果を左側の変数に代入します。 左側には変数を記述します。

<<=	ビット左シフトと代入 (2)	左側の変数の値を右側の式の値が示すビット数だけ左シフトし、その結果を左側の変数に代入します。左側には変数を記述します。
>>=	ビット右シフトと代入 (2)	左側の変数の値を右側の式の値が示すビット数だけ右シフトし、その結果を左側の変数に代入します。左側には変数を記述します。
%=	剰余と代入 (2)	左側の変数の値を右側の式の値で除算し、その余りを左側の変数に代入します。左側には変数を記述します。
*=	乗算と代入 (2)	左側の変数の値と右側の式の値を乗算し、その結果を左側の変数に代入します。左側には変数を記述します。
/=	除算と代入 (2)	左側の変数の値を右側の式の値で除算し、その商を左側の変数に代入します。左側には変数を記述します。
+=	加算と代入 (2)	左側の変数の値と右側の式の値を加算し、その結果を左側の変数に代入します。左側には変数を記述します。
-=	減算と代入 (2)	左側の変数の値から右側の式の値を減算し、その結果を左側の変数に代入します。左側には変数を記述します。
論理演算子		
&&	論理積 (3)	左側の式の値が真(0でない)で、かつ右側の式の値も真(0でない)の場合に、真(値1)を返します。それ以外の場合は偽(値0)を返します。
	論理和 (3)	左側の式の値が真(0でない)か、また右側の式の値が真(0でない)の場合に、真(値1)を返します。それ以外の場合には偽(値0)を返します。
&	ビット論理積 (11)	左側の式の値と右側の式の値をビット論理積した値を返します。
	ビット論理和 (11)	左側の式の値と右側の式の値をビット論理和した値を返します。
^	ビット排他的論理和 (11)	左側の式の値と右側の式の値をビット排他的論理和した値を返します。
関係演算子		
>=	関係演算：以上 (4)	左側の式の値が右側の式の値と同じか、より大きい場合に真(値1)を返します。それ以外の場合は偽(値0)を返します。
<=	関係演算：以下 (4)	左側の式の値が右側の式の値と同じか、より小さい場合に真(値1)を返します。それ以外の場合は偽(値0)を返します。

==	関係演算：等しい(4)	左側の式の値と右側の式の値が等しい場合に真(値1)を返します。 それ以外の場合は偽(値0)を返します。
!=	関係演算：等しくない(4)	左側の式の値と右側の式の値が異なる場合に真(値1)を返します。 それ以外の場合は偽(値0)を返します。
>	関係演算：より大きい(4)	左側の式の値が右側の式の値より大きい場合に真(値1)を返します。 それ以外の場合は偽(値0)を返します。
<	関係演算：より小さい(4)	左側の式の値が右側の式の値より小さい場合に真(値1)を返します。 それ以外の場合は偽(値0)を返します。
算術演算子		
+	加算(5)	左側の式の値と右側の式の値を加算した値を返します。
-	減算(5)	左側の式の値から右側の式の値を減算した値を返します。
*	乗算(7)	左側の式の値と右側の式の値を乗算した値を返します。
/	除算(7)	左側の式の値を右側の式の値で除算したときの商を返します。
%	乗除(9)	左側の式の値を右側の式の値で除算したときの余りを返します。
シフト演算子		
<<	ビット左シフト(10)	左側の式の値を、右側の式の値が示すビット数分、左にシフトした値を返します。
>>	ビット右シフト(10)	左側の式の値を、右側の式の値が示すビット数分、右にシフトした値を返します。

同じ演算子または優先順位の高い演算子では、後に記述した方(右側の方)の演算が、先に実行されます。この仕様はDOS版MIFESのMIL言語とは逆ですので注意してください。MIL/W言語はC言語と同じ仕様です。

例 @1=@2=0 と @1=(@2=0) は同じ意味です。

ただし、演算子の優先順位はC言語とまったく同じではありません。1つの式の中に複数の演算子を記述する場合には、優先させたい演算部分を()でくくって記述してください。

論理和に使用する演算子!は[SHIFT]+[¥]キーで入力します。



メモ



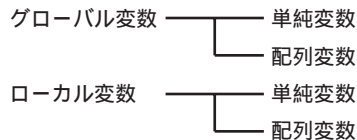
メモ

変数と定数について

変数

MIL/W 言語には以下の 6 種類の変数があります。

ユーザー変数



calc()関数用の浮動小数点変数

システム変数

ユーザー変数

ユーザーが自由に使える変数のことを、システム変数に対して「ユーザー変数」と呼びます。ユーザー変数のうち、変数名をもつユーザーが定義できる変数のことを「ユーザー定義変数」と呼びます。

ユーザー変数にはグローバル変数とローカル変数の 2 種類があり、そのそれぞれに単純変数と配列変数があります。

グローバル変数とローカル変数の違い

グローバル変数の値はマクロコマンドの実行後も、MIFESを終了するまで有効です。(ただしユーザーが名前をつけたグローバルユーザー定義変数の場合は操作が必要になります。次項「グローバル変数」を参照してください。)

それに対して、ローカル変数の値は 1 つのマクロコマンドの中だけで有効になります。そのため、1 つのソースプログラムの中からシステム関数 macro で別のマクロコマンドが呼び出された場合でも同じローカル変数名を使うことができます。

単純変数と配列変数の違い

単純変数には、10進定数、16進定数、文字定数、および式の結果などの値が代入できます。それに対して、配列変数には文字列定数も代入できます。それぞれの定数については、次項で詳しく説明しています。

グローバル変数

1. グローバル単純変数(符号付き32ビット整数)

@1, @2, @3, ..., @16

上記以外にも、ユーザーが自由に名前を付けられる変数が16個あります。これらをグローバルユーザー定義変数といいます。グローバルユーザー定義変数名は、半角の @ で始まる最大32バイト(@を除いて31バイト)の文字列です。このときの英字の半角文字は、大文字と小文字

の区別をしません。

また、グローバルユーザー定義変数は宣言せずに使用できます。

2. グローバル配列変数(符号なし16ビット整数を要素とする1024要素の配列)

```
@str1[ 1024 ]
.
.
@str&[ 1024 ]
```

配列変数名だけを記述した場合、その配列変数の先頭位置を指定したことになります。

例： @str1 と &@str1[0]は同じ意味になります。

また、下記のようにも指定できます。

例： &@str1[10]とすると @str1 の 10 要素目からの配列となります。

配列変数の要素は符号なし16ビット整数です。しかし、この要素を演算するときは、上位に16ビット分の 0 を付加して32ビットに変換後、演算します。

グローバルユーザー定義変数は、一般的には1つのマクロコマンドの中だけで使用します。複数のマクロコマンドを通して同じ変数名を使用する場合は、それぞれのマクロコマンドの先頭で同じ順に変数をダミー的に宣言してください。



```
例： *macro-oya    親マクロ
      @@1=@start   ;ダミー使用
      @@1=@end     ;ダミー使用
      : ( @start, @end を使った処理 )
macro( " macro-ko " )
      :
*macro-ko    子マクロ
      @@1=@start   ;ダミー使用
      @@1=@end     ;ダミー使用
      : ( @start, @end を使った処理 )
```

ローカル変数

1. ローカル単純変数(符号付き32ビット整数)

```
@@1, @@2, @@3, ..., @@16
```

上記以外にも、ユーザーが自由に名前を付けられる変数が16個あります。これらをローカルユーザー定義変数といいます。ローカルユーザー定義変数名は、半角の @@ で始まる最大32バイト(@@ を除いて30バイト)の文字列です。このときの英字の半角文字は、大文字と小文字の区別をしません。

また、ローカルユーザー定義変数は宣言せずに使用できます。

2. ローカル配列変数(符号なし16ビット整数を要素とする1024要素の配列)

```
@@str1[ 1024 ]  
:  
@@str8[ 1024 ]
```

配列変数名だけを記述した場合、その配列変数の先頭位置を指定したことになります。

例 @@str2 と &@@str2[0]は同じ意味になります。

また、下記のようにも指定できます。

例 &@@str2[10]とすると @@str2 の 10 要素目からの配列となります。

配列変数の要素は符号なし16ビット整数です。しかし、この要素を演算するときには、上位に16ビット分の 0 を付加して32ビットに変換後、演算を実行します。

calc()関数用の浮動小数点変数

```
@F0, @F1, @F2, @F3, @F4
```

calc()関数で指定する演算式の中だけで使用できる浮動小数点型の変数が、上記の5つあります。このうち、@F0は演算の結果が常に自動的に格納される変数です。これらの変数で扱える値の範囲は、- 1.7E308 ~ + 1.7E308 です。精度は15桁です。いわゆる倍精度浮動小数点型です。

システム変数

システム変数とは、エディタ内のさまざまな状態を表す変数です。エディタ内の最新の状態を表すように自動的に更新されています。

システム変数は参照用に使ったり、値を代入することで、エディタ内のさまざまな状態を変更できます。たとえば、カーソルを任意の位置にジャンプさせたり、カーソル位置に文字を入力したりすることができます。

ただし、代入を禁止しているシステム変数もあります。代入を禁止しているシステム変数に値を代入しても無視されます。各システム変数に代入ができるかどうかは、第4章の各システム変数の説明を参照してください。

システム変数は符号付き32ビット整数で、正の数(ビット31が0となった数)です。正常な値が返せない場合は、負の数(通常-1)になります。

定数

MIL/W 言語の定数には次の5つがあります。

- 10進定数
- 16進定数
- 文字定数
- 文字列定数
- マクロ定数

10 進定数

32ビット符号付き整数です。直前に負の符号 - をつけると負の数を指定できます。

例1 : 32

例2 : -2500

16 進定数

32ビット符号付き整数です。先頭に 0x または \$ を付けて記述します。

指定した数値が32ビット分(8桁分)に満たない場合は、上位部分に0が入ります。したがって、0xffff と記述すると 0x0000ffff となります。

0xfffffffff(10進で-1)とはなりませんので、注意してください。

例1 : 0x0d0a

例2 : \$7f

文字定数

32ビット符号付き定数です。1文字を半角のシングルクォーツ " でくくって記述します。

半角文字の場合は最下位の8ビットに文字コードが入り、残り上位24ビットが0になります。

全角文字の場合は1バイト目のコードが中下位の8ビットに入り、2バイト目のコードが最下位の8ビットに入ります。中上位8ビットと最上位8ビットは0になります。

例1 : 'A' (0x00000041)

例2 : '漢' (0x00008ABF)

また、以下のメタ文字も指定できます。

'¥n' 改行文字 (0x00000d0a)

'¥t' タブ文字 (0x00000009)

'¥x??' 任意コードの文字 (0x000000??) (??は16進2桁の文字)

'¥¥' 文字の¥ (0x0000005c)

文字列定数

文字列を半角のダブルクォーツ " でくくって記述します。最大512バイト(半角512文字、全角256文字)まで記述できます。

例 : "メッセージ"

また、文字列中に以下のメタ文字も指定できます。

"¥n" 改行文字 (0x0d,0x0a)

"¥r" CRコード (0x0d)

"¥t" タブ文字 (0x09)

"¥x??" 任意コードの文字 (0x??) (??は16進2桁の文字)

"¥¥" 文字¥ (0x5c)

"¥"" 文字" (0x22)



¥x00(ヌル)は文字列の終りと見なされます。注意してください

システム関数の引数を指定するために、文字列定数を使用します。
ただし、システム関数や引数の位置によってはメタ文字が使えない場合があります。文字列定数の引数が次のようなデータを表す場合、メタ文字は使えません。

- ・ 検索文字列、置換文字列
- ・ ファイル名、パス名、ディレクトリ名
- ・ child()関数の子プロセスコマンド
- ・ printf()、sprintf()関数の書式指定文字列
- ・ strlist()関数の第2引数以降

¥には、文字列定数のメタ文字、検索文字列や書式指定文字列のメタ文字、またはディレクトリの区切りを表す記号というように、いくつかの意味があります。そのため、メタ文字の指定を複雑にしないために、上記のような場合には文字列定数にメタ文字は使えません。

ただし、メタ文字¥(文字")は常に使えます。これは¥"はコンパイラが処理し、その他のメタ文字はインタプリタが各システム関数を実行するときに処理しているためです。

マクロ定数

32ビット符号付き定数です。必ず半角の大文字で記述します。

マクロ定数はビット論理演算に使用するため、符号付きか符号なしかは関係ありません。

システム関数の引数に指定するときや、システム変数の値を判断するときに使います。各マクロ定数については、第4章の各システム関数やシステム変数の説明を参照してください。

例：@1 = waitevent(EVENT_VK | EVENT_COMMAND, @str4)

関数や変数の調べ方

ソースプログラムを記述していて、実行すべき処理がわかっているのにシステム関数やシステム変数がわからなかったり、どのシステム関数を使うかはわかっているけどパラメータ(引数)などの記述方法がわからないというときがあります。このようなときに効率的に調べる方法を説明します。

処理内容から関数や変数を調べる

マクロライブラリ(MIW.LIB)の中には、システム関数、システム変数、およびマクロ定数を調べられるマクロコマンドが入っています。これらのマクロコマンドもMIL/W言語で作成されています。

constant : マクロ定数を調べる
sysvari : システム変数を調べる
sysfunc : システム関数を調べる

これらのマクロコマンドを実行すると、マクロ定数、システム変数、およびシステム関数をそれぞれ用途別に分類し表示します。その中から目的のものを探して、ソースプログラムに入力することもできます。



この機能を使うためには、マクロライブラリに次のマクロコマンドが格納されている必要があります。この3つのソースプログラムはMIW.MACの中にあります。必要に応じて、コンパイルライブラリに格納してください。ただし、出荷時にはデフォルトでライブラリに格納されています。

*constant MIL/W 言語のマクロ定数の選択と挿入
*sysvari MIL/W 言語のシステム変数名の選択と挿入
*sysfunc MIL/W 言語のシステム関数名の選択と挿入

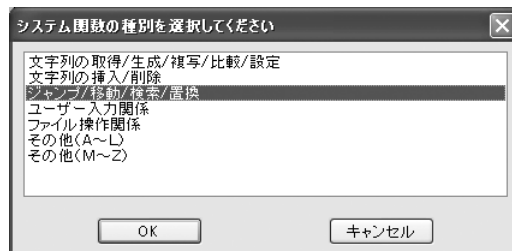
コンパイル方法、ライブラリへの格納方法については第3章を参照してください。

たとえば、文字列を検索するためのシステム関数がわからないときには以下の手順で調べます。

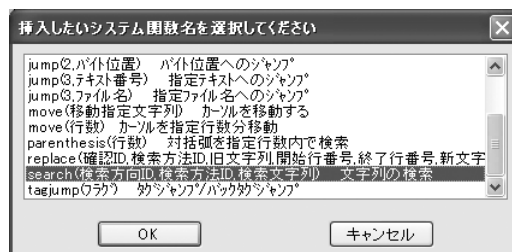
- ① 【マクロ(M)】-【指定マクロコマンドの実行(X)】を選択してsysfuncを実行します。



- ② システム関数で可能な処理が、目的別に表示されます。文字列の検索ですから、[ジャンプ/移動/検索/置換]を選択し、[OK]ボタンをクリックします。



- ③ 分類されたシステム関数の一覧が表示されます。文字列の検索を実行する [search] を選択し、[OK]ボタンをクリックします。



- ④ 選択したシステム関数と引数が、カーソル位置に入力されます。

```
search(検索方向ID, 検索方法ID, 検索文字列)
```



入力できたシステム関数のパラメータ(引数)の指定方法などがわからない場合には、ヘルプやイージーヘルプを使うとたいへん便利です。

ヘルプやイージーヘルプの使い方は次項またはユーザーズマニュアルを参照してください。

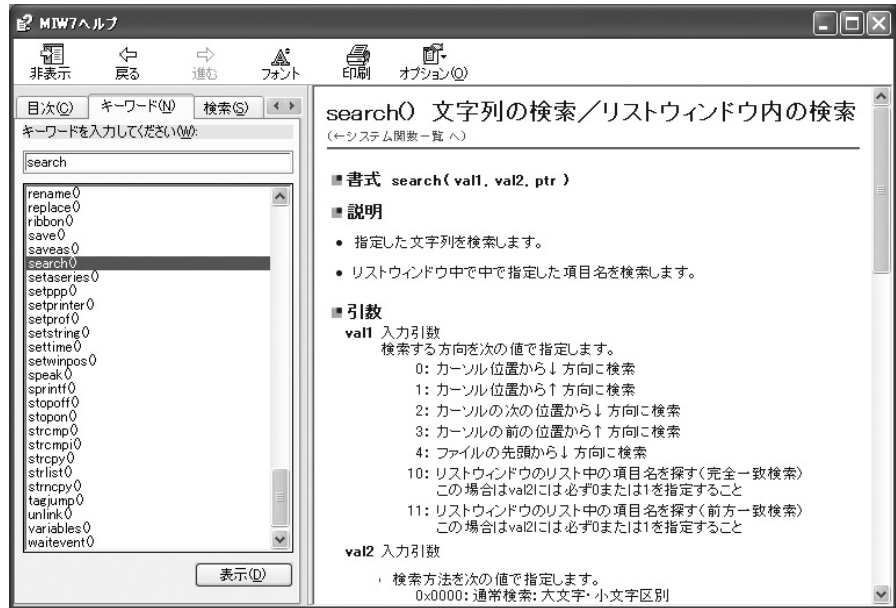
ヘルプを使う(関数・変数の意味やパラメータを調べる)

システム関数名はわかっているが、パラメータに設定する値がわからなかったり、参考にして
いるソースプログラム中のシステム変数やマクロ定数の意味を調べたりしたい場合があります。
このような場合に、マニュアル(本書)の索引から探すほかに、ヘルプやイーザーヘルプで
検索するという方法があります。

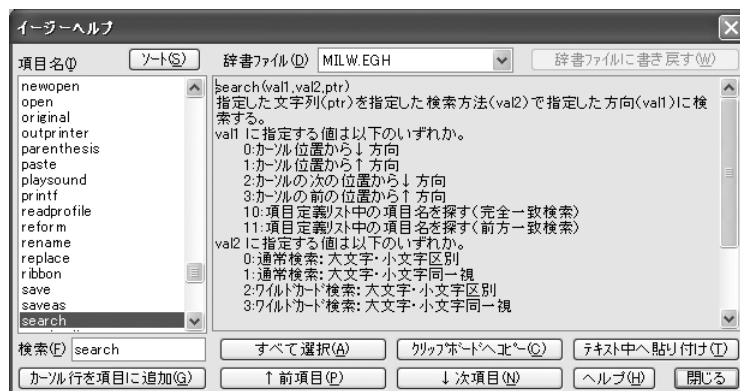
- 1 調べたいマクロ定数、システム変数またはシステム関数の位置にカーソルを移動します。変数名 / 関数名内であればカーソルの位置に関係なくヘルプは表示されます。また直後の位置でもヘルプは表示されます。

例: `search`

- 2 【ヘルプ(H)】-【カーソル位置の語をヘルプ(R)】を選びます。
`search`のヘルプが表示されます。



または、【ヘルプ(H)】-【カーソル位置の語をイーザーヘルプ(E)】を選びます。



参照するだけでなく、イーザーヘルプの内容をソースプログラムに貼り付けたり、クリップボードにコピーすることもできます。



メモ

ここではすでに辞書ファイルとしてMILW.EGHを選択していることを前提に説明しています。イーザーヘルプにはマクロのシステム変数やシステム関数について表示されるMILW.EGHと、HTMLのタグや属性について表示されるHTML.EGHの2種類が同梱されており、出荷時にはデフォルトでMILW.EGHが設定されています。



参照

MILW.EGHは、MIFESのポストプロセッサ機能を使って作成されたイーザーヘルプ辞書です。イーザーヘルプ辞書は、ユーザーが自由に作成・編集できます。詳しくはユーザーズマニュアル、またはヘルプを参照してください。

第2章 チュートリアル

この章では具体的に5つのマクロコマンドの作成方法について説明しています。

チュートリアル1ではMIL/W言語の基本的な記述方法を詳しく説明しています。そのため、はじめてMIL/W言語を使って外部マクロを作成される方は、チュートリアル1から読まれることをおすすめします。

チュートリアルには同じシステム関数や変数の説明が含まれている場合があります。これはチュートリアルを順番に読まれなかった場合を想定して説明しているためですのでご了承ください。また、チュートリアルで使用するシステム関数や変数のすべてのパラメータについては説明していません。各チュートリアルに必要なパラメータのみ記述していますので、詳しくは第4章の各システム関数や変数の説明を参照してください。

5つのチュートリアルには比較の実用性の高いマクロを用意しました。これらのソースファイルはCD-ROM内のTUTORIALフォルダに収録されています。

目次

チュートリアル1 右寄せ	30
チュートリアル2 改行だけの行を削除	35
チュートリアル3 指定した行範囲の印刷	41
チュートリアル4 範囲指定内の文字列の置換	51
チュートリアル5 C言語関数内の文字列の置換	60

右寄せ

機能 カーソル位置の語を右寄せします。

マクロの内容

右寄せする文字列の最後の位置を記憶し、右マージンの設定値から文字列のバイト数分を引きます。その引いた値分の半角スペースを文字列の前に挿入し、右寄せします。

- 1.カーソルを行末に移動
- 2.挿入する文字数を計算
- 3.計算した文字数分、半角スペースを挿入

マクロ実行時の制限

- ・マクロコマンドを実行する前に右寄せする行にカーソルがあること。
- ・右寄せする行の行末は改行であること。

作成手順

マクロの定義行を記述します。

行頭に半角でアスタリスク*、その後にマクロコマンド名とマクロコマンドのコメントを記述します。

```
*right      右寄せ
```

マクロコマンド名は、半角で最大15バイトまで記述できます。

コメントは、最大59バイトまで記述できます。コメントにはマクロコマンドの概要を記述します。マクロコマンド名とコメントは、タブまたは半角スペースで区切ります。

ここでは、マクロコマンド名を「right」とし、コメントには「右寄せ」と記述します。

マクロソースをコンパイル後マクロライブラリに格納すると、このマクロコマンド名とコメントが表示されます。



マクロライブラリに格納したマクロコマンドを一覧表示したり、実行するには「マクロコマンド一覧」ダイアログボックスを使います。このダイアログボックスのリストボックスにはコメントが38バイトまで表示できます。39バイト目以降は、リストボックスの水平スクロールバーを使って表示させることができます。

[カレントマクロコマンド] 欄のコメントは59バイトまで表示できます。

マクロコマンド名と、マクロソースファイル名は共通にすることをおすすめします。共通にしていると、マクロコマンドの修正が必要になった場合にマクロソースファイルを見つけやすくなります。

コンパイルするとソースプログラムは中間コードに変換されます。その中間コードをマクロライブラリに格納すると、ソースファイルはマクロの実行には必要なくなります。

マクロコマンド(中間コード)だけで、マクロが実行できます。

しかし、マクロライブラリに格納したマクロコマンドをソースファイルに戻すことはできませんので、将来、マクロコマンドを修正する可能性がある場合は、ソースファイルを残しておくことをおすすめします。



メモ

コンパイルモードを、中間コードにソースプログラムを埋め込む設定にしていると、マクロコマンド(中間コード)からソースプログラムを取り出すことができます。コンパイルモードは【マクロ(M)】-【マクロモード設定/コンパイル(M)】で設定します。マクロモードについては、第3章の「マクロモードについて」を参照してください。

MIL/W 言語にはソースファイルの拡張子に決まりはありません。そのため、独自で拡張子を付けることもできます。ソースファイル共通の拡張子を付けると、その他のファイルと区別しやすくなります。(例 .src、.mac、.mil など)

MIL/W 言語ではマクロ定義行の半角のアスタリスク * から、次の定義行のアスタリスクの直前、またはファイルの最後までを1つのマクロコマンドとみなします。

1つのソースファイルに複数のマクロコマンドを記述したり、1つ1つマクロコマンドごとにソースファイルを作成することもできます。

マクロプログラムを記述するときの基本は以下の通りです。

- (1) 1行には1文だけを記述します。
- (2) 1行(1論理行)が1000バイト以下になるように記述します。
1000バイト以上記述すると、正常にコンパイルできない場合があります。
- (3) すべての文は、半角スペースまたはタブを使って段落をつけられます。
- (4) 1行中で半角のセミコロン ; 以降の文字列はコメントとみなします。
行の先頭の文字が ; のときは、その行全体がコメントになります。



ポイント

半角スペースやタブを使って段落をつけると、マクロソースが見やすくなります。また、コメントをこまめに記述しておく、一度作ったマクロを修正したり、変更したりする場合に便利です。

では、マクロプログラムの記述をはじめます。

① カーソルを行末に移動

右寄せする文字列の最後の位置を取得するために、まずカーソルを行末に移動します。カーソル移動にはシステム関数 `move` を使います。行末に移動するときには `)` を使います。`)` をダブルクォーテーション “ ” でくくって記述します。その後ろに、タブで区切ってコメントを記述します。コメントのはじめには半角セミコロン ; を記述します。その後には「行末にカーソル移動」と処理の説明を記述します。

```
*right 右寄せ  
  
    move(" )") ;行末にカーソル移動
```

② 挿入する文字数を計算

次に文字列の前に挿入する文字数を計算します。挿入する文字数は右マージンの値から文字列の桁数を引いて算出します。つまり、半角で5文字の文字列があり、右マージンが80桁の場合には、 $80 - 5 = 75$ の75バイト分の半角スペースを挿入することになります。

文字列の桁数は行末の桁位置で取得できます。カーソルを上記①で行末に移動していますので、カーソルの桁位置が文字列の終わりまでの桁数を表しています。カーソルの桁位置はシステム変数 `@col` で取得できます。また、右マージンはシステム変数 `@margin` で取得できます。この `@margin` から `@col` を引きます。引いた値をグローバル単純変数 `@1` に代入します。代入演算子 `=` を使い、以下のように記述します。`@1` に挿入する文字数を代入したことになります。 `@1` は次の③で、半角スペースを挿入するときのカウンタとして使います。

```
*right 右寄せ  
  
    move(" )") ;行末にカーソル移動  
    @1 = @margin - @col ;挿入する文字数の計算
```

3

計算した文字数分、半角スペースを挿入

挿入する文字数が計算ができましたので、半角スペースを挿入します。しかし、はじめにカーソルを行末に移動していますので、現在のカーソル位置は行末にあります。この位置で半角スペースを挿入しても意味がありませんので、カーソルを行頭に移動します。カーソルの移動は同じようにシステム関数 `move` を使い、行頭に移動する (を記述します。

```
*right  右寄せ

move(" ")           ;行末にカーソル移動
@l = @margin - @col ;挿入する文字数の計算
move(" ( ")         ;行頭にカーソル移動
```

ここまでで半角スペースを挿入するための準備ができました。

先程取得した文字数分の半角スペースを挿入するのですが、これは同じ処理の繰り返しになります。このように処理を繰り返すときには `while` 構文を使います。 `while` 構文は式が真の間、 `wend` までの処理を繰り返します。ここで式には「挿入する文字数分を繰り返す」という記述が必要です。そこで、演算子のポストデクリメントを使います。

つまり、 `@l` (挿入する文字数) から 1 ずつ引き、 `@l` が 0 になったとき、処理を終了します。

これを記述すると以下ようになります。

また、 `wend` も合わせて記述します。 `wend` を記述し忘れるとコンパイルエラーになりますので、 `while` を記述するときに合わせて記述することをおすすめします。

```
*right  右寄せ

move(" ")           ;行末にカーソル移動
@l = @margin - @col ;挿入する文字数の計算
move(" ( ")         ;行頭にカーソル移動
while (@l-- )       ;計算した文字数まで繰り返す
wend
```

半角スペースを挿入するには、文字をカーソル位置に挿入するシステム関数 `insstr` を使います。半角スペースはメタ文字で `¥s` です。ダブルクォーテーションでくくり、以下のように記述します。この `insstr` 文は `while` 構文の中で繰り返すのですから、タブで段落をつけて記述します。

```
*right  右寄せ

move(" ")           ;行末にカーソル移動
@l = @margin - @col ;挿入する文字数の計算
move(" ( ")         ;行頭にカーソル移動
while (@l-- )       ;計算した文字数まで繰り返す
    insstr(" ¥s ")   ;半角スペースの挿入
wend
```

ここまでで右寄せの処理の記述はできました。

4

終了

最後にマクロの終了を表すシステム関数 `exit` を記述します。

これで、右寄せマクロのソースプログラムは完成です。

```
*right 右寄せ

move(" ") ;行末にカーソル移動
@l = @margin - @col ;挿入する文字数の計算
move(" ") ;行頭にカーソル移動
while (@l--);計算した文字数まで繰り返す
    insstr(" %s ") ;半角スペースの挿入
wend
exit() ;マクロの終了
```

このマクロは、コンパイルして中間コード(マクロコマンド)に変換しなければ実行できません。コンパイルの方法、マクロライブラリへの登録、およびマクロコマンドの実行は、第3章を参照してください。

これ以降のチュートリアルは必要に応じて読み進めてください。



マクロコマンド `right` のソースファイルは、CD-ROM 内の `TUTORIAL` フォルダに収録されています。

改行だけの行を削除

機能 改行以外に文字列のない行を削除します。

マクロの内容

カレントウィンドウの先頭からカーソルを 1 行ずつ下げて、その行が改行だけの行かどうかをチェックしていきます。改行だけの場合には、行を削除し、改行だけではない場合にはカーソルをもう 1 行下に移動します。ファイル(カレントウィンドウ)の最後までこの処理を繰り返します。

1. ファイルの先頭へ移動
2. 行末に移動
3. ファイルの最後ならば終了
4. 改行だけならば行削除

作成手順

①

ファイルの先頭へ移動

カーソル位置に関係なく、ファイル全体で改行だけの行を削除するために、はじめにファイルの先頭へカーソルを移動します。

ファイルの先頭はバイト位置が 1 です。カーソル位置のバイト情報はシステム変数 @byte で取得できます。また @byte には、値を代入することができ、代入した値のバイト位置にカーソルを移動できます。ここでは、1 を代入してカーソルをファイルの先頭へ移動します。

```
*delline 改行だけの行を削除

@byte = 1 ;先頭へジャンプ
```

②

行末に移動

カーソルを行末に移動します。

これは次の処理③でカーソル位置がファイルの最後かどうかを確認するために使います。ファイルの最後を表す文字コード 0xff1a または 0xffff が必ず行頭にあるとは限りません。1 行の中に文字列があり、その後にこのコードがある場合も考えられます。そのために、いったんカーソルを行末に移動します。もちろん、その行が改行だけの場合はカーソルは移動しません。

```
*delline  改行だけの行を削除
```

```
@byte = 1          ;先頭へジャンプ  
move(" ")         ;行末に移動
```

3

ファイルの最後ならば終了

カーソルがファイルの最後まで移動したかどうかを判定します。

カーソル位置の文字コードはシステム変数 @code でわかります。

@code がファイルの最後の文字コードかどうか判定するには、関係演算子 == (equal) を使います。また、ファイルの最後の文字コードは2種類ありますので、それぞれの条件式を論理演算子 || (or) でくくります。それぞれの条件式は括弧 () でくくり、次のように記述します。



ファイルの最後を表す [EOF] コードには2種類あります。1つは「テキスト (^Zまで)」モードでファイルを新規に開くと保存時にファイルの最後に付加されるもので、コードは 0xff1a です。もう1つはデータとしては保存されない [EOF] マークで、コードは 0xffff です。



論理演算子 || (or) は、[SHIFT]+[¥] 併せて ! を2つ入力します。

```
*delline  改行だけの行を削除
```

```
@byte = 1          ;先頭へジャンプ  
move(" ")         ;行末に移動  
if(@code==0xff1a) || (@code==0xffff) ;ファイルの最後かどうか
```

カーソルがファイルの最後まで移動すると、マクロを終了します。マクロの終了はシステム関数 exit を使います。if 構文の条件式で真(ファイルの最後まで移動した)ときにマクロを終了するように記述すると次のようになります。

if 構文の終わりには endif を記述します。

```
*delline  改行だけの行を削除
```

```
@byte = 1          ;先頭へジャンプ  
move(" ")         ;行末に移動  
if(@code==0xff1a) || (@code==0xffff) ;ファイルの最後かどうか  
    exit()        ;ファイルの最後なら終了  
endif
```

4

改行だけならば行削除

次にカーソル位置(行末)がファイルの最後ではない場合の処理を記述します。

まず、カーソル位置が改行かどうかを判定します。次にカーソル位置の行に改行以外の文字列がないかどうかの判定をします。この2つの判定をして、カーソルのある行が改行だけの場合、その行を削除します。

改行かどうかは文字コードで判定します。カーソル位置の文字コードはシステム変数 @code で取得できます。改行を表す文字コードは 0x0d0a です。if 構文の条件式に @code==0x0d0a と記述します。

次にこの条件式が真(改行)のときに、その行に改行以外の文字列があるかどうかを判定します。

カーソル位置の行が改行だけならば、カーソルは必ず行頭にあります。行頭にカーソルがある場合は、カーソルの桁位置が1になります。そのため、桁位置を使って、カーソルが改行だけの行にあるかどうかを判定できます。カーソルの桁位置はシステム変数 @col で取得できます。

後から見やすいようにタブを使って段落をつけ、行頭かどうかを判定する if 構文の条件式には @col == 1 と記述します。この条件式が真(改行だけ)のときに行を削除します。

行を削除するにはシステム関数 delline を使います。

```
*delline  改行だけの行を削除

@byte = 1                                ;先頭へジャンプ
move(" ")                                 ;行末に移動
if(@code==0xff1a) || (@code==0xffff)     ;ファイルの最後かどうか
    exit()                                 ;ファイルの最後なら終了
endif
if(@code == 0x0d0a)                       ;改行かどうか
    if(@col == 1)                           ;行頭かどうかのチェック
        delline()                           ;行削除
```

カーソルの位置の行が改行だけではない場合は、次の行にカーソルを移動してその行の判定をします。

次の行を判定するためにカーソルを1行下に移動します。カーソル移動のシステム関数 move を使い、1行下に移動する d を記述します。それぞれの if 構文に endif を記述します。

*delline 改行だけの行を削除

```
@byte = 1 ;先頭へジャンプ
move(" ") ;行末に移動
if(@code==0xff1a) || (@code==0xffff) ;ファイルの最後かどうか
    exit() ;ファイルの最後なら終了
endif
if(@code == 0x0d0a) ;改行かどうか
    if(@col == 1) ;行頭かどうかのチェック
        delline() ;行削除
    else
        move(" d ") ;カーソル1つ下に移動
    endif
endif
```

以上で、改行だけの行を削除する処理はできました。ただし、このままではファイルの先頭行だけの処理です。この処理をファイルの最後まで繰り返すように記述します。

処理を繰り返すには、while ~ wend 構文を使います。カーソルを行末に移動するmove関数の前行にwhile、最後のendifの次行にwendを書き加えます。ファイルの最後までカーソルが移動したかどうかの判断はすでにif構文で行っていますので、whileの条件式はつねに処理を繰り返すように記述します。つねにwhile構文の処理をするための条件式は1です。そのため以下のようにwhile(1)と記述します。

*delline 改行だけの行を削除

```
@byte = 1 ;先頭へジャンプ
while (1)
    move(" ") ;行末に移動
    if(@code==0xff1a) || (@code==0xffff) ;ファイルの最後かどうか
        exit() ;ファイルの最後なら終了
    endif
    if(@code == 0x0d0a) ;改行かどうか
        if(@col == 1) ;行頭かどうかのチェック
            delline() ;行削除
        else
            move(" d ") ;カーソル1つ下に移動
        endif
    endif
endif
wend
```

これでファイルの最後まで処理できるようになりました。

ただし、このままではこのマクロを実行する上で、一部制限があります。ファイル中のすべての行は折り返してないこと、という制限です。折り返し桁位置を1バイトだけ超えた行があると、行頭は改行になります。つまり、上記の処理ではこの折り返し桁位置を1バイト超えたときの改行まで削除してしまいます。

そこでこの制限をなくすために、処理を追加します。

まず、グローバル単純変数 @1 を使ってフラグを立てることにします。

行末が改行ではなく、かつ桁位置が1(行頭)ではないときに @1 に1を代入します。つまり、折り返し桁位置を超えた行にカーソルが移動したときに @1 に1が代入されることとなります。@1 に1を代入し、カーソルを1行下に移動して、再度処理を続けます。これを改行かどうかを判定するif構文の偽のところ else を追加して、以下のように記述します。

```
*delline  改行だけの行を削除

@byte = 1 ;先頭へジャンプ
while (1) ;つねに処理を繰り返す
  move(" ") ;行末に移動
  if(@code==0xff1a) || (@code==0xffff) ;ファイルの最後かどうか
    exit() ;ファイルの最後なら終了
  endif
  if(@code == 0x0d0a) ;改行かどうか 【A】
    if(@col == 1) ;行頭かどうかのチェック
      delline() ;行削除
    else
      move("d") ;カーソル1つ下に移動
    endif
  else
    @1 = 1 ;フラグに1をたてる
    move("d") ;カーソル1つ下に移動
  endif
wend
exit() ;マクロの終了
```

しかし、まだこれでも不十分です。

改行かどうか判定するif構文【A】の条件式に @1 が1でない場合を追加する必要があります。また、グローバル単純変数には初期値があります。@1 は1です。仮に他のマクロコマンドを実行した後で、そのマクロコマンドが @1 を使っていた場合は、最後に代入された値が残っています。そのため、while 構文の処理に入る前に初期化する必要があります。初期化は @1 に0を代入します。

最初に初期化していると、@1 に1が代入されるのは折り返し桁位置を超えたときだけになり、それ以外はずねに0ということになります。そのため、if構文【A '】の条件式に @1 が0の場合の記述を追加します。カーソル位置が行頭で、かつ @1 が0の場合に、行を削除するこ

とになります。それぞれの条件式は括弧()でくくり、論理演算子 &&(and)で結合します。コメントも条件に合わせて「先頭とフラグのチェック」と変更します。

また、1行の処理が終わり次の行の処理に移る前に、この@1を0にしておきます【B】。【B】で0にしなければ、折り返し桁位置を超える行にカーソルが移動した後、つねに@1に1が代入された状態になってしまうためです。

最後にマクロの終了を表すシステム関数exitを記述して、このマクロは完成です。

```
*delline  改行だけの行を削除

@byte = 1                ;先頭へジャンプ
@1 = 0                   ;フラグの初期化
while (1)                ;つねに処理を繰り返す
    move(" ")            ;行末に移動
    if(@code==0xff1a) || (@code==0xffff) ;ファイルの最後かどうか
        exit()           ;ファイルの最後なら終了
    endif
    if(@code == 0x0d0a)   ;改行かどうか【A】
        if(@col == 1) && (@1 == 0) ;先頭とフラグのチェック【A'】
            delline()      ;行削除
        else
            move(" d ")    ;カーソル1つ下に移動
        endif
        @1 = 0            ;フラグの初期化【B】
    else
        @1 = 1            ;フラグに1をたてる
        move(" d ")      ;カーソル1つ下に移動
    endif
wend
exit()                   ;マクロの終了
```



マクロコマンドdellineのソースファイルは、CD-ROM内のTUTORIALフォルダに収録されています。

指定した行範囲の印刷

機能 指定した行範囲を印刷します。

マクロの内容

行範囲を指定してコピーし、新規ウィンドウに貼り付けします。プリンタを設定し、フォントと文字サイズを選択して、新規ウィンドウの内容を印刷します。印刷後、新規ウィンドウを保存しないで閉じます。

- 1.印刷する行範囲の指定
- 2.コピーと貼り付け
- 3.プリンタの設定
- 4.フォントと文字サイズの設定
- 5.印刷してファイルを閉じる

マクロ実行時の条件

- ・マクロコマンド実行時のカーソル位置が行選択の開始位置。



このチュートリアルからは1つの処理ごとに、ソースプログラムの記述を記載します。ソースプログラム全体を確認したい場合は、このチュートリアルの最後のページを参照してください。また、処理ごとに処理の内容のコメント行を記述しています。

作成手順

①

印刷する行範囲の指定

範囲選択を開始するには、システム変数 @selmode を使います。@selmode に値を代入すると、範囲選択の状態を変更できます。行単位で範囲選択するには2を代入します。

```
*lineprint 指定した行範囲の印刷

;***** 行範囲選択の指定 *****
@selmode = 2 ;行範囲選択の開始
```

「行範囲を指定してください」というメッセージをウィンドウに表示させるために、システム関数 message を使います。表示させたいメッセージ(文字列)をダブルクォーテーション " でくくり記述します。また、行の範囲選択は改行キーで終了するようにしますので、そのメッセージも合わせて次のように記述します。



メッセージはウィンドウの多目的バーに表示されます。

```

*lineprint      指定した行範囲の印刷

;***** 行範囲選択の指定 *****
  @selmode = 2                ;行範囲選択の開始
  message(" 行範囲を指定してください[ Enter ]キーで終了)")
                                ;改行コマンドで確定

```

改行コマンドが割り当てられた [Enter]キーを押すと行範囲の選択を終了するようにします。そのために、イベント待ちするシステム関数 `waitevent` を使います。

書式 `waitevent(val,vptr)`

ここではコマンドを待つ、つまり改行コマンドの入力を待つのですから、第1引数にはコマンド待ちのマクロ定数 `EVENT_COMMAND` を記述します。第2引数には、配列変数を記述します。ここではローカル配列変数 `@@str1` を使います。グローバル配列変数 (`@str1` など) を使うこともできますが、ここではローカル配列変数を使います。

`waitevent` 関数を使用する際には、`waitevent` 文の前で配列変数の要素にイベントを待つマクロ定数を指定します。そのため、はじめにローカル配列変数の第1要素 `@@str1[0]` に、改行コマンドを表すマクロ定数 `COMMAND_RET` を代入します。ここでは改行コマンドだけをイベント待ちしますので、次の要素 `@@str1[1]` には最後を表す `0` を代入します。改行コマンドは実際には [Enter]キーに割り当てられているため、[Enter]キーの入力で指定のイベントが発生することになります。[Enter]キー以外のキーに改行コマンドが割り当てられている場合には、そのキーでも指定のイベントが発生することになります。

このローカル配列変数 `@@str1` を `waitevent` 関数の第2引数に記述します。

これで、行範囲を選択できました。

```

*lineprint      指定した行範囲の印刷

;***** 行範囲選択の指定 *****
  @selmode = 2                ;行範囲選択の開始
  message(" 行範囲を指定してください[ Enter ]キーで終了)")
                                ;改行コマンドで確定

  @@str1[0] = COMMAND_RET
  @@str1[1] = 0
  waitevent(EVENT_COMMAND,@@str1)

```

2

コピーと貼り付け

選択した行範囲をコピーするにはシステム関数 `copy` を記述します。

```

;***** コピーと貼り付け *****
  copy()                      ;コピー

```

新しくウィンドウを開き、行カットバッファにコピーした内容を貼り付けます。

印刷のためのシステム関数 outprinter はファイル全体を印刷します。そのため、新しく印刷専用のウィンドウを開きます。システム関数 newopen を使います。貼り付けはシステム関数 paste を使い、引数には行カットバッファの貼り付けを意味する 0 を記述します。ここで開いたウィンドウは印刷した後に閉じます。その処理は後述します。



MIFES は行単位でコピーや切り取りを実行すると、行カットバッファという MIFES 独自のファイルに格納します。詳しくはユーザーズマニュアルまたはヘルプを参照してください。

```
;***** コピーと貼り付け *****
copy()                ;コピー
newopen()             ;新規ウィンドウを開く
paste(0)              ;行カットバッファを貼り付ける
```

3

プリンタの設定

MIL/W 言語には「プリンタ設定」ダイアログボックスを表示させるシステム関数はありません。そのため、機能番号と、機能番号を実行するシステム関数 execcmd を使います。プリンタ設定の機能番号 146 を次のように記述します。

```
;***** プリンタの設定 *****
execmd(146)           ;プリンタの設定
```



このマクロでは「プリンタ設定」ダイアログボックスを表示しますが、つねに同じプリンタを使う場合は、システム関数 setprinter を使って特定のプリンタを設定してください。

4

フォントと文字サイズの設定

フォントを選択するためのリストボックスを表示させます。リストボックスを表示させるにはシステム関数 listbox を使います。

```
書式 listbox(ptr,val,vptr)
ptr    : タイトル文字列
val    : デフォルト選択項目
vptr   : 項目文字列の並び
```

リストボックスの内容は第 3 引数に指定します。まず先に第 3 引数の準備をします。システム関数 strlist を使ってリストボックスに表示させる文字列(項目)を指定します。

```
書式 strlist(vptr,str,,,str)
```

ここでは、グローバル配列変数 @str1 に文字列(項目)を指定します。strlist 関数の第 2 引数以降はダブルクォーテーション “ でくくり、文字列を記述します。

リストボックスに表示できる項目は最大30個です。ここでは「MS ゴシック」と「MS 明朝」の2つのフォント名を記述します。

listbox 関数の第1引数には、リストボックスのタイトルになる文字列をダブルクォーテーション “ ” でくり記述します。第2引数にはデフォルトの項目を記述します。ここではMS ゴシックをデフォルトにしますので、0と記述します。第3引数には、strlist 関数で準備しておいた @str1 を記述します。

また、listbox 関数の関数値は、選択した項目の番号を返します。この関数値をグローバル単変数 @1 に代入します。項目の番号は0から始まります。仮に最大の30項目ある場合は、0～29までの番号をそれぞれ返します。

ここでは@1に「MS ゴシック」を選択したときは0、「MS 明朝」を選択したときには1を返します。

```
;***** フォントと文字サイズの指定 *****
strlist(@str1, "MS ゴシック", "MS 明朝")
@1 = listbox("フォントを指定してください", 0, @str1)
```

@1を条件式にして、switch 構文でグローバル配列変数 @str1 にフォント名を代入します。フォント名を代入した @str1 は、印刷するときに指定するシステム関数 outprinter の引数として使うための準備です。

配列変数に文字列を代入するにはシステム関数 strcpy を使います。代入したい文字列をダブルクォーテーション “ ” でくり第2引数として記述します。

フォント名を @str1 に代入した後、switch 構文を抜けて次の処理に移すために、break 文を記述します。break 文は endsw 文の次の処理に移ります。

```
;***** フォントの指定 *****
strlist(@str1, "MS ゴシック", "MS 明朝")
@1 = listbox("フォントを指定してください", 0, @str1)
switch (@1)
  case 0
    strcpy(@str1, "MS ゴシック")
    break
  case 1
    strcpy(@str1, "MS 明朝")
    break
endsw
```

フォント名を選択するリストボックスで選択を中止する([取消] ボタンをクリックする)と、listbox 関数の関数値には-1が、項目の文字列(フォント名)に誤りがあるときは-3が返されます。

[取消] ボタンをクリックするか、またはフォント名に誤りがあるときは、「デバイス依存フォント」で印刷します。if 構文の条件式は @1 が負の数の場合とし、確認のメッセージボックスを表示させます。メッセージボックスを表示させるにはシステム関数 messagebox を使います。

書式 `messagebox(ptr1,ptr2,val)`

`ptr1` : 表示するメッセージ

`ptr2` : メッセージボックスのタイトル

`val` : 次のマクロ定数のうちのいずれかを指定する

`MB_OK`

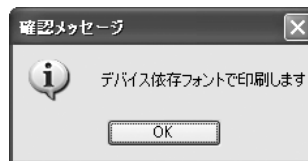
`MB_OKCANCEL`

`MB_YESNO`

`MB_YESNOCANCEL`

`MB_ABORTRETRYIGNORE`

ここでは、以下のように指定します。第3引数の`MB_ICONINFORMATION`はメッセージボックスにインフォメーションマーク「i」も合わせて表示させるために、演算子のビット論理和`|`で結んでいます。このメッセージボックスは以下のように表示されます。



```
if(@1<0) ; 取消を選択したら
    messagebox(" デバイス依存フォントで印刷します ",
               " 確認メッセージ",MB_OK | MB_ICONINFORMATION)
```



メモ

論理演算子`|`は、`[SHIFT]+[¥] 併ー`で入力します。

フォント名を正しく選択した場合、次に印刷する文字サイズの指定をします。

上記のif構文のelseで文字サイズを指定します。

文字サイズもリストボックスを使って選択できるように、`listbox`関数および`strlist`関数を使います。



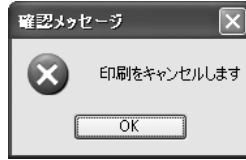
参照

`listbox`関数、`strlist`関数は「フォントの指定」で説明しています。

グローバル単純変数`@2`に`listbox`関数の関数値を代入します。この関数には、選択を中止した([取消]ボタンをクリックするときには`-1`が、および項目の文字列(文字サイズ)に誤りがあるときは、`-3`が返されます。

[取消]ボタンをクリックするか、または文字サイズに誤りがあるときは、印刷をキャンセルします。if構文の条件式は`@2`が負の数の場合とし、確認のメッセージボックスを表示させます。メッセージボックスを表示させるにはシステム関数`messagebox`を使います。

ここでは、次のように指定します。第3引数の`MB_ICONSTOP`はメッセージボックスにストップマーク「X」も合わせて表示させるために、演算子のビット論理和`|`で結んでいます。このメッセージボックスは次のように表示されます。



印刷をキャンセル後、ウィンドウを閉じて、マクロを終了します。
マクロを終了する前にウィンドウを閉じなければ、コピーした内容を貼り付けたウィンドウが残ったままになってしまいます。そのため、カレントウィンドウを閉じるシステム関数 `close` を使います。マクロを終了するには、システム関数 `exit` を使って以下のように記述します。

```
else
;***** 文字サイズの指定 *****
    strlist(@str2, "8", "10", "12", "18", "24", "36")
    @2 = listbox("ポイント数を指定してください", 0, @str2)
    if (@2 < 0)                ;取消を選択したら
        messagebox("印刷をキャンセルします", "確認メッセージ",
                    MB_OK|MB_ICONSTOP)
        close()                ;ファイルを閉じる
        exit()                 ;マクロの終了
    endif
```

印刷するときに指定するシステム関数 `outprinter` の第 1 引数の準備をします。
グローバル単純変数 `@3` に次の値を代入します。

0x01017300 と代入すると、以下の意味になります。

文字サイズ	: デバイス依存
行の間隔	: 通常間隔
段組印刷	: 1 段
段と段の間	: 空白
用紙 1 枚に印刷するページ数	: 1 ページ
行番号の付加	: 行わない
禁則処理	: 行う
インテリジェント改ページ	: 行わない
0 と O を区別した印刷	: 行う
英文ワードラップ	: 行わない
罫線接続処理	: 行う
折り返し位置で改行	: 行わない
ヘッダの位置	: 右端
フッタ位置	: 右端
ヘッダ/フッタ印刷位置	: 余白の上下端

この第 1 引数の値は必要に応じて変更してください。

switch 構文を使い、@2 を条件式にして、@3 に文字サイズを加算します。最初に代入した @3 の値では文字サイズにデバイス依存(00)を設定しています。そのため、リストボックスで選択した文字サイズを代入演算子 += を使い、@3 に代入します。+= は左辺の値と右辺の値を加算して、その結果は左辺 (@3) に代入する演算子です。

@3 に文字サイズを加算して代入すると、switch 構文を抜けて次の処理に移すために、break 文を記述します。break 文は endsw 文の次の処理に移ります。

```
@3 = 0x01017300
switch (@2)
  case 0
    @3 += 8
    break
  case 1
    @3 += 10
    break
  case 2
    @3 += 12
    break
  case 3
    @3 += 18
    break
  case 4
    @3 += 24
    break
  case 5
    @3 += 36
    break
endsw
endif
```

5 印刷して、ウィンドウを閉じる

印刷するにはシステム関数 outprinter を使います。

書式 outprinter(val1, val2, val3, val4, val5, ptr1, ptr2, ptr3)

- val1 : フラグ
- val2 : 左側余白(mm)
- val3 : 右側余白(mm)
- val4 : 上側余白(mm)
- val5 : 下側余白(mm)
- ptr1 : ヘッダ文字列
- ptr2 : フッタ文字列
- ptr3 : フォント名

ヘルプでoutprinterを参照してください。

この第1引数(フラグ)には次の意味があります。

ビット0～7	: 文字サイズ	(0: デバイス依存フォント, 1: 行桁数で指定(str3を指定))
ビット8～9	: フッタ位置	(0: なし, 1: 左端, 2: 中央, 3: 右端)
ビット10	: 未使用	(0)
ビット11	: ヘッダ/フッタの印刷位置	(0: 余白の上下端, 1: 余白の中央)
ビット12～13	: ヘッダ位置	(0: なし, 1: 左端, 2: 中央, 3: 右端)
ビット14	: 0とOを区別した印刷	(0: 区別しない, 1: 区別して印刷)
ビット15	: キーワードの明示	(0: 明示しない, 1: 明示する)
ビット16	: 罫線接続処理	(0: 行わない, 1: 行う)
ビット17～18	: 行の間隔	(0: 通常間隔, 1: 1.5倍, 2: 2倍, 3: 3倍)
ビット19	: インテリジェント改ページ	(0: 行わない, 1: 行う)
ビット20	: 行番号	(0: 付加しない, 1: 付加する)
ビット21～23	: 用紙1枚に印刷するページ数	(0: 1ページ, 1: 4ページ, 2: 横2ページ, 3: 縦2ページ, 4: 9ページ, 5: 16ページ, 6: 25ページ)
ビット24	: 禁則処理	(0: 行わない, 1: 行う)
ビット25～26	: 段組み印刷	(0: 1段組み, 1: 2段組み, 2: 3段組み)
ビット27	: 画面上の折り返し位置での改行	(0: 改行しない, 1: 改行する)
ビット28	: 英文ワードラップ	(0: 行わない, 1: 行う)
ビット29～30	: 段と段の間	(0: 空白, 1: 単線, 2: 二重線)
ビット31	: 印刷禁止フラグ	(0: 印刷実行, 1: パラメータの設定のみ)

すでに第1引数(文字サイズ)および第8引数(フォント名)は、グローバル変数でそれぞれ@3、@str1と準備していますので以下のように記述します。また第2～7引数は余白が左側15mm、右側0mm、上側15mm、下側15mm、ヘッダ文字列に日付、時間およびファイル名、フッタ文字列にはページ数を設定しています。設定は必要に応じて変更してください。

印刷後、ウィンドウを閉じて、マクロを終了します。

ウィンドウを閉じるには、システム関数closeを使います。

マクロの終了を表すシステム関数exitを記述して、このマクロは完成です。

```
;***** 印刷 *****
outprinter(@3,15,0,15,15, "%y年%m月%d日%t(%F)",
"Page %p",@str1)
close()          ;ウィンドウを閉じる
exit()          ;マクロの終了
```

ソースプログラムは以下のようになります。

```
*lineprint      指定した行範囲の印刷

;***** 行範囲選択の指定 *****
@selmode = 2          ;行範囲選択の開始
message(" 行範囲を指定してください([ Enter ]キーで終了)") ;改行キーで確定
@@str1[0] = COMMAND_RET
@@str1[1] = 0
waitevent(EVENT_COMMAND,@@str1)

;***** コピーと貼り付け *****
copy()              ;コピー
newopen()           ;新規ウィンドウを開く
paste(0)            ;行カットバッファを貼り付ける

;***** プリンタの設定 *****
execmd(146)         ;プリンタの設定

;***** フォントと文字サイズの指定 *****
strlist(@str1, "MS ゴシック", "MS 明朝")
@1 = listbox(" フォントを指定してください",0,@str1)
switch (@1)
  case 0
    strcpy(@str1, "MS ゴシック")
    break
  case 1
    strcpy(@str1, "MS 明朝")
    break
endsw
if (@1 < 0);取消を選択したら
  messagebox(" デバイス依存フォントで印刷します ", "確認メッセージ",
    MB_OK|MB_ICONINFORMATION)
else
  strlist(@str2, "8", "10", "12", "18", "24", "36")
  @2 = listbox(" ポイント数を指定してください",0,@str2)
  if (@2 < 0)          ;取消を選択したら
    messagebox(" 印刷をキャンセルします ", "確認メッセージ",
      MB_OK|MB_ICONSTOP)
    close()           ;ウィンドウを閉じる
    exit()            ;マクロの終了
  endif
```

```

@3 = 0x01017300
switch (@2)
  case 0
    @3 += 8
    break
  case 1
    @3 += 10
    break
  case 2
    @3 += 12
    break
  case 3
    @3 += 18
    break
  case 4
    @3 += 24
    break
  case 5
    @3 += 36
    break
endsw
endif

;***** 印刷 *****
outprinter(@3,15,0,15,15, %y年%m月%d日%t(%F) "; Page%p ";@str1)

close()          ;ウィンドウを閉じる
exit()           ;マクロの終了

```



MIFESには「行カットバッファの印刷」のキーボードマクロが標準でライブラリに登録されています。このキーボードマクロは、すでに行範囲指定とコピーを終えたことを前提に行カットバッファを印刷しています。

【マクロ(M)】-【キーボードマクロ(K)】-「ライブラリから取出(G)」で実行できます。



マクロコマンドlineprintのソースファイルは、CD-ROM内のTUTORIALフォルダに収録されています。

範囲指定内の文字列の置換

機能 指定した範囲内で任意の文字列の置換、タブを半角スペースに置換、または括弧の全角を半角に置換します。

マクロの内容

範囲を指定します。リストボックスの中から置換するメニューを選択し実行します。

1. 置換する範囲の指定
2. メニューの選択
3. 任意の文字列の置換
4. タブを半角スペースに置換
5. 括弧の全角を半角に置換
6. マクロの終了

マクロ実行条件

- ・ マクロコマンド実行時のカーソル位置が範囲選択の開始位置。

作成手順

①

置換する範囲の指定

範囲指定を開始するバイト位置を取得します。カーソルのバイト位置はシステム変数 @byte で取得できます。この @byte をグローバル単純変数 @1 に代入します。

また、開始位置の論理行番号も取得します。カーソル位置の論理行番号はシステム変数 @num で取得できます。この @num をグローバル単純変数 @2 に代入します。

範囲選択を開始するには、システム変数 @selmode を使います。@selmode に値を代入すると、範囲選択の状態を変更できます。文字列単位で範囲選択するには3を代入します。

```
*replace 指定範囲内の文字列置換

;***** 文字列の範囲指定 *****
@1 = @byte      ;開始するバイト位置の取得
@2 = @num       ;開始位置の論理行番号の取得
@selmode = 3    ;文字列選択の開始
```

「範囲を指定してください」というメッセージをウィンドウに表示させるために、システム関数 message を使います。表示させたいメッセージ(文字列)をダブルクォーテーション“ ”でくり記述します。また、文字列の範囲選択は改行コマンドで終了するようにしますので、そのメッセージも合わせて次のように記述します。



メッセージはウィンドウの多目的バーに表示されます。

```
message(" 範囲を指定してください([ Enter ]キーで終了)") ;改行コマンドで確定
```

改行コマンドが割り当てられた[Enter]キーを押すと文字列の範囲の選択を終了するようにします。そのために、イベント待ちするシステム関数 `waitevent` を使います。

```
書式 waitevent(val,vptr)
```

ここではコマンドを待つ、つまり改行コマンドの入力を待つのですから、第1引数にはコマンド待ちのマクロ定数 `EVENT_COMMAND` を記述します。第2引数には、配列変数を記述します。ここではローカル配列変数 `@@str1` を使います。グローバル配列変数 (`@str1` など) を使うこともできますが、ここではローカル配列変数を使います。

`waitevent` 関数を使用する際には、`waitevent` 文の前で配列変数の要素にイベントを待つマクロ定数を指定します。そのため、はじめにローカル配列変数の第1要素 `@@str1[0]` に、改行コマンドを表すマクロ定数 `COMMAND_RET` を代入します。ここでは改行コマンドだけをイベント待ちしますので、次の要素 `@@str1[1]` には最後を表す `0` を代入します。改行コマンドは実際には[Enter]キーに割り当てられているため、ユーザーの [Enter]キーの入力で指定のイベントが発生したことになります。

このローカル配列変数 `@@str1` を `waitevent` 関数の第2引数に記述します。

これで、文字列範囲の選択ができました。

```
@@str1[0] = COMMAND_RET
@@str1[1] = 0
waitevent(EVENT_COMMAND,@@str1)
```

範囲選択を終了するために、システム変数 `@selmode` に `0` を代入します。

現在ソースプログラム上ではカーソル位置は範囲選択の終了位置に移動しています。そのため、終了位置の論理行番号 (`@num`) をグローバル単純変数 `@3` に代入します。

```
@selmode = 0 ;文字列選択の終了
@3 = @num ;終了の論理行番号の取得
```

2

メニューの選択

ここで、カーソルを範囲指定の開始位置に戻すため、上記でグローバル単純変数 `@1` に代入した開始位置をシステム変数 `@byte` に代入します。`@byte` は値を代入すると、代入したバイト位置にカーソルをジャンプできます。

システム変数によっては、代入できないものもあります。詳しくは第4章のそれぞれのシステム変数の説明を参照してください。



```
***** ;メニューの選択 *****
@byte = @1 ;開始位置へジャンプ
```

メニューの選択をするためのリストボックスを表示させます。リストボックスを表示させるにはシステム関数 `listbox` を使います。

```
書式 listbox(ptr,val,vptr)
      ptr    : タイトル文字列
      val    : デフォルト選択項目
      vptr   : 項目文字列の並び
```

リストボックスの内容は第3引数に指定します。

まず先に第3引数の準備をします。システム関数 `strlist` を使ってリストボックスに表示させる文字列(項目)を指定します。

```
書式 strlist(vptr,str,,,str)
```

ここでは、グローバル配列変数 `@str1` に文字列(項目)を指定します。`strlist` 関数の第2引数以降はダブルクォーテーション “ でくくり、文字列を記述します。

リストボックスに表示できる項目は最大30個です。ここでは「任意の文字列」、「TAB 半角スペース」および「括弧 () [] { } 全角 半角」の3つのメニュー名を記述します。

`listbox` 関数の第1引数には、リストボックスのタイトルになる文字列を、ダブルクォーテーション “ でくくり記述します。第2引数にはデフォルトの項目を記述します。ここでは「任意の文字列」をデフォルトにしますので、0と記述します。第3引数には、`strlist` 関数で準備しておいた `@str1` を記述します。

また、`listbox` 関数の関数値には、選択した項目の番号を返します。この関数値をグローバル単変数 `@4` に代入します。項目の番号は0から始まります。仮に最大の30項目ある場合は、0～29までの番号をそれぞれ返します。

ここでは `@4` に0、1または2を返します。「任意の文字列」を選択したときは0、「TAB 半角スペース」を選択したときは1、「括弧 () [] { } 全角 半角」を選択したときには2を返します。

```
strlist(@str1, "任意の文字列",
        "TAB 半角スペース", "括弧 ( ) [ ] { } 全角 半角")
@4 = listbox("どの置換をしますか",0,@str1)
```

`@4` を条件式にして、`switch` 構文で、それぞれのメニューにジャンプさせます。

これ以降は各メニューの記述を説明します。

各メニューの記述の前に以下のように `case` 文と `endsw` 文を記述します。また、各 `case` 文の処理が終わると `switch` 構文を抜け出させるために、先に `break` 文も記述しておきます。

これ以降の各メニューの処理は下記の `case` 文と `break` 文の間に記述します。

ソースプログラムの構造が深くなると、`endsw` 文を記述し忘れることがあります。`endsw` 文がないとコンパイルエラーにもなりますので、`switch` 構文と合わせて先に記述することをおすすめします。



```

switch (@4)
  case 0
    break
  case 1
    break
  case 2
    break
endsw

```

3

任意の文字列の置換

case 0 と break 文の間に記述していきます。

文字列置換の方法をリストボックスを使って選択できるように、listbox 関数および strlist 関数を使います。

listbox 関数の関数値をグローバル単純変数 @5 に代入します。@5 は文字列の置換のシステム関数 replace の第 2 引数に使います。replace 関数については後で説明します。



listbox 関数、strlist 関数は「メニューの選択」で説明しています。

```

;***** 文字列の置換 *****
  case 0
    strlist(@str1, "通常検索:大文字・小文字区別",
            "通常検索:大文字・小文字同一視",
            "ワイルドカード検索:大文字・小文字区別",
            "ワイルドカード検索:大文字・小文字同一視",
            "正規表現", "あいまい検索")
    @5 = listbox(" 検索方法を選択してください",0,@str1)

```

次に置換する旧文字列と、新文字列をそれぞれ入力するウィンドウを表示させるため、システム関数 input を使います。

書式 input(vptr,ptr)

第 1 引数には配列変数を指定します。ここではローカル配列変数 @@str2 に旧文字列、@@str3 に新文字列のそれぞれを指定します。

第 2 引数はウィンドウに表示させるメッセージ(タイトル)をダブルクォーテーション “ ” でくり記述します。

文字列の置換にはシステム関数 replace を使います。

書式 replace(val1,val2,ptr1,val3,val4,ptr2)

val1 : 置換時の確認操作 (0:あり 1:なし)

val2 : 旧文字列の検索方法 (0 ~ 5)

ptr1 : 旧文字列(検索文字列)

val3 : 置換開始論理行番号(1 ~)

val4 : 置換終了論理行番号 (~ MAX_NUMBER)

ptr2 : 新文字列 (置換文字列)

ここでは確認なしにしますので第1引数には1を記述します。第2、3、5および6引数はすべて変数に代入済みですので、それぞれを以下のように記述します。第4引数は置換の開始位置を指定します。「メニューの選択」で先にカーソルを開始位置にジャンプしていますので、ここでは現在のカーソル位置からはじめるため、0を指定します。

文字列の置換を終えると、カーソルを置換開始位置に戻すため、グローバル単純変数 @1 に代入した開始位置をシステム変数 @byte に代入します。

```
input(@@str2, "旧文字列を入力してください")
input(@@str3, "新文字列を入力してください")
replace(1,@5,@@str2,0,@3,@@str3) ;文字列の置換(確認なし)
@byte = @1 ;開始位置へジャンプ
break
```

4

タブを半角スペースに置換

case 1 と break 文の間に記述していきます。

タブを半角スペースに置き換えるには、まずタブを検索し、検索したタブの終了位置を取得します。タブを削除し、取得しておいたタブの終了位置まで半角スペースを挿入します。この処理を指定した範囲内繰り返します。

処理を繰り返しますので、while ~ wend 構文を使います。指定範囲まで処理が終了したかどうかの判定は while 構文の中で処理しますので、while 構文の条件式にはつねに処理を繰り返すように1を記述します。

```
;***** TAB 半角スペース *****
case 1
while (1)
wend
```

タブを検索するにはシステム関数 search を使います。

```
書式 search(val1,val2,ptr)
val1 : 検索する方向 ( 0 ~ 3 )
val2 : 検索方法 ( 0 ~ 5 )
ptr : 検索する文字列
```

ここでは、カーソル位置から 方向に検索するため第1引数には0、第2引数には「通常検索の大文字・小文字区別」の0を記述します。第3引数には、タブを表すメタ文字 ¥t をダブルクォーテーション “ でくり記述します。

また、search 関数の関数値をグローバル単純変数 @5 に代入するよう記述します。この関数値は見つかった文字列の文字数を返します。見つからなかった場合には 0 を返し、指定に誤

りがある場合には負の数を返します。@5を使ってタブが検索できたかどうかをチェックします。検索できなかったときにはwhile構文を抜ける(処理を終了する)ので、break文を記述します。

検索できたかどうかのチェックより先に、指定した範囲内かどうかをチェックする必要があります。すでに範囲指定の終了位置を@3に取得していますのでこれを使います。カーソル位置の論理行番号を表すシステム変数@numを使って、指定範囲の終了位置に達したかどうかを判定します。終了位置に達すれば、処理を終了するため、break文を記述します。

```
;***** TAB 半角スペース *****
case 1
  while (1)
    @5 = search(0,0, "%t ")
    if(@3 <= @num)           ;終了位置のチェック
      break
    endif
    if(@5 <= 0)             ;検索できたかどうか
      break
    endif
```

タブが見つかったときの処理を記述します。

タブの終了位置を取得するため、まずカーソルを右に1つ移動します。カーソル移動にはシステム関数moveを使います。右に移動するときにはrを使います。rをダブルクォーテーション“ ”でくって記述します。タブの終了位置にカーソルを移動したので、このカーソルの桁位置を取得します。

カーソル位置の桁位置はシステム変数@colで取得できます。@colをグローバル単純変数@6に代入します。

次にカーソル位置をタブの開始位置に戻すため、move関数を使い、左に1つカーソルを移動します。左に移動するにはl(エル)を使います。

```
else           ;タブが見つかったら
  move(" r ")  ;カーソル右に1つ移動
  @6 = @col     ;タブ終了桁位置の取得
  move(" l ")   ;カーソル左に1つ戻す
```

タブを削除します。文字列を削除するシステム関数delcharを使います。delchar関数は削除する文字数を指定しますので、ここでは1を記述します。

半角スペースをタブの桁数分挿入するには、システム関数insstrを使います。半角スペースはメタ文字で%sです。ダブルクォーテーションでくくり、以下のようにパラメータを記述します。この処理をタブの終了桁位置まで繰り返すため、while ~ wend構文を記述します。条件式にはカーソルの桁位置が先に取得したタブの終了桁位置(@6)まで繰り返すように記述します。

```

delchar(1)
while (@col < @6) ;タブの終了桁位置まで繰り返す
    insstr("¥s ") ;半角スペースの挿入
wend
endif
wend

```

タブを半角スペースに置き換えた後にカーソルを置換開始位置に戻すため、最初にグローバル単純変数 @1 に代入した開始位置をシステム変数 @byte に代入します。@byte は値を代入すると、代入したバイト位置にカーソルをジャンプできます。

以上でタブを置き換える処理は完了していますが、このまま処理を終了すると、次のタブを反転表示している状態になります。そのため、画面の再描画をします。

再描画するにはシステム変数 @disp を使い、3 を代入します。

```

@byte = @1 ;開始位置へジャンプ
@disp = 3 ;再描画
break

```

5

括弧の全角を半角に置換

case 2 と break 文の間に記述していきます。

括弧の置換は、文字列の置換のシステム関数 replace を使います。

書式 replace(val1,val2,ptr1,val3,val4,ptr2)

第 1 引数には、置換時の確認なしの 1 を指定し、第 2 引数には「通常検索の大文字・小文字の区別」の 0 を指定します。第 4、5 引数には最初に取得した置換開始位置 @2 と終了位置 @3 を記述します。第 3、6 引数には旧文字列と新文字列をダブルクォーテーション “ ” でくり記述します。

replace 関数については「3.任意の文字列の置換」で説明しています。

```

;***** 括弧 全角 半角 *****
case 2
    replace(1,0,“(”,@2,@3,“(“) ;( ( (確認なし)
    replace(1,0,”)”,@2,@3,”)”) ;) ) (確認なし)
    replace(1,0,“(”,@2,@3,“(“) ;[ [ (確認なし)
    replace(1,0,”)”,@2,@3,”)”) ;] ] (確認なし)
    replace(1,0,“(”,@2,@3,“(“) ;{ { (確認なし)
    replace(1,0,”)”,@2,@3,”)”) ;} } (確認なし)
break

```



参照

6

マクロの終了

最後にマクロの終了を表すシステム関数exitを記述して、このマクロは完成です。

```
endsw
exit()      ;マクロの終了
```

ソースプログラムは以下のようになります。

```
*replace   指定範囲内の文字列置換

;***** 文字列の範囲指定 *****
@1 = @byte           ;開始するバイト位置の取得
@2 = @num            ;開始位置の論理行番号の取得
@selmode = 3        ;文字列選択の開始
message(" 範囲を指定してください([ Enter ]で終了)") ;改行キーで確定
@@str1[0] = COMMAND_RET
@@str1[1] = 0
waitevent(EVENT_COMMAND,@@str1)
@selmode = 0        ;文字列選択の終了
@3 = @num           ;終了の論理行番号の取得

;***** メニューの選択 *****
@byte = @1          ;開始位置へジャンプ
strlist(@str1,"任意の文字列","TAB 半角スペース",
        "括弧()[ ]{ }全角 半角")
@4 = listbox(" どの置換をしますか",0,@str1)
switch (@4)

;***** 文字列の置換 *****
case 0
    strlist(@str1,"通常検索:大文字・小文字区別",
            "通常検索:大文字・小文字同一視",
            "ワイルドカード検索:大文字・小文字区別",
            "ワイルドカード検索:大文字・小文字同一視",
            "正規表現","あいまい検索")
    @5 = listbox(" 検索方法を選択してください",0,@str1)
    input(@@str2,"旧文字列を入力してください")
    input(@@str3,"新文字列を入力してください")
    replace(1,@5,@@str2,0,@3,@@str3);文字列の置換(確認なし)
    @byte = @1      ;開始位置へジャンプ
    break
```



```

;***** TAB 半角スペース *****
case 1
  while (1)
    @5 = search(0,0, "%t ") ;タブの検索
    if(@3 <= @num) ;終了位置のチェック
      break
    endif
    if(@5 <= 0) ;検索できたかどうか
      break
    else ;タブが見つかったら
      move(" r ") ;カーソル右に1つ移動
      @6 = @col ;タブ終了桁位置の取得
      move(" l ") ;カーソル左に1つ戻す
      delchar(1)
      while (@col < @6) ;タブの終了桁位置まで繰り返す
        insstr("%s ") ;半角スペースの挿入
      wend
    endif
  wend
  @byte = @1 ;開始位置へジャンプ
  @disp = 3 ;再描画
  break

;***** 括弧 全角 半角 *****
case 2
  replace(1,0, "{ ",@2,@3, "(" ) ;{ ( (確認なし)
  replace(1,0, "}")",@2,@3, ")" ) ;} ) (確認なし)
  replace(1,0, "[ ",@2,@3, "[" ) ;[ [ (確認なし)
  replace(1,0, "]"",@2,@3, "]" ) ;] ] (確認なし)
  replace(1,0, "{ ",@2,@3, "{" ) ;{ { (確認なし)
  replace(1,0, "}"",@2,@3, "}" ) ;} } (確認なし)
  break
endsw
exit() ;マクロの終了

```



マクロコマンド `replace` のソースファイルは、CD-ROM 内の TUTORIAL フォルダに収録されています。

C 言語関数内の文字列の置換

機能 括弧 { を検索し、{ と対になる括弧 } までの範囲で、文字列を置き換えます。

マクロの内容

カーソル位置から括弧 { を 方向に検索し、その { と対になる括弧 } を検索します。{ と } の間の範囲内の指定の文字列を置き換えます。

- 1.括弧 { の検索
- 2.対になる括弧 } の検索
- 3.任意の文字列の置換
- 4.置換した文字数の表示
- 5.マクロの終了

マクロ実行時の条件

- ・マクロコマンド実行時のカーソル位置が括弧 { より前にあること。
- ・{ と同じ行に対になる } がないこと。

作成手順

①

括弧 { の検索

括弧 { を検索します。

括弧の検索にはシステム関数 `search` を使います。

```
書式 search(val1,val2,ptr)
      val1 : 検索する方向(0 ~ 3)
      val2 : 検索方法 (0 ~ 5)
      ptr  : 検索する文字列
```

ここでは、カーソル位置から 方向に検索するため第1引数には0、第2引数には「通常検索の大文字・小文字区別」の0を記述します。第3引数には括弧 { をダブルクォーテーション “ でくり記述します。

文字列を置き換える開始位置を取得します。カーソル位置の論理行番号はシステム変数 `@num` を使います。グローバル単純変数 `@1` に代入し、カーソル位置の論理行番号を取得します。

```
*creplace C 言語関数内の文字列置換
```

```
search(0,0,"{") ;{の検索
@1 = @num       ;開始位置の論理行番号の取得
```

2

対になる括弧 } の検索

検索した括弧 { と対になる括弧を検索します。対になる括弧の検索はシステム関数 `parenthesis` を使います。

書式 `parenthesis(val)`
`val` : 検索する論理行の行数

ここでは、検索範囲をファイルの最後まで有効にするためにファイルの最大論理行番号を表すマクロ定数 `MAX_NUMBER` を記述します。

対になる括弧を検索できたら、そのカーソル位置の論理行番号を取得します。

これは、後で文字列を置き換えるときの終了位置として使います。カーソル位置の論理行番号を取得するにはシステム変数 `@num` を使います。`@num` をグローバル単純変数 `@2` に代入します。

対になる括弧が見つからなかった場合は、マクロを終了します。

`parenthesis` 関数は対になる括弧が見つからなかった場合、カーソルを移動しません。

これを利用してマクロを終了する条件とします。検索開始位置は `@1` にすでに取得しているので、`@1` と `@2` (終了位置の論理行番号) が等しければ、マクロを終了します。if 構文の条件式には関係演算子 `==` (equal) を使い、次のように記述します。

マクロの終了はシステム関数 `exit` を使います。

```
parenthesis(MAX_NUMBER)    ;{と対になる括弧 } の検索
@2 = @num                  ;終了位置の論理行番号の取得
if(@1 == @2)
    messagebox(" 対になる括弧がありません ", "エラー", MB_OK)
    exit()                  ;マクロの終了
endif
```

3

任意の文字列の置換

次に置換する旧文字列と、新文字列をそれぞれ入力するウィンドウを表示させるため、システム関数 `input` を使います。

書式 `input(vptr,ptr)`

第 1 引数には配列変数を指定します。ここではローカル配列変数 `@str1` に旧文字列、`@str2` に新文字列のそれぞれを指定します。

第 2 引数はウィンドウに表示させるメッセージ(タイトル)をダブルクォーテーション “ でくり記述します。

```
input(@str1, "旧文字列を入力してください")
input(@str2, "新文字列を入力してください")
```

旧文字列で指定した文字列 `@str1` を語単位で置換するために、`@str1` を加工して `@str3` に代入します。後で使う文字列を置き換えるシステム関数 `replace` の第 3 引数として使うためにこの加工が必要です。ここでは、`replace` 関数で語単位で文字列を置き換えるために、正規表現

検索を指定し、旧文字列にメタ文字を使います。このときに使うメタ文字は # (語の先頭)と / (語の最後)です。この2つのメタ文字を旧文字列、ここでは @str1 の前後に付けて、@str3 に代入します。文字列を加工し配列変数に代入するにはシステム関数 printf を使います。

書式 printf(vptr,str,arg)

printf 関数は第3引数を第2引数で指定するように加工し、第1引数に代入します。ここでは「#@str1/」と加工したい訳ですから、第3引数 @str1 をそのまま出力するように %s を指定します。



旧文字列で指定した文字列を語単位で置換というのは、仮に次の例で「Abc」という文字列を「Def」という文字列に置き換えようとしたときに、以下のような結果になるようにするということです。

例	置換前	置換後	
	Abc,ppp,mil	Def,ppp,mil	置き換わる
	mil Abc ppp	mil Def ppp	置き換わる
	Abcppp	Abcppp	置き換わらない
	MILAbc	MILAbc	置き換わらない

```
printf(@str3, "%s/ ",@str1)
```

以上で、置換の準備はできました。

replace 関数を使い、文字列を置き換えます。

書式 replace(val1,val2,ptr1,val3,val4,ptr2)

val1 : 置換時の確認操作 (0:あり 1:なし)

val2 : 旧文字列の検索方法 (0 ~ 5)

ptr1 : 旧文字列 (検索文字列)

val3 : 置換開始論理行番号 (1 ~)

val4 : 置換終了論理行番号 (~ MAX_NUMBER)

ptr2 : 新文字列 (置換文字列)

ここでは「確認なし」にしますので第1引数には1、第2引数には正規表現検索の2をそれぞれ記述します。第3引数には、上記で printf 関数で加工した @str3 を指定します。第4、5 および6引数もすでに変数に代入済みですので、それぞれを以下のように記述します。

次に置き換えた文字列の合計数をメッセージボックスで表示させます。replace 関数の関数値は置き換えた文字列の数を返しますので、これをグローバル単純変数 @3 に代入し使います。

また、文字列を置き換えし終わると、カーソルを置換開始位置に戻します。グローバル単純変数 @1 に代入した開始位置をシステム変数 @num に代入して、開始位置にカーソルを移動します。

```
@3=replace(1,4,@str3,@1,@2,@str2) ;文字列の置換(確認なし)
@num = @1 ;開始位置へジャンプ
```



ここでは、語単位で検索するために正規表現検索を使いましたが、replace関数の第2引数(検索方法)に「語単位で検索する」オプションを追加することもできます。詳しくはヘルプを参照してください。

4

置換した文字数の表示

メッセージボックスに表示させるには配列変数でなければなりません。そのため、上記で代入した @3 を sprintf 関数を使い加工します。「 個の文字列を置換しました」と表示させるために、@3 を 10 進数の数字に変換する %d を使い、グローバル配列変数 @str4 に代入します。

```
printf(@str4, "%d 個の文字列を置換しました ", @3)
```

メッセージボックスに表示させるにはシステム関数 messagebox を使います。

書式 messagebox(ptr1, ptr2, val)

ptr1 : 表示するテキスト

ptr2 : メッセージボックスのタイトル

val : マクロ定数

ここでは、第1引数に sprintf 関数で加工した @str4 を記述し、タイトルの第2引数にはダブルクォーテーション “ ” でくくり、「結果」と記述します。第3引数にはマクロ定数 MB_OK を記述します。MB_OK はメッセージボックスが表示されたときに、OK ボタンが表示されます。OK ボタンをクリックすると、メッセージボックスが消えます。

```
messagebox(@str4, '結果', MB_OK)
```

5

マクロの終了

最後にマクロの終了を表すシステム関数 exit を記述して、このマクロは完成です。

```
exit() ;マクロの終了
```

ソースプログラムは以下のようになります。

```
*creplace C 言語関数内の文字列置換

search(0,0,"{") ; { の検索
@1 = @num ; 開始位置の論理行番号の取得
parenthesis(MAX_NUMBER) ; { と対になる括弧 } の検索
@2 = @num ; 終了位置の論理行番号の取得
if(@1 == @2)
    messagebox(" 対になる括弧がありません ", "エラー" ,MB_OK)
    exit() ;マクロの終了
endif
input(@str1, "旧文字列を入力してください")
input(@str2, "新文字列を入力してください")
sprintf(@str3, "%s/", @str1)
@3=replace(1,4,@str3,@1,@2,@str2) ;文字列の置換 (確認なし)
@num = @1 ;開始位置へジャンプ
sprintf(@str4, "%d 個の文字列を置換しました", @3)
messagebox(@str4, "結果" ,MB_OK)
exit() ;マクロの終了
```



マクロコマンド creplace のソースファイルは、CD-ROM 内の TUTORIAL フォルダに収録されています。



第 3 章 マクロコマンドを実行する

この章ではソースプログラムを作成した後、実際にマクロコマンドを実行するまでを説明しています。

目次

コンパイルする	66
1つのコマンドずつコンパイルする ...	66
ファイル全体をコンパイルする	67
.....	
マクロコマンドを実行する	68
カレントマクロの実行	68
ライブラリを使う	68
ボタンやキー操作に割り当てて実行する ...	70
MIW.MAC について	71
マクロコマンドの中止	71
.....	
その他	72
マクロモードについて	72
1行ずつマクロコマンドを実行する ...	73
ソースコードの取り出し	74
ユーザー変数の確認と変更	74
他のライブラリからの登録	76
(マクロライブラリのマージ)	

コンパイルする

MIL/W 言語はコンパイラ言語のため、ソースプログラムをコンパイルして中間コード（マクロコマンド）に変換しなければ実行できません。

MIFES にはコンパイルする機能として次の 2 つの方法があります。

- ・ 1 コマンドずつコンパイルする
- ・ ファイル全体をコンパイルする

また、コンパイル時にマクロモード（コンパイルや実行に関する設定）を指定することができます。

マクロモードについては、P.72 を参照してください。

1 つのコマンドずつコンパイルする

MIL/W 言語ではマクロ定義行の半角アスタリスク * から、次の定義行の半角アスタリスクの直前、またはファイルの最後までを 1 つのマクロコマンドとみなします。そのため、ソースプログラムファイルに複数のマクロ定義行があるときに、マクロコマンドの 1 つ 1 つをコンパイルすることができます。以下の手順でコンパイルします。

①

マクロ定義行の半角アスタリスクのある行にカーソルを合わせます。



ポイント

1 つのコマンドずつコンパイルするときにはマクロ定義行の半角アスタリスクのある行にカーソルがないと、コンパイル終了後、「マクロ定義行がありません」のメッセージが表示され、正常にコンパイルできません。そのため、必ず半角アスタリスクのある行にカーソルを合わせてから、コンパイルしてください。

②

【マクロ(M)】-【1 コマンド分のコンパイル(C)】を選択します。

③

コンパイルが終了すると、次のマクロ定義行までカーソルが移動します。

ソースプログラム内に記述の誤りなどがあるときは、コンパイルエラーとなりその位置でカーソルが止まります。また、ウィンドウの多目的バーにエラーメッセージが表示されます。このエラーメッセージを確認してソースプログラムの修正をしてください。

正常にコンパイルが終了すると、このマクロコマンドがカレントマクロになります。

マクロの実行方法やライブラリへの格納方法については、次節の「マクロコマンドを実行する」を参照してください。

ファイル全体をコンパイルする

カレントウィンドウに表示しているソースファイルを、ファイルの最初から最後までコンパイルします。1つのソースプログラムの中に複数のマクロコマンドの記述がある場合には、すべてのマクロコマンドをコンパイルできます。

① 【マクロ(M)】-【ファイル全体のコンパイル(F)】を選択します。

② コンパイルが終了すると、カーソルはファイルの最下行に移動します。

ソースプログラム内に記述の誤りなどがあるときは、コンパイルエラーとなりその位置でカーソルが止まります。また、ウィンドウの多目的バーにエラーメッセージが表示されます。このエラーメッセージを確認してソースプログラムの修正をしてください。

正常にコンパイルが終了すると、ファイル中の最後のマクロコマンドがカレントマクロになります。

マクロの実行方法やライブラリへの格納方法については、次節の「マクロコマンドを実行する」を参照してください。



メモ

ファイル全体をコンパイルするときには、カーソルをマクロ定義行の半角アスタリスクのある行に合わせる必要はありません。コンパイル実行時のカーソル位置に関係なく、ファイルの最初から最後までをコンパイルします。



ポイント

ファイル全体のコンパイルでは、マクロモードの「コンパイル後のライブラリ自動格納」を指定しておく便利です。ファイル全体のコンパイルと、マクロモードを合わせて使うと、1つ1つのソースプログラムをコンパイルして、それぞれの中間コード(マクロコマンド)をライブラリに格納する手間が省けます。マクロモードについては、次節の「マクロモードについて」を参照してください。

マクロコマンドを実行する

マクロコマンドの実行には、カレントマクロを実行する方法と、ライブラリから実行する方法の2とおりがあります。ここではカレントマクロの実行方法と、ライブラリの使い方について説明しています。

カレントマクロの実行

カレントマクロとは、正常にコンパイルが終了したときに、カレントマクロバッファというメモリ領域に格納されているマクロのことをいいます。また、ライブラリからマクロコマンドを実行したときもカレントマクロバッファに格納されます。最後に、実行したマクロコマンドがカレントマクロになります。

1

【マクロ(M)】-【カレントマクロの実行(R)】を選択します。



注意

カレントマクロコマンドは、コンパイル/実行のたびに更新されます。また、MIFESを終了させると消去されます。

繰り返し使うマクロコマンドは次項を参考に、ライブラリに登録してください。



メモ

カレントマクロが格納されているかどうかは、【マクロ(M)】-【指定マクロコマンドの実行(X)】で表示される、ダイアログボックスの中の【カレントマクロコマンド】の欄で確認できます。

なお、【カレントマクロコマンド】欄右側の【実行】ボタンをクリックすると、ダイアログボックスからカレントマクロコマンドを実行できます。

ライブラリを使う

ライブラリとはMIFESのデータベースファイルで、ファイル名はMIW.LIBです。MIW.LIBの中にはマクロコマンドを格納できます。このライブラリにはマクロコマンド以外に、子プロセスコマンドやキーボードマクロなども格納されます。

マクロコマンドをライブラリに格納しておく、キーやボタン、メニューバー、右クリックメニューに割り付けることができます。よく使うマクロコマンドはライブラリに格納するようにしてください。



用語

ライブラリは、ロードディレクトリ上の「MIW.LIB」という名前のファイルです。ここでロードディレクトリとは、MIFESの専用ファイルを保存しているディレクトリのことを指します。「MIWディレクトリ」とも呼ぶこともあります。

ロードディレクトリは通常はMIFES本体(MIW.EXE)のあるディレクトリですが、起動時のオプション /L を指定した場合は、/L のあとに指定したディレクトリがロードディレクトリになります。詳しくはヘルプを参照してください。

ライブラリへの格納

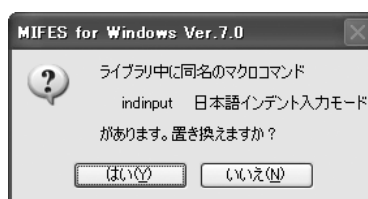
ライブラリに格納するには、ソースプログラムがコンパイルされてカレントマクロバッファに格納されていなければなりません。

カレントマクロは、次の手順でライブラリに格納できます。

- ① 【マクロ(M)】-【カレントマクロのライブラリ格納(L)】を選択します。
- ② 「カレントマクロコマンド(マクロコマンド名)をライブラリに格納しました」というメッセージが表示され、ライブラリに格納されます。

ライブラリではマクロをマクロコマンド名で識別します。そのため、同じマクロコマンド名のマクロを2つ以上ライブラリに格納することはできません。

同じ名前のマクロコマンドを格納しようとする、次のようなメッセージが表示されます。



ライブラリ内のマクロコマンドを上書きしてよければ、[はい(Y)]ボタンをクリックしてください。



ソースプログラムをコンパイル後、ライブラリに自動的に格納するように設定することもできます。1つのファイルに複数のマクロコマンドを記述している場合は、一度にライブラリに格納されるので便利です。詳しくは次節の「マクロモードについて」を参照してください。

ライブラリからの実行

ライブラリに格納したマクロコマンドは、以下の手順で実行します。

- 1 【マクロ(M)】-【指定マクロコマンドの実行(X)】を選択します。

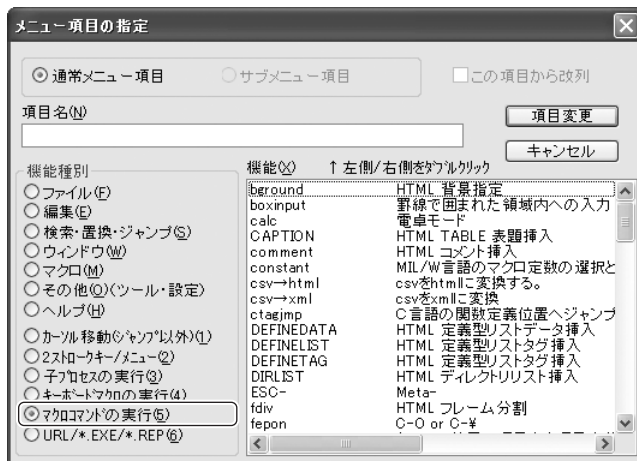


- 2 [ライブラリ内マクロコマンド(L)]の中からマクロコマンドを選択し、[OK] ボタンをクリックします。

ボタンやキー操作に割り当てて実行する

ライブラリに登録したマクロコマンドは、ユーザー定義バーのボタンやキー操作などに割り当てて実行することができます。

割り当ての操作については、ユーザーズマニュアル4章を参照してください。



MIW.MAC について

MIW.MAC ファイルは自動マクロ定義ファイルといい、設定によりロードディレクトリ (MIFES をインストールしたディレクトリ) 上のこのファイルを、MIFES の起動時に自動的にコンパイルすることができます。

MIW.MAC を起動時にコンパイルすると、MIW.MAC に記述されているマクロコマンドは【マクロ(M)】 - 【指定マクロコマンドの実行(X)】機能で、「MIW.MAC ファイル内マクロコマンド(M)」欄から実行することができます (ライブラリに登録する必要はありません)。



メモ

起動時にMIW.MACをコンパイルする設定

【設定(O)】 - 【環境設定(E)】 - [起動] タブの「起動時に"MIW.MAC"をコンパイルする」にチェックをつけます。

開発中や動作確認中のマクロコマンドはMIW.MAC ファイルに記述しておく、ライブラリに登録することなく繰り返し実行できます。

また途中で修正を行っても、【マクロ(M)】 - 【指定マクロコマンドの実行(X)】機能で表示される「マクロコマンド一覧」ダイアログボックスからコンパイルすることができ ([MIW.MACをコンパイル] ボタン) ライブラリに再登録することなく実行することができます。

MIW.MAC には、最大100コマンドまで定義することができます。

ただし、あまりたくさん定義するとコンパイルに時間がかかり、MIFES の起動が遅くなってしまいます。

動作に問題ないことが確認できた時点で、そのコマンドのソースをMIW.MAC から削除し、任意のソースファイルに保存しておくといいいでしょう。

マクロコマンドの中止

[Pause 并ー (NEC PC98 シリーズの場合は [STOP 并ー]) でマクロコマンドの実行をいつでも中止させることができます。

マクロモードについて

マクロモードとは、ソースプログラムをコンパイルするときやマクロコマンドを実行するときの設定です。マクロモードは【マクロ(M)】-【マクロモード設定/コンパイル(M)】で設定します。

モードを変更したあとで、この画面から直接コンパイルすることもできます。



中間コードへのソース埋め込み

[ON]にすると、コンパイルした中間コード(マクロコマンド)に、1行ずつソースプログラムを埋め込みます。

ソースプログラムを埋め込んだマクロコマンドをシングルステップで実行すると、1行分を実行するごとに、次に実行する1行分のソースプログラムをウィンドウの多目的バーに表示します。



また、ソースプログラムを埋め込んでいると、マクロコマンドからソースコード(ソースプログラム)を取り出すこともできます。詳しくは「ソースコードの取り出し」を参照してください。

コンパイル後のライブラリ自動格納

[ON]にすると、ソースプログラムをコンパイルしてエラーがない場合は、マクロコマンドを自動的にライブラリに格納します。

ただし、ファイル全体のコンパイルを実行した際に、マクロライブラリの中に同じマクロコマンド名が格納されていると、自動的に上書きされます。このとき、確認のメッセージは表示されません。

1コマンド分のコンパイルを実行した際は、常に確認のメッセージが表示されます。

ユーザー定義変数の使用

[許可]にすると、ユーザー定義変数を使うことができます。

この場合、ソースプログラムのシステム変数を単純に書き間違えているときもユーザー変数とみなされ、コンパイルエラーにはなりません。このような単純ミスを防止するには、[禁止]

に設定してください。

ユーザー定義変数の使用を禁止すると、その変数領域を利用してグローバル単純変数 @17 ~ @32 とローカル単純変数 @@17 ~ @@32 を使えるようになります。

switch 構文の実行制御

switch ~ case ~ endsw 構文を、Ver.5.0 までと互換性のあるタイプと、C 言語の switch と互換性のあるタイプの、どちらのタイプで実行するか指定します。

[旧タイプ] にすると、case 文以降に switch 構文を抜け出る条件は、break 文、case 文、または endsw 文が見つかるまで、となります。[新タイプ] では break 文または case 文が見つかるまでとなります。

1 行ずつマクロコマンドを実行する

シングルステップ実行

[ON] にすると、ソースプログラムの 1 行分ごとにマクロコマンドを実行します。これをシングルステップ実行と呼びます。

1 行分のマクロコマンドを実行後、一時停止します。一時停止することをブレークといいます。

次の 1 行分を実行するには、【マクロ(M)】 - 【ブレークマクロ再開(B)】を選択します。

ブレークを再開すると、次の 1 行分を実行し、再度ブレークします。

また、コンパイル時に中間コード(マクロコマンド)へのソースの埋め込みを指定していると、ウィンドウの多目的バーに 1 行分のソースが表示されます。マクロコマンドへのソースの埋め込みについては前節の「マクロモードについて」を参照してください。

マクロコマンドの中止

ブレーク中のマクロコマンドの実行を中止するには、【マクロ(M)】-【ブレークマクロ中止(S)】を選択します。

実行中のマクロコマンドは [Pause] キーまたは [STOP] キーで中止することができます。

シングルステップの中止

シングルステップの実行を中止し、通常のマクロコマンド実行に切り替えることができます。

切り替えるには、【マクロ(M)】-【シングルステップ実行の解除(G)】を選択します。

ソースコードの取り出し

ソースコードの取り出しとは、ソースプログラムをコンパイルして変換した中間コード（マクロコマンド）からソースプログラムを抜き出すことをいいます。

ただし、コンパイルしたときにマクロモードで「中間コードへのソース埋め込み」をONにしてあったものに限られます。ソースの埋め込みを指定してコンパイルしていないと、ソースコードの取り出しはできません。

ソースコードを取り出したいマクロコマンドをカレントマクロにし、【マクロ(M)】-【カレントマクロのソースコード取出(U)】を選択します。カレントウィンドウのカーソル位置にソースコード（ソースプログラム）が挿入されます。

このとき、コメントだけの行、および改行だけの行は削除されています。コメントだけの行とは、コメントの始まりを表す半角セミコロン；で始まる行のことをいいます。



マクロモードについては前項「マクロモードについて」を参照してください。



カレントマクロは、【マクロ(M)】-【指定マクロコマンドの実行(X)】を選択すると、確認できません。

ユーザー変数の確認と変更

ユーザー変数の確認および変更ができます。ただし、ローカルユーザー変数については、シングルステップ実行中でブレークしているとき、システム関数bpでブレークしているとき、およびシステム関数variablesを実行したときにだけ、確認と変更ができます。

なお、演算式の入力やcalc()関数用の変数の確認、変更もできます。

ユーザー変数とは次の5種類の変数です。

- ・グローバル単純変数 @1 ~ @32
- ・グローバル配列変数 @str1 ~ @str8
- ・ローカル単純変数 @@1 ~ @@32
- ・ローカル配列変数 @@str1 ~ @@str8
- ・calc()関数用の浮動小数点変数 @F0 ~ @F4



変数について詳しくは、第1章の「変数と定数について」の変数を参照してください。

【マクロ(M)】-【マクロ用ユーザー変数の表示/変更(V)】を選択すると、以下のダイアログボックスが表示されます。



値の変更をする場合は、

① 変更したい変数名をダブルクリックします。

② [変更入力 / calc()関数の演算式(C)]に「(変数名)=」と表示されるので、「=」のあとに値を入力します。

例：

10進数の場合 @1=15
16進数の場合 @5=0x0c

配列変数の場合 @str4=test

演算式の場合 @1=5*3.44/2 (結果 @1=8)
@F1=1.0 : @F2=2.0 : @F3=sqrt(@F1+@F2)
(結果 @F1=1.0, @F2=2.0, @F3=1.73...)
5*3.44/2 (結果 @F0=8.6)

配列変数に文字列を代入する場合、以下のメタ文字が指定できます。

¥n 改行文字
¥t タブ文字
¥0 スペース文字
¥x?? 16進数の2桁のコード



シングルステップについて詳しくは、前項「1行ずつマクロコマンドを実行する」を参照してください。

演算式の書式については、第4章のcalc()のページを参照してください。

他のライブラリからの登録 (マクロライブラリのマージ)

ロードディレクトリ上の"MIW.LIB"以外の、任意のライブラリファイル中のマクロコマンドを実行したり登録することができます。

- 1 【マクロ(M)】-【指定マクロコマンドの実行(X)】を選択します。
- 2 「マクロコマンド一覧」ダイアログボックスの【他のライブラリ(O)】ボタンをクリックし、ライブラリファイルを指定します。
- 3 右側のリストに指定したライブラリファイル中のマクロコマンドが表示されます。



マクロコマンド選択し、【実行】ボタンをクリックすると実行できます。

有効なマクロライブラリ(ロードディレクトリ上の"MIW.LIB")に登録する場合は、マクロコマンドを選択して【登録(R)】ボタンをクリックします。



キーやボタン、メニューに割り当てたマクロコマンドを実行する場合、マクロコマンド名を元に、以下の順序で検索されます。同じ名前のマクロコマンドが1から3のなかに重複してある場合は注意してください。

- 1 . カレントマクロコマンド
- 2 . "MIW.MAC"中にあるマクロコマンド(ただしコンパイル済の場合のみ)
- 3 . ロードディレクトリ上のライブラリ"MIW.LIB"中にあるマクロコマンド

第 4 章 リファレンス

この章では、システム変数、システム関数、およびコンパイルエラーのメッセージ一覧を記載しています。

目次

システム変数	78
1. カーソル位置の情報	78
2. カレントウィンドウの情報	80
3. 以降に開くウィンドウに関する情報 ...	84
4. ユーザー入力情報	86
5. その他の情報.....	89
.....	
システム関数	94
1. 文字列操作関数	94
2. 文字列の挿入, 削除関数	106
3. ジャンプ, 移動, 検索, 置換関数 ...	113
4. ユーザー入力関数	122
5. ファイル操作関数	129
6. その他の関数.....	143
.....	
エラーメッセージ一覧.....	164
.....	
索引	165

システム変数

以下のようにシステム変数が表す情報で分類しています。

- 1.カーソル位置の情報
- 2.カレントウィンドウの情報
- 3.以降に開くウィンドウに関する情報
- 4.ユーザー入力情報
- 5.その他の情報

1. カーソル位置の情報

@byte

バイト位置

カーソル位置のファイル先頭からのバイト位置(1 ~)を表します。ファイルの先頭のバイト位置が1になります。ファイル中でのカーソル位置を取得するときは @byte を使います。

@byte にバイト位置を代入すると、代入したバイト位置、または移動できる最も近いバイト位置に、カーソルを移動します。

カレントウィンドウがないときに参照や代入をするとマクロエラーになります。

@code

文字コード

カーソル位置の文字コードを表します。

改行文字は0x0d0a です。また、CR コードを含まないLF コードだけのときは0x000a になります。

[EOF]マークは0xff1a(コード0x1aとして保存されるもの) または 0xffff(データとしては保存されないもの)です。

漢字は1バイト目が上位8ビット、2バイト目が下位8ビットで表す16ビットのコードになります。

@code に文字コードを代入すると、代入した文字をカーソル位置に挿入または上書きできます。カレントウィンドウがないときに参照や代入をするとマクロエラーになります。



[EOF]マークの0xffff は正確には0x0000ffff のことです。0xffff と -(0xffffffff)とは異なります。

@col

論理桁位置

カーソル位置のファイル中での桁位置(1 ~ 3000)(論理桁位置)を表します。

@col に桁位置を代入すると、代入した桁位置、または移動できる最も近い桁位置に、カーソルを移動します。

カレントウィンドウがないときに参照や代入をするとマクロエラーになります。
@colと@scolは通常同じ値です。フリーカーソル状態のときに、カーソルが改行文字や
[EOF] マークよりも右側にある場合にのみ異なる値になります。

@cposx

ウィンドウ内の桁位置

ウィンドウ内での現在のカーソル位置の桁位置(0 ~ X 表示桁位置)を表します。
テキスト表示領域の左端が0になります。行番号などを表示する行ゲージはテキスト表示領域外
と見なします。また、画面上での桁位置を表すため、最大値はウィンドウの横幅に依存します。
@cposx に値は代入できません。
カレントウィンドウがないときに参照するとマクロエラーとなります。

@cposy

ウィンドウ内の行位置

ウィンドウ内での現在のカーソル位置の行位置(0 ~)を表します。
テキスト表示領域の最上行が0になります。カーソル位置情報などを表示するガイドラインや
桁ゲージはテキスト表示領域外と見なします。また、画面上での行位置を表すため、最大値は
ウィンドウの高さに依存します。
@cposy に値は代入できません。
カレントウィンドウがないときに参照をするとマクロエラーとなります。

@line

表示行番号

カーソル位置の表示行番号(1 ~)を表します。
@line に表示行番号を代入すると、代入した表示行番号、または移動できる最も近い表示行番
号の行頭に、カーソルを移動します。
カレントウィンドウがないときに参照や代入をするとマクロエラーとなります。

@num

論理行番号

カーソル位置の論理行番号(1 ~)を表します。
@num に論理行番号を代入すると、代入した論理行番号、または移動できる最も近い論理行
番号の行頭に、カーソルを移動します。
カレントウィンドウがないときに参照や代入をするとマクロエラーとなります。



メモ

表示行番号とは、画面上に1行で表示されるテキストを1行として数えた行番号です。
これに対し論理行番号は、改行文字から改行文字までの間のテキストを1行として数えた行番
号です。
折り返し桁位置を超えて折り返した行があると、折り返し桁位置の値によって表示行番号の値
は変わります。論理行番号の値は、改行文字から改行文字を1行とするために、値は変わりま
せん。そのため、ファイル中のカーソル位置は、@byte や@num を使うと正確に取得できま
す。@byte はカーソル位置のバイト位置を表し、@num はカーソル位置の論理行番号を表し
ます。

@scol

画面上の論理桁位置

カーソル位置の画面上での桁位置(1 ~ 3000 桁論理桁位置)を表します。

@scol に桁位置を代入すると、代入した桁位置、または移動できる最も近い桁位置に、カーソルを移動します。

カレントウィンドウがないときに参照や代入をするとマクロエラーとなります。

@col と @scol は通常同じ値です。フリーカーソル状態のときに、カーソルが改行文字や [EOF マークよりも右側にある場合にのみ異なる値になります。

2. カレントウィンドウの情報

@astat

動作状態

カレントウィンドウの動作状態を各ビットで表します。

@astat に値を代入すると、カレントウィンドウの動作状態を変更できます。

カレントウィンドウがないときに参照や代入をするとマクロエラーとなります。

各ビットは以下のマクロ定数を使って、ビット論理演算で参照や代入を実行します。

ASTAT_INDENT	オートインデント (0:無効 1:有効)
ASTAT_EDGESTOP	行端でのカーソル移動 (0:止まらず 1:止まる)
ASTAT_FREECURSOR	フリーカーソルモード (0:OFF 1:ON)
ASTAT_CHGSCALE	半角英文字の変換 (通常は Ctrl + 冫ー) (0 : 大文字へ変換 1:小文字へ変換)
ASTAT_SHIFTUPDN	Shift+ キーの動作 (0:高速にロールアップ/ダウン 1:範囲選択しながら移動)
ASTAT_SHIFTLTRT	Shift+ キーの動作 (0:行の左右端に移動 1:範囲選択しながら移動)
ASTAT_KEISEN	使用する罫線 (0:NEC罫線 1:JIS罫線)
ASTAT_SOFTTAB	Tab キーの動作 (0:ハードタブ 1:ソフトタブ)
ASTAT_MEMOMARK	コード FDH、FEH を使用したメモマーク (0:無効 1:有効)
ASTAT_SAVECLEAR	保存時の変更行マークの処理 (0:そのまま 1:クリアする)
ASTAT_ENTERLF	Enter キーで挿入する改行文字 (0:CR + LF 1:LFのみ)
ASTAT_HSCROLL	横スクロール動作 (0: 8 桁単位でスクロール 1:スムーズにスクロール)
ASTAT_EOFCTRL	ファイルの最後の判定 (0:EOF(0x1a) 1:データの終わり)
ASTAT_NOKANJI	漢字の処理 (0:行う 1:行わない)
ASTAT_MOUSEPOS	マウスによるカーソル移動 (0:禁止 1:許可)

ASTAT_EOFCTRL と ASTAT_NOKANJI のビットは参照専用です。この 2 つのビットには代入できません。

ASTAT_EOFCTRL と ASTAT_NOKANJI のビットの両方が 0 の時は「テキスト(^Zまで)」モードのウィンドウです。

ASTAT_EOFCTRL だけが1の時は「テキスト」モードのウィンドウです。
ASTAT_EOFCTRLとASTAT_NOKANJIのビットの両方が1のときは「バイナリ」モードのウィンドウです。

なお、旧バージョンにあった以下のマクロ定数は、バージョン7では使用できません。

```
ASTAT_BACKUPFILE
ASTAT_MOUSEPOS
ASTAT_READER
ASTAT_DBLREADER
```

例 フリーカーソルモードのON / OFFを切り替える

```
if @astat&ASTAT_FREECURSOR                ;フリーカーソルモードがONならば
    @astat &= ( ASTAT_FREECURSOR)         ;フリーカーソルモードOFF
else
    @astat |= ASTAT_FREECURSOR            ;フリーカーソルモードON
endif
```

@dstat 表示状態

カレントウィンドウの表示状態を各ビットで表します。

@dstat に値を代入すると、カレントウィンドウの表示状態を変更できます。

カレントウィンドウがないときに参照や代入をするとマクロエラーとなります。

各ビットは以下のマクロ定数を使って、ビット論理演算で参照や代入をしてください。

DSTAT_COLGAGE	桁ゲージ(0:非表示 1:表示)
DSTAT_NUMGAGE	行ゲージ(0:非表示 1:表示)
DSTAT_NUMTYPE	行ゲージのタイプ(0:論理行番号 1:表示行番号)
DSTAT_VSCROLL	垂直スクロールバー(0:使用せず 1:使用する)
DSTAT_HSCROLL	水平スクロールバー(0:使用せず 1:使用する)
DSTAT_CRLF	改行文字の明示(0:明示せず 1:明示する)
DSTAT_HTAB	ハードタブの明示(0:明示せず 1:明示する)
DSTAT_KANSPLACE	全角スペースの明示(0:明示せず 1:明示する)
DSTAT_REDGE	折り返し位置の明示(0:明示せず 1:明示する)
DSTAT_UNDERLINE	カーソル行アンダーライン(0:表示せず 1:表示する)
DSTAT_VERTICALLINE	カーソル桁バーチカルライン(0:表示せず 1:表示する)
DSTAT_2VLINES	カーソル桁バーチカルラインのタイプ(0:1本線 1:2本線)
DSTAT_UPDATE	変更行の明示(0:明示せず 1:明示する)
DSTAT_GUIDELINE	ガイドラインの表示(0:表示しない 1:表示する)
DSTAT_EOFMARK	ファイル最後尾の明示(0:[EOF]マーク 1:明示せず)
DSTAT_BACKLINE	背景横罫線の表示(0:表示せず 1:表示する)
DSTAT_KAKKO	対応括弧の明示(0:明示しない 1:明示する)
DSTAT_KAKKOMARK	対応括弧の明示方法(0:反転表示 1:枠で表示)

例 カーソル行アンダーラインの表示 / 非表示を切り替える

```
if @dstat&DSTAT_UNDERLINE      ;カーソル行アンダーラインが表示ならば
    @dstat &= ( DSTAT_UNDERLINE) ;カーソル行アンダーライン非表示
else
    @dstat |= DSTAT_UNDERLINE    ;カーソル行アンダーライン表示
endif
```

@htab

ハードタブ

カレントウィンドウのハードタブの桁間隔(2、 4、 8、 16)を表します。

@htab に値を代入すると、カレントウィンドウのハードタブの桁間隔を変更し、代入したハードタブの値でカレントウィンドウを再表示します。

カレントウィンドウがないときに参照や代入をするとマクロエラーとなります。

@margin

折り返し桁位置

カレントウィンドウの折り返し桁位置(16 ~ 3000)を表します。

@margin に値(16 ~ 3000)を代入すると、カレントウィンドウの折り返し桁位置を変更できます。0 を代入した場合には、現在のウィンドウ横幅に合わせて折り返し桁位置を自動的に調整します。@margin に値を代入すると、代入した新しい折り返し位置にあわせてカレントウィンドウを再表示します。

カレントウィンドウがないときに参照や代入をするとマクロエラーとなります。

@size

カレントウィンドウのファイルサイズ

カレントウィンドウのファイルサイズを表します。カレントウィンドウがないときは-1 (0xffffffff)になります。

@size に値は代入できません。

@text

テキスト番号

カレントウィンドウのテキスト番号(0 ~ 99)を表します。

カレントウィンドウがないときは -1(0xffffffff)になります。

@text はカレントウィンドウがあるかないかを調べるときにも使用できます。

@text に値(0 ~ 99)を代入すると、代入したテキスト番号のウィンドウに切り替えます。代入したテキスト番号のウィンドウがないときには無視します。

@tmpmargin 折り返し桁位置

カレントウィンドウの折り返し桁位置(16 ~ 3000)を表します。

@tmpmargin に値(16 ~ 3000)を代入すると、カレントウィンドウの折り返し位置を変更できません。

しかし、値を代入してもウィンドウの再表示はしません。値を代入後に変更した行だけを、代入した新しい桁位置で折り返して表示します。

カレントウィンドウがないときに参照や代入をするとマクロエラーとなります。

@winstat カレントウィンドウの状態

カレントウィンドウの状態を表します。

カレントウィンドウがないときは -1(0xffffffff) になります。各ビットの意味は以下のとおりです。

ビット 0 ~ 1

- 0: 通常の編集可能ウィンドウ
- 1: 読み取り専用属性ファイルのウィンドウ
- 2: バックアップファイル(.bak, .bk?)
- 3: 読み取り専用で開かれたウィンドウ

ビット 2 ~ 3

- 0: 通常ウィンドウ
- 1: 「新規:nn」ウィンドウ
- 2: 起動時に自動的に開かれた「新規:00」ウィンドウ
- 3: 「グローバル検索結果」ウィンドウ、「DOS シェル・エスケープ」ウィンドウ、または「一括ファイル比較結果」ウィンドウ

ビット 4 ~ 5

- 0: 通常サイズウィンドウ
- 1: アイコン化ウィンドウ
- 2: MDI 最大化ウィンドウ

ビット 6

- 0: 常に 0

ビット 7

- 0: 変更操作なし
- 1: 変更操作あり(保存の必要あり)

ビット 0 ~ 1 とビット 7 は、@winstat に値を代入することで変更できます。例えば、ビット 7 を 1 にすることで、カレントウィンドウを強制的に変更操作のあったウィンドウとすることができます。

ビット 2 ~ 6 は変更できません。@winstat に値を代入するときには十分注意してください。

@winx

桁数

カレントウィンドウのテキスト表示領域の横幅を桁数に換算した値(1~)を表します。

行番号を表示する行ゲージはテキスト表示領域外と見なします。

この値はウィンドウサイズ、フォントサイズ、垂直スクロールバーがあるかないか、および行ゲージが何桁で表示されているか、などに依存します。

@winx に値は代入できません。

カレントウィンドウがないときに参照や代入をするとマクロエラーとなります。

@winy

行数

カレントウィンドウのテキスト表示領域の高さを行数に換算した値(1~)を表します。

カーソル位置情報などを表示するガイドラインや桁ゲージはテキスト表示領域外と見なします。

この値はウィンドウサイズ、フォントサイズ、水平スクロールバーがあるかないか、および桁ゲージが表示か非表示かなどに依存します。

@winy に値は代入できません。

カレントウィンドウがないときに参照や代入をするとマクロエラーとなります。

3. 以降に開くウィンドウに関する情報

@openwin

ウィンドウ形状

以降に開くウィンドウの初期のウィンドウ形状を表します。

この値は以下のとおりです。

- 0: カスケード状態で開く(重ねて開く)
- 1: MDI 最大化で開く
- 2: 最大サイズのウィンドウで開く
- 3: デフォルトのウィンドウ形状で開く

@openwin に値を代入すると、以降に開かれるウィンドウの初期のウィンドウ形状を変更できます。

@sys_astat

動作状態

以降に開くウィンドウの初期状態での動作状態を各ビットで表します。

MIFES では、この状態をカレントウィンドウの動作状態とは別に保持しています。

@sys_astat に値を代入すると、以降に開くウィンドウの初期状態での動作状態を変更できます。

各ビットは以下のマクロ定数を使って、ビット論理演算で参照や代入をしてください。

ASTAT_INDENT	オートインデント(0:無効 1:有効)
ASTAT_EDGESTOP	行端でのカーソル移動(0:止まらず 1:止まる)
ASTAT_FREECURSOR	フリーカーソルモード(0:OFF 1:ON)
ASTAT_CHGSCALE	半角英文字の変換(通常は Ctrl + 冫ー) (0:大文字へ変換 1:小文字へ変換)
ASTAT_SHIFTUPDN	Shift+ キーの動作 (0:高速にロールアップ/ダウン 1:範囲選択しながら移動)
ASTAT_SHIFTLTRT	Shift+ キーの動作 (0:行の左右端に移動 1:範囲選択しながら移動)
ASTAT_KEISEN	使用する罫線(0:NEC罫線 1:JIS罫線)
ASTAT_SOFTTAB	Tabキーの動作(0:ハードタブ 1:ソフトタブ)
ASTAT_MEMOMARK	コードFDH、FEHを使用したメモマーク(0:無効 1:有効)
ASTAT_SAVECLEAR	保存時の変更行マークの処理(0:そのまま 1:クリアする)
ASTAT_ENTERLF	Enterキーで挿入する改行文字(0:CR + LF 1:LFのみ)
ASTAT_HSCROLL	横スクロール動作 (0:8桁単位でスクロール 1:スムーズにスクロール)
ASTAT_NO3CLICK	マウスのトリプルクリックの禁止 (0:トリプルクリック許可 1:トリプルクリック禁止) ASTAT_NO3CLICKの設定だけは、ウィンドウごとに別々の設定にすることはできません。 システム変数 @sys_astat が示す ASTAT_NO3CLICK の設定は、以降に開くウィンドウの初期状態だけでなく、既に開いているすべてのウィンドウの状態も同時に示しています。
ASTAT_EOFCTRL	ファイルの最後の判定(0:EOF(0x1a) 1:データの終わり)
ASTAT_NOKANJI	漢字の処理(0:行う 1:行わない)
ASTAT_MOUSEPOS	マウスによるカーソル移動(0:禁止 1:許可)

例：フリーカーソルモードのON / OFFを切り替える

```

if @sys_astat&ASTAT_FREECURSOR           ;フリーカーソルモードがONならば
    @sys_astat &= (ASTAT_FREECURSOR)     ;フリーカーソルモードOFF
else
    @sys_astat |= ASTAT_FREECURSOR       ;フリーカーソルモードON
endif

```

@sys_dstat

表示状態

以降に開くウィンドウの初期状態での表示状態を各ビットで表します。

MIFESでは、この状態をカレントウィンドウの表示状態とは別に保持しています。

@sys_dstat に値を代入すると、以降に開くウィンドウの初期状態での表示状態を変更できます。

各ビットは以下のマクロ定数を使って、ビット論理演算で参照や代入をしてください。

DSTAT_COLGAGE	桁ゲージ(0:非表示 1:表示)
DSTAT_NUMGAGE	行ゲージ(0:非表示 1:表示)
DSTAT_NUMTYPE	行ゲージのタイプ(0:論理行番号 1:表示行番号)
DSTAT_VSCROLL	垂直スクロールバー(0:使用せず 1:使用する)
DSTAT_HSCROLL	水平スクロールバー(0:使用せず 1:使用する)
DSTAT_CRLF	改行文字の明示(0:明示せず 1:明示する)
DSTAT_HTAB	ハードタブの明示(0:明示せず 1:明示する)

DSTAT_KANSPLACE	全角スペースの明示(0:明示せず 1:明示する)
DSTAT_REDEGE	折り返し位置の明示(0:明示せず 1:明示する)
DSTAT_UNDERLINE	カーソル行アンダーライン(0:表示せず 1:表示する)
DSTAT_VERTICALLINE	カーソル桁バーチカルライン(0:表示せず 1:表示する)
DSTAT_2VLINES	カーソル桁バーチカルラインのタイプ(0:1本線 1:2本線)
DSTAT_UPDATE	変更行の明示(0:明示せず 1:明示する)
DSTAT_GUIDELINE	ガイドラインの表示(0:表示しない 1:表示する)
DSTAT_EOFMARK	ファイル最後尾の明示(0:[EOF]マーク 1:明示せず)
DSTAT_BACKLINE	背景横罫線の表示(0:表示せず 1:表示する)
DSTAT_KAKKO	対応括弧の明示(0:明示しない 1:明示する)
DSTAT_KAKKOMARK	対応括弧の明示方法(0:反転表示 1:枠で表示)

例 カーソル行アンダーラインの表示 / 非表示を切り替える

```
if @sys_dstat&DSTAT_UNDERLINE ;カーソル行アンダーラインが表示ならば
    @sys_dstat &= ( DSTAT_UNDERLINE) ;カーソル行アンダーライン非表示
else
    @sys_dstat |= DSTAT_UNDERLINE ;カーソル行アンダーライン表示
endif
```

@sys_hstab

ハードタブ

以降に開くウィンドウの初期状態でのハードタブの桁間隔(2、4、8、16)を表します。

@sys_hstab に値を代入すると、以降に開くウィンドウの初期状態でのハードタブの桁間隔を変更できます。

@sys_margin

折り返し桁位置

以降に開くウィンドウの初期状態での折り返し桁位置(16~3000)を表します。

@sys_margin に値(16~3000)を代入すると、以降に開くウィンドウの初期状態での折り返し桁位置を変更できます。

4. ユーザー入力情報

@char

文字コード

システム関数 waitevent で文字入力イベントを待っていたときに、入力された文字コードを表します。

文字コードは @code と同じです。例えば、改行文字は 0x0d0a で、タブ文字は 0x0009 です。漢字は 1 バイト目が上位 8 ビット、2 バイト目が下位 8 ビットで表す 16 ビットのコードになります。

waitevent 関数を実行した直後のみ参照してください。それ以外のときは意味がありません。また、@char に値は代入できません。

@command

入力されたコマンド

システム関数 `waitevent` でコマンド入力イベントを待っていたときに、入力された機能番号 (1~541) を表します。機能番号についてはヘルプを参照してください。

`waitevent` 関数を実行した直後のみ参照してください。それ以外のときは意味がありません。また、`@command` に値は代入できません。

@key

入力されたキー

システム関数 `waitevent` でキー入力イベントを待っていたとき、入力されたキーコード (仮想キー番号) を表します。この値は以下のとおりです。

```

700 ~ 711 [ F1 ] 押下 ~ [ F12 ] 押下
           ( 709: [ F10 ] 押下 は除く )
712 ~ 715 [ F13 ] 押下 ~ [ F16 ] 押下
716 ~ 731 [ Shift ] 押下 [ F1 ] 押下 ~ [ Shift ] 押下 [ F16 ] 押下
732 ~ 747 [ Ctrl ] 押下 [ F1 ] 押下 ~ [ Ctrl ] 押下 [ F16 ] 押下
           ( 735: [ Ctrl ] 押下 [ F4 ] 押下 と 737: [ Ctrl ] 押下 [ F6 ] 押下 は除く )
748 ~ 773 [ Ctrl ] 押下 [ A ] 押下 ~ [ Ctrl ] 押下 [ Z ] 押下
           ( 755: [ Ctrl ] 押下 [ H ] 押下 は除く )
774      [ PageUp ] 押下      ( VK_PAGEUP )
775      [ PageDown ] 押下   ( VK_PAGEDOWN )
776      [ Insert ] 押下     ( VK_INS )
777      [ Delete ] 押下     ( VK_DEL )
778      [   ] 押下         ( VK_UP )
779      [   ] 押下         ( VK_LEFT )
780      [   ] 押下         ( VK_RIGHT )
781      [   ] 押下         ( VK_DOWN )
782      [ Home ] 押下      ( VK_HOME )
783      [ End ] 押下       ( VK_END )
784      [ Shift ] 押下 [ PageUp ] 押下
785      [ Shift ] 押下 [ PageDown ] 押下
786      [ Shift ] 押下 [ Insert ] 押下
787      [ Shift ] 押下 [ Delete ] 押下
788      [ Shift ] 押下 [   ] 押下
789      [ Shift ] 押下 [   ] 押下
790      [ Shift ] 押下 [   ] 押下
791      [ Shift ] 押下 [   ] 押下
792      [ Shift ] 押下 [ Home ] 押下
793      [ Shift ] 押下 [ End ] 押下
794      [ Ctrl ] 押下 [ PageUp ] 押下
795      [ Ctrl ] 押下 [ PageDown ] 押下
796      [ Ctrl ] 押下 [ Insert ] 押下
797      [ Ctrl ] 押下 [ Delete ] 押下
798      [ Ctrl ] 押下 [   ] 押下
799      [ Ctrl ] 押下 [   ] 押下
800      [ Ctrl ] 押下 [   ] 押下
801      [ Ctrl ] 押下 [   ] 押下

```

802 [Ctrl]+[Home]
 803 [Ctrl]+[End]
 804 [Esc] (VK_ESC)
 805 [Shift]+[Bs]
 806 [Ctrl]+[Bs]
 807 [Shift]+[Tab]
 808 [Ctrl]+[@]
 809 [Ctrl]+[]
 810 [Ctrl]+[¥]
 811 [Ctrl]+[]
 812 [Ctrl]+[^]
 813 [Ctrl]+[_]
 814 ~ 823 [Ctrl]+[0] ~ [Ctrl]+[9]
 824 [Shift]+[Enter]
 825 [Alt]+[Enter]
 828 ~ 843 [Alt]+[F1] ~ [Alt]+[F16]
 (831[Alt]+[F4] と 833[Alt]+[F6] は除く)
 844 [Alt]+[PageUp]
 845 [Alt]+[PageDown]
 846 [Alt]+[Insert]
 847 [Alt]+[Delete]
 848 [Alt]+[]
 849 [Alt]+[]
 850 [Alt]+[]
 851 [Alt]+[]
 852 [Alt]+[Home]
 853 [Alt]+[End]
 854 ~ 869 [Ctrl]+[Shift]+[F1] ~ [Ctrl]+[Shift]+[F16]
 (857[Ctrl]+[Shift]+[F4] と 859[Ctrl]+[Shift]+[F6] は除く)
 870 ~ 895 [Ctrl]+[Shift]+[A] ~ [Ctrl]+[Shift]+[Z]
 (877[Ctrl]+[Shift]+[H] は除く)
 896 [Ctrl]+[Shift]+[PageUp]
 897 [Ctrl]+[Shift]+[PageDown]
 898 [Ctrl]+[Shift]+[Insert]
 899 [Ctrl]+[Shift]+[Delete]
 900 [Ctrl]+[Shift]+[]
 901 [Ctrl]+[Shift]+[]
 902 [Ctrl]+[Shift]+[]
 903 [Ctrl]+[Shift]+[]
 904 [Ctrl]+[Shift]+[Home]
 905 [Ctrl]+[Shift]+[End]

waitevent関数を実行した直後のみ参照してください。それ以外のときは意味がありません。また、@keyに値は代入できません。

[Enter] や [Tab] などカスタマイズができないキーについては、コマンド入力イベント (EVENT_COMMAND) で入力してください。[F10] や [Ctrl]+[Tab] などはWindowsで特殊な処理が割り当てられています。そのため、キーを取得できない場合があります。813[Ctrl]+[_] など、一部のキーは日本語入力状態では入力できない場合があります。

5. その他の情報

@autosave

オートセーブ状態

現在のオートセーブの状態を表します。

ビット0 ~ 1

- 0: オートセーブなし
- 1: オートセーブあり
- 3: アクティブ中にもオートセーブあり

ビット2

- 0: 通常の保存
- 1: 復旧用の保存

@autosave に値を代入すると、オートセーブの状態を変更できます。



システム変数 @saveinterval もあわせて参照してください。

@disp

テキスト表示状態

現在のテキスト表示状態を表します。

@disp に値を代入すると、以下の表示動作を行います。

- 0: 以降テキスト表示は行わない
- 1: 以降テキスト表示を開始。カーソル行を表示する
- 2: 以降テキスト表示を開始。カレントウィンドウを表示する
- 3: 以降テキスト表示を開始。カラーの変更を画面に反映する
- 4: テキストカーソルの再表示を行う

@findmode

カレント検索方法

カレント検索文字列(最後に実行した検索)の検索方法を表します。

ビット0 ~ 3

- 0: 通常検索(大文字・小文字区別)
- 1: 通常検索(大文字・小文字同一視)
- 2: ワイルドカード検索(大文字・小文字区別)
- 3: ワイルドカード検索(大文字・小文字同一視)
- 4: 正規表現検索
- 5: あいまい検索

ビット4

- 0: 通常の方法
- 1: 語単位で探す

@findmode に値は代入できません。

@insmode

挿入、上書き状態

挿入状態 (1) または上書き状態 (0) を表します。

@insmode に値を代入すると、挿入または上書き状態に変更できます。

@keisen

トレース用の罫線種類

現在のトレース用の罫線種類を表します。この値は以下のとおりです。

- 1: 太罫線
- 2: 細罫線
- 0: 空白線

@keisen に値を代入すると、トレース用罫線種を切り替えます。

@nextbyte

論理行テキストの残り

システム関数 `getline` を実行した直後に、行が長すぎて論理行未まで取得できなかったときに、残りの未取得部分の先頭文字のバイト位置 (1 ~) を表します。

`getline` 関数で論理行全体が取得できたときには 0 になります。

また、システム関数 `fread` を実行した直後に、1 行の文字列が長くて指定した配列変数内にすべて取り込めないときに 0 より大きな数になります。

`getline` 関数または `fread` 関数を実行した直後のみ参照してください。それ以外のときは意味がありません。また、@nextbyte に値は代入できません。

@profile

カスタマイズの設定

現在のカスタマイズ設定の状態を表します。

@profile に以下の値を代入すると、カスタマイズ設定の状態を変更できます。

- 0: 次回はデフォルト状態で起動する
- 1: 今回だけはカスタマイズファイルへ書き出される
- 2: 読み取り専用のカスタマイズファイルとして使用
- 3: 読み書き両用のカスタマイズファイルとして使用
- 4: 終了時に開いていたファイルを次回起動時に開く

@retcode

callDll 関数の関数値

システム関数 `callDll` を実行したときの関数値を表します。

`callDll` 関数を実行した直後のみ参照してください。それ以外のときは意味がありません。

@ribbon

ツールバー表示の変更

現在のツールバー/ユーザー定義バー/多目的バーの表示状態を各ビットで表します。

@ribbon に値を代入することにより、ツールバー/ユーザー定義バー/多目的バーの表示を変更することができます。

この変数の各ビットの意味は、システム関数 ribbon () の第一引数と全く同じです。

- ビット3 多目的バーの非表示(NOMULTIBAR)(0:表示する 1:表示しない)
- ビット5 ツールバーの表示(TOOLBAR)(0:表示しない 1:表示する)
- ビット6 ユーザー定義バー1の表示(USERBAR1)(0:表示しない 1:表示する)
- ビット7 ユーザー定義バー2の表示(USERBAR2)(0:表示しない 1:表示する)
- ビット9 ツールバーの下部配置(0:上部に配置 1:下部に配置)
- ビット10 ユーザー定義バー1の下部配置(0:上部に配置 1:下部に配置)
- ビット11 ユーザー定義バー2の下部配置(0:上部に配置 1:下部に配置)
- ビット25 多目的バーの上部配置(0:下部に配置 1:上部に配置)
- ビット31 多目的バーの2行表示(DBLMULTIBAR)(0:1行で表示 1:2行で表示)

@saveinterval

オートセーブの間隔

現在のオートセーブのチェックの時間間隔(単位は分)を表します。

@saveinterval に値(1 ~ 60)を代入すると、オートセーブのチェックの時間間隔を変更できます。



システム変数 @autosave もあわせて参照してください。

@sel_end

選択範囲の最終位置

範囲選択状態のときに、選択範囲の最終文字の次のバイト位置を表します。

行単位選択または文字列選択状態のときにのみ、値を取得できます。

箱型選択状態のときには有効な値を表しません。

@sel_start

選択範囲の先頭位置

範囲選択状態のときに、選択範囲の先頭文字のバイト位置を表します。

行単位選択または文字列選択状態のときにのみ、値を取得できます。

箱型選択状態のときには有効な値を表しません。

@selmode 範囲選択状態

現在の範囲選択状態を表します。

- 0: 非選択状態
- 1: 箱型選択状態
- 2: 行単位選択状態
- 3: 文字列選択状態(モードに入った状態)
- 4: 文字列選択状態(モードに入らない状態)

@selmode に値を代入すると、範囲選択の状態を変更できます。

カレントウィンドウがないときに値を代入してもエラーにはなりません。しかし、代入は無視されます。



文字列選択状態には、モードに入った状態(3)と、モードに入っていない状態(4)の2種類があります。モードに入った状態では、文字列選択状態を維持したまま、カーソルの移動/ジャンプ/検索などができます。モードに入っていない状態では、カーソルを1文字移動しただけでも文字列選択状態が解除されます。



@selmode に値を代入する場合、4(モードに入らない文字列選択状態)は絶対に代入しないでください。この状態を使用したマクロコマンドは、シングルステップ実行では正常に動作しないことがあります。

また、次のマクロ定数を指定しても、選択状態を変更できます。

SEL_CANCEL	非選択状態(0)
SEL_BOX	箱型選択状態(1)
SEL_LINE	行単位選択状態(2)
SEL_STRING	文字列選択状態(3)



@selmode を使って文字列を切り貼りする場合、環境設定の内容によって選択される範囲が異なりますので注意してください。

【その他(O)】-【環境設定(E)】の [動作] タブで [方向範囲選択においてカーソル位置も範囲に含める] がONのときには、カーソル位置の文字も選択範囲に含まれます。

以下の記述で、マクロコマンド実行中に範囲選択の状態を操作できます。

例1:

```
@sys_stat &= SYS_NEWSEL ; 「カーソル位置を含まず」に設定する
```

例2:

```
@sys_stat |= SYS_NEWSEL ; 「カーソル位置も含む」に設定する
```

@sys_stat

エディタ状態

現在のMIFESの各種状態を各ビットで表します。

@sys_stat に値を代入すると、以降の各種状態を変更できます。

各ビットは以下のマクロ定数を使って、ビット論理演算で参照や代入をしてください。

SYS_NOCODECHK	ファイル内容による自動コード判定 (0:有効 1:無効)
SYS_OPEN	起動時に「新規:00」ウィンドウを開く(0:開かない 1:開く)
SYS_OVW	起動時の挿入/上書(0:挿入状態 1:上書状態)
SYS_MULTI	起動モード(0:MDIモード 1:SDIモード)
SYS_CUTBUFF	終了時カットバッファファイル(0:削除しない 1:削除する)
SYS_NOBOMUNICODE	Unicode ファイルへの保存時のBOMの書き込み禁止 (0:BOMを書き込む 1:BOMを書き込まない)
SYS_NOBOMUNICBIG	Unicode Big endian ファイルへの保存時のBOMの書き込み禁止 (0:BOMを書き込む 1:BOMを書き込まない)
SYS_NOBOMUTF8	UTF-8 ファイルへの保存時のBOMの書き込み禁止 (0:BOMを書き込む 1:BOMを書き込まない)
SYS_NEWSEL	方向範囲選択においてカーソル位置の扱い (0:選択範囲に含めない 1:選択範囲に含める)
SYS_FNMENU	【ファイル(F)】メニュー中のファイル履歴 (0:右側に表示 1:下側に表示)
SYS_SDISIZE	SDIウィンドウの横幅 (0:カレントウィンドウと同じ横幅 1:スクリーンの幅の3/4)
SYS_KEYMTOLIB	キーボードマクロ定義時(0:そのまま 1:ライブラリへ登録)
SYS_UPPERBAR	多目的バーの配置(0:下部に配置 1:上部に配置)
SYS_MDIWINMENU	【ウィンドウ(W)】メニューへのファイル名の表示 (0:表示する 1:表示しない) バージョン7では、SDIモード時にもこの設定が有効です。
SYS_BALOON	ボタンの吹き出し表示(0:表示しない 1:表示する)
SYS_MIWMAC	起動時"MIW.MAC"の自動コンパイル (0:コンパイルする 1:コンパイルしない)
SYS_DIALOG	「ファイルを開く」ダイアログボックスのタイプ (0:MIFES専用ダイアログ 1:コモンダイアログ)
SYS_PAGEID	PageUp/PageDownの動作 (0:半画面単位 1:1画面単位)
SYS_CFDIR	「ファイルを開く」ダイアログでの初期のリスト表示 (0:カレントディレクトリ 1:カレントファイルのディレクトリ)

システム関数

以下のようにシステム関数を分類しています。

1. 文字列操作関数
2. 文字列の挿入、削除関数
3. ジャンプ、移動、検索、置換関数
4. ユーザー入力関数
5. ファイル操作関数
6. その他の関数

本文中で使用する引数記号の意味は以下のとおりです。

val	変数、定数、関数、およびそれらを演算子で結んだもの(式)
ptr	要素名の指定を省略した配列変数名 配列要素名にアドレス演算子 & を付けたもの 文字列定数
vptr	要素名の指定を省略した配列変数名 配列要素名にアドレス演算子 & を付けたもの
arg	val または ptr
str	文字列定数

1. 文字列操作関数

calc(vptr,ptr1,ptr2)

文字列による実数演算の実行

文字列で指定された演算式を実行し、指定の書式の文字列で結果を得ます。

演算式の中では以下のものを指定できます。

- 整数
- 浮動小数点数
- 算術関数
- 一部のマクロ変数や浮動小数点変数

また、半角のコロン(:)で区切って複数の演算式を一度に記述することもできます。その場合、左側から順に解釈・実行されます。

なお、内部での演算は倍精度浮動小数点数(double)で行います。

引数

- vptr 演算の結果としての文字列を得る配列変数内の位置
- ptr1 演算の結果を文字列化する際の書式を指定する書式指定文字列

形式

書式指定文字列の中に、半角の % 記号で始まる以下のパターンを置くことによって、演算結果の数値(浮動小数点数)をその場所に出力できます。なお、ptr1 にヌル文字列を指定し

た場合には、第1引数 `vptr` は無視されます(演算結果の文字列出力は行われず)。特に書式を指定する必要がない場合は、"`%18.10G`"と指定してください。

```
%[flag][width][.precision]type
```

flag

- 指定されたフィールド幅に結果を左詰めする。
- + 出力値が符号付きの場合は、出力値の前に + または - の符号を付ける。
- 0 最小幅 `width` まで 0 が付加される。
- # 出力値に強制的に小数点を入れる。

width

出力する最小文字数(フィールド幅)を指定する。

precision

オプションの数字出力フィールドの全体または一部に表示する最大文字数、または整数値として表示する最小桁数を指定する。

type

- e 符号付きの浮動小数点数で出力。指数の前には `e` が付く。
- E 指数の前に `E` が付く点を除いて `e` の書式と同じ。
- f 符号付きの固定小数点数で出力。小数点の前の桁数はその数の絶対値によって決定され、小数点の後の桁数は要求された精度によって決定される。
- g `f` または `e` の書式のうち、値および精度を表現できる短い方の書式。
- G 指数の前に付くのが `e` ではなく `E` である点を除いて、`g` の書式と同じ

`ptr2` 計算したい演算式を表す文字列。演算式は以下に示す関数、演算子、変数、定数を用いて記述します。また、演算子の優先順位を明確に指定するために半角の小括弧 () を使用することもできます。さらに、複数の演算式を一度に指定する場合には、半角のコロン : で区切って指定します。

関数

関数名は半角で指定します。大文字、小文字のどちらでも構いません。

<code>FABS(x)</code>	<code>x</code> の絶対値
<code>ABS(x)</code>	<code>x</code> の絶対値
<code>SINH(x)</code>	ハイパブリックサイン、 <code>x</code> はラジアン
<code>COSH(x)</code>	ハイパブリックコサイン、 <code>x</code> はラジアン
<code>TANH(x)</code>	ハイパブリックタンジェント、 <code>x</code> はラジアン
<code>ASIN(x)</code>	アークサイン、 $-1 \leq x \leq 1$
<code>ACOS(x)</code>	アークコサイン、 $-1 \leq x \leq 1$
<code>ATAN(x)</code>	アークタンジェント
<code>ATN(x)</code>	アークタンジェント
<code>SIN(x)</code>	サイン、 <code>x</code> はラジアン
<code>COS(x)</code>	コサイン、 <code>x</code> はラジアン
<code>TAN(x)</code>	タンジェント、 <code>x</code> はラジアン

LOG10(x)	常用対数、x は正の数
LOG(x)	自然対数、x は正の数
LN(x)	自然対数、x は正の数
EXP(x)	e の x 乗、e=2.718281828
SQRT(x)	x の平方根、x は正の数
SQR(x)	x の平方根、x は正の数
CEIL(x)	x 以上の最小の整数
FLOOR(x)	x 以下の最大の整数
INT(x)	x 以下の最大の整数
RAD(x)	° 60 分法 からラジアン変換
DEG(x)	ラジアンから ° 60 分法 へ変換
!(x)	x の階乗、1 <= x <= 170 の整数
POW(x,y)	x の y 乗
FMOD(x,y)	x を y で割った剰余

演算子

・ビット演算子

以下の記号はすべて半角で指定してください。

x	x の 2 の補数
x ^ y	x と y のビット排他的論理和
x&y	x と y のビット論理積
x y	x と y のビット論理和
x<<y	x を y ビット左シフト
x>>y	x を y ビット右シフト



オペランド x , y は、演算に先立ち 32 ビット符号なし整数に変換されます。

・算術演算子

以下の記号はすべて半角で指定してください。

- x	x の負の数 (0 - x)
! x	x の階乗、1 <= x <= 170 の整数
x%y	x を y で割った剰余 (浮動小数点の剰余)
x*y	x に y を乗じた値 (乗算)
x / y	x を y で割った値 (除算)
x + y	x に y を加えた値 (加算)
x - y	x から y を引いた値 (減算)

・代入演算子

x=y 式 y の値を変数 x に代入する



左辺の変数が @1 ~ @16、@@1 ~ @@16 の場合、右辺の式の値を 32 ビット符号付き整数に変換して代入します。このとき、右辺の式の値が 32 ビット符号付き整数の範囲内 (-2,147,483,648 ~ +2,147,483,647) でない時には演算エラーとなります。

変数

@1 ~ @16	32ビット符号付き整数型の変数 (マクロ言語のグローバル単純変数)
@@1 ~ @@16	32ビット符号付き整数型の変数 (マクロ言語のローカル単純変数)
@F0	64ビット倍精度浮動小数点型の変数 演算の結果が自動的にこの変数に格納されます。
@F1 ~ @F4	64ビット倍精度浮動小数点型の変数



@F0、および @F1 ~ @F4 は、calc()関数の中でのみ使用可能なdouble型の変数ですが、@1 ~ @16と同様に静的な変数なので、calc()関数の実行後もその値は保存されています。そのため、以降のcalc()関数の演算式の中で値を参照することができます。

定数

・ 16進定数

頭に0x または\$ を付ける。16進定数は32ビット符号なし整数と見なされる。これは、8桁を超える16進定数の記述は意味がないことを示す。

例： 0x1800 \$5b

・ 2進定数

頭に0b を付ける。2進定数は32ビット符号なし整数と見なされる。これは、32桁を超える2進定数の記述は意味がないことを示す。

例： 0b10111111 0B000011001011

・ 10進定数：整数、固定小数点数、浮動小数点数

10進定数は64ビット倍精度浮動小数点数と見なされる。小数点や指数部のない整数形式の場合でも倍精度浮動小数点数と見なされる。これは、32ビット整数の範囲(-2,147,483,648 ~ +2,147,483,647)を超える整数値も記述可能であることを示す。

例： 1057 -128 0.09876 125.123 -1.4578e12 3.87698e-5

・ 特殊定数

3.14159
PAI 3.14159

関数値

演算の結果として得られた文字列の文字数を関数値に返します。0以上なら演算が正常に実行されたことを示します。負の数の場合、-1または-2の場合には、演算式の記述に何らかの誤りがあることを示します。

-10の場合には、演算式の計算の際に何らかのエラー(オーバーフロー、アンダーフロー、0割りなど)が発生したことを示します。なお、演算の結果としての値は、専用の変数 @F0 に自動的に格納されます。

例

```
・ calc(@str1,"%G","pow(1.017,30)")
・ calc(@str1,"%20.12g","@f1=2.3e2:@f2=56.3:@f3=sqrt(@f1*@f1+@f2*@f2)")
・ calc(@str2,"%18.10G","!(@f3)")
```



この calc()関数は、DOS 版 MIFES に付属していた電卓ツール MICAL.EXE の内部処理を移植したもので、演算式の仕様は基本的に MICAL.EXE と同様です。

getdir(val,vptr) ディレクトリ名の取得

val で指定したディレクトリ名を vptr に代入します。

引数

val には以下のいずれかの値を指定します。

- 0: カレントディレクトリ名を代入する
- 1: ユーザー定義ディレクトリ 1 名を代入する
- 2: ロードディレクトリ名を代入する
- 3: Windows ディレクトリ名を代入する
- 4: MIW.EXE のあるディレクトリ名を代入する
- 5: マイドキュメント・ディレクトリ名を代入する
- 6: 現在編集中のファイルのあるディレクトリ名を代入する
- 11: ユーザー定義ディレクトリ 1 のディレクトリ名を代入する
- 12: ユーザー定義ディレクトリ 2 のディレクトリ名を代入する
- 13: ユーザー定義ディレクトリ 3 のディレクトリ名を代入する
- 14: ユーザー定義ディレクトリ 4 のディレクトリ名を代入する
- 15: ユーザー定義ディレクトリ 5 のディレクトリ名を代入する
- 16: ユーザー定義ディレクトリ 6 のディレクトリ名を代入する
- 17: ユーザー定義ディレクトリ 7 のディレクトリ名を代入する
- 18: ユーザー定義ディレクトリ 8 のディレクトリ名を代入する

代入するディレクトリ名は ¥ で終わる文字列です。

このときのユーザー定義ディレクトリとは、「ファイルを開く」ダイアログボックスまたは「保存するファイル名の指定」ダイアログボックスで設定しているディレクトリです。

ファイラで設定しているホームディレクトリではありません。

関数値

代入したディレクトリ名の文字数を返します。バイト数ではありません。

この値は最後の ¥ の次の位置を表します。

例

```
getdir(2,@str1)
```

ロードディレクトリ名を @str1 に得ます。

ロードディレクトリが "C:¥miw" のときは、@str1 に "C:¥miw¥" を代入します。

このときの関数値は 7 です。



マイドキュメント・ディレクトリ名について詳しくは、ヘルプを参照してください。

getenv(ptr,vptr)

環境文字列の取得

環境変数に定義された文字列を vptr に代入します。

引数

ptr には環境変数名を大文字で指定します。

関数値

代入した環境文字列の文字数を返します。バイト数ではありません。

環境変数が定義されていない場合は、関数値に 0 を返します。

例

```
getenv("TEMP",@str1)
```

@str1 に環境変数 TEMP に定義された文字列を代入します。

環境変数 TEMP が定義されている場合は、その文字列(通常はディレクトリ名)の文字数を関数値に返し、定義されていない場合は 0 を返します。

gethistory(val,vptr)

検索履歴文字列の取得

検索文字列履歴の中から、指定した val 番目の文字列を vptr に代入します。

引数

val には 0 ~ 29 の値が指定できます。値が小さいほど、新しく検索した文字列になります。つまり、一番新しく検索した文字列を代入するには 0 を指定します。反対に履歴の中の一番古い文字列を代入するには 29 を指定します。

関数値

代入した検索文字列の文字数を返します。バイト数ではありません。

説明

vptr の最後の配列要素には文字列の終わりを表すヌルコード(0)が入ります。その直後の配列要素には検索方法を表す値(0 ~ 5)が入ります。さらにその直後の配列要素に、拡張された検索方法を表す値が入ります。この拡張された検索方法の値は、通常の検索方法の値に、「語単位で探す」場合にだけ 16(0x0010)を加えたものです。

関数値でヌルコードの位置がわかりますので、その次の配列要素からその文字列を検索したときの検索方法がわかります。さらに、その次の配列要素から、拡張された検索方法がわかります。

検索した文字列の履歴の中に、指定した文字列がないときには、関数値に 0 を返します。i 番目の検索文字列を指定して関数値に 0 が返された(i 番目に指定した文字列がなかった)場合には、i ~ 29 番目までの検索文字列が記録されていないことを意味します。

また、検索文字列の履歴全体を削除するには、valに-1を指定します。

例

```
@1 = gethistory(0,@str2) ;一番新しい検索文字列を代入する
@2 = @str2[@1+1] ;検索方法を@2に代入する
search(0,@2,@str2) ;代入した検索文字列で検索実行
```

@1には指定した検索文字列の文字数が入ります。@2に@str2[@1+1]を代入すると検索方法が@2に得られます。検索方法の0~5の値についてはsearch関数を参照してください。

getline(vptr)

カーソル行の取得

カーソル行の内容をvptrに代入します。

カーソルのある表示行の行頭から、その行の論理行末まで(改行文字含む)の内容を代入します。

関数値

取得した文字列の文字数を返します。バイト数ではありません。

説明

行が長く、配列変数が足りない場合など、論理行末までの文字列を代入できなかった場合には、システム変数@nextbyteに代入できなかった部分の先頭文字のバイト位置を返します。論理行末までのすべての内容を代入できた場合には@nextbyteに0を返します。

例

```
getline(@str2) ;@str2にカーソル行の内容を代入
```

getpath(val,vptr)

編集ファイル名の取得

編集中のファイルのフルパス名をvptrに代入します。

引数

valにはテキスト番号(0~99)を指定します。カレントウィンドウのパス名を代入するときには、valに-1を指定します。

「ウィンドウ一覧」ダイアログボックスの中で「」マークの付いたウィンドウのパス名を代入するときには、valに-2を指定します。

関数値

代入したパス名の文字数を返します。バイト数ではありません。

説明

有効なパス名が代入できない場合には関数値に0を返します。有効なパス名が得られた場合、vptrの最後の配列要素にはパス名の終わりを表すヌルコード(0)が入ります。その直後の配列要素には指定したパス名のファイルのテキスト番号を返します。

例

```
@1=getpath(-2,@str1)
if @1 then @2=@str1[@1+1]
```

@str1 には「ウィンドウ一覧」ダイアログボックスの中で「 」マークの付いたウィンドウのパス名を代入します。@1 には得られたフルパス名の文字数を代入し、@2 にはパス名を取得したウィンドウのテキスト番号を代入します。



注意

MIFES では編集するファイル名はロングファイル名として扱っています。

ファイル名中の半角の英大文字 (A ~ Z) と英小文字 (a ~ z) は区別して記録していますが、認識上は同一視します。



メモ

カレントウィンドウのテキスト番号はシステム変数 @text で調べられます。

そのため、getpath(@text,@str1) と指定することもできます。

getprof(val,vptr)

カスタマイズ情報の取得

起動時に読み込んだカスタマイズファイル (MIW.INI) 中のマクロ専用キーワード MACRODEF1 ~ MACRODEF16 に定義した数値と文字列を指定した配列変数に代入します。

引数

val キーワードID (1 ~ 16)
vptr 文字列を代入する配列変数内位置

関数値

指定したキーワードで定義した値を返します。この値が -1 (0xffffffff) のとき、指定したキーワードが未定義であることを意味します。

例

```
@12=getprof(1,@str4)
```



参照

システム関数 setprof もあわせて参照してください。

getselstring(vptr)

選択文字列の取得

現在の範囲選択中の文字列を vptr に代入します。最大 1023 文字まで代入できます。ただし箱型選択中の文字列は代入できません。

関数値

取得した文字列の文字数を返します。バイト数ではありません。

例

```
@2=getselstring(@str3)
```

範囲選択している文字列を @str3 に代入します。@2 には取得した文字列 @str3 の文字数を返します。

getstring(vpstr,val)

カーソル位置からの文字列を取得

現在のカーソル位置から指定文字数 val 分の文字列を配列変数位置 vpstr に取り出します。

val には 1 ~ 1023 までの文字数が指定できます。

改行文字も 1 文字として取り出します。

関数値

取得した文字列の文字数を返します。バイト数ではありません。

この値は通常、第 1 引数の val の値と一致しますが、それより小さくなることもあります。

getword(vpstr)

カーソル位置の 1 語取得

カーソル位置の 1 語を vpstr に代入します。

カーソル位置がデリミタ文字の場合にはカーソル位置の直前の 1 語を代入します。

関数値

代入した文字列の文字数を返します。バイト数ではありません。

有効な 1 語が取得できない場合には 0 を返します。

例

```
@2=getword(@str3)
```

@str3 にカーソル位置の 1 語を代入します。@2 には取得した 1 語 @str3 の文字数が得られます。

sprintf(vpstr,str,arg,,,arg)

書式文字列の取得

指定した書式 str に従って文字列を変換し、配列変数位置 vpstr に出力します。

引数

出力する書式を指定する文字列定数 str は、Windows の API の sprintf() と基本的に同じです。ただし、%d と %ld、%x と %lx、%u と %lu、および %i と %li は、それぞれ同じ意味になります。また、sprintf() にはない文字列の書式化方法として、%e と %E が使用できます。

例：&@str1[10]

%s に対応する arg には、文字列が格納された配列変数位置を指定します。
配列変数位置とは、要素番号の指定を省略した配列変数名、または配列要素名にアドレス演算子&を付けたものです（例 . &str1[10]）。
なお、%s に対応する arg に文字列定数は指定できないので注意してください。

%e または %E に対応する arg には、文字列が格納された配列変数位置を指定します。
なお、%e または %E に対応する arg に文字列定数は指定できないので注意してください。

%d、%x、%u、%c に対応する arg には、文字列定数や配列変数位置は指定できません。

書式指定文字列 str には、メタ文字（%n、%t、%x?? など）も指定できます。

関数値

得られた文字列の文字数を返します。バイト数ではありません。
vptr の最後の配列要素には文字列の終わりを表すヌルコード（0）が入ります。
関数値でこのヌルコードの位置がわかります。

形式

書式を指定する第二引数 str は、以下の形式の文字列で指定します。
なお、[と] で囲まれた内容は省略できます。

%[-][#][0][width][.precision]type

-（半角のハイフン）

出力値を左揃えします。この指定を省略した場合は右揃えになります。

#

出力値の直前に16進数のプレフィックス 0x または 0X を出力します。
この指定は type に x または X を指定した場合にのみ有効です。

0（ゼロ）

出力値の前に必要な数だけの0を付けて出力幅を埋めます。この指定を省略した場合は、半角スペースで出力幅を埋めます。

width

出力値の最小の出力幅を半角文字の文字数で指定します。
この指定を省略した場合、precision の指定に基づいて出力値のすべての文字が出力されます。

precision

出力値の精度を最小の桁数で指定します。この指定を省略した場合、精度は1桁になります。

type

対応する引数の型と書式化の方法を指定します。以下の文字が指定できます。

- c 対応する引数の数値を文字コードとする1文字を出力します。
引数に指定する文字コードは、半角と全角を区別しません。
- d,i 対応する引数の数値を符号付き10進整数にして出力します。
- u 対応する引数の数値を符号なし10進整数にして出力します。

- x,X 対応する引数の数値を符号なし 16 進整数にして出力します。
- s 対応する引数の配列変数位置からの文字列をそのまま出力します。
- e 対応する引数の配列変数位置からの文字列を出力します。
ただし、漢字はEUCによる小文字の16進表記に変換して出力します。
例: "EUC 漢字" を "%e" で出力 "EUC%b4%c1%bb%fa" と出力される
- E 対応する引数の配列変数位置からの文字列を出力します。
ただし、漢字はEUCによる大文字の16進表記に変換して出力します。
例: "EUC 漢字" を "%e" で出力 "EUC%B4%C1%BB%FA" と出力される

例

```
sprintf(@str3, "%s / ", @str1)
```

strcmp(ptr1, ptr2)

文字列の比較：区別

2つの文字列(ptr1とptr2)を比較します。
大文字と小文字を区別して比較します。

関数値

2つの文字列の文字コードが一致した場合は0を返します。
意味は次のとおりです。

- 関数値 < 0 ptr1の方が小さい
- 関数値 = 0 ptr1とptr2は一致
- 関数値 > 0 ptr1の方が大きい

例

```
if strcmp("sample", @str4) > 0
```

strcmpi(ptr1, ptr2)

文字列の比較：同一視

2つの文字列(ptr1とptr2)を比較します。大文字と小文字は区別せずに比較します。
この関数はWindowsのAPIのlstrcmpi()と同じ結果を返します。

関数値

2つの文字列の文字コードが一致した場合は0を返します。
意味は以下のとおりです。

- 関数値 < 0 ptr1の方が小さい
- 関数値 = 0 ptr1とptr2は一致
- 関数値 > 0 ptr1の方が大きい

例

```
if strcmpi("sample", @str4) > 0
```

strcpy(vptr,ptr)

文字列の複写

文字列 ptr を配列変数 vptr に複写します。

関数値

複写した文字数を代入します。バイト数ではありません。

例

```
strcpy(@str4, "CALC.EXE ")
```

このときの関数値は8です。

strlist(vptr,str,,,str)

文字列並びの生成

文字列の並びを生成し、vptr に代入します。

指定する文字列の並びには制限はありません。ただし、必ず文字列定数を指定します。文字列定数にメタ文字は指定できません。

この関数は、breplace 関数、combobox 関数、listbox 関数の引数に与える文字列の並びを生成するために使用します。

関数値

生成した文字列の並びの文字数-1の値を返します。

文字列は次のようになっています。

文字列0文字列0文字列00

文字列の並びは 0(ヌル)で区切り、並びの最後には 0(ヌル)が入ります。

関数値はこの最後の2つの0のうちの2つ目の0の位置を表します(上記下線部)。この値を使うと、生成した文字列の並びの後に、さらに文字列を追加できます。

例

```
strlist(@str1, "通常検索", "ワイルドカード検索", "正規表現", "あいまい検索")
```

```
@5 = listbox("検索方法を選択してください", 0, @str1)
```

文字列の並びとは、複数の文字列をヌルコード(00H)で区切って配列変数内に並べたもので、その終端マークは2つのヌルコード(00H,00H)になります。

文字列の並びには制限はありませんが、配列変数内に終端マークまでのすべてを収めなければなりません。

ただし、combobox 関数、listbox 関数の最大文字列数は30です。これらの関数の引数として使う場合に、strlist 関数で30以上の文字列を生成しても無視されますので注意してください。



メモ

strncpy(vptr,ptr,val) 指定文字数の文字列複写

文字列またはワードデータ(符号なし16ビット整数)を指定した val 分の長さだけ複写します。

引数

vptr には複写先の配列変数位置を指定します。

val には 1 ~ 1024 の値を指定します。

複写元 ptr に文字列定数を指定した場合、文字列を指定文字数 val 分だけ複写します。文字列が指定文字数よりも短い場合にはその文字列全体(最後のヌル文字も含む)を複写します。

複写元 ptr に配列変数内位置を指定した場合には、指定ワード数 val 分のデータを複写します。

関数値

複写したワード数を返します。

例

```
strncpy(@str2,&@str4[256],128)
```

2. 文字列の挿入, 削除関数

copy() 選択範囲の複写

現在選択中の範囲をカットバッファにコピーします。

関数値

以下のコピーした種類を返します。

- 0: コピーしなかった
- 1: 箱型のコピー
- 2: 行単位のコピー
- 3,4: 文字列単位のコピー



範囲選択状態にするには、システム変数 @selmode に 1 ~ 4 の値を代入します。
詳しくは @selmode を参照してください。

cut() 選択範囲の切り取り

現在選択中の範囲を切り取ります。

関数値

実行した切り取りの種類を表す以下の値を返します。

- 0: 切り取りしなかった
- 1: 箱型の切り取り
- 2: 行単位の切り取り
- 3,4: 文字列単位の切り取り



範囲選択状態にするには、システム変数 @selmode に 1 ~ 4 の値を代入します。
詳しくは @selmode を参照してください。

delbyte(val)

指定バイト数の削除

カーソル位置から指定バイト数分の文字列を削除します。

引数

val に指定する値によって以下の文字列を削除します。

val = 1 ~ 500 カーソル位置から val バイト分を削除

val = 0 カーソル位置からの 1 語分を削除

val = -1 カーソル位置から論理行末までを削除

関数値

削除した文字列の文字数を返します。バイト数ではありません。



val に -1 を指定しても、カーソル位置から論理行末までが 500 バイト以上の場合、削除できません。このときの関数値には 0 を返します。

500 バイトを超える範囲を削除する場合は、以下のように記述してください。

@selmode = 3 ;範囲選択状態に入る

@byte += 1000 ;カーソル位置から 1000 バイトを選択

cut() ;選択範囲を削除する

delchar(val)

指定文字数の削除

カーソル位置から指定文字数分の文字列を削除します。

引数

val に指定する値によって以下の文字列を削除します。

val = 1 ~ 500 カーソル位置から val 文字分を削除

val = 0 カーソル位置からの 1 語分を削除

val = -1 カーソル位置から論理行末までを削除



val に -1 を指定しても、カーソル位置から論理行末までが 500 バイト以上の場合、削除できません。このときの関数値には 0 を返します。

delline()

カーソル行の削除

カーソルのある行(表示行の1行分)を削除します。

関数値

削除した文字列の文字数を返します。バイト数ではありません。

dupline(val)

カーソル行の2重化

カーソルのある行(論理行)をその場で val 行分複写します。

1行分の複写動作は正確には以下のような動作になります。

カーソルのある論理行の中のカーソルのある表示行以降の部分だけを複写します。ただし、この部分が500バイトを超える場合、複写するのは500バイトを超えた最初の表示行末までの部分だけです。注意してください。

引数

val には 1 ~ 10000 の値を指定できます。

関数値

複写した行数 val を返します。

insstr(ptr)または(val)

文字の挿入

指定した文字列、または文字コードをカーソル位置に挿入します。

文字列定数のときは、改行文字は `¥n`、ハードタブは `¥t` と指定します。

配列変数のときは、改行文字を `0x0d0a`、ハードタブを `0x0009` と指定します。

例

- ・ `insstr("windows¥t ")` ;windows とタブを挿入
- ・ `insstr(0x0d0a)` ;改行を挿入



`insstr(val)`関数は指定の文字コードの文字をカーソル位置に挿入します。

文字をカーソル位置に上書きする場合には、`@insmode` と `@code` を使います。カーソルを上書き状態にしてから、指定の文字コードをシステム変数 `@code` に代入します。

例

```
@insmode = 0
@code = ' A '
```

insstrex(val,ptr)または(val1,val2)

別のウィンドウに文字を挿入

指定した文字列または文字コードを、指定したテキスト番号のウィンドウのカーソル位置に挿入します。

この関数を使うと、カレントウィンドウ以外のウィンドウに文字列を挿入できます。

引数

val、val1 にはテキスト番号を指定します。

ptr には文字列、val2 には文字コードを指定します。

文字列定数のときは、改行文字は `¥n`、ハードタブは `¥t` と指定します。

配列変数のときは、改行文字を `0x0d0a`、ハードタブを `0x0009` と指定します。

outaseries(val)

等差文字列の出力

等差数字列を指定回数分カーソル位置に挿入します。

等差数字列の初期化には、システム関数 `setaseries` を使います。

関数値

挿入した文字列の総バイト数を返します。

paste(val)

文字列の貼り付け

カットバッファ中の文字列をカーソル位置に貼り付けます。

引数

val には以下の種類を指定します。

0: 行カットバッファから貼り付け

1: クリップボードから貼り付け

10: 箱型カットバッファから箱型に挿入

11: 箱型カットバッファから箱型に上書き

12: 箱型カットバッファから文字列状に挿入

関数値

貼り付けを実行した場合には 1 を返します。実行できなかった場合には 0 を返します。

printf(str,arg,,,arg)

書式文字列の挿入

指定した書式 `str` に従って文字列を変換し、カーソル位置に挿入します。

引数

書式を指定する文字列定数 `str` は Windows の API の `wsprintf()` と同じです。ただし、`%d` と `%ld`、`%x` と `%lx`、`%u` と `%lu`、および `%i` と `%li` は、それぞれ同じ意味になります。

%s に対応する arg には、必ず配列変数位置を指定します。配列変数位置とは要素番号の指定を省略した配列変数名、または配列要素名にアドレス演算子&を付けたものです(例 &@str1[10])。

また、%d、%x、%u、%c に対応する arg には、配列変数位置や文字列定数は指定できません。str にはメタ文字(¥n、¥t、¥x?? など)も指定できません。

関数値

挿入した文字列のバイト数を返します。文字数ではありません。

形式

書式を指定する文字列の形式は以下のとおりです。[と] で囲まれた内容は省略できます。

%[-][#][0][width][.precision]type

- (半角のハイフン)

出力値を左揃えします。この指定を省略した場合は右揃えになります。

#

出力値の直前に16進数のプレフィックス 0x または 0X を出力します。この指定は type に x または X を指定した場合にのみ有効です。

0 (ゼロ)

出力値の前に必要な数だけの 0 を付けて出力幅を埋めます。この指定を省略した場合は、半角スペースで出力幅を埋めます。

width

出力値の最小の出力幅を半角文字の文字数で指定します。この指定を省略した場合、precision の指定に基づいて出力値のすべての文字が出力されます。

precision

出力値の精度を最小の桁数で指定します。この指定を省略した場合、精度は1桁になります。

type

対応する引数の型と書式化の方法を指定します。以下の文字が指定できます。

- c 引数の数値を文字コードとする1文字を出力
引数に指定する文字コードは、半角および全角を区別しません。
- d,i 引数の数値を符号付き10進整数にして出力
- u 引数の数値を符号なし10進整数にして出力
- x,X 引数の数値を符号なし16進整数にして出力
- s 引数の文字列定数または配列変数位置からの文字列を出力

例

```
printf(" 値(10進)=%d 文字列=%s¥n ",@10,@str4)
```

setaseries(ptr,val1,val2)

等差数字列の初期化

等差数字列を初期化します。

引数

ptr 書式制御文字列(最大31バイト)
val1 初期値
val2 公差

ptr はWindowsのAPIの `wsprintf()` と同様な書式制御文字列で指定します。その一般形は以下のとおりになります。[] で囲まれた中は省略可能です。

< と > で囲まれた中はカンマで区切られた中の1つを指定します。

[プレフィックス文字列] %[- #] 0 フィールド最小文字数 [ld,ix,IX,lu] [ポストフィックス文字列]

書式制御文字列は全体で最大31バイトまで指定できます。

プレフィックス文字列とポストフィックス文字列にはどんな文字列でも指定できます。ただし改行文字は `¥n` で、タブ文字は `¥t` で指定します。

- (半角のハイフン)

指定したフィールド内で左詰めにするを指定します。指定しない場合は右詰めになります。

#

16進数の場合(`ix,IX`)に、小文字(`ix`)なら `0x` を、大文字(`IX`)なら `0X` を前に置くことを指定します。

0 (ゼロ)

フィールドの余った部分を0で埋めることを指定します。指定しない場合には、フィールドの余った部分は半角の空白で埋めます。

フィールド最小文字数

出力する最小の文字数を指定します。出力値の文字数がこの値より小さい時には、余った部分は半角の空白または0で埋めます。出力値の文字数がこの値より大きくなっても出力値の文字数が切り詰められる訳ではありません。

`ld` は符号付き10進数での出力を指定します。`ix` は小文字による16進数での出力を指定します。`IX` は大文字による16進数での出力を指定します。また、`lu` は符号なし10進数での出力を指定します。

関数値

正しく初期化できた場合は1を返します。指定したパラメータ(書式制御文字列)にエラーがあれば0を返します。

setstring(val,ptr)

カーソル位置の文字列置き換え

現在のカーソル位置から指定した文字数 val 分の文字列を、指定した文字列 ptr に置き換えます。val に 0 を指定すれば文字列 ptr の挿入になります。また、ptr にヌル文字列を指定すれば val 文字の文字列削除になります。

引数

val 置き換えたい文字数（バイト数ではありません）
ptr 置き換える文字列

関数値

関数値には、実際に置き換えられた文字数が返されます。この値は、val の値よりも小さいこともあります。この関数実行後のカーソル位置は、置き換えた文字列の直後の文字の位置になります。

例

```
while 1
  @1=search(0,0,"ファイル")
  if @1 <= 0 then break
  setstring(@1,"ファイル")
wend
```

3. ジャンプ，移動，検索，置換関数

`breplace(val1,val2,vptr1,vptr2)`

複数置換

複数置換を実行します。

引数

`val1` ビット0 置換時の確認操作
 0x00:確認あり (REP_QUERY)
 0x01:確認なし (REP_BATCH)
 ビット4～5 置換の範囲
 0x00:ファイル全体
 0x10:カーソル位置以降
 0x20:ファイル全体
 0x30:選択範囲 (選択状態でない時はカーソル位置以降)

`val2` 旧文字列の検索方法 (0～5)
 0x0000:通常検索:大文字・小文字区別
 0x0001:通常検索:大文字・小文字同一視
 0x0002:ワイルドカード検索:大文字・小文字区別
 0x0003:ワイルドカード検索:大文字・小文字同一視
 0x0004:正規表現検索
 0x0005:あいまい検索

なお、この値に0x0010を加えた値を指定すると、語単位で検索文字列を探します。

`vptr1` 旧文字列の並び (最大20文字列)

`vptr2` 新文字列の並び (最大20文字列)

旧文字列の並びとは、複数の旧文字列をヌルコード(0)で区切り最後が2つのヌルコード(0,0)で終わっている文字列の並びのことです。

新文字列の並びについても同様です。

実際には、文字列の並びを生成するためのシステム関数 `strlist()` を使用して文字列の並びを配列変数変数内に作成し、それを引数に指定します。

関数値

置換した文字列の数(0～)を返します。指定に誤りがある場合には負の数を返します。

例

```
strlist(@str3, "int", "char", "unsigned", "long")
strlist(@str4, "INT", "BYTE", "WORD", "LONG")
breplace(1,0,@str3,@str4)
```

@str3の文字列を@str4に一括で置換します(int INT,)

gsearch(val1,val2,ptr1,ptr2,ptr3)

グローバル検索

文字列のグローバル検索を実行します。

引数

val1	実行フラグ。この値の各ビットの意味は以下の通りです。
ビット0 ~ 2	開いているファイルの検索(0x00000003) 0:検索しない 1:一致 検索 2:全てを検索
ビット4	開いてないファイルの検索(0x00000010) 0:検索しない 1:一致 検索
ビット8	下位ディレクトリの検索(0x00000100) 0:検索しない 1:検索する
ビット11 ~ 12	検索結果の出力先(0x00001800) 0:グローバル検索結果ウィンドウに出力 1:リストウィンドウに出力 2:カレントウィンドウに追加
ビット13	ファイル内容による自動コード判定(0x00002000) 0:自動コード判定を行わない 1:自動コード判定を行う
ビット14	集計結果の出力(0x00004000) 0:出力しない 1:出力する
ビット16 ~ 19	検索の条件(0x000f0000) 0:文字列を含む行を探す(検索文字列 1 つ) 1:すべての文字列を含む行を探す(検索文字列 2 つ以上) 2:いずれかの文字列を含む行を探す(検索文字列 2 つ以上) 3: 1 番目と 2 番目の文字列を含み 3 番目の文字列を含まない 行を探す(検索文字列 2 つ以上) 4:文字列を含まない行を探す(検索文字列 1 つ) 5:いずれの文字列も含まない行を探す(検索文字列 2 つ以上)
ビット22	タグジャンプ時の一斉表示(0x00400000) 0:一斉表示なし 1:一斉表示あり
ビット24 ~ 27	タイムスタンプ条件(0x0f000000) 0:タイムスタンプ条件なし 1:タイムスタンプ条件なし 2:最近 1 時間に更新されたファイル 3:最近 3 時間に更新されたファイル 4:最近 1 日間に更新されたファイル 5:最近 1 週間に更新されたファイル 6:最近 1 ヶ月間に更新されたファイル 7:最近 1 年間に更新されたファイル
ビット28	結果の表示(0x10000000) 0:検索ファイルごとに結果を表示 1:結果は最後にまとめて表示
ビット29	検索結果の出力時の折り返し(0x20000000) 0:250 桁で折り返す 1:980 桁で折り返す
ビット30	検索結果のバイト位置出力(0x40000000) 0:バイト位置出力も許可 1:バイト位置出力は禁止

通常は以下のマクロ定数を指定します。

GS_NORMAL 下位ディレクトリも検索 (0x00000111)
 GS_1DIR 指定ディレクトリだけ検索 (0x00000011)
 GS_ALLOPEN 開いているファイルだけを検索 (0x00000002)
 GS_APPEND 結果をカレントウィンドウに追加出力する (0x00001000)

他のマクロ定数との論理和で使用する。

GS_UNICODE ファイル内容による自動コード判定を行う (0x00002000)
 GS_SUMUP 集計結果も出力する (0x00004000)
 GS_AND すべての文字列を含む行を検索 (0x00010000)
 GS_OR いずれかの文字列を含む行を検索 (0x00020000)
 GS_NOT 指定文字列を含まない行を検索 (0x00040000)
 GS_NOTBOTH いずれの文字列も含まない行を検索 (0x00050000)

val2 検索方法 (0 ~ 5)

0x0000:通常検索:大文字・文字区別
 0x0001:通常検索:大文字・小文字同一視
 0x0002:ワイルドカード検索:大文字・小文字区別
 0x0003:ワイルドカード検索:大文字・小文字同一視
 0x0004:正規表現検索
 0x0005:あいまい検索

なお、この値に0x0100を足した値を指定すると、指定された検索文字列を「検索文字列の履歴」には記録しません。

また、この値に0x0010を足した値を指定すると、語単位で検索文字列を探します。

ptr1 検索文字列

検索の条件(第1引数のビット16~19で指定)に検索文字列を2つ以上必要な条件を指定した場合には、この引数には検索文字列の並びを指定します。

検索文字列の並びとは、複数の検索文字列をヌルコード(0)で区切り、最後が2つのヌルコード(0,0)で終わっている文字列の並びのことです。

検索文字列の並びは配列変数位置(vptr)で指定します。

なお、検索文字列に文字列定数が指定された場合、検索の条件に検索文字列が2つ以上必要なものが指定されていても無視し、指定の文字列を含む行を検索します。

ptr2 検索ファイル名/ワイルドカード

半角スペースまたはセミコロンで区切って8つまで指定可能です。

ptr3 検索開始ディレクトリ位置

以下の特殊な文字列を指定することで、全ドライブ検索を指定することもできます。

“ ” ヌル文字列:全ドライブ検索(ローカルのみ)
 “ ? ” 半角の? 1文字で全ドライブ検索(リモート含む)
 “ * ” 半角の* 1文字で全ネットワーク検索

関数値

グローバル検索が実行されれば1が、実行されなければ0を返します。

また、引数の指定に誤りがあり、グローバル検索が実行されない場合には負の数を返します。

jump(val1,val2)または(0,ptr) ジャンプ, ウィンドウ切り替え

カーソルを指定した位置へジャンプします。

引数

val1 の値によって val2 の意味が異なります。

- val1 = 0 val2 の論理行番号(1 ~)へジャンプ
- val1 = 1 val2 の表示行番号(1 ~)へジャンプ
- val1 = 2 val2 のバイト位置(1 ~)へジャンプ
- val1 = 3 val2 のテキスト番号(0 ~ 99)のウィンドウへジャンプ

第 2 引数が ptr のときは、指定したファイル名のウィンドウにカーソルをジャンプします。このとき val1 は無視されます。ディレクトリ位置が異なる同名のファイルが複数開かれている場合を考慮して、ファイル名はフルパス名で指定します。単純ファイル名で指定した場合、ファイル名部分が一致する最も小さいテキスト番号のウィンドウにジャンプします。行番号やバイト位置を指定してジャンプするときにジャンプ先が存在しない場合、指定先のできるだけ近くにジャンプしますので注意してください。

関数値

正常にジャンプできた場合には 1 を返します。

ジャンプできなかった、または付近へジャンプした場合は 0 を返します。

例

- ・ jump(0,564) ;論理行番号へのジャンプ
- ・ jump(1,300) ;表示行番号へのジャンプ
- ・ jump(2,0x1000+1) ;バイト位置へのジャンプ
- ・ jump(3,3) ;指定したテキスト番号のウィンドウへ切り替える
- ・ jump(0, "a:%config.sys ") ;指定したパス名のウィンドウへ切り替える



以下のように、jump 関数を使わずに、システム変数に値を代入して指定の位置へカーソルをジャンプすることもできます。詳しくは各システム変数の説明を参照してください。

論理行番号へのジャンプ

システム変数 @num に論理行番号を代入します。

例 : @num = 1000

バイト位置へのジャンプ

システム変数 @byte にバイト位置を代入します。

例 : @byte = 1024

表示行番号へのジャンプ

システム変数 @line に表示行番号を代入します。

例 : @line = 1208

ウィンドウ間のジャンプ

システム変数 @text にウィンドウのテキスト番号を代入します。

例 : @text = 9

move(ptr)または(val)

カーソル移動

カーソルを移動します。

引数

ptr には以下の文字からなる文字列を指定します。

- u カーソルを 1 行上に移動する
- d カーソルを 1 行下に移動する
- l カーソルを 1 文字左に移動する
- r カーソルを 1 文字右に移動する
- U 罫線トレースしながらカーソルを 1 行上に移動する
- D 罫線トレースしながらカーソルを 1 行下に移動する
- L 罫線トレースしながらカーソルを 1 文字左に移動する
- R 罫線トレースしながらカーソルを 1 文字右に移動する
- O カーソル位置の罫線を最適化する
- < カーソルを 1 語後方()に移動する
- > カーソルを 1 語前方()に移動する
- (カーソルを行の左端に移動する
-) カーソルを行の右端に移動する

val には移動する行数を指定します。val が正の数なら 方向に移動し、負の数なら 方向に移動します。

例

- ・ move(" ddr ") ;カーソルを 2 行下に移動し、1 文字右に移動
- ・ move(-5) ;5 行 方向に移動

parenthesis(val)

対応する括弧の検索

現在のカーソル位置の括弧と対となる括弧を、現在のカーソルのある論理行から指定した論理行内で検索します。

現在のカーソル位置の文字が半角の括弧 ({ [] }) の場合のみ検索します。指定した行数内で対応する括弧が見つからない場合には、カーソルは移動しません。

関数値

対応する括弧が見つかった場合には 1 が、見つからなかった場合には 0 を返します。

例

parenthesis(50) ;カーソル位置から 50 行以内で検索

replace(val1,val2,ptr1,val3,val4,ptr2)

文字列の置換

文字列の置換を実行します。

引数

- val1 ビット0 置換時の確認操作
0:確認あり(REP_QUERY)
1:確認なし(REP_BATCH)
ビット4～5 置換の範囲
0x00:第4引数(val3)と第5引数(val4)で指定した行範囲
0x10:カーソル位置以降
0x20:ファイル全体
0x30:選択範囲(選択状態でない時はカーソル位置以降)
- val2 旧文字列の検索方法(0～5)
0x0000:通常検索:大文字・小文字区別
0x0001:通常検索:大文字・小文字同一視
0x0002:ワイルドカード検索:大文字・小文字区別
0x0003:ワイルドカード検索:大文字・小文字同一視
0x0004:正規表現検索
0x0005:あいまい検索
なお、この値に0x0100を足した値を指定すると、指定された検索文字列を「検索文字列の履歴」には記録しません。置換文字列も「置換文字列の履歴」には記録しません。
さらに、この値に0x0010を足した値を指定すると、語単位で旧文字列を探します。
- ptr1 旧文字列(検索文字列)
- val3 置換開始論理行番号(0、1～)
0を指定すると現在のカーソル位置から 方向に置換を開始します。
第一引数(val1)のビット4～5が0の時のみ有効。
- val4 置換終了論理行番号(～MAX_NUMBER)
マクロ定数MAX_NUMBERを指定すると、ファイルの最終行まで置換できます。
第一引数(val1)のビット4～5が0の時のみ有効。
- ptr2 新文字列(置換文字列)

関数値

置換した文字列の数(0～)を返します。

例

```
replace(1,0,"Windows95",1,MAX_NUMBER,"WindowsXP")
```

置換中に確認せずに、ファイルの先頭から最後まで文字列を置き換えます(Windows95
WindowsXP)。

search(val1,val2,ptr)

文字列の検索 / リストウィンドウ内の検索

指定した文字列を検索します。または、リストウィンドウ中で指定した項目名を検索します。

引数

val1 検索する方向

- 0:カーソル位置から 方向に検索(カーソル位置も検索)
- 1:カーソル位置から 方向に検索(カーソル位置も検索)
- 2:カーソルの次の文字から 方向に検索
- 3:カーソルの直前文字から 方向に検索
- 4:ファイルの先頭から 方向に検索
- 10:リストウィンドウのリスト中の項目名を探す(完全一致検索)
この場合はval2には必ず0または1を指定すること
- 11:リストウィンドウのリスト中の項目名を探す(前方一致検索)
この場合はval2には必ず0または1を指定すること

val2 検索方法

- 0x0000:通常検索:大文字・小文字区別
 - 0x0001:通常検索:大文字・小文字同一視
 - 0x0002:ワイルドカード検索:大文字・小文字区別
 - 0x0003:ワイルドカード検索:大文字・小文字同一視
 - 0x0004:正規表現検索
 - 0x0005:あいまい検索
- なお、この値に0x0100を足した値を指定すると、指定された検索文字列を「検索文字列の履歴」には記録しません。
- また、この値に0x0010を足した値を指定すると、語単位で検索文字列を探します。

ptr 検索文字列

関数値

通常の文字列検索の場合(val1=0~4)、見つかった文字列の文字数(バイト数ではない)を返します。

見つからなかった場合には0を返します。

検索文字列などに誤りがある場合には負の数を返します。

リストウィンドウのリスト中の項目名を検索する場合(val1=10~11)、項目名が見つかった場合には1が、見つからなかった場合には0が、リストウィンドウが表示されてない場合には-2が返されます。項目名が何を意味するのは、リストの種類により異なりますので注意してください。

- | | |
|----------------|--------------------|
| 「ファイル履歴」リスト | ファイル名(パス名ではない) |
| 「ディレクトリ」リスト | ファイル名(パス名ではない) |
| 「ファイル名検索結果」リスト | ファイル名(パス名ではない) |
| 「グローバル検索結果」リスト | ファイル名(パス名ではない) |
| 「C言語関数定義」リスト | 関数名 |
| 「見出し行検索結果」リスト | 見出し文字列(見出し行の最初の1語) |

なお、項目名が見つかった時、SDIモードの場合にはそのリストウィンドウを持つウィンドウにジャンプします。そして、リストウィンドウにフォーカスが移り、その項目が選択された状態になります。

MDIモードの場合には、リストウィンドウにフォーカスが移り、その項目が選択された状態になります。

例

```
search(2,4, "¥¥¥+")
```

正規表現検索で、行頭が¥で、¥が繰り返す文字列を検索します。



各検索方法で指定できるメタ文字については、ユーザーズマニュアルまたはヘルプを参照してください。

reform(ptr1,ptr2,val1)

文書整形の実行

指定の文書整形機能を実行します。

引数

ptr1 文書整形用の外部DLLのファイル名(*.REP)を指定します。または、MIFES 内部の文書整形機能を表す半角の英字1文字に続く半角のコロンを指定します。

- A: 半角の小文字(a~z)を大文字(A~Z)に変換します
- B: 半角の大文字(A~Z)を小文字(a~z)に変換します
- C: タブコード(09H)を相当する半角スペース(20H)へ変換します
- D: 半角スペース(20H)を相当するタブコード(09H)に変換します
- E: 折り返し位置の直前に改行文字(0DH,0AH)を挿入します
- F: 改行文字を削除します
- G: 論理行頭にある半角スペース、全角スペース、ハードタブをすべて削除します
- H: 論理行末にある半角スペース、全角スペース、ハードタブを全て削除します
- I: 論理行末にある半角スペース、全角スペース、ハードタブを全て削除します
- J: 半角のカタカナを全角のカタカナへ変換します
- K: 全角のカタカナを半角のカタカナへ変換します
- L: 全角文字に変換可能な半角文字をすべて全角文字に変換します
- M: 半角文字に変換可能な全角文字をすべて半角文字に変換します
- N: 論理行頭にその行の行番号文字列を付加します
- O: 論理行頭に[挿入する文字列(S)]にユーザーが入力した文字列を付加します
- P: 行の文字列を左寄せします
- Q: 行の文字列をセンタリングします
- R: 行の文字列を右寄せします
- S: HTMLのタグを削除します
- T: ウィンドウ一覧上で の付いたファイルを置換定義ファイルと見なし、この指定にしたがって編集集中のファイルの指定された範囲に対して置換を繰り返し実行します

ptr2 行頭に付加する文字列を指定します。ptr1 に 0 を指定した場合にだけ有効です。

val1 ビット0 ~ 3 0:カーソル行の整形
1:指定範囲の整形
2:ファイル全体の整形
ビット4 0:確認なし 1:確認あり

関数値

置換した文字列の数を返します(0 ~)。

DLL が見つからないなど、文書整形機能が実行できなかった場合には負の数を返します。

例

- ・ reform("A:", "", 0x02)
- ・ reform("ADJFIELD.REP", "", 0x02)

tagjump(val)

タグジャンプ、バックタグジャンプ

引数

val にはジャンプの種類を指定します。

0: タグジャンプ

1: バックタグジャンプ

2: リストウィンドウに表示されているリストが以下の場合に、現在選択している項目へジャンプ

- ・ グローバル検索結果
- ・ C 関数定義の検索
- ・ 見出し行の検索

関数値

正常にジャンプでき場合は0を返します。

ジャンプできない場合には0以外の値を返します。

4. ユーザー入力関数

combobox(vp1,ptr,val,vp2)

コンボボックスで文字列の入力

コンボボックスを表示します。

引数

vp1 入力した文字列を返す配列変数位置

ptr タイトル文字列

val デフォルト選択項目

vp2 項目文字列の並び

項目文字列の並びとは、複数の項目文字列をヌルコード(0)で区切り最後が2つのヌルコード(0,0)で終わっている文字列の並びのことです。

文字列の並びを生成するにはstrlist関数を使います。

コンボボックスに表示できる項目文字列は最大30個です。それ以上の文字列を指定していても表示できません。

関数値

入力した文字列の文字数を返します。バイト数ではありません。

なにも入力せずに[OK]ボタンを押したときは0を返します。

入力を中止したときは-1を返します。

項目文字列の並びに指定の誤りがあるときは-3を返します。

例

```
strlist(@str4, "東京都", "埼玉県", "栃木県", "群馬県")
combobox(@str2, "県名を入力してください", 0, @str4)
```

この場合、項目文字列の文字数がすべて3のため、combobox関数の関数値は、文字を入力せずに選択のみ行った場合には3です。

また、デフォルト選択項目は0を指定しているため、ユーザーが選択操作を行わずに単に[Enter]キーを押した(または[OK]ボタンをクリックした)場合には、@str2に文字列"東京都"が返されます。

fep(val)

日本語入力システム(FEP)のオン/オフ

日本語入力システム(FEP)のオンとオフを制御します。

引数

valには以下の値を指定します。

- 0: FEPをオフにする
- 1: FEPをオンにする
- 2: それまでのFEPの状態を反転する

関数値

それまでのFEPの状態を返します。オフのときは0、オンのときは1を返します。

finput(vpstr,ptr)

ファイル名の入力

ファイル名を入力するためのダイアログボックスを表示します。

引数

- vpstr 入力したファイル名を代入する配列変数
- ptr ダイアログボックスのタイトル文字列

関数値

ファイル名の文字数を返します。バイト数ではありません。なにも入力せずに [OK] ボタンをクリックしたときには0を返します。入力を中止したときには-1を返します。ファイル名は常にフルパス名で返されますので、関数値には、ユーザーが入力したファイル名の文字数ではなくて、フルパス名の文字数が返されます。

この関数を実行した直後は、この関数で表示したダイアログボックスで最後に表示したドライブとディレクトリが、カレントドライブおよびカレントディレクトリになります。また、この関数で表示するダイアログボックスは、【ファイル(F)】-【開く(O)】を選択したときに表示されるダイアログボックスです。

例

```
finput(@str4, "ファイル名を指定してください")
```

input(vpstr,ptr)

文字列の入力

文字列を入力するダイアログボックスを表示します。

引数

- vpstr 入力した文字列を代入する配列変数
- ptr ダイアログボックスのタイトル文字列

関数値

入力した文字数を返します。バイト数ではありません。

なにも入力せずに [入力] ボタンをクリックした場合には0を返します。入力を中止したときには-1を返します。

例

```
input(@str4, "文字列を入力してください")
```

ここで "マイフェス" と入力すると、@str4 に "マイフェス" と代入します。関数値は 5 になります。

listbox(ptr, val, vptr)

リストボックスでの選択

リストボックスを表示します。

引数

ptr	ダイアログボックスのタイトル文字列
val	デフォルト選択項目
vptr	項目文字列の並び

項目文字列の並びとは、複数の項目文字列をヌルコード(0)で区切り最後が2つのヌルコード(0,0)で終わっている文字列の並びのことです。

文字列の並びを生成するにはstrlist関数を使います。

リストボックスに表示できる項目文字列は最大30個です。それ以上の文字列を指定していても表示できません。

関数値

選択した項目の番号(0~29)を返します。選択を中止したときには-1を返します。

項目文字列の並びに指定の誤りがあるときは-3を返します。

例

```
strlist(@str4, "東京都", "埼玉県", "栃木県", "群馬県")  
@3=listbox("県名を入力してください", 0, @str4)
```

このとき、埼玉県を選択すると、listbox関数の関数値@3は1になります。

また、デフォルト選択項目は0を指定しているため、ユーザーが選択操作を行わずに[Enter]キーを押す(または[OK]ボタンを押す)と、東京都を選択したことになり、関数値は0になります。

messagebox(ptr1, ptr2, val)

メッセージボックスの表示

メッセージボックスを表示します。

引数

ptr1	表示するメッセージの文字列
ptr2	メッセージボックスのタイトル文字列
val	以下のマクロ定数のいずれかを指定します。

MB_OK	[OK] ボタン
MB_OKCANCEL	[OK] [キャンセル] ボタン
MB_YESNO	[はい] [いいえ] ボタン
MB_YESNOCANCEL	[はい] [いいえ] [キャンセル] ボタン
MB_ABORTRETRYIGNORE	[中止] [再試行] [無視] ボタン

以下のいずれかのマクロ定数を論理和で指定することもできます。

MB_ICONSTOP	[STOP] アイコン付き
MB_ICONQUESTION	疑問符アイコン付き
MB_ICONINFORMATION	[i] アイコン付き
MB_ICONEXCLAMATION	感嘆符アイコン付き

関数値

IDABORT	[中止] ボタンが押された
IDCANCEL	[キャンセル] ボタンが押された
IDIGNORE	[無視] ボタンが押された
IDNO	[いいえ] ボタンが押された
IDOK	[OK] ボタンが押された
IDRETRY	[再試行] ボタンが押された
IDYES	[はい] ボタンが押された

例

```
messagebox(" 指定に誤りがあります ", "エラー", MB_OK | MB_ICONSTOP)
```

STOP アイコンと [OK] ボタンのあるメッセージボックスが表示されます。このときの関数値は必ず IDOK になります。

stopoff()

マクロ中止ウィンドウの消去

マクロ中止ウィンドウを消去します。

マクロ中止ウィンドウを消去した後でも、[Pause] キーまたは [STOP] キーでいつでも実行中のマクロコマンドを中止させることはできます。

この関数は stopon 関数と対で使用します。

stopon(ptr1, ptr2)

マクロ中止ウィンドウの表示

マクロコマンドの実行を中止するためのウィンドウ(マクロ中止ウィンドウ)を表示します。

マクロ中止ウィンドウには、マクロコマンドの実行中に刻々と変化する状態を表示させることができます。そのため、マクロ中止ウィンドウに表示された内容を見ながら、マクロコマンドの実行を中止させることができます。

この関数は、処理に長い時間がかかるような場合に使用します。

引数

ptr1 マクロ中止ウィンドウのタイトル文字列
ptr2 マクロ中止ウィンドウ中に表示するメッセージ

すでにマクロ中止ウィンドウを表示しているときは、指定したメッセージ ptr2 だけを表示します。この場合、タイトルの指定 ptr1 は無視されます。

ptr2 は処理の途中で変化する状態などを知るためのメッセージとして使います。

マクロ中止ウィンドウの[中止]ボタンをクリックするか、または[ESC]キーを押すと、実行中のマクロコマンドを中止し、マクロ中止ウィンドウを消去します。なお、マクロ中止ウィンドウを使用しなくても、[Pause]キーでいつでも実行中のマクロコマンドを中止させることができます。



マクロ中止ウィンドウを表示している間は、マクロコマンドを中止する以外の操作はできません。なお、マクロコマンドをシングルステップで実行しているときには、この関数は無効です。

また、マクロコマンドの中で以下のシステム関数を実行したときには、これらの関数を実行する直前に、自動的にマクロ中止ウィンドウは消去されます。これらの関数はユーザーの入力を伴うか、その可能性がある関数です。

bp、breplace、callDll、child、combobox、execmd、finput、gsearch、input、listbox、messagebox、outprinter、replace、variables、waitevent

variables(ptr)

マクロ用ユーザー変数の表示と変更

「マクロ用ユーザー変数の表示と変更」ダイアログボックスを表示します。このダイアログボックスの上部に指定した文字列 ptr を表示します。

variables 関数を使うと、マクロコマンド実行中に値や文字列を変数に直接入力できます。

waitevent(val,vptr)

イベント待ち

指定したイベントを待ちます。

イベントには以下の種類があります。

1. コマンドの指定
2. 仮想キーの入力
3. 文字入力イベント
4. タイマーイベント
5. マウスイベント

引数

val 以下のマクロ定数のいずれか、またはそれらのビット論理和(！)を指定します。
EVENT_COMMAND コマンドの指定を待つ
EVENT_VK 仮想キー入力を待つ

EVENT_CHAR	文字の入力を待つ
EVENT_TIMER01	0.1 秒間待つ(タイマーイベント)
EVENT_TIMER1	1 秒間待つ(タイマーイベント)
EVENT_TIMER10	10 秒間待つ(タイマーイベント)
EVENT_TIMER60	60 秒間待つ(タイマーイベント)
EVENT_LBUTTON	左ボタンのクリックを待つ(マウスイベント)
EVENT_RBUTTON	右ボタンのクリックを待つ(マウスイベント)
EVENT_LDBLCLK	左ボタンのダブルクリックを待つ(マウスイベント)
EVENT_RDBLCLK	右ボタンのダブルクリックを待つ(マウスイベント)

vptr コマンドの指定または仮想キーの入力を待つとき、待ちたいコマンドの機能番号、および待ちたい仮想キーの仮想キー番号を配列変数内にセットし、その配列変数内の位置を vptr に指定します。

文字入力イベント、マウスイベント、またはタイマーイベントを指定する場合は省略できます。

コマンドの指定(EVENT_COMMAND)、または仮想キーの入力(EVENT_VK)を指定する場合は、配列要素にそれぞれの機能番号または仮想キー番号を指定します。最大64個まで指定できます。最後の配列要素には必ず0を代入します。

1. ユーザーが特定のコマンドを実行することを待つ場合(EVENTCOMMAND)、指定を待ちたいコマンドの機能番号を配列変数 vptr 内にセットします。配列変数内にセットするコマンドの機能番号として、以下のマクロ定数が指定できます。それ以外のコマンドは、機能番号を直接記述してください。

COMMAND_DEL	1文字削除
COMMAND_BS	直前の1文字削除
COMMAND_TAB	タブ文字挿入/次のタブ位置へ
COMMAND_RET	行分割/改行
COMMAND_UP	カーソル 移動
COMMAND_DOWN	カーソル 移動
COMMAND_LEFT	カーソル 移動
COMMAND_RIGHT	カーソル 移動
COMMAND_CUT	選択範囲のカット
COMMAND_COPY	選択範囲のコピー
COMMAND_PASTEL	行カットバッファのペースト
COMMAND_PASTES	クリップボードのペースト
COMMAND_INSMODE	挿入/上書の切り換え

ユーザーが、vptr に指定したコマンドのいずれかを実行しようとしたとき、waitevent 関数から返ります。このとき、関数値にはEVENT_COMMAND が返され、ユーザーが実行しようとしたコマンドの機能番号がシステム変数 @command に返されます。

2. ユーザーが特定のキーを押すのを待ちたい場合(EVENT_VK)、待ちたいキーの仮想キー番号を配列変数 vptr にセットします。配列変数内にセットする仮想キーの番号として、以下のマクロ定数が指定できます。それ以外の仮想キーは、仮想キー番号を直接記述します。

仮想キー番号については、システム変数 @key を参照してください。

VK_F1	F1 キー
VK_F2	F2 キー
VK_F3	F3 キー
VK_F4	F4 キー
VK_F5	F5 キー
VK_F6	F6 キー
VK_F7	F7 キー
VK_F8	F8 キー
VK_F9	F9 キー
VK_F10	F10 キー
VK_F11	F11 キー
VK_F12	F12 キー
VK_PAGEDOWN	PageDown キー
VK_PAGEUP	PageUp キー
VK_INS	INS キー
VK_DEL	DEL キー
VK_UP	キー
VK_LEFT	キー
VK_RIGHT	キー
VK_DOWN	キー
VK_HOME	Home キー
VK_END	End キー
VK_ESC	ESC キー

ユーザーが vptr に指定したキーのいずれかを押したとき、waitevent 関数から返ります。このとき、関数値には EVENT_VK が返され、ユーザーが押したキーの仮想キー番号がシステム変数 @key に返されます。

3. 文字の入力を待つ場合に指定します。
ユーザーが文字を入力したとき、関数値には EVENT_CHAR が返され、入力された文字の文字コードがシステム変数 @char に返されます。
4. タイマーイベントは、イベント待ちする時間を制限するときに指定します。
ダイアログボックスを表示しているときや、処理に時間のかかる機能を実行しているときに待ち時間が経過した場合には、それぞれの処理を終了した後にタイマーイベントが発生します。

関数値

指定したイベント指定マクロ定数のうち、実際に発生したイベントを示すマクロ定数が返されます。

タイマーイベントのマクロ定数が返された場合は、タイマーイベント以外のイベントが発生しなかったことを意味しています。



Windows 上でイベント待ち状態のときに、イベント待ちとして指定していないイベントが発生すると、指定していないイベントはすべて通常時(マクロコマンドを実行していない時)と同様に自動的に処理されます。すなわち、イベント待ち状態の間に、挿入/上書、範囲選択、カーソル位置などの、MIFES のいろいろな状態が変化することもあります。

例

```
@str4[0] = COMMAND_DEL;
@str4[1] = COMMAND_RET;
@str4[2] = VK_F1;
@str4[3] = 0;
@1 = waitevent(EVENT_COMMAND | EVENT_VK,@str4)
```

コマンドで「1文字の削除」または「行分割/改行」をイベント待ちします。または、仮想キーの[F1]キーの入力をイベント待ちすることを意味します。

仮にユーザーが1文字を削除しようとする、その削除を指定した操作が何であったかには関係なく、実際の削除は行われずに waitevent 関数から返ります。そして、@1 には EVENT_COMMAND が返され、システム変数 @command には COMMAND_DEL が返されます。



同時に複数のイベントを待っていた場合、どのイベントの発生により waitevent 関数から返ったのかを知るために関数値が重要です。

5. ファイル操作関数

chdir(ptr)

カレントディレクトリの変更

カレントディレクトリを指定のディレクトリ ptr に変更します。
ドライブ名を含むフルパス名で指定すると、カレントドライブも変更できます。

関数値

正しくカレントディレクトリが変更されると 0 を返します。



カレントディレクトリは、システム関数 finput を使っても変更できます。
また、カレントディレクトリ名は、システム関数 getdir で取得することができます。
詳しくは各システム関数の説明を参照してください。

例

```
・ chdir(" a:¥miw ")
  カレントドライブを a ドライブに変更し、カレントディレクトリを miw に変更します。
・ chdir(" c:¥temp¥¥ ")
  カレントドライブを c ドライブに変更し、カレントディレクトリを temp に変更します。
```

close()

ウィンドウを閉じる

現在編集中のウィンドウ(カレントウィンドウ)を閉じます。
閉じるウィンドウに変更がある場合も、確認のメッセージボックスを表示しないで閉じます。

copyfile(ptr1,ptr2,val)

ファイルの複写

ファイルを複写します。

テキストファイル以外のファイルも複写できます。

引数

ptr1 複写先のファイル名
ptr2 複写元のファイル名
val マクロ定数を指定します。

ptr1 にはドライブ名(:で終わる文字列)やディレクトリ名(¥で終わる文字列)も指定できます。ドライブ名やディレクトリ名だけを指定したときは、複写元のファイルのあるドライブやディレクトリに複写されます。

また、複写先のファイル名には現在編集中のファイル名は指定できません。

ptr2 の複写元のファイル名はできる限り、フルパス名で指定します。

単純ファイル名で指定したときには、カレントディレクトリ上のファイルと見なします。

また、現在編集中のファイル名を指定したときに、そのファイルに保存していない内容がある場合は、複写する前に自動的にそのファイルを保存します。

val には以下のマクロ定数が指定できます。

複数のマクロ定数を指定するときにはビット論理和(|)で指定します。マクロ定数を指定しないときには 0 を指定します。

COPY_CHECKTIME

複写先に同じファイルがあり、タイムスタンプが複写元のタイムスタンプと同じか、または新しい場合には複写しません。

COPY_CHECKATTR

複写元のファイルに保存属性(アーカイブ属性)がない場合には複写しません。

COPY_READONLY

複写先に同じファイルがあり、そのファイルが読み取り専用属性の場合も複写します。このマクロ定数を指定しないと、複写先が読み取り専用属性の場合は、複写しないでエラーになります。

関数値

- 0: 正常に複写した
- 1: 複写の必要なし
- 2: 複写元の指定が不当
- 3: 複写元ファイルが存在しない
- 4: 複写先が現在編集中のファイル
- 5: 複写先の指定が不当
- 6: 複写元と複写先が同じファイル
- 7: 複写元が読み出せない
- 8: 複写先に書き込めない

- 9: 複写先が読み取り専用属性のファイル
- 10: 複写元の保存中にエラー発生
- 11: その他のエラー

例

```
copyfile("c:¥temp¥¥", "a:¥miw¥miw.txt", COPY_READONLY)
```

aドライブのmiwディレクトリの中のmiw.txtファイルを、cドライブのtempディレクトリに複写します。



半角文字¥で終わる文字列を文字列定数として指定する場合には、文字列の最後の¥だけは2つ続けて ¥¥ と指定しなければいけません。

fclose(val)

ファイルのクローズ

指定したファイルハンドル val をクローズします。

このファイルハンドルは、システム関数 fopen または creat の関数値として返されたものです。

fcreat(ptr)

ファイルの書き込みオープン

指定ファイル ptr を書き込み用にオープンします。

ディスク上にすでに存在するファイルを指定した場合、そのファイルを削除後、新しいファイルとして、再度オープンします。特殊属性のファイルまたは現在編集中のファイルは、書き込み用としてオープンできません。

同時にオープンできるファイル数は、fcreat 関数と fopen 関数を合わせて最大4ファイルです。ディスク上に存在する同名のファイルを削除しないためには、まず先にディスク上にそのファイルが存在しないことを確認します。この確認にはシステム関数 fnamchk を使います(関数値が104であること)。確認後、このfcreat関数でファイルをオープンしてください。

関数値

オープンしたファイルをアクセスするためのファイルハンドル(0以上の値)を返します。

エラーが発生したときには以下の負の値を返します。

- 1: その他のエラー
- 2: 編集中のファイルのとき
- 3: 不当なファイル名のとき



この関数の関数値のファイルハンドルを使うと、fwrite 関数によりテキストファイルに文字列をシーケンシャルに書き込みできます。

findfile(ptr)

ファイル名リストの作成

指定されたファイル名に一致するファイル名リストを作成します。

このファイル名リストからファイル名を1つずつ取り出すのに getfile() 関数を使用します。

引数

ptr には探し出したいファイル名を指定します。ディレクトリ名やワイルドカードを指定することもできます。複数のワイルドカードを指定する場合には、半角のセミコロン ; で区切って指定します。

例1. @5 = findfile("*.TXT")

例2. @5 = findfile("C:¥SAMPLE¥*.TXT")

例3. @5 = findfile("C:¥SAMPLE¥*.TXT;*.DAT")

関数値

関数値には、作成したファイル名リスト中のファイル数が返されます。ファイル名が1つも見つからない場合には0が返されます。

例

```
@2 = findfile("*.LOG")
```

; カレントディレクトリ上の拡張子が".LOG"のファイル名リストを得る

```
@1 = 0
```

```
while @1 < @2
```

```
    getfile(@1,@str1) ;ファイル名リストからファイル名を1つ得る
```

```
    open(@str1,"auto",OPEN_TEXT) ;ファイルを開く
```

```
    insstr("0,0,0,0¥t先頭データ¥n") ;ファイルの先頭に1行挿入
```

```
    if save() < 0 then break ;ファイルを保存
```

```
    close() ;ファイルを閉じる
```

```
    @1++ ;ファイル名リスト中の次のファイルへ
```

```
wend
```



ファイル名リストには上限があります。記録可能なファイル数の上限は2048ファイルで、ファイル名を記録するバッファサイズの上限は32Kバイト(15バイトのファイル名で2048ファイル分)です。

また、ファイル名リストは同時には1つしか作成できません。すなわち、新たなファイル名リストを作成すると、以前のファイル名リストは自動的になくなります。さらに、ディレクトリ名は検索できません。複数のディレクトリにまたがったファイル名のリストを作成することもできません。

なお、ファイル名リストへのファイル名の格納は、ファイル名を見つけた順に格納されます。これは「ファイルを開く」ダイアログボックス中の「出現順」と同じです。

fnamchk(ptr)

ファイルのチェック

指定したファイルをチェックします。

関数値

- 0 ~ 99 すでに開かれている
- 101 存在しないディレクトリ名
- 102 不当なファイル名
- 103 拡張子が.BAKまたは.BK?の新しいファイル
- 104 新しいファイル
- 105 既存のファイル
- 106 既存の読み取り専用属性のファイル

fopen(ptr)

ファイルの読み込みオープン

指定ファイルptrを読み込み用にオープンします。

ディスク上に存在しないファイルや現在編集中のファイルはオープンできません。
同時にオープンできるファイル数は、fcreat関数とfopen関数を合わせて最大4ファイルです。

関数値

- オープンしたファイルにアクセスするためのファイルハンドル(0以上の値)を返します。
エラーが発生したときには以下の負の値を返します。
- 1: その他のエラー
 - 2: 編集中的ファイルのとき
 - 3: ファイルが存在しないか、不当なファイル名のとき



この関数の関数値のファイルハンドルを使うと、fread関数によりテキストファイルから文字列をシーケンシャルに読み込みます。

fread(val,vptr)

ファイルから1行読み込み

指定のファイルハンドルから1行分の文字列を配列変数に読み込みます。

行末のCRコード(0DH)とLFコード(0AH)は、文字列の終端マークであるヌルコード(00H)に変換して代入します。

引数

- val fopen関数によるファイルハンドル
- vptr 読み込んだ文字列を格納する配列変数

このファイルハンドルは、システム関数fopenの関数値に返されたものです。

関数値

読み込んだ文字列の文字数を返します。バイト数ではありません。

また、以下の値も返します。

- 0: 改行文字だけの行のとき
- 1: 読み込みエラー
- 2: ファイルの終わりに達したとき



1 行の文字列が長く、指定した配列変数内に読み切れない場合には、システム変数 @nextbyte に 0 より大きな数を代入します。1 行を最後まで読み込んだ場合には @nextbyte に 0 を代入します。1 行が 1024 文字以上の場合には 2 回以上に分けて読み込みます。

1 行の文字数が多い場合で関数値に 0 以上の値を返したときには、システム変数 @nextbyte で、最後まで読み込んだかを確認してください。また、読み込み終了後には、fclose 関数を実行してください。

fwrite(val, ptr)

ファイルへ 1 行書き込み

指定の文字列 ptr を指定のファイルハンドル val に書き込みます。

指定した文字列と、その文字列の後に CR コード (0DH) と LF コード (0AH) を自動的に書き込みます。この CR コードと LF コードも関数値のバイト数に含まれます。

引数

val	fcreat 関数によるファイルハンドル
ptr	書き込む文字列

このファイルハンドルは、システム関数 fcreat の関数値に返されたものです。

関数値

書き込んだ文字列のバイト数を返します。文字数ではありません。

通常のシステム関数は文字数を返すのに対し、この関数はバイト数を返します。注意してください。

正常に書き込みが終了したときは 2 以上の値を返します。エラーが発生したときには負の数を返します。



1 行が 1024 文字以上の長い行は書き出せません。また、書き込み終了後には、fclose 関数を実行してください。

getfile(val,vptr)

ファイル名リストからファイル名を得る

findfile() 関数で作成したファイル名リスト中からファイル名を1つ取り出します。

引数

- val ファイル名リスト中のval番目のファイル名を取り出します。この値には0からfindfile()関数で返された関数値 - 1までの値が指定できます。
- vptr ファイル名を取り出す配列変数位置を指定します。ファイル名は常にフルパス名で返されます。

関数値

得られたファイル名(フルパス名)の文字数が関数値に返されます。

例

```
@2 = findfile("*.LOG")
;カレントディレクトリ上の拡張子が".LOG"のファイル名リストを得る

@1 = 0
while @1 < @2
  getfile(@1,@str1) ;ファイル名リストからファイル名を1つ取り出す
  open(@str1,"auto",OPEN_TEXT) ;取り出したファイルを開く
  insstr("0,0,0,0%t先頭データ%t") ;ファイルの先頭に1行挿入
  if save() < 0 then break ;ファイルを保存する
  close() ;ファイルを閉じる
  @1++ ;ファイル名リスト中の次のファイルへ
wend
```

getftime(ptr,vptr)

ファイルの日時を得る

指定されたファイルのタイムスタンプ情報を得ます。

引数

- ptr タイムスタンプ情報を得たいファイル名
- vptr タイムスタンプ情報を得る配列変数内の位置
- この配列要素位置を[0]として、以下の配列要素に情報が返されます。
- [0] 最終書き込み：西暦年(1601～)
 - [1] 最終書き込み：月(1～12)
 - [2] 最終書き込み：週(0:日曜, 1:月曜,,, 6:土曜)
 - [3] 最終書き込み：日(1～31)
 - [4] 最終書き込み：時(0～23)
 - [5] 最終書き込み：分(0～59)
 - [6] 最終書き込み：秒(0～59)
 - [7] 最終書き込み：ミリ秒(0～999)
 - [8] ファイル作成：西暦年(1601～)

- [9] ファイル作成 : 月(1 ~ 12)
- [10] ファイル作成 : 週(0:日曜 , 1:月曜 , , , 6:土曜)
- [11] ファイル作成 : 日(1 ~ 31)
- [12] ファイル作成 : 時(0 ~ 23)
- [13] ファイル作成 : 分(0 ~ 59)
- [14] ファイル作成 : 秒(0 ~ 59)
- [15] ファイル作成 : ミリ秒(0 ~ 999)
- [16] 最終アクセス : 西暦年(1601 ~)
- [17] 最終アクセス : 月(1 ~ 12)
- [18] 最終アクセス : 週(0:日曜 , 1:月曜 , , , 6:土曜)
- [19] 最終アクセス : 日(1 ~ 31)
- [20] 最終アクセス : 時(0 ~ 23)
- [21] 最終アクセス : 分(0 ~ 59)
- [22] 最終アクセス : 秒(0 ~ 59)
- [23] 最終アクセス : ミリ秒(0 ~ 999)
- [24] ファイルサイズの最下位 16 ビット
- [25] ファイルサイズの中下位 16 ビット
- [26] ファイルサイズの中上位 16 ビット(通常は 0)
- [27] ファイルサイズの最上位 16 ビット(通常は 0)
- [28] ファイル属性の下位 16 ビット
- [29] ファイル属性の上位 16 ビット

関数値

関数値には、正常にファイルの情報が得られれば30が返され、ファイルが見つからないなどで情報が得られなかった場合には0が返される。

例

```
@str2[0] = '日'
@str2[1] = '月'
@str2[2] = '火'
@str2[3] = '水'
@str2[4] = '木'
@str2[5] = '金'
@str2[6] = '土'

if getftime("sample.txt",@str1) == 30
    printf("sample.txt:%4d年%2d月%2d日(%c曜日)%n",
        @str1[0],@str1[1],@str1[3],@str2[@str1[2]]);
endif
```

insfile(ptr)

外部ファイルの挿入

指定したファイル ptr をカーソル位置(カーソル行の上)に挿入します。

関数値

ファイルが挿入できれば 1 を返します。挿入できないときには 0 を返します。

例

```
insfile("miw.txt")
```

newopen()

新規ウィンドウを開く

「新規:nn」ファイルを開きます。

関数値

「新規:nn」ウィンドウを開くことができれば 1 を返します。開くことができなければ 0 を返します。

open(ptr1,ptr2,val)

ファイルを開く

指定したファイルを開きます。

指定したファイルがすでに開いている場合でも、カレントウィンドウの切り替えは行いません。

引数

ptr1	開きたいファイル名
ptr2	開く際に使用するプリプロセッサ名
val	オープンフラグ

ファイルを開く時にプリプロセッサを使用しない場合には、ptr2 にヌル文字列("")を指定します。プリプロセッサを自動設定にしたい場合には、ptr2 に文字列"auto"を指定します。プリプロセッサ名を指定する場合は、パス名ではなく単純ファイル名で拡張子(.PPP)もあわせて指定します。



ptr2 がヌル文字列の場合は、ファイル内容による自動コード判定は行われません。また、ptr2 が"auto"の場合で、指定したファイルの拡張子が拡張子設定のいずれの定義にも一致しない場合(デフォルトの定義が適用される場合)は、ファイル内容による自動コード判定が行われません。ただし、「ファイルを開く」ダイアログボックス中の「ファイル内容による自動コード判定を禁止する」にチェックがついている場合は、ファイル内容による自動コード判定は行われません。

オープンフラグは以下のマクロ定数のいずれか、またはそれらのビット論理和(OR)で指定します。

OPEN_READONLY	読み取り専用で開く(他のマクロ定数との論理和で指定)
OPEN_EOFTEXT	「テキスト(^\zまで)」で開く
OPEN_TEXT	「テキスト」で開く
OPEN_BINARY	「バイナリ」で開く
OPEN_AUTO	「自動設定」で開く
OPEN_NOHISTORY	開きたいファイル名を「開いたファイル名の履歴」には記録しない(他のマクロ定数との論理和で指定)

関数値

0 ~ 99	すでに開かれている
101	存在しないディレクトリ名
102	不当なファイル名
103	拡張子が.BAKの新しいファイル
104	新しいファイル
105	既存のファイル
106	既存の読み取り専用属性ファイル
-1	プリプロセッサが見つからない

例

```
open("c:¥temp¥test.txt", "miweuc.ppp", OPEN_TEXT)
```

Cドライブのtempディレクトリのtest.txtファイルを、プリプロセッサmiweuc.pppを使用して開きます。通常テキストモードで開きます。

original()

編集のやり直し

現在編集中のウィンドウ(カレントウィンドウ)を前回保存した状態に戻します。

関数値

正常に終了すれば1を返します。
カレントウィンドウが「新規:nn」ウィンドウのときは0を返します。
ディスク関係のエラーが発生したときには負の数を返します。

outprinter(val1, val2, val3, val4, val5, ptr1, ptr2, ptr3)

印刷の実行

指定した印刷設定で現在のプリンタにカレントウィンドウの内容を印刷します。

引数

val1

特に明記されていない場合、()内は各ビットが1のときの状態です。

ビット0 ~ 7	印刷フォントのポイント数 0: デバイス依存フォント、1: 行桁数で指定(ptr3も指定)
ビット8 ~ 9	フッタの位置(0: なし、1: 左端、2: 中央、3: 右端)

ビット10	空き(常に0)
ビット11	ヘッダ/フッタの印刷位置(0:余白の上下端、1:余白の中央)
ビット12 ~ 13	ヘッダの位置(0:なし、1:左端、2:中央、3:右端)
ビット14	0(ゼロ)とO(オー)を区別した印刷(区別する)
ビット15	キーワードの明示(明示する)
ビット16	罫線接続処理(行う)
ビット17 ~ 18	行の間隔(0:通常、1:1.5倍、2:2倍、3:3倍)
ビット19	インテリジェント改頁(行う)
ビット20	行番号(付加する)
ビット21 ~ 23	用紙1枚に印刷するページ数 (0:1ページ、1:4ページ、2:横2ページ、3:縦2ページ、 4:9ページ、5:16ページ、6:25ページ)
ビット24	禁則処理(行う)
ビット25 ~ 26	段組印刷(0:1段組、1:2段組、2:3段組)
ビット27	画面上の折返し位置での改行(行う)
ビット28	英文ワードラップ(行う)
ビット29 ~ 30	段と段の間(0:空白、1:単線、2:二重線)
ビット31	印刷禁止フラグ(0:印刷実行、1:パラメータ設定のみ)
val2	左側余白(mm)
val3	右側余白(mm)
val4	上側余白(mm)
val5	下側余白(mm)
ptr1	ヘッダ文字列
ptr2	フッタ文字列
ptr3	印刷フォントの書体名 フォントのポイント数の指定(val1)が1の場合には、ptr3の文字列が、カンマで区切った2つの半角10進数により、行数と桁数を指定しているものと見なします。 (例."60,84")

関数値

0:	印刷しなかったとき
1:	印刷したとき
負の数:	エラーが発生したとき

例

```
outprinter(0x0101b300,15,15,15,15,"%F","%p", "")
```

フォント	: デバイス依存フォント
フッタおよびヘッダ位置	: 右端
罫線接続処理	: 行う
行番号	: 付加しない
禁則処理	: 行う
段組み	: 1段組

英文ワードラップ	: 行わない
イテリジェント改ページ	: 行わない
キーワードの明示	: 行う
用紙 1 枚に 1 ページを印刷	
左～下側余白	: 15mm
ヘッダ文字列	: ファイル名
フッタ文字列	: ページ番号
印刷フォント	: なし(デバイス依存を指定しているため)

rename(ptr)

ファイル名の変更

現在編集中のウィンドウ(カレントウィンドウ)のファイル名を指定したファイル名 ptr に変更します。

カレントウィンドウが読み取り専用で開かれているときは、ファイル名は変更できません。また、変更するために指定したファイル名のファイルがすでに開かれているときも、ファイル名は変更できません。

ファイル名に単純ファイル名で指定したときには、ファイル名の部分だけを変更します。ドライブやディレクトリは変更しません。

一方、フルパス名で指定すると、ドライブやディレクトリまで変更できます。

関数値

ファイル名を変更したときには 1 を返します。変更できなかったときには 0 を返します。

例

```
rename(" a:%test%sample.txt ")
```

save()

ファイルへの保存

現在編集中のウィンドウ(カレントウィンドウ)をファイルへ保存します。

関数値

- 0: 保存が必要ないとき
- 1: 正常に保存できたとき
- 負の数: 保存中にエラーが発生したとき

保存が必要ないときは、カレントウィンドウが以下の場合をさします。

- ・読み取り専用で開いているとき
- ・「新規:nn」ウィンドウの場合
- ・ファイルの内容を変更していない場合
- ・拡張子が.BAKまたは.BK?のファイルのウィンドウの場合
- ・「グローバル検索結果」ウィンドウの場合

- ・「一括ファイル比較結果」ウィンドウの場合
- ・「DOSシェルエスケープ」ウィンドウの場合
- ・読み取り専用属性ファイルのウィンドウの場合

saveas(ptr)

名前を付けて保存

現在編集中のウィンドウ(カレントウィンドウ)の内容を、指定したファイル名のファイルに保存します。

関数値

- 0: 指定したファイル名が不当なとき
- 1: 保存できたとき
- 負の数: ディスクエラーが発生したとき

「新規:nn」ウィンドウを【ファイル(F)】-【名前を付けて保存(A)】を選択して指定ファイルに保存した場合、ウィンドウのタイトルバーに表示されているファイル名は指定したファイル名に変更されますが、saveas()関数で保存した場合には、「新規:nn」ウィンドウのタイトルバーは常に「新規:nn」のままです。

setftime(ptr,vptr)

ファイルの日時を設定

指定されたファイルのタイムスタンプ情報を変更します。

引数

- ptr タイムスタンプ情報を変更したいファイル名
 - vptr 変更したいタイムスタンプ情報を指定する配列変数内の位置
- この配列要素位置を[0]として以下の配列要素に変更したい情報を指定すること。
- [0] 最終書き込み：西暦年(1601～)
 - [1] 最終書き込み：月(1～12)
 - [2] 最終書き込み：週(0:日曜、1:月曜、、、6:土曜)
曜日の指定はダミーです。通常0を指定しておいてください。
 - [3] 最終書き込み：日(1～31)
 - [4] 最終書き込み：時(0～23)
 - [5] 最終書き込み：分(0～59)
 - [6] 最終書き込み：秒(0～59)
 - [7] 最終書き込み：ミリ秒(0～999)

関数値

関数値には、正常にタイムスタンプ情報が変更できれば1が、ファイルが見つからないなどで変更できなかった場合には0が返される。

例

```
@str1[0]=2001
@str1[1]=11
@str1[2]=0 ;ダミー
@str1[3]=23
@str1[4]=0
@str1[5]=0
@str1[6]=0
@str1[7]=0
setftime("sample.txt",@str1) ;2001年11月23日0時0分0秒に設定
```

setprinter(ptr1,ptr2,ptr3)

プリンタの設定

現在のプリンタを指定したプリンタに設定します。

現在のプリンタとは、「通常使うプリンタ」として設定しているプリンタをさします。

引数

ptr1	デバイス名
ptr2	ドライバ名
ptr3	ポート名

関数値

正常に変更できれば 1 を返します。エラーが発生したときには 0 を返します。

例

```
setprinter("NEC PC-PR1000/4", "NPDL2", "LPT1:")
```

unlink(ptr)

ファイルの削除

指定したファイル名 ptr のファイルをディスク上から削除します。

ファイル名はできる限りフルパス名で指定します。編集中のファイルや特殊属性のファイルは削除できません。

関数値

正常にファイルが削除されれば 0 を返します。

例

```
unlink("c:¥temp¥test.txt")
```

6. その他の関数

atoi(ptr)

文字列を数値に変換

10進文字列、または16進文字列を数値に変換します。

10進文字列の前には負の符号 - を付けることができます。

16進文字列は \$ または 0x で始まる文字列を指定します。

関数値

変換された数値を返します。

例

```
@2=atoi("0x0a")
```

このとき、@2に10を返します。



メモ

数値に変換するということは、文字コードの列をバイナリ値に変換するということです。

beep(val)

ビーブ音を鳴らす

ビーブ音を鳴らします。

valにはビーブ音の以下のID値を指定します。

-1以外の値を指定したときは、レジストリの [sounds] セクションに設定しているサウンドを再生します。

-1	標準ビーブ音を再生
0	SystemDefaultで指定したもの
0x10	SystemHandで指定したもの
0x20	SystemQuestionで指定したもの
0x30	SystemExclamationで指定したもの
0x40	SystemAsteriskで指定したもの

この関数はWindowsのAPIのMessageBeep()と同じ動作をします。

bp(ptr)

マクロ実行のブレーク

マクロコマンドの実行を中断してブレーク状態にし、ptrで指定した文字列を多目的バーに表示します。

マクロモードがシングルステップ実行に設定されているときは、bp関数は無効です。



関数値

マクロモードがシングルステップ実行になっているときには 0 を返します。

例

bp(“ たいまブレーク中です”)

ブレーク状態からマクロコマンドの実行を再開するには、【マクロ(M)】-[ブレークマクロ再開(B)] を選択します。

ブレーク状態のマクロコマンドを中止するには、【マクロ(M)】-[ブレークマクロ中止(S)] を選択するか、[Pause] キーまたは [STOP] キーを押します。

callDll(ptr1,ptr2)

外部DLLの呼び出し

指定したダイナミックリンクライブラリ ptr1 の中の指定した関数 ptr2 を呼び出します。

ダイナミックリンクライブラリ名 ptr1 には単純ファイル名と拡張子 (.DLL) を指定します。呼び出すダイナミックリンクライブラリは、MIFES 用の機能拡張モジュールを必ず指定します。また、このファイルはロードディレクトリ(通常は MIW.EXE と同じディレクトリ)に置いておきます。

この関数は、MIFES 本体と機能拡張モジュールとの間の橋渡しをするためのものです。この関数を使うと、本体のプログラム(MIW.EXE)はそのままでも、機能拡張用のダイナミックリンクライブラリとそれを呼び出すマクロコマンドを追加するだけで、エディタの機能拡張ができます。

この callDll 関数で呼び出す関数は、以下の引数を伴って呼び出されます。

```
LONG FAR PASCAL ptr2(hinst,frame,mdi,child,gvar,gary,lvar,lary)
INSTANCE hinst;      /* インスタンスハンドル */
HWND frame;         /* フレームウィンドウのウィンドウハンドル */
HWND mdi;           /* MDI ウィンドウのウィンドウハンドル */
HWND child;         /* カレントの MDI 子ウィンドウのウィンドウハンドル */
/* MDI 子ウィンドウがないときは 0 */
LONG *gvar;          /* マクロ変数 @1,,,@16 が連続した領域 */
/* のアドレス。すなわち (LONG FAR *)&@1 */
WORD *gary;          /* マクロ配列変数 @str1[1024],,,@str8[1024] */
/* が連続した領域のアドレス */
/* すなわち (WORD FAR *)&@str1[0] */
LONG *lvar;          /* マクロ変数 @@1,,,@@16 が連続した領域 */
/* のアドレス。すなわち (LONG FAR *)&@@1 */
WORD *lary;          /* マクロ配列変数 @@str1[1024],,,@@str8[1024] */
/* が連続した領域のアドレス */
/* すなわち (WORD FAR *)&@@str1[0] */
```

関数値

呼び出した関数を実行すれば 1 を返します。実行しなければ 0 を返します。

呼び出した関数の直接の関数値は、システム変数 @retcode に代入します。

説明

この callDll 関数で呼び出されるダイナミックリンクライブラリは、MIFES の起動後に最初に呼び出されたときにロード (LoadLibrary) され、MIFES の終了時に解放 (FreeLibrary) されます。

単純な引数を持つ関数ならば MIFES 専用でない DLL でも呼び出せるように、システム関数 callDll0() ~ callDll5() が用意されています。これらのシステム関数では、ロードディレクトリ上にない DLL 中の関数も呼び出せます。(システムが探し出せるディレクトリ、つまり、Windows ディレクトリ、Windows システムディレクトリ、PATH の通ったディレクトリ、呼び出すタスクのカレントディレクトリ、呼び出すタスクの本体モジュールのあるディレクトリにあれば呼び出せます)

これらのシステム関数の使用に当たっては、**すべてユーザーの責任において行ってください。そして、十分に注意して行ってください。**

```
callDll0(ptr1,ptr2)
callDll1(ptr1,ptr2,arg1)
callDll2(ptr1,ptr2,arg1,arg2)
callDll3(ptr1,ptr2,arg1,arg2,arg3)
callDll4(ptr1,ptr2,arg1,arg2,arg3,arg4)
callDll5(ptr1,ptr2,arg1,arg2,arg3,arg4,arg5)
```

ptr1	DLL の名前
ptr2	エクスポートされた関数の名前
arg1 ~ arg5	関数への引数(val または ptr)

ptr にグローバル配列変数内の位置を指定した場合、その位置にある文字列を関数に渡すか、または、関数から文字列をその位置に受け取るものとみなします。このため、文字列データのマクロ言語フォーマットと通常シフト JIS フォーマット (BYTE*) との相互変換を自動的に行ないません。なお、UNICODE 関数 (...W) は呼び出せません。

一方、ptr にローカル配列変数内の位置を指定した場合には、その位置にある(またはその位置で受け取る)データを文字列と仮定せずに、単にその位置をポインタとして関数に渡します。文字列以外のポインタを引数にしたい場合に使用します。

arg1 が関数の第 1 引数に、arg5 が関数の第 5 引数になります。引数のタイプ (val と ptr) を間違えて指定すると、関数が暴走したり保護エラーを起こしたりするので、arg1 ~ arg5 の指定には十分注意してください。

関数値

関数が実行できれば 1 を、DLL や関数が見つからないなどで実行できなかった場合には 0 を返します。

@retcode 実行した関数が返した関数値

32ビット整数値を返すような関数、または関数値を返さないような関数だけが実行できることを意味します。
指定した部分を指定した色に変更します。

chgcolor(val1,val2)

カラーの変更

引数

val1 変更する部分を指定する

val2 色を COLORREF 値で指定する

val1 の値は以下の部分を意味します。

- 0 : 通常ウィンドウ背景色
- 1 : 通常文字色
- 2 : 改行文字色
- 3 : アンダーライン色
- 4 : ガイドライン背景色
- 5 : 特殊文字表示色
- 6 : 変更行通常文字
- 7 : 通常文字色 : 変更行
- 8 : ガイドライン文字色
- 9 : ユーザー定義バー上ボタンの背景色
- 10 : キーワード 1 の文字
- 11 : コメント色
- 12 : キーワード 1 の色 : 変更行
- 13 : コメント色 : 変更行
- 14 : #ifdef ブロック色
- 15 : #ifdef ブロック色 : 変更行
- 16 : 行ゲージ背景色
- 17 : 行ゲージ文字色
- 18 : 桁ゲージ上各種マーク色
- 19 : 多目的バー背景色
- 20 : 多目的バー文字色
- 21 : ユーザー定義バー上ボタンの文字色
- 22 : ツールバー/多目的バー背景色
- 23 : [EOF] マーク文字色
- 24 : 読専ウィンドウ背景色
- 25 : 行・桁ゲージ境界線色
- 26 : キーワード 2 の色
- 27 : キーワード 2 の色 : 変更行
- 28 : キーワード 3 の色
- 29 : キーワード 3 の色 : 変更行
- 30 : キーワード 4 の色
- 31 : キーワード 4 の色 : 変更行
- 32 : 行マーク表示背景色
- 33 : 背景罫線色
- 34 : ガイドライン上の枠の色
- 35 : ガイドライン上枠内背景色
- 36 : ガイドライン上ボタン背景色
- 37 : ガイドライン上ボタン文字色

- 38 : ツールバー上ボックス枠の色
- 39 : ツールバー上ボックス内背景色
- 40 : ツールバー上ボックス内文字色
- 41 : ツールバー上セパレータ線左色
- 42 : ツールバー上セパレータ線右色
- 43 : 桁ゲージ背景色
- 44 : 桁ゲージ文字色
- 45 : [EOF]マーク文字色
- 46 : リストウィンドウ上部文字色
- 47 : 各種リスト中の境界線色
- 48 : 多目的バー可変タブ線色
- 49 : 多目的バーショートカット文字色
- 50 : 多目的バー変更マーク背景色
- 51 : ホームページアドレス明示色
- 52 : メールアドレス明示色
- 53 : HTML タグ色
- 54 : HTML タグ色 : 変更行
- 55 : HTML コメントタグ色
- 56 : HTML コメントタグ色 : 変更行
- 57 : 文字列定数色
- 58 : 文字列定数色 : 変更行
- 59 : リストウィンドウ背景色
- 60 : リストウィンドウ文字色
- 61 : リストウィンドウ選択背景色
- 62 : リストウィンドウ選択文字色
- 63 : リストウィンドウ・ボタン背景色
- 64 : リストウィンドウ・ボタン文字色

val2 に指定する COLORREF 値は32ビットの整数値です。最下位8ビットが赤の輝度を、その上の8ビットが緑の輝度を表します。さらに、その上の8ビットが青の輝度を表します。最上位の8ビットは常に0です。なお、-1(0xffffffff)を指定したときには、指定部分の色は変更しません。

関数値

指定した部分の変更する前の色の COLORREF 値を返します。

val2 で -1 を指定したときは、その部分の色を関数値に返すだけです。

例

- ・ `chgcOLOR(2,0x000000ff)` ; 改行文字を赤色
- ・ `chgcOLOR(4,0x0000ff00)` ; ガイドラインを緑色
- ・ `chgcOLOR(5,0x00ff0000)` ; メモマークやタブなどを青色
- ・ `chgcOLOR(6,0x00ffff00)` ; 変更行の文字を水色
- ・ `chgcOLOR(8,0x00ffffff)` ; ガイドライン上の文字を白色
- ・ `chgcOLOR(9,0x00000000)` ; ユーザ定義バーの背景色を黒色

変更した色をすぐに画面に反映させるには、システム変数 @disp に 3 を代入しウィンドウを再描画してください。



メモ

chgcursor(val)

マウスカーソルの形状変更

マウスカーソルの形状を変更します。

シングルステップ実行中のときには、chgcursor 関数は無効になります。

また以下の処理などをすると、自動的にマウスカーソルが通常の矢印に戻りますので注意してください。

- ・ waitevent 関数を実行したとき
- ・ ダイアログボックスを表示したとき
- ・ マクロコマンドの実行が終了したとき

引数

val の値は以下のとおりです。

- 0: 砂時計カーソル
- 1: 矢印カーソル



val に次のマクロ定数を指定しても形状を変更できます。

- CURSOR_WAIT 砂時計カーソル
- CURSOR_NORMAL 矢印カーソル

chgfont(val1,val2,ptr)

表示フォントの変更

表示フォントを指定のフォントに変更します。

引数

- val1 変更フラグ
- val2 フォントのポイントサイズと最小の行間サイズを指定
- ptr フォントの書体名を指定

val1 に以下のいずれかのマクロ定数を指定します。

- FONT_WIN カレントウィンドウのフォントを変更
- FONT_SYS 以降に開くウィンドウのフォントを変更
- FONT_BOTH 両方のフォントを変更

val2

- ビット0 ~ 7 フォントのポイントサイズ(0,4 ~ 72)
0を指定すると標準フォントと見なされます。
- ビット8 ~ 15 最小の行間サイズをピクセル単位で指定(0 ~ 20)
0を指定すると標準行間(1ピクセル)と見なされます。
フォントの外部レディングが指定の最小行間より大きい場合は、
行間のサイズはフォントの外部レディングのサイズになります。
- ビット16 ~ 31 未使用(0を指定)

ptr フォントの書体名を指定
val2 でフォントのポイントサイズ(ビット0~7)に0を指定した場合、書体名の指定 ptr は無視されます。
書体名を変えずにポイントサイズや行間サイズだけを変更したい場合には ptr にヌル文字列を指定してください。

関数値

正常にフォントが変更できたときには 0 を返します。
指定したフォントがないなどの理由で変更できなかったときには 0 以外の値を返します。

フォントが変更できなかった場合のフォントは、この関数を実行する前のフォントのままです。

例

```
chgfont(FONT_WIN,10,"")  
;カレントウィンドウのフォントサイズだけを10ポイントに変更  
(フォントと行間は変更しない)
```

chghelp(val,ptr)

ヘルプファイルの変更

拡張ヘルプファイル、またはイージーヘルプ辞書を変更します。

引数

val ヘルプファイルの指定(1: 拡張ヘルプファイル、2: イージーヘルプ辞書)
ptr ヘルプファイル名

拡張ヘルプファイル(val=1)を変更するときは、ptr にはフルパス名を指定します。
イージーヘルプ辞書名(val=2)を変更するときは、ptr に単純ファイル名を指定します。

関数値

正常にファイル設定が変更できたときは 0 を返します。
エラーがあったときには 0 以外の値を返します。

child(val,ptr1,ptr2)

子プロセスの実行

子プロセスを実行します。

引数

val 0: 直接実行
 1: COMMAND.COM を介して実行
 2: DOS シェルエスケープ上で実行
ptr1 実行コマンド文字列
ptr2 実行時ディレクトリ(通常ヌル文字列でよい)



関数値

正常に子プロセスを実行できれば 1 を返します。エラーがあれば 0 を返します。

子プロセスが終了する前に、この child 関数が終了する場合があります。これは Windows が
プリエンティブマルチタスク方式のためです。

また、DOS シェルエスケープ上で子プロセスを実行したときに、関数値が 1 の場合でも
子プロセスが正常に実行されていない場合があります。これは、指定した子プロセスが
command.com を介して実行する場合、command.com が正常に実行した時点で関数値に
1 が返り、command.com の後の実行でエラーになっても関数値が 1 のままであるため、子
プロセスが正常に実行できたかどうか分からないことによります。

例

```
child(0, "calc.exe", ""); ;Windows アクセサリの電卓を起動
```

clsmess()

メッセージの消去

システム関数 message で表示したメッセージを消去します。

compile(val)

マクロコマンドのコンパイル

カレントウィンドウ上の MIL/W 言語で記述したソースプログラムをコンパイルします。

引数

val には以下のマクロ定数のいずれか、またはそれらのビット論理和(|)を指定します。

COMPILE_COMMAND	カーソル位置から 1 コマンド分をコンパイル
COMPILE_FILE	カレントウィンドウ全体をコンパイル
COMPILE_LIBRARY	コンパイル結果をライブラリに自動格納
COMPILE_SOURCE	中間コード中にソースコードを埋め込む
COMPILE_NOMIWMAC	MIW.MAC の自動コンパイルを「なし」に設定

COMPILE_COMMAND、または COMPILE_FILE のどちらかは必ず指定してください。そ
の他のマクロ定数は必要に応じて指定してください。

この関数の実行後に、強制的にマクロコマンドの実行を終了します。そのため、関数値はあ
りません。この関数は必ずマクロコマンドの最後に記述してください。この関数以降に記述
した処理は実行しません。

この関数を実行すると、カレントマクロコマンドが自動的に置き代わります。

カレントウィンドウ全体をコンパイル指定した場合(COMPILE_FILE)、コンパイルエラー
が発生した時点で、コンパイル処理も中止されます。

COMPILE_LIBRARY を指定した場合、コンパイル結果の中間コード(マクロコマンド)を

ライブラリ(MIW.LIB)に格納します。

このときライブラリ(MIW.LIB)の中に同じマクロコマンド名が格納されていると、自動的に上書きされます。確認のメッセージは表示されません。

COMPILE_NOMIWMACの指定は、コンパイラが正常に終了したときにだけ有効です。コンパイルエラーが発生した場合には、この指定は無視されます。

ctype(val)

文字の種類を取得

指定した文字コードの文字の種類を返します。

関数値

- 0: 半角非デリミタ
- 1: 半角デリミタ
- 2: 全角非デリミタ
- 3: 全角デリミタ
- 9: その他の文字
- 13: 全角罫線

奇数の場合はデリミタ文字です。10以上の場合は罫線文字です。

end(val)

エディタの終了

MIFESを終了します。

引数

- 0: 変更のあるファイルが開かれている場合も保存せずに終了する
- 1: 変更のあるすべてのファイルを保存した後に終了する
- 2: MIFESの次のトップレベルウィンドウをアクティブにする

2を指定すると、MIFESは非アクティブになりますが、終了はしません。次のトップレベルウィンドウとは、MIFESを終了したときに次にアクティブになるウィンドウのことです。つまり、2を指定した場合は、ユーザーがマウスで別のアプリケーションのウィンドウをクリックするのと同じ意味になります。

MIFESを終了せずに別のアプリケーションにユーザーの操作を移せます。

execmd(val) 機能番号の実行

指定した機能番号を実行します。

引数

val で指定できる機能番号は 1 ~ 249 です。

関数値

実行されれば 1 を返します。

引数の指定を誤ったときには 0 を返します。



機能番号の一覧はヘルプを参照してください。

exeurl(pstr) 指定した URL をブラウザで表示

指定された文字列を URL と見なし、ブラウザを起動して URL を表示します。

URL 文字列の作成に関しては、システム関数 `sprintf()` などを使用して、マクロ言語の中でプログラムすることになります。

呼び出すブラウザはレジストリの定義に依存します。

具体的には、レジストリ中で拡張子 `.htm` に関連付けされた `open` コマンドを実行することにより、ブラウザを呼び出します。

引数

pstr 表示させたい URL 文字列

関数値

ブラウザが呼び出せた場合には 1 が返されます。

何らかの原因でブラウザが呼び出せなかった場合には 0 が返されます。

例 「インターネット検索」機能 (機能番号 552) は以下のように記述できます。

```
if getselstring (@str1) > 0
    sprintf (@str2, "http://search.yahoo.co.jp/bin/query?p=%e&hc
            =0&hs=0 ", @str1)
    exeurl (@str2)
endif
```

gettime(vptr)

ローカル日時を得る

現在のローカルな日時を得ます。

引数

vp_{tr} 日時の情報を得る配列変数内の位置
この配列要素位置を [0] として、以下の配列要素に情報が返されます。

- [0] 西暦年 (1601 ~)
- [1] 月 (1 ~ 12)
- [2] 週 (0 : 日曜、1 : 月曜...6 : 土曜)
- [3] 日 (1 ~ 31)
- [4] 時 (0 ~ 23)
- [5] 分 (0 ~ 59)
- [6] 秒 (0 ~ 59)
- [7] ミリ秒 (0 ~ 999)

関数値

関数値には常に8が返されます。

getwinpos(vptr)

フレームウィンドウの位置を得る

フレームウィンドウの位置とサイズを得ます。

引数

vp_{tr} フレームウィンドウの位置とサイズの情報を得る配列変数内の位置
この配列要素位置を[0]として、以下の配列要素に情報が得られる。

- [0] フレームウィンドウ左端のX座標 (ピクセル)
- [1] フレームウィンドウ上端のY座標 (ピクセル)
- [2] フレームウィンドウの横幅 (ピクセル)
- [3] フレームウィンドウの高さ (ピクセル)
- [4] スクリーンの横幅 (ピクセル)
- [5] スクリーンの高さ (ピクセル)

関数値

関数値には、常に6が返される。

例

```
;画面の右側にはみ出したウィンドウ位置を調整する
getwinpos(@str1)
if @str1[0]+@str1[2] > @str1[4]
    @str1[0] = @str1[4]-@str1[2]
    if @str1[0] < 0
        @str1[0] = 0
        @str1[2] = @str1[4]
    endif
    setwinpos(@str1)
endif
```

itemlist(ptr1,ptr2,ptr3,val)

リストウィンドウの表示

機能

リストウィンドウを表示し、「C言語関数定義」リスト、「見出し行検索結果」リスト、または「ファイル名検索結果」リストを表示します。

引数

ptr1 検索の対象となるディレクトリ位置

または検索開始ディレクトリ位置

ptr2 検索の対象となるファイル名/ワイルドカード

ptr3 見出し行検索の場合、行頭で探すマーク文字列を指定します。

C言語関数定義の検索の場合、ヌル文字列を指定します。

ファイル名検索の場合、ファイル中で検索する文字列を指定します。検索しない時はヌル文字列を指定します。

見出し行検索のマーク文字列を指定する場合、以下のメタ文字が使用できます。

? 任意の1文字(半角も全角も1文字と見なす)

* 0文字以上の任意の文字列

(1つのマーク中で2つ以上の*は指定しないでください。)

¥? 文字?

¥* 文字*

¥¥ 文字¥

val 各ビットの意味は以下の通りです。ファイル名検索の時にはビット28～31に1を指定してください。

ビット0 1ならば検索対象のファイル中で現在開いているファイルのみを検索

ビット1 1ならばリストの左側に番号を付ける

ビット2 1ならばマーク文字列を項目名に含める

ビット3 未使用(バージョン7から未使用)

ビット4～5 0:C言語関数名を「関数名」の形で表示

1:C言語関数名を「関数名(クラス名)」の形で表示

2:C言語関数名を「クラス名::関数名」の形で表示

ビット6 1ならばクリップボードにも書き出す

ビット7 未使用(0を指定すること)

ビット8～9 0:項目名順にソート表示

1:ファイル名順にソート表示

2:検索順(ソートなし)表示

ビット10～11 未使用(0を指定すること)

ビット12 1ならば下位ディレクトリも検索する

ビット13～19 未使用(0を指定すること)

ビット20～23 ファイル名検索の場合のタイムスタンプ条件

0:タイムスタンプ条件なし

1:タイムスタンプ条件なし

2:最近1時間に更新されたファイル

3:最近3時間に更新されたファイル

4:最近1日間に更新されたファイル

5:最近1週間に更新されたファイル

	6:最近1ヶ月間に更新されたファイル
	7:最近1年間に更新されたファイル
ビット24	ファイル名検索の場合の下位ディレクトリの検索 0:検索しない 1:検索する
ビット25	ファイル名検索の場合の文字列の検索方法 0:半角の英大文字と英小文字は区別する 1:半角の英大文字と英小文字を同一視する
ビット28 ~ 31	検索の動作 0:見出し行検索/C言語関数定義の検索 1:ファイル名検索

関数値

作成したリスト中の項目数を返します。

関数値が負の場合、何らかの原因でリストが作成できなかったか、リストウィンドウが表示できなかったことを示します。

exit()**マクロコマンドの終了**

マクロコマンドの実行を終了します。

マクロの終了を表すため、ソースプログラムの最後に記述します。または、マクロコマンドの実行を中止して終了する場合に使用します。

macro(ptr)**マクロコマンドの実行**

指定したマクロコマンドを実行します。

関数値

マクロコマンドを実行したときには1を返します。

指定したマクロコマンドが見つからないなどで実行できなかったときには0を返します。

指定したマクロコマンドの実行が終了すると、このmacro関数から返ります。この関数で指定して実行するマクロコマンドはネスティング実行しています。並列で実行するわけではありません。

macro関数で実行したマクロコマンドの中から、さらにmacro関数を使って別のマクロコマンドの実行もできます。このようなマクロコマンドのネスティング実行は、実用上8レベル程度まで可能です。

なお、この関数で実行できるマクロコマンドは、コンパイル済みのMIW.MAC中にあるか、マクロライブラリに格納してあるものだけです。

参考

正確には、マクロコマンドのネスティング実行は、実行中のマクロコマンドを退避しておくバッファ（実行中のマクロコマンドの中間コード、ローカル単純変数、ローカル配列変数、実行スタック、GOSUB スタック、などの、各種データを一時的に保存しておくスタック形式のバッファ）に十分な空きがある限り、何レベルでも可能です。1つのマクロコマンドをバッファに退避するには、バッファ中に「約17Kバイト+中間コードサイズ」の空きが必要です。そして、バッファの全体のサイズは約160Kバイトです。ここから計算すると、実行するマクロコマンドの中間コードサイズにもよりますが、実用上は8レベル程度まで、マクロコマンドのネスティング実行が可能と言えます。

例

```
macro("lprint")
```

ライブラリの中のlprintというマクロコマンドを実行します。

message(ptr)

メッセージの表示

指定したメッセージをウィンドウの多目的バーに表示します。

例

```
message(" 範囲を指定してください:ESCキーで終了 ")
```

playsound(val,ptr)

ウェーブフォームの再生

指定したウェーブフォームファイルを非同期で再生します。

非同期での再生とは、再生を開始後すぐに関数から戻ることを意味します。

引数

val 0 または 1

ptr ウェーブフォームファイル名

val に 0 を指定すると、指定したウェーブフォームを 1 回だけ再生します。

1 を指定すると、指定したウェーブフォームを繰り返し再生します。

例：playsound(0, 'tada.wav') ;1 回だけ再生

ptr にヌル文字列を指定すると、現在再生中のサウンドをすべて停止します。

例：playsound(0, "") ;サウンドの停止

関数値

再生できたときには 1 を返します。再生できなかったときには 0 を返します。

ウェーブフォームファイルとは、Windows ディレクトリなどにある拡張子が.wav のファイルです。ウェーブフォームファイルをパス名で指定することもできます。

例：playsound(1, 'a:%windows%msremind.wav') ;繰り返し再生

playsound関数は、現在インストールされているウェブフォームオーディオドライバで再生できるものだけを再生します。

指定したファイルが、インストールされているドライバでは再生できない場合は、再生できません。

readprofile(val,ptr)

カスタマイズファイルの読み込み

カスタマイズファイルの読み込みを実行します。

指定のカスタマイズファイルを読み込み、各種の定義や履歴の更新などを行います。

引数

val	各ビットが1の場合にその情報を読み込みます。()内はカスタマイズファイル内のキーワードです。
ビット0	ツールバー/ユーザー定義バー定義 (RIBBON,BUTTON1,BUTTON2)
ビット1	キー定義 (VK_F,VK_SF,VK_CF,VK_CTRL,VK_EXKEY)
ビット2	メニューバー定義 (MENUW)
ビット3	カラー定義 (COLOR)
ビット4	ワイルドカード履歴 (WILDCARD)
ビット5	拡張子定義 (EXTENSION)
ビット6	明示キーワード定義 (EXKEYWORDS)
ビット7	検索・置換履歴 (HISTORY)
ビット8	ファイル履歴 (PATH)
ビット9	グローバル検索履歴 (GSDIR,GSPARAM)
ビット10	文字列の登録定義 (旧「英字バッファ定義」)(AZBUFFER)
ビット11	複数置換履歴 (REPbatch1 ~ REPbatch10)
ビット12	リストウィンドウ設定 (旧「項目定義リスト履歴」) (LISTWINDOW,ITEMLIST)
ビット13	印刷設定 (PRNfont,PRNheader,PRNFutter,PRNkeisen, PRNKINSOKU,PRNnumber,PRNleft,PRNright,PRNup, PRNdown)
ビット14	動作設定 (A_STATUS,SYSMODE,EXSYSMODE,XXXXMODE,KEISEN, FILEPROTECT,SCROLLSPEED)
ビット15	表示/フォント設定 (D_STATUS,FONT_HEIGHT,MARGIN,HARDTAB,SOFTTAB)
ビット16	ユーザー定義ディレクトリ設定 (HOMEDIR)
ビット17	ポップアップメニュー定義 (MENU0 ~ MENU9)
ビット18	カスタマイズ設定 (PROFILE)
ビット19	その他の設定 「カスタマイズファイルの読み書き」ダイアログボックスにおける 「その他の設定」は、ビット14とビット18とビット19に1を指定 したものの (0x000c4000) に相当します。 (MACRODEF1 ~ MACRODEF16,TOSAFMT,AUTOSAVE, SAVEINTERVAL,FINDID,EASYDIC,SORTID,EXTHELP,

BINPASTE,BINOPERATION,WORDSIZE,ROWWIDTH,
BIGFILE,DIFFPARAM,URLFORMAT,ENVIRONMENT,
MIWOPTION)

ビット20 ~ 30 空き (必ず0を指定すること)

ビット31 このビットが1の場合、指定されたカスタマイズファイルを、以降、起動時/終了時に自動的に読み書きするカスタマイズファイルとします。

ptr 読み込むカスタマイズファイル名を指定します。
確実に MIW ディレクトリ上のファイル名を指定したい場合には、システム関数 `getdir()` を使用してフルパス名にして指定してください。

関数値

関数値には、実際に読み込まれた情報に対応するビットだけが1となった32ビット整数値が返されます。

この数値のビット31が1だった場合、指定したカスタマイズファイルを、以降、起動時/終了時に自動的に読み書きするカスタマイズファイルとした(そのようにレジストリに設定した)ことを意味します。

例

```
readprofile(0x0008,"カラー_背景青.INI") ;色設定を読み込む
```

ribbon(val,ptr)

ツールバー / ユーザー定義バー / 多目的バーの表示

ツールバー / ユーザー定義バー / 多目的バーの表示を変更します。

引数

val 各ビットの意味は以下の通りです。

ビット3 多目的バーの非表示(NOMULTIBAR)

0:表示する 1:表示しない

ビット5 ツールバーの表示(TOOLBAR)

0:表示しない 1:表示する

ビット6 ユーザー定義バー1の表示(USERBAR1)

0:表示しない 1:表示する

ビット7 ユーザー定義バー2の表示(USERBAR2)

0:表示しない 1:表示する

ビット9 ツールバーの下部配置

0:上部に配置 1:下部に配置

ビット10 ユーザー定義バー1の下部配置

0:上部に配置 1:下部に配置

ビット11 ユーザー定義バー2の下部配置

0:上部に配置 1:下部に配置

ビット25 多目的バーの上部配置

0:下部に配置 1:上部に配置

ビット31 多目的バーの2行表示 (DBLMULTIBAR)

0: 1行で表示 1: 2行で表示

なお、旧バージョンに対して変更されたのは、ビット4(ガイドリボンの表示)とビット8(ガイドリボンの下部配置)が無効になり、新たにビット25(多目的バーの上部配置)が有効になりました。

また、以下のような用語変更に伴ないコンパイル用のマクロ定数名も全面的に変更されました。

「ガイドリボン」	なし
「システムリボン」	「ツールバー」
「ユーザー定義リボン1」	「ユーザー定義バー1」
「ユーザー定義リボン2」	「ユーザー定義バー2」
「最下行」	「多目的バー」

ptr ツールバーファイルのファイル名(*.RBN)

ptrには単純ファイル名で拡張子も含めて指定してください。

ヌル文字列 ""を指定すると本体内ツールバーに変更します。

関数値

表示を変更したときには1を、変更しなかったときには0を返します。

例

```
ribbon(0x0034,"MIW7 ロング.RBN") ;多目的バー、ツールバー( MIW7 ロング.RBN)、ユーザー定義バーをすべて上部に表示
```

sendmail(ptr1,ptr2,ptr3,ptr4,ptr5,ptr6,ptr7)

メールの送信

指定されたメールを送信します。

引数

ptr1 宛先名

メールの受け取り人の名前かメールアドレスを指定します。名前指定の場合、この名前はアドレス帳に登録されていなければなりません。

複数の宛先名を一度に指定したい場合には、半角のセミコロン ; で区切って指定してください。

ptr1 がヌル文字列の場合、宛先などを指定するためのダイアログボックスが表示されます。

ptr2 CC名

メールのコピー先の宛名を、受け取り人の名前かメールアドレスで指定します。

名前指定の場合、この名前はアドレス帳に登録されていなければなりません。

複数の宛名を一度に指定したい場合には、半角のセミコロン ; で区切って指定してください。

CC名の指定が不要の場合には、ヌル文字列を指定してください。

- ptr3 BCC 名
メールのブラインド・コピー先の宛名を、受け取り人の名前かメールアドレスで指定します。名前指定する場合、この名前はアドレス帳に登録されていなければなりません。
複数の宛名を一度に指定したい場合には、半角のセミコロン ; で区切って指定してください。
BCC 名の指定が不要の場合には、ヌル文字列を指定してください。
- ptr4 差出人名
メールの差出人の名前かメールアドレスを指定します。この名前やメールアドレスは、メールアドレスアカウント中に定義されていなければなりません。
差出人名は通常指定する必要はありません(デフォルトの差出人名が適用される)。
差出人名の指定を省略する場合には、ヌル文字列を指定してください。
- ptr5 メールの件名
256 バイト以下の文字列で指定してください。
- ptr6 メール本文
この指定がヌル文字列で、しかも、範囲選択中(箱型選択中の場合を除く)の場合には、範囲選択中の文字列をメール本文として送信します。
マクロ言語の制約により、文字列定数では最大 512 バイト(全角 256 文字)、配列変数では最大 1024 文字までの文字列しか指定できませんが、範囲選択で文字列を指定する場合には、指定できる文字列の長さに事実上の制限はありません。しかし、あまり長い本文を指定するとエラーになります。
本文の指定がない場合、宛先や本文などを指定するためのダイアログボックスが表示されます。
- ptr7 添付ファイル名
添付ファイル名にヌル文字列が指定された場合には、添付ファイルはないと見なされます。
複数の添付ファイルを一度に指定する場合には、半角のセミコロン ; で区切って指定してください。
最初の添付ファイル名は、必ずフルパス名で指定してください。2 つ目以降の添付ファイル名はディレクトリ名を省略できます(省略した場合は最初の添付ファイルと同じディレクトリと見なされます)。
なお、添付ファイルは本文の最後に添付されます。

関数値

正常にメールが送信できると関数値に 0 が返されます。メール送信に失敗した場合には、失敗原因を示す値を関数値に返します。主な関数値を以下に示します。

0	: 正常に送信した
1	: ユーザー操作により中止した
2	: 原因不明の失敗
11	: 添付ファイルが見つからない
14	: 受け取り人が不明(宛先名やCC名の指定ミス)
999	: Simple MAPI のAPI が呼び出せなかった

setenv(ptr1,ptr2)

環境変数の追加/変更/削除

環境変数の追加/変更/削除を行います。

設定した環境変数は、MIFES 自身、および MIFES 上から実行する子プロセスの環境になります。それ以外のアプリケーションには何の影響もありません。

なお、追加/変更した環境変数は、マクロコマンドの終了後もその定義が維持されます。次回以降の MIFES においても維持されます。

引数

ptr1	環境変数名(最大 31バイトの文字列)
ptr2	環境変数の値(最大 511バイトの文字列) ヌル文字列 "" が指定された場合には、指定された環境変数を削除します。

関数値

正常に追加/変更/削除できた場合には 1 が返され、エラーの場合には 0 が返されます。

例

```
setenv("INCLUDE", "C:¥¥Program Files¥¥MEGASOFT¥¥MIFES for Windows  
¥¥MIW7 ")
```

setppp(ptr)

ポストプロセッサの設定

カレントウィンドウに対して、指定したポストプロセッサを設定します。

引数

ptr にはポストプロセッサ名を単純ファイル名で拡張子(.PPP)もあわせて指定します。また、ポストプロセッサの設定を解除する場合にはヌル文字列("")を指定します。

関数値

設定ができれば 1 を返します。

指定したポストプロセッサが見つからないときには 0 を返します。

setprof(val1, val2, ptr) カスタマイズ情報の設定

カスタマイズファイル"MIW.INI"用のマクロ専用キーワード MACRODEF1 ~ MACRODEF16 に値と文字列を定義します。

定義した値と文字列は MIFES の終了時にカスタマイズファイル"MIW.INI"に書き出します。

引数

val1 キーワードID(1 ~ 16)
val2 キーワードに定義したい値
ptr キーワードに定義したい文字列

キーワードをカスタマイズファイルに書き出さないためには、val2 に -1(0xffffffff) を指定します。

ptr には文字列を最大127バイトまで指定できます。(半角で127文字、全角で63文字)

関数値

正常に定義できれば 1 を返します。指定に誤りがあれば 0 を返します。



システム関数 getprof もあわせて参照してください。

setwinpos(vptr) ウィンドウの位置を設定

フレームウィンドウ位置の位置とサイズを変更します。

引数

vptr フレームウィンドウの位置とサイズの情報を指定する配列変数内の位置を指定します。この配列要素位置を [0] として、以下の配列要素に情報を指定してください。

- [0] フレームウィンドウ左端の X 座標 (ピクセル)
- [1] フレームウィンドウ上端の Y 座標 (ピクセル)
- [2] フレームウィンドウの横幅 (ピクセル)
- [3] フレームウィンドウの高さ (ピクセル)

関数値

関数値には、常に 1 が返されます。

例

;画面の右側にはみ出したウィンドウ位置を調整する

```
getwinpos(@str1)
if @str1[0]+@str1[2] > @str1[4]
    @str1[0] = @str1[4]-@str1[2]
if @str1[0] < 0
```



```
        @str1[0] = 0
        @str1[2] = @str1[4]
    endif
    setwinpos(@str1)
endif
```

speak()

指定テキストの読み上げ

この関数は、バージョン7では使用できなくなりました。
記述してあってもコンパイルエラーにはなりませんが、実行しても何も行いません。
関数値には常に -1(0xffffffff) が返されます。

windows(val)

子ウィンドウの整列

MDI子ウィンドウのウィンドウ形状を設定します。

引数

val には以下のマクロ定数のいずれかを指定します。

WINDOWS_TILE	タイル表示にします(並べて表示します)
WINDOWS_CASCADE	カスケード表示にします(重ねて表示します)
WINDOWS_ICONS	アイコンを再配置します
WINDOWS_MAXSIZE	カレントウィンドウを最大化にします
WINDOWS_MAXIMIZE	カレントウィンドウをMDI 最大化にします
WINDOWS_SPLIT1	カレントウィンドウ()と直前のカレントウィンドウ()で画面を左右2分割にする
WINDOWS_SPLIT2	カレントウィンドウ()と直前のカレントウィンドウ()で画面を上下2分割にする

説明

WINDOWS_ICONS は、アイコン化されたMDI子ウィンドウが存在しないときには無効です。
WINDOWS_MAXIMIZE は、カレントウィンドウがすでにMDI 最大化されているときには無効です。

カレントウィンドウの最大化とMDI 最大化とは意味が異なります。

MDI 最大化とは、フレームウィンドウとMDI子ウィンドウのタイトルバーを重ね合わせることです。

一方、カレントウィンドウの最大化は、フレームウィンドウとMDI子ウィンドウのタイトルバーを重ね合わせることなく、MDI子ウィンドウのサイズを可能な限り大きくすることです。

エラーメッセージ一覧

MIL/W 言語のコンパイルエラーメッセージには以下のものがあります。

- 1 (文字列)文の記述に誤りがあります
- 2 case 文では定数以外は指定できません
- 3 else はブロック if 構文でのみ使用できます
- 4 endif 文が足りません
- 5 endsw 文が足りません
- 6 if ~ else ~ endif の構造が変です
- 7 switch ~ endsw の構造が変です
- 8 switch と endsw の間にありません
- 9 wend 文が足りません
- 10 while ~ wend の構造が変です
- 11 while と wend の間 / switch と endsw の間にありません
- 12 アドレス演算子 & は配列要素にのみ指定可能です
- 13 意味のない文です
- 14 演算子(文字列)のオペランドに誤りがあります
- 15 解析不可能な文字列があります
- 16 括弧が正しく対応していません
- 17 関数(文字列)の引き数に誤りがあります
- 18 構文的に誤っています
- 19 式の記述が複雑過ぎます
- 20 対応する if 構文がありません
- 21 対応する switch 構文がありません
- 22 対応する while 構文がありません
- 23 中間コードバッファがオーバーフローしました
- 24 配列(文字列)の要素番号の指定が誤っています
- 25 配列変数の使用方法が誤っています
- 26 プリ/ポスト・インクリメント/デクリメントは単変数にのみ可能です
- 27 ブロック if 構文(if-else-endif)の記述に誤りがあります
- 28 文の記述方法が誤っています
- 29 文法的に誤っています
- 30 変数名の指定が誤ってします
- 31 文字定数(1文字)が長過ぎます
- 32 文字列定数(最大512バイト)が長過ぎます
- 33 ユーザー定義変数が多過ぎます(グローバル/ローカル各16個まで)
- 34 ラベル(文字列)が未定義です
- 35 ラベルの参照場所が多過ぎます
- 36 ラベルの定義が多過ぎます

索引

用語索引

英数字

1 コマンド分のコンパイル	66
10 進定数	23
16 進定数	23
2 項演算子	17
break 文	14
continue 文	15
DOS 版マクロ	3
gosub 文	13
goto 文	12
if 構文	9
MIL/W 言語	2,4,7
MIL 言語	3
MIW.LIB	68,76
MIW.MAC	71
return 文	14
switch 構文	11
while 構文	10

あ行

イーザーヘルプ	27
エラー	164
エラーメッセージ	164
演算子	17
演算式	17

か行

カレントマクロの実行	68
関係演算子	18
関数	94
キー	70
キーボードマクロ	2,6
旧バージョンのマクロ	2,4
グローバル単純変数	20
グローバル配列変数	20
グローバル変数	20
構文	8
コメント	8
コンパイル	66
コンパイルエラー	164

さ行

算術演算子	19
式	12,16
システム変数	22,78
実行文	9,12
シフト演算子	19

小数	20,94
シングルステップ	73
ソースコードの取り出し	74
ソースファイルの書式	7
ソースプログラム	2,66

た行

代入演算子	17
単項演算子	17
単純変数	20,21
中間コード	2,66,72
定数	22

は行

配列変数	20,21
パラメータ	94
引数	94
ファイル全体のコンパイル	67
浮動小数点変数 (calc 関数)	22
変数	20,72,74
ボタン	70

ま行

マクロ定数	24
マクロコマンド名	7
マクロコマンドの実行	67, 68
マクロコマンドの中止	71
マクロモード	72
マクロライブラリ	68,69,70
文字定数	23
文字列定数	23

や行

ユーザー定義変数	72
ユーザー変数	20,72,74

ら行

ライブラリ	68,69,70
ライブラリから実行	70
ライブラリへ格納	69
ラベル	8
ローカル単純変数	21
ローカル配列変数	21
ローカル変数	21
論理演算子	18

システム変数

カーソル位置情報

カーソル位置のバイト位置	@byte	78
カーソル位置の文字コード	@code	78
カーソル位置の論理桁位置	@col	78
カーソルのウィンドウ内の桁位置	@cposx	79
カーソルのウィンドウ内の行位置	@cposy	79
カーソル位置の表示行番号	@line	79
カーソル位置の論理行番号	@num	79
画面上カーソルの論理桁位置	@scol	80

カレントウィンドウ情報

カレントウィンドウの動作状態	@astat	80
カレントウィンドウの表示状態	@dstat	81
カレントウィンドウのハードタブ桁間隔	@htab	82
カレントウィンドウの折り返し桁位置	@margin	82
カレントウィンドウのファイルサイズ	@size	82
カレントウィンドウのテキスト番号	@text	82
カレントウィンドウの折り返し桁位置	@tmpmargin	83
カレントウィンドウの状態	@winstat	83
カレントウィンドウの桁数	@winx	84
カレントウィンドウの行数	@winy	84

以降に開くウィンドウに関する情報

以降に開くウィンドウのウィンドウ形状	@openwin	84
以降に開くウィンドウの動作状態	@sys_astat	84
以降に開くウィンドウの表示状態	@sys_dstat	85
以降に開くウィンドウのハードタブ桁間隔	@sys_htab	86
以降に開くウィンドウの折り返し桁位置	@sys_margin	86

ユーザー入力情報

入力された文字コード	@char	86
入力されたコマンド	@command	87
入力された仮想キー	@key	87

その他の情報

オートセーブ状態	@autosave	89
オートセーブの間隔	@saveinterval	91
テキスト表示状態	@disp	89
カレント検索方法	@findmode	89
挿入 / 上書き状態	@insmode	90
トレース用罫線種	@keisen	90
論理行テキストの残り	@nextbyte	90
プロファイル設定	@profile	90
callDll()の関数値	@retcode	90
ツールバー表示の状態	@ribbon	91
範囲選択状態	@selmode	92
選択範囲の先頭位置	@sel_start	91
選択範囲の最終位置	@sel_end	91
起動属性の状態	@sys_stat	93

システム関数

文字列操作関数

文字列による実数演算の実行	calc()	94
ディレクトリ名の取得	getdir()	98
環境文字列の取得	getenv()	99
検索履歴文字列の取得	gethistory()	99
カーソル行の取得	getline()	100
編集ファイル名の取得	getpath()	100
カスタマイズ情報の取得	getprof()	101
選択文字列の取得	getselstring()	101
カーソル位置の文字列取得	getstring()	102
カーソル位置の1語取得	getword()	102
書式文字列の取得	sprintf()	102
文字列の比較：区別	strcmp()	104
文字列の比較：同一視	strcmpi()	104
文字列の複写	strcpy()	105
文字列並びの生成	strlist()	105
指定文字数の文字列複写	strncpy()	106

文字列の挿入、削除関数

選択範囲の複写	copy()	106
選択範囲の移動	cut()	106
指定バイト数の削除	delbyte()	107
指定文字数の削除	delchar()	107
カーソル行の削除	delline()	108
カーソル行の2重化	duplicate()	108
文字の挿入	insstr()	108
別のウィンドウに文字列を挿入	insstrex()	109
等差数数列の出力	outaseries()	109
等差数数列の初期化	setaseries()	111
文字列のペースト	paste()	109
書式文字列の挿入	printf()	109
カーソル位置の文字列置き換え	setstring()	112

ジャンプ、移動、検索、置換関数

複数置換	breplace()	113
グローバル検索	gsearch()	114
ジャンプ、ウィンドウ切り替え	jump()	116
カーソル移動	move()	117
対応する括弧の検索	parenthesis()	117
文字列の置換	replace()	118
文字列の検索	search()	119
文書整形	reform()	120
タグジャンプ、バックタグジャンプ	tagjump()	121

ユーザー入力関数

コンボボックスで文字列の入力	combobox()	122
日本語入力システム(FEP)のオン/オフ	fep()	122
ファイル名の入力	finput()	123
文字列の入力	input()	123
リストボックスでの選択	listbox()	124
メッセージボックスの表示	messagebox()	124
マクロ中止ウィンドウの消去	stopoff()	125

マクロ中止ウィンドウの表示	stopon()	125
マクロ用ユーザ変数の表示と変更	variables()	126
イベント待ち	waitevent()	126

カスタマイズ情報の設定	setprof()	162
フレームウィンドウ位置を設定する	setwinpos()	162
指定テキストの読み上げ	speak()	163
子ウィンドウの整列	windows()	163

ファイル操作関数

カレントディレクトリの変更	chdir()	129
ウィンドウのクローズ	close()	129
ファイルの複写	copyfile()	130
ファイルのクローズ	fclose()	131
ファイルの書き込みオープン	fcreat()	131
ファイル名リストの作成	findfile()	132
ファイルのチェック	fnamchk()	133
ファイルの読み込みオープン	fopen()	133
ファイルから1行読み込み	fread()	133
ファイルへ1行書き込み	fwrite()	134
ファイル名リストからファイル名を得る	getfile()	135
ファイルの日時を得る	getftime()	135
外部ファイルの挿入	insfile()	137
新規ウィンドウを開く	newopen()	137
ウィンドウを開く	open()	137
編集のやり直し	original()	138
印刷の実行	outprinter()	138
ファイル名の変更	rename()	140
ファイルへの保存	save()	140
別ファイルへの保存	saveas()	141
ファイルの日時を設定	setftime()	141
プリンタの設定	setprinter()	142
ファイルの削除	unlink()	142

その他の関数

文字列を数値に変換	atol()	143
ビーブ音を鳴らす	beep()	143
マクロ実行のブレーク	bp()	143
外部DLLの呼び出し	calldll()	144
カラーの変更	chgcolor()	146
マウスカーソルの形状変更	chgcursor()	148
表示フォントの変更	chgfont()	148
ヘルプファイルの変更	chghelp()	149
子プロセスの実行	child()	149
メッセージの消去	clsmess()	150
マクロコマンドのコンパイル	compile()	150
文字の種類を取得	ctype()	151
エディタの終了	end()	151
機能番号の実行	execmd()	152
指定したURLをブラウザで表示	exeurl()	152
マクロコマンドの終了	exit()	155
ローカル日時を得る	gettime()	153
フレームウィンドウ位置を得る	getwinpos()	153
リストウィンドウの表示	itemlist()	154
マクロコマンドの実行	macro()	155
メッセージの表示	message()	156
ウェーブフォームの再生	playsound()	156
カスタマイズファイルの読み込み	readprofile()	157
ツールバー表示の変更	ribbon()	158
メールの送信	sendmail()	159
環境変数の追加/変更/削除	setenv()	161
ポストプロセッサの設定	setppp()	161

マクロ定数

動作状態定義

EOFコード無視	ASTAT_EOFCTRL	80, 85
JIS 罫線有効	ASTAT_KEISEN	80, 85
SHIFT+ キーで範囲選択	ASTAT_SHIFTLTRT	80, 85
SHIFT+ キーで範囲選択	ASTAT_SHIFTUPDN	80, 85
オートインデント有効	ASTAT_INDENT	80, 84
大文字変換状態	ASTAT_CHGSCALE	80, 85
漢字処理オフ	ASTAT_NOKANJI	80, 85
行端でカーソルをストップ	ASTAT_EDGESTOP	80, 85
ソフトタブ有効	ASTAT_SOFTTAB	80, 85
フリーカーソルモード	ASTAT_FREECURSOR	80, 85
「保存」時の変更行マークの処理	ASTAT_SAVECLEAR	80, 85
マウスによるカーソル移動有効	ASTAT_MOUSEPOS	80, 85
メモマーク有効	ASTAT_MEMOMARK	80, 85
Enter キーで挿入する改行文字	ASTAT_ENTERLF	80, 85
横スクロール動作	ASTAT_HSCROLL	80, 85
マウスのトリプルクリックの禁止	ASTAT_NO3CLICK	85

表示状態定数

[EOF] マーク非表示	DSTAT_EOFMARK	81, 86
アンダーライン非表示	DSTAT_UNDERLINE	81, 86
改行文字明示	DSTAT_CRLF	81, 85
ガイドライン表示	DSTAT_GUIDELINE	81, 86
桁ゲージ表示	DSTAT_COLGAGE	81, 85
行ゲージ表示	DSTAT_NUMGAGE	81, 85
垂直スクロールバー表示	DSTAT_VMSCROLL	81, 85
水平スクロールバー表示	DSTAT_HSCROLL	81, 85
パーティカルライン2本	DSTAT_2VLINES	81, 86
パーティカルライン表示	DSTAT_VERTICALLINE	81, 86
ハードタブ明示	DSTAT_HTAB	81, 85
表示行番号ゲージ	DSTAT_NUMTYPE	81, 85
変更行明示	DSTAT_UPDATE	81, 86
右マージン右側を明示	DSTAT_REDGE	81, 86
背景横罫線の表示	DSTAT_BACKLINE	81, 86
対応括弧の明示	DSTAT_KAKKO	81, 86
対応括弧の明示方法	DSTAT_KAKKOMARK	81, 86
全角スペースの明示	DSTAT_KANSPLACE	81, 86

システム状態定数

MIW.MACの自動コンパイルせず	SYS_MIWMAC	93
SDIウィンドウの横幅	SYS_SDISIZE	93
1画面単位Pageup/PageDown	SYS_PAGEID	93
「ウィンドウ」メニュー中のファイル名非表示	SYS_MDIWINMENU	93
起動時上書きモード	SYS_OVW	93

起動時「新規:nn」ウィンドウを開く	SYS_OPEN	93
キーボードマクロ定義時	SYS_KEYMTOLIB	93
コモンダイアログボックス	SYS_DIALOG	93
終了時カットバッファ削除	SYS_CUTBUFF	93
同時複数起動許可	SYS_MULTI	93
ファイル内容による自動コード判定禁止	SYS_NOCODECHK	93
「ファイル」メニュー中のファイル名	SYS_FNMENU	93
ボタンの吹き出し表示	SYS_BALOON	93
Unicodeファイルへの保存時のBOM書き込み禁止	SYS_NOBOMUNICODE	93
Unicode Big endianファイルへのBOM書き込み禁止	SYS_NOBOMUNICBIG	93
UTF-8ファイルへのBOM書き込み禁止	SYS_NOBOMUTF8	93
方向範囲選択時のカーソル文字の扱い	SYS_NEWSEL	5, 92
「ファイルを開く」ダイアログボックスの初期リスト	SYS_CFDIR	93
多目的バーの配置位置	SYS_UPPERBAR	93

イベント定数

コマンドの指定を待つ	EVENT_COMMAND	24, 42, 49, 52, 58, 88, 126, 129
仮想キー入力を待つ	EVENT_VK	24, 126
文字の入力を待つ	EVENT_CHAR	127
0.1秒間待つ	EVENT_TIMER01	127
1秒間待つ	EVENT_TIMER1	127
10秒間待つ	EVENT_TIMER10	127
60秒間待つ	EVENT_TIMER60	127
左ボタンのクリックを待つ	EVENT_LBUTTON	127
右ボタンのクリックを待つ	EVENT_RBUTTON	127
左ボタンのダブルクリックを待つ	EVENT_LDBLCLK	127
右ボタンのダブルクリックを待つ	EVENT_RDBLCLK	127

仮想キー番号定数

[F1]キー	VK_F1	128
[F2]キー	VK_F2	128
[F3]キー	VK_F3	128
[F4]キー	VK_F4	128
[F5]キー	VK_F5	128
[F6]キー	VK_F6	128
[F7]キー	VK_F7	128
[F8]キー	VK_F8	128
[F9]キー	VK_F9	128
[F10]キー	VK_F10	128
[F11]キー	VK_F11	128
[F12]キー	VK_F12	128
[PageDown]キー	VK_PAGEDOWN	87, 128
[PageUp]キー	VK_PAGEUP	87, 128
[Ins]キー	VK_INS	87, 128

[Del]キー	VK_DEL	87, 128
[]キー	VK_UP	87, 128
[]キー	VK_LEFT	87, 128
[]キー	VK_RIGHT	87, 128
[]キー	VK_DOWN	87, 128
[Home]キー	VK_HOME	87, 128
[End]キー	VK_END	87, 128
[Esc]キー	VK_ESC	12, 88, 128

コマンド番号定数

1文字削除	COMMAND_DEL	127, 129
直前の1文字削除	COMMAND_BS	10, 127
タブ文字挿入/次のタブ位置へ	COMMAND_TAB	127
行分割/改行	COMMAND_RET	10, 42, 49, 52, 58, 127
カーソル 移動	COMMAND_UP	127
カーソル 移動	COMMAND_DOWN	127
カーソル 移動	COMMAND_LEFT	127
カーソル 移動	COMMAND_RIGHT	127
選択範囲の切り取り	COMMAND_CUT	127
選択範囲のコピー	COMMAND_COPY	127
行カットバッファの貼り付け	COMMAND_PASTEL	127
クリップボードの貼り付け	COMMAND_PASTES	127
挿入/上書の切り替え	COMMAND_INSMODE	127

メッセージボックス定数

[OK] ボタン	MB_OK	45, 49, 61, 63, 125
[OK][キャンセル] ボタン	MB_OKCANCEL	45, 125
[はい][いいえ] ボタン	MB_YESNO	45, 125
[はい][いいえ][キャンセル] ボタン	MB_YESNOCANCEL	45, 125
[中止][再試行][無視] ボタン	MB_ABORTRETRYIGNORE	45, 125
[STOP] アイコン付き	MB_ICONSTOP	45, 49, 125
疑問符アイコン付き	MB_ICONQUESTION	125
[i] アイコン付き	MB_ICONINFORMATION	45, 49, 125
感嘆符アイコン付き	MB_ICONEXCLAMATION	125
[中止] ボタンが押された	IDABORT	125
[キャンセル] ボタンが押された	IDCANCEL	125
[無視] ボタンが押された	IDIGNORE	125
[いいえ] ボタンが押された	IDNO	125
[OK] ボタンが押された	IDOK	125
[再試行] ボタンが押された	IDRETRY	125

[はい] ボタンが押された	IDYES	125
-----------------	-------	-----

ウィンドウ形状定数

タイル表示	WINDOWS_TILE	163
カスケード表示	WINDOWS_CASCADE	163
アイコン整列	WINDOWS_ICONS	163
最大サイズ	WINDOWS_MAXSIZE	163
MDI 最大化	WINDOWS_MAXIMIZE	163
左右スプリット	WINDOWS_SPLIT1	163
上下スプリット	WINDOWS_SPLIT2	163

検索/置換用定数

下位ディレクトリもグローバル検索	GS_NORMAL	115
指定ディレクトリだけグローバル検索	GS_1DIR	115
開いているファイルだけをグローバル検索	GS_ALLOPEN	115
グローバル検索結果をカレントウィンドウに追加出力	GS_APPEND	115
グローバル検索時のUNICODE ファイル判断	GS_UNICODE	115
グローバル検索集計結果も出力する	GS_SUMUP	115
他マクロとの論理積グローバル検索	GS_AND	115
他マクロとの論理和グローバル検索	GS_OR	115
文字列を含まない行を検索	GS_NOT	115
どちらの文字列も含まない行を検索	GS_NOTBOTH	115
確認付き置換	REP_QUERY	113, 118
確認なし置換	REP_BATCH	113, 118
最終行番号	MAX_NUMBER	55, 61, 64, 118

OPEN 関数用定数

読み取り専用で開く	OPEN_READONLY	138
「テキスト (^Z) まで」で開く	OPEN_EOFTEXT	138
「テキスト」で開く	OPEN_TEXT	132, 135, 138
「バイナリ」で開く	OPEN_BINARY	138
「自動設定」で開く	OPEN_AUTO	138
ファイル履歴に記録する	OPEN_NOHISTORY	138

選択状態定数

行単位選択	SEL_LINE	92
箱型選択	SEL_BOX	92
範囲選択中止	SEL_CANCEL	92
文字列選択	SEL_STRING	92

その他の定数

1コマンドのコンパイル	COMPILE_COMMAND	150
MIW.MAC自動コンパイルのキャンセル	COMPILE_NOMIWMAC	150
以降に開くウィンドウのフォント変更	FONT_SYS	148
カレントウィンドウのフォント変更	FONT_WIN	148
ソースコード埋め込み	COMPILE_SOURCE	150
ファイル全体のコンパイル	COMPILE_FILE	150
複写先が新しいなら複写せず	COPY_CHECKTIME	130
複写先が読み取り専用属性でも複写せず	COPY_READONLY	130
複写元に保存属性なしなら複写せず	COPY_CHECKATTR	130
ライブラリ自動格納	COMPILE_LIBRARY	150
両方のフォント変更	FONT_BOTH	148
砂時計カーソル	CURSOR_WAIT	148
矢印カーソル	CURSOR_NORMAL	148

システム関数

A

@astat	カレントウィンドウの動作状態	80
ASTAT_CHGSCALE	大文字変換状態	80, 85
ASTAT_EDGESTOP	行端でカーソルをストップ	80, 85
ASTAT_ENTERLF	Enterキーで挿入する改行文字	80, 85
ASTAT_EOFCTRL	EOFコード無視	80, 85
ASTAT_FREECURSOR	フリーカーソルモード	80, 85
ASTAT_HSCROLL	横スクロール動作	80, 85
ASTAT_INDENT	オートインデント有効	80, 84
ASTAT_KEISEN	JIS 罫線有効	80, 85
ASTAT_MEMOMARK	メモマーク有効	80, 85
ASTAT_MOUSEPOS	マウスによるカーソル移動有効	80, 85
ASTAT_NO3CLICK	マウスのトリプルクリックの禁止	85
ASTAT_NOKANJI	漢字処理オフ	80, 85
ASTAT_SAVECLEAR	「保存」時の変更行マークの	80, 85
ASTAT_SHIFTLTR	SHIFT+ キーで範囲選択	80, 85
ASTAT_SHIFTUPDN	SHIFT+ キーで範囲選択	80, 85
ASTAT_SOFTTAB	ソフトタブ有効	80, 85
atol()	文字列を数値に変換	143
@autosave	オートセーブ状態	89

B

beep()	ビーブ音を鳴らす	143
bp()	マクロ実行のブレーク	143
breplace()	複数置換	113
@byte	カーソル位置のバイト位置	78

C

calc()	文字列による実数演算の実行	94
callDll()	外部DLLの呼び出し	144
@char	入力された文字コード	86
chdir()	カレントディレクトリの変更	129
chgcolor()	カラーの変更	146
chgcursor()	マウスカーソルの形状変更	148
chgfont()	表示フォントの変更	148
chghelp()	ヘルプファイルの変更	149
child()	子プロセスの実行	149
close()	ウィンドウのクローズ	129
clsmess()	メッセージの消去	150
@code	カーソル位置の文字コード	78
@col	カーソル位置の論理桁位置	78
combobox()	コンボボックスで文字列の入力	122
@command	入力されたコマンド	87
COMMAND_BS	直前の1文字削除	10, 127
COMMAND_COPY	選択範囲のコピー	127
COMMAND_CUT	選択範囲の切り取り	127
COMMAND_DEL	1文字削除	127, 129
COMMAND_DOWN	カーソル 移動	127
COMMAND_INSMODE	挿入/上書の切り替え	127
COMMAND_LEFT	カーソル 移動	127
COMMAND_PASTEL	行カットパッファの貼り付け	127
COMMAND_PASTES	クリップボードの貼り付け	127
COMMAND_RET	行分割/改行	10, 42, 49, 52, 58, 127

COMMAND_RIGHT	カーソル 移動	127
COMMAND_TAB	タブ文字挿入/次のタブ位置へ	127
COMMAND_UP	カーソル 移動	127
compile()	マクロコマンドのコンパイル	150
COMPILE_COMMAND	1コマンドのコンパイル	150
COMPILE_FILE	ファイル全体のコンパイル	150
COMPILE_LIBRARY	ライブラリ自動格納	150
COMPILE_NOMIWMAC		
MIW.MAC	自動コンパイルのキャンセル	150
COMPILE_SOURCE	ソースコード埋め込み	150
copy()	選択範囲の複写	106
COPY_CHECKATTR		
	複写元に保存属性なしなら複写せず	130
COPY_CHECKTIME		
	複写先が新しいなら複写せず	130
COPY_READONLY		
	複写先が読み取り専用属性でも複写せず	130
copyfile()	ファイルの複写	130
@cposx	カーソルのウィンドウ内の桁位置	79
@cposy	カーソルのウィンドウ内の行位置	79
ctype()	文字の種類を取得	151
CURSOR_NORMAL	矢印カーソル	148
CURSOR_WAIT	砂時計カーソル	148
cut()	選択範囲の移動	106

D

delbyte()	指定バイト数の削除	107
delchar()	指定文字数の削除	107
delline()	カーソル行の削除	108
@disp	テキスト表示状態	89
@dstat	カレントウィンドウの表示状態	81
DSTAT_2VLINES	パーチカルライン2本	81, 86
DSTAT_BACKLINE	背景横罫線の表示	81, 86
DSTAT_COLGAGE	桁ゲージ表示	81, 85
DSTAT_CRLF	改行文字明示	81, 85
DSTAT_EOFMARK	[EOF]マーク非表示	81, 86
DSTAT_GUIDELINE	ガイドライン表示	81, 86
DSTAT_HSCROLL	水平スクロールバー表示	81, 85
DSTAT_HTAB	ハードタブ明示	81, 85
DSTAT_KAKKO	対応括弧の明示	81, 86
DSTAT_KAKKOMARK	対応括弧の明示方法	81, 86
DSTAT_KANSPEC	全角スペースの明示	81, 86
DSTAT_NUMGAGE	行ゲージ表示	81, 85
DSTAT_NUMTYPE	表示行番号ゲージ	81, 85
DSTAT_REDGE	右マージン右側を明示	81, 86
DSTAT_UNDERLINE	アンダーライン非表示	81, 86
DSTAT_UPDATE	変更行明示	81, 86
DSTAT_VERTICALLINE	パーチカルライン表示	81, 86
DSTAT_VSCROLL	垂直スクロールバー表示	81, 85
dupline()	カーソル行の2重化	108

E

end()	エディタの終了	151
EVENT_CHAR	文字の入力を待つ	127
EVENT_COMMAND	コマンドの指定を待つ	24, 42, 49, 52, 58, 88, 126, 129
EVENT_LBUTTON	左ボタンのクリックを待つ	127

EVENT_LDBLCLK	左ボタンのダブルクリックを待つ	127
EVENT_RBUTTON	右ボタンのクリックを待つ	127
EVENT_RDBLCLK	右ボタンのダブルクリックを待つ	127
EVENT_TIMER01	0.1秒間待つ	127
EVENT_TIMER1	1秒間待つ	127
EVENT_TIMER10	10秒間待つ	127
EVENT_TIMER60	60秒間待つ	127
EVENT_VK	仮想キー入力を待つ	24, 126
execmd()	機能番号の実行	152
exeurl()	指定したURLをブラウザで表示	152
exit()	マクロコマンドの終了	155

F

fclose()	ファイルのクローズ	131
fcreat()	ファイルの書き込みオープン	131
fep()	日本語入力システム(FEP)のオン/オフ	122
findfile()	ファイル名リストの作成	132
@findmode	カレント検索方法	89
finput()	ファイル名の入力	123
fnamchk()	ファイルのチェック	133
FONT_BOTH	両方のフォント変更	148
FONT_SYS	以降に開くウィンドウのフォント変更	148
FONT_WIN	カレントウィンドウのフォント変更	148
fopen()	ファイルの読み込みオープン	133
fread()	ファイルから1行読み込み	133
fwrite()	ファイルへ1行書き込み	134

G

getdir()	ディレクトリ名の取得	98
getenv()	環境文字列の取得	99
getfile()	ファイル名リストからファイル名を得る	135
getftime()	ファイルの日時を得る	135
gethistory()	検索履歴文字列の取得	99
getline()	カーソル行の取得	100
getpath()	編集ファイル名の取得	100
getprof()	カスタマイズ情報の取得	101
getselstring()	選択文字列の取得	101
getstring()	カーソル位置の文字列取得	102
gettime()	ローカル日時を得る	153
getwinpos()	フレームウィンドウ位置を得る	153
getword()	カーソル位置の1語取得	102
GS_1DIR	指定ディレクトリだけグローバル検索	115
GS_ALLOPEN	開いているファイルだけをグローバル検索	115
GS_AND	他マクロとの論理積グローバル検索	115
GS_APPEND		
	グローバル検索結果をカレントウィンドウに追加出力	115
GS_NORMAL	下位ディレクトリもグローバル検索	115
GS_NOT	文字列を含まない行を検索	115
GS_NOTBOTH	どちらの文字列も含まない行を検索	115
GS_OR	他マクロとの論理和グローバル検索	115
GS_SUMUP	グローバル検索集計結果も出力する	115
GS_UNICODE		
	グローバル検索時のUNICODEファイル判断	115
gsearch()	グローバル検索	114

H	
@htab	カレントウィンドウのハードタブ間隔 82

I	
IDABORT	[中止] ボタンが押された 125
IDCANCEL	[キャンセル] ボタンが押された 125
IDIGNORE	[無視] ボタンが押された 125
IDNO	[いいえ] ボタンが押された 125
IDOK	[OK] ボタンが押された 125
IDRETRY	[再試行] ボタンが押された 125
IDYES	[はい] ボタンが押された 125
input()	文字列の入力 123
insfile()	外部ファイルの挿入 137
@insmode	挿入 / 上書き状態 90
insstr()	文字の挿入 108
insstrex()	別のウィンドウに文字列を挿入 109
itemlist()	リストウィンドウの表示 154

J	
jump()	ジャンプ、ウィンドウ切り替え 116

K	
@keisen	トレース用罫線種 90
@key	入力された仮想キー 87

L	
@line	カーソル位置の表示行番号 79
listbox()	リストボックスでの選択 124

M	
macro()	マクロコマンドの実行 155
@margin	カレントウィンドウの折り返し桁位置 82
MAX_NUMBER	最終行番号 55, 61, 64, 118
MB_ABORTRETRYIGNORE	[中止][再試行][無視] ボタン 45, 125
MB_ICONEXCLAMATION	感嘆符アイコン付き 125
MB_ICONINFORMATION	[i] アイコン付き 45, 49, 125
MB_ICONQUESTION	疑問符アイコン付き 125
MB_ICONSTOP	[STOP] アイコン付き 45, 49, 125
MB_OK	[OK] ボタン 45, 49, 61, 63, 125
MB_OKCANCEL	[OK][キャンセル] ボタン 45, 125
MB_YESNO	[はい][いいえ] ボタン 45, 125
MB_YESNOCANCEL	[はい][いいえ][キャンセル] ボタン 45, 125
message()	メッセージの表示 156
messagebox()	メッセージボックスの表示 124
move()	カーソル移動 117

N	
newopen()	新規ウィンドウを開く 137
@nextbyte	論理行テキストの残り 90

@num	カーソル位置の論理行番号 79
------	-----------------

O	
open()	ウィンドウを開く 137
OPEN_AUTO	「自動設定」で開く 138
OPEN_BINARY	「バイナリ」で開く 138
OPEN_EOFTEXT	「テキスト(^Z)まで」で開く 138
OPEN_NOHISTORY	ファイル履歴に記録する 138
OPEN_READONLY	読み取り専用で開く 138
OPEN_TEXT	「テキスト」で開く 132, 135, 138
@openwin	以降に開くウィンドウのウィンドウ形状 84
original()	編集のやり直し 138
outaseries()	等差数字列の出力 109
outprinter()	印刷の実行 138

P	
parenthesis()	対応する括弧の検索 117
paste()	文字列のペースト 109
playsound()	ウェブフォームの再生 156
printf()	書式文字列の挿入 109
@profile	プロファイル設定 90

R	
readprofile()	カスタマイズファイルの読み込み 157
reform()	文書整形 120
rename()	ファイル名の変更 140
REP_BATCH	確認なし置換 113, 118
REP_QUERY	確認付き置換 113, 118
replace()	文字列の置換 118
@retcode	callDll()の戻り値 90
@ribbon	ツールバー表示の状態 91
ribbon()	ツールバー表示の変更 158

S	
save()	ファイルへの保存 140
saveas()	別ファイルへの保存 141
@saveinterval	オートセーブの間隔 91
@scol	画面上カーソルの論理桁位置 80
search()	文字列の検索 119
SEL_BOX	箱型選択 92
SEL_CANCEL	範囲選択中止 92
@sel_end	選択範囲の最終位置 91
SEL_LINE	行単位選択 92
@sel_start	選択範囲の先頭位置 91
SEL_STRING	文字列選択 92
@selmode	範囲選択状態 92
sendmail()	メールの送信 159
setaseries()	等差数字列の初期化 111
setenv()	環境変数の追加 / 変更 / 削除 161
setftime()	ファイルの日時を設定 141
setppp()	ポストプロセッサの設定 161
setprinter()	プリンタの設定 142
setprof()	カスタマイズ情報の設定 162
setstring()	カーソル位置の文字列置き換え 112
setwinpos()	フレームウィンドウ位置を設定する 162

@size	カレントウィンドウのファイルサイズ	82	VK_ESC	[Esc]キー	12, 88, 128
speak()	指定テキストの読み上げ	163	VK_F1	[F1]キー	128
sprintf()	書式文字列の取得	102	VK_F10	[F10]キー	128
stopoff()	マクロ中止ウィンドウの消去	125	VK_F11	[F11]キー	128
stopon()	マクロ中止ウィンドウの表示	125	VK_F12	[F12]キー	128
strcmp()	文字列の比較：区別	104	VK_F2	[F2]キー	128
strcmppi()	文字列の比較：同一視	104	VK_F3	[F3]キー	128
strcpy()	文字列の複写	105	VK_F4	[F4]キー	128
strlist()	文字列並びの生成	105	VK_F5	[F5]キー	128
strncpy()	指定文字数の文字列複写	106	VK_F6	[F6]キー	128
@sys_astat	以降に開くウィンドウの動作状態	84	VK_F7	[F7]キー	128
SYS_BALOON	ボタンの吹き出し表示	93	VK_F8	[F8]キー	128
SYS_CFDIR			VK_F9	[F9]キー	128
	「ファイルを開く」ダイアログボックスの初期リスト	93	VK_HOME	[Home]キー	87, 128
SYS_CUTBUFF	終了時カットバッファ削除	93	VK_INS	[Ins]キー	87, 128
SYS_DIAOG	コモンダイアログボックス	93	VK_LEFT	[]キー	87, 128
@sys_dstat	以降に開くウィンドウの表示状態	85	VK_PAGEDOWN	[PageDown]キー	87, 128
SYS_FNMENU	「ファイル」メニュー中のファイル名	93	VK_PAGEUP	[PageUp]キー	87, 128
@sys_htab	以降に開くウィンドウのハードタブ桁間隔	86	VK_RIGHT	[]キー	87, 128
SYS_KEYMTOLIB	キーボードマクロ定義時	93	VK_UP	[]キー	87, 128
@sys_margin	以降に開くウィンドウの折り返し桁位置	86			
SYS_MDIWINMENU					
	「ウィンドウ」メニュー中のファイル名非表示	93			
SYS_MIWMAC	MIW.MACの自動コンパイルせず	93			
SYS_MULTIC	同時複数起動許可	93			
SYS_NEWSEL					
	方向範囲選択時のカーソル文字の扱い	5, 92			
SYS_NOBOMUNICBIG					
	Unicode Big endian ファイルへのBOM書き込み禁止	93			
SYS_NOBOMUNICODE					
	Unicode ファイルへの保存時のBOM書き込み禁止	93			
SYS_NOBOMUTF8					
	UTF-8 ファイルへのBOM書き込み禁止	93			
SYS_NOCODECHK					
	ファイル内容による自動コード判定禁止	93			
SYS_OPEN					
	起動時「新規:nn」ウィンドウを開く	93			
SYS_OVW	起動時上書きモード	93			
SYS_PAGEID	1画面単位Pageup/PageDown	93			
SYS_SDSIZE	SDIウィンドウの横幅	93			
@sys_stat	起動属性の状態	93			
SYS_UPPERBAR	多目的バーの配置位置	93			

T

tagjump()	タグジャンプ、バックタグジャンプ	121
@text	カレントウィンドウのテキスト番号	82
@tmpmargin	カレントウィンドウの折り返し桁位置	83

U

unlink()	ファイルの削除	142
----------	---------	-----

V

variables()	マクロ用ユーザ変数の表示と変更	126
VK_DEL	[Del]キー	87, 128
VK_DOWN	[]キー	87, 128
VK_END	[End]キー	87, 128

W

waitevent()	イベント待ち	126
windows()	子ウィンドウの整列	163
WINDOWS_CASCADE	カスケード表示	163
WINDOWS_ICONS	アイコン整列	163
WINDOWS_MAXIMIZE	MDI最大化	163
WINDOWS_MAXSIZE	最大サイズ	163
WINDOWS_SPLIT1	左右スプリット	163
WINDOWS_SPLIT2	上下スプリット	163
WINDOWS_TILE	タイル表示	163
@winstat	カレントウィンドウの状態	83
@winx	カレントウィンドウの桁数	84
@winy	カレントウィンドウの行数	84

MEMO

MEMO

MEMO

MEMO

MIFES for Windows Ver.7.0 マクロマニュアル

2005年2月18日 初版

制 作 メガソフト株式会社 技術本部

発行元 メガソフト株式会社
〒564-0053 大阪府吹田市江の木町1-38 西谷東急ビル
TEL.06-6386-6810 FAX.06-6386-9983

Copyright© 2005 MEGASOFT Inc.
