

AJAN

標準コマンド
リファレンス

目 次

第1章 はじめに	8
第2章 標準コマンド	9
2.1 コマンド一覧	9
2.2 AJAN メイン操作に関する関数・命令	14
2.2.1 ASSERT	14
2.2.2 CLS	14
2.2.3 END	15
2.2.4 ERF\$	15
2.2.5 ERL	16
2.2.6 ERM\$	16
2.2.7 ERN\$	17
2.2.8 ERR	17
2.2.9 ERRMSG\$	18
2.2.10 ERRSUB	18
2.2.11 ERROR	19
2.2.12 OPTION ERROR	21
2.2.13 OPTION STOP	22
2.2.14 REM	23
2.2.15 SETERRTBL	24
2.2.16 SLEEP	25
2.2.17 STOP	25
2.3 数値・文字列に関する関数・命令	26
2.3.1 ABS	26
2.3.2 ARRAY2CSV\$	27
2.3.3 ASC	28
2.3.4 ASCB	28
2.3.5 ATN	29
2.3.6 BIN\$	29
2.3.7 CDATETIME	30
2.3.8 CHR\$	31
2.3.9 CHRB\$	31
2.3.10 CLOCK	32
2.3.11 COS	32
2.3.12 CSV2ARRAY\$	33
2.3.13 CVD	35
2.3.14 CVH	36
2.3.15 CVI	37
2.3.16 CVL	38
2.3.17 CVS	38
2.3.18 DATE\$	39

2.3.19 EXP	39
2.3.20 FIX	40
2.3.21 HEX\$.....	40
2.3.22 INSTR	41
2.3.23 INSTRREV	42
2.3.24 INT	43
2.3.25 ISDATETIME.....	44
2.3.26 ISNUMERIC.....	45
2.3.27 ISNAN.....	46
2.3.28 ISINF	46
2.3.29 LEN	47
2.3.30 LENB	47
2.3.31 LINSTR.....	48
2.3.32 LOG.....	49
2.3.33 LSTRIP\$	49
2.3.34 MEMINFO.....	50
2.3.35 MID\$	51
2.3.36 MIDB\$	52
2.3.37 MKD\$.....	53
2.3.38 MKH\$.....	54
2.3.39 MKI\$	55
2.3.40 MKL\$	56
2.3.41 MKS\$	56
2.3.42 RANDOMIZE.....	57
2.3.43 REPLACES\$	57
2.3.44 RND	58
2.3.45 ROUND.....	59
2.3.46 RSTRIP\$	60
2.3.47 SIN	61
2.3.48 SPLIT\$	62
2.3.49 SQR.....	63
2.3.50 STR\$.....	63
2.3.51 STRDEL\$.....	64
2.3.52 STRINS\$.....	65
2.3.53 STRIP\$.....	65
2.3.54 TAN.....	66
2.3.55 TIMES\$	67
2.3.56 TRIM\$.....	69
2.3.57 VAL.....	69
2.3.58 WEEK	70
2.4 型の宣言や変換に関する関数・命令	71
2.4.1 BOOL	71
2.4.2 CONST.....	72
2.4.3 CONST BOOL.....	72
2.4.4 CONST DATETIME	73
2.4.5 DATETIME	74

2.4.6	DEFINE STRUCT～END STRUCT	75
2.4.7	ENUM～END ENUM	77
2.4.8	LOCAL	78
2.4.9	LOCAL BOOL	79
2.4.10	LOCAL CONST	80
2.4.11	LOCAL CONST BOOL	81
2.4.12	LOCAL CONST DATETIME	82
2.4.13	LOCAL DATETIME	82
2.4.14	LOCAL ENUM～END ENUM	83
2.4.15	LOCAL STRUCT	84
2.4.16	STRUCT	85
2.4.17	VARTYPE	86
2.5	配列に関する関数・命令	87
2.5.1	ARRAY2DICT	87
2.5.2	CDIM	88
2.5.3	CLEAR_DICT	88
2.5.4	DEL_DICT_KEY	89
2.5.5	DICT	90
2.5.6	DICT BOOL	91
2.5.7	DICT DATETIME	91
2.5.8	DICT STRUCT	92
2.5.9	DICT2ARRAY\$	93
2.5.10	DIM	94
2.5.11	GET_DICT_KEY\$	96
2.5.12	HAS_DICT_KEY	96
2.5.13	LDICT	98
2.5.14	LDIM	98
2.5.15	LIST	99
2.5.16	LIST BOOL	100
2.5.17	LIST DATETIME	100
2.5.18	LIST DICT	101
2.5.19	LIST DICT BOOL	102
2.5.20	LIST DICT DATETIME	103
2.5.21	LIST DICT STRUCT	104
2.5.22	LIST STRUCT	105
2.5.23	LOCAL DICT	106
2.5.24	LOCAL DICT BOOL	106
2.5.25	LOCAL DICT DATETIME	107
2.5.26	LOCAL DICT STRUCT	107
2.5.27	LOCAL LIST	107
2.5.28	LOCAL LIST BOOL	108
2.5.29	LOCAL LIST DATETIME	108
2.5.30	LOCAL LIST DICT	108
2.5.31	LOCAL LIST DICT BOOL	109
2.5.32	LOCAL LIST DICT DATETIME	109
2.5.33	LOCAL LIST DICT STRUCT	110

2.5.34	LOCAL LIST STRUCT	110
2.5.35	ONEDIM INSERT	111
2.5.36	ONEDIM REMOVE	112
2.5.37	REDIM	113
2.5.38	TWODIM INSERT	114
2.5.39	TWODIM REMOVE	116
2.5.40	UBOUND	117
2.6	繰り返し・条件分岐に関する関数・命令	118
2.6.1	DO WHILE～LOOP	118
2.6.2	EXIT DO	119
2.6.3	EXIT FOR	119
2.6.4	FOR～TO～STEP～NEXT	120
2.6.5	IF～THEN～ELSE～END IF	121
2.6.6	SELECT CASE～END SELECT	122
2.7	ファイル・フォルダに関する関数・命令	123
2.7.1	参考：FIFO 特殊ファイル(名前付きパイプ)を使ったプロセス間通信	123
2.7.2	CHDIR	125
2.7.3	CLOSE	125
2.7.4	DIREXISTS	126
2.7.5	EOF	127
2.7.6	FILECOPY	128
2.7.7	FILEEXISTS	129
2.7.8	FILELIST\$	129
2.7.9	FILESTAT	130
2.7.10	FREEFILE	131
2.7.11	INCLUDE	132
2.7.12	INPUT	133
2.7.13	KILL	135
2.7.14	LINE INPUT	136
2.7.15	MKDIR	137
2.7.16	NAME	137
2.7.17	OPEN	138
2.7.18	PATH GET ABSPATH\$	139
2.7.19	PATH GET BASENAME\$	139
2.7.20	PATH GET DIRNAME\$	140
2.7.21	PATH JOIN\$	140
2.7.22	PATH SPLITEXT\$	141
2.7.23	PRINT, ?	142
2.7.24	RMDIR	143
2.7.25	SEEKGET	144
2.7.26	SEEKSET	144
2.7.27	STR_FREADALL\$	145
2.7.28	STR_FWRITEALL	145
2.7.29	WAIT FILE	146
2.8	サブルーチンに関する関数・命令	147
2.8.1	<ラベル名>:	149

2.8.2	CALL	149
2.8.3	ERROR ON / OFF	150
2.8.4	EXIT FUNCTION	151
2.8.5	EXIT SUB	151
2.8.6	FUNCTION～END FUNCTION	152
2.8.7	GOTO	154
2.8.8	KEY ON / OFF / STOP	155
2.8.9	ON END CALL	156
2.8.10	ON ERROR CALL	157
2.8.11	ON KEY CALL	158
2.8.12	ON TIME\$ CALL	159
2.8.13	ON TIMER CALL	160
2.8.14	SUB～END SUB	162
2.8.15	TIME\$ ON / OFF / STOP	164
2.8.16	TIMER ON / OFF / STOP	165
2.9	スレッドに関する関数・命令	166
2.9.1	スレッドについて	166
2.9.2	ATTACH THREAD	169
2.9.3	DETACH THREAD	170
2.9.4	DEFINE THREAD ～ END THREAD	172
2.9.5	LOCK	173
2.9.6	THREAD INFO	174
2.9.7	THREAD STATUS	174
2.9.8	UNLOCK	175
2.10	プロセス連携に関する関数・命令	176
2.10.1	プロセス連携について	176
2.10.2	プロセス連携の Tips	177
2.10.3	SHELL , SHELL RUN	179
2.10.4	SHELL CODE	179
2.10.5	SHELL OUTPUT\$	181
2.10.6	SHELL CALLOUT\$	182
2.10.7	SHELL OPEN	183
2.10.8	SHELL SYSTEM	184
2.10.9	SHELL SPAWN	185
2.10.10	SHELL WAITPID	186
2.10.11	SHELL KILLPID	187
第3章	サンプルプログラム	188
3.1	サンプルプログラム	188
第4章	エラーコードリファレンス	192
4.1	エラーコードリファレンス	192
第5章	アスキーコード一覧	194
第6章	コマンドの制限事項	195

6.1	トレース実行で使⽤できないコマンド	195
6.2	デバッグ実行中に停⽌できないコマンド	195
第 7 章	索引	196
第 8 章	重要な情報	199

第1章 はじめに

本ドキュメントは、AJANの各種コマンドおよび関数の説明を記載しております。標準コマンド以外のコマンド(GUIコマンド、IO制御コマンドなど)は、別マニュアルを用意しています。

本ドキュメントでは、説明で表現している表記として下記のように定義します。

- ・ コマンドの書式の説明において、[]内の引数は省略できます。
- ・ 文字の大小について

コマンドは大文字/小文字のどちらでも動作します。

変数名は大文字/小文字も同じものとして扱われます。




ファイルパス/ファイル名は大文字/小文字で区別されます。



本ドキュメント記載の、AJANはIoT用プログラミング言語です。
Interface Linux System上でのみ動作可能です。

第2章 標準コマンド

AJANで使用できる標準コマンドの使い方について記載します。

	制限事項については、「注意」に記載しています。
	使用例は動作を保証するものではありません。 実際の使い方は各種サンプルプログラムを参照してください。
	一部のコマンドは、使用状況によって、管理者権限(スーパーユーザー)で実行する必要があります。

2.1 コマンド一覧

コマンド名	機能
AJANメイン操作に関する関数・命令	
ASSERT	式を評価し、結果が偽の場合にエラーとします。
CLS	画面をクリアします。
END	プログラムを終了します。
ERF\$	最後にエラーが発生したAJANファイル名を返します。
ERL	最後に発生したエラーの行番号を返します。
ERM\$	最後に発生したエラーメッセージを返します。
ERN\$	最後にエラーが発生した命令または関数名を返します。
ERR	最後に発生したエラーコードを返します。
ERRMSG\$	最後に発生したエラーの詳細メッセージを返します。
ERRSUB	最後に発生したエラーの詳細コードを返します。
ERROR	エラー発生シミュレート、エラーコードのユーザー定義を行います。
OPTION ERROR	実行中にエラーが発生しても継続動作させるかどうかを設定します。
OPTION STOP	通常実行時、STOP命令を実行した際の挙動を指定します。
REM	プログラムにコメントを入れます。
SETERRTBL	独自にユーザー定義した、エラーコードとエラーメッセージを追加します。
SLEEP	指定の秒数の間だけ休止します。
STOP	プログラムの実行を一時中断します。
数値・文字列に関する関数・命令	
ABS	絶対値を返します。
ARRAY2CSV\$	2次元文字列配列を、CSV形式の文字列に変換します。
ASC	文字のキャラクターコードを返します。
ASCB	文字の先頭の1バイトデータを返します。
ATN	逆正接を返します。
BIN\$	10進数を2進数の文字列に変換します。
CDATETIME	引数値を、日付時刻型値に変換します。
CHR\$	指定したキャラクターコードが持つ文字列を返します。
CHRB\$	指定したバイトデータ値を持つバイトデータ文字列を返します。
CLOCK	経過時間の指標となる値を秒単位で得ます。
COS	余弦を返します。
CSV2ARRAY\$	CSV形式の文字列を、2次元文字列配列に変換します。
CVD	文字列を倍精度実数値データに変換します。
CVH	文字列を単精度整数値データに変換します。

コマンド名	機能
CVI	文字列を単精度整数値データに変換します。
CVL	文字列を倍精度整数値データに変換します。
CVS	文字列を単精度実数値データに変換します。
DATE\$	日付を返します。
EXP	底がeである指数関数の値を返します。
FIX	引数の整数部分を返します。
HEX\$	10進数を16進数の文字列に変換します。
INSTR	文字列中の指定文字列を検索し、位置を返します。
INSTRREV	文字列中の指定文字列を末尾から検索し、位置を返します。
INT	引数を超えない最大の整数値を返します。
ISDATETIME	引数を日付時刻型値に変換できるか調べます。
ISNUMERIC	文字列を数値に変換できるか調べます。
ISNAN	引数の値がNaN値か調べます。
ISINF	引数の値が無限大か調べます。
LEN	文字列の長さを文字数で返します。
LENB	文字列の長さをバイト数で返します。
LINSTR	文字列中の指定文字列を検索し、その数を返します。
LOG	自然対数を返します。
LSTRIP\$	文字列の先頭部分に対して、指定した文字集合を除去した文字列を返します。
MEMINFO	システムのメモリに関する情報を取得します。
MID\$	文字列の中から任意の長さの文字列を抜き出します。
MIDB\$	文字列の中から任意のバイト数分の文字列を抜き出します。
MKD\$	倍精度実数値データを数値の内部表現に対応した文字列に変換します。
MKH\$	単精度整数値データを数値の内部表現に対応した文字列に変換します。
MKI\$	単精度整数値データを数値の内部表現に対応した文字列に変換します。
MKL\$	倍精度整数値データを数値の内部表現に対応した文字列に変換します。
MKS\$	単精度実数値データを数値の内部表現に対応した文字列に変換します。
RANDOMIZE	新しい乱数系列を設定します。
REPLACE\$	文字列中の指定文字を別の文字列に置き換えます。
RND	0以上1未満の乱数を返します。
ROUND	指定数値を指定位置で四捨五入します。
RSTRIP\$	文字列の末尾部分に対して、指定した文字集合を除去した文字列を返します。
SIN	正弦を返します。
SPLIT\$	区切り文字列から文字列を区切り、1次元配列の文字列を生成します。
SQR	平方根を返します。
STR\$	数値を文字列に変換します。
STRDEL\$	文字列の任意の位置から指定した文字長の文字列を削除します。
STRINS\$	文字列の任意の位置に指定した文字列を挿入します。
STRIP\$	文字列の両端部分に対して、指定した文字集合を除去した文字列を返します。
TAN	正接を返します。
TIMES	時刻を返します。
TRIM\$	文字列の前後から全半角スペースを削除します。
VAL	数値表記の文字列を、実際の数値に変換します。
WEEK	指定日の曜日を返します。
型の宣言や変換に関する関数・命令	
BOOL	論理型変数を宣言します。
CONST	定数を宣言します。
CONST BOOL	論理型定数を宣言します。
CONST DATETIME	日付時刻型定数を宣言します。
DATETIME	日付時刻型変数を宣言します。
DEFINE STRUCT ~ END STRUCT	構造体型の定義を宣言します。
ENUM~END ENUM	列挙型定数を宣言します。

コマンド名	機能
LOCAL	ローカル変数を宣言します。
LOCAL BOOL	論理型ローカル変数を宣言します。
LOCAL CONST	ローカル定数を宣言します。
LOCAL CONST BOOL	論理型ローカル定数を宣言します。
LOCAL CONST DATETIME	日付時刻型ローカル定数を宣言します。
LOCAL DATETIME	日付時刻型ローカル変数を宣言します。
LOCAL ENUM~END ENUM	列挙型ローカル定数を宣言します。
LOCAL STRUCT	構造体型ローカル変数を宣言します。
STRUCT	構造体型変数を宣言します。
VARTYPE	変数の型を調べます。
配列に関する関数・命令	
ARRAY2DICT	2次元の文字列配列から連想配列にセットします。
CDIM	配列の次元数を返します。
CLEAR_DICT	連想配列の内容をクリアします。
DEL_DICT_KEY	連想配列に対して、指定したキーに紐付けされたデータを削除します。
DICT	連想配列変数を宣言します。
DICT BOOL	論理型連想配列変数を宣言します。
DICT DATETIME	日付時刻型連想配列変数を宣言します。
DICT STRUCT	構造体型連想配列変数を宣言します。
DICT2ARRAY\$	連想配列から2次元の文字列配列として得ます。
DIM	配列変数を宣言します。
GET_DICT_KEYS\$	連想配列のキー一覧を文字列配列として得ます。
HAS_DICT_KEY	連想配列に対して、指定のキーが存在するか否かを得ます。
LDICT	連想配列に対して登録されているキーの総数を返します。
LDIM	配列の要素数を返します。
LIST	可変長配列変数を宣言します。
LIST BOOL	論理型可変長配列変数を宣言します。
LIST DATETIME	日付時刻型可変長配列変数を宣言します。
LIST DICT	連想配列の可変長配列変数を宣言します。
LIST DICT BOOL	論理型連想配列の可変長配列変数を宣言します。
LIST DICT DATETIME	日付時刻型連想配列の可変長配列変数を宣言します。
LIST DICT STRUCT	構造体型連想配列の可変長配列変数を宣言します。
LIST STRUCT	構造体型の可変長配列変数を宣言します。
LOCAL DICT	連想配列のローカル変数を宣言します。
LOCAL DICT BOOL	論理型連想配列のローカル変数を宣言します。
LOCAL DICT DATETIME	日付時刻型連想配列のローカル変数を宣言します。
LOCAL DICT STRUCT	構造体型連想配列のローカル変数を宣言します。
LOCAL LIST	可変長配列のローカル変数を宣言します。
LOCAL LIST BOOL	論理型可変長配列のローカル変数を宣言します。
LOCAL LIST DATETIME	日付時刻型可変長配列のローカル変数を宣言します。
LOCAL LIST DICT	連想配列のローカル可変長配列変数を宣言します。
LOCAL LIST DICT BOOL	論理型連想配列のローカル可変長配列変数を宣言します。
LOCAL LIST DICT DATETIME	日付時刻型連想配列のローカル可変長配列変数を宣言します。
LOCAL LIST DICT STRUCT	構造体型連想配列のローカル可変長配列変数を宣言します。
LOCAL LIST STRUCT	構造体型可変長配列のローカル変数を宣言します。
ONEDIM INSERT	1次元配列の指定した位置に、変数(配列可)を挿入します。
ONEDIM REMOVE	1次元配列の指定した位置から、指定した要素を削除します。
REDIM	可変長配列の次元数および各次元の要素数を任意に変更します。
TWODIM INSERT	2次元配列の指定した行／列位置に、1行／列分の配列変数を挿入します。
TWODIM REMOVE	2次元配列の指定した位置から、指定した行／列数を削除します。
UBOUND	配列の指定した次元の添字最大値を得ます。
繰り返し・条件分岐に関する関数・命令	

AJAN標準コマンドリファレンス

コマンド名	機能
DO WHILE～LOOP	DOからLOOPまでの区間中にある一連の命令を、指定条件を満たす(TRUE)間、繰り返して実行します。
EXIT DO	DO～LOOP文の繰り返しから脱出します。
EXIT FOR	FOR～NEXT文の繰り返しから脱出します。
FOR～TO～STEP～NEXT	FORからNEXTまでの区間中にある一連の命令を、繰り返して実行します。
IF～THEN～ELSE～END IF	式の値の条件判定を行います。
SELECT CASE ～ END SELECT	式の値と続くCASE文に従い、処理を分岐します。
ファイル・フォルダに関する関数・命令	
CHDIR	現在の作業フォルダを変更します。
CLOSE	ファイルをクローズします。
DIREXISTS	フォルダが存在するか確認し、結果を返します。
EOF	ファイルの終了コードを調べます。
FILECOPY	ファイルをコピーします。
FILEEXISTS	ファイルが存在するか確認し、結果を返します。
FILELIST\$	指定したパターンにマッチするパス名を文字列配列形式で得ます。
FILESTAT	指定したファイルパス／ファイル番号のファイル情報を得ます。
FREEFILE	使用可能なファイル番号を得ます。
INCLUDE	指定した外部ファイルのプログラムを参照できるように追加します。
INPUT	キーボードやファイルから入力されたデータを変数に代入します。
KILL	指定したファイルまたはディレクトリを削除します
LINE INPUT	キーボードやファイルから入力された1行単位のデータを文字型変数に代入します。
MKDIR	新しいフォルダを作成します。
NAME	ファイル/フォルダの名前を変更します。
OPEN	ファイルをオープンします。
PATH GET ABSPATH\$	相対パス形式の文字列から、絶対パスに変換します。
PATH GET BASENAME\$	パス形式の文字列から、末尾のファイル名に相当する文字部分を取り出します
PATH GET DIRNAME\$	パス形式の文字列から、ディレクトリ名に相当する文字部分を取り出します
PATH JOIN\$	1つあるいは複数のパス要素を結合して、パス形式の文字列を作ります。
PATH SPLITEXT\$	パス形式の文字列から、ファイルの拡張子とそれ以外の文字列の配列に分離します。
PRINT, ?	文字列や数値等のデータを画面、またはファイルに出力します。
RMDIR	既存のフォルダを削除します。
SEEKGET	ファイルの読み書きする現在位置を取得します。
SEEKSET	ファイルから次に読み書きする位置を設定します。
STR_FREADALL\$	指定したファイルから全データをバイナリ文字列に読み取ります。
STR_FWRITEALL	バイナリ文字列を指定したファイルに書き込みます。
WAIT FILE	指定したファイル番号の読み書きイベントを待ちます。
サブルーチンに関する関数・命令	
<ラベル名>:	GOTOによるジャンプ先のラベルを設定します。
CALL	SUB～END SUBで定義したサブルーチンを呼び出します。
ERROR ON / OFF	エラー発生による割り込みの許可、禁止を指定します。
EXIT FUNCTION	ユーザ定義関数から脱出します。
EXIT SUB	引数を使用するサブルーチンから脱出します。
FUNCTION ～ END FUNCTION	ユーザ定義関数を定義します。
GOTO	指定したラベル名へ移動します。
KEY ON / OFF / STOP	ファンクションキーによる割り込みの許可、禁止、保留を指定します。
ON END CALL	プログラム終了直前に、呼び出されるサブルーチンを定義します。
ON ERROR CALL	エラーが発生した時に、呼び出されるサブルーチンを定義します。
ON KEY CALL	ファンクションキーが押された時に、呼び出されるサブルーチンを定義します。

コマンド名	機能
ON TIMES\$ CALL	指定時刻に、呼び出されるサブルーチンを定義します。
ON TIMER CALL	指定時間間隔に、呼び出されるサブルーチンを定義します。
SUB～END SUB	サブルーチンを定義します。
TIMES\$ ON / OFF / STOP	TIMES\$割り込みの許可、禁止、保留を指定します。
TIMER ON / OFF / STOP	TIMER割り込みの許可、禁止、保留を指定します。
スレッドに関する関数・命令	
ATTACH THREAD	スレッドを作成します。
DETACH THREAD	スレッドを終了します。
DEFINE THREAD ～ END THREAD	スレッドを定義します。
LOCK	マルチスレッド／割り込み時の排他処理用に、ロックをかけます。
THREAD INFO	スレッドに関する情報を取得します。
THREAD STATUS	スレッドが動作しているか確認します。
UNLOCK	マルチスレッド／割り込み時の排他処理用に、ロックを解除します。
プロセス連携に関する関数・命令	
SHELL , SHELL RUN	外部プログラムを呼び出します。
SHELL CODE	SHELL RUNで実行した、外部プログラムの戻り値を取得します。
SHELL OUTPUT\$	SHELL RUNで実行した、外部プログラムの出力結果文字列を取得します。
SHELL CALLOUT\$	外部プログラムをシェル経由で呼び出し、その出力結果文字列を取得します。
SHELL OPEN	指定したファイルを、関連付けに応じてデフォルトアプリケーションで開きます。
SHELL SYSTEM	外部プログラムをシェル経由で呼び出し。その戻り値を得ます。
SHELL SPAWN	外部プログラムを非同期で呼び出し、そのプロセスID値を得ます。
SHELL WAITPID	指定したプロセスIDの状態変化を待ち、その実行結果を得ます。
SHELL KILLPID	指定したプロセスIDに対してシグナルを送ります。

2.2 AJANメイン操作に関する関数・命令

2.2.1 ASSERT

命令			
機 能	式を評価し、結果が偽の場合にエラーとします。		
書 式	ASSERT <①式> , <②エラー時メッセージ>		
パラ メータ	①	<式>	式
	評価式を記述します。		
パラ メータ	②	<エラー時メッセージ>	文字列
	評価式の結果が偽の場合、エラーとしますが、その際に表示するメッセージを指定します。 文字列形式で渡してください。		
注 意	ASSERTでのエラーでは、ON ERROR CALLで設定したサブルーチンは呼び出されません。 必ず停止します。		
使用例	ASSERT A=1, "Aが1でない" 上の例を実行した時、Aが1の場合、特に問題なく動作します。 Aが1以外の場合、エラーが発生してプログラムの実行が止まります。		

2.2.2 CLS

命令			
機 能	画面をクリアします。		
書 式	CLS [<①モード>]		
パラ メータ	①	<モード>	数値
	画面クリア時のモードを指定します。 1：コンソールをクリア(省略時)		
備 考	クリアは Linuxのコマンドで clear相当が実行されます。		
使用例1	CLS <モード>を省略すると、1が指定されたものとして動作します。 入力された文字列およびコマンドの実行によって出力された文字列をクリアします。		
使用例2	CLS 1 <モード> 1を指定します。 入力された文字列およびコマンドの実行によって出力された文字列をクリアします。		

2.2.3 END

命令	
機 能	プログラムを終了します。
書 式	END
備 考	<ul style="list-style-type: none"> ・メインスレッドから呼ばれると、全てのスレッドを終了させます。 (スレッドの説明については、「2.9 スレッドに関する関数・命令」を参照してください。) ・ワーカースレッドから呼ばれると、自身のスレッドのみ終了させます。
使用例	<pre>PRINT "Hello" END PRINT "AJAN"</pre> <p>「Hello」だけ表示されてプログラムを終了します。「AJAN」は表示されません。</p>

2.2.4 ERF\$

関数			
機 能	最後にエラーが発生したAJANファイル名を返します。		
書 式	<(戻り値)エラー時ファイル名> = ERF\$		
戻り値	戻り値	<エラー時ファイル名>	文字列
	最後にエラーが発生したAJANファイル名を、文字列で得られます。		
備 考	<ul style="list-style-type: none">・エラーコードリファレンス(→「4.1 エラーコードリファレンス」)にエラーメッセージの一覧があります。・スレッドを使用する場合、この値はスレッド毎に保持されます。・INCLUDE されたAJANファイル内でエラーが発生すると、そのファイル名となります。 エラー時 行番号を知りたい場合は、ERL関数を使用してください。		
注意点	得られるファイル名は、絶対パスでなくファイル名のみです。		
使用例	SMP\$ = ERF\$ エラーが最後に発生した命令または関数名を変数SMP\$に代入します。		

2.2.5 ERL

関数		
機 能	最後にエラーが発生した行番号を返します。	
書 式	<(戻り値)エラー行番号> = ERL	
戻り値	戻り値	数値
	<エラー行番号> 最後にエラーが発生した行番号が得られます。	
備 考	<ul style="list-style-type: none"> ・スレッドを使用する場合、この値はスレッド毎に保持されます。 ・INCLUDE されたAJANファイル内でエラーが発生すると、そのファイルの行番号となります。 エラー時 ファイル名を知りたい場合は、ERF\$関数を使用してください。	
使用例	C = ERL 最後にエラーが発生した行番号を、変数Cに代入します。	

2.2.6 ERM\$

関数		
機 能	最後にエラーが発生したエラーメッセージを返します。	
書 式	<(戻り値)エラーメッセージ> = ERM\$	
戻り値	戻り値	文字列
	<エラーメッセージ> 最後にエラーが発生したエラーメッセージが得られます。	
備 考	<ul style="list-style-type: none"> ・エラーコードリファレンス(→「4.1 エラーコードリファレンス」)にエラーメッセージの一覧があります。 ・ERM\$で得られる、エラーメッセージの内容は、エラーコード(ERR関数で取得)に対応します。 ・スレッドを使用する場合、この値はスレッド毎に保持されます。 	
使用例	SMP\$ = ERM\$ 最後に発生したエラーメッセージを変数SMP\$に代入します。	

2.2.7 ERN\$

関数		
機 能	最後にエラーが発生した命令または関数名を返します。	
書 式	<(戻り値)エラーコマンド名> = ERN\$	
戻り値	戻り値	<エラーコマンド名>
	最後にエラーが発生した命令または関数名を、文字列で得られます。	
備 考	<ul style="list-style-type: none"> エラーコードリファレンス(→「4.1 エラーコードリファレンス」)にエラーメッセージの一覧があります。 スレッドを使用する場合、この値はスレッド毎に保持されます。 	
注意点	制御文や代入式の代入中、あるいは 評価式を評価中にエラーが発生した場合、ERN\$には、その前に実行された命令または関数名が記録されます。	
使用例	SMP\$ = ERN\$	
	エラーが最後に発生した命令または関数名を変数SMP\$に代入します。	

2.2.8 ERR

関数		
機 能	最後にエラーが発生したエラーコードを返します。	
書 式	<(戻り値)エラーコード> = ERR	
戻り値	戻り値	<エラーコード>
	最後にエラーが発生したエラーコードが得られます。	
備 考	<ul style="list-style-type: none"> エラーコードリファレンス(→「4.1 エラーコードリファレンス」)にエラーコードの一覧があります。 スレッドを使用する場合、この値はスレッド毎に保持されます。 	
使用例	C = ERR	
	最後に発生したエラーコードを変数Cに代入します。	

2.2.9 ERRMSG\$

関数		
機 能	最後に発生したエラーの詳細メッセージを返します。 特に、ドライバや依存ライブラリ呼び出し時のエラー文字列が入ります。	
書 式	<(戻り値)エラー詳細メッセージ> = ERRMSG\$	
戻り値	戻り値	<エラー詳細メッセージ> 文字列 最後にエラーが発生したエラーの詳細メッセージが得られます。
備 考	<ul style="list-style-type: none"> エラーコードリファレンス(→「4.1 エラーコードリファレンス」)にエラーメッセージの一覧があります。 スレッドを使用する場合、この値はスレッド毎に保持されます。 	
使用例	SMP\$ = ERRMSG\$ 最後に発生したエラーの詳細メッセージを変数SMP\$に代入します。	

2.2.10 ERRSUB

関数		
機 能	最後に発生したエラーの詳細コードを返します。 特に、ドライバや依存ライブラリ呼び出し時のエラーコードが入ります。	
書 式	<(戻り値)エラー詳細コード> = ERRSUB	
戻り値	戻り値	<エラー詳細コード> 数値 最後にエラーが発生したエラーの詳細コードが得られます。
備 考	<ul style="list-style-type: none"> エラーコードリファレンス(→「4.1 エラーコードリファレンス」)にエラーコードの一覧があります。 スレッドを使用する場合、この値はスレッド毎に保持されます。 	
使用例	C = ERRSUB 最後に発生したエラーの詳細コードを変数Cに代入します。	

2.2.11 ERROR

命令		
機 能	エラー発生シミュレート、エラーコードのユーザー定義を行います。 エラーを発生させたとき、ON ERROR CALLを使用することで、独自のエラー処理を行うことができます。	
書 式1	ERROR <①エラーコード>	
書 式2	ERROR <①エラーコード>, <③詳細エラーメッセージ>, <④エラー時コマンド名>	
書 式3	ERROR <①エラーコード>, <②詳細エラーコード>, <③詳細エラーメッセージ>, <④エラー時コマンド名>	
パラメータ	①	<div> <div><エラーコード></div> <div>数値</div> </div> 発生させるエラーコードを指定します。 エラーコードには任意の値を指定できます。 エラー発生時、ここで設定した値は、ERR関数で読み取れます。
	②	<div> <div><詳細エラーコード></div> <div>数値</div> </div> 発生させるエラーの詳細エラーコードを指定します。 任意の値を指定できます。 エラー発生時、ここで設定した値は、ERRSUB関数で読み取れます。
	③	<div> <div><詳細エラーメッセージ></div> <div>文字列</div> </div> 発生させるエラーの詳細エラーメッセージを指定します。 任意の文字列を指定できます。 エラー発生時、ここで設定した値は、ERRMSG\$関数で読み取れます。
	④	<div> <div><エラー時コマンド名></div> <div>文字列</div> </div> 発生させるエラーのエラー時コマンド名を指定します。 任意の文字列を指定できます。 エラー発生時、ここで設定した値は、ERN\$関数で読み取れます。
備 考	<ul style="list-style-type: none"> エラーコードは、エラーコードリファレンス(→「4.1 エラーコードリファレンス」)にあるエラーコードか「SETERRTBL (P. 24)」命令で定義したエラーコードを、指定してください。 指定外のエラーコードを指定した場合、ERM\$関数で得られる文字列は不定です。 エラーコードに「0」を与えたとき、エラーは発生せず、コマンド実行は正常終了します。 結果、ERL関数、ERN\$関数、ERRMSG\$関数、ERRSUB関数で読み込む内容がクリアされます。 (ERR関数で得る値は、設定した通りの 0です) 	
使用例1	ERROR &h01000006 エラーコード &h01000006 のエラーを発生させます。	
使用例2	<pre> SUB ONERR(E_R, E_M\$, E_L) IF E_R = &h01000006 THEN ... END IF END SUB ON ERROR CALL ONERR ERROR ON ... ERROR &h01000006 ... END 「ERROR &h01000006」を実行した時点でエラーコード &h01000006 のエラーが発生し、 ON ERROR CALL記述に従ってサブルーチンONERRが実行されます。ON ERROR CALL により </pre>	

	実行されるサブルーチン内でエラーコードの取得・判定を行うことで、独自のエラーとエラー発生時の処理を定義することができます。
使用例3	<pre>SUB TEST (ARG1) IF ARG1 < 0 THEN ERROR &h01000004, "ARG1値は0以上を与えてください", "TEST" END IF ...</pre> <p>ユーザー定義のTESTサブルーチンで、引数ARG1が0未満だった場合に、エラーを発生させます。</p> <p>このように記述する事で、自作の処理で意図しない値や状況になった時に、利用者にエラー詳細を通知する事が可能となります。</p>

2.2.12 OPTION ERROR

命令			
機能	実行中にエラーが発生しても継続動作させるかどうかを設定します。		
書式	OPTION ERROR <①設定値>		
パラメータ	①	<設定値>	数値
	設定値を指定します。 0: 継続動作を無効に設定します。 1: 継続動作を有効に設定します。		
注意点	<ul style="list-style-type: none"> 継続動作を有効に設定するとエラーメッセージは表示されなくなります。エラーが発生したかどうかはERR(→2.2.8ERR)関数やERM\$(→2.2.6ERM\$)関数で確認することができます。 確認した後、エラー情報をクリアしたい時は、ERROR関数を使って「ERROR 0」のように呼び出してください。 本命令は有効/無効に限らず、プログラム中で1回だけ指定することができます。 ASSERT (→2.2.1ASSERT) 命令は本命令の有効/無効の影響を受けません。 本命令は実行中のエラーにのみ対応します。コンパイル時のエラーは対象外です。 OPTION ERRORで継続動作を有効にした場合、ON ERROR CALL(→2.8.10 ON ERROR CALL)は無効となります。 		
使用例	OPTION ERROR 1 エラーが発生しても継続して動作します。		

2.2.13 OPTION STOP

命令											
機 能	通常実行時、STOP命令を実行した際の挙動を指定します。										
書 式	OPTION STOP <①設定値>										
パラ メータ	①	<設定値>	数値								
	通常実行時、STOP命令を実行した際の挙動を、以下から指定します。										
	<table><tr><th>設定値</th><th>動作</th></tr><tr><td>1</td><td>STOP命令で何もせずに、そのまま次行を続行します。 (OPTION STOPを指定しない場合、この動作が採用されます)</td></tr><tr><td>2</td><td>STOP命令で「break in <行番号>」と表示して実行を終了します。</td></tr><tr><td>3</td><td>STOP命令で「break in <行番号>」と表示した後、「Press any key to continue ...」と表示し、キー入力押されるまで待機します。何かキーが押されると次行を続行します。</td></tr></table>			設定値	動作	1	STOP命令で何もせずに、そのまま次行を続行します。 (OPTION STOPを指定しない場合、この動作が採用されます)	2	STOP命令で「break in <行番号>」と表示して実行を終了します。	3	STOP命令で「break in <行番号>」と表示した後、「Press any key to continue ...」と表示し、キー入力押されるまで待機します。何かキーが押されると次行を続行します。
	設定値	動作									
1	STOP命令で何もせずに、そのまま次行を続行します。 (OPTION STOPを指定しない場合、この動作が採用されます)										
2	STOP命令で「break in <行番号>」と表示して実行を終了します。										
3	STOP命令で「break in <行番号>」と表示した後、「Press any key to continue ...」と表示し、キー入力押されるまで待機します。何かキーが押されると次行を続行します。										
注意点	<ul style="list-style-type: none">・ 本命令は有効/無効に限らず、プログラム中で1回だけ指定することができます。・ 本命令は通常実行中のSTOP命令の動作にのみ対応します。 デバッグ実行中のSTOP命令では、実行が中断され、AJAN IDEでステップ実行や再開などのデバッグ操作が可能となります。										
使用例1	OPTION STOP 1 ? 123 STOP ? 456 通常実行時、STOPで停止せず、次行の「? 456」の処理を続行します。										
使用例2	OPTION STOP 2 ? 123 STOP ? 456 通常実行時、STOPで実行を終了します。										
使用例3	OPTION STOP 3 ? 123 STOP ? 456 通常実行時、STOPで一時停止します。何かキーが押されると次行の「? 456」の処理を続行します。										

2.2.14 REM

命令			
機能	プログラムにコメントを入れます。		
書式1	REM <①コメント>		
書式2	' <①コメント>		
	' (シングルクォーテーション) で、REM命令の代わりとなります。		
パラメータ	①	<コメント>	文字列
	一行コメントを入力します。		
使用例1	REM SAMPLE 1		
	文字列 「SAMPLE 1」 がコメント扱いになります。		
使用例2	' SAMPLE 2		
	文字列 「SAMPLE 2」 がコメント扱いになります。		

2. 2. 15 SETERRTBL

命令					
機 能	独自にユーザー定義した、エラーコードとエラーメッセージを追加します。				
書 式	SETERRTBL <①エラー設定ファイル名>				
パラメータ	<table><tr><td>①</td><td><エラー設定ファイル名></td><td>文字列</td></tr></table> <p>独自にユーザー定義する、エラーコードとエラーメッセージを記述したCSV形式のファイル名を指定します。</p> <p>ファイルの内容は、以下の書式に従ってください。</p> <ul style="list-style-type: none">・1列目：エラーコード。2列目：エラーメッセージ・テキストは、UTF-8形式の文字列	①	<エラー設定ファイル名>	文字列	
①	<エラー設定ファイル名>	文字列			
注 意	<ul style="list-style-type: none">・ユーザー定義するエラーコードは、以下の予約範囲外の数値を利用ください。 &h0000 0000～&h0000 0085 および &h0100 0000～&h0100 FFFF・SETERRTBLを複数回呼び出す時、設定済みのエラーコードを重複指定してはいけません。				
使用例	<p>’ ユーザー定義されたエラーコードとメッセージのファイルを設定します。</p> <p>SETERRTBL "custom_err.csv"</p> <p>↑</p> <p>custom_err.csvのファイル内容は、以下のものとします。</p> <table><tr><td>10000, テスト</td></tr><tr><td>10001, インタフェース</td></tr><tr><td>10002, AJAN</td></tr></table> <p>’ ユーザー定義された 10001のエラーコードを使ってエラーを発生させます。</p> <p>’ ERR関数で得られる値は10001、ERM\$関数で得られる文字列は「インタフェース」です。</p> <p>ERROR 10001</p>		10000, テスト	10001, インタフェース	10002, AJAN
10000, テスト					
10001, インタフェース					
10002, AJAN					

2.2.16 SLEEP

命令			
機 能	指定の秒数の間だけ、現在のスレッドの実行を休止します。		
書 式	SLEEP <①休止時間(秒)>		
パラ メータ	①	<休止時間(秒)>	数値
注 意	<p>休止時間を0以上の倍精度実数で指定します。</p> <p>・ 休止時間の指定は、小数点以下を含める事が可能ですが、休止時間の粒度および正確性は、OS(システム)に依存する為、保証できません。 実機にて、実際に計測する事をお勧めします。</p>		
使用例1	<p>SLEEP 5</p> <p>5 秒間プログラムの実行を休止します。</p>		
使用例2	<p>SLEEP 0.001</p> <p>0.001秒間プログラムの実行を休止します。</p>		

2.2.17 STOP

命令	
機 能	<p>デバッグ実行時、プログラムの実行を一時中断します。</p> <p>メインスレッド、ワーカースレッドに関わらず全てのプログラムが一時中断されます。</p> <p>通常実行時は、OPTION STOP命令の指定動作に依存します。</p>
書 式	STOP
使用例	<pre>PRINT "Hello." STOP PRINT "World" -----</pre> <p>実行結果</p> <pre>Hello.</pre> <p>デバッグ実行時、「Hello. 」を表示後にSTOPコマンドが実行された所でプログラムは一時停止状態になります。再開するまで次行以降の処理は実行されません。</p>

2.3 数値・文字列に関する関数・命令

2.3.1 ABS

関数			
機 能	絶対値を返します。		
書 式	<(戻り値)絶対値> = ABS(<①数式>)		
戻り値	戻り値	<絶対値>	数値
	絶対値が得られます。		
パラ メータ	①	<数式>	数値
	絶対値を求める数式を指定します。		
使用例	NUM = ABS(-3)		
	変数NUMに3（-3の絶対値）を代入します。		

2.3.2 ARRAY2CSV\$

関数		
機 能	2次元文字列配列を、CSV形式の文字列に変換します。	
書 式	<(戻り値) CSV形式文字列> = ARRAY2CSV\$("<①2次元文字列配列>" [, "<②デリミタ>" [, "<③囲い文字>" [, <④行列スイッチ>]])	
戻り値	戻り値	<CSV形式文字列> 文字列 CSV変換された文字列。
パラ メータ	①	<2次元文字列配列> 配列 変換対象となる、2次元文字列配列。
	②	<デリミタ> 文字列 1項目を区切る記号となる文字を指定します。 省略時、「,(カンマ)」として扱います。
	③	<囲い文字> 文字列 項目を囲う文字を指定します。これを指定すると、囲い文字の範囲中に、デリミタ文字や改行文字が含まれる事が可能です。 省略時、囲い文字はありません。改行文字が格納された文字列配列をCSV形式に変換する場合、「"(ダブルクォーテーション)"」を指定すると良いでしょう。
	④	<行列スイッチ> 真偽値 TRUE：列×行の2次元文字列配列として変換します。(省略時、デフォルト) FALSE：行×列の2次元文字列配列として変換します。
	備考	
使用例	改行に判定する文字は CHR\$(10) です。	
	<pre> S\$ = ''' col01, col02, col03, col04, col05 0-0, 0-1, 0-2, 0-3, 0-4 1-0, 1-1, 1-2, 1-3, 1-4 2-0, 2-1, 2-2, 2-3, 2-4 ''' LIST ARY\$ ARY\$ = CSV2ARRAY\$(S\$, ",", "''", FALSE) S2\$ = ARRAY2CSV\$(ARY\$, ",", "''", FALSE) CSV形式の文字列(S\$)を、CSV2ARRAY\$で、2次元文字列配列に変換します。 次に、ARRAY2CSV\$で、再びCSV形式の文字列に戻します。 </pre>	

2.3.3 ASC

関数			
機 能	文字のキャラクターコードを返します。		
書 式	<(戻り値)キャラクターコード> = ASC(<①文字> [, <②オプション>])		
戻り値	戻り値	<キャラクターコード>	数値
	文字のキャラクターコードが得られます。		
パラ メータ	①	<文字>	文字列
	キャラクターコードを求める文字を指定します。 文字列を入力した場合は、先頭の1文字だけを判定します。		
	②	<オプション>	文字列
	2022/3 より使用可能です。 "array" を指定すると、文字列を1文字ずつ分解してキャラクターコードの 1次元の数値配列を得ます。 省略すると、先頭の1文字だけのキャラクターコードを得ます。		
備 考	<ul style="list-style-type: none"> ・キャラクターコードを求める文字は、ユニコード文字(UCS-4)をサポートしています。 ・キャラクターコードは10進数で返します。 		
使用例1	C = ASC("Z") 変数Cに90 ("Z"のキャラクターコード) を代入します。		
使用例2	C = ASC("歩") 変数Cに&h6b69("歩"のキャラクターコード)を代入します。		

2.3.4 ASCB

関数			
機 能	文字の先頭の1バイトデータを返します。		
書 式	<(戻り値)キャラクターコード> = ASCB(<①文字> [, <②オプション>])		
戻り値	戻り値	<キャラクターコード>	数値
	文字の1バイトデータのキャラクターコードが得られます。		
パラ メータ	①	<文字>	文字列
	先頭の1バイトデータを求める文字を指定します。 文字列を入力した場合でも、先頭の1バイトだけを判定します。		
	②	<オプション>	文字列
	2022/3 より使用可能です。 "array" を指定すると、文字列を1バイトずつ分解してキャラクターコードの 1次元の数値配列を得ます。 省略すると、先頭の1バイトだけのキャラクターコードを得ます。		
備 考	<ul style="list-style-type: none"> ・ この関数は、文字列のデータをバイトデータとして扱う為に用意されています。 ・ 文字列中、先頭1バイト目を数値にして返します。(0～255) ・ 「第5章 アスキーコード一覧」に載っていないコードは、画面表示が文字化けする恐れがあります。 		
使用例	C = ASCB(CHRB\$(&h80)) 変数Cに&h80 (CHRB\$でバイトデータ化された値) を代入します。		

2.3.5 ATN

関数			
機 能	逆正接を返します。		
書 式	<(戻り値) 逆正接値> = ATN(<①数式>)		
戻り値	戻り値	<逆正接値>	数値
	逆正接値が得られます。		
パラ メータ	①	<数式>	数値
	逆正接を求める数式を指定します。		
使用例	NUM = ATN(1)		
	変数NUMに $\tan^{-1}(1) \div 0.785398163397448$ を代入します。		

2.3.6 BIN\$

関数			
機 能	10進数を2進数の文字列に変換します。		
書 式	<(戻り値) 2進数文字列> = BIN\$(<①数値> [, <②桁数>])		
戻り値	戻り値	<2進数文字列>	文字列
	数値を2進数の文字列に変換した値が得られます。		
パラ メータ	①	<数値>	数値
	2進数文字列を求める数値を指定します。 指定された数値が整数でない場合は、四捨五入してから評価されます。		
	②	<桁数>	数値
	1以上を指定すると、表示する桁数を強制できます。 結果は、桁数に従い、桁数不足時に"0"が左に付加されます。 指定できる桁数の最大値は、与える数値の型に依存します。例えば、単精度整数は32、倍精度整数は64です。		
備 考	倍精度整数の範囲（→基本マニュアルの「整数型定数」）を超える値の場合は"0"が返ります。		
使用例1	SMP\$ = BIN\$(255)		
	変数SMP\$に"11111111"（255の2進数表示）を代入します。		
使用例2	PRINT BIN\$(&H12) 10010 PRINT BIN\$(&H12, 8) 00010010		
	桁数指定機能を使った場合と使わない場合の例です。		

2.3.7 CDATETIME

関数			
機 能	引数を日付時刻型値に変換します。		
書 式	<(戻り値) 日付時刻値> = CDATETIME(<①引数>)		
戻り値	戻り値	<日付時刻値>	日付時刻
	日付時刻型値を得ます。		
パラメータ	①	<引数>	数値／文字列
	<p>日付時刻型値に変換する対象となる文字列を指定します。 文字列は以下の書式で指定してください。 日付: "2017/01/01" yyyy/mm/dd(年/月/日) 時刻: "12:34:56" hh:mm:ss(時:分:秒) 日付時刻: "2017/01/01 12:34:56" yyyy/mm/dd hh:mm:ss(年/月/日 時:分:秒)</p> <p>数値を引数に与えると、日付時刻型として計算します。</p>		
備 考	<ul style="list-style-type: none"> 2022/3より、時刻にミリ秒など秒以下の値を指定できるようになりました。 例えば、「12:34:56.789」のような形式で記述できます。 ただし、ミリ秒以下の値を文字列化したい場合は、「FORMATDATETIME\$」を使用して ください。「PRINT」や「STR\$」は、秒までの書式となります。 2022/3より、ISO8601形式の文字列を指定できるようになりました。 また、区切り記号に「T」でなく「」（空白）を指定する事も可能です。(RFC3339 で サポートされる形式に相当します) 		
使用例1	DATETIME A A = CDATETIME("2017/01/01") 年月日を指定し日付時刻型変数のAへ代入します。時刻は00:00:00とされます。		
使用例2	DATETIME A A = CDATETIME(DATE\$+" "+TIME\$) 現在の日付と時刻を取得しAへ代入します。		
使用例3	DATETIME A A = CDATETIME(DATE\$ + " " + TIME\$) ? A - CDATETIME("01:00:00") 現在から1時間前を演算して表示します。		
使用例4	DATETIME A A = CDATETIME(DATE\$) B = CDATETIME("2017/1/1") C% = A - B 変数 C%に、2017年1月1日から現在の日付までの日数を代入します。		

2.3.8 CHR\$

関数			
機 能	指定したキャラクターコードに対応する文字を返します。		
書 式	<(戻り値) 文字> = CHR\$(<①キャラクターコード>)		
戻り値	戻り値	<文字>	文字列
	指定したキャラクターコードが持つ文字を返します。		
パラ メータ	①	<キャラクターコード>	数値
	文字を求めるキャラクターコードを指定します。 ※UCS-4		
備 考	<ul style="list-style-type: none"> ・キャラクターコードは、ユニコード文字(UCS-4)の範囲を指定してください。 ・「第5章 アスキーコード一覧」に載っていないキャラクターコードを指定すると、画面表示が文字化けする恐れがあります。 ・Ver. 1.00より、配列で与えると、文字を連結して返すようになりました。 		
使用例1	SMP\$ = CHR\$(65) 変数SMP\$へ“A” (キャラクターコード 65 に該当する文字) を代入します。		
使用例2	SMP\$ = CHR\$(&h6B69) 変数SMP\$へ“歩” (ユニコード値 &h6B69 に該当する文字) を代入します。		
使用例3	SMP\$ = CHR\$([&H40; &H41; &H42; &H43]) 変数SMP\$へ “@ABC” を代入します。		

2.3.9 CHRB\$

関数			
機 能	指定したバイトデータ値に対応するバイトデータ文字列を返します。		
書 式	<(戻り値) バイト文字> = CHRB\$(<①バイトデータ>)		
戻り値	戻り値	<バイト文字>	文字列
	指定したバイトデータ値を持つバイト文字を返します。		
パラ メータ	①	<バイトコード>	数値
	バイトデータ値(0～255)を指定します。		
備 考	<ul style="list-style-type: none"> ・ この関数は、文字列のデータをバイトデータとして扱う為に用意されています。 ・ 「第5章 アスキーコード一覧」に載っていないコードは、画面表示が文字化けする恐れがあります。 ・ 得られる値は文字列形式ですが、PRINT文などで表示できない場合があります。(PRINT文で表示するには、UTF-8形式の文字列である必要があります) ・ Ver. 1.00 より、配列で与えると、文字を連結して返すようになりました。 		
使用例1	SMP\$ = CHRB\$(&h80) 変数SMP\$へ1バイトの&h80を保持するバイトデータ(文字列形式)を代入します。		
使用例2	SMP\$ = CHRB\$([&H40; &H41; &H42; &H43]) 変数SMP\$へ “@ABC” を代入します。		

2.3.10 CLOCK

関数			
機 能	経過時間の指標となる値を秒単位で得ます。		
書 式	<(戻り値)経過秒> = CLOCK		
戻り値	戻り値	<経過秒>	数値
	経過時間の指標となる値を秒単位で得ます。		
備 考	戻り値は、0～259200000秒の範囲で返ります。		
使用例1	ST_TM = CLOCK 何かの処理 EN_TM = CLOCK PRINT "経過時間"; (EN_TM - ST_TM); "秒"		
	上の例は、「何かの処理」の間を、CLOCK関数で読み取った値で差分を得る事で、処理に掛かった時間を算出しています。		
使用例2	PRINT CLOCK 例えば、「1572598873.1518」のような小数点数付きの経過秒が得られます。		

2.3.11 COS

関数			
機 能	余弦を返します。		
書 式	<(戻り値)余弦値> = COS(<①数式>)		
戻り値	戻り値	<余弦値>	数値
	余弦値が得られます。		
パラメータ	①	<数式>	数値
	コサインを求める角度を、ラジアンを単位として指定します。 角度が度で表されている場合は、PI / 180 を掛けてラジアンに変換します。 PI = 3.14159265358979		
使用例	NUM = COS(1)		
	変数NUMへ cos(1) ≒ 0.54030230586814を代入します。		

2.3.12 CSV2ARRAY\$

関数											
機能	CSV形式の文字列を、2次元文字列配列に変換します。										
書式	<(戻り値)2次元文字列配列> = CSV2ARRAY\$("<①CSV形式文字列>" [, "<②デリミタ>" [, "<③囲い文字>" [, <④行列スイッチ>]]])										
戻り値	戻り値	<2次元文字列配列> 配列 CSV変換された、2次元文字列配列。									
パラメータ	①	<CSV形式文字列> 文字列 CSV変換する対象となる文字列。 期待する文字列の形式は、デリミタ文字(例えばカンマ文字)で区切られた文字列を、項目毎の列情報とし、改行文字により行情報として扱います。 例： CSV 文字列(例) <div>S\$ = "A, B, C" + CHR\$(10) S\$ = S\$ + "D, E, F" LIST ARY\$ ARY\$ = CSV2ARRAY\$(S\$, ",", "","", FALSE)</div> <div>期待する文字列配列</div> <div><table><tr><td colspan="3">列</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>D</td><td>E</td><td>F</td></tr></table>⇕行</div>	列			A	B	C	D	E	F
	列										
	A	B	C								
	D	E	F								
	②	<デリミタ> 文字列 1項目を区切る記号となる文字を指定します。 省略時、「, (カンマ)」として扱います。									
③	<囲い文字> 文字列 項目を囲う文字を指定します。これを指定すると、囲い文字の範囲中に、デリミタ文字や改行文字が含まれる事が可能です。 省略時、囲い文字はありません。CSV形式で改行を含めたい場合、「" (ダブルクォーテーション)」を指定すると良いでしょう。										
④	<行列スイッチ> 真偽値 TRUE：列(1次元目)×行(2次元目)の2次元文字列配列を得ます。(省略時、デフォルト) FALSE：行(1次元目)×列(2次元目)の2次元文字列配列を得ます。										
備考	・改行に判定する文字は CHR\$(10)です。 ・変換対象となるCSV形式の文字列は、CSVの規格 RFC4180に準拠したものを用意してください。(規格で改行は CRLFとなっていますが、LF(CHR\$(10))で良いです) 規格に準拠していない文字列を読み込ませると、意図しない結果が得られる事があります。										
使用例	S\$ = '''あ,い,う え,お,"hello AJAN world"''' ''' ' S\$の文字列をCSV形式として解析して、ARY\$の配列に変換して表示します。 LIST ARY\$ ARY\$ = CSV2ARRAY\$(S\$, ",", "","", FALSE) ? ARY\$ ' 以下を実行すると「hello」、「AJAN」、「world」という文字列が3行に並んで表示										

	されます ? ARY\$(1, 2)
--	-----------------------

2.3.13 CVD

関数		
機 能	文字列を倍精度実数値データに変換します。MKD\$関数を用いて変換した文字列を、数値データに戻す際に使用します。	
書 式	<(戻り値) 倍精度実数値> = CVD("<①8文字の文字列>" [, <②オプション>])	
戻り値	戻り値	数値
	<倍精度実数値> 文字列を倍精度実数値データに変換した値が得られます。	
パラ メータ	①	文字列
	<8文字の文字列> 数値データを求める8文字の文字列を指定します。	
	②	文字列
	<オプション> オプション文字列を指定する事で、文字列から数値を読み取る際に、 特殊な変換を施せます。	
	オプション	効果
	big-endian	値を読み取る順番を、ビッグエンディアンで行います。
	なし	特になし。省略時と同じ。
備 考	関数MKD\$(→「MKD\$」) によって出力された文字列を数値に変換します。	
使用例	TMP\$ = MKD\$(12345) NUM = CVD(TMP\$) 変数NUMに12345 (MKD\$の引数として与えた元の数値) を代入します。	

2.3.14 CVH

関数		
機 能	文字列を単精度整数値データに変換します。(int16_t相当の読み取り) MKH\$関数を用いて変換した文字列を、数値データに戻す際に使用します。	
書 式	<(戻り値) 単精度整数値> = CVH("<①2文字の文字列>" [, <②オプション>])	
戻り値	戻り値	<単精度整数値> 数値
文字列を単精度整数値データに変換した値が得られます。 得られる値の範囲は、C言語でint16_t相当の範囲。32767~-32768です。		
パラ メータ	①	<2文字の文字列> 文字列
	数値データを求める2文字の文字列を指定します。	
	②	<オプション> 文字列
	オプション文字列を指定する事で、文字列から数値を読み取る際に、特殊な変換を施せます。	
	オプション	効果
	big-endian	値を読み取る順番を、ビッグエンディアンで行います。
	なし	特になし。省略時と同じ。
備 考	関数MKH\$(→「MKH\$」)によって出力された文字列を数値に変換します。	
使用例	TMP\$ = MKH\$(12345) NUM = CVH(TMP\$) 変数NUMに12345 (MKH\$の引数として与えた元の数値) を代入します。	

2.3.15 CVI

関数		
機 能	文字列を単精度整数値データに変換します。MKI\$関数を用いて変換した文字列を、数値データに戻す際に使用します。	
書 式	<(戻り値) 単精度整数値> = CVI("<①4文字の文字列>" [, <②オプション>])	
戻り値	戻り値	数値
	<単精度整数値> 文字列を単精度整数値データに変換した値が得られます。	
パラメータ	①	文字列
	<4文字の文字列> 数値データを求める4文字の文字列を指定します。	
	②	文字列
	<オプション> オプション文字列を指定する事で、文字列から数値を読み取る際に、特殊な変換を施せます。	
	オプション	効果
	big-endian	値を読み取る順番を、ビッグエンディアンで行います。
	なし	特になし。省略時と同じ。
備 考	関数MKI\$(→「MKI\$」)によって出力された文字列を数値に変換します。	
使用例	TMP\$ = MKI\$(12345) NUM = CVI(TMP\$) 変数NUMに12345 (MKI\$の引数として与えた元の数値) を代入します。	

2.3.16 CVL

関数			
機 能	文字列を倍精度整数値データに変換します。MKL\$関数を用いて変換した文字列を、数値データに戻す際に使用します。		
書 式	〈(戻り値) 倍精度整数値〉 = CVL(“〈①8文字の文字列〉” [, 〈②オプション〉])		
戻り値	戻り値	〈倍精度整数値〉	数値
	文字列を倍精度整数値データに変換した値が得られます。		
パラ メータ	①	〈8文字の文字列〉	文字列
	数値データを求める8文字の文字列を指定します。		
	②	〈オプション〉	文字列
	オプション文字列を指定する事で、文字列から数値を読み取る際に、特殊な変換を施せます。		
	オプション	効果	
	big-endian	値を読み取る順番を、ビッグエンディアンで行います。	
	なし	特になし。省略時と同じ。	
備 考	関数MKL\$(→「MKL\$」)によって出力された文字列を数値に変換します。		
使用例	TMP\$ = MKL\$(12345) NUM = CVL(TMP\$)		
	変数NUMに12345 (MKL\$の引数として与えた元の数値) を代入します。		

2.3.17 CVS

関数			
機 能	文字列を単精度実数値データに変換します。MKS\$関数を用いて変換した文字列を、数値データに戻す際に使用します。		
書 式	＜(戻り値) 単精度実数値＞ = CVS(“＜①4文字の文字列＞” [, ＜②オプション＞])		
戻り値	戻り値	＜単精度実数値＞	数値
	文字列を単精度実数値データに変換した値が得られます。		
パラ メータ	①	＜4文字の文字列＞	文字列
	数値データを求める4文字の文字列を指定します。		
	②	＜オプション＞	文字列
	オプション文字列を指定する事で、文字列から数値を読み取る際に、特殊な変換を施せます。		
	オプション	効果	
	big-endian	値を読み取る順番を、ビッグエンディアンで行います。	
	なし	特になし。省略時と同じ。	
備 考	関数MKS\$(→「MKS\$」)によって出力された文字列を数値に変換します。		
使用例	TMP\$ = MKS\$(12345) NUM = CVS(TMP\$) 変数NUMに12345 (MKS\$の引数として与えた元の数値) を代入します。		

2.3.18 DATE\$

関数			
機 能	日付を返します。		
書 式 1	<(戻り値) 日付文字列> = DATE\$ 今日の日付を返します。		
書 式 2	<(戻り値) 日付文字列> = DATE\$(<①年>, <①月>, <①日>) 今日から指定の期間だけ前後した日付を返します。		
書 式 3	<(戻り値) 日付文字列> = DATE\$(<②日付時刻型値>) 日付時刻型の値から日付を返します。		
戻り値	戻り値	<日付文字列>	文字列
	年(y)月(m)日(d)を、"yyyy/mm/dd"の形式で、文字列形式で得られます。		
パラ メータ	①	<年>, <月>, <日>	数値
	求める日付の今日から数えた期間を指定します。 過去の場合は負数で指定します。例えば、1日前は-1で表します。		
	②	<日付時刻型値>	日付時刻
備 考	日付時刻型の値を指定します。		
	・ある日付からの相対日付を求めたい場合、DATEADD関数を検討ください。		
使用例1	SMP\$ = DATE\$ 実行時点での日付が2017年1月1日の場合、"2017/01/01"を変数SMP\$へ代入します。		
使用例2	SMP\$ = DATE\$(1, -2, 3) 今日の日付へ+1年, -2ヶ月, +3日した日付を変数SMP\$へ代入します。 例えば、実行時点での日付が2017年1月1日の場合、"2017/11/04"を代入します。		
使用例3	SMP\$ = DATE\$(CDATETIME("2021/7/28 10:34:55")) 変数SMP\$に、日付時刻型の日付情報のみ(2021/07/28)が代入されます。		

2.3.19 EXP

関数			
機 能	底がeである指数関数の値を返します。		
書 式	<(戻り値) 指数関数値> = EXP(<①数式>)		
戻り値	戻り値	<指数関数値>	数値
	底がeである指数関数の値を返します。		
パラ メータ	①	<数式>	数値
	指数関数の値を求める数式を指定します。		
備 考	演算結果がオーバーフローすると値は inf(無限大) を返します。 演算結果がアンダーフローすると値は 0 を返します。		
使用例	NUM = EXP(1) eの1乗である 2.71828182845905 を変数NUMへ代入します。		

2.3.20 FIX

関数			
機 能	引数の整数部分を返します。		
書 式	<(戻り値) 整数値> = FIX(<①数値>)		
戻り値	戻り値	<整数値>	数値
	引数の整数部分を返します。		
パラ メータ	①	<数値>	数値
	整数部分を求める数値を指定します。		
使用例1	NUM = FIX(9.99) 変数NUMへ指定値「9.99」の整数部分「9」を代入します。		
使用例2	NUM = FIX(-9.99) 変数NUMへ指定値「-9.99」の整数部分「-9」を代入します。		

2.3.21 HEX\$

関数			
機 能	10進数を16進数の文字列に変換します。		
書 式	<(戻り値) 16進数文字列> = HEX\$(<①数値> [, <②桁数>])		
戻り値	戻り値	<16進数文字列>	文字列
	数値を16進数の文字列に変換した値が得られます。		
パラ メータ	①	<数値>	数値
	16進数文字列を求める数値を指定します。 指定された数値が整数でない場合は、小数第一位を四捨五入した値が変換されます。		
	②	<桁数>	数値
	1以上を指定すると、表示する桁数を強制できます。 結果は、桁数に従い、桁数不足時に"0"が左に付加されます。 指定できる桁数の最大値は、与える数値の型に依存します。例えば、単精度整数は8、倍精度整数は16です。		
備 考	倍精度整数の範囲（→基本マニュアルの「整数型定数」）を超える値の場合は"0"が返ります。		
使用例1	SMP\$ = HEX\$(255) 変数SMP\$へ"FF"（10進数の指定値「255」を16進数へ変換した値の文字列）を代入します。		
使用例2	PRINT HEX\$(123) 7B PRINT HEX\$(123, 4) 007B 桁数指定機能を使った場合と使わない場合の例です。		

2.3.22 INSTR

関数		
機 能	文字列中の指定文字列を検索し、位置を返します。 見つからなければ0を返します。	
書 式	〈(戻り値) 検索結果値〉 = INSTR ([〈①開始位置〉,] 〈②文字列〉, 〈③指定文字列〉 [, 〈④比較スイッチ〉])	
戻り値	戻り値	〈検索結果値〉
	文字列中の指定文字列を検索し、位置を返します。 見つからなければ0を返します。	
パラ メータ	①	〈開始位置〉
	検索を始める位置を1以上、文字列の長さ以下で指定します。 省略した場合は、先頭から検索します。	
	②	〈文字列〉
	検索対象の文字列を指定します。	
	③	〈指定文字列〉
パラ メータ	検索する文字列を指定します。	
	④	〈比較スイッチ〉
	検索比較する際の動作を指定します。 0：通常の文字列比較。省略時、この比較です。 1：大文字小文字を区別せずに比較します。	
備 考	<ul style="list-style-type: none"> ・〈指定文字列〉に空の文字列を指定すると、〈開始位置〉を返します。 ・文字列末尾から検索したい場合は、INSTREV 関数を使用します。 	
使用例1	NUM= INSTR ("ABCDEF", "C") 文字列「ABCDEF」から指定文字列「C」を検索し、該当した位置 3 を変数NUMへ数値型で代入します。	
使用例2	NUM= INSTR (4, "ABCABC", "C") 文字列「ABCABC」から開始位置 4 以降で指定文字列「C」を検索し、該当した位置 6 を変数NUMへ数値型で代入します。	
使用例3	NUM= INSTR ("あいうえお", "A") 文字列「あいうえお」から指定文字列「A」を検索し、該当が無いため 0 を変数NUMへ代入します。	

2.3.23 INSTRREV

関数		
機 能	文字列中の指定文字列を末尾から検索し、位置を返します。 見つからなければ0を返します。	
書 式	<(戻り値) 検索結果値> = INSTRREV(" <①文字列>", " <②指定文字列>" [, <③開始位置> [, <④比較スイッチ>]])	
戻り値	戻り値	<検索結果値> 数値
	文字列中の指定文字列を検索し、位置を返します。 見つからなければ0を返します。	
パラ メータ	①	<文字列> 文字列
	検索対象の文字列を指定します。	
	②	<指定文字列> 文字列
	検索する文字列を指定します。	
	③	<開始位置> 数値
	検索を始める位置を1以上、文字列の長さ以下で指定します。 省略した場合は -1とし、末尾から検索します。	
	④	<比較スイッチ> 数値
	検索比較する際の動作を指定します。 0 : 通常の文字列比較。省略時、この比較です。 1 : 大文字小文字を区別せずに比較します。	
備 考	・ <指定文字列>に空の文字列を指定すると、<開始位置>を返します。 ・ 文字列先頭から検索したい場合は、INSTR 関数を使用します。	
使用例	PRINT INSTR ("あいうえおあいうえお", "うえお") ' 3が得られます PRINT INSTRREV ("あいうえおあいうえお", "うえお") ' 8が得られます	

2.3.24 INT

関数			
機 能	引数を超えない最大の整数値を返します。		
書 式	<(戻り値) 整数値> = INT(<①数値>)		
戻り値	戻り値	<整数値>	数値
	引数を超えない最大の整数値を返します。		
パラ メータ	①	<数値>	数値
	整数値を求める数値を指定します。		
使用例1	NUM=INT(9.99)		
	指定値 9.99 を超えない最大の整数値 9 を変数NUMへ代入します。		
使用例2	NUM=INT(-9.99)		
	指定値 -9.99 を超えない最大の整数値 -10を変数NUMへ代入します。		

2.3.25 ISDATETIME

関数			
機 能	引数を日付時刻型値に変換できるか調べます。		
書 式	〈(戻り値)判定結果〉 = ISDATETIME(〈①引数〉 [, 〈②判定オプション〉])		
戻り値	戻り値	〈判定結果〉	
	引数が日付時刻型として変換可能であればTRUE。 そうでなければFALSEが得られます。		
パラ メータ	①	〈引数〉	
	日付時刻型として変換可能か調べる値を指定します。 文字列か数値を指定します。		
	②	〈判定オプション〉	
	判定を行う際の、期待する引数の書式を指定します。		
	判定オプション	期待する引数の書式	
	1	日付(年/月/日)の形式である事を期待する。	
2	時刻(時:分:秒)の形式である事を期待する。		
3	日付、時刻、日付時刻(年/月/日 時:分:秒)のどれかの形式である事を期待する。		
備 考	変換可能であれば、CDATETIME 関数で日付時刻型に変換できます。		
使用例	A\$ = "2017/1/1" IF ISDATETIME(A\$) = TRUE THEN PRINT CDATETIME(A\$) END IF ' 文字列(A\$)が、日付時刻型として変換可能かチェックして、変換します。		

2.3.26 ISNUMERIC

関数			
機 能	文字列を数値に変換できるか調べます。		
書 式	<(戻り値)判定結果> = ISNUMERIC(<①文字列>)		
戻り値	戻り値	<判定結果>	真偽値
	引数を数値に変換可能であればTRUE。 そうでなければFALSEが得られます。		
パラ メータ	①	<文字列>	文字列
	数値に変換できるか調べる文字列を指定します。		
備考	<ul style="list-style-type: none"> 変換可能であれば、VAL関数で数値に変換できます。 引数の文字列中、数値に変換できない文字にあたる場合、結果はFALSEです。 		
使用例	<pre>PRINT ISNUMERIC("123.45") ' 数値に変換できる場合、TRUEが得られます。 PRINT ISNUMERIC("HOGE") ' 数値に変換できない場合、FALSEが得られます。 PRINT ISNUMERIC("123HOGE") ' 途中数値に変換できない場合、FALSEが得られます。</pre>		

2.3.27 ISNAN

関数			
機 能	引数の値がNaN値(非数)か調べます。		
書 式	<(戻り値)判定結果> = ISNAN(<①数値>)		
戻り値	戻り値	<判定結果>	真偽値
	<値>がNaN値の場合、論理型定数（→基本マニュアルの「論理型定数」）のTRUEを返します。 そうでない場合 FALSEを返します。		
パラ メータ	①	<数値>	数値
	NaN値か調べる値を指定します。		
備考	NaN値とは実数値の中で数値として認識されない値を指します。		
使用例	? ISNAN(SQR(-1)) TRUE		
	指定した値がNaN値であるか表示します。		

2.3.28 ISINF

関数			
機 能	引数の値が無限大か調べます。		
書 式	<(戻り値)判定結果> = ISINF(<①数値>)		
戻り値	戻り値	<判定結果>	数値
	<値>が正の無限大の場合 1を返します。負の無限大の場合 - 1を返します。無限大でない場合 0を返します。		
パラ メータ	①	<数値>	数値
	無限大か調べる値を指定します。		
使用例1	? ISINF(EXP(710)) 1		
	指定した値が無限大であるか表示します。		

2.3.29 LEN

関数			
機 能	文字列の長さを文字数で返します。		
書 式	<(戻り値)文字数> = LEN("<①文字列>")		
戻り値	戻り値	<文字数>	数値
	文字列の長さを文字数で返します。		
パラメータ	①	<文字列>	文字列
	長さを求める文字列を指定します。		
使用例1	NUM=LEN("TEST") 指定文字列「TEST」の文字数「4」を数値型で変数NUMへ代入します。		
使用例2	NUM=LEN("あいう") 指定文字列「あいう」の文字数「3」を数値型で変数NUMへ代入します。		

2.3.30 LENB

関数			
機 能	文字列の長さをバイト数で返します。		
書 式	<(戻り値)バイト数> = LENB("<①文字列>")		
戻り値	戻り値	<バイト数>	数値
	文字列の長さをバイト数で返します。		
パラメータ	①	<文字列>	文字列
	バイト長を求める文字列を指定します。		
備 考	<ul style="list-style-type: none"> この関数は、文字列のデータをバイトデータとして扱う為に用意されています。 全角文字1文字あたり複数バイト、半角ASCII文字1文字あたり1バイトとして計算します。 		
使用例1	NUM=LENB("TEST") 指定文字列「TEST」のバイト数「4」を数値型で変数NUMへ代入します。		
使用例2	NUM=LENB("あいう") 指定文字列「あいう」は全角文字です。各1文字あたり3バイトの為、バイト数「9」を数値型で変数NUMへ代入します。		

2.3.31 LINSTR

関数		
機 能	文字列中の指定文字列を検索し、その数を返します。 見つからない場合は、0を返します。	
書 式	〈(戻り値) 検索結果値〉 = LINSTR([〈①開始位置〉,] 〈②文字列〉, 〈③指定文字列〉 [, 〈④比較スイッチ〉])	
戻り値	戻り値	検索結果値 数値 文字列中の指定文字列を検索し、その数を返します。 見つからなければ0を返します。
パラ メータ	①	開始位置 数値 検索を始める位置を1以上、文字列の長さ以下で指定します。 省略した場合は、先頭から検索します。
	②	文字列 文字列 検索対象の文字列を指定します。
	③	指定文字列 文字列 検索する文字列を指定します。
	④	比較スイッチ 数値 検索比較する際の動作を指定します。 0：通常の文字列比較。省略時、この比較です。 1：大文字小文字を区別せずに比較します。
	備考 〈指定文字列〉に空の文字列を指定すると、0を返します。 指定文字列は左から順に検索します。例えば、文字列が"CCC"で指定文字列が"CC"の場合、 初めの2文字のみが検索対象となり、結果は1となります。	
使用例1	NUM=LINSTR ("ABCDEF", "C") 文字列「ABCDEF」の中で指定文字列「C」がいくつあるか検索した結果「1」を変数NUMへ代入します。	
使用例2	NUM=LINSTR (4, "ABCABC", "C") 文字列「ABCABC」の中で指定文字列「C」が開始位置「4」以降にいくつあるか検索した結果 「1」を変数NUMへ代入します。	
使用例3	NUM=LINSTR ("ABCABC", "C") 文字列「ABCABC」の中で指定文字列「C」がいくつあるか検索した結果「2」を変数NUMへ代入 します。	
使用例4	NUM=LINSTR ("AAAAAAA", "AAA") 文字列「AAAAAAA」の中で指定文字列「AAA」がいくつあるか検索した結果「2」を変数NUMへ 代入します。	

2.3.32 LOG

関数			
機 能	対数を返します。		
書 式	<(戻り値) 対数値> = LOG(<①数式>[, <②底>])		
戻り値	戻り値	<対数値>	数値
	対数値を返します。		
パラ メータ	①	<数式>	数値
	対数を求める数式を指定します。		
	②	<底>	数値
備 考	対数の底となる値を指定します。指定しない場合は自然対数を返します。		
	結果が非数の場合はnan、無限大の場合はinfを返します。		
使用例1	NUM = LOG(1)		
使用例2	変数NUMに自然対数 $\log_e 1 = 0$ を代入します。		
	NUM = LOG(81, 9)		
	変数NUMに対数 $\log_9(81) = 2$ を代入します。		

2.3.33 LSTRIP\$

関数			
機 能	文字列の先頭部分に対して、指定した文字集合を除去した文字列を返します。		
書 式	<(戻り値) 除去後文字列> = LSTRIP\$(<①文字列> [, <②除去する文字集合>])		
戻り値	戻り値	<除去後文字列>	文字列
	先頭部分から除去された文字列が得られます。		
パラ メータ	①	<文字列>	文字列
	除去対象となる文字列を与えます。		
	②	<除去する文字集合>	文字列
備 考	除去する文字集合を与えます。省略すると空白文字が除去対象となります。		
	<ul style="list-style-type: none"> 文字集合は、文字列の各文字が除去対象を指します。 例えば、“123”を文字集合とすると、“1”と“2”と“3”のいずれかが除去対象です。 先頭部分の部分文字列を削除したい場合は、STR_REMOVEPREFIX\$ の使用を検討してください。 		
使用例	S\$ = LSTRIP\$("1232145TEST54321", "123")		
	文字列「1232145TEST54321」に対して、先頭から文字集合「123」を除去した、		
	「45TEST54321」が得られます。		

2.3.34 MEMINFO

関数			
機 能	システムのメモリに関する情報を取得します。		
書 式	<(戻り値)メモリ容量> = MEMINFO(<①機能番号>)		
戻り値	戻り値	<メモリ容量>	数値
	機能番号に対応する、メモリ容量値が得られます。		
パラ メータ	①	<機能番号>	数値
	取得する情報を番号で指定します。 0：システム全体の空きメモリ容量をKB単位で得ます 1：システム全体のメモリ容量をKB単位で得ます 2：本プログラムが使用しているメモリ容量をKB単位で得ます		
備 考	プログラム中で大量のメモリを消費し、システムの継続動作が困難になるとOSからAJANが強制終了される場合があります。 本関数を使用することでメモリ使用量に関する警告を出したり、正常に終了できるようになります。		
注 意	取得できる情報はOSが認識するメモリ情報と同じです。		
使用例	IF MEMINFO(0) < (100 * 1024) THEN PRINT "システムの空きメモリ容量が100MBを下回りました" END IF システムの空きメモリ容量が100MB(100 * 1024 KB) を下回った場合に警告のメッセージを表示します。		

2.3.35 MID\$

関数		
機 能	文字列の中から任意の長さの文字列を抜き出します。	
書 式	〈(戻り値) 部分文字列〉 = MID\$("〈①文字列〉", 〈②開始位置〉 [, 〈③文字数〉])	
戻り値	戻り値	文字列
	文字列の中から任意の長さで抜き出した文字列が得られます。	
パラ メータ	①	文字列
	抜き出し対象の文字列を指定します。	
	②	数値
	抜き出す開始位置を1以上で指定します。 〈文字列〉の文字数より大きければ、空の文字列を返します。	
	③	数値
	抜き出す文字数を0以上で指定します。 省略した場合や、〈開始位置〉以降の文字数より大きい値を指定した場合は、〈開始位置〉以降の文字列をすべて抜き出します。	
使用例1	SMP\$ = MID\$("SAMPLE", 2, 3) 変数SMP\$に"AMP" ("SAMPLE"の2文字目から3文字取り出した文字列) を代入します。	
使用例2	SMP\$ = MID\$("SAMPLE", 3) 変数SMP\$に"MPLE" ("SAMPLE"の3文字目以降を取り出した文字列) を代入します。	
使用例3	SMP\$ = MID\$("あいうえお", 2, 3) 変数SMP\$に"いうえ"を代入します。 〈文字変数〉の内容や〈文字列〉が全角文字の場合でも使用できます。	

2.3.36 MIDB\$

関数		
機 能	文字列の中から任意のバイト数分の文字列を抜き出します。	
書 式	<(戻り値)部分文字列> = MIDB\$("<①文字列>", <②開始位置> [, <③バイト数>])	
戻り値	戻り値	<部分文字列> 文字列
	文字列の中から任意の長さで抜き出した文字列が得られます。	
パラ メータ	①	<文字列> 文字列
	抜き出し対象の文字列を指定します。	
	②	<開始位置> 数値
	抜き出す開始位置を1以上のバイト数単位で指定します。 <文字列>のバイト数より大きければ、空の文字列を返します。	
	③	<バイト数> 数値
	抜き出すバイト数を0以上で指定します。 省略した場合や、<開始位置>以降のバイト数より大きい値を指定した場合は、<開始位置>以降の文字列をすべて抜き出します。	
備 考	<ul style="list-style-type: none"> この関数は、文字列のデータをバイトデータとして扱う為に用意されています。 得られる値は文字列形式ですが、PRINT文などで表示できない場合があります。 (PRINT文で表示するには、UTF-8形式の文字列である必要があります) 	
使用例	SRC\$=CHRB\$(0)+CHRB\$(1)+CHRB\$(2)+CHRB\$(3) SMP\$=MIDB\$(SRC\$, 2, 2) SRC\$は、4バイトのバイトデータ(文字列形式)です。 MIDB\$により、先頭2バイト目から2バイトのバイトデータを抜き出し、SMP\$には 「CHRB\$(1)+CHRB\$(2)」のバイトデータが得られます。	

2.3.37 MKD\$

関数		
機 能	倍精度実数値データを数値の内部表現に対応した文字列に変換します。数値データをバイト列として扱う際、文字列に変換するために使用します。 元に戻すにはCVD関数を使用します。	
書 式	〈(戻り値)8文字の文字列〉 = MKD\$ (〈①倍精度実数値〉 [, 〈②オプション〉])	
戻り値	戻り値	文字列
	倍精度実数値データを数値の内部表現に対応した、8文字の文字列に変換します。	
パラメータ	①	数値
	倍精度実数値	
	8文字の文字列を求める倍精度実数値を指定します。	
	②	文字列
	オプション	
	オプション文字列を指定する事で、数値から文字列に書き込む際に、特殊な変換を施せます。	
	オプション	効果
	big-endian	データを書き込む順番を、ビッグエンディアンで行います。
	なし	特になし。省略時と同じ。
備 考	8文字の文字列に変換できる実数値は実質的にCVD(→「2.3.13 CVD」)によって得られる値のみです。	
使用例	TMP# = CVD("ABCDEFGH") SMP\$ = MKD\$(TMP#) 変数SMP\$にMKD\$の実行結果を代入します。	

2.3.38 MKH\$

関数								
機 能	単精度整数値データ (int16_t相当の範囲のみ扱う) を数値の内部表現に対応した文字列に変換します。 数値データをバイト列として扱う際、文字列に変換するために使用します。 元に戻すにはCVH関数を使用します。							
書 式	<(戻り値)2文字の文字列> = MKH\$(<①単精度整数値> [, <②オプション>])							
戻り値	戻り値	<2文字の文字列> 文字列 単精度整数値データを数値の内部表現に対応した、2文字の文字列に変換します。						
パラメータ	①	<単精度整数値> 数値 2文字の文字列を求める単精度整数値を指定します。 値は、C言語でint16_t相当の範囲。32767~-32768 までを指定してください。						
	②	<オプション> 文字列 オプション文字列を指定する事で、数値から文字列に書き込む際に、特殊な変換を施せます。 <table><tr><td>オプション</td><td>効果</td></tr><tr><td>big-endian</td><td>データを書き込む順番を、ビッグエンディアンで行います。</td></tr><tr><td>なし</td><td>特になし。省略時と同じ。</td></tr></table>	オプション	効果	big-endian	データを書き込む順番を、ビッグエンディアンで行います。	なし	特になし。省略時と同じ。
オプション	効果							
big-endian	データを書き込む順番を、ビッグエンディアンで行います。							
なし	特になし。省略時と同じ。							
備 考	2文字の文字列に変換できる整数値は実質的にCVH(→「CVH」)によって得られる値のみです。							
使用例	SMP\$ = MKH\$(12345) 変数SMP\$にMKH\$の実行結果を代入します。							

2.3.39 MKI\$

関数		
機 能	単精度整数値データを数値の内部表現に対応した文字列に変換します。数値データをバイト列として扱う際、文字列に変換するために使用します。 元に戻すにはCVI関数を使用します。	
書 式	<(戻り値)4文字の文字列> = MKI\$(<①単精度整数値> [, <②オプション>])	
戻り値	戻り値	<4文字の文字列>
	単精度整数値データを数値の内部表現に対応した、4文字の文字列に変換します。	
パラメータ	①	<単精度整数値>
	4文字の文字列を求める単精度整数値を指定します。	
	②	<オプション>
	オプション文字列を指定する事で、数値から文字列に書き込む際に、特殊な変換を施せます。	
	オプション	効果
	big-endian	データを書き込む順番を、ビッグエンディアンで行います。
	なし	特になし。省略時と同じ。
備 考	4文字の文字列に変換できる整数値は実質的にCVI(→「2.3.15 CVI」)によって得られる値のみです。	
使用例	TMP% = CVI("ABCD") SMP\$ = MKI\$(TMP%) 変数SMP\$にMKI\$の実行結果を代入します。	

2.3.40 MKL\$

関数		
機 能	倍精度整数値データを数値の内部表現に対応した文字列に変換します。数値データをバイト列として扱う際、文字列に変換するために使用します。 元に戻すにはCVL関数を使用します。	
書 式	<(戻り値)8文字の文字列> = MKL\$(<①倍精度整数値> [, <②オプション>])	
戻り値	戻り値	<8文字の文字列> 文字列 倍精度整数値データを数値の内部表現に対応した、8文字の文字列に変換します。
パラメータ	①	<倍精度整数値> 数値 8文字の文字列を求める倍精度整数値を指定します。
	②	<オプション> 文字列 オプション文字列を指定する事で、数値から文字列に書き込む際に、特殊な変換を施せます。
	オプション	効果
	big-endian	データを書き込む順番を、ビッグエンディアンで行います。
	なし	特になし。省略時と同じ。
備 考	8文字の文字列に変換できる整数値は実質的にCVL(→「2.3.16 CVL」)によって得られる値のみです。	
使用例	TMP& = CVL("ABCDEFGH") SMP\$ = MKL\$(TMP&) 変数SMP\$にMKL\$の実行結果を代入します。	

2.3.41 MKS\$

関数		
機 能	単精度実数値データを数値の内部表現に対応した文字列に変換します。数値データをバイト列として扱う際、文字列に変換するために使用します。 元に戻すにはCVS関数を使用します。	
書 式	<(戻り値)4文字の文字列> = MKS\$(<①単精度実数値> [, <②オプション>])	
戻り値	戻り値	<4文字の文字列> 文字列 単精度実数値データを数値の内部表現に対応した、4文字の文字列に変換します。
パラメータ	①	<単精度実数値> 数値 4文字の文字列を求める単精度実数値を指定します。
	②	<オプション> 文字列 オプション文字列を指定する事で、数値から文字列に書き込む際に、特殊な変換を施せます。
	オプション	効果
	big-endian	データを書き込む順番を、ビッグエンディアンで行います。
	なし	特になし。省略時と同じ。
備 考	4文字の文字列に変換できる実数値は実質的にCVS(→「2.3.17 CVS」)によって得られる値のみです。	
使用例	TMP! = CVS("ABCD") SMP\$ = MKS\$(TMP!) 変数SMP\$にMKS\$の実行結果を代入します。	

2.3.42 RANDOMIZE

命令			
機能	新しい乱数系列を設定します。		
書式	RANDOMIZE [<①数式>]		
パラメータ	①	<数式>	数値
	RND関数で発生させる擬似乱数の発生系列を単精度整数型で指定します。 範囲を超えて指定した場合は、単精度整数型に丸められます。 同じ系列を指定すると、RND関数は同じ繰返しで擬似乱数を発生させます。 省略した場合は1になります。		
備考	RND (→「2.3.44 RND」)		
使用例1	RANDOMIZE 5 NUM = RND 発生系列に5を指定してRANDOMIZEを実行して乱数系列を設定します。 その後変数NUMにRNDによって生成した乱数を代入します。		
	RANDOMIZE NUM = RND RANDOMIZEを実行して乱数系列を設定します。 その後変数NUMにRNDによって生成した乱数を代入します。 発生系列は1となります。		
使用例2	RANDOMIZE NUM = RND RANDOMIZEを実行して乱数系列を設定します。 その後変数NUMにRNDによって生成した乱数を代入します。 発生系列は1となります。		

2.3.43 REPLACE\$

関数			
機能	文字列中の指定文字を別の文字列に置き換えます。		
書式	<(戻り値) 置換後文字列> = REPLACE\$("<①文字列>", "<②指定文字列>", "<③置換文字列>" [, <置換回数>])		
戻り値	戻り値	<置換後文字列>	文字列
	文字列中の指定文字列を別の置換文字列に置き換えた文字列が得られます。		
パラメータ	①	<文字列>	文字列
	置換対象の文字列を指定します。		
	②	<指定文字列>	文字列
	置換される文字列を指定します。		
	③	<置換文字列>	文字列
	置換する文字列を指定します。		
	③	<置換回数>	数値
	置換する回数を指定します。		
	負数ないしは省略すると、全て置換します。		
使用例1	PRINT REPLACE\$("123456123456", "123", "AB") ' AB456AB456 が表示されます。 PRINT REPLACE\$("123456123456", "123", "AB", 1) ' AB456123456 が表示されます。		
使用例2	CHAR\$ = REPLACE\$("あいうえお", "あいう", "かきく") 変数CHAR\$に"かきくえお" ("あいうえお"の"あいう"を"かきく"に置き換えた文字列) が代入されます。		

2.3.44 RND

関数			
機能	0以上1未満の乱数を返します。 得られる乱数は、実行されるごとにいつも同系列となります。 ただし、RANDOMIZEを使用すれば、この系列を変えることができます。		
書式	<(戻り値) 乱数値> = RND[(<①設定値>)]		
戻り値	戻り値	<乱数値>	数値
	0以上1未満の乱数値を返します。		
パラメータ	①	<設定値>	数値
	本関数の挙動を以下の値で指定します。 省略した場合、正の数を指定したのと同じ機能になります。 負の数：乱数系列を引数の絶対値で初期化した後に、乱数の値をとる 0 : 1つ前に発生した乱数の値をとる(繰り返す) 正の数：次の乱数を発生する		
備考	RANDOMIZE (→「2.3.42 RANDOMIZE」)		
使用例	PRINT RND PRINT RND PRINT RND(0) ----- 実行結果 0.447415203064408 0.291984428321935 0.291984428321935 RND(0)では前に実行したRNDと同じ値が出力されます。		

2.3.45 ROUND

関数			
機 能	数値を四捨五入して指定された桁数 にします。		
書 式	<(戻り値) 丸め値> = ROUND(<①数値>[, <②桁数>])		
戻り値	戻り値	<丸め値>	数値
	数値を四捨五入して、指定された桁数の丸め数値が得られます。		
パラ メータ	①	<数値>	数値
	四捨五入の対象となる数値を指定します。		
	②	<桁数>	数値
	数値を四捨五入した結果の桁数を指定します。 正の値を指定すると小数点以下、負の値だと小数点以上(整数部)を四捨五入します。 省略すると小数点以下1桁目を四捨五入します。		
使用例1	NUM = ROUND(123.456) 変数NUMに123 (123.456の小数第1位を四捨五入した値) を代入します。		
使用例2	NUM = ROUND(123.456789, 3) 変数NUMに123.457 (123.456789の小数第4位を四捨五入した値) を代入します。		
使用例3	NUM = ROUND(123.456, -1) 変数NUMに120 (123.456の整数部の1桁目を四捨五入した値) を代入します。		

2.3.46 RSTRIP\$

関数			
機 能	文字列の末尾部分に対して、指定した文字集合を除去した文字列を返します。		
書 式	<(戻り値) 除去後文字列> = RSTRIP\$(<①文字列> [, <②除去する文字集合>])		
戻り値	戻り値	<除去後文字列>	文字列
	末尾部分から除去された文字列が得られます。		
パラ メータ	①	<文字列>	文字列
	除去対象となる文字列を与えます。		
	②	<除去する文字集合>	文字列
	除去する文字集合を与えます。省略すると空白文字が除去対象となります。		
備 考	<ul style="list-style-type: none">文字集合は、文字列の各文字が除去対象を指します。 例えば、“123”を文字集合とすると、“1”と“2”と“3”のいずれかが除去対象です。末尾部分の部分文字列を削除したい場合は、STR_REMOVESUFFIX\$ の使用を検討してください。		
使用例	S\$ = RSTRIP\$("12345TEST5412321", "123") 文字列「"12345TEST5412321"」に対して、末尾から文字集合「"123"」を除去した、「"12345TEST54"」が得られます。		

2.3.47 SIN

関数			
機 能	正弦を返します。		
書 式	<(戻り値) 正弦値> = SIN(<①数式>)		
戻り値	戻り値	<正弦値>	数値
	正弦値が得られます。		
パラ メータ	①	<数式>	数値
	サインを計算する角度を、ラジアンを単位として指定します。 角度が度を単位として表されている場合は、PI / 180 をかけてラジアンに変換します。 PI = 3.14159265358979		
使用例	PI = 3.14159265358979 NUM = SIN(PI / 6)		
	変数NUMにsin(PI / 6) ≒ 0.5を代入します。		

2.3.48 SPLIT\$

関数			
機 能	文字列に対して、指定した区切り文字列に従い分割します。分割した文字列は1次元文字列配列に格納し返します。		
書 式	<(戻り値)文字列配列> = SPLIT\$(<①対象文字列> [, <②区切り文字列> [, <③まとめフラグ>]])		
戻り値	戻り値	<文字列配列>	配列
	区切り文字列により、分割された1次元の文字列配列です。		
パラ メータ	①	<対象文字列>	文字列
	区切り記号入りの文字列です。 この文字列を区切り文字列で分割して、1次元の文字列配列を生成します。		
	②	<区切り文字列>	文字列
	区切り記号入り文字列である対象文字列に対する、区切り文字列を指定します。 省略すると、半角1文字の空白(" ")となります。		
	③	<まとめフラグ>	真偽値
	TRUE を指定すると、区切り文字列が連続する際、1つの区切り文字列として処理します。 省略時、FALSE として扱います。 この機能は、Ver.1.00 より追加されました。		
備 考	・ 戻り値の配列の要素数が事前に判らない場合、LIST命令で可変長配列を宣言する事で、全て受け取る事ができます。		
使用例1	LIST A\$ A\$ = SPLIT\$("hello,AJAN,world", ",") ["hello"; "AJAN"; "world"] の1次元の文字列配列が得られます。		
使用例2	LIST A\$ A\$ = SPLIT\$("hello,,world", ",") ["hello"; " "; "world"] の1次元の文字列配列が得られます。		
使用例3	PRINT SPLIT\$("ab12cd12ef12gh", "12") [ab, cd, ef, gh] "12"を区切り文字とした例です。		
使用例4	PRINT SPLIT\$("A:B:::C", ":", TRUE) ' [A, B, C] が得られます PRINT SPLIT\$("A:B:::C", ":", FALSE) ' [A, B, , , C] が得られます。		

2.3.49 SQR

関数			
機 能	正の平方根を返します。		
書 式	<(戻り値) 正の平方根値> = SQR(<①数式>)		
戻り値	戻り値	<正の平方根値>	数値
	正の平方根が得られます。		
パラ メータ	①	<数式>	数値
	平方根を求める数式を指定します。		
備 考	・ 結果が非数の場合はNaN値を返します。		
使用例	NUM# = SQR(3)		
	変数NUM#に $\sqrt{3} \div 1.73205080756888$ を代入します。		

2.3.50 STR\$

関数			
機 能	数値を文字列に変換します。		
書 式	<(戻り値) 数文字列> = STR\$(<①数値>)		
戻り値	戻り値	<数文字列>	文字列
	数値を文字列に変換した値が得られます。		
パラ メータ	①	<数値>	数値
	文字列に変換する数値を指定します。		
備 考	・ 日付時刻型の変数および定数を入れると、「年/月/日 時:分:秒」の文字列として変換されます。		
使用例1	SMP\$ = STR\$(123)		
	変数SMP\$に"123"を代入します。		
使用例2	DATETIME A A = CDATETIME (DATE\$) PRINT STRDEL\$(STR\$(A+7), 11)		
	本日の一週間後の日付を表示します		

2.3.51 STRDEL\$

関数			
機 能	文字列の任意の位置から指定した文字長の文字列を削除します。		
書 式	<(戻り値)削除後文字列> = STRDEL\$("<①文字列>", <②開始位置> [, <③削除する文字数>])		
戻り値	戻り値	<削除後文字列>	文字列
	指定文字列の開始位置から指定した文字長の文字列を削除した文字列が得られます。		
パラ メータ	①	<文字列>	文字列
	操作対象の文字列を指定します。		
	②	<開始位置>	数値
	削除する開始位置を1以上で指定します。 <文字列>の文字数より大きいと文字長と同じになります。		
	③	<削除する文字数>	数値
	削除する文字数を0以上で指定します。 省略した場合は、<開始位置>以降の文字列をすべて削除します。 <開始位置>以降の文字数より大きい値を指定した場合は、<開始位置>以降の文字列をすべて削除します。		
使用例1	SMP\$ = STRDEL\$("1い3ろ5は", 4) 文字列 "1い3ろ5は"の4番目以降の文字を全て削除します。 変数SMP\$には、"1い3"が得られます。		
使用例2	SMP\$ = STRDEL\$ ("1い3ろ5は", 2, 3) 文字列 "1い3ろ5は"の2番目から3文字を削除します。 変数SMP\$には、"15は"が得られます。		

2.3.52 STRINS\$

関数		
機 能	文字列の任意の位置に指定した文字列を挿入します。	
書 式	<(戻り値)挿入後文字列> = STRINS\$("<①文字列>", <②開始位置>, "<③挿入する文字列>")	
戻り値	戻り値	<挿入後文字列> 文字列の中から開始位置に文字列を挿入した文字列が得られます。
パラ メータ	①	<文字列> 操作対象の文字列を指定します。
	②	<開始位置> 挿入する開始位置を1以上で指定します。 <文字列>の文字数より大きいと文字数と同じになります。
	③	<挿入する文字列> 挿入する文字列を指定します。
使用例	SMP\$ = STRINS\$("1い3ろ5は", 3, "a歩") 文字列 "1い3ろ5は" の3番目から "a歩" を挿入します。 変数SMP\$には、"1いa歩3ろ5は" が得られます。	

2.3.53 STRIP\$

関数		
機 能	文字列の両端部分に対して、指定した文字集合を除去した文字列を返します。	
書 式	<(戻り値)除去後文字列> = STRIP\$(<①文字列> [, <②除去する文字集合>])	
戻り値	戻り値	<除去後文字列> 両端部分から除去された文字列が得られます。
パラ メータ	①	<文字列> 除去対象となる文字列を与えます。
	②	<除去する文字集合> 除去する文字集合を与えます。省略すると空白文字が除去対象となります。
備 考	<ul style="list-style-type: none"> 文字集合は、文字列の各文字が除去対象を指します。 例えば、"123"を文字集合とすると、"1"と"2"と"3"のいずれかが除去対象です。 先頭部分の部分文字列を削除したい場合は、STR_REMOVEPREFIX\$ の使用を検討してください。 末尾部分の部分文字列を削除したい場合は、STR_REMOVESUFFIX\$ の使用を検討してください。 	
使用例	S\$ = STRIP\$("1232145TEST5412321", "123") 文字列「"1232145TEST5412321"」に対して、末尾から文字集合「"123"」を除去した、「"45TEST54"」が得られます。	

2.3.54 TAN

関数			
機 能	正接を返します。		
書 式	<(戻り値)正接値> = TAN(<①数式>)		
戻り値	戻り値	<正接値>	数値
	正接値が得られます。		
パラ メータ	①	<数式>	数値
	正接を求める数式を指定します。 タンジェントを求める角度を、ラジアンを単位として指定します。 角度が度で表されている場合は、PI / 180 を掛けてラジアンに変換します。 PI = 3.14159265358979		
使用例	PI = 3.141592653589793 NUM = TAN(PI / 4) 変数NUMにtan(PI / 4) ≒ 1を代入します。		

2.3.55 TIME\$

関数			
機 能	時刻を返します。		
書 式 1	<(戻り値)時刻文字列> = TIME\$ 今の時刻を返します。		
書 式 2	<(戻り値)時刻文字列> = TIME\$(<①秒数>) "00:00:00"を起点として指定秒数を時刻に換算して返します。		
書 式 3	<(戻り値)時刻文字列> = TIME\$(<②時>, <②分>, <②秒>) 今から指定の時間だけ前後した時刻を返します。		
書 式 4	<(戻り値)時刻文字列> = TIME\$(<③単位オプション>) 今の時刻を返すとき、ミリ秒、マイクロ秒、ナノ秒まで付加して返します。		
書 式 5	<(戻り値)時刻文字列> = TIME\$(<④日付時刻型値>) 日付時刻型の値から時刻を返します。		
戻り値	戻り値	<時刻文字列>	文字列
	書式1, 書式2, 書式3の時: 時(h)分(m)秒(s)を、"hh:mm:ss"の形式で、文字列形式で得られます。 書式4の時: 時(h)分(m)秒(s)秒以下数値(x)を、"hh:mm:ss.xxx"の形式で、文字列形式で得られます。		
パラ メータ	①	<秒数>	数値
	求める時刻の起点から数えた秒数を指定します。		
	②	<時>, <分>, <秒>	数値
	求める時刻の今から数えた時間を指定します。 過去の場合は負数で指定します。例えば、1秒前は-1で表します。		
	③	<単位オプション>	文字列
	今の時刻を得る際に、ミリ秒、マイクロ秒、ナノ秒の単位まで情報を付加できるよう、指定します。		
	設定値	内容	
	"m"	ミリ秒単位で得られます。	
	"u"	マイクロ秒単位で得られます。(非推奨)	
	"n"	ナノ秒単位で得られます。(非推奨)	
	⑤	<日付時刻型値>	日付時刻
	日付時刻型の値を指定します。		
	書式4で得られる、ミリ秒、マイクロ秒、ナノ秒単位の付加値は、CDATETIME 関数などで日付時刻型に変換して得られます。 日付時刻型への変換後、再び文字列に再変換する場合は、FORMATDATETIME 関数を使用します。 (ただしミリ秒まで。マイクロ秒・ナノ秒情報は内部で誤差が生じる為、利用できません) その他の命令・関数は、日付時刻型から文字列に変換する際に、秒単位に切り捨てられます。		
	使用例1 SMP\$ = TIME\$ 変数SMP\$に現在時刻を代入します。 例えば、現在時刻が1時2分3秒の場合、"01:02:03"を代入します。		
	使用例2 SMP\$ = TIME\$(119) 変数SMP\$に119秒を時刻に換算した文字列"00:01:59"を代入します。		
	使用例3 SMP\$ = TIME\$(1, -2, 3) 変数SMP\$に現在時刻の58分3秒後 (1, -2, 3 = 1時間後, 2分前, 3秒後のため) の時刻を		

AJAN標準コマンドリファレンス

	代入します。 例えば、現在時刻が1時2分3秒の場合、“02:00:06” を代入します。
使用例4	SMP\$ = TIME\$ (“m”) 変数SMP\$に現在時刻を代入します。 例えば、“01:02:03.456” のような結果が得られます。
仕様例5	SMP\$ = TIME\$ (CDATETIME (“2021/7/28 10:34:55”)) 変数SMP\$に、日付時刻型の時刻情報のみ(10:34:55)が代入されます。

2.3.56 TRIM\$

関数		
機 能	文字列の前後から全半角スペースを削除します。 (文字列中のスペースは削除されません。)	
書 式	<(戻り値)変換後文字列> = TRIM\$("<①文字列>")	
戻り値	戻り値	<変換後文字列> 文字列
	文字列の前後から全半角スペースを削除した文字列が得られます。	
パラ メータ	①	<文字列> 文字列
	スペースを削除する文字列を指定します。	
使用例1	CHAR\$ = TRIM\$(" 123 ") 変数CHAR\$に"123"(文字列の先頭および終端が全半角スペース以外になるように全半角スペースを削除したもの)を代入します。	
使用例2	CHAR\$ = TRIM\$(" あいうえお 123 ") 引数に全角文字が含まれている場合でも使用できます。 変数CHAR\$に"あいうえお 123"を代入します。	

2.3.57 VAL

関数		
機 能	数値表記の文字列を、実際の数値に変換します。	
書 式	<(戻り値)数値> = VAL("<①数値表記の文字列>")	
戻り値	戻り値	<数値> 数値
	文字列表記の数値を実際の数値に変換した値が得られます。	
パラ メータ	①	<数値表記の文字列> 文字列
	数値に変換する文字列を指定します。 整数表記(2進数形式, 10進数形式, 16進数形式)および実数表記(通常的小数点形式, 指数形式)のいずれも指定できます。(ただし、文字列で指定する事)	
備 考	<ul style="list-style-type: none"> 最初の文字が数字、または +, -, & でなければ、VALの値は0になります。 文字列中に数字以外の文字が見つかったら、それ以上の読み込みを中止します。 	
使用例1	NUM = VAL("123") 変数NUMに123を代入します。	
使用例2	NUM = VAL("&HFF") 変数NUMに255を代入します。	
使用例3	NUM = VAL("1.23e+3") 変数NUMに1230を代入します。	
使用例4	' 「HOGE」は数値として読み取れないので 0 が得られます PRINT VAL("HOGE") ' 「123HOGE」は、途中の「123」まで読み取って中断した結果、123 が得られます。 PRINT VAL("123HOGE")	

2.3.58 WEEK

関数			
機 能	指定日（文字列）の曜日を返します。		
書 式	<(戻り値)曜日の数値> = WEEK(<①日付文字列または日付時刻値>)		
戻り値	戻り値	<曜日の数値>	数値
	指定日の曜日を 1～7の数値で返します。 1：日曜日 2：月曜日 3：火曜日 4：水曜日 5：木曜日 6：金曜日 7：土曜日		
パラ メータ	①	<日付文字列または日付時刻値>	文字列／ 日付時刻
	曜日を求める日付を指定します。 日付は以下のフォーマットの文字列か、日付時刻型の値が指定できます。 yyyy/mm/dd（年/月/日）		
備 考	<日付文字列>が日付でない場合(例：WEEK("TEST"))、0を返します。 閏年ではない年に閏日を入力すると、翌月の初日に繰り越されます。 月日が明示的に範囲を越えている場合(例：WEEK("2023/3/99"))は、エラーになります。		
使用例1	NUM = WEEK("2013/07/15") 変数NUMに2を代入します。		
使用例2	DATETIME A A = CDATETIME (DATE\$ (1, 0, 0)) NUM = WEEK (A) 変数NUMに、来年の曜日を代入します。		
使用例3	NUM = WEEK("2013/07/15") LIST ARY\$ ARY\$ = ["Sunday"; "Monday"; "Tuesday"; "Wednesday"; "Thursday"; "Friday"; "Saturday"] PRINT "曜日："; ARY\$(NUM - 1) WEEKの戻り値を添字に、曜日を文字列に求める事例です。		

2.4 型の宣言や変換に関する関数・命令

2.4.1 BOOL

命令			
機能	論理型変数を宣言します。		
書式	BOOL <①変数名> [= <②真偽値>] [, <①変数名> …]		
パラメータ	①	<変数名>	変数名
	論理型で宣言する変数名を指定します。		
	②	<真偽値>	真偽値
変数が真(TRUE)か偽(FALSE)かを指定します。			
使用例1	<pre> BOOL A = TRUE, B = FALSE IF A THEN PRINT "AはTrueです。" ELSE PRINT "AはFalseです。" END IF IF B THEN PRINT "BはTrueです。" ELSE PRINT "BはFalseです。" END IF ----- 実行結果 AはTrueです。 BはFalseです。 最初に指定した変数A,Bの真偽をそれぞれIf文で評価し、処理を分岐させています。 </pre>		
使用例2	<pre> BOOL A, B A = TRUE B = FALSE 変数宣言と真偽の指定を分ける事もできます。 </pre>		
使用例3	<pre> BOOL A(1) A(0) = TRUE A(1) = FALSE ? A ----- 実行結果 [TRUE , FALSE] 変数名の後に(数値)とすることで配列変数を宣言することができます。 </pre>		

2.4.2 CONST

命令			
機 能	定数を宣言します。		
書 式	CONST <①定数名> = <②定数値> [, <①定数名> ...]		
パラ メータ	①	<定数名>	定数名
	定数名を指定します。		
	②	<定数値>	数値
	定数の値を指定します。		
使用例1	CONST A = 100 定数Aを宣言し、100で初期化します。 以降の処理でAの値は変更できません。		
使用例2	CONST A(1) = [100 ; 200] 定数配列A(要素数2)を宣言し、それぞれ100, 200で初期化します。		

2.4.3 CONST BOOL

命令			
機 能	論理型定数を宣言します。		
書 式	CONST BOOL <①定数名> = <②定数値> [, <①定数名> ...]		
パラ メータ	①	<定数名>	定数名
	論理型定数名を指定します。		
	②	<定数値>	真偽値
	定数の値を指定します。		
使用例1	CONST BOOL A = TRUE 定数Aを宣言し、TRUEで初期化します。 以降の処理でAの値は変更できません。		
使用例2	CONST BOOL A(1) = [TRUE ; FALSE] 定数配列A(要素数2)を宣言し、それぞれTRUE, FALSEで初期化します。		

2.4.4 CONST DATETIME

命令			
機 能	日付時刻型定数を宣言します。		
書 式	CONST DATETIME <①定数名> = <②定数値> [, <①定数名> ...]		
パラ メータ	①	<定数名>	定数名
	日付時刻型定数名を指定します。		
	②	<定数値>	数値
定数の値を指定します。			
使用例1	CONST DATETIME A = CDATETIME (DATE\$) 定数Aを宣言し、本日0時で初期化します。 以降の処理でAの値は変更できません。		
使用例2	CONST DATETIME A(1) = [CDATETIME (DATE\$) ; CDATETIME (DATE\$) + 1] 定数配列A(要素数2)を宣言し、それぞれ本日, 明日で初期化します。		

2.4.5 DATETIME

命令			
機 能	日付時刻型変数を宣言します。		
書 式	DATETIME <①変数名> [, <①変数名> ...]		
パラ メータ	①	<変数名>	変数名
備 考	<p>日付時刻型変数名を指定します。</p> <ul style="list-style-type: none"> 日付時刻型とは、日付と時刻を表現する特殊なデータ型です。内部としては整数部が日付、小数部が時刻を表します。倍精度実数に代入することができます。 STR\$関数 (→2.3.50 STR\$) で文字列に変換すると、「年/月/日 時:分:秒」となります。 同様に、PRINT命令 (→2.7.23 PRINT, ?) で表示する時、「年/月/日 時:分:秒」として表示します。 . 		
注 意	<ul style="list-style-type: none"> 日付時刻型は1を西暦1年1月1日 0:0:0としています。暦の換算はグレゴリオ暦を採用しています。 整数部を増減すると日付の操作ができます。元の値に+1すると1日進み、-1すると1日戻ります。 小数部は時間を表すため、1時間は、$1.0 \div 24$に相当し、1分は、$1.0 \div (24 \times 60)$に相当します。 例：0.5を指定すると12時間となります。 時間の演算をする場合はCDATETIME関数 (→2.3.7 CDATETIME) を使用してください。 例：現在時刻から1時間戻す場合 DATETIME A = CDATETIME (DATE\$ ()+" "+TIME\$ ()) ' 本日の日付時刻を代入 A = A - CDATETIME ("01:00:00") ' 1時間戻す 以下の関数が、引数に日付時刻型を受け取れるようになっています。 STR\$, VAL, CDATETIME, WEEK 		
使用例1	DATETIME A, B 日付時刻型変数A, Bを宣言します。		
使用例2	DATETIME A(1) 日付時刻型の配列変数A(要素数2) を宣言します。		

2.4.6 DEFINE STRUCT～END STRUCT

命令		
機能書式	構造体型の定義を宣言します。 DEFINE STRUCT <①構造体名> [<②変数の宣言文>] <③変数名> ... END STRUCT	
パラメータ	①	<構造体名> 構造体名を指定します。
	②	<変数の宣言文> 変数名の型を明示的に宣言する文を指定します。 指定可能な宣言文は、DIM文(→「2.5.10 DIM」)、BOOL文(→「2.4.1 BOOL」)、STRUCT文(→「2.4.16 STRUCT」)、DATETIME文(→「2.4.5 DATETIME」)、LIST文(→「2.5.15 LIST」)、DICT文(→「2.5.5 DICT」)です。(各文の記述書式は、それぞれの説明を参照ください)
	③	<変数名> 構造体メンバの変数名を指定します。
	命令	
注意	<ul style="list-style-type: none"> この命令で定義した構造体を使用するには、「2.4.16 STRUCT」で構造体変数を宣言して使用します。 構造体自体のPRINT、構造体同士の代入はできません。 既存のコマンド名や関数名を、構造体型名や変数名に指定することはできません。 異なる型の同じ名前のメンバー名を定義しないでください。 	
使用例1	<pre> DEFINE STRUCT SITE ID ADDR\$ END STRUCT STRUCT SITE OFFICE OFFICE.ID = 1 OFFICE.ADDR\$ = "HIROSHIMA" PRINT OFFICE.ID, OFFICE.ADDR\$ SITEという名前の構造体を宣言し、メンバ ID, ADDR\$ を指定しています。 SITE型の構造体型変数OFFICEを宣言します。 値の格納後、PRINTにて値を表示しています。 </pre>	
使用例2	<pre> DEFINE STRUCT SITE ID ADDR\$ END STRUCT STRUCT SITE OFFICE(1) OFFICE(0).ID = 1 : OFFICE(0).ADDR\$ = "HIROSHIMA" OFFICE(1).ID = 2 : OFFICE(2).ADDR\$ = "OITA" PRINT OFFICE(0).ID, OFFICE(0).ADDR\$ PRINT OFFICE(1).ID, OFFICE(1).ADDR\$ SITEという名前の構造体を宣言し、メンバ ID, ADDR\$ を指定しています。 SITE型の構造体型変数OFFICEを配列で宣言します。 </pre>	

	値の格納後、PRINTにて値を表示しています。
使用例3	<pre>DEFINE STRUCT SITE ID DIM ADDR\$(2) ' メンバ変数に配列を宣言する例です END STRUCT STRUCT SITE V V.ID = 123 V.ADDR\$ = ["0"; "1"; "2"] ' あるいは V.ADDR\$(0) = "0" V.ADDR\$(1) = "1" V.ADDR\$(2) = "2"</pre> <p>構造体のメンバ ADDR\$を配列に宣言する例です。</p>
使用例4	<pre>DEFINE STRUCT SITE ID DIM ADDR\$ END STRUCT DEFINE STRUCT GRP STRUCT SITE ARY(2) END STRUCT STRUCT GRP V V.ARY(0).ID = 123 V.ARY(0).ADDR\$ = "Hello" V.ARY(1).ID = 456 V.ARY(1).ADDR\$ = "AJAN" V.ARY(2).ID = 789 V.ARY(2).ADDR\$ = "World"</pre> <p>構造体 SITEを内包する、構造体 GRP を宣言する例です。</p>

2.4.7 ENUM～END ENUM

命令			
機能		列挙型定数を宣言します。	
書式		<pre> ENUM <①定数名> [= <②整数値>] ... END ENUM </pre>	
パラメータ	①	<定数名>	定数名
	列挙型で宣言する定数名を指定します。		
	②	<整数値>	数値
	列挙型定数の値を指定します。 省略した場合は、先頭の定数ならば0で初期化されます。 先頭以外ならば、前の定数に1加算した値で初期化されます。		
使用例1		<pre> ENUM A = 1 B = 2 END ENUM </pre> 定数A,Bを列挙型で宣言します。Aを1、Bを2で初期化します。	
使用例2		<pre> ENUM A B END ENUM </pre> 定数A,Bを列挙型で宣言します。Aを0、Bを1で初期化されます。	
使用例3		<pre> ENUM A = 10 B C END ENUM </pre> 定数A,B,Cを列挙型で宣言します。Aを10、Bを11、Cを12で初期化されます。	

2.4.8 LOCAL

命令			
機能	ローカル変数を宣言します。		
書式	LOCAL <①変数名> [= <②代入式>] [, <①変数名> …]		
パラメータ	①	<変数名>	変数名
	ローカル変数名を指定します。		
	②	<代入式>	値
	ローカル変数へ値を代入します。		
注意	<ul style="list-style-type: none">• SUB～END SUB、FUNCTION～END FUNCTION、DEFINE THREAD～END THREAD等のサブルーチン内でのみ使用できます。• ローカル変数はサブルーチンを抜けると解放され、不定の状態になります。• ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」）		
使用例1	LOCAL A, B ローカル変数A, Bを宣言します。		
使用例2	LOCAL A(1) 要素数2のローカル配列変数Aを宣言します。		

2.4.10 LOCAL CONST

命令			
機能	ローカル定数を宣言します。		
書式	LOCAL CONST <①定数名> = <②定数値> [, <①定数名> ...]		
パラメータ	①	<定数名>	定数名
	ローカル定数名を指定します。		
	②	<定数値>	数値
ローカル定数の値を指定します。			
注意	<ul style="list-style-type: none">• SUB～END SUB、FUNCTION～END FUNCTION、DEFINE THREAD～END THREAD等のサブルーチン内でのみ使用できます。• ローカル定数はサブルーチンを抜けると解放され、不定の状態になります。• ローカル定数を宣言せずに、ローカル定数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」）		
使用例1	LOCAL CONST A = 100 ローカル定数Aを宣言し、100で初期化します。 以降の処理でAの値は変更できません。		
使用例2	LOCAL CONST A(1) = [100; 200] 要素数2のローカル配列定数Aを宣言し、各要素を100, 200で初期化します。 以降の処理でAの値は変更できません。		

2.4.11 LOCAL CONST BOOL

命令		
機 能	論理型ローカル定数を宣言します。	
書 式	LOCAL CONST BOOL <①定数名> = <②真偽値> [, <①定数名> …]	
パラ メータ	①	定数名
	論理型ローカル定数名を指定します。	
	②	真偽値
ローカル定数が真(TRUE)か偽(FALSE)かを指定します。		
注 意	<ul style="list-style-type: none"> • SUB～END SUB、FUNCTION～END FUNCTION、DEFINE THREAD～END THREAD 等のサブルーチン内でのみ使用できます。 • ローカル定数はサブルーチンを抜けると解放され、不定の状態になります。 • ローカル定数を宣言せずに、ローカル定数にアクセスしようとししないでください。(参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」) 	
使用例1	LOCAL CONST BOOL A = TRUE 論理型ローカル定数Aを宣言し、TRUEで初期化します。 以降の処理でAの値は変更できません。	
使用例2	LOCAL CONST BOOL A(1) = [TRUE; FALSE] 論理型ローカル配列定数Aを宣言し、各要素をTRUE, FALSEで初期化します。 以降の処理でAの値は変更できません。	

2.4.12 LOCAL CONST DATETIME

命令			
機能	日付時刻型ローカル定数を宣言します。		
書式	LOCAL CONST DATETIME <①定数名> = <②定数値> [, <①定数名> …]		
パラメータ	①	<定数名>	定数名
	日付時刻型ローカル定数名を指定します。		
	②	<定数値>	数値
ローカル定数の内容を指定します。			
注意	<ul style="list-style-type: none"> SUB～END SUB、FUNCTION～END FUNCTION、DEFINE THREAD～END THREAD 等のサブルーチン内でのみ使用できます。 ローカル定数はサブルーチンを抜けると解放され、不定の状態になります。 ローカル定数を宣言せずに、ローカル定数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 		
使用例1	LOCAL CONST DATETIME A = CDATETIME (DATE\$) 日付時刻型ローカル定数Aを宣言し、本日で初期化します。 以降の処理でAの値は変更できません。		
使用例2	LOCAL CONST DATETIME A(1) = [CDATETIME (DATE\$); CDATETIME (DATE\$)+1] 日付時刻型ローカル配列定数Aを宣言し、各要素を本日、明日で初期化します。 以降の処理でAの値は変更できません。		

2.4.13 LOCAL DATETIME

命令			
機能	日付時刻型ローカル変数を宣言します。		
書式	LOCAL DATETIME <①変数名> [, <①変数名> …]		
パラメータ	①	<変数名>	変数名
	日付時刻型ローカル変数名を指定します。		
注意	<ul style="list-style-type: none"> SUB～END SUB、FUNCTION～END FUNCTION、DEFINE THREAD～END THREAD 等のサブルーチン内でのみ使用できます。 ローカル変数はサブルーチンを抜けると解放され、不定の状態になります。 ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 		
使用例1	LOCAL DATETIME A, B 日付時刻型ローカル変数A, Bを宣言します。		
使用例2	LOCAL DATETIME A(1) 要素数2の日付時刻型ローカル配列変数Aを宣言します。		

2.4.14 LOCAL ENUM～END ENUM

命令			
機能	列挙型ローカル定数を宣言します。		
書式	LOCAL ENUM <①定数名> [= <②整数値>] : END ENUM		
パラメータ	①	<定数名>	定数名
	列挙型ローカル定数名を指定します。		
	②	<整数値>	数値
列挙型ローカル定数の値を指定します。 省略した場合は、先頭の定数ならば0で初期化されます。 先頭以外ならば、前の定数に1加算した値で初期化されます。			
注意	<ul style="list-style-type: none"> ・ SUB～END SUB、FUNCTION～END FUNCTION、DEFINE THREAD～END THREAD 等のサブルーチン内でのみ使用できます。 ・ ローカル定数はサブルーチンを抜けると解放され、不定の状態になります。 ・ ローカル定数を宣言せずに、ローカル定数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 		
使用例1	LOCAL ENUM A = 1 B = 2 END ENUM 定数A,Bを列挙型で宣言します。Aを1、Bを2で初期化します。		
使用例2	LOCAL ENUM A B END ENUM 定数A,Bを列挙型で宣言します。Aを0、Bを1で初期化されます。		
使用例3	LOCAL ENUM A = 10 B C END ENUM 定数A,B,Cを列挙型で宣言します。Aを10、Bを11、Cを12で初期化されます。		

2.4.15 LOCAL STRUCT

命令			
機能	構造体型ローカル変数を宣言します。		
書式	LOCAL STRUCT <①構造体名> <②変数名> [, <③変数名>…]		
パラメータ	①	<構造体名>	構造体名
	構造体名を指定します。		
	②	<変数名>	変数名
構造体型ローカル変数名を指定します。			
備考	STRUCT (→「2.4.16 STRUCT」)		
注意	<ul style="list-style-type: none"> ・ SUB～END SUB、FUNCTION～END FUNCTION、DEFINE THREAD～END THREAD 等のサブルーチン内でのみ使用できます。 ・ 構造体名は、「2.4.6 DEFINE STRUCT～END STRUCT」で定義します。 ・ 構造体自体のPRINTはできません。 ・ 構造体配列の部分指定、範囲指定を利用した代入 (→2.5 配列に関する関数・命令) はできません。要素数が同じかつ構造体型が同じ配列同士の代入や、配列の1要素同士の代入は可能です。 ・ ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。(参考: 基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」) 		
使用例1	<pre> DEFINE STRUCT SITE ID ADDR\$ END STRUCT SUB OFFICE_SUB LOCAL STRUCT SITE OFFICE OFFICE.ID = 1 OFFICE.ADDR\$ = "HIROSHIMA" PRINT OFFICE.ID, OFFICE.ADDR\$ END SUB </pre> <p>IDとADDR\$をメンバ変数に持つ構造体SITEを宣言し、SITE型の構造体型ローカル変数OFFICEを宣言します。</p> <p>メンバ変数IDに1、ADDR\$に"HIROSHIMA"を設定し、同じ内容を画面に出力します。</p>		
使用例2	<pre> DEFINE STRUCT SITE ID ADDR\$ END STRUCT SUB OFFICE_SUB LOCAL STRUCT SITE OFFICE(1) OFFICE(0).ID = 1 OFFICE(0).ADDR\$ = "HIROSHIMA" OFFICE(1).ID = 2 OFFICE(1).ADDR\$ = "OITA" PRINT OFFICE(0).ID, OFFICE(0).ADDR\$ PRINT OFFICE(1).ID, OFFICE(1).ADDR\$ END SUB </pre> <p>IDとADDR\$をメンバ変数に持つ構造体SITEを宣言し、SITE型の構造体型ローカル変数OFFICEを宣言します。</p> <p>各メンバ変数へ値を設定し、同じ内容を画面に出力します。</p>		

2.4.17 VARTYPE

関数			
機 能	変数の型を調べます。		
書 式	<(戻り値)型の値> = VARTYPE(<①変数名>)		
戻り値	戻り値	<型の値>	数値
	以下の値の組み合わせ(OR)で得られます。 例えば、倍精度実数型の配列の場合、&H10(倍精度実数型) OR &H8000000(配列) = &H8000010 が得られます。		
	戻り値	解説	
	&h01	文字列型	
	&h02	単精度整数型	
	&h04	倍精度整数型	
	&h08	単精度実数型	
	&h10	倍精度実数型	
	&h20	論理型	
	&h40	列挙値型	
	&h80	構造体型	
	&h1000	日付時刻型	
	&h2000	オブジェクト型	
	&h8000000	配列	
	&h20000000	連想配列	
	&h40000000	可変長配列	
パラ メータ	①	<変数名>	変数名
	調べたい変数名を記述します。		
使用例1	A\$="hello" ? VARTYPE(A\$) 1が得られます。		
使用例2	DIM A\$(10) ? HEX\$(VARTYPE(A\$)) 8000001 が得られます。配列の場合は、組み合わせの結果が得られます。		
使用例3	LIST A# ? HEX\$(VARTYPE(A#)) 48000010 が得られます。使用例2と同様に、組み合わせの結果が得られます。		

2.5 配列に関する関数・命令

2.5.1 ARRAY2DICT

命令			
機 能	2次元の文字列配列から連想配列にセットします。		
書 式	ARRAY2DICT <①連想配列変数名>, <②2次元文字列配列>		
パラ メータ	①	<連想配列変数名>	変数名
	連想配列変数名を指定します。		
	②	<2次元文字列配列>	配列
	連想配列に対してセットしたい2次元の文字列配列を指定します。 1次元目が、連想配列のキー数。 2次元目が、キー名と値を文字列化した2つの要素数です。		
備 考	<ul style="list-style-type: none"> ・ 構造体の連想配列を指定できません。 ・ 2次元文字列配列の引数には、DICT2ARRAY\$で得られる配列形式と同じものを指定します。 ・ 連想配列の型が数値の場合、2次元文字列配列中の文字列を数値に変換した上でセットします。VAL関数による数値変換と同じ挙動を行います。 		
使用例	<pre> DIM S\$(2,1) S\$ = [{"hoge"; "123"}, {"fuga"; "456"}, {"test"; "789"}] DICT ARY ARRAY2DICT ARY, S\$ PRINT ARY </pre> <p>以下のような表示結果が得られます。</p> <pre> (test)=789, (fuga)=456, (hoge)=123 </pre>		

2.5.2 CDIM

関数			
機 能	配列の次元数を返します。		
書 式	<(戻り値)次元数> = CDIM(<①配列変数名>)		
戻り値	戻り値	<次元数>	数値
	配列の次元数を返します。		
パラ メータ	①	<配列変数名>	変数名
	対象となる配列変数名を指定します。 指定した配列変数の次元数が得られます。 配列変数でない場合、0が返ります。		
使用例	DIM A(2, 1) ? CDIM(A) 2 2次元配列に対して、CDIM関数を呼び出すと、2が返ります。		

2.5.3 CLEAR_DICT

命令			
機 能	連想配列の内容をクリアします。		
書 式	CLEAR_DICT <①連想配列変数名> [, <①連想配列変数名> ...]		
パラ メータ	①	<連想配列変数名>	変数名
	クリアしたい連想配列変数名を指定します。		
使用例	CLEAR_DICT A 連想配列変数Aの内容をクリアします。		

2.5.4 DEL_DICT_KEY

命令		
機 能	連想配列に対して、指定したキーに紐付けされたデータを削除します。	
書 式	DEL_DICT_KEY <①連想配列変数名>, "<②キー名>" [, <③強制削除フラグ>]	
パラ メータ	①	変数名
	<連想配列変数名> 連想配列変数名を指定します。	
	②	文字列
	<キー名> 連想配列に対して削除したいキー名を、文字列形式で与えます。	
	③	真偽値
	<強制削除フラグ> FALSEを指定すると、連想配列にキー名が無いと、エラーになります。 TRUEを指定すると、連想配列にキー名が無くとも、エラーになりません。	
使用例	DICT A A("HOGE") = 123 A("FUGA") = 456 DEL_DICT_KEY A, "FUGA" 連想配列 Aに対して、"FUGA"というキーに紐付けされたデータを削除します。	

2.5.5 DICT

命令			
機 能	連想配列変数を宣言します。		
書 式	DICT <①変数名> [, <①変数名> ...]		
パラ メータ	①	<変数名>	変数名
備 考	<p>連想配列変数名を指定します。</p> <p>連想配列変数とは、配列の添字に数値以外のキーを指定可能なデータ型です。キーに対して、一つの値を格納する事ができ、例えば書き込みは以下のように記述します。</p> <pre>DICT ARY ARY("TEST1") = 123 ARY("TEST2") = 456</pre> <p>ここでは、丸カッコ内の"TEST1"がキーであり、代入演算子の後の「123」が値です。読み出しは、キーを指定する事で得られます。</p> <pre>PRINT ARY("TEST1") ' 123が得られます PRINT ARY("TEST2") ' 456が得られます。</pre> <p>また、連想配列の配列を定義することができます。</p> <p>これにより、連想配列の固定長配列を表現することができます。</p> <pre>DICT A(3) ' 要素数4の連想配列の配列</pre> <p>配列部分とキーは丸カッコで分けて記載します。</p> <pre>ARY(20)("TEST1") = 123 ' 感覚的には20行目の"TEST1"に代入</pre> <p>連想配列に保持することの出来る値の型は、変数名の型修飾に由来します。</p> <p>例えば、「ARY\$」というように文字列型修飾を付加すると、文字列値が保持できます。</p>		
注 意	<ul style="list-style-type: none"> 連想配列変数に格納可能な値は、名前の型修飾に由来します。 連想配列は、普通の配列のような添字の順序の概念や要素数の概念が存在しません。従って、PRINT文などで連想配列を表示した際、表示される順番は0Sに依存します。 連想配列のキーの情報は、GET_DICT_KEYS\$でキーの一覧を、HAS_DICT_KEYでキーが存在するか否かを得る事ができます。 得られたキーの配列は、ONEDIM SORT\$ 関数で整列できます。 <変数名>に指定可能な型は、数値、文字列をサポートしています。 連想配列の可変長配列を定義する場合はLIST DICTを利用してください。 		
使用例1	<pre>DICT A\$, B\$</pre> <p>連想配列変数A\$, B\$を宣言します。</p>		
使用例2	<pre>DICT A\$(1)</pre> <p>連想配列の配列変数(要素数2)A\$を宣言します</p>		

2.5.6 DICT BOOL

命令			
機 能	論理型連想配列変数を宣言します。		
書 式	DICT BOOL <①変数名> [, <①変数名> ...]		
パラ メータ	①	<変数名>	変数名
注 意	<p>論理型連想配列変数名を指定します。</p> <ul style="list-style-type: none"> 連想配列の情報は、GET_DICT_KEY\$\$でキーの一覧を、HAS_DICT_KEYでキーが存在するか否かを得る事ができます。 可変長の連想配列変数を定義する場合はLIST DICT BOOLを利用してください。 連想配列は、普通の配列のような添字の順序の概念や要素数の概念が存在しません。従って、PRINT文などで連想配列を表示した際、表示される順番はOSに依存します。 		
使用例1	DICT BOOL A, B B("mogu") = FALSE B("helo") = TRUE 論理型連想配列変数A, Bを宣言します。		
使用例2	DICT BOOL A(1) 論理型連想配列の配列変数(要素数2)Aを宣言します。		

2.5.7 DICT DATETIME

命令			
機 能	日付時刻型連想配列変数を宣言します。		
書 式	DICT DATETIME <①変数名> [, <①変数名> ...]		
パラ メータ	①	<変数名>	変数名
注 意	<p>日付時刻型連想配列変数名を指定します。</p> <ul style="list-style-type: none"> 連想配列の情報は、GET_DICT_KEY\$\$でキーの一覧を、HAS_DICT_KEYでキーが存在するか否かを得る事ができます。 可変長の連想配列変数を定義する場合はLIST DICT DATETIMEを利用してください。 連想配列は、普通の配列のような添字の順序の概念や要素数の概念が存在しません。従って、PRINT文などで連想配列を表示した際、表示される順番はOSに依存します。 		
使用例1	DICT DATETIME A, B B("mogu") = CDATETIME("2019/9/20 10:23:33") B("helo") = CDATETIME("2019/9/24 8:0:0") 日付時刻型連想配列変数A, Bを宣言します。		
使用例2	DICT DATETIME A(1) 日付時刻型連想配列の配列変数(要素数2)Aを宣言します。		

2.5.8 DICT STRUCT

命令		
機能	構造体型連想配列変数を宣言します。	
書式	DICT STRUCT <①構造体名> <②変数名> [, <②変数名> ...]	
パラメータ	①	<構造体名> 構造体名
	DEFINE STRUCTで定義した構造体名を指定します。	
	②	<変数名> 変数名
構造体型連想配列変数名を指定します。		
備考	要素の各メンバを指定する場合は、「変数名(キー).メンバ名」で指定します。	
注意	<ul style="list-style-type: none"> ・ 構造体名は、「2.4.6 DEFINE STRUCT～END STRUCT」で定義します。 ・ 連想配列の情報は、GET_DICT_KEYS\$でキーの一覧を、HAS_DICT_KEYでキーが存在するか否かを得る事ができます。 ・ 可変長の連想配列変数を定義する場合はLIST DICT STRUCTを利用してください。 ・ 連想配列は、普通の配列のような添字の順序の概念や要素数の概念が存在しません。従って、PRINT文などで連想配列を表示した際、表示される順番は0Sに依存します。 	
使用例1	<pre> DEFINE STRUCT RECORD ID ADDR\$ END STRUCT DICT STRUCT RECORD A, B A("HEADOFFICE").ID = 1 A("HEADOFFICE").ADDR\$ = "TOKYO, JAPAN" </pre> <p>構造体型連想配列変数A, Bを宣言します。</p>	
使用例2	<pre> DICT STRUCT RECORD A(1) </pre> <p>構造体型連想配列の配列変数(要素数2)Aを宣言します。</p>	

2.5.9 DICT2ARRAY\$

関数		
機 能	連想配列から2次元の文字列配列として得ます。	
書 式	<(戻り値) 文字列配列> = DICT2ARRAY\$(<①連想配列変数名>)	
戻り値	戻り値	<文字列配列>
	配列	
	<p>連想配列を2次元の文字列配列に変換した値を返します。</p> <p>1次元目が、連想配列のキー数。</p> <p>2次元目が、キー名と値を文字列化した2つの要素数です。</p>	
パラ メータ	①	<連想配列変数名>
	変数名	
	<p>連想配列変数名を指定します。</p> <p>指定できるのは、文字列ないしは数値の連想配列です。</p>	
備 考	<ul style="list-style-type: none"> ・ 構造体の連想配列を指定できません。 ・ 得た文字列配列から連想配列にセットするには、ARRAY2DICTを使用します。 	
使用例	<pre>DICT A A("HOGE") = 123 A("FUGA") = 456 PRINT DICT2ARRAY\$(A)</pre>	
	<p>以下の表示結果が得られます。</p> <pre>[[HOGE, 123], [FUGA, 456]]</pre>	

2.5.10 DIM

命令			
機能	配列変数を宣言します。 カンマ区切りで複数宣言することもできます。		
書式	DIM <①変数名>(<②添字最大値> [, <②添字最大値> ...]) [= <③代入式>] [, <①変数名>(<②添字最大値> [, <②添字最大値> ...]) ...]		
パラメータ	①	<変数名>	変数名
	配列で宣言する変数名を指定します。		
	②	<添字最大値>	数値
	配列の添字の最大値を指定します。		
	③	<代入式>	値
	配列変数へ値を代入します。		
備考	<p>配列の代入方法として下記が使用できます。</p> <p>配列変数名 = 値の列記 配列変数名 = 配列変数名 配列変数名(添字の列記) = 値の列記 配列変数名(添字の列記) = 配列変数名(添字の列記)</p> <p>値/添字の列記 $:= \left[\begin{array}{c} \text{式} > \text{TO} <\text{式}> \end{array} \right] \left[\begin{array}{c} \\ \\ \\ ; \\ \end{array} \right] \dots$</p> <p>値の列記時の区切りは,","も";"も同等に扱います。添字の列記は、同次元の<式の>区切りは";"を使い、次元の区切りでは","を使用してください。</p> <p>T0指定は、ステップが1となります。</p> <p>xx T0 yy</p> <p>xx < yyの場合 ステップが1でインクリメントされます。</p> <p>xx > yyの場合 ステップが1でデクリメントされます。</p>		
注意	<ul style="list-style-type: none">・宣言済みの配列変数は再度宣言できません。・本命令はグローバルの数値型変数、文字列型変数に適用できます。それ以外は各命令の使用例を参照してください。・()内は添字の最大値です。配列数ではありません。 例: DIM A(2) と2を指定した場合、A(0), A(1), A(2) の3つが宣言されます。・配列の添字最小値は0です。・添字に指定できる最大値は、単精度整数の最大(2,147,483,646)までです。・添字は必ず指定する必要があります。AJAN では、暗黙的に定義されません。・次元に指定できる最大値は、1次元に換算して、添字の最大値までです。・実際に配列が確保できるかどうかは、使用しているコンピュータのメモリ容量の制限を受けます。		
使用例1	<pre>DIM a(3), b(3) b(0)=10 b(1)=20 b(2)=30 a=b PRINT a(0), a(1), a(2) ----- 実行結果 10 20 30</pre>		

	配列a, bをそれぞれ添字最大値「3」で宣言しています。 配列bへ値をセットしています。 配列bの値を配列aへ代入しています。 配列aの値を画面表示しています。												
使用例2	<pre>DIM a(5) a= [10 to 15] PRINT a -----</pre> <p>実行結果 [10;11;12;13;14;15]</p> <p>配列aを添字最大値5で宣言しています。 10～15の値(10, 11, 12, 13, 14, 15)を配列の先頭から順に代入しています。 配列aの値を全て画面表示しています。</p>												
使用例3	<pre>DIM a(5) a= [1, 2, 7 to 9, 20] PRINT a -----</pre> <p>実行結果 [1;2;7;8;9;20]</p> <p>値代入時に、値をカンマ区切りとステップによって指定する例です。</p>												
使用例4	<pre>DIM a(5) a= [10;20;30;40;50;60] PRINT a -----</pre> <p>実行結果 [10;20;30;40;50;60]</p> <p>値代入時に、値をセミコロンで区切って指定する例です。</p>												
使用例5	<pre>DIM a(5) a=[10, 20, 30, 40, 50, 60] PRINT a -----</pre> <p>実行結果 [10;20;30;40;50;60]</p> <p>値代入時に、値をカンマで区切って指定する例です。</p>												
使用例6	<pre>DIM a(4), b(2, 4) b(1;2, 1;2;3;4) =[1;2;3;4, 10;11;12;13] a(1;2;3;4) = [100;200;300;400] PRINT b(1, 1), b(1, 2), b(1, 3), b(1, 4) PRINT b(2, 1), b(2, 2), b(2, 3), b(2, 4) PRINT a(1), a(2), a(3), a(4) -----</pre> <p>実行結果</p> <table><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>10</td><td>11</td><td>12</td><td>13</td></tr><tr><td>100</td><td>200</td><td>300</td><td>400</td></tr></table> <p>二次元配列を用いて値の格納、使用を行う例です。</p>	1	2	3	4	10	11	12	13	100	200	300	400
1	2	3	4										
10	11	12	13										
100	200	300	400										

2.5.11 GET_DICT_KEYS\$

関数			
機 能	連想配列のキー一覧を文字列配列として得ます。		
書 式1	<(戻り値)キー一覧配列> = GET_DICT_KEYS\$(<①連想配列変数名>)		
書 式2	<(戻り値)キー一覧配列> = GET_DICT_KEYS(<①連想配列変数名>)		
	古い書式です。GET_DICT_KEYS\$ と記述する事を奨めます。		
戻り値	戻り値	<キー一覧配列>	配列
	指定した連想配列のキー一覧を、1次元の文字列配列にして返します。		
パラメータ	①	<連想配列変数名>	変数名
	連想配列のキー一覧を文字列変数として得る対象を指定します。		
注 意	<ul style="list-style-type: none"> 空の連想配列か否かを判定するには、LDICT 関数で、0(空)か0以外かで判断してください。空の連想配列に対して、GET_DICT_KEYS\$ を呼び出した結果は、保証されません。 		
備 考	<ul style="list-style-type: none"> 得られる文字列配列は、順序保障されません。得られたキーの配列は、ONEDIM SORT\$ 関数で整列できます。2020/05より、GET_DICT_KEYS\$ が推奨されます。 		
使用例1	<pre> DICT A\$ A\$("KEY1") = "VALUE1" A\$("KEY2") = "VALUE2" A\$("KEY3") = "VALUE3" PRINT GET_DICT_KEYS\$(A\$) ' "KEY1", "KEY2", "KEY3" の文字列配列が得られます。(文字列配列の内容は順不同です) </pre>		
使用例2	<pre> DICT A\$ A\$("alpha") = "val1" A\$("beta") = "val2" A\$("gamma") = "val3" LIST K\$ K\$ = GET_DICT_KEYS\$(A\$) ? K\$ ' GET_DICT_KEYS\$で取得したキーの配列は順番が保証されていません K\$ = ONEDIM SORT\$(K\$) ? K\$ ' ONEDIM SORT\$ で昇順に整列できます GET_DICT_KEYS\$ で得られたキーの文字列配列を、整列し直す例です。 </pre>		

2.5.12 HAS_DICT_KEY

関数			
機 能	連想配列に対して、指定したキーが存在するか否かを得ます。		
書 式	<(戻り値)キーの有無> = HAS_DICT_KEY (<①連想配列変数名>, "<②キー名>")		
戻り値	戻り値	<キーの有無>	真偽値
	連想配列のキー一覧に対して、指定したキー名が存在すればTRUE、存在しなければFALSEを返します。		
パラメータ	①	<連想配列変数名>	変数名
	連想配列変数名を指定します。		
	②	<キー名>	文字列
	連想配列に対して探索を行いたいキー名を、文字列形式で与えます。		
使用例	<pre> DICT A A("ELEMENT") = 123 </pre>		

	? HAS_DICT_KEY(A, "ELEMENT")
--	------------------------------

	連想配列 Aに対して、“ELEMENT”というキーが存在するかを表示します。
--	--

2.5.13 LDICT

関数			
機 能	連想配列に対して登録されているキーの総数を取得します。		
書 式	<(戻り値)キーの総数> = LDICT(<①連想配列変数名>)		
戻り値	戻り値	<キーの総数>	数値
	連想配列に対して登録されているキーの総数を取得します。		
パラメータ	①	<連想配列変数名>	変数名
	DICT命令などで宣言した連想配列変数名を指定します。		
使用例	DICT A\$? LDICT(A\$) 連想配列変数A\$のキーの総数を取得します。この場合は宣言直後なので0が表示されます。		

2.5.14 LDIM

関数			
機 能	配列の要素数を返します。		
書 式	<(戻り値)配列の要素数> = LDIM(<①配列変数名>[, <②次元>])		
戻り値	戻り値	<配列の要素数>	数値
	配列の要素数を返します。		
パラメータ	①	<配列変数名>	変数名
	対象となる配列変数名を指定します。		
	②	<次元>	数値
	1から始まる、要素数を返す次元を指定します。		
	指定しない場合、全要素数を返します。		
備 考	<ul style="list-style-type: none"> 存在しない次元を指定するとエラーが返ります。 指定した次元の添字最大値を得たい場合は「UBOUND」を使用してください。 		
使用例1	DIM A\$(5) L=LDIM(A\$) 変数 L に配列A\$の要素数 6 を代入します。		
使用例2	DIM A\$(2, 5) L=LDIM(A\$) <次元>を指定しない場合、変数 L に配列A\$の全要素数 18 を代入します。		
使用例3	DIM A\$(10, 5) L=LDIM(A\$, 2) 配列A\$のうち、指定した <次元> 2 の要素数 6 を変数 L に代入します。		

2.5.15 LIST

命令			
機能	可変長配列変数を宣言します。 この命令で宣言した変数は右辺の要素数によって自動で要素数を変更します。		
書式	LIST <①変数名>[, <①変数名> , ...]		
パラメータ	①	<変数名>	変数名
注意	<p>可変長配列変数名を指定します。</p> <ul style="list-style-type: none"> ・ 宣言時に予め要素数を決めることはできません。 ・ 宣言時に値を代入することはできません。 ・ 変数の型は通常の変数と同じように決定されます。 ・ 本命令の直後は要素数1の配列として使用することができます。 DIM命令で例えるなら「DIM 変数名 (0)」と宣言したのと同じです。 ・ 宣言後に、配列の大きさを変更するには、REDIM命令で変更するか、 ONEDIM INSERT, ONEDIM REMOVE で 要素を挿入／削除します。 ・ 実行時の要素数より大きい添字の要素を参照することはできません(参照による拡張は行われません)。 <p>例:</p> <pre>LIST A\$ ' 可変長配列の宣言 B\$ = "A, B, C, D" A\$ = SPLIT\$(B\$, ", ") ' 要素数は4 A\$(10000) = "TEST" ' 要素10000は存在しないのでエラー</pre>		
使用例1	<pre>LIST A\$ A\$ = ["a"; "b"; "c"] ? A\$? LDIM(A\$) ' 3が得られます。</pre> <p>可変長配列 A\$を定義し、代入します。</p>		
使用例2	<pre>LIST A\$ B\$ = "A, B, C, D" A\$ = SPLIT\$(B\$, ", ") 可変長配列A\$を定義し、B\$に代入された文字列を", "で分割して代入します。</pre>		
使用例3	<pre>LIST A\$? LDIM(A\$) ' 配列の要素数は「1」 ONEDIM INSERT A\$, -1, "A" ? LDIM(A\$) ' 配列の要素数は「2」に増えます ONEDIM INSERT A\$, -1, "B" ? LDIM(A\$) ' 配列の要素数は「3」に増えます ONEDIM INSERT A\$, -1, "C" ? LDIM(A\$) ' 配列の要素数は「4」に増えます</pre> <p>可変長配列 A\$を定義し、ONEDIM INSERT を使って 配列を拡張します。</p>		
使用例4	<pre>LIST ARY REDIM ARY(10) FOR I=0 TO UBOUND(ARY) ARY(I) = I+1 NEXT I PRINT ARY</pre> <p>可変長配列 ARYを定義し、REDIM を使って任意の大きさに変更します。</p>		

2.5.16 LIST BOOL

命令		
機 能	論理型可変長配列変数を宣言します。 この命令で宣言した変数は右辺の要素数によって自動で要素数を変更します。	
書 式	LIST BOOL <①変数名>[, <①変数名> , ...]	
パラ メータ	①	変数名
	論理型可変長配列変数名を指定します。	
注 意	<ul style="list-style-type: none"> ・ 宣言時に予め要素数を決めることはできません。 ・ 宣言時に値を代入することはできません。 ・ 実行時の要素数より大きい添字の要素を参照することはできません(参照による拡張は行われません)。 	
使用例	LIST BOOL A A = [TRUE; FALSE; TRUE] ? A 論理型可変長配列 Aを定義します。	

2.5.17 LIST DATETIME

命令		
機 能	日付時刻型可変長配列変数を宣言します。 この命令で宣言した変数は右辺の要素数によって自動で要素数を変更します。	
書 式	LIST DATETIME <①変数名>[, <①変数名> , ...]	
パラ メータ	①	変数名
	日付時刻型可変長配列変数名を指定します。	
注 意	<ul style="list-style-type: none"> ・ 宣言時に予め要素数を決めることはできません。 ・ 宣言時に値を代入することはできません。 ・ 実行時の要素数より大きい添字の要素を参照することはできません(参照による拡張は行われません)。 	
使用例1	LIST DATETIME ARY ARY = CDATETIME("2019/9/20 10:20:05") ONEDIM INSERT ARY, -1, CDATETIME("2020/1/1 0:1:2") PRINT ARY 日付時刻型可変長配列 ARYを定義し、代入と追加を行います。	
使用例2	LIST DATETIME A A = [CDATETIME("2019/9/20 1:2:3"); CDATETIME("2020/2/3 3:4:5")] ? A 日付時刻型可変長配列 A を定義と代入します。	

2.5.18 LIST DICT

命令		
機 能	連想配列の可変長配列変数を宣言します。 この命令で宣言した変数は右辺の要素数によって自動で要素数を変更します。	
書 式	LIST DICT <①変数名>[, <①変数名> , ...]	
パラ メータ	①	変数名
注 意	連想配列の可変長配列変数名を指定します。	
	<ul style="list-style-type: none"> ・ 宣言時に予め要素数を決めることはできません。 ・ 宣言時に値を代入することはできません。 ・ 実行時の要素数より大きい添字の要素を参照することはできません(参照による拡張は行われません)。 ・ 連想配列は、普通の配列のような添字の順序の概念や要素数の概念が存在しません。従って、PRINT文などで連想配列を表示した際、表示される順番はOSに依存します。 	
使用例	<p>連想配列の可変長配列 Aを定義します。</p> <pre>LIST DICT A A(0)("test") = 111 A(0)("interface") = 777 ? A 連想配列を定義して、最初の設定</pre> <pre>DICT B B("hoge") = 123 B("fuga") = 456 ONEDIM INSERT A, -1, B ? A 連想配列Bを作っておいて、A配列に追加。</pre>	

2.5.19 LIST DICT BOOL

命令		
機能	論理型連想配列の可変長配列変数を宣言します。 この命令で宣言した変数は右辺の要素数によって自動で要素数を変更します。	
書式	LIST DICT BOOL <①変数名>[, <①変数名> , ...]	
パラメータ	①	変数名
注意	論理型連想配列の可変長配列変数名を指定します。 <ul style="list-style-type: none">・ 宣言時に予め要素数を決めることはできません。・ 宣言時に値を代入することはできません。・ 実行時の要素数より大きい添字の要素を参照することはできません(参照による拡張は行われません)。・ 連想配列は、普通の配列のような添字の順序の概念や要素数の概念が存在しません。従って、PRINT文などで連想配列を表示した際、表示される順番は0Sに依存します。	
使用例	論理型連想配列の可変長配列 Aを定義します。 LIST DICT BOOL A A(0)("test") = TRUE A(0)("hoge") = FALSE ? A 変数Aに設定 DICT BOOL B B("mogu") = FALSE B("helo") = TRUE ONEDIM INSERT A, -1, B ? A 変数Bを、変数Aに追加	

2.5.20 LIST DICT DATETIME

命令		
機 能	日付時刻型連想配列の可変長配列変数を宣言します。 この命令で宣言した変数は右辺の要素数によって自動で要素数を変更します。	
書 式	LIST DICT DATETIME <①変数名>[, <①変数名> , ...]	
パラ メータ	①	変数名
注 意	日付時刻型連想配列の可変長配列変数名を指定します。	
使用例	<ul style="list-style-type: none"> ・ 宣言時に予め要素数を決めることはできません。 ・ 宣言時に値を代入することはできません。 ・ 実行時の要素数より大きい添字の要素を参照することはできません(参照による拡張は行われません)。 ・ 連想配列は、普通の配列のような添字の順序の概念や要素数の概念が存在しません。従って、PRINT文などで連想配列を表示した際、表示される順番はOSに依存します。 	
	<p>日付時刻型連想配列の可変長配列 Aを定義します。</p> <pre> LIST DICT DATETIME A A(0)("test") = CDATETIME("2019/1/1 0:1:2") A(0)("hoge") = CDATETIME("2020/2/3 2:3:4") ? A ' 変数Aに設定 DICT DATETIME B B("mogu") = CDATETIME("2019/9/20 10:23:33") B("helo") = CDATETIME("2019/9/24 8:0:0") ONEDIM INSERT A, -1, B ? A ' 変数Bを、変数Aに追加 </pre>	

2.5.21 LIST DICT STRUCT

命令		
機能	構造体型連想配列の可変長配列変数を定義します。 この命令で宣言した変数は右辺の要素数によって自動で要素数を変更します。	
書式	LIST DICT STRUCT <①構造体名> <②変数名>[, <②変数名> , ...]	
パラメータ	①	<構造体名> 構造体名
	DEFINE STRUCTで定義した構造体名を指定します。	
	②	<変数名> 変数名
注意	<p>構造体型連想配列の可変長配列変数名を指定します。</p> <ul style="list-style-type: none"> ・ 構造体名は、「2.4.6 DEFINE STRUCT～END STRUCT」で定義します。 ・ 宣言時に予め要素数を決めることはできません。 ・ 宣言時に値を代入することはできません。 ・ 実行時の要素数より大きい添字の要素を参照することはできません(参照による拡張は行われません)。 ・ 連想配列は、普通の配列のような添字の順序の概念や要素数の概念が存在しません。従って、PRINT文などで連想配列を表示した際、表示される順番はOSに依存します。 	
使用例	<p>構造体型連想配列の可変長配列 Aを定義します。</p> <pre> DEFINE STRUCT RECORD ID ADDR\$ END STRUCT LIST DICT STRUCT RECORD A A(0)("test").ID = 1 A(0)("test").ADDR\$ = "test" ? A DICT STRUCT RECORD B B("mogu").ID = 2 B("mogu").ADDR\$ = "japan" ONEDIM INSERT A, -1, B ? A </pre>	

2.5.22 LIST STRUCT

命令		
機能	構造体型の可変長配列変数を宣言します。 この命令で宣言した変数は右辺の要素数によって自動で要素数を変更します。	
書式	LIST STRUCT <①構造体名> <②変数名>[, <②変数名> , ...]	
パラメータ	①	構造体名
	DEFINE STRUCTで定義した構造体名を指定します。	
	②	変数名
注意	<ul style="list-style-type: none"> 構造体名は、「2.4.6 DEFINE STRUCT～END STRUCT」で定義します。 宣言時に予め要素数を決めることはできません。 宣言時に値を代入することはできません。 実行時の要素数より大きい添字の要素を参照することはできません(参照による拡張は行われません)。 	
使用例	<p>構造体型の可変長配列 Aを定義します。</p> <pre> DEFINE STRUCT RECORD ID ADDR\$ END STRUCT LIST STRUCT RECORD A A(0). ID = 123 A(0). ADDR\$ = "hoge" ? A STRUCT RECORD B B. ID = 456 B. ADDR\$ = "test" ONEDIM INSERT A, -1, B ? A </pre>	

2.5.23 LOCAL DICT

命令			
機 能	連想配列のローカル変数を宣言します。		
書 式	LOCAL DICT <①変数名> [, <①変数名> ...]		
パラ メータ	①	<変数名>	変数名
	連想配列のローカル変数名を指定します。		
注 意	・ 連想配列のローカル変数に格納可能な値は、名前の型修飾に由来します。 ・ ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。 (参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」)		
使用例	LOCAL DICT A, B 連想配列のローカル変数A, Bを宣言します。 使用例は、「DICT」も併せて参考にしてください。		

2.5.24 LOCAL DICT BOOL

命令			
機 能	論理型連想配列のローカル変数を宣言します。		
書 式	LOCAL DICT BOOL <①変数名> [, <①変数名> ...]		
パラメータ	①	<変数名>	変数名
	論理型連想配列のローカル変数名を指定します。		
注 意	・ ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」）		
使用例	LOCAL DICT BOOL A, B 論理型連想配列のローカル変数A, Bを宣言します。 使用例は、「DICT BOOL」も併せて参考にしてください。		

2.5.25 LOCAL DICT DATETIME

命令		
機 能	日付時刻型連想配列のローカル変数を宣言します。	
書 式	LOCAL DICT DATETIME <①変数名> [, <①変数名> …]	
パラ メータ	①	変数名
	日付時刻型連想配列のローカル変数名を指定します。	
注 意	<ul style="list-style-type: none"> ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 	
使用例	LOCAL DICT DATETIME A, B 日付時刻型連想配列のローカル変数A, Bを宣言します。 使用例は、「 DICTIONARY 」も併せて参考にしてください。	

2.5.26 LOCAL DICT STRUCT

命令		
機 能	構造体型連想配列のローカル変数を宣言します。	
書 式	LOCAL DICT STRUCT <①構造体名> <②変数名> [, <②変数名> …]	
パラ メータ	①	構造体名
	DEFINE STRUCTで定義した構造体名を指定します。	
	②	変数名
	構造体型連想配列のローカル変数名を指定します。	
注 意	<ul style="list-style-type: none"> 構造体名は、「2.4.6 DEFINE STRUCT～END STRUCT」で定義します。 ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 	
使用例	LOCAL DICT STRUCT RECORD A, B 構造体型連想配列のローカル変数A, Bを宣言します。（構造体 RECORDは別途定義されているものとします） 使用例は、「 DICTIONARY 」も併せて参考にしてください。	

2.5.27 LOCAL LIST

命令		
機 能	可変長配列のローカル変数を宣言します。	
書 式	LOCAL LIST <①変数名> [, <①変数名> …]	
パラ メータ	①	変数名
	可変長配列のローカル変数名を指定します。	
注 意	<ul style="list-style-type: none"> ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 	
使用例	LOCAL LIST A\$, B\$ ローカル可変長配列変数A\$, B\$を宣言します。 使用例は、「 LIST 」も併せて参考にしてください。	

2.5.28 LOCAL LIST BOOL

命令		
機 能	論理型可変長配列のローカル変数を宣言します。	
書 式	LOCAL LIST BOOL <①変数名> [, <①変数名> ...]	
パラ メータ	①	変数名 論理型可変長配列のローカル変数名を指定します。
注 意	<ul style="list-style-type: none"> ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 	
使用例	LOCAL LIST BOOL A, B 論理型可変長配列のローカル変数A, Bを宣言します。 使用例は、「LIST BOOL」も併せて参考になしてください。	

2.5.29 LOCAL LIST DATETIME

命令		
機 能	日付時刻型可変長配列のローカル変数を宣言します。	
書 式	LOCAL LIST DATETIME <①変数名> [, <①変数名> ...]	
パラ メータ	①	変数名 日付時刻型可変長配列のローカル変数名を指定します。
注 意	<ul style="list-style-type: none"> ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 	
使用例	LOCAL LIST DATETIME A, B 日付時刻型可変長配列のローカル変数A, Bを宣言します。 使用例は、「LIST DATETIME」も併せて参考になしてください。	

2.5.30 LOCAL LIST DICT

命令		
機 能	連想配列のローカル可変長配列変数を宣言します。	
書 式	LOCAL LIST DICT <①変数名> [, <①変数名> ...]	
パラ メータ	①	変数名 連想配列のローカル可変長配列変数名を指定します。
注 意	<ul style="list-style-type: none"> ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 	
使用例	LOCAL LIST DICT A, B 連想配列のローカル可変長配列変数A, Bを宣言します。 使用例は、「LIST DICT」も併せて参考になしてください。	

2.5.31 LOCAL LIST DICT BOOL

命令		
機 能	論理型連想配列のローカル可変長配列変数を宣言します。	
書 式	LOCAL LIST DICT BOOL <①変数名> [, <①変数名> …]	
パラ メータ	①	変数名
	論理型連想配列のローカル可変長配列変数名を指定します。	
注 意	<ul style="list-style-type: none"> ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 	
使用例	LOCAL LIST DICT BOOL A, B	
	論理型連想配列のローカル可変長配列変数A,Bを宣言します。 使用例は、「LIST DICT BOOL」も併せて参考にしてください。	

2.5.32 LOCAL LIST DICT DATETIME

命令		
機 能	日付時刻型連想配列のローカル可変長配列変数を宣言します。	
書 式	LOCAL LIST DICT DATETIME <①変数名> [, <①変数名> …]	
パラ メータ	①	変数名
	日付時刻型連想配列のローカル可変長配列変数名を指定します。	
注 意	<ul style="list-style-type: none"> ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 	
使用例	LOCAL LIST DICT DATETIME A, B	
	日付時刻型連想配列のローカル可変長配列変数A,Bを宣言します。 使用例は、「LIST DICT DATETIME」も併せて参考にしてください。	

2.5.33 LOCAL LIST DICT STRUCT

命令		
機能	構造体型連想配列のローカル可変長配列変数を宣言します。	
書式	LOCAL LIST DICT STRUCT <①構造体名> <②変数名> [, <②変数名> ...]	
パラメータ	①	<div> <div><構造体名></div> <div>構造体名</div> </div>
	DEFINE STRUCTで定義した構造体名を指定します。	
	②	<div> <div><変数名></div> <div>変数名</div> </div>
構造体型連想配列のローカル可変長配列変数名を指定します。		
注意	<ul style="list-style-type: none"> ・構造体名は、「2.4.6 DEFINE STRUCT～END STRUCT」で定義します。 ・ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 	
使用例	LOCAL LIST DICT STRUCT RECORD A, B 構造体型連想配列のローカル可変長配列変数A, Bを宣言します。（構造体RECORDは別途定義されているものとします） 使用例は、「LIST DICT STRUCT」も併せて参考にしてください。	

2.5.34 LOCAL LIST STRUCT

命令		
機能	構造体型可変長配列のローカル変数を宣言します。	
書式	LOCAL LIST STRUCT <①構造体名> <②変数名> [, <②変数名> ...]	
パラメータ	①	<div> <div><構造体名></div> <div>構造体名</div> </div>
	DEFINE STRUCTで定義した構造体名を指定します。	
	②	<div> <div><変数名></div> <div>変数名</div> </div>
構造体型可変長配列のローカル変数名を指定します。		
注意	<ul style="list-style-type: none"> ・構造体名は、「2.4.6 DEFINE STRUCT～END STRUCT」で定義します。 ・ローカル変数を宣言せずに、ローカル変数にアクセスしようとししないでください。（参考：基本マニュアルの「未初期化のローカル変数へのアクセスは動作不定」） 	
使用例	LOCAL LIST STRUCT RECORD A, B 連想配列を要素とした構造体型ローカル可変長配列変数A, Bを宣言します。（構造体RECORDは別途定義されているものとします） 使用例は、「LIST STRUCT」も併せて参考にしてください。	

2.5.35 ONEDIM INSERT

命令		
機 能	1次元配列の指定した位置に、変数(配列可)を挿入します。	
書 式	ONEDIM INSERT <①挿入先配列変数名>, <②挿入先位置>, <③挿入元変数>	
パラ メータ	①	変数名 <div> <div><挿入先配列変数名></div> <div>挿入される配列変数名を指定します。 数値型もしくは文字列型の1次元配列を与えてください。</div> </div>
	②	数値 <div> <div><挿入先位置></div> <div>挿入する配列の添字を指定します。 後尾に追加したい場合、-1を指定することができます。</div> </div>
	③	変数名 <div> <div><挿入元変数></div> <div>挿入する変数を指定します。 1次元配列か1要素の変数で、挿入先変数の型と同じ型を与えてください。</div> </div>
注 意	<ul style="list-style-type: none"> 挿入先配列変数名に配列ではない変数を指定すると、予期しない動作を起こす恐れがあります。 	
使用例	<pre> DIM A(3), B(2) A = [1; 2; 3; 4] B = [10; 11; 12] ONEDIM INSERT A, 1, B PRINT A ----- 実行結果 [1;10;11;12;2;3;4] 配列Aの添字1へ配列Bを挿入します。 </pre>	

2.5.36 ONEDIM REMOVE

命令		
機能	1次元配列の指定した位置から、指定した要素を削除します。	
書式	ONEDIM REMOVE <①削除配列変数名>, <②削除位置>, <③削除要素数>	
パラメータ	①	変数名
	<削除配列変数名> 要素を削除される配列変数名を指定します。 数値型もしくは文字列型の1次元配列を与えてください。	
	②	数値
	<削除位置> 削除する配列の添字を指定します。	
	③	数値
	<削除要素数> 削除する要素数を指定します。 -1を指定すると、削除位置からすべてを削除できます。	
注意	・ 削除配列変数名に配列ではない変数を指定すると、予期しない動作を起こす恐れがあります。 ・ すべての要素を削除することはできません。	
使用例	DIM A(3) A = [1; 2; 3; 4] ONEDIM REMOVE A, 1, 2 PRINT A ----- 実行結果 [1;4] 配列Aの添字1から2要素分削除します。	

2.5.37 REDIM

命令			
機 能	可変長配列の次元数および各次元の要素数を任意に変更します。		
書 式	REDIM [①PRESERVE] <②変数名> (<③添字最大値> [, ...])		
パラ メータ	①	PRESERVE	キーワード
	PRESERVE キーワードを指定すると、元の変数の内容を保持しようとします。 省略すると、値は全てクリアされます。		
	②	<変数名>	変数名
	配列変数名を指定します。 ここで指定する変数名は、LIST 命令などで指定する 可変長配列変数です。 DIM 命令で指定する 固定長の配列変数は指定できません。		
	③	<添字最大値>	数値
	配列変数に対する、各次元の要素数を指定します。		
注 意	<ul style="list-style-type: none"> ・ PRESERVEキーワードを指定すると、元の変数の内容を保持しようとしませんが、元の要素数より少ない要素数を指定すると、その分切り捨てられます。 また、元の要素数より多い要素数を指定すると、追加分は空または0の値が入ります。 ・ PRESERVEキーワードを指定しつつ、次元数を変更するとき、元の変数の内容は 先頭から詰めて入ります。 ・ 構造体のメンバ変数に対して、REDIM を適用できません。 		
使用例1	LIST A REDIM A(9) A変数を、添字0～9が指定可能な、10の要素を持つ配列変数に拡張します。		
使用例2	LIST A A = [1; 2; 3; 4; 5] REDIM PRESERVE A(9) A変数を、10の要素を持つ配列変数に拡張します。PRESERVEキーワードが指定されているので、元の値は保持されます。		
使用例3	LIST A A = [1; 2; 3; 4; 5; 6] ? A ' [1, 2, 3, 4, 5, 6] のように表示されます REDIM PRESERVE A(1, 1) ? A ' [[1, 2], [3, 4]] のように表示されます。 1次元配列のA変数を、2次元配列の配列変数に変更します。		
使用例4	DEFINE STRUCT INFO ID ADDR\$ END STRUCT DEFINE STRUCT INFO2 LIST STRUCT INFO ARY END STRUCT STRUCT INFO2 V LIST STRUCT INFO TMP_V REDIM TMP_V(10) ' 構造体配列をREDIMします V.ARY = TMP_V ' 構造体のメンバに、REDIMした配列を代入します ' 構造体のメンバ変数はREDIMできません。その代替りの手段を示しています。		

2.5.38 TWODIM INSERT

命令		
機 能	2次元配列の指定した行／列位置に、1行／列分の配列変数を挿入します。	
書 式	TWODIM INSERT <①方向>, <②挿入先配列変数名>, <③挿入先位置>, <④挿入元変数>	
パラ メータ	①	<div> <div><方向></div> <div>キーワード</div> </div> ROWを指定すると行方向に挿入します。 COLUMNを指定すると列方向に挿入します。
	②	<div> <div><挿入先配列変数名></div> <div>変数名</div> </div> 挿入される配列変数名を指定します。 数値型もしくは文字列型の2次元配列を与えてください。
	③	<div> <div><挿入先位置></div> <div>数値</div> </div> 挿入行／列位置を指定します。 後尾に追加したい場合、-1を指定することができます。
	④	<div> <div><挿入元変数></div> <div>変数名</div> </div> 挿入する変数を指定します。 挿入先変数の型と1行／列分の要素数の1次元配列を与えてください。
備 考	・2次元配列の評価は、以下の行列と見なして動作します。 DIM 変数(行数, 列数)	
注 意	・挿入先配列変数名に配列ではない変数を指定すると、予期しない動作を起こす恐れがあります。	
使用例1	<pre> DIM A(1,1) = [0;1;2;3] DIM B(1) = [4; 5] TWODIM INSERT ROW, A, -1, B ? A ----- 実行結果 [[0; 1]; [2; 3] ; [4; 5]] 2次元配列Aの末尾行に1次元配列Bの内容を挿入します。 </pre>	
使用例2	<pre> DIM A(1,2) = [[0; 1; 2] , [3; 4; 5]] DIM B(1) = [6; 7] TWODIM INSERT COLUMN, A , 1 , B ? A ----- 実行結果 [[0; 6; 1; 2]; [3; 7; 4; 5]] 2次元配列Aの添字1の列に1次元配列Bの内容を挿入します。 </pre>	
使用例3	<pre> LIST A\$, B\$ A\$ = "A" ? CDIM(A\$), LDIM(A\$) ' 次元数:1、要素数:1 です B\$ = "B" TWODIM INSERT COLUMN, A\$, -1, B\$? CDIM(A\$), LDIM(A\$, 1), LDIM(A\$, 2) ' 次元数:2、行数:1、列数:2 です ? A\$ ' [[A, B]] と表示 TWODIM INSERT ROW, A\$, -1, B\$ ' 次元数:2、行数:2、列数:2 です ? CDIM(A\$), LDIM(A\$, 1), LDIM(A\$, 2) ' [[A, B], [C, D]] と表示 ? A\$ </pre>	

	可変長変数を宣言しておき、そこから列数を増やし、行数を増やす事例です。
--	-------------------------------------

2. 5. 39 TWODIM REMOVE

命令			
機 能	2次元配列の指定した位置から、指定した行数／列数を削除します。		
書 式	TWODIM REMOVE <①方向>, <②削除配列変数名>, <③削除位置>, <④削除数>		
パラ メータ	①	<方向>	キーワード
	ROWを指定すると行方向に対して削除します。 COLUMNを指定すると列方向に対して削除します。		
	②	<削除配列変数名>	変数名
	要素を削除される配列変数名を指定します。 数値型もしくは文字列型の2次元配列を与えてください。		
	③	<削除位置>	数値
	削除する開始行／列位置を指定します。		
	④	<削除数>	数値
削除する行／列数を指定します。 -1を指定すると、削除位置からすべてを削除できます。			
備 考	・ 2次元配列の評価は、以下の行列と見なして動作します。 DIM 変数(行数, 列数)		
注 意	・ 削除配列変数名に配列ではない変数を指定すると、予期しない動作を起こす恐れがあります。 ・ すべての要素を削除することはできません。		
使用例1	DIM A(1,1) = [[0;1] , [2;3]] TWODIM REMOVE ROW, A, 1, 1 ? A ----- 実行結果 [0; 1] 2次元配列Aの末尾行を削除します。		
使用例2	DIM A(1,2) = [[0; 1; 2] , [3; 4; 5]] TWODIM REMOVE COLUMN, A , 1 , -1 ? A ----- 実行結果 [[0] ; [3]] 2次元配列Aの2列目以降を削除します。		

2.5.40 UBOUND

関数			
機 能	配列の指定した次元の添字最大値を得ます。		
書 式	<(戻り値) 添字最大値> = UBOUND(<①配列変数名> [, <②次元>])		
戻り値	戻り値	<添字最大値>	数値
	配列の指定した次元の添字最大値が返ります。		
パラ メータ	①	<配列変数名>	変数名
	対象となる配列変数名を指定します。		
	②	<次元>	数値
	1から始まる、添字最大値を返す次元を指定します。		
	省略すると1が指定されたものとします。		
備 考	<ul style="list-style-type: none"> ・ 配列変数でない、または存在しない次元を指定するとエラーが返ります。 ・ 指定した次元の要素数を得たい場合は「LDIM」を使用してください。 		
使用例1	DIM A\$(5) L=UBOUND(A\$) 変数 L に配列A\$の添字最大値 5 を代入します。		
使用例2	DIM A\$(2, 5) L = UBOUND(A\$, 1) M = UBOUND(A\$, 2) 変数Lは、配列A\$の1次元目の添字最大値：2が。 変数Mは、配列A\$の2次元目の添字最大値：5が得られます。		

2.6 繰り返し・条件分岐に関する関数・命令

2.6.1 DO WHILE～LOOP

命令			
機能	DOからLOOPまでの区間中にある一連の命令を、指定条件を満たす(TRUE)間、繰り返し実行します。		
書式	DO WHILE <①条件式> ... LOOP		
パラメータ	①	<条件式>	式
注意	繰り返し実行する条件を指定します。 ・ループ内へ外部からジャンプして入ってきたり、ループ内から外部へGOTOジャンプしたりするプログラムは、その動作が保証されなくなります。 ループ内で途中からの脱出には、EXIT DO文(→「2.6.2 EXIT DO」)を利用ください。 ・<条件式> に TRUE を指定すると、永久ループになります。		
使用例1	<pre>I = 0 DO WHILE I<6 ... LOOP</pre> <p>DO WHILE～LOOPの中に記述された処理を指定条件(Iが6より小さい)を満たす間繰り返し実行します。</p>		
使用例2	<pre>I = 0 DO WHILE TRUE IF I < 6 THEN EXIT DO END IF I = I + 1 ... LOOP</pre> <p>DO WHILE～LOOPを永久ループにして、中の処理でIF文判定により脱出できる形式としたものです。</p>		

2.6.2 EXIT DO

命令	
機 能	DO～LOOP文の繰り返しから脱出します。
書 式	EXIT DO
備 考	DO WHILE～LOOP(→「2.6.1 DO WHILE～LOOP」)からの脱出に用います。
注 意	
使用例	<pre> I = 0 DO WHILE I<10 I = I + 1 IF I>5 THEN EXIT DO END IF LOOP </pre> <p>Iの値が5より大きくなった時に脱出する例です。</p>

2.6.3 EXIT FOR

命令	
機 能	FOR～NEXT文の繰り返しから脱出します。
書 式	EXIT FOR
注 意	
備 考	FOR～TO～STEP～NEXT(→「2.6.4 FOR～TO～STEP～NEXT」)からの脱出に用います。
使用例	<pre> FOR I=0 TO 10 STEP 1 IF I>5 THEN EXIT FOR END IF NEXT I </pre> <p>Iの値が5より大きくなった時に脱出する例です。</p>

2.6.4 FOR～TO～STEP～NEXT

命令		
機 能	FORからNEXTまでの区間中にある一連の命令を、繰り返し実行します。	
書 式	FOR <①変数名> = <②初期値> TO <③終値> [STEP <④増減分>] ... NEXT [<変数>]	
パラ メータ	①	<変数名> 変数名 繰り返し条件に使用する変数名を指定します。
	②	<初期値> 数値 変数の初期値を指定します。
	③	<終値> 数値 変数の終値を指定します。
	④	<増減分> 数値 初期値から終値までの増減分を整数で指定します。 省略した場合、次のように自動で決定されます。 <初期値>が<終値>より小さい場合：1 <初期値>が<終値>より大きい場合：-1
注 意	・ ループ内へ外部からジャンプして入ってきたり、ループ内から外部へGOTOジャンプしたりするプログラムは、その動作が保証されなくなります。 ループ内で途中からの脱出には、EXIT FOR文(→「2.6.3 EXIT FOR」)を ご利用ください。 ・ 繰り返しの実行回数(終値)や増減分は、FOR文の最初の実行時に確定します。 従って、繰り返し実行中に変動したい場合は、DO WHILE～LOOP文を使用してください。	
使用例1	FOR I=0 TO 10 STEP 1 SUM = SUM + I NEXT I 変数Iの初期値を0として10になるまで1ずつ加算しながらFOR～NEXT文の区間中にある処理を繰り返します。	
使用例2	FOR I=10 TO 1 ... NEXT 変数Iの初期値を10として0になるまで1ずつ減算しながらFOR～NEXT文の区間中にある処理を繰り返します。	

2.6.5 IF～THEN～ELSE～END IF

命令		
機能	式の値の条件判定を行います。条件式が真の場合は、THEN以降の命令が実行されます。条件式が偽の場合は、ELSE以降の命令が実行されます。条件式を複数使いたい時はELSEIF文を使います。	
書式	IF <①条件式> THEN ... [ELSEIF <①条件式> THEN] ... [ELSE] ... END IF	
パラメータ	①	<div><条件式></div> <div>式</div> 条件判定する式を指定します。条件式では関係演算子や論理演算子を使用できます。詳しくは、基本マニュアルの「式と演算について」を参照してください。
注意	ELSEIF文は複数行形式のみです。THENやELSEの後に続けて、同一行に文を続けないでください。エラーになります。	
使用例1	IF I=1 THEN PRINT "OK!" END IF 変数Iの値が1の時、「OK」と表示します。	
使用例2	IF I=1 THEN PRINT "OK!" ELSE PRINT "NG!" END IF 変数Iの値が1の時、「OK」と表示、それ以外の場合は「NG」と表示します。	
使用例3	IF I=1 THEN PRINT "OK!" ELSEIF I=0 THEN PRINT "NG!" ELSE PRINT "UNKNOWN" END IF 変数Iの値が1の時は「OK」と表示、0の時は「NG」と表示、それ以外の場合は「UNKNOWN」と表示します。	
使用例4	IF I = 1 AND J = 0 THEN PRINT "OK!" ELSE PRINT "NG!" END IF 変数Iの値が1の時 かつ 変数Jの値が0の時は「OK」と表示、それ以外の場合は「NG」と表示します。	
使用例5	IF I = 1 OR J = 0 THEN PRINT "OK!" ELSE PRINT "NG!" END IF 変数Iの値が1の時 または 変数Jの値が0の時は「OK」と表示、それ以外の場合は「NG」と表示します。	

2.6.6 SELECT CASE～END SELECT

命令			
機能	式の値と続くCASE文に従い、処理を分岐します。 CASE文に該当する値がなければ、CASE ELSE文の分岐に入ります。		
書式	SELECT CASE <①式> CASE <②式の値> ... [CASE ELSE] ... END SELECT		
パラ メータ	①	<式>	式
	分岐の判定に使う式を指定します。文字型、数値型のどちらでも指定できます。		
	②	<式の値>	値
分岐するための式の値を指定します。文字型、数値型のどちらでも指定できます。			
使用例1	SELECT CASE X CASE 1 PRINT "OK!" CASE 0 PRINT "NG!" END SELECT 変数Xの値が 1 の時は「OK」と表示、0の時は「NG」と表示します。		
使用例2	SELECT CASE X CASE 1 PRINT "OK!" CASE 0 PRINT "NG!" CASE ELSE PRINT "UNKNOWN" END SELECT 変数Xの値が 1 の時は「OK」と表示、0の時は「NG」と表示、それ以外の場合は「UNKNOWN」と表示します。		

2.7 ファイル・フォルダに関する関数・命令

2.7.1 参考：FIFO特殊ファイル(名前付きパイプ)を使ったプロセス間通信

Linuxには、FIFO特殊ファイル(名前付きパイプ)と呼ばれる特殊なファイル機構があり、これを使うと容易にプロセス間通信が実現できます。

FIFO特殊ファイルを作成するには、Linuxの端末上で「mkfifo」コマンド(AJANコマンドではありません)を呼び出します。

以下は、「hoge」という名前のFIFO特殊ファイルを作ります。

```
$ mkfifo hoge
$ ls -l
...
prw-r--r--  1 user user      0  5月 30 18:46 hoge
-rw-r--r--  1 user user    368  4月  4 17:56 test.py
```

作成したFIFO特殊ファイルで、プロセス間通信を行うには、書き込み側と読み込み側のプログラムを作成して、実行する必要があります。

```
' 書き込み側の例
OPEN "hoge" FOR OUTPUT AS #1
PRINT #1, "hello AJAN"
CLOSE #1
```

```
' 読み込み側の例
OPEN "hoge" FOR INPUT AS #1
LINE INPUT #1, A$
PRINT "読み取った:"; A$
CLOSE #1
```

FIFO特殊ファイルは、書き込む側と読み込む側が揃わないとオープンが完了しません。

例えば、書き込む側を実行開始した後、読み込む側を実行するまで、書き込む側のOPEN命令で処理は止まります。これは、OSの仕様に依存します。

このように止まるのではなく、どうしても書き込む側、読み込む側が揃わない時でも、OPEN命令を進行させたい場合、OPEN 命令で NONBLOCKING を指定します。

```
' 書き込み側の例
OPEN "hoge" FOR OUTPUT NONBLOCKING AS #1
PRINT #1, "hello AJAN"
CLOSE #1
```

```
' 読み込み側の例
OPEN "hoge" FOR INPUT NONBLOCKING AS #1
LINE INPUT #1, A$
PRINT "読み取った:"; A$
CLOSE #1
```

NONBLOCKING を指定すると、ファイルアクセスに同期待ちをせずに動作する為、OPEN命令は そのまま進行します。

なお、相互が待たずにOPEN命令を進めても、FIFO特殊ファイルで読み書きを進めるには、相互が揃わないとできない点に注意してください。

例えば、FIFO特殊ファイルに対して、書き込みプログラムを実行完了して後に、読み込みプログラムを実行すると、書き込んだ文字列は読み取られずに、空文字が得られるだけです。

これを避けるには、双方がOPEN完了してから読み書きを行うか、FIFO特殊ファイルを「FOR OUTPUT NONBLOCKING」で、開いたままとする別のプログラムを実行しておく。といった工夫が必要です。

2.7.2 CHDIR

命令		
機 能	現在の作業フォルダを変更します。	
書 式	CHDIR <①フォルダ名>	
パラ メータ	①	<div><フォルダ名></div> <div>文字列</div> 作業フォルダを移すフォルダ名を指定します。
使用例1	CHDIR "SAMPLE" 作業フォルダを" SAMPLE" に変更します。	
使用例2	CHDIR ".." 作業フォルダを1階層上に変更します。	

2.7.3 CLOSE

命令		
機 能	ファイルをクローズします。	
書 式	CLOSE [[#]<①ファイル番号> [, [#]<①ファイル番号>] ...]	
パラ メータ	①	<div><ファイル番号></div> <div>数値</div> クローズするファイル番号を指定します。 省略された場合、オープン済み(クローズ状態か初期状態以外)のファイルをすべてクローズします。
使用例1	CLOSE #1, #2 ファイル番号1, 2のファイルをクローズします。	
使用例2	CLOSE オープン済みのファイルをすべてクローズします。	

2.7.4 DIREXISTS

関数			
機能	フォルダが存在するか確認し、結果を返します。 存在する場合はTRUE、存在しない場合はFALSEを返します。		
書式	<(戻り値)フォルダの存在有無> = DIREXISTS (<①フォルダ名>)		
戻り値	戻り値	<フォルダの存在有無>	真偽値
	フォルダが存在する場合はTRUE、存在しない場合はFALSEを返します。		
パラメータ	①	<フォルダ名>	文字列
	存在を確認するフォルダ名を指定します。		
使用例	IF DIREXISTS("/media/fd") = FALSE THEN PRINT "フォルダが存在しません" END IF 指定したフォルダ (/media/fd) が存在しなかった場合、「フォルダが存在しません」とメッセージを表示します。		

2.7.5 EOF

関数			
機 能	ファイルの終了コードを調べます。 ファイルの終端の場合はTRUEが、終端ではない場合はFALSEが返ります。		
書 式	<(戻り値)ファイルの終端有無> = EOF(<①ファイル番号>)		
戻り値	戻り値	<ファイルの終端有無>	真偽値
	ファイルの終端の場合はTRUEが、終端ではない場合はFALSEが返ります。		
パラ メータ	①	<ファイル番号>	数値
	OPEN (→「2.7.17 OPEN」) でオープンしたファイル番号を指定します。		
備 考	・ ファイルをOUTPUTまたはAPPENDでオープンした場合は、常にTRUEが返ります。		
使用例	<pre>OPEN "SAMPLE.TXT" FOR INPUT AS #1 DO WHILE EOF(1) = FALSE ... LOOP</pre> <p>EOFを使用することで、ファイルが終端ではないとき（ファイルが空ではないとき）に特定の処理を実行することができます。戻り値によってファイルの終端になるまで特定の処理を実行することができます。</p>		

2.7.6 FILECOPY

命令		
機能	ファイルをコピーします。	
書式	FILECOPY <①コピー元ファイル名>, <②コピー先ファイル名> [, <③オプション指定>]	
パラメータ	①	<コピー元ファイル名> 文字列
	コピー元のファイル名を指定します。	
	②	<コピー先ファイル名> 文字列
	コピー先のファイル名を指定します。	
	③	<オプション指定> 文字列
	コピー方法のオプションを指定します。	
	何も指定しない：コピー先にファイルが存在するとエラーとなります。	
	"0"：コピー先にファイルが存在しても、コピーを強行します。	
備考	<ul style="list-style-type: none">本コマンドは、アクセス対象となるファイルまたはコピー先のアクセス権が不足すると、使用できません。アクセス権を保持するユーザーでの実行、または上位の管理者権限(スーパーユーザー)で実行する必要があります。	
使用例	FILECOPY "SRC.TXT" , "DST.TXT", "0" "SRC.TXT" を "DST.TXT" としてコピーします。(引数"0"により、DST.TXTが存在する場合も上書きして強制的にコピーします。)	

2.7.7 FILEEXISTS

関数			
機 能	ファイルが存在するか確認し、結果を返します。 存在する場合はTRUE、存在しない場合はFALSEを返します。		
書 式	<(戻り値)ファイルの存在有無> = FILEEXISTS("<①ファイル名>")		
戻り値	戻り値	<ファイルの存在有無>	真偽値
	ファイルが存在する場合はTRUE、存在しない場合はFALSEを返します。		
パラ メータ	①	<ファイル名>	文字列
	存在を確認するファイル名を指定します。		
使用例	<pre>IF FILEEXISTS("OPEN.AJN") = FALSE THEN PRINT "ファイルが存在しません" END IF</pre> <p>指定したファイル (OPEN.AJN) が存在しなかった場合、「ファイルが存在しません」とメッセージを表示します。</p>		

2.7.8 FILELIST\$

関数									
機能	指定したパターンにマッチするパス名を文字列配列形式で得ます。								
書式	〈(戻り値)パス名の文字列配列〉 = FILELIST\$("〈①ファイルパターン名〉")								
戻り値	戻り値	〈パス名の文字列配列〉	配列						
	ファイルパターン名にマッチしたパス名が、文字列配列形式で得られます。 パス名がディレクトリの場合、後尾に「/(スラッシュ)」が付加されます。 マッチするパス名が見つからない場合、空文字列が得られます。								
パラメータ	①	〈ファイルパターン名〉	文字列						
	ファイルのパス名を取得する為の、ワイルドカードパターンを指定します。 以下のメタ文字が使用可能です。								
<table><tr><th>メタ文字</th><th>動作</th></tr><tr><td>*</td><td>任意の文字列に一致</td></tr><tr><td>?</td><td>任意の1文字に一致</td></tr></table>				メタ文字	動作	*	任意の文字列に一致	?	任意の1文字に一致
メタ文字	動作								
*	任意の文字列に一致								
?	任意の1文字に一致								
空文字を指定すると、「*」を指定したのと同じになります。									
使用例1	PRINT FILELIST\$("*") カレントフォルダの全てのファイル一覧が得られます。								
使用例2	LIST ARY\$, LIST命令で可変長の文字列配列を宣言する ARY\$ = FILELIST\$("*.*JN") , 可変長の文字列配列を受ける PRINT ARY\$ PRINT "個数="; LDIM(ARY\$) , LDIM関数で配列の要素数を得られる 拡張子が「.*JN」に適合するファイル名一覧を得た後に、ARY\$ 配列変数に代入します。								

2.7.9 FILESTAT

関数																					
機 能	指定したファイルパス／ファイル番号のファイル情報を得ます。																				
書 式 1	<(戻り値)ステータス情報> = FILESTAT(<①ファイルパス名> [, <③取得ID>])																				
書 式 2	<(戻り値)ステータス情報> = FILESTAT(<②ファイル番号> [, <③取得ID>])																				
戻り値	戻り値	<ステータス情報>	数値																		
	取得IDに対応したファイル情報の値が得られます。																				
パラ メータ	①	<ファイルパス名>	文字列																		
	ファイル情報を取得したい、ファイルパス名を指定します。																				
	②	<ファイル番号>	数値																		
	ファイル情報を取得したい、ファイル番号を指定します。 ファイル番号は、OPEN命令で指定したファイル番号を与えます。																				
	③	<取得ID >	数値																		
	取得したいファイル情報の種類を指定します。 省略時、取得IDは、「3」で扱われます。																				
<table><tr><th>取得ID</th><th>得られる値</th><th>値の型</th></tr><tr><td>3</td><td>ファイルの種類とアクセス権の情報が得られます。</td><td>単精度整数</td></tr><tr><td>8</td><td>ファイルサイズをバイト単位で得られます。</td><td>単精度整数</td></tr><tr><td>11</td><td>最終アクセス時間を得られます。(atime) ファイルアクセス時に反映されます。</td><td>DATETIME型</td></tr><tr><td>12</td><td>最終修正時間を得られます。(mtime) ファイルの内容を書き換えると反映されます。 Linuxの lsコマンド を「ls -l」と呼び出した際に、 表示される時間でもあります。</td><td>DATETIME型</td></tr><tr><td>13</td><td>最終状態変更時間を得られます。(ctime) Linuxのinode情報が更新されると反映されます。</td><td>DATETIME型</td></tr></table>				取得ID	得られる値	値の型	3	ファイルの種類とアクセス権の情報が得られます。	単精度整数	8	ファイルサイズをバイト単位で得られます。	単精度整数	11	最終アクセス時間を得られます。(atime) ファイルアクセス時に反映されます。	DATETIME型	12	最終修正時間を得られます。(mtime) ファイルの内容を書き換えると反映されます。 Linuxの lsコマンド を「ls -l」と呼び出した際に、 表示される時間でもあります。	DATETIME型	13	最終状態変更時間を得られます。(ctime) Linuxのinode情報が更新されると反映されます。	DATETIME型
取得ID	得られる値	値の型																			
3	ファイルの種類とアクセス権の情報が得られます。	単精度整数																			
8	ファイルサイズをバイト単位で得られます。	単精度整数																			
11	最終アクセス時間を得られます。(atime) ファイルアクセス時に反映されます。	DATETIME型																			
12	最終修正時間を得られます。(mtime) ファイルの内容を書き換えると反映されます。 Linuxの lsコマンド を「ls -l」と呼び出した際に、 表示される時間でもあります。	DATETIME型																			
13	最終状態変更時間を得られます。(ctime) Linuxのinode情報が更新されると反映されます。	DATETIME型																			
上記以外の数値は、指定しないでください。																					
備 考	・ 取得ID：3の返却値は、stat(2)のst_modeフィールドのマスク値と同じです。 ・ 取得ID：3を指定時に、ファイルパス名で指定したファイルが見つからなかった時、 戻り値は「0」です。																				
使用例	PRINT FILESTAT("hoge.txt") hoge.txt ファイルの、ファイル情報を得ます。																				

2.7.10 FREEFILE

関数			
機 能	使用可能なファイル番号を得ます。		
書 式	<(戻り値)ファイル番号> = FREEFILE()		
戻り値	戻り値	<ファイル番号>	数値
	OPEN で使用可能な、ファイル番号が得られます。 全て使用中で空きがない場合、0が得られます。		
使用例	FNUM% = FREEFILE() PRINT "使用可能なファイル番号="; FNUM% OPEN "hoge.tmp" FOR INPUT AS #FNUM% LINE INPUT #FNUM%, A\$		

2.7.11 INCLUDE

命令			
機能	指定した外部ファイルのプログラムを参照できるように追加します。 これにより、外部ファイルのプログラムに書かれた関数やサブルーチン、設定値などが利用できるようになります。		
書式	INCLUDE "<①ファイル名>"		
パラメータ	①	<ファイル名>	文字列
備考	<p>参照追加するファイル名を指定します。 文字列定数で指定します。変数は使用できません。</p> <ul style="list-style-type: none"> ・ 処理内容的には、エディタ上で外部ファイルのプログラム内容を取り込んで貼り付けているのと同じです。 一般的には、共通処理的なプログラム内容を、外部ファイルに括り出し、INCLUDEで参照させる事により、プログラム記述の手間が省けます。 ・ 相対パスでファイルを指定した時、基本はカレントディレクトリを基準位置にファイルを検索します。 ・ ここでファイルが見つからない時、 /usr/share/interface/AJANPro/include フォルダを基準にファイルが見つからないか、検索します。 ・ /usr/share/interface/AJANPro/includeフォルダは、構造体や定数の各種定義を行った利便性の高い定義ファイルを格納しています。 勝手に内容を変更しないよう、お願いいたします。 		
注意	<ul style="list-style-type: none"> ・ サブルーチン内では使用できません。予期しない動作をする恐れがあります。 ・ <ファイル名>の先のプログラムコードを含めてコンパイルする必要がある為、必ず文字列定数を与えてください。 ・ 指定するファイル名は、大文字・小文字を区別します。 		
備考	<ul style="list-style-type: none"> ・ 実行(E)→デバッグ実行(D)を選択して実行すると、AJAN統合開発環境に参照先のプログラムタブが追加されます。 		
使用例	<p>INCLUDE "SAMPLE.AJN"</p> <p>次の実行文が"SAMPLE.AJN"の内容に移動します。 "SAMPLE.AJN"の終端まで実行すると、INCLUDE文の次の文から実行されます。</p>		

2.7.12 INPUT

命令		
機 能	キーボードやファイルから入力されたデータを変数に代入します。	
書 式 1	INPUT <①変数名> [, <①変数名> ...] キーボードから入力待ちの状態となり、入力されたデータが指定された変数に代入されます。入力待ち時「?」と画面に出力されます。	
書 式 2	INPUT <②プロンプト文>; <①変数名> [, <①変数名> ...] 書式1と同じですが、入力待ち時 プロンプト文に続けて「?」と画面に出力されます。	
書 式 3	INPUT <②プロンプト文>, <①変数名> [, <①変数名> ...] 書式1と同じですが、入力待ち時 プロンプト文のみが画面に出力されます。 (書式2のような「?」は付加されません)	
書 式 4	INPUT #<③ファイル番号>, <①変数名> [, <①変数名> ...] ファイル番号で指定されたファイルから入力されたデータが、指定された変数に代入されます。	
パラ メータ	①	<div> <div><変数名></div> <div>変数名</div> </div> キーボードないしはファイルから入力されたデータを、指定した変数に代入します。 変数が数値型の場合、データから数値が読み取られ、 文字列型の場合、データから文字列が読み取られます。 読み取るデータは、単一値です。配列変数を指定しても最初の添字のみ更新されます。
	②	<div> <div><プロンプト文></div> <div>文字列</div> </div> キーボードからデータを受ける前に、画面に表示するメッセージを指定できます。
	③	<div> <div><ファイル番号></div> <div>数値</div> </div> 入力先のファイル番号を指定します。 ファイル番号に変数を与えたい場合、他の書式の区別に、「#<変数名>」の記述としてください。
備 考	<ul style="list-style-type: none"> ファイルから入力する場合、OPENコマンドでFOR INPUT/APPENDを指定してください。 変数名を複数列挙したとき、 キーボードからの入力では、データの区切りに「,(カンマ)」を指定します。 ファイルからの入力では、データの区切りに「,(カンマ)」の他に空白文字(数値のみ)と改行が使用できます。 文字列を変数に入力するとき、「,(カンマ)」や文字列の前後に意味のある空白文字を入力するには、「”(ダブルクォーテーション)」で文字列を囲ってください。 指定しない時は、データの区切り「,(カンマ)」または改行までを読み取ります。 キーボードから入力するとき、何も入力しないで Enterキーだけ押すと、0 ないしは 空文字列が入力されたと見なします。 単精度整数と倍精度整数を変数名に与えたとき、「&B」の2進数形式の表記、「&H」の16進数形式の表記で入力できます。 	
注 意	<ul style="list-style-type: none"> 変数名に、構造体、連想配列を指定しないでください。 キーボードから入力したデータと変数の型が一致しない時、「Redo from start」と画面出力され再度入力待ちとなります。ファイルの場合は、エラーとなります。 キーボードから入力するとき、変数に値を読み取った後、まだ読み取り可能な有意データがあると、入力不正となり再入力となります。 ファイルから入力するとき、ファイルの終端に達して、更にデータの入力が必要なときエラーになります。 	
使用例1	INPUT A, B キーボードから「10, 20」と入力すると、A変数が10、B変数に20が代入されます。 なお「10 20」と入力すると、再入力を求められます。	
使用例2	INPUT #1, A, B ファイル番号 1のファイルから「10, 20」または「10 20」の入力データで、A変数に10、	

	B変数に20が代入されます。
使用例3	<p>INPUT A\$, B\$</p> <p>キーボードから「hoge, fuga」と入力すると、A\$変数に「hoge」、B\$変数に「fuga」が入力されます。</p> <p>また「hoge fuga, test」と入力すると、A\$変数に「hoge fuga」、B\$変数に「test」が入力されます。</p> <p>また「" hoge, fuga", test」と入力すると、A\$変数に「 hoge, fuga」、B\$変数に「test」が入力されます。</p> <p>また「,hoge」と入力すると、A\$変数は空文字列、B\$変数に「hoge」が入力されます。</p>

2.7.13 KILL

命令		
機 能	指定したファイルまたはディレクトリを削除します	
書 式	KILL <①パス名> [, <②ごみ箱スイッチ>]	
パラ メータ	①	<パス名> 文字列
	削除したいファイルまたはディレクトリ名を指定します。	
	②	<ごみ箱スイッチ> 真偽値
TRUEで、ファイルまたはフォルダをごみ箱に入れようとしています。 FALSE または 省略で、削除します。		
備 考	<ul style="list-style-type: none"> ・ディレクトリを削除する場合、中が空である必要があります。(Linuxのrmdirコマンドと同じ) ・ごみ箱に入れるには、ファイルシステムがごみ箱の仕掛けをサポートしている必要があります。サポートしていない場合、OSの動作に依存します。 ・削除するファイルまたはディレクトリが存在しない場合、エラーにならずに呼び出しは成功します。 ・アクセス権などの事由により、OSから削除を拒否される場合、エラーが通知されます。 	
使用例	KILL "test.txt" test.txt を削除しようとしています。	

2.7.14 LINE INPUT

命令		
機 能	キーボードやファイルから入力された1行単位のデータを文字型変数に代入します。プロンプト文でデータ入力前メッセージを表示させることもできます。	
書 式1	LINE INPUT ["<①プロンプト文>";] <②変数名> キーボードから入力された1行単位のデータを文字列変数に代入します。	
書 式2	LINE INPUT #<③ファイル番号>, <②変数名> ファイルから入力された1行単位のデータを文字列変数に代入します。	
パラ メータ	①	<プロンプト文> 文字列
	データ入力前メッセージを指定します。	
	②	<変数名> 変数名
	データを代入する文字型変数を指定します。	
注 意	③	<ファイル番号> 数値
	データを読み込むファイル番号を指定します。	
	ファイル番号に変数を与えたい場合、他の書式の区別に、「#<変数名>」の記述としてください。	
	<ul style="list-style-type: none"> ・ ファイルから文字列を読み込む場合、OPENコマンドでFOR INPUTを指定してください。 ・ IDEでデバッグ実行した際、キーボードでの文字入力待ちだと停止や終了はできません。入力後に停止や終了をします。 ・ LINE INPUT呼び出し中は、ON KEY CALLなどの割り込みは機能しません。 	
使用例1	LINE INPUT TEST\$ 変数TEST\$へキーボードから入力された文字列を代入します。	
使用例2	LINE INPUT "コメントを入力してください"; TEST\$ 変数TEST\$へキーボードから入力された文字列を代入します。 入力前にプロンプト文で「コメントを入力してください」と表示します。	
使用例3	LINE INPUT #1, TEST\$ 変数TEST\$へファイル番号1で開いているファイルから1行分の文字列を代入します。	
使用例4	NUM% = 1 LINE INPUT #NUM%, TEST\$ 使用例3のファイル番号の指定を、NUM%変数に置き換えた例です。	

2.7.15 MKDIR

命令			
機 能	新しいフォルダを作成します。		
書 式	MKDIR <①フォルダ名>		
パラ メータ	①	<フォルダ名>	文字列
	作成するフォルダ名を指定します。		
使用例	MKDIR "SAMPLE"		
	SAMPLE という名前で、新しいフォルダを作成します。		

2.7.16 NAME

命令			
機 能	ファイル/フォルダの名前を変更します。		
書 式	NAME <①旧ファイル or フォルダ名> AS <②新ファイル or フォルダ名>		
パラ メータ	①	<旧ファイル or フォルダ名>	文字列
	名前を変更するファイル or フォルダ名を指定します。		
	②	<新ファイル or フォルダ名>	文字列
	新しいファイル or フォルダ名を指定します。		
備 考	<ul style="list-style-type: none"> 指定したファイルがオープンされていると、エラーが発生します。 ファイル名を変更する前に、CLOSEを使用してファイルをクローズしてください。 		
注 意	<ul style="list-style-type: none"> 本コマンドは、アクセス対象となるファイルまたはフォルダ先のアクセス権が不足すると、使用できません。 アクセス権を保持するユーザーでの実行、または上位の管理者権限(スーパーユーザー)で実行する必要があります。 		
使用例	NAME "OLD. TXT" AS "NEW. TXT"		
	OLD. TXTというファイルを NEW. TXT というファイル名へ変更します。		

2.7.17 OPEN

命令			
機能	ファイルをオープンします。		
書式	OPEN "<①ファイル名>" FOR <②入出力モード> [ACCESS <③アクセスモード>] AS [#]<④ファイル番号>		
パラメータ	①	<ファイル名>	文字列
	<p>オープンするファイル名を指定します。 ファイル名は大文字/小文字を区別します。</p>		
	②	<入出力モード>	キーワード
	<p>ファイルの入出力モードを指定します。 FOR INPUT : 既存のファイルから入力を行います。 FOR OUTPUT : 新規作成したファイルに出力を行います。 FOR APPEND : 既存のファイルに追加出力を行います。 指定のファイルが存在しない場合は、新規作成したファイルに出力を行います。</p>		
	③	<アクセスモード>	キーワード
備考	④	<ファイル番号>	数値
	<p>オープンするファイルに関連付けるファイル番号を1～15の範囲で指定します。 ファイル番号に変数を与えたい場合、他の書式の区別に、「#<変数名>」の記述としてください。</p>		
注意	<ul style="list-style-type: none"> 相対パスでファイルを指定した時、基本はカレントディレクトリを基準位置にファイルを探索します。 Linuxのmkfifoコマンドで作れる FIFO特殊ファイル(名前付きパイプ)に対して、アクセスモードに NONBLOCKING を指定すると、同期待ちせずに読み書きが行えます。 アクセスモードに LOCK を指定するとき、オープンするファイルに対して 排他ロックをかけます。クローズする際、自動的に排他ロックは解除されます。 		
使用例1	<p>OPEN "SAMPLE.TXT" FOR INPUT AS #1 SAMPLE.TXTというファイルをファイル番号 1 で入力モードとして開きます。</p>		
使用例2	<p>OPEN "SAMPLE.TXT" FOR OUTPUT AS #1 SAMPLE.TXTというファイルをファイル番号 1 で出力モードとして開きます。</p>		
使用例3	<p>OPEN "SAMPLE.TXT" FOR APPEND AS #1 SAMPLE.TXTというファイルをファイル番号 1 で追加出力モードとして開きます。</p>		

2.7.18 PATH GET ABSPATH\$

関数		
機 能	相対パス形式の文字列から、絶対パスに変換します。	
書 式	<(戻り値) 絶対パス名> = PATH GET ABSPATH\$(<①パス文字列>)	
戻り値	戻り値	<絶対パス名> 文字列
	相対パス形式の文字列から、絶対パスに相当する部分の文字列が得られます。	
パラメータ	①	<パス文字列> 文字列
	相対パス形式の文字列を与えてください。	
備 考	<ul style="list-style-type: none"> 「~(チルダ)」を指定してのパス変換は、本来シェルスクリプト(/bin/sh)が行います。従って、同様の効果を得たい場合は、「ENVIRON\$("HOME")」の結果値と差し替えるか、realpath というLinux外部コマンドを別途呼び出して、その結果値を得る等をしてください。 	
使用例	<pre>? PATH GET ABSPATH\$("./app") /home/user/test/app ' 相対パスから絶対パスに変換されます。 ? PATH GET ABSPATH\$(GETCOMMANDLINEARGS\$()) ' GETCOMMANDLINEARGS\$と組み合わせると、コマンドラインから実行コマンドを相対パスで呼び出したとしても、必ず絶対パスとなります。</pre>	

2.7.19 PATH GET BASENAME\$

関数		
機 能	パス形式の文字列から、末尾のファイル名に相当する文字部分を取り出します	
書 式	<(戻り値) ベース名> = PATH GET BASENAME\$(<①パス文字列>)	
戻り値	戻り値	<ベース名> 文字列
	パス形式の文字列から、末尾のファイル名に相当する部分の文字列が得られます。	
パラメータ	①	<パス文字列> 文字列
	パス形式の文字列を与えてください。	
備 考	パス文字列の終端が「/」の場合、フォルダ名である事が自明なので、得られる文字列は空文字列です。	
使用例1	<pre>? PATH GET BASENAME\$("/home/user/test.AJN") test.AJN ' ファイルパスの末尾のファイル名が得られます</pre>	
使用例2	<pre>? PATH GET BASENAME\$("/home/user/") ' 末尾が「/」でフォルダ名なのが自明なので、得られる文字列は空文字列です</pre>	

2.7.20 PATH GET DIRNAME\$

関数		
機 能	パス形式の文字列から、ディレクトリ名に相当する文字部分を取り出します	
書 式	<(戻り値)ディレクトリ名> = PATH GET DIRNAME\$(<①パス文字列>)	
戻り値	戻り値	文字列
	<ディレクトリ名> パス形式の文字列から、ディレクトリ名に相当する部分の文字列が得られます。	
パラ メータ	①	文字列
	<パス文字列> パス形式の文字列を与えてください。	
使用例1	? PATH GET DIRNAME\$("/home/user/test.AJN") /home/user ' ファイルパスのディレクトリ名が得られます	
使用例2	? PATH GET DIRNAME\$("/home/user/") /home/user ' 末尾が「/」でフォルダ名なのが自明なので、得られる文字列はパス形式相当です	

2.7.21 PATH JOIN\$

関数		
機 能	1つあるいは複数のパス要素を結合して、パス形式の文字列を作ります。	
書 式	<(戻り値)パス文字列> = PATH JOIN\$(<①パス要素> [, <①パス要素> ...])	
戻り値	戻り値	文字列
	<パス文字列> パス要素を結合して、パス形式の文字列が得られます。	
パラ メータ	①	文字列
	<パス要素> 結合したい、パス要素を指定します。	
使用例	? PATH JOIN\$("/home/", "user", "test.AJN") /home/user/test.AJN ' パス形式の文字列が得られます	

2.7.22 PATH SPLITEXT\$

関数		
機 能	パス形式の文字列から、ファイルの拡張子とそれ以外の文字列の配列に分離します。	
書 式	<(戻り値) 分離配列> = PATH SPLITEXT\$(<①パス文字列>)	
戻り値	戻り値	<分離配列> 文字列
	<p>パス形式の文字列から、ファイルの拡張子とそれ以外の文字列の配列で得られます。</p> <p>以下の形式で、2要素の文字列配列で得られます。</p> <p>添え字0に、ファイルの拡張子以外の部分。</p> <p>添え字1に、ファイルの拡張子部分。</p>	
パラ メータ	①	<パス文字列> 文字列
	パス形式の文字列を与えてください。	
備 考	パス文字列に拡張子が無い時は、添え字1は空文字が得られます。	
使用例	LIST SS\$	
	SS\$ = PATH SPLITEXT\$("/home/user/test.AJN")	
	’ SS\$(0) は、「/home/user/test」が得られます。	
	’ SS\$(1) は、「.AJN」が得られます。	

2.7.23 PRINT, ?

命令		
機 能	文字列や数値等のデータを画面、またはファイルに出力します。	
書 式 1	PRINT <①文字列/数値> [; , <①文字列/数値> ...] [; ,] 文字列や数値等のデータを画面に出力します。	
書 式 2	? <①文字列/数値> [; , <①文字列/数値> ...] [; ,] 書式1と同じです。「?(クエスチョンマーク)」で「PRINT」の代替えとします。	
書 式 3	PRINT #<②ファイル番号>, <①文字列/数値> [; , <①文字列/数値> ...] [; ,] 文字列や数値等のデータをファイルに出力します。	
書 式 4	? #<②ファイル番号>, <①文字列/数値> [; , <①文字列/数値> ...] [; ,] 書式3と同じです。「?(クエスチョンマーク)」で「PRINT」の代替えとします。	
パラ メータ	①	値
	<文字列/数値>	
	出力する文字列/数値を指定します。セミコロン(;)で区切ると、各文字列/数値を連続して出力します。カンマ(,)で区切ると、各文字列/数値間にタブを出力します。最後にセミコロンやカンマを指定すると、改行が起りません。	
パラ メータ	②	数値
	<ファイル番号>	
	出力先のファイル番号を指定します。 ファイル番号に変数を与えたい場合、他の書式の区別に、「#<変数名>」の記述としてください。	
備 考	ファイルに出力する場合、OPENコマンドでFOR OUTPUT/APPENDを指定してください。	
注 意	構造体の表示はできません。	
使用例1	A = 50 PRINT "A="; A A=50 と表示します。	
使用例2	A = 50 B = 60 ? A, B 50 60 と表示します。	
使用例3	PRINT #1, "TEST OK" ファイル番号1で開いているファイルへ TEST OK という文字列を書き込みます。	
使用例4	NUM% = 1 PRINT #NUM%, "TEST OK" 使用例3のファイル番号の指定を、NUM%変数に置き換えた例です。	
使用例5	A = 50 B = 60 ? #1, A, B ファイル番号1で開いているファイルへ変数A,Bの値 50 60 という文字列を書き込みます。	

2.7.24 RMDIR

命令			
機 能	既存のフォルダを削除します。		
書 式	RMDIR <①フォルダ名>		
パラ メータ	①	<フォルダ名>	文字列
	削除するフォルダ名を指定します。		
備 考	・ 削除するフォルダ内にファイルがあると、エラーになります。		
使用例	RMDIR "SAMPLE"		
	SAMPLE という名前の既存のフォルダを削除します。		

2.7.25 SEEKGET

関数			
機 能	ファイルの読み書きする現在位置を取得します。		
書 式	<(戻り値)位置> = SEEKGET(<①ファイル番号>)		
戻り値	戻り値	<位置>	数値
	読み書きする現在位置を、先頭からバイト単位で取得します。		
パラ メータ	①	<ファイル番号>	数値
	対象のファイル番号を指定します。 ファイル番号は、OPEN命令で指定したファイル番号を与えます。		
使用例	OPEN "hoge.txt" FOR INPUT AS #1 LINE INPUT #1, A\$ ' ファイルから1行分読み取ります PRINT "位置="; SEEKGET(1) ' 読み取った現在の位置を表示します		

2.7.26 SEEKSET

命令			
機 能	ファイルから次に読み書きする位置を設定します。		
書 式	SEEKSET #<①ファイル番号>, <②位置>		
パラ メータ	①	<ファイル番号>	数値
	対象のファイル番号を指定します。 ファイル番号は、OPEN命令で指定したファイル番号を与えます。		
	②	<位置>	数値
備 考	ファイルから次に読み書きする位置を、先頭からバイト単位で指定します。		
	・ 入力ファイル時、ファイルサイズを越えた位置を指定した時、INPUT 命令などによる次の読み取り結果は不明です。		
使用例	OPEN "hoge.txt" FOR OUTPUT AS #1 SEEKSET #1, 100 ' ファイル先頭から100バイトの位置に移動 PRINT #1, "hello" ' 100バイトの位置から文字列を書き込む		

2.7.27 STR_FREADALL\$

関数			
機 能	指定したファイルから全データをバイナリ文字列に取り読み取ります。		
書 式	<(戻り値) ファイル内容文字列> = STR_FREADALL\$(<①ファイル名>)		
戻り値	戻り値	<ファイル内容文字列>	文字列
	ファイル名から読み込んだデータが、バイナリ文字列に格納されます。		
パラ メータ	①	<ファイル名>	文字列
	読み込むファイル名を指定します。		
備 考	<ul style="list-style-type: none"> 読み込むデータは、一切の加工なく全データが読み込まれます。データ途中に、ヌルコード(CHR\$(0))が含まれていても、読み込みます。従って、読み込んだデータを、PRINT文で判読可能に表示できるとは限りません。 読み込むデータの終端を強く意識する必要がある場合、データを作成するツールの仕様に注意ください。 例えば、テキストエディタの種類によっては(例: gedit)、データ終端に改行コードを必ず付加するなどの自動処理を行うものがあります。		
使用例	A\$ = STR_FREADALL\$("memo.txt") "memo.txt"というファイルのデータ内容を、A\$に全て読み取ります。		

2.7.28 STR_FWRITEALL

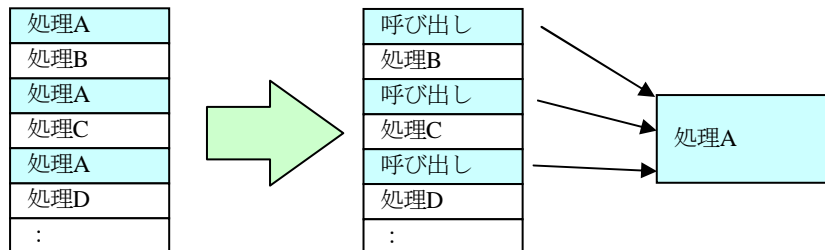
命令			
機 能	バイナリ文字列を指定したファイルに書き込みます。		
書 式	STR_FWRITEALL <①ファイル名>, <②ファイル内容文字列>		
パラ メータ	①	<ファイル名>	文字列
	書き込むファイル名を指定します。		
	②	<ファイル内容文字列>	文字列
備 考	<ul style="list-style-type: none"> 書き込むデータは、一切の加工なく全データを書き込みます。文字列のデータ途中に、ヌルコード(CHR\$(0))が含まれていても、書き込みます。(いわゆるバイナリ文字列形式) 		
使用例	A\$ = "hello world" STR_FWRITEALL "memo.txt", A\$ A\$のデータ内容を、"memo.txt"というファイルに、全て書き込みます。		

2.7.29 WAIT FILE

関数		
機 能	指定したファイル番号の読み書きイベントを待ちます。	
書 式	<(戻り値)イベントステータス> = WAIT FILE(<①ファイル番号> [, <②待ち時間ms>])	
戻り値	戻り値	<イベントステータス> 数値
	イベントが検知されずにタイムアウトになると、0を。 イベントが検知されると、以下の表の、値の組み合わせとなります。	
	値	意味
	&h01	読み取り可能となった。
	&h04	書き込み可能となった。
	&h08	エラーを検知。
	&h10	ハングアップ。ファイルが閉じられた、など。
	&h20	不正な要求。オープンされてない、など。
パラ メータ	①	<ファイル番号> 数値
	OPEN (→「2.7.17 OPEN」) でオープンしたファイル番号を指定します。	
	①	<待ち時間ms> 数値
	読み書きイベントを検知するまでの待機時間をms単位で指定します。 0を指定すると、現在のイベント状態を取得して戻ります。 負数を指定すると、イベントが検知されるまで永遠に待ちます。	
備 考	・ イベントが検知されると、待ち時間を待たずに、即値を抱えて戻ります。	
使用例	<pre>OPEN "/proc/self/fd/0" FOR INPUT AS #1 DO WHILE TRUE IF WAIT FILE(1, 1000) <> 0 THEN LINE INPUT #1, A\$? "読み取った:"; A\$ END IF LOOP</pre> <p>自プロセスの標準入力(/proc/self/fd/0)に対して、何か入力が発生したら、LINE INPUT 命令で、1行分の文字列を読み取ろうとします。 (注意：このコード例は、端末上で直接実行してください。IDE上では動作しません)</p>	

2.8 サブルーチンに関する関数・命令

サブルーチンとは、意味や内容、動きが同じプログラムを1つにまとめ、何度も同じ記述をしなくて済むようにするやり方です。



上記の例では、全く同じ書き方をする「処理A」というプログラムが何度も出てきます。左のように記載すると、プログラムは長くなり、さらにプログラムを修正したい場合、何ヶ所にもある「処理A」をすべて修正する必要があります。これを「処理A」だけを1つにまとめることを、サブルーチン化といい、これにより、プログラムは短くなり、「処理A」の修正は1回で済みます。

サブルーチンの記述は、いくつかの方法があります。

1. CALL文でSUB文に指定したルーチンにジャンプし、END SUB文で戻る方式<例1>
2. FUNCTION文で定義した関数名にジャンプし、END FUNCTION文で戻る方式<例2>

なお、FUNCTION文のみ関数名に値を代入することで、呼び出し元に値を返却することができます。

<例1>

```
' サブルーチン
SUB MENSEKI (A, B)          'サブルーチン MENSEKI
  AA = A*B/2
END SUB

PRINT "三角形の面積を求めます"
PRINT "底辺：2"
TEIHEN = 2
PRINT "高さ：4"
TAKASA = 4
CALL MENSEKI (TEIHEN, TAKASA)  'サブルーチン MENSEKIを呼び出し
PRINT "面積は ";AA
END
```

<例2>

```
' サブルーチン
FUNCTION MENSEKI (A, B)          'サブルーチン MENSEKI
    MENSEKI = A*B/2
END FUNCTION

PRINT "三角形の面積を求めます"
PRINT "底辺：2"
TEIHEN = 2
PRINT "高さ：4"
TAKASA = 4
AA = MENSEKI (TEIHEN, TAKASA)    'サブルーチン MENSEKIを呼び出し
PRINT "面積は ";AA
END
```

2.8.1 <ラベル名>:

命令		
機 能	GOTO文で使用するラベルを設定します。	
書 式	<①ラベル名>:	
パラ メータ	①	<ラベル名> ラベル名
	ラベル名を指定します。 ラベル名の後ろには:(コロン)を付けます。	
使用例	<pre>A = 0 GOTO SAMPLE PRINT "A=", A SAMPLE: A = 10 PRINT "A=", A ----- 実行結果 A=10</pre> <p>SAMPLEというラベル名を設定し、ラベル名を指定して処理を移動する事ができます。</p>	

2.8.2 CALL

命令		
機 能	SUB~END SUBで定義したサブルーチン呼び出します。	
書 式	CALL <①サブルーチン名>[(<②引数> [, <②引数> ...])]	
パラ メータ	①	<サブルーチン名> サブルーチン名
	SUB~END SUBで定義したサブルーチン名を指定します。	
	②	<引数> 値
使用例	サブルーチンの引数を指定します。 引数で渡された値は、サブルーチン内でローカル変数として扱われます。	
	<pre>SUB SMP (A, B\$) ... END SUB NUM = 1 CALL SMP (NUM, "A")</pre> <p><サブルーチン名> で指定した SMP というサブルーチン呼び出します。</p>	

2.8.3 ERROR ON / OFF

命令	
機能	エラー発生による割り込みの許可，禁止を指定します。 割り込みのサブルーチンはON ERROR CALLで定義します。 プログラム終了時にはERROR OFFを実行してください。
書式1	ERROR ON 割り込みを許可します。これ以降、エラーが発生すると、サブルーチンが呼び出されます。
書式2	ERROR OFF 割り込みを禁止します。これ以降、エラーが発生しても、サブルーチンは呼び出されません。
注意点	OPTION ERRORで継続動作を有効にした場合、本機能は無効となります。
備考	ON ERROR CALL (→「2.8.10 ON ERROR CALL」)の制御に用います。
使用例	<pre>SUB SAMPLE(E_N, E_M\$, E_L) ERROR OFF PRINT "ERROR!" END SUB ...</pre> <p>ON ERROR CALL SAMPLE ERROR ON</p> <p>エラーが発生すると、SAMPLEというサブルーチンを呼び出します。 割り込み発生後にERROR OFFで、これ以降の割り込みの禁止を行っています。</p>

2.8.4 EXIT FUNCTION

命令	
機 能	ユーザ定義関数から無条件に抜けます。
書 式	EXIT FUNCTION
備 考	FUNCTION～END FUNCTION(→「2.8.6 FUNCTION～END FUNCTION」)からの脱出に用います。
使用例	<pre> FUNCTION SUM(A, B) IF A+B > 5 THEN EXIT FUNCTION END IF SUM=A+B END FUNCTION EXIT FUNCTION を用いる事で、途中で関数を抜ける事ができます。 </pre>

2.8.5 EXIT SUB

命令	
機 能	サブルーチンから無条件に抜けます。
書 式	EXIT SUB
備 考	SUB～END SUB(→「2.8.14 SUB～END SUB」)からの脱出に用います。
使用例	<pre> SUB SMP(A, B\$) IF A > 5 THEN EXIT SUB END IF PRINT "STRING=", B\$ END SUB CALL SMP(1, "A") EXIT SUB を用いる事で、途中でサブルーチンを抜ける事ができます。 </pre>

2.8.6 FUNCTION～END FUNCTION

命令		
機能	ユーザ定義関数を定義します。	
書式	FUNCTION <①関数名>[(<②引数> [AS <③引数の型>] [, <②引数> …])] [AS <④戻り値の型>] ... <①関数名> = <⑤戻り値> END FUNCTION	
パラメータ	①	<関数名> 関数名 ユーザ定義関数の名前を指定します。 戻り値の型は、変数の型定義と同じように指定します。 例えば、文字列を返す場合は「\$」を最後に付加し、構造体を返す場合は <戻り値の型> で「AS STRUCT 構造体名」を付加します。
	②	<引数> 変数名 ユーザ定義関数の引数を指定します。引数はローカル変数として扱われます。
	③	<引数の型> キーワード ユーザ定義関数の引数の型を指定します。 以下が指定可能です。 LIST, DICT, LIST DICT, BOOL, LIST BOOL, DICT BOOL, LIST DICT BOOL, DATETIME, LIST DATETIME, DICT DATETIME, LIST DICT DATETIME, STRUCT, LIST STRUCT, DICT STRUCT, LIST DICT STRUCT, MEMORY, POINTER, JSON
	④	<戻り値の型> キーワード ユーザ定義関数の戻り値の型を指定します。 以下が指定可能です。 LIST, DICT, LIST DICT, BOOL, DATETIME, STRUCT, OBJECT MEMORY, OBJECT POINTER, OBJECT JSON
	⑤	<戻り値> 値 ユーザ定義関数の戻り値を指定します。
	備考	
注意	<ul style="list-style-type: none">この命令で作成した関数は、再帰呼び出しできます。関数を呼び出せる深さは、最大64段までです。戻り値を記述しなかった場合、数値型であれば0, 文字型であれば空文字("")が戻ります。引数の型で、構造体(STRUCT)を使用する時、必ず「<引数> AS STRUCT <構造体名>」のように、構造体名を付加してください。戻り値の型で、構造体(STRUCT)を使用する場合、必ず「AS STRUCT <構造体名>」のように、構造体名を付加してください。引数の型で、オブジェクト型(MEMORY, POINTER)を使用する場合、引数名はオブジェクト型を示す「@」を指定しつつ、「<引数>@ AS MEMORY」のように、オブジェクト型名(MEMORY, POINTER)を付加してください。戻り値の型で、オブジェクト型(MEMORY, POINTER, JSON)を使用する場合、関数名はオブジェクト型を示す「@」を指定しつつ、「AS OBJECT MEMORY」のように、オブジェクト型名(MEMORY, POINTER, JSON)を付加してください。	
	<ul style="list-style-type: none">この命令で作成した関数の戻り値は、関数と同じ名前の変数に代入する必要があります。ユーザ関数定義内へ外部からジャンプして入ってきたり、ユーザ関数定義内から外部へジャンプしたりするプログラムは、その動作が保証されなくなります。 処理途中からの脱出には、EXIT FUNCTION文(→「2.8.4 EXIT FUNCTION」)を利用ください。ここで定義する関数名は、SUB～END SUBのサブルーチン名、DEFINE THREAD ～	

	END THREADのスレッド名と重複してはいけません。
使用例1	<pre> FUNCTION SUM(A, B) SUM=A+B END FUNCTION PRINT SUM(1, 2) END </pre> <p>FUNCTIONにより定義する例です。 引数AとBを与えるとA+Bの処理を行うSUM という関数を作成します。 関数の処理内容はFUNCTION～END FUNCTIONの間に記述します。</p>
使用例2	<pre> FUNCTION SUM\$(A, B) SUM\$=STR\$(A+B) END FUNCTION PRINT SUM\$(1, 2) END </pre> <p>文字列を返すFUNCTIONの例です。A+Bの計算結果を文字列で返します。</p>
使用例3	<p>配列Vを渡します。 関数内で、受け取った配列に対して、引数Bを足し算した結果を返します。</p> <pre> FUNCTION SMP(ARY AS LIST, B) AS LIST PRINT "配列ARY="; ARY FOR I=0 TO LDIM(ARY)-1 ARY(I) = ARY(I) + B NEXT I SMP = ARY ' 戻り値を代入 END FUNCTION DIM V(10) V = [0 to 10] PRINT "結果="; SMP(V, 100) </pre>

2.8.7 GOTO

命令			
機 能	指定したラベル名へ移動します。		
書 式	GOTO <①ラベル名>		
パラ メータ	①	<ラベル名>	ラベル名
	移動先のラベル名を指定します。		
注 意	ユーザ関数定義内またはサブルーチン内へ外部からジャンプして入ってきたり、ユーザ関数定義内またはサブルーチン内から外部へジャンプしたりするプログラムは、その動作が保証されなくなります。		
使用例	GOTO SAMPLE ... SAMPLE: A = A + 1 ... SAMPLE というラベルへ移動します。		

2.8.8 KEY ON / OFF / STOP

命令			
機能	ファンクションキーによる割り込みの許可、禁止、保留を指定します。 割り込みのサブルーチンはON KEY CALLで定義します。 プログラム終了時にはKEY OFFを実行してください。		
書式1	KEY[(① キー番号)] ON 割り込みを許可します。これ以降、ファンクションキーが押されると、サブルーチンが呼び出されます。		
書式2	KEY[(① キー番号)] OFF 割り込みを禁止します。これ以降、ファンクションキーが押されても、サブルーチンは呼び出されません。		
書式3	KEY[(① キー番号)] STOP 割り込みを保留します。これ以降、ファンクションキーが押されても、サブルーチンは呼び出されません。 KEY STOPで割り込みを保留した状態でファンクションキーが押され、再度KEY ONで割り込みが許可されると、すぐにサブルーチンが呼び出されます。		
パラメータ	①	＜キー番号＞	数値
	割り込み処理を設定するファンクションキーの番号を1～12で指定します。 省略した場合、すべてのファンクションキーによる割り込みを設定します。		
備考	ON KEY CALL (→「2.8.11 ON KEY CALL」)の制御に用います。		
使用例1	<pre>SUB SAMPLE(KEY_NUM) ? "押されたキーはF";KEY_NUM KEY OFF PRINT "TEST-OK" END SUB ...</pre> <pre>ON KEY(1) CALL SAMPLE ON KEY(2) CALL SAMPLE KEY ON</pre> <p>F1キーかF2キーを押されると、SAMPLEというサブルーチンを呼び出します。 引数を省略すると、登録してある全てのファンクションキー割り込みに対して操作します。 SAMPLEサブルーチンでは、押されたキーを表示しています。 割り込み発生後にKEY OFFで、これ以降の割り込みの禁止を行っています。</p>		
使用例2	<pre>SUB SAMPLE(KEY_NUM) KEY(1) STOP PRINT "PUSH-[F";KEY_NUM;"]KEY" END SUB ...</pre> <pre>ON KEY(1) CALL SAMPLE ON KEY(2) CALL SAMPLE KEY(1) ON</pre> <p>F1キーを押すと、SAMPLEというサブルーチンを呼び出します。 割り込みによる処理中にはKEY STOPで割り込みの保留を行っています。 他のキーでは、サブルーチンは呼び出されません。</p>		

2.8.9 ON END CALL

命令			
機能	プログラム終了直前に、呼び出されるサブルーチンを定義します。		
書式	ON END CALL <①サブルーチン名>		
パラメータ	①	<サブルーチン名>	サブルーチン名
	サブルーチン名を指定します。 サブルーチンは、以下の定義に従います。 SUB サブルーチン名() 処理内容 END SUB		
備考	・ END命令やプログラム終端に達したとき、プログラム終了直前に指定したサブルーチンが実行されて、終了します。		
注意	・ この割り込みは、本命令を呼び出した時点で有効となります。 ・ 本命令は、メインスレッドでのみ呼び出してください。 ワーカースレッド内では、呼び出さないでください。		
使用例	SUB CB_END() PRINT "プログラムを終了します" END SUB ... ON END CALL CB_END PRINT "Hello AJAN" END END命令後に、ON END CALLで指定した CB_END サブルーチンが実行されて終了します。		

2.8.10 ON ERROR CALL

命令		
機能	エラーが発生した時に、呼び出されるサブルーチンを定義します。 本命令の呼び出し後、ERROR ONを呼び出すと割り込み有効になります。	
書式	ON ERROR CALL <①サブルーチン名>	
パラメータ	①	<div> <div><サブルーチン名></div> <div>サブルーチン名</div> </div> エラーが発生した際に呼ばれる、サブルーチン名を指定します。 サブルーチンは、以下の定義に従います。 SUB サブルーチン名(<エラー番号>, <エラーメッセージ>, <エラー発生行>) 処理内容 END SUB
注意	<ul style="list-style-type: none"> エラー処理のサブルーチンを定義すると、エラーが発生した際、自動的にエラーメッセージは表示されなくなります。 エラー処理のサブルーチンから戻る時、エラーの発生した次の行に戻ります。 エラー処理中にエラーを起こした場合、そこでプログラムは停止します。 (エラー処理のサブルーチンが多重に呼び出されることはありません) ERROR ONでエラー検出を有効にしている間は、本命令を使用することはできません。一度ERROR OFFで無効にしてから使用してください。 本命令を呼び出し後、ERROR ON で割り込みが有効になります。 スレッドを使用する際、エラー処理のサブルーチンの登録は、スレッド毎に行う必要があります。 OPTION ERRORで継続動作を有効にした場合、本機能は無効となります。 	
使用例1	<pre> SUB SAMPLE(E_N, E_M\$, E_L) PRINT "エラーが発生しました。" PRINT "エラーコード:"; E_N PRINT "エラーメッセージ:"; E_M\$ PRINT "エラー行:"; E_L END SUB ... ON ERROR CALL SAMPLE ERROR ON エラー発生時に SAMPLE というサブルーチンへ分岐するように定義します。 サブルーチンに引数を3つ(数値、文字列、数値)と定義すると、サブルーチンが呼び出された時点のエラー番号、エラーメッセージ、エラー行番号が各引数に格納されます。 </pre>	
使用例2	<pre> ON ERROR CALL SAMPLE ERROR ON PRINT "エラー処理ルーチンが有効化されました" ERROR OFF PRINT "エラー処理ルーチンが無効化されました" 登録したエラー処理ルーチンを無効化するには、ERROR OFFを呼び出します。 </pre>	

2.8.11 ON KEY CALL

命令			
機能	ファンクションキーが押された時に、呼び出されるサブルーチンを定義します。 本命令の呼び出し後、KEY ONを呼び出すと割り込み有効になります。		
書式	ON KEY(<①キー番号>) CALL <②サブルーチン名>		
パラメータ	①	<キー番号>	数値
	割り込み処理を設定するファンクションキーの番号を1~12で指定します。		
	②	<サブルーチン名>	サブルーチン名
	ファンクションキー押した際に呼ばれる、サブルーチン名を指定します。 サブルーチンは、以下の定義に従います。 SUB サブルーチン名(<キー番号>) 処理内容 END SUB		
注意	<ul style="list-style-type: none">・ トレース実行中には使用できません。・ OSで割り当てられているファンクションキーを押すと、同時にその機能も動作します。実行には、Linux上で、xtermコマンドで端末を起動させることをお勧めします。・ KEY ONでファンクションキー割り込みを有効にしている間は、本命令を使用することはできません。一度KEY OFFで無効にしてから使用してください。・ 本命令を呼び出し後、KEY ON で割り込みが有効になります。・ その他の割り込みコマンドと併用した際の、割り込み発生順番は、割り込み処理のサブルーチンの登録順です。・ LINE INPUTなど、一部のキーボードからの入力コマンドを呼び出し中は、ON KEY CALLによる割り込みは、機能しません。(LINE INPUTの入力処理が、OSにより優先されます)。これは、マルチスレッド使用時でも同じです。		
使用例	<pre>SUB SUB_TEST(FNUM) ? "ファンクション番号"; FNUM; "が押されました" END SUB ... ON KEY(1) CALL SUB_TEST ON KEY(2) CALL SUB_TEST KEY ON</pre> <p>押されたファンクションキーを表示します。F1キーかF2キーが押されるとSUB_TESTが呼び出されます。 サブルーチン引数のFNUMに押されたファンクションキーの番号が格納されます。</p>		

2.8.12 ON TIME\$ CALL

命令			
機能	指定時刻に、呼び出されるサブルーチンを定義します。 本命令の呼び出し後、TIME\$ ONを呼び出すと割り込み有効になります。		
書式	ON TIME\$("<①時刻>") CALL <②サブルーチン名>		
パラメータ	①	<時刻>	文字列
	サブルーチンへ分岐する時刻を"hh:mm:ss"形式で指定します。 特定の時分秒を指定して他を"*"にすると、特定の時間周期を指定できます。 "*:*:00"とした場合、毎分0秒の時点でサブルーチンへ分岐します。 "0*:1*:00"といった"*"の前に数字を指定することはできません。		
	②	<サブルーチン名>	サブルーチン名
	サブルーチン名を指定します。 サブルーチンは、以下の定義に従います。 SUB サブルーチン名(<時刻文字列>) 処理内容 END SUB 引数の時刻文字列には、サブルーチンが呼び出された時刻が文字列で格納されます。		
注意	<ul style="list-style-type: none"> TIME\$ ONで有効にしている間は、本命令を使用することはできません。一旦TIME\$ OFFで無効にしてから使用してください。 本命令を呼び出し後、TIME\$ ON で割り込みが有効になります。 その他の割り込みコマンドと併用した際の、割り込み発生順番は、割り込み処理のサブルーチンの登録順です。 		
使用例1	<pre> SUB SAMPLE(TM\$) PRINT TM\$;"になりました。" END SUB ... ON TIME\$("<*:00:00">") CALL SAMPLE TIME\$ ON </pre> <p>毎正時にSAMPLE というサブルーチンへ分岐するように定義します。 サブルーチンでは呼び出された時刻を表示します。</p>		

2.8.13 ON TIMER CALL

命令

機能	指定時間間隔に、呼び出されるサブルーチンを定義します。 本命令の呼び出し後、TIMER ONを呼び出すと割り込み有効になります。								
書式1	ON TIMER(<①間隔ms>) CALL <②サブルーチン名> この書式では、スレッド毎に設定が有効となります。 TIMER ON / OFF / STOPの呼び出しは、この命令と同じスレッドで呼び出してください。								
書式2	ON TIMER(<③タイマ番号>, <①間隔ms> [, <④モード>]) CALL <②サブルーチン名> この書式では、タイマ番号毎に設定が有効となります。 TIMER ON / OFF / STOPの呼び出しは、設定したタイマ番号と同じ引数で呼び出してください。								
パラメータ	①	<間隔ms>	数値						
	サブルーチンへ分岐する間隔を1以上のミリ秒単位で指定します。								
	②	<サブルーチン名>	サブルーチン名						
	サブルーチン名を指定します。 サブルーチンは、以下の定義に従います。 <書式1で定義した場合> SUB サブルーチン名() 処理内容 END SUB <書式2で定義した場合> SUB サブルーチン名(<タイマ番号>) 処理内容 END SUB								
	③	<タイマ番号>	数値						
	指定するタイマ番号を、1～15の範囲で指定します。 タイマ番号毎に、間隔msの指定が可能です。 書式2で有効です。								
④	<モード>	数値							
タイマの動作モードを切り替えます。 この引数は、2020/11から追加されました。									
<table><tr><th>設定値</th><th>動作</th></tr><tr><td>0</td><td>従来モードです。引数を省略した際も同じです。 以下のように、動作します。 1. 間隔ms経過すると、サブルーチンが呼ばれます。 2. サブルーチンから戻ってから、また間隔ms経過したら、次のサブルーチンが呼ばれます。</td></tr><tr><td>1</td><td>定周期呼び出しモードです。 以下のように、動作します。 1. 間隔ms経過すると、サブルーチンが呼ばれます。 2. 次の間隔ms経過した時、サブルーチン呼び出しから戻っていれば、次のサブルーチンが呼ばれます。 3. 2.の時点で、まだサブルーチン呼び出しから戻ってない時、戻り次第呼ばれますが、呼ばれる時間間隔は不定です。</td></tr></table>				設定値	動作	0	従来モードです。引数を省略した際も同じです。 以下のように、動作します。 1. 間隔ms経過すると、サブルーチンが呼ばれます。 2. サブルーチンから戻ってから、また間隔ms経過したら、次のサブルーチンが呼ばれます。	1	定周期呼び出しモードです。 以下のように、動作します。 1. 間隔ms経過すると、サブルーチンが呼ばれます。 2. 次の間隔ms経過した時、サブルーチン呼び出しから戻っていれば、次のサブルーチンが呼ばれます。 3. 2.の時点で、まだサブルーチン呼び出しから戻ってない時、戻り次第呼ばれますが、呼ばれる時間間隔は不定です。
設定値	動作								
0	従来モードです。引数を省略した際も同じです。 以下のように、動作します。 1. 間隔ms経過すると、サブルーチンが呼ばれます。 2. サブルーチンから戻ってから、また間隔ms経過したら、次のサブルーチンが呼ばれます。								
1	定周期呼び出しモードです。 以下のように、動作します。 1. 間隔ms経過すると、サブルーチンが呼ばれます。 2. 次の間隔ms経過した時、サブルーチン呼び出しから戻っていれば、次のサブルーチンが呼ばれます。 3. 2.の時点で、まだサブルーチン呼び出しから戻ってない時、戻り次第呼ばれますが、呼ばれる時間間隔は不定です。								
注意	・ TIMER ONで有効にしている間は、本命令を使用することはできません。一旦TIMER OFFで無効にしてから使用してください。								

	<ul style="list-style-type: none"> ・ 本命令を呼び出し後、TIMER ON で割り込みが有効になります。 ・ その他の割り込みコマンドと併用した際の、割り込み発生順番は、割り込み処理のサブルーチンの登録順です。 ・ 指定した間隔msの呼び出し間隔精度は、動作時のCPUの性能および負荷状況により変化します。定周期サンプリングなど、精度を確保したい場合は、ハードウェアによる支援機能を持つ、別の手段を検討ください。 ・ 割り込み処理で呼ばれるサブルーチンは、別スレッドで呼ばれます。 呼び出されるサブルーチン先でエラー処理を行いたい場合は、割り込み処理先の ON ERROR CALL 命令でエラー処理の定義が必要です。
使用例1	<pre> SUB SAMPLE PRINT "10秒経過しました。" END SUB ... ON TIMER(10000) CALL SAMPLE TIMER ON </pre> <p>10000ミリ秒(10秒)間隔でSAMPLEというサブルーチンへ分岐するように定義します。</p>
使用例2	<pre> SUB CB_TM(TNO) PRINT "タイマ番号="; TNO; "の呼び出し" END SUB ON TIMER(1, 500) CALL CB_TM ' 500ms毎のタイマ TIMER ON 1 ON TIMER(2, 1000) CALL CB_TM ' 1000ms毎のタイマ TIMER ON 2 </pre> <p>書式2の定義を使って、複数のタイマを定義します。</p>

2.8.14 SUB～END SUB

命令			
機能	サブルーチンを定義します。		
書式	SUB <①サブルーチン名>[(<②引数> [AS <③引数の型>] [, <②引数> …])] … END SUB		
パラメータ	①	<サブルーチン名>	サブルーチン名
	サブルーチンの名前を指定します。		
	②	<引数>	変数名
	ユーザ定義関数の引数を指定します。引数はローカル変数として扱われます。		
パラメータ	③	<引数の型>	キーワード
	ユーザ定義関数の引数の型を指定します。		
	以下が指定可能です。		
	LIST, DICT, LIST DICT, BOOL, LIST BOOL, DICT BOOL, LIST DICT BOOL, DATETIME, LIST DATETIME, DICT DATETIME, LIST DICT DATETIME, STRUCT, LIST STRUCT, DICT STRUCT, LIST DICT STRUCT, MEMORY, POINTER, JSON		
備考	<ul style="list-style-type: none"> この命令で作成したサブルーチンは、再帰呼び出しできます。 サブルーチンを呼び出せる深さは、最大64段までです。 引数の型で、構造体(STRUCT)を使用する時、必ず「<引数> AS STRUCT <構造体名>」のように、構造体名を付加してください。 引数の型で、オブジェクト型(MEMORY, POINTER, JSON)を使用する場合、引数名はオブジェクト型を示す「@」を指定しつつ、「<引数>@ AS MEMORY」のように、オブジェクト型名(MEMORY, POINTER, JSON)を付加してください。 		
注意	<ul style="list-style-type: none"> サブルーチン定義内へ外部からジャンプして入ってきたり、サブルーチン定義内から外部へジャンプしたりするプログラムは、その動作が保証されなくなります。処理途中からの脱出には、EXIT SUB文(→「2.8.5 EXIT SUB」)を利用ください。 ここで定義するサブルーチン名は、FUNCTION～END FUNCTIONの関数名、DEFINE THREAD～END THREADのスレッド名と重複してはいけません。 		
使用例1	SUB SMP … END SUB CALL SMP 引数なしのサブルーチン SMP を定義します。		
使用例2	SUB SMP(A, B\$) … END SUB CALL SMP(1, "A") 引数に数値型変数Aと文字型変数B\$を持ったサブルーチン SMP を定義します。 変数Aに数値 1, 変数B\$にAという文字を受け取っています。		
使用例3	配列Vを渡します。サブルーチンで受け取った配列内容を表示します。 SUB SMP(ARY AS LIST) PRINT "配列ARY="; ARY END SUB DIM V(10) V = [0 to 10]		

	CALL SMP (V)
--	--------------

2.8.15 TIME\$ ON / OFF / STOP

命令	
機 能	TIME\$割り込みの許可，禁止，保留を指定します。 割り込みの発生条件、サブルーチンはON TIME\$ CALLで定義します。 プログラム終了時にはTIME\$ OFFを実行してください。
書 式1	TIME\$ ON 割り込みを許可します。これ以降、指定の時刻になると、サブルーチンが呼び出されます。
書 式2	TIME\$ OFF 割り込みを禁止します。これ以降、指定の時刻になっても、サブルーチンは呼び出されません。
書 式3	TIME\$ STOP 割り込みを保留します。これ以降、指定の時刻になっても、サブルーチンは呼び出されません。 TIME\$ STOP で割り込みを保留している間に指定時刻になり、その後TIME\$ ONで再度割り込みが許可されると、すぐにサブルーチンが呼び出されます。
備 考	ON TIME\$ CALL(→「2.8.12 ON TIME\$ CALL」)での割り込み制御に用います。
使用例	<pre>SUB SAMPLE(TM\$) TIME\$ STOP PRINT "割り込み保留中" END SUB ... ON TIME\$("00:00:00") CALL SAMPLE TIME\$ ON ... TIME\$ OFF</pre> <p>TIME\$割り込みに対して、割り込み許可、割り込み保留、割り込み禁止をそれぞれ指定しています。</p>

2.8.16 TIMER ON / OFF / STOP

命令		
機能	TIMER割り込みの許可、禁止、保留を指定します。 割り込みの発生条件、サブルーチンはON TIMER CALLで定義します。 プログラム終了時にはTIMER OFFを実行してください。	
書式1	TIMER ON 割り込みを許可します。 これ以降、指定の間隔が経つと、サブルーチンが呼び出されます。 「ON TIMER CALL」の書式1の定義と対応します。	
書式2	TIMER OFF 割り込みを禁止します。 これ以降、指定の間隔が経っても、サブルーチンは呼び出されません。 「ON TIMER CALL」の書式1の定義と対応します。	
書式3	TIMER STOP 割り込みを保留します。 これ以降、指定の間隔が経っても、サブルーチンは呼び出されません。 TIMER STOPで割り込みを保留している間に指定の間隔が経ち、その後再度TIMER ONで割り込みが許可されると、すぐにサブルーチンが呼び出されます。 「ON TIMER CALL」の書式1の定義と対応します。	
書式4	TIMER ON <①タイマ番号> 割り込みを許可します。 これ以降、指定の間隔が経つと、サブルーチンが呼び出されます。 「ON TIMER CALL」の書式2の定義と対応します。	
書式5	TIMER OFF <①タイマ番号> 割り込みを禁止します。 これ以降、指定の間隔が経っても、サブルーチンは呼び出されません。 「ON TIMER CALL」の書式2の定義と対応します。	
書式6	TIMER STOP <①タイマ番号> 割り込みを保留します。 これ以降、指定の間隔が経っても、サブルーチンは呼び出されません。 TIMER STOPで割り込みを保留している間に指定の間隔が経ち、その後再度TIMER ONで割り込みが許可されると、すぐにサブルーチンが呼び出されます。 「ON TIMER CALL」の書式2の定義と対応します。	
パラメータ	①	<タイマ番号> 数値 「ON TIMER CALL」の書式2 で、設定したタイマ番号を指定します。
備考	ON TIMER CALL(→「2.8.13 ON TIMER CALL」)での割り込み制御に用います。	
使用例1	<pre> SUB SAMPLE TIMER STOP PRINT "割り込み保留中" END SUB ... ON TIMER(1000) CALL SAMPLE TIMER ON ... TIMER OFF </pre> <p>TIMER割り込みに対して、割り込み許可、割り込み保留、割り込み禁止をそれぞれ指定しています。</p>	

2.9 スレッドに関する関数・命令

2.9.1 スレッドについて

AJANは、マルチスレッドを扱う事ができます。

スレッドとは、複数のAJANプログラムやサブルーチンを同時に並行して動かす場合の処理の単位です。複数同時に動かす場合を特にマルチスレッドと呼びます。

処理の単位としては、スレッドの他にプロセスがあります。プロセスは通常メモリ空間が独立していますが、スレッドは同一という違いがあります。

マルチスレッドを用いる用途としては、ボタンなどのユーザが操作する処理のバックグラウンドで、計測データの計算処理を行いたい、といった時に多く用いられます。

この場合、メインのスレッドで対話処理を行いつつ、別のスレッドでは計算処理を実行します。計算結果が得られたら、メインのスレッドと計算処理のスレッドが情報のやり取りを行う事で、対話処理を遅滞なく行う事ができるようになります。

AJANでは、プログラム開始時に動くスレッド(=プログラム/サブルーチン)を、「メインスレッド」と呼びます。メインスレッドは、プログラムの実行中に必ず存在します。

それに対して、「ATTACH THREAD」命令によって任意に生成できる別スレッドを「ワーカースレッド」と呼びます。生成できるワーカースレッドの最大数は15です。

AJANでは、メインスレッドとワーカースレッドを合わせて、同時に16のスレッドが動かせます。

メインスレッドは他のスレッドに比べて特別扱いされます。

例えば、メインスレッドで「END」命令が実行されると他のワーカースレッドはすべて終了しますが、ワーカースレッドで「END」命令が実行されると自身のスレッドのみ終了します。

各スレッドをデバッグするにはブレークポイントを「ATTACH THREAD」命令の直後に設定し、デバッグ実行で実行したあと、プログラムを中断した状態でIDE上のスレッドのプログラムタブを選択して、デバッグしたいスレッドに切り替えてください。

メインスレッドとワーカースレッドの特徴を、下表にまとめます。

	メインスレッド	ワーカースレッド
生成方法	プログラム実行中は必ず存在します。	「ATTACH THREAD」命令で任意に生成できます。
実行数	1つしか存在しません。	15まで生成できます。
終了方法	「END」命令で終了します。	以下のいずれかの方法で終了します。 <ul style="list-style-type: none">・自身からの「END」命令。・他者からの「DETACH THREAD」命令。・メインスレッドからの「END」命令。
制限事項	特にありません。	使用できない命令・関数があります。(例：GUI系のコマンドなど)

スレッド同士での情報のやり取りは、グローバル変数にて行います。

例えば、「CNT」というグローバル変数を、ワーカースレッドがインクリメントして、メインスレッドが参照しようとする場合、プログラムを以下のように記述します。

<メインスレッド側のプログラム>

```
CNT = 0
ATTACH THREAD 1, THREAD_1(0)      ' ワーカースレッドを生成&実行

DO WHILE CNT < 10
  ? "CNT の値:", CNT              ' CNT の値を参照して PRINT
LOOP

END
```

<ワーカースレッド側のプログラム>

```
DEFINE THREAD THREAD_1(NUMBER, USERDATA)
THREAD_LABEL:
  CNT = CNT + 1                    ' CNT の値をインクリメント
  DO WHILE TM < 10000
    TM = TM+1
  LOOP
  GOTO THREAD_LABEL
END THREAD
```

スレッド機能を使うプログラムをデバッグする場合、ワーカースレッド固有の挙動として以下の特徴があります。

1. ワーカースレッドはプログラムの実行状態と同じく、実行、終了、停止の3状態の他に、「存在しない」状態が在ります。
ワーカースレッドは、任意に生成&実行される為、最初は「存在しない」状態です。
2. 通常時、ワーカースレッドの終了は、そのまま「存在しない」状態に遷移します(終了状態には遷移しません)。
ワーカースレッドの停止は、メインスレッドと同じように停止で待機します。
3. プログラム実行時、生成されるすべてのワーカースレッドは、実行状態になります。
メインスレッドまたはワーカースレッドのいずれかが停止すると、すべてのワーカースレッドは停止し、メインスレッドが終了すると、すべてのワーカースレッドは終了します。
なお、ワーカースレッドが終了しても他のスレッドは動き続けます。
4. IDEのステップ実行は、選択されたスレッドのみプログラムを進めます。
5. 「ON ERROR CALL」命令などのエラー処理系は、スレッド毎に宣言する必要があります。「ERR」関数で得られるエラーコードは、スレッド毎に独立しています。
6. ワーカースレッド内で変数のLOCAL宣言は有効になります。

基本的に、どれかのスレッドでエラー等が発生し、「ON ERROR CALL」命令が無い場合、全スレッドが中断します。

どれかのスレッドだけを動かす事ができるのは、トレース等のデバッグ実行をAJANから指定したときのみです。

また、マルチスレッドでプログラミングを行う場合、一部のコマンドや関数（例えば「ON ERROR CALL」コマンドや「ERR」関数）は、スレッド毎に状態を持つ事に留意してください。

例えばワーカースレッド内でエラーにより処理が中断した状態で、ワーカースレッドで「?ERR」とした時は、ワーカースレッド内で発生したエラーコード値が得られます。しかし、メインスレッドを選択して「?ERR」とした時はメインスレッド側では何もエラーが発生していないので、正常を示す「0」が得られます。

主な操作とそのときの挙動をまとめると以下のようになります。

操作	挙動
実行	すべてのスレッドは実行状態に入ります。
停止 (IDEでの中断含む)	すべてのスレッドは停止状態に入ります。
終了	メインスレッドから実行すると、すべてのスレッドは終了状態に入ります。 ワーカースレッドから実行すると、自身のスレッドのみ終了状態に入ります。

2.9.2 ATTACH THREAD

命令			
機 能	スレッドを生成します。		
書 式	ATTACH THREAD <①スレッド番号>, <②スレッド名> [(<③ユーザデータ>)]		
パラ メータ	①	<スレッド番号>	数値
	生成するスレッド番号を1～15の範囲で指定します。		
	②	<スレッド名>	スレッド名
	スレッドを生成して実行するスレッド名を指定します。 スレッドはDEFINE THREAD(→「DEFINE THREAD ～ END THREAD」)にて定義してください。		
	③	<ユーザデータ>	値
	DEFINE THREADで定義したユーザデータへ渡す値を指定します。 DEFINE THREADでユーザデータを渡さないように記述した場合、この指定は省略してください。		
備 考	・ 既にスレッドが稼働中の場合、エラーとなります。		
使用例1	<pre> DEFINE THREAD TEST(NUMBER, USERDATA) THREAD_LABEL: CNT = CNT + 1 TM = 0 DO WHILE TM < 10000 TM = TM+1 LOOP GOTO THREAD_LABEL END THREAD CNT = 0 ATTACH THREAD 1, TEST(0) DO WHILE CNT < 10 ? CNT LOOP DETACH THREAD 1 END ATTACH THREADで、TESTルーチンを実行するスレッド1を作成します。 スレッド1は、変数CNTをインクリメントします。 メインスレッドは、変数CNTを監視して、10を超えたら終了します。 </pre>		
使用例2	<pre> DEFINE THREAD TEST(NUM) PRINT "スレッド番号"; NUM END THREAD ATTACH THREAD 1, TEST DEFINE THREADで、TESTルーチンにユーザデータを渡さないよう記述した場合、ATTACH THREADも、ユーザデータは指定しません。 </pre>		

2.9.3 DETACH THREAD

命令			
機能	スレッドを終了します。		
書式	DETACH THREAD <①スレッド番号> [, ②NOCHK]		
パラメータ	①	<スレッド番号>	数値
	終了するスレッド番号を指定します。		
	②	<NOCHK>	キーワード
	NOCHKを指定すると、スレッドが動作しているか否かチェックしません。 省略すると、スレッドが動作していないとエラーになります。 このオプションは、スレッド自身がEND等で終了する前に、まだ動作しているかもしれないスレッドを終了させるのに用います。		
備考	<ul style="list-style-type: none"> スレッドが終了するまで待機します。 		
注意	<ul style="list-style-type: none"> 「DETACH THREAD」コマンドは、他のスレッドから任意のスレッドを終了する際に用います。メインスレッド以外のワーカースレッドで、自身が終了する場合は「END」コマンドを利用してください。 スレッドの終了は、実行しているスレッドの行間のタイミングで終了します。例えば、SLEEP 命令で、異様に長い待ち時間を設定すると、スレッドが終了して戻るまで、長い時間待ち続ける事になります。 		
使用例1	<pre> DEFINE THREAD TEST(NUMBER, USERDATA) THREAD_LABEL: CNT = CNT + 1 DO WHILE TM < 10000 TM = TM+1 LOOP GOTO THREAD_LABEL END THREAD CNT = 0 ATTACH THREAD 1, TEST(0) DO WHILE CNT < 10 ? CNT LOOP DETACH THREAD 1 END ATTACH THREADで、TESTルーチンを実行するスレッド1を作成します。 DETACH THREADで、稼働中のスレッド1を停止させます。 </pre>		
使用例2	<pre> DEFINE THREAD TEST(NUMBER, USERDATA) THREAD_LABEL: CNT = CNT + 1 DO WHILE TM < 10000 TM = TM+1 LOOP IF CNT > USERDATA THEN END GOTO THREAD_LABEL END THREAD </pre>		

	<pre>CNT = 0 ATTACH THREAD 1, TEST(10) DO WHILE CNT < 10 ? CNT LOOP DETACH THREAD 1, NOCHK END</pre> <p>NOCHK を指定することで、スレッド1が稼働中の場合はスレッドを停止し、すでに停止している場合はそのまま通過します。</p>
--	--

2.9.4 DEFINE THREAD ～ END THREAD

命令		
機 能	スレッドを定義します。 ATTACH THREADを呼び出す事でスレッドとして起動し、動作します。	
書 式	DEFINE THREAD <①スレッド名>(<②スレッド番号>[, <③ユーザデータ>]) ... END THREAD	
パラ メータ	①	スレッド名 スレッド名を指定します。
	②	スレッド番号 スレッド番号を格納するための変数を定義します。 スレッドが起動した時に、スレッド番号が格納されます。
	③	ユーザデータ ユーザデータを格納するための変数を定義します。 ATTACH THREADで指定したユーザ指定の値が格納されます。 ATTACH THREADでデータの値を指定する場合は、必ずDEFINE THREADでも指定する必要があります。
注 意	<ul style="list-style-type: none"> スレッド定義内へ外部からジャンプして入ってきたり、スレッド定義内から外部へジャンプしたりするプログラムは、その動作が保証されなくなります。 ここで定義するスレッド名は、FUNCTION～END FUNCTIONの関数名、SUB～END SUBのサブルーチン名と重複してはいけません。 DEFINE THREAD～END THREADの間に、以下の命令は記述することはできません。 <ul style="list-style-type: none"> SUB～END SUB FUNCTION～END FUNCTION DEFINE THREAD～END THREAD 	
使用例	<pre> DEFINE THREAD TEST (NUMBER, USERDATA) THREAD_LABEL: CNT = CNT + 1 DO WHILE TM < 10000 TM = TM+1 LOOP GOTO THREAD_LABEL END THREAD CNT = 0 ATTACH THREAD 1, TEST(0) DO WHILE CNT < 10 ? CNT LOOP DETACH THREAD 1 END DEFINE THREADで、スレッドを定義します。 </pre>	

2.9.5 LOCK

命令	
機 能	マルチスレッド／割り込み時の排他処理用に、ロックをかけます。 LOCK～UNLOCKまでの間、他の処理に対してブロックします。
書 式	LOCK
注 意	・ 同じスレッド／割り込みで、LOCK中にLOCKを呼び出さないでください。 間違っって呼び出した際、OSの動作に依存しますが、デッドロックします。
使用例	2つのスレッドから同じルーチン (TH) が呼び出されますが、LOCK - UNLOCKまでの間の処理は、ロックを掛ける為に、安全に変数Xが更新できます。 DEFINE THREAD TH(NUM) DO WHILE TRUE LOCK X = X + 1 UNLOCK SLEEP 0.5 LOOP END THREAD ATTACH THREAD 1, TH ATTACH THREAD 2, TH

2.9.6 THREAD INFO

関数			
機 能	スレッドに関する情報を取得します。		
書 式	<(戻り値)ステータス> = THREAD INFO(<①情報ID>)		
戻り値	戻り値	<ステータス>	数値
	情報IDで指定した、スレッドの情報が得られます。		
	情報ID	得られる内容	
	"FREE"	使われてないスレッド番号を得ます。 全て使用中で空きが無い場合、0 が得られます。	
	"SELF"	自身のスレッド番号を得ます。	
パラメータ	①	<情報ID>	文字列
情報を取得したい種別を、文字列で指定します。			
使用例	DEFINE THREAD CB_TH(TH%) SLEEP 1.0 PRINT "自分のスレッド番号="; THREAD INFO("SELF") END THREAD NUM% = THREAD INFO("FREE") ' 未使用のスレッド番号を得ます ATTACH THREAD NUM%, CB_TH ' 未使用のスレッド番号を指定します 使われてないスレッド番号に対して、スレッドを起動します。		

2.9.7 THREAD STATUS

関数			
機 能	スレッドが動作しているか確認します。		
書 式	<(戻り値)動作状態> = THREAD STATUS(<①スレッド番号>)		
戻り値	戻り値	<動作状態>	真偽値
	スレッドが動作しているならTRUE。動作していないならFALSE が返ります。		
パラ メータ	①	<スレッド番号>	数値
	動作確認を行うスレッド番号を指定します。		
使用例	<pre>DEFINE THREAD TEST1(NUM) SLEEP 1.0 END THREAD ATTACH THREAD 1, TEST1 DO WHILE THREAD STATUS(1) ? "スレッド動作中" LOOP ? "スレッド終了した"</pre>		

2.9.8 UNLOCK

命令	
機 能	マルチスレッド／割り込み時の排他処理用に、ロックを解除します。 LOCK～UNLOCKまでの間、他の処理に対してブロックします。
書 式	UNLOCK
注 意	・ 同じスレッド／割り込みで、未LOCK中にUNLOCKを呼び出さないでください。 間違って呼び出した際、OSの動作に依存しますが、デッドロックします。
使用例	LOCKの例を参照ください。

2.10 プロセス連携に関する関数・命令

2.10.1 プロセス連携について

プロセス連携は、「ip」コマンドや「nkf」コマンドなどのLinuxコマンド、外部プログラムを呼び出す機能です。

例えば、IPアドレスを知るために、Linux端末上で「ip a」と入力すると、以下のような情報が得られます。

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
   link/ether 00:80:62:81:67:b2 brd ff:ff:ff:ff:ff:ff
   inet xx.xx.xx.xx/24 brd 192.168.41.255 scope global eth0
       valid_lft forever preferred_lft forever
<略>
```

この呼び出しを、AJANで実行するには、目的により、呼び出すコマンドに違いがあります。

単にコマンドを呼び出したい	SHELL , SHELL RUN または SHELL SYSTEM
呼び出し結果値を知りたい	SHELL CODE または SHELL SYSTEM
コマンドが出力する文字列を得たい	SHELL OUTPUT\$ または SHELL CALLOUT\$

コマンドが出力する文字列を得たい場合は、SHELL CALLOUT\$ 関数を呼び出すか、SHELL コマンドを呼び出した後に SHELL OUTPUT\$ 関数で、文字列を取り込みます。

コマンドの呼び出し結果値を得るには、SHELL SYSTEM 関数を呼び出すか、SHELL コマンドを呼び出した後に SHELL CODE 関数で、値を得ます。

ここで、コマンドの呼び出し結果値とは、一般に Linuxで 外部コマンドを実行して終了すると、コマンドは終了コードと呼ばれる 整数値を返す事になっています。

一般的には、コマンドの実行が正常に終了したら 0の数値を、異常または想定外の結果となったら 0 以外の数値を返します。(どの値を返すかは、各コマンドのヘルプを参照ください)

AJANは、この値を SHELL SYSTEM または SHELL CODE 関数で得られる事ができます。

2.10.2 プロセス連携のTips

プロセス連携を行う際のTipsについて、幾つか紹介します。



ここで紹介する Tipsは、Linuxのコマンド呼び出しの技術紹介と同じです。

■ コマンドの出力結果を元に、別のコマンドで加工したい

Linuxでは、コマンドの出力結果を元に、別のコマンドで加工処理する事が簡単にできます。

例えば、以下のコマンド呼び出しは、「ip a」コマンドの出力結果を元に、「grep "inet "」で文字列をフィルタリングした結果を出力するというものです。

各コマンドの呼び出しを繋ぐのは、「| (パイプ)」記号です。

```
$ ip a | grep "inet "
  inet 127.0.0.1/8 scope host lo
  inet xxx.xxx.xxx.xxx/24 brd 192.168.41.255 scope global eth0
  inet yyy.yyy.yyy.yyy/24 brd 10.80.40.255 scope global dynamic eth1
```

AJANでも、同じ記述が可能です。

```
S$ = SHELL CALLOUT$("ip a | grep ""inet """)
```

■ コマンドの終了結果を待たずに、次に進められるようにしたい

基本的に、プロセス連携のコマンドは、外部コマンドの実行が終了するまで戻ってきません。

これは、Linuxの端末上で、外部コマンドを実行するときも同じです。

Linuxでは、コマンド実行をバックグラウンドで実行させて、次のコマンド呼び出しをさせる事ができます。

例えば、以下のように呼び出すと、「xeyes」という目玉のプログラムを実行させつつ、次の「pwd」コマンドの実行が可能です。

```
$ xeyes &
[1] 11148
$ pwd
/home/user
```

上の例で、「xeyes」コマンドをバックグラウンドで実行させたのは、文末に付けた「&(アンパサンド)」記号です。これを付加する事で、前に指定したコマンドをバックグラウンドで実行可能となります。

AJANでも、同じ記述が可能です。

```
S$ = SHELL SYSTEM("xeyes &")      ' xeyes が完了するのを待たずに、次の行に実行できる
S$ = SHELL SYSTEM("pwd")
```

- 「| (パイプ)」を使ったコマンドを呼び出していると、「Broken pipe」の警告が出るケース
「| (パイプ)」記号を使うと、コマンドの出力結果を自由にフィルタ加工できるので、非常に便利です。

以下のように、AJANコマンドの呼び出し事例は、「cat」コマンドで、「hoge.txt」ファイルの内容をテキスト出力しつつ、次の「head」コマンドの「-n1」オプションで、先頭1行のみをフィルタした結果がPRINT出力されます。

```
PRINT SHELL CALLOUT$("cat hoge.txt | head -n1")
```

もし、「hoge.txt」が非常に大きなファイルだと、上のプログラムを実行した時、以下のように意図しないメッセージ(赤文字部分)が出力される場合があります。

```
$ ./test1 <=== AJAN プログラムを呼び出したとします
先頭 1 行のメッセージ
cat: 書き込みエラー: Broken pipe
```

これは「cat」コマンドのファイル出力が続いている中で、「head」コマンドが先頭1行を出力して、即終了する為、「cat」コマンドのファイル出力が行き場を無くして、上記のような警告を出します。なお、実行結果としては、正常に終了します。

対策の一つとして、警告メッセージを見せなくする方法があります。

```
PRINT SHELL CALLOUT$("cat hoge.txt 2>/dev/null | head -n1")
```

これは、警告メッセージが標準エラー出力と呼ばれるデバイスに出力されるので、そのメッセージ自体を、一種のごみ箱(/dev/null というデバイス)に出力するように、指示するものです。これらの使い方と意味については、Linuxおよびシェルと呼ばれる機能を扱った書籍を参照ください。

2.10.3 SHELL , SHELL RUN

命令		
機 能	外部プログラムを呼び出します。	
書 式1	SHELL "<①外部プログラム引数>" [, <②タイムアウト秒数>]	
書 式2	SHELL RUN "<①外部プログラム引数>" [, <②タイムアウト秒数>] 書式1と同じ動作です。	
パラ メータ	①	<外部プログラム引数> 文字列
	呼び出す外部プログラムと引数のリストを指定します。 文字列で指定してください。	
パラ メータ	②	<タイムアウト秒数> 数値
	外部プログラムのタイムアウト時間を秒数で指定します。 指定しないと外部プログラムが終了するまで本命令は終了しません。 タイムアウト秒数を経過しても外部プログラムが終了しない場合は、エラーとなります。	
注 意	<ul style="list-style-type: none"> ・VFATでフォーマットされたUSBメモリにあるプログラムは実行できません。 ・呼び出すプログラムおよびLinuxコマンドから、フォルダや隠しフォルダ等を、不用意に削除しようとししないでください。システムが正常に起動しなくなる恐れがあります。 ・呼び出した処理が完了するまで制御は戻りません(IDEによる処理の停止や終了もできません)。 ・外部プログラムの終了を待たずに次に進めたい場合は、SHELL SYSTEM で文字列終端に「&」を付けてバックグラウンド実行させるか、SHELL SPAWN でプログラムを非同期に呼び出してください。 ・外部プログラムの出力結果文字列を受け取りたいだけの場合は、SHELL CALLOUT\$ が呼び出しで結果が得られるのでお奨めです。 	
使用例1	SHELL RUN "ls > files.txt"	
	入力値の ls > files.txt というコマンドを実行します。	
使用例2	SHELL "ls > files.txt"	
	入力値の ls > files.txt というコマンドを実行します。	
使用例3	SHELL "ls > files.txt" , 5	
	入力値の ls > files.txt というコマンドを実行しますが、5秒以内に終わらないとエラーとします。	

2.10.4 SHELL CODE

関数		
機 能	SHELL RUNで実行した、外部プログラムの戻り値を取得します。	
書 式	<(戻り値)実行結果値> = SHELL CODE	
戻り値	戻り値	<実行結果値> 数値
	外部プログラムの実行結果値が得られます。	
注 意	<ul style="list-style-type: none"> ・SHELL RUNコマンドを実行した、最後の結果を取得します。複数実行した場合や、同時に実行した場合は、最後に実行された実行結果が取得できます。 ・実行結果の取得は、スレッド毎に保持されます。 	
使用例	SHELL RUN "ls" RET = SHELL CODE ls というコマンドを実行し、実行結果を取得します。	

2.10.5 SHELL OUTPUT\$

関数			
機 能	SHELL RUNで実行した、外部プログラムの出力結果文字列を取得します。		
書 式	<(戻り値)出力結果文字列> = SHELL OUTPUT\$		
戻り値	戻り値	<出力結果文字列>	文字列
	外部プログラムを呼び出した結果の、標準出力の内容を文字列で取得します。		
注 意	<ul style="list-style-type: none">• SHELL RUNコマンドを実行した、最後の出力結果を取得します。複数実行した場合や、同時に実行した場合は、最後に実行された出力結果が取得できます。• 出力結果の取得は、スレッド毎に保持されます。		
使用例	SHELL RUN "ls" SMP\$ = SHELL OUTPUT\$ ls というコマンドを実行し、出力結果文字列を取得します。		

2.10.6 SHELL CALLOUT\$

関数			
機 能	外部プログラムをシェル経由で呼び出し、その出力結果文字列を取得します。		
書 式	<(戻り値)出力結果文字列> = SHELL CALLOUT\$(<①外部プログラム引数> [, <②実行結果値>])		
戻り値	戻り値	<出力結果文字列>	文字列
	外部プログラムをシェル経由で呼び出した結果の、標準出力の内容を取得して返します。		
パラ メータ	①	<外部プログラム引数>	文字列
	呼び出す外部プログラム名を文字列で指定します。		
	②	<実行結果値>	数値
ここで指定した変数に対して、外部プログラムを呼び出した戻り値を受け取る事ができます。			
注 意	・ 外部プログラムをシェルから呼び出せない時、失敗した時、エラーが発生します。		
備 考	・ 本コマンドは、C言語標準ライブラリのpopen関数を使用しています。		
使用例	S\$ = SHELL CALLOUT\$("ls - l") ls というコマンドを実行し、出力結果を取得します。		

2.10.7 SHELL OPEN

命令		
機 能	指定したファイルを、関連付けに応じてデフォルトアプリケーションで開きます。	
書 式	SHELL OPEN <①URIパス文字列>	
パラ メータ	①	<div> <div><URIパス文字列></div> <div>文字列</div> </div> 開きたいファイルのパス または URI形式の文字列を指定します。 例えば、「http://」形式のURIパス文字列を指定すると、デフォルトブラウザが起動し、指定したURIパス文字列で開こうとします。
注 意	<ul style="list-style-type: none"> 外部プログラムの呼び出しの失敗などの判定は行いません。 	
備 考	<ul style="list-style-type: none"> 本コマンドは、Linuxコマンドの <code>gio open</code> コマンドの動作に相当します。 	
使用例1	SHELL OPEN "test.odt" ' test.odt 'に 関 連 付 け ら れ た Writer アプリケーションが起動し、test.odtが開きます。	
使用例2	SHELL OPEN "http://www.interface.co.jp" ' http: 'に 関 連 付 け ら れ た ブラウザ (firefox など) が 起 動 し、 「http://www.interface.co.jp」のURIが開きます。	

2.10.8 SHELL SYSTEM

関数			
機 能	外部プログラムをシェル経由で呼び出し、その戻り値を得ます。		
書 式	<(戻り値) 実行結果値> = SHELL SYSTEM(<①外部プログラム引数>)		
戻り値	戻り値	<実行結果値>	数値
	外部プログラムを呼び出した戻り値を得ます。		
パラ メータ	①	<外部プログラム引数>	文字列
	呼び出す外部プログラム名を文字列で指定します。		
注 意	<ul style="list-style-type: none">外部プログラムが実行されて戻ってきた際の、実行結果はチェックされません。外部プログラムの呼び出しの失敗などの判定は行いません。		
備 考	<ul style="list-style-type: none">本コマンドは、C言語標準ライブラリのsystem関数を使用して /bin/sh を経由し、外部プログラムを呼び出しています。		
使用例	? SHELL SYSTEM("ip a") ' ip というコマンドを実行し、その戻り値が得られます。		

2.10.9 SHELL SPAWN

関数		
機 能	外部プログラムを非同期で呼び出し、そのプロセスID値を得ます。	
書 式	<(戻り値)プロセスID値> = SHELL SPAWN(<①外部プログラム引数>)	
戻り値	戻り値	<プロセスID値> 数値
	外部プログラムを非同期で呼び出し、そのプロセスID値を得ます。 得られたプロセスIDの実行結果値を得るには、SHELL WAITPID 関数を用います。	
パラ メータ	①	<外部プログラム引数> 文字列
	呼び出す外部プログラム名を文字列で指定します。	
注 意	<ul style="list-style-type: none"> 外部プログラムの呼び出しの失敗などの判定は行いません。 SHELL WAITPID 関数 を使って、外部プログラムの終了確認を行うようにしてください。終了確認しないと、外部プログラムが終了しても、Linux は ゾンビプロセスという状態で保持し続けます。 	
備 考	<ul style="list-style-type: none"> 本コマンドは、glibライブラリの g_spawn_async関数を使用しています。 	
使用例	<pre>PID = SHELL SPAWN("ip a") PRINT "PID="; PID PRINT "RET="; SHELL WAITPID(PID) ' 「ip a」という外部プログラムを起動し、プロセスIDを得ます。 ' その後、外部プログラムが終了するのを待機し、その実行結果を得ます。</pre>	

2. 10. 10 SHELL WAITPID

関数			
機 能	指定したプロセスIDの状態変化を待ち、その実行結果を得ます。		
書 式	<(戻り値)プロセスの実行結果値> = SHELL WAITPID(<①プロセスID値> [, <②非同期フラグ>])		
戻り値	戻り値	<プロセスの実行結果値>	数値
	<p>指定したプロセスIDが終了するまで待機し、その実行結果値を得ます。 指定したプロセスIDが存在しない場合、-1 が得られます。</p> <p>非同期フラグに1を指定した場合、プロセスIDが実行中は 0、終了して存在しない場合は -1 が得られます。</p>		
パラ メータ	①	<プロセスID値>	数値
	状態変化を監視したいプロセスID値を指定します。		
	②	<非同期フラグ>	数値
1を指定すると、プロセスIDの終了を待たず、現在の状態を得ます。 0を指定するか省略すると、プロセスIDの終了を待ち、その実行結果を得ます。			
備 考	・ 本コマンドは、C言語標準ライブラリの、waitpid関数を使用しています。		
使用例1	<pre>PID = SHELL SPAWN("ip a") PRINT "PID="; PID PRINT "RET="; SHELL WAITPID(PID)</pre> <p>’ 「ip a」という外部プログラムを起動し、プロセスIDを得ます。 ’ その後、外部プログラムが終了するのを待機し、その実行結果を得ます。</p>		
使用例2	<pre>PID = SHELL SPAWN("gedit") PRINT "PID="; PID DO WHILE TRUE IF SHELL WAITPID(PID, 1) = -1 THEN EXIT DO PRINT PID; "が実行中" SLEEP 1 LOOP</pre> <p>’ geditという外部プログラムを起動し、プロセスID値を得ます。 ’ その後、外部プログラムが終了するのを非同期で監視し続けます。</p>		

2. 10. 11 SHELL KILLPID

関数		
機 能	指定したプロセスIDに対してシグナルを送ります。	
書 式	<(戻り値)成功可否> = SHELL KILLPID(<①プロセスID値> [, <②シグナル番号>])	
戻り値	戻り値	<成功可否> 数値
	シグナル送信が成功したら 0、失敗したら -1 が得られます。	
パラ メータ	①	<プロセスID値> 数値
	シグナルを送信したい プロセスID値を指定します。	
	②	<シグナル番号> 数値
	プロセスに対して、送信したいシグナル番号を指定します。	
	省略すると、9(SIGKILLシグナル。強制終了を依頼する) を送信します。	
備 考	<ul style="list-style-type: none"> 本コマンドは、C言語標準ライブラリの、kill関数を使用しています。 シグナルについては、Linux端末上で「man 7 signal」あるいは「man kill」を実行した際の説明か、Linuxの関連書籍を参照ください。 	
使用例	<pre>PID = SHELL SPAWN("gedit") PRINT "PID="; PID PRINT "KILL?="; SHELL KILLPID(PID)</pre> <p>' geditという外部プログラムを起動し、プロセスIDを得ます。</p> <p>' その後、外部プログラムが強制終了するよう、シグナルを送ります。</p>	

第3章 サンプルプログラム

AJANのサンプルプログラムについて記載します。

サンプルプログラムは「/usr/share/interface/AJANPro/samples/BASE/」に格納されています。

AJAN統合開発環境を起動すると、左ペインのエクスプローラウィンドウ内の「Samples/BASE/」に、ファイルが取り込まれて配置されます。

3.1 サンプルプログラム

ファイル名	内容
AJAN メイン操作	
ASSERT. AJN	ASSERT 命令で、入力値と条件が一致しない場合に、エラーを表示します。
CLS. AJN	1秒ごとに、CLS 命令で画面をクリア後に1行ずつ文字列を表示します。
END. AJN	1秒ごとに、文字列を切り替えて表示しますが、途中 END 命令で終了します。
ERL_ERR_ERM. AJN	プログラムの途中でエラーが発生すると、ON ERROR CALL 命令で定義した サブルーチン(ERR_SUB)にジャンプし、エラー発生行、エラーコード、エラーメッセージを表示して終了します。
ERROR_SETERRTBL. AJN	SETERRTBL で独自に定義したエラーコードを登録し、ユーザー定義した関数(TEST_DIV)で、引数のチェックを行って、不正な引数値であれば、ERROR 命令で 任意のエラーを発生します。
REM. AJN	1秒ごとに文字列を切り替えて表示しますが、一部はREM 命令と ' (コメント) 記号により、コメントとして扱われる為、実行されない事を確認します。
SHELL. AJN	SHELL RUN 関数を使って、Linux の ls コマンドを呼び出し、SHELL CODE 関数で戻り値と、SHELL OUTPUT\$ 関数で、ファイル一覧の文字列情報を取得して表示します。
SHELL_CALLOUT_SYSTEM. AJN	SHELL CALLOUT\$ 関数と SHELL SYSTEM 関数を使って、Linux の zenity コマンドを使った、UI 的な対話の事例を示します。
SHELL_SPAWN_WAITPID. AJN	SHELL SPAWN 関数を使って、Linux の xeyes アプリを起動します。 xeyes アプリの pid 値 を使って、SHELL WAITPID 関数で終了待ちします。
STOP_CONT. AJN	文字列を順番に表示しますが、途中 STOP 命令でプログラムを一時停止させます。 AJAN 統合開発環境のメニューから「再開」を選択すると続きを実行します。
STOP_CONTRACE. AJN	文字列を順番に表示しますが、途中 STOP 命令でプログラムを一時停止させます。 AJAN 統合開発環境のメニューから「実行(R)」→「自動トレース(A)」を選択すると続きをトレース実行します。
数値・文字列	
ABS. AJN	入力した数値を、ABS 関数で絶対値を求めて表示します。
ASC. AJN	入力した文字列に対して、1文字ずつ ASC 関数でキャラクターコードを取り出し、英小文字を大文字に変換して表示します。

ファイル名	内容
CLOCK. AJN	CLOCK 関数で作業の開始時間と終了時間を記録し、その経過時間の差を求めてかかった時間を表示します。
TORIGONOMETRIC. AJN	入力した数値(ラジアン値)の、正弦(SIN 関数)、余弦(COS 関数)、正接(TAN 関数)、逆正接(ATN 関数)を表示します。
STR. AJN	0～16までの数値を、変換関数を使って、10進(STR\$関数)、2進(BIN\$関数)、16進(HEX\$関数)表記で、表示します。
CHR. AJN	CHR\$関数で、キャラクタコードを文字に変換して表示します。
CHRB_ASCB. AJN	0～255までの値を格納したバイナリデータを、CHRB\$関数で 1 バイトずつ文字列に変換します。 その後、文字列を MIDB\$関数で1バイトずつ取り出して、ASCB 関数でバイナリデータに変換します。
CSV2ARRAY. AJN	CSV2ARRAY\$関数を使い、CSV 形式の文字列を読み取って、配列に変換&表示します。
DATE. AJN	今日の日付を DATE\$関数で求めて、WEEK 関数で曜日表示します。
DAYS. AJN	入力した日付と今日の日付を日時時刻型に格納し、入力日付と今日との日数の差を表示します。
MATH. AJN	入力した数値に対して、様々な数学演算をします。 EXP 関数で指数関数の値、LOG 関数で対数の値、SQR 関数で平方根の値を求めます。
MEMINFO. AJN	MEMINFO 関数で、システムのメモリ情報を取得し、表示します。
FIX_INT_ROUND. AJN	入力された数値から、FIX 関数で 整数部分、INT 関数で 元の値を超えない最大整数値、ROUND 関数で 四捨五入した値を表示します。
STR_CHANGE. AJN	文字列に対して、各種文字列操作関数(INSTR, LINSTR, LEN, LENB, MID\$, REPLACE\$, STRDEL\$, STRINS\$, TRIM\$)を使って、様々な操作をします。
RND. AJN	RANDOMIZE 命令で乱数系列を設定した後、RND 関数で乱数値を取得&表示します。
SPLIT. AJN	SPLIT\$ 関数を使い、文字列をカンマやスペースで分割して文字列配列に代入します。
TIME. AJN	TIME\$関数で、今の時刻、指定秒数の時刻、現時刻からの前後時刻を表示します。
MID. AJN	MID\$関数で、入力文字列を2文字ずつに分割して表示します。
VAL. AJN	VAL 関数で、文字列を数値に変換して演算します。
WEEK. AJN	日付時刻値に対して、WEEK 関数を使って曜日を求めます。
型の宣言や変換	
BOOL. AJN	BOOL 命令で宣言した、変数の真偽をそれぞれ IF 文で評価し、処理を分岐させます。
DATETIME. AJN	日付時刻型を使って、明日の日付や、1時間前の時刻を計算して表示します。
STRUCT. AJN	ID とアドレスを記録する構造体(DEFINE STRUCT～END STRUCT 命令)の配列変数を宣言し、値を代入した後、それぞれのメンバ値を表示します。
VARTYPE. AJN	VARTYPE 関数で、様々な変数の型情報を得ます。
配列	
DIM. AJN	DIM 命令で配列を宣言し、値の格納と取り出し、要素数の取得(LDIM)などの処理を行います。
DICT. AJN	DICT 命令で宣言した連想配列にデータを格納し、キーの数(LDICT)、キーの一覧(GET_DICT_KEYS\$)、キーの有無(HAS_DICT_KEY)、指定したキーの値を表示します。

ファイル名	内容
LIST. AJN	LIST 命令で STRING_DATA\$ 変数を宣言し、入力した文字列を「,」で区切って、可変長の文字列配列として代入します。 その後、配列の内容を 1 つずつ表示します。
ONEDIM_INSERT. AJN	ONEDIM INSERT 命令で、1次元配列の指定した位置に、変数を挿入します。
ONEDIM_REMOVE. AJN	ONEDIM REMOVE 命令で、1次元配列の指定した位置から、3つの要素を削除します。
TWODIM_INSERT. AJN	TWODIM INSERT 命令で、2次元配列の指定した行列の位置に、1行分の配列変数を挿入します。
TWODIM_REMOVE. AJN	TWODIM REMOVE 命令で、2次元配列の指定した行列の位置から、1行または1列分削除します。
繰り返し・条件分岐	
DO_WHILE_LOOP. AJN	DO WHILE〜LOOP で、1〜5 までの数をループして表示します。
EXIT_DO. AJN	DO WHILE〜LOOP で 1〜5までループします。 途中、3より大きくなったら、EXIT DO で ループを中断します。
EXIT_FOR. AJN	FOR〜NEXT で 1〜7 までループします。 5を超えたら、EXIT FOR で ループを抜けます。
FOR_NEXT. AJN	FOR〜NEXT で、数値を様々な演算をしながら繰り返しループ処理します。
IF. AJN	暗証番号(BANGOU)を定義しておき、入力値と一致するか否か、IF 命令で判定し分岐します。
MENU. AJN	ENUM〜END ENUM で列挙型を定義し、SELECT CASE 命令を利用して、メニュー選択機能を作ります
ファイル・フォルダ	
INCLUDE. AJN	INCLUDE 命令で、外部ファイルの MY_LIBRARY. AJN を参照&追加します。
INPUT. AJN	INPUT 命令で、文字列や数値を入力して、変数に代入します。
OPEN. AJN	日付と時刻をファイルに書き出し、同じファイルを読み込んで表示します。 OPEN 命令でファイルを開いて、PRINT 命令で日付と時刻を書き込みます。 CLOSE 命令でファイルを閉じて、その後、同じファイルを開いて LINE INPUT 命令で読み取って表示します。
PRINT. AJN	PRINT 命令で、文字列や変数の値を表示します。
サブルーチン	
GOTO. AJN	入力値と条件が合わなければ、指定したラベルまで GOTO 命令でジャンプします。
CALL. AJN	三角形の面積を求めるサブルーチン(SUB〜END SUB)を定義し、CALL 命令で呼び出して、結果を表示します。
FUNCTION. AJN	三角形の面積を求めるユーザー定義関数(FUNCTION〜END FUNCTION)を定義し、ユーザー定義関数を呼び出して、結果を表示します。
EXIT_FUNCTION. AJN	三角形の面積を求めるユーザー定義関数を定義します。 引数の底辺、高さの値が負数の場合、EXIT FUNCTION 命令で、ユーザー定義関数を途中で抜けます。
EXIT_SUB. AJN	三角形の面積を求めるサブルーチンを定義します。 底辺、高さの値が負数の場合、EXIT SUB 命令で、サブルーチンを途中で抜けます。
ON_KEY_CALL. AJN	F1〜F4キーを押すと、ON KEY CALL 命令で定義したサブルーチンにジャンプし、様々な処理を行います。

ファイル名	内容
ON_TIME_CALL. AJN	時間が00秒になるまで、現在の時刻を表示し続けます。 指定した時間になると、ON TIME\$ CALL 命令で定義したサブルーチンにジャンプし、フラグ(Flag)を立てて終了します。
ON_TIMER_CALL. AJN	プログラム開始から5秒間、現在の時刻を表示し続けます。 1秒毎に、ON TIMER CALL 命令で定義したサブルーチンにジャンプし、5回呼び出されると、フラグ(FLAG)を立てて終了します。
ON_TIMER_CALL_2. AJN	500ms および 2秒の時間間隔に、複数のタイマ割り込みを行う事例です。 ON TIMER CALL 命令で、500ms 毎 と 2秒毎に呼び出される、2つのタイマーを設定します。 7秒ほど経過したら、プログラムを終了します。
ON_ERROR_CALL. AJN	エラーが発生したら、ON ERROR CALL 命令で定義したサブルーチン(OnErr)にジャンプし、エラーコード、エラーメッセージ、エラー発生行番号を表示します。
スレッド	
THRAED. AJN	DEFINE THREAD〜END THREAD 命令でスレッドルーチンを定義し、ATTACH THREAD 命令で、複数のスレッドを作成し、それぞれが別の変数に対して並列に演算します。 演算後、DETACH THREAD 命令で、スレッドを終了します。

第4章 エラーコードリファレンス

AJAN使用時のエラーコードリファレンスについて記載します。

4.1 エラーコードリファレンス

値(16進数 / 10進数)	表示エラー	対策・確認
&h00000000 / 0	正常状態です	
&h01000001 / 16777217	内部で異常が発生しました	内部で致命的な異常、または想定外の異常が発生した事を意味します。 ON ERROR CALL による回復処理は期待できません。
&h01000002 / 16777218	命令が書式通りになっていません	文法や構文に間違いが無いか、引数の数がマニュアル通りか確認ください。
&h01000003 / 16777219	使用する変数に問題があります	入力パラメータの期待する型が異なっていないか確認ください。
&h01000004 / 16777220	値が許容範囲外です	入力パラメータの値が期待する範囲外でないか確認ください。
&h01000005 / 16777221	演算結果が許容範囲を超えました	計算結果がオーバーフローなど許容範囲外になった可能性があります。
&h01000006 / 16777222	配列の操作に問題があります	たとえば、引数に1次元配列を渡すのに、2次元配列や配列でない値を渡した。などの可能性があります。
&h01000007 / 16777223	デバイスに関する問題が発生しました	I/Oデバイスをオープンせずに使用したり、使用方法や順番に問題がある可能性があります。
&h01000008 / 16777224	ファイルのオープンに失敗しました	ファイル名の間違いにより、既存ファイルのオープンに失敗した。または、作成しようとするファイルの場所が、アクセス権不足により作成できなかった。などの可能性があります。
&h01000009 / 16777225	メモリの確保に失敗しました	システムがメモリ不足により、要求されたメモリが確保できませんでした。
&h0100000A / 16777226	0による除算が実行されました	0で数値を割ろうとしました。
&h0100000B / 16777227	読むべきデータがありません	これ以上、取得すべきデータが無い。データが格納されてない可能性があります。
&h0100000C / 16777228	本機能はサポートされていません	将来的に対応する予定だが現在は未対応の機能と呼び出そうとした。 または、引数に対して期待する型以外の型の変数を渡してしまった。 コマンドの中から、外部ライブラリを呼び出そうとしたが、要求するAPIが見つからない。 などがあります。
&h0100000D / 16777229	そのエリアでは使用できません	スコープ外の変数を参照しようとしている。 サブルーチンを定義中に、さらに別のルーチンを定義しようとした。などがあります。
&h0100000E / 16777230	ASSERT 確認で異常を検知しました	ASSERT命令の条件を満たさなかった。
&h0100000F / 16777231	指定動作が動作中です	オープン中に再度オープンしようとした。 ファイル番号などの指定ID番号がオープン済み。 などがあります。

値(16進数 / 10進数)	表示エラー	対策・確認
&h01000010 / 16777232	指定動作が停止中です	ファイル番号などの指定ID番号でオープンしてないのに使用しようとした。 ファイルが既にクローズされている。 などがあります。
&h01000011 / 16777233	強制停止が指示されました	OSまたはIDEから、強制停止が指示されました。
&h01000012 / 16777234	ドライバまたはライブラリの呼び出しに失敗しました	コマンドの中からドライバや外部ライブラリ等の呼び出しが失敗しました。 詳細はERRSUB関数やERRMSG\$関数で得られます。
&h01000013 / 16777235	入力 / 出力エラーです	ファイルやネットワークの読み書き または 送受信時にエラーを検知しました。
&h01000014 / 16777236	不正な記述子です	ファイル番号や指示ID値に、範囲外の値を与えた。またはオープンされてない値を与えた。などがあります。
&h01000015 / 16777237	タイムアウトが発生しました	ネットワーク等で、タイムアウトが発生しました。
&h01000016 / 16777238	そのようなファイルまたはディレクトリはありません	存在しないパスを指定したり、パスの指定が不足している。などがあります。
&h01000017 / 16777239	無効な引数が指定されました	引数の記述が期待通りでない。引数が足りないまたは多すぎる。などがあります。
&h01000018 / 16777240	配列の添字が範囲外です	たとえば、10の要素数の配列に対して、範囲外の20を参照しようとした。などがあります。
&h01000019 / 16777241	配列内にキーが見つかりません	たとえば、連想配列に登録外のキーを与えて参照しようとした。などがあります。
&h0100001A / 16777242	正しくない UTF-8 文字列です	文字列がUTF-8文字列として正しくない為、文字列としての操作ができません。 (MIDB\$関数などバイト単位で操作する関数の場合、UTF-8文字列である必要はないので使用できます)
&h0100001B / 16777243	バッファが不足または大きすぎます	指定したバッファ量が適切になるよう調整してください。
&h0100001C / 16777244	値が許容範囲を下回りました	計算結果が、アンダーフローしたなど許容範囲外になった可能性があります。
&h0100001D / 16777245	リンク接続に失敗しました	IPアドレスの不正などにより、サーバに接続できなかった可能性があります。
&h0100001E / 16777246	指定された型への変換が出来ません	文字列から数値への変換、文字列から日付時刻型への変換が、書式間違いなどにより、できませんでした。
&h0100001F / 16777247	指定した名前が見つかりません	指定した変数名、関数名、構造体のメンバが見つからなかった可能性があります。
&h01000020 / 16777248	限界値に達しました	呼び出し回数が限界に達した。オープンできる最大数に達した。などがあります。
&h01000021 / 16777249	未定義の構造体を呼び出そうとしました	DEFINE STRUCT命令による構造体定義を行ってない。などがあります。
&h01000022 / 16777250	未定義のサブルーチンまたは関数名を呼び出そうとしました	たとえば、ON TIMER CALL命令で、割り込み先のサブルーチンを定義してない。などがあります。
&h01000023 / 16777251	サブルーチンまたは関数名が重複しています	既にサブルーチンまたは関数名が定義済み。などがあります。
&h01000024 / 16777252	同じ名前の変数が存在しています	既に変数名が定義済み。などがあります。
&h01000025 / 16777253	実行時権限が不足しています	使用するコマンドを実行する為に、管理者権限(スーパーユーザー)が必要です。 例えば、参照先のファイルが、rootでしか参照できない。とか。

第5章 アスキーコード一覧

AJANでのアスキーコードは下記になります。

コード	文字	コード	文字	コード	文字
9	タブ	63	?	96	`
10	LF(改行)	64	@	97	a
32		65	A	98	b
33	!	66	B	99	c
34	"	67	C	100	d
35	#	68	D	101	e
36	\$	69	E	102	f
37	%	70	F	103	g
38	&	71	G	104	h
39	'	72	H	105	i
40	(73	I	106	j
41)	74	J	107	k
42	*	75	K	108	l
43	+	76	L	109	m
44	,	77	M	110	n
45	-	78	N	111	o
46	.	79	O	112	p
47	/	80	P	113	q
48	0	81	Q	114	r
49	1	82	R	115	s
50	2	83	S	116	t
51	3	84	T	117	u
52	4	85	U	118	v
53	5	86	V	119	w
54	6	87	W	120	x
55	7	88	X	121	y
56	8	89	Y	122	z
57	9	90	Z	123	{
58	:	91	[124	
59	;	92	\	125	}
60	<	93]	126	~
61	=	94	^		
62	>	95	_		

第6章 コマンドの制限事項

6.1 トレース実行で使用できないコマンド

トレース実行では、イベント処理はできません。

トレース実行で使用できないコマンド		
ON ERROR CALL	ON KEY CALL	

6.2 デバッグ実行中に停止できないコマンド

以下は、コマンド実行終了までIDEの制御(停止や終了など)ができません。

デバッグ実行中に停止できないコマンド		
SHELL (SHELL RUN)	LINE INPUT (キーボード入力)	

第7章 索引

:		DEFINE THREAD ~ END THREAD	172
:	149	DEL_DICT_KEY	89
A		DETACH THREAD	170
ABS	26	DICT	90
ARRAY2CSV\$	27	DICT BOOL	91
ARRAY2DICT	87	DICT DATETIME	91
ASC	28	DICT STRUCT	92
ASCB	28	DICT2ARRAY\$	93
ASSERT	14	DIM	94
ATN	29	DIREXISTS	126
ATTACH THREAD	169	DO WHILE~LOOP	118
B		<i>E</i>	
BIN\$	29	END	15
BOOL	71	ENUM~END ENUM	77
C		EOF	127
CALL	149	ERF\$	15
CDATETIME	30	ERL	16
CDIM	88	ERM\$	16
CHDIR	125	ERN\$	17
CHR\$	31	ERR	17
CHRB\$	31	ERRMSG\$	18
CLEAR_DICT	88	ERROR	19
CLOCK	32	ERROR ON / OFF	150
CLOSE	125	ERRSUB	18
CLS	14	EXIT DO	119
CONST	72	EXIT FOR	119
CONST BOOL	72	EXIT FUNCTION	151
CONST DATETIME	73	EXIT SUB	151
COS	32	EXP	39
CSV2ARRAY\$	33	<i>F</i>	
CVD	35	FILECOPY	128
CVH	36	FILEEXISTS	129
CVI	37	FILELIST\$	129
CVL	38	FILESTAT	130
CVS	38	FIX	40
D		FOR~TO~STEP~NEXT	120
DATE\$	39	FREEFILE	131
DATETIME	74	FUNCTION~END FUNCTION	152
DEFINE STRUCT~END STRUCT	75	<i>G</i>	
		GET_DICT_KEYS\$	96

GOTO	154	LOCAL ENUM~END ENUM	83
<i>H</i>		LOCAL LIST	107
		LOCAL LIST BOOL	108
HAS_DICT_KEY	96	LOCAL LIST DATETIME	108
HEX\$	40	LOCAL LIST DICT	108
<i>I</i>		LOCAL LIST DICT BOOL	109
IF~THEN~ELSE~END IF	121	LOCAL LIST DICT DATETIME	109
INCLUDE	132	LOCAL LIST DICT STRUCT	110
INPUT	133	LOCAL LIST STRUCT	110
INSTR	41	LOCAL STRUCT	84
INSTRREV	42	LOCK	173
INT	43	LOG	49
ISDATETIME	44	LSTRIP\$	49
ISINF	46	<i>M</i>	
ISNAN	46	MEMINFO	50
ISNUMERIC	45	MID\$	51
<i>K</i>		MIDB\$	52
KEY ON / OFF / STOP	155	MKD\$	53
KILL	135	MKDIR	137
<i>L</i>		MKH\$	54
LDICT	98	MKI\$	55
LDIM	98	MKL\$	56
LEN	47	MKS\$	56
LENB	47	<i>N</i>	
LINE INPUT	136	NAME	137
LINSTR	48	<i>O</i>	
LIST	99	ON END CALL	156
LIST BOOL	100	ON ERROR CALL	157
LIST DATETIME	100	ON KEY CALL	158
LIST DICT	101	ON TIME\$ CALL	159
LIST DICT BOOL	102	ON TIMER CALL	160
LIST DICT DATETIME	103	ONEDIM INSERT	111
LIST DICT STRUCT	104	ONEDIM REMOVE	112
LIST STRUCT	105	OPEN	138
LOCAL	78	OPTION ERROR	21
LOCAL BOOL	79	OPTION STOP	22
LOCAL CONST	80	<i>P</i>	
LOCAL CONST BOOL	81		
LOCAL CONST DATETIME	82	PATH GET ABSPATH\$	139
LOCAL DATETIME	82	PATH GET BASENAME\$	139
LOCAL DICT	106	PATH GET DIRNAME\$	140
LOCAL DICT BOOL	106	PATH JOIN\$	140
LOCAL DICT DATETIME	107	PATH SPLITEXT\$	141
LOCAL DICT STRUCT	107	PRINT, ?	142

<i>R</i>		STR_FREADALL\$	145
RANDOMIZE	57	STR_FWRITEALL	145
REDIM	113	STRDEL\$	64
REM, '	23	STRINS\$	65
REPLACE\$	57	STRIP\$	65
RMDIR	143	STRUCT	85
RND	58	SUB~END SUB	162
ROUND	59	<i>T</i>	
RSTRIP\$	60	TAN	66
<i>S</i>		THREAD INFO	174
SEEKGET	144	THREAD STATUS	174
SEEKSET	144	TIME\$	67
SELECT CASE~END SELECT	122	TIME\$ ON / OFF / STOP	164
SETERRTBL	24	TIMER ON / OFF / STOP	165
SHELL , SHELL RUN	179	TRIM\$	69
SHELL CALLOUT\$	182	TWODIM INSERT	114
SHELL CODE	179	TWODIM REMOVE	116
SHELL KILLPID	187	<i>U</i>	
SHELL OPEN	183	UBOUND	117
SHELL OUTPUT\$	181	UNLOCK	175
SHELL SPAWN	185	<i>V</i>	
SHELL SYSTEM	184	VAL	69
SHELL WAITPID	186	VARTYPE	86
SIN	61	<i>W</i>	
SLEEP	25	WAIT FILE	146
SPLIT\$	62	WEEK	70
SQR	63		
STOP	25		
STR\$	63		

第8章 重要な情報

保証の内容と制限

弊社は本ドキュメントに含まれるソースプログラムの実行が中断しないこと、またはその実行に誤りが無いことを保証していません。

本製品の品質や使用に起因する、性能に起因するいかなるリスクも使用者が負うものとします。

弊社はドキュメント内の情報の正確さに万全を期しています。万一、誤記または誤植などがあつた場合、弊社は予告無く改訂する場合があります。ドキュメントまたはドキュメント内の情報に起因するいかなる損害に対しても弊社は責任を負いません。

ドキュメント内の図や表は説明のためであり、ユーザ個別の応用事例により変化する場合があります。

著作権、知的所有権

弊社は本製品に含まれるおよび本製品に対する権利や知的所有権を保持しています。

本製品はコンピュータ ソフトウェア、映像/音声(例えば図、文章、写真など)を含んでいます。

医療機器/器具への適用における注意

弊社の製品は人命に関わるような状況下で使用される機器に用いられる事を目的として設計、製造された物では有りません。

弊社の製品は人体の検査などに使用するに適する信頼性を確保する事を意図された部品や検査機器と共に設計された物では有りません。

医療機器、治療器具などの本製品の適用により、製品の故障、ユーザ、設計者の過失などにより、損傷/損害を引き起こす場合があります。

複製の禁止

弊社の許可なく、本ドキュメントの全て、または一部に関わらず、複製、改変などを行うことはできません。

責任の制限

弊社は、弊社または再販売者の予見の有無にかかわらず発生したいかなる特別損害、偶発的損害、間接的な損害、重大な損害について、責任を負いません。

本製品(ハードウェア、ソフトウェア)のシステム組み込み、使用、ならびに本製品から得られる結果に関する一切のリスクについては、本製品の使用者に帰属するものとします。

本製品に含まれる不都合、あるいは本製品の供給(納期遅延)、性能もしくは使用に起因する付帯的損害もしくは間接的損害に対して、弊社に全面的に責がある場合でも、弊社はその製品に対する改良(有償サービスの利用)、代品交換までとし、製品の予防交換並びに、代金減額等、金銭面での賠償の責任は負わないものとします。

本製品は、日本国内仕様です。

商標/登録商標

本書に掲載されている会社名、製品名は、それぞれ各社の商標または登録商標です。

改訂履歴

Ver.	年 月	改 訂 内 容
0.90	2019年10月	新規作成
1.00	2022年1月	誤記修正。 コマンド追加。PATH SPLITTEXT\$, DIREXISTS, KILL, CHDIR, MKDIR, RMDIR, NAME, FILECOPY, FILESTAT, SEEKGET, SEEKSET
1.20	2023年3月	オプション追加。ASC, ASCB, CDATETIME, REPLACES\$ コマンド追加。ISNUMERIC, DICT2ARRAY\$, ARRAY2DICT

このマニュアルは、製品の改良その他により将来予告なく改訂しますので、予めご了承ください。