



Cipher Basic for 8 Series Mobile Terminal リファレンスマニュアル Part-I

英文・和文で相違がある場合は、英文を優先して解釈をお願いします

[illegible]

1. 本書の内容に関しては、将来予告無しに変更することがあります。
2. 本取扱説明書の全部又は一部を無断で複製することはできません。
3. 本書内に記載されている製品名等の固有名詞は各社の商標又は登録商標です。
4. 本書内において、万一誤り、記載漏れなどお気付きのことがありましたらご連絡ください。
5. 運用した結果の影響について、責任を一切負いかねます。

INDEX

1. はじめに	7
2. 開発環境	8
2.1. 構成ファイルとインストール	8
2.2. BASICランタイムインジック	9
2.3. 開発ツール	10
2.3.1. BASICランタイムインジックのダウンロード	10
2.3.2. BASICプログラムのコンパイル	11
2.3.3. BASICオブジェクトファイルのダウンロード	11
3. BASICコマンドを使ってみよう	13
3.1. File Menu (ファイルメニュー)	14
3.2. Edit Menu (編集メニュー)	15
3.3. Configure Menu (コンフィグレーションメニュー)	16
3.4. Compile Menu (コンパイルメニュー)	18
3.5. Help Menu (ヘルプメニュー)	19
4. CipherLab BASIC言語の基礎	20
4.1. 定数	20
4.1.1. 文字列型	20
4.1.2. 数値型	20
4.2. 変数	20
4.2.1. 変数名と修飾子	20
ELSE	21
END IF	21
4.2.2. 配列変数	22
4.3. 演算子	23
4.3.1. 代入演算子	23
4.3.2. 算術演算子	23
4.3.3. 比較演算子	23
4.3.4. 論理演算子	24
4.3.5. 演算子の優先順位	24
4.4. ラベル	25
4.5. サブルーチン	26
5. BASICリファレンス	27
5.1. 一般的なコマンド	28
ABS	28
BIT_OPERATOR	28
DIM	28
GOSUB	28
GOTO	29
INT	29
REM	29
SET_PRECISION	29
SGN	30
5.2. 条件処理コマンド	31
IF ... THEN ... [ELSE] END IF	31
IF ... THEN ... {ELSE IF ...} [ELSE] END IF	31
ON ... GOSUB ...	32
ON ... GOTO ...	32
5.3. ループ処理コマンド	33
EXIT	33
FOR ... NEXT	33
WHILE ... WEND	34
5.4. 文字列処理コマンド	35
5.4.1. 文字列の結合	35
5.4.2. 文字列の比較	35
5.4.3. 文字列の長さ	35
LEN	35
5.4.4. 文字列の検索	36
INSTR	36

5.4.5. 文字列の抽出・編集	36
LEFT\$	36
MID\$	37
RIGHT\$	37
TRIM_LEFT\$	37
TRIM_RIGHT\$	38
5.4.6. 文字列の変換	38
ASC	38
CHR\$	38
HEX\$	38
LCASE\$	39
OCT\$	39
STR\$	39
UCASE\$	39
VAL	40
VALR	40
5.4.7. 文字列の生成	40
STRING\$	40
5.5. イベント処理コマンド	41
5.5.1. 割り込みイベント	41
OFF ALL	41
OFF COM	42
OFF ESC	42
OFF HOUR_SHARP	42
OFF KEY	43
OFF KEY	43
OFF MINUTE_SHARP	44
OFF READER	44
OFF TCPIP	44
OFF TIMER	45
OFF TOUCH_SCREEN	45
ON COM ... GOSUB ...	45
ON ESC GOSUB ...	46
ON HOUR_SHARP GOSUB ...	46
ON KEY ... GOSUB ...	46
ON KEY ... GOSUB ...	47
ON MINUTE_SHARP GOSUB ...	47
ON POWER_ON GOSUB ...	48
ON READER GOSUB ...	48
ON TCPIP GOSUB ...	49
ON TIMER ... GOSUB ...	49
ON TOUCHSCREEN ... GOSUB ...	49
5.5.2. 割り込みイベントロックとアンロック	50
LOCK	50
UNLOCK	50
5.6. システムコマンド	51
AUTO_OFF	51
CHANGE_SPEED	51
IOPIN_STATUS	52
MENU	53
POWER_ON	54
RESTART	54
DEVICE_ID\$	54
GET_TARGET_MACHINE\$	54
SYSTEM_INFORMATION\$	55
VERSION	57
SYSTEM_PASSWORD	57
DOWNLOAD_BASIC	58
UPDATE_BASIC	59
5.7. パーフォリタ	61
DISABLE READER	61
ENABLE READER	61
GET_READER_DATA\$	62
READER_CONFIG	62
CODE_TYPE	63
GET_READER_SETTING	65
READER_SETTING	65

5.8. RFIDリーダ /ライタ	66
仮想COMポート	66
パラメータとデフォルト	67
SET_RFID_READ	67
ENABLE_READER	67
GET_RFID_KEY	68
SET_RFID_KEY	68
5.9. キーボードインターフェイス	69
SEND_WEDGE	69
SET_WEDGE	69
SEND_READY	71
5.10. ブザーコマンド	72
BEEP	72
STOP_BEEP	73
5.11. LEDコマンド	74
LED	74
5.12. バイブレータコマンド	75
VIBRATOR	75
5.13. リアルタイムクロックコマンド	76
DATE\$	76
DAY_OF_WEEK	76
TIME\$	77
TIMER	77
WAIT	78
5.14. バッテリーコマンド	79
BACKUP_BATTERY	79
MAIN_BATTERY	79
5.15. キーボードコマンド	80
CLR_KBD	80
INKEY\$	80
INPUT	80
INPUT_MODE	81
KEY_CLICK	81
PUTKEY	82
ALPHA_LOCK	82
GET_ALPHA_LOCK	83
GET_ALPHA_STATE	83
FUNCTION_TOGGLE	84
5.16. ディスプレイコマンド	85
BACK_LIGHT_DURATION	85
BACKLIT	85
LCD_CONTRAST	86
SET_VIDEO_MODE	86
CURSOR	86
CURSOR_X	87
CURSOR_Y	87
LOCATE	87
FILL_RECT	88
ICON_ZONE_PRINT	88
PRINT	89
WAIT_HOURLGLASS	89
CLR_RECT	89
CLS	90
GET_IMAGE	90
SHOW_IMAGE	91
CIRCLE	92
LINE	93
PUT_PIXEL	93
RECTANGLE	94
5.17. タッチスクリーンコマンド	95
DISABLE_TOUCHSCREEN	95
ENABLE_TOUCHSCREEN	95
GET_SCREENITEM	95
SET_SCREENITEMS	96
SET_SIGNAREA	96

5.17. フォントコマンド	97
GET_LANGUAGE	97
GET_LANGUAGE	98
SELECT_FONT	99
5.18. メモリコマンド	100
MEMORY_INFORMATION	101
FLASH_READ\$	101
FLASH_WRITE	102
ROM_SIZE	102
FREE_MEMORY	103
RAM_SIZE	103
SD_FREE_MEMORY	103
SD_SIZE	103
5.19. ファイル操作コマンド	104
5.19.1. トランザクションファイル (DATファイル)	104
DEL_TRANSACTION_DATA	104
DEL_TRANSACTION_DATA_EX	105
EMPTY_TRANSACTION	105
EMPTY_TRANSACTION_EX	106
GET_TRANSACTION_DATA\$	106
GET_TRANSACTION_DATA_EX\$	107
SAVE_TRANSACTION	107
SAVE_TRANSACTION_EX	108
TRANSACTION_COUNT	108
TRANSACTION_COUNT_EX	108
UPDATE_TRANSACTION	109
UPDATE_TRANSACTION_EX	109
5.19.2. DBFファイルとIDXファイル	110
ADD_RECORD	110
DEL_RECORD	111
EMPTY_FILE	112
FIND_RECORD	112
GET_RECORD	113
GET_RECORD_NUMBER	113
MOVE_TO	114
MOVE_TO_NEXT	114
MOVE_TO_PREVIOUS	114
RECORD_COUNT	114
UPDATE_RECORD	115
5.19.3. ファイルエラーコード	116
GET_FILE_ERROR	116
5.20. SDカード	117
トランザクションファイル (DATファイル)	117
DBFファイルとIDXファイル	117
5.20.1. ファイルシステム	117
5.20.2. ディレクトリ	118
5.20.3. ファイル名	119
補足 1. スキャ配列	120
スキャ配列表 1	120
スキャ配列表 2	125
補足 2. ホストコマンド	131
CLEAR	131
READ	132
REMOVE	133
TR	134
TW	135
補足 3. デバグコマンド	136
START_DEBUG	136
STOP_DEBUG	136
デバグメッセージ	137
デバグメッセージ例	142
補足 4. ランタイムエラー	143
補足 5. キーボードテーブル	144

1. はじめに

Cipher Basic は、CipherLab 独自の OS (Cipher OS) を搭載した 8 シリーズの 16 ビットマシン (以下、ターミナル) 用ユーザ-アプリケーションを安価で容易に開発することができる Windows ベースの開発プラットフォームです。COM ポート設定、データベース設定、バーコードリーダー設定など、簡単にメニュー形式で行えるため、コーディングを最低限に抑えることができます。

Cipher Basic は、1997 年にリリースされ、現在に至るまで進化し続けています。より早く、容易に、そして安価にアプリケーション開発が行える Cipher Basic の世界をお楽しみください。

尚、本書は、BASIC 言語によるプログラミングを学ぶための入門書ではありません。BASIC プログラミング経験をお持ちの方を対象に書かれたリファレンスマニュアルです。BASIC プログラミングの知識をお持ちでない方は、適切な入門書をお読みください。

2. 開発環境

Cipher Basic をインストールする前に、ご使用になる PC 環境をチェックしてください。

環境条件

- CPU Windows XP 以上が動作する CPU
- OS Windows XP/Vista/7
- RAM 128MB
- HDD 20MB 以上の空き領域

2.1. 構成ファイルとインストール

Cipher Basic の CD は、幾つかのフォルダで構成され、それぞれに BASIC コンパイル、ダウンロードユーティリティ、BASIC ランタイム、フォントファイルが収録されています。Cipher Basic のインストールは非常にシンプルで、ご使用になる PC に任意の作業フォルダを準備し、必要な以下のファイルをコピーすれば完了です。

BASIC Compiler

bc.exe BASIC コンパイル本体です。
release.txt 改訂履歴です。
samples BASIC サンプルプログラムです。プログラム本体(.bas)と初期化ファイル(.ini)がセットになります。

Download Utility

progload.exe ターミナルにプログラムをダウンロードするためのユーティリティプログラムです。
RS232C/IrDA・ケーブル IR・TCP/IP 経由で下記のプログラムファイルがダウンロード可能です。

- ✓ モトローラフォーマットオブジェクトファイル(.shx)
- ✓ BASIC オブジェクトファイル(.syn & .ini)

BASIC Runtimes

bc8000.shx 8000/8001 シリーズターミナル用 BASIC ランタイムです。
bc8200.shx 8200 シリーズターミナル用 BASIC ランタイムです。
bc8300.shx 8300 シリーズターミナル用 BASIC ランタイムです。
bc8400.shx 8400 シリーズターミナル用 BASIC ランタイムです。
bc8500.shx 8500 シリーズターミナル用 BASIC ランタイムです。
bc8700.shx 8700 シリーズターミナル用 BASIC ランタイムです。

Font Files

8000.8001 シリーズターミナル用フォント

フォントファイル名	フォントサイズ	言語
font-hebrew.shx	6x8, 8x16	ヘブライ語
font-japanese.shx	16x16 (4 行)	日本語
font-Japanese12.shx	6x12, 12x12 (5 行)	日本語
font-korean.shx	16x16 (4 行)	韓国語
font-korean12.shx	6x12, 12x12 (5 行)	韓国語
font-nordic.shx	6x8, 8x16	ノルディック語
font-polish.shx	6x8, 8x16	ポーランド語
font-russian.shx	6x8, 8x16	ロシア語
font-simplifiedchinese.shx	16x16 (4 行)	中国語
font-simplifiedchinese12.shx	6x12, 12x12 (5 行)	中国語
font-tradisionalchinese.shx	16x16 (4 行)	中国語
font-tradisionalchinese12.shx	6x12, 12x12 (5 行)	中国語
font-multi-language.shx	6x8, 8x16	多言語

Font Files

8200.8400.8700 シリーズ ターミナル用フォント

フォントファイル名	フォントサイズ	言語
font8x00-hebrew.shx	6x8, 8x16	ヘブライ語
font8x00-japanese.shx	16x16 (4 行)	日本語
font8x00-Japanese12.shx	6x12, 12x12 (5 行)	日本語
font8x00-korean.shx	16x16 (4 行)	韓国語
font8x00-korean12.shx	6x12, 12x12 (5 行)	韓国語
font8x00-nordic.shx	6x8, 8x16	ルンディック語
font8x00-polish.shx	6x8, 8x16	ポーランド語
font8x00-russian.shx	6x8, 8x16	ロシア語
font8x00-simplifiedchinese.shx	16x16 (4 行)	中国語
font8x00-simplifiedchinese12.shx	6x12, 12x12 (5 行)	中国語
font8x00-tradisionalchinese.shx	16x16 (4 行)	中国語
font8x00-tradisionalchinese12.shx	6x12, 12x12 (5 行)	中国語
font8x00-multi-language.shx	6x8, 8x16	多言語

8500 シリーズ ターミナル用フォント

フォントファイル名	フォントサイズ	言語
font8500-hebrew.shx	6x8, 8x16	ヘブライ語
font8500-japanese.shx	16x16 (4 行)	日本語
font8500-Japanese12.shx	6x12, 12x12 (5 行)	日本語
font8500-korean.shx	16x16 (4 行)	韓国語
font8500-korean12.shx	6x12, 12x12 (5 行)	韓国語
font8500-nordic.shx	6x8, 8x16	ルンディック語
font8500-polish.shx	6x8, 8x16	ポーランド語
font8500-russian.shx	6x8, 8x16	ロシア語
font8500-simplifiedchinese.shx	16x16 (4 行)	中国語
font8500-simplifiedchinese12.shx	6x12, 12x12 (5 行)	中国語
font8500-tradisionalchinese.shx	16x16 (4 行)	中国語
font8500-tradisionalchinese12.shx	6x12, 12x12 (5 行)	中国語
font8500-multi-language.shx	6x8, 8x16	多言語

2.2. BASICランタイムインジック

BASIC プログラムを実行するには、ターミナルに専用 BASIC ランタイムインジックをダウンロードしておく必要があります。BASIC ランタイムインジックは、BASIC コマンドにより生成された BASIC オブジェクトファイルを解釈しながら実行するインタープリタのような働きをします。

2.3. 開発ツール


BASICプログラムの開発は非常にシンプルで、下記の3ステップになります。

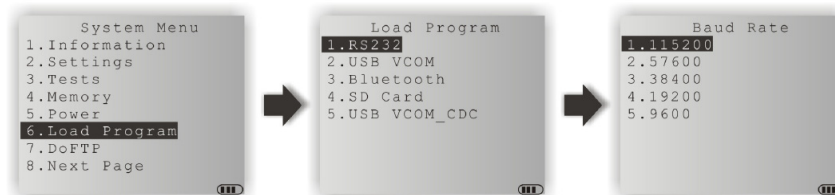
- Step 1 BASICランタイムインジンをターミナルにダウンロードする。
- Step 2 BASICプログラムをコーディングし、コンパイルする。
- Step 3 BASICオブジェクトファイルをターミナルにダウンロードして実行する

2.3.1. BASICランタイムインジンのダウンロード

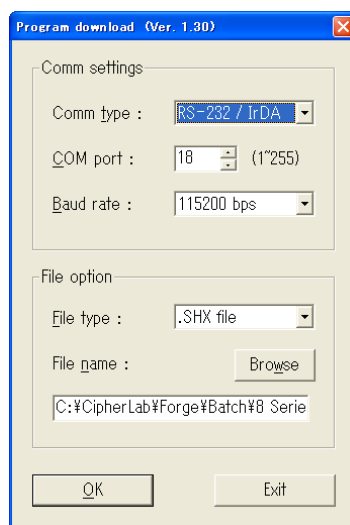
ダウンロードユーティリティ「proload.exe」を使って、システムメニューから BASICランタイムインジンのダウンロードを行ってください。

システムメニューの起動方法

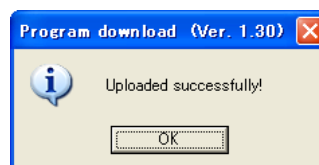
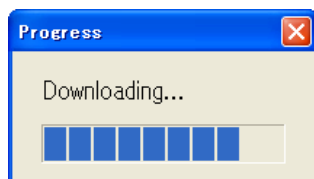
1. ターミナルの電源がオになっていることを確認します。
2. [7]キーと[9]キーを押しながら、電源[]キーを押して、ターミナルを立ち上げると、ピーピーというブザー音とともにシステムメニューが立ち上がります。




3. ダウンロードを行うインターフェイスを選択します。
4. ターミナルにインターフェイスケーブルを正しく接続します。クルードル経由の場合は、ターミナルをクルードルにセットします。
5. PC側で、プログラムダウンロードユーティリティ proload.exe を起動し、インターフェイス選択、BASICランタイムインジンファイルの選択を行い「OK」ボタンをクリックします。



6. ターミナルとの接続が正しく行われると、プログラムダウンロードが始まり、プログレスバーが表示されます。「Upload successfully!」と表示されればダウンロード完了です。エラーが発生する場合は、配線などをチェックの上、再度トライしてください。



参考

-  プログラムダウンロードユーティリティ proload.exe は、UART を制御しているため、ダウンロード中は、PCで他の作業を行わないようにしてください。
-  同様の手順で、必要なフォントファイルのダウンロードを行ってください。

2.3.2. BASICプログラムでコンパイル

BASICコンパイラ(bc.exe)は、シンプルなテキストファイルを装備しているため、他のソフトウェアを使用せず、コンパイラのみでコンパイルからプログラムダウンロードまでを行うことができます。

テキストファイルでコンパイルされたプログラムは、任意のファイル名で保存可能で、プログラム本体(.bas)と各種設定値を含む初期化ファイル(.ini)の2つのファイルに保存されます。また、BASICプログラムがエラー無しにコンパイルされると、同じ名前の BASIC オブジェクトファイル(.syn)が生成されます。

初期化ファイル(.ini)は、BASIC プログラムの一部であるため、プログラム配布する際には、必ず BASIC オブジェクトファイル(.syn)とセットで配布する必要があります。

2.3.3. BASICオブジェクトファイルのダウンロード


BASIC オブジェクトファイルのダウンロードは、BASIC コンパイラ(bc.exe)又はプログラムダウンロードユーティリティ(progload.exe)で行います。BASIC オブジェクトファイル.syn と.ini が必ずセットになっている必要があります。一方が存在しないとプログラムとして成立しないため、ダウンロードも行えませんので、ご注意ください。

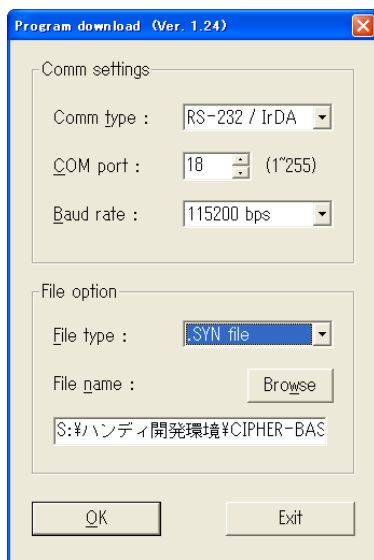
BASIC オブジェクトファイルのダウンロードが終了すると、ターミナルは自動的に再起動し、プログラムの実行を試みます。プログラムにエラーがある場合、ランタイムエラーが発生し、エラーコードとエラーメッセージがディスプレイに表示されます。下記を参照ください。

エラーコード	エラーメッセージ	説明
1	Unkown operator	不明な演算子です。
2	Operand count mismatch	演算子に対する引数の数が合いません。
3	Type mismatch	型が合いません。
4	Can't perform type conversion	型変換が行えません。
5	No available temp string	使用できる temp スtring がありません。
6	Illegal operand	許可されていない引数です。
7	Not an L-value	長整数(long)ではありません。
8	Float error	浮動小数点数エラー(float)です。
9	Bad array subscript	配列指定に誤りがあります。
10	Unkown function	不明な関数です。
11	Illegal function call	許可されていない関数コールです。
12	Return without GOSUB	Return ステートメントに対応する GOSUB ステートメントがありません。

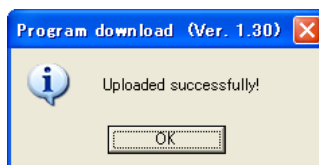
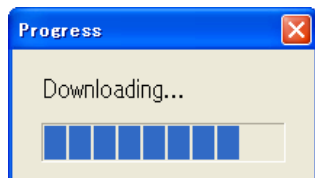
エラーが発生した場合は、ソースコードを修正し、再コンパイル後、ダウンロードを行ってください。

progload.exe を使用した BASIC オブジェクトファイルのダウンロード手順

1. ターミナルの電源が切になっていることを確認します。
2. [7]キーと[9]キーを押しながら、電源[]キーを押して、ターミナルを立ち上げると、ピーピーというブザー音とともにシステムメニューが立ち上がります。
3. 「6.Load Program」→「2.Load Basic」を選択します。
4. ダウンロードを行うインターフェイスなどを選択し、ダウンロード待ち状態「*Ready to Download*」にします。
5. ターミナルにインターフェイスケーブルを正しく接続します。クルートル経由の場合は、ターミナルをクルートルにセットします。
6. PC 側で、プログラムダウンロードユーティリティ progload.exe を起動し、インターフェイス選択、ファイルタイプ (.syn) 選択、BASIC オブジェクトファイルの選択を行い「OK」ボタンをクリックします。



7. ターミナルとの接続が正しく行われると、プログラムダウンロードが始まり、プログレスバーが表示されます。「Upload successfully!」と表示されればダウンロード完了です。エラーが発生する場合は、配線などをチェックの上、再度トライしてください。



参考

- プログラムダウンロードユーティリティ Proload.exe は、UART をコントロールしているため、ダウンロード中は、PC で他の作業を行わないようにしてください。

3. BASICコマンドを使ってみよう

Cipher BASIC コマンドの見た目は、昔ながらの Windows へのアプリケーションという感じで、ファイルマネージメント、テキストデータ機能、その他機能をツツプルにまとめ上げた BASIC プログラム開発環境になります。BASIC コマンドは、下記の Windows OS 上で実行可能です。

- ✓ Windows XP
- ✓ Windows Vista
- ✓ Windows 7

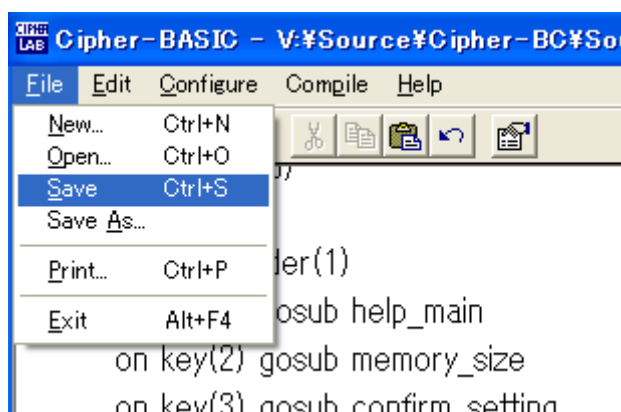
下記の 5 つのメニューが提供されており、各メニューには、それぞれサブメニューやコマンドが割り付けられています。

- ✓ File Menu (ファイルメニュー)
- ✓ Edit Menu (編集メニュー)
- ✓ Configure Menu (ソフトウェアレゾリューションメニュー)
- ✓ Compile Menu (コマンドビルドメニュー)
- ✓ Help Menu (ヘルプメニュー)

各メニューや機能に関する説明は、各節を参照ください。

3.1. File Menu(ファイルメニュー)

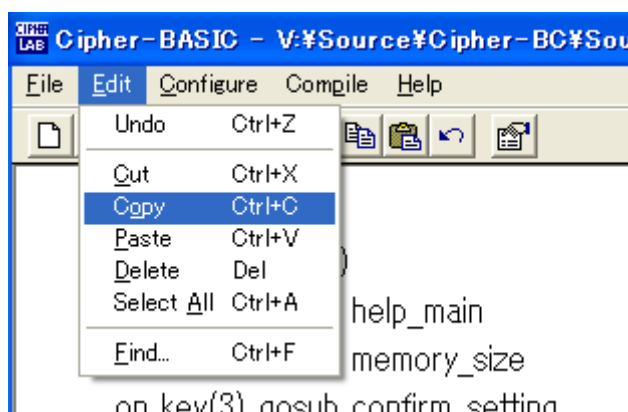
ファイルメニューには、6つのコマンドが用意されています。



メニュー/コマンド	説明
New (新規作成)	<p>機能説明 BASIC プログラムを新規作成します。</p> <p>アクセス方法 「File」...「New」の順でクリックします。又は、[CTRL]キーを押しながら[N]キーを押すか、 アイコンをクリックします。</p>
Open (開く)	<p>機能説明 保存されている BASIC プログラムファイルを開きます。</p> <p>アクセス方法 「File」...「Open」の順でクリックします。又は、[CTRL]キーを押しながら[O]キーを押すか、 アイコンをクリックします。</p>
Save (上書き保存)	<p>機能説明 編集中の BASIC プログラムファイルを上書き保存します。</p> <p>アクセス方法 「File」...「Save」の順でクリックします。又は、[CTRL]キーを押しながら[S]キーを押すか、 アイコンをクリックします。</p>
Save As (名前を付けて保存)	<p>機能説明 編集中の BASIC プログラムファイルに名前を付けて保存します。</p> <p>アクセス方法 「File」...「Save As」の順でクリックします。又は、[CTRL]キーを押しながら[S]キーを押すか、 アイコンをクリックします。</p>
Print (印刷)	<p>機能説明 編集中の BASIC プログラムファイルを印刷します。</p> <p>アクセス方法 「File」...「Print」の順でクリックします。又は、[CTRL]キーを押しながら[P]キーを押すか、 アイコンをクリックします。</p>
Exit (終了)	<p>機能説明 BASIC システムを終了します。</p> <p>アクセス方法 「File」...「Exit」の順でクリックします。又は、[ALT]キーを押しながら[F4]キーを押します。</p>

3.2. Edit Menu(編集メニュー)

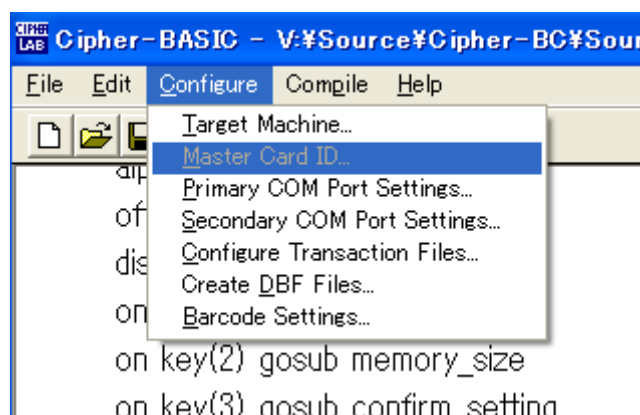
編集メニューには、BASICプログラムの編集に必要な7つのコマンドが用意されています。



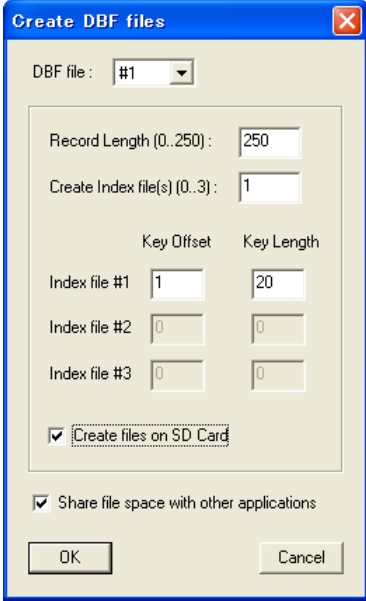
メニュー/コマンド	説明
Undo (元に戻す)	<p>機能説明 1 つ前の入力やコマンドをキャンセルし、元に戻します。</p> <p>アクセス方法 「Edit」...「Undo」の順でクリックします。又は、[CTRL]キーを押しながら[Z]キーを押すか、 アイコンをクリックします。</p>
Cut (切り取り)	<p>機能説明 選択されている文字列をクリップボードに切り取ります。</p> <p>アクセス方法 「Edit」...「Cut」の順でクリックします。又は、[CTRL]キーを押しながら[X]キーを押すか、 アイコンをクリックします。</p>
Copy (コピー)	<p>機能説明 選択されている文字列をクリップボードにコピーします。</p> <p>アクセス方法 「Edit」...「Copy」の順でクリックします。又は、[CTRL]キーを押しながら[C]キーを押すか、 アイコンをクリックします。</p>
Paste (貼り付け)	<p>機能説明 クリップボードの内容を貼り付けます。</p> <p>アクセス方法 「Edit」...「Paste」の順でクリックします。又は、[CTRL]キーを押しながら[V]キーを押すか、 アイコンをクリックします。</p>
Delete (削除)	<p>機能説明 選択されている文字列を削除します。クリップボードにはコピーされません。</p> <p>アクセス方法 「Edit」...「Delete」の順でクリックします。</p>
Select All (全て選択)	<p>機能説明 全ての文字列を選択します。</p> <p>アクセス方法 「Edit」...「All」の順でクリックします。又は、[CTRL]キーを押しながら[A]キーを押します。</p>
Find (検索)	<p>機能説明 文字列の検索を行います。ポップアップされた検索ウィンドウに検索したい文字列などを入力して、検索を実行します。</p> <p>アクセス方法 「Edit」...「Find」の順でクリックします。又は、[CTRL]キーを押しながら[F]キーを押すか、 アイコンをクリックします。</p>

3.3. Configure Menu(ソフトウェア レーションメニュー)

ソフトウェア レーションメニューには、システム設定を行うための 7 つのサブメニューが用意されています。



メニュー/コマンド	説明
Target Machine (開発対象ターミナル型式)	<p>機能説明 開発対象となるターミナルの型式をドロップダウンメニューから選択します。ターミナルによりトランザクションファイルの数など利用できるリソースが異なります。</p> <p>アクセス方法 「Configure」...「Target Machine」の順でクリックします。</p>
Master Card ID (マスターカード ID)	<p>機能説明 この設定は、固定式ターミナル専用です。</p>
Primary COM Port Settings (第 1 COMポート設定)	<p>機能説明 第 1 COMポートの設定を行います。</p> <p>アクセス方法 「Configure」...「Primary COM Port Settings」の順でクリックします。</p>
Secondary COM Port Settings (第 2 COMポート設定)	<p>機能説明 第 2 COMポートの設定を行います。</p> <p>アクセス方法 「Configure」...「Secondary COM Port Settings」の順でクリックします。</p>
Configure Transaction Files (トランザクションファイル設定)	<p>機能説明 トランザクションファイルの設定を行います。トランザクションファイルは、6 つ迄定義でき、それぞれ 1~255 バイトの固定データ長 (Fix Data Length) を設定します。データ長が決定すれば、アプリケーションプログラム用のトランザクションファイルエリアがターミナルのメモリ内に確保されます。ターミナルが SD カードに対応している場合は、SD カードにトランザクションファイルを作成することも可能です。</p> <div data-bbox="770 1469 1259 1928"> </div> <p>アクセス方法 「Configure」...「Configure Transaction Files」の順でクリックします。</p>

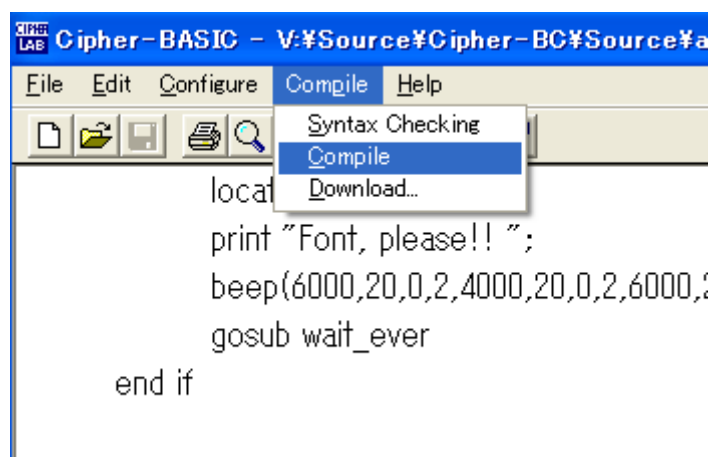
メニュー/コマンド	説明
Create DBF Files (DBF ファイル設定)	<p><u>機能説明</u></p> <p>DBF ファイルの設定を行います。DBF ファイルは、5 つ迄定義でき、それぞれにインデックス (IDX) ファイルを作成できます。レコード長は最大 250 バイトで、各 DBF ファイルに 3 つの迄のインデックスファイルを作成できます。ターミナルが SD カードに対応している場合は、SD カードに DBF ファイルを作成することも可能です。</p>  <p><u>アクセス方法</u></p> <p>「Configure」...「Create DBF Files」の順でクリックします。</p>
Barcode Setting (読取バーコード設定)	<p><u>機能説明</u></p> <p>読取バーコード及びバーコードリーダの設定を行います。ここで設定した値がアプリケーションプログラムの初期値となりますが、アプリケーションプログラムで任意の設定値に変更することが可能です。</p> <p><u>アクセス方法</u></p> <p>「Configure」...「Barcode Setting」の順でクリックします。</p>


参考

- 「Share file space ...」にチェックを入れると、新しいアプリケーションプログラムがダウンロードされてもトランスファクションファイル/DBF ファイルは削除されません。チェックを外すと、より大きいファイルシステムサイズを利用することができます。

3.4. Compile Menu(コンパイルメニュー)

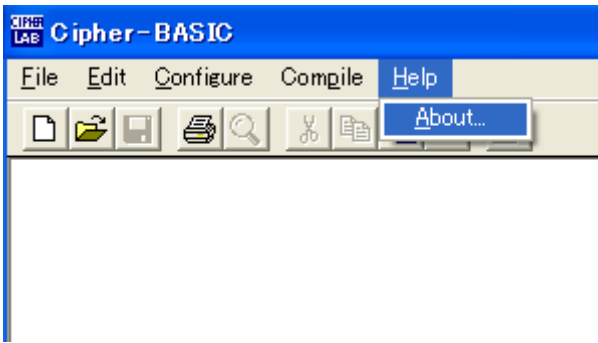
コンパイルメニューには、3つのコマンドが用意されています。



メニュー/コマンド	説明
Syntax Checking (構文チェック)	<p>機能説明 編集中の BASIC プログラムの構文チェックを行います。</p> <p>操作方法 「Compile」...「Syntax Checking」の順でクリックします。</p>
Compile (コンパイル)	<p>機能説明 編集中の BASIC プログラムをコンパイルします。コンパイルエラーが発生した場合、エラーが発生した行番号とエラー内容が出力されます。エラー箇所を修正し、再度コンパイルを行ってください。</p> <div data-bbox="794 1064 1236 1220" data-label="Image"> </div> <p>コンパイルが正常に終了すると、「Build successfully ...」というメッセージが表示されます。続けて、BASIC オブジェクトファイルをターミナルにダウンロードする場合は、「はい」をクリックし、編集画面に戻る場合は、「いいえ」をクリックします。</p> <div data-bbox="750 1377 1284 1556" data-label="Image"> </div> <p>操作方法 「Compile」...「Compile」の順でクリックします。又は、 アイコンをクリックします。</p>
Download (ダウンロード)	<p>機能説明 BASIC オブジェクトファイル(.syn)をターミナルにダウンロードします。ファイルの選択ウィンドウが開くので、ダウンロードしたい BASIC オブジェクトファイル(.syn)を選択してください。</p> <p>操作方法 「Compile」...「Download」の順でクリックします。</p>

3.5. Help Menu(ヘルプメニュー)

ヘルプメニューには、1つのサブメニューが用意されています。



メニュー/コマンド	説明
About (バージョン表示)	<div><p><u>機能説明</u></p><p>BASIC コンパイラのバージョン情報を表示します。</p></div> <div></div> <div><p><u>操作方法</u></p><p>「Help」...「About」の順でクリックします。</p></div>

4. CipherLab BASIC言語の基礎

4.1. 定数

文字列型と数値型の 2 種類の定数が使用できます。

4.1.1. 文字列型

ダブルクォーテーションで囲まれた英数字及び記号から構成された 255 文字までの文字列です。下記に例を示します。

- ✓ "Hello"
- ✓ "\$20,000.00"
- ✓ "12 students"

4.1.2. 数値型

下記の 3 種類の数値型定数が使用できます。

整数型定数	小数点を含まない -32,768~+32,767 の範囲の整数値
実数型定数	小数点を含む正・負の実数値 (例 5.34, -10.0 など)
長整数型定数	-2,147,483,648 ~ +2,147,483,647 の範囲の整数値

4.2. 変数

変数は、プログラム内で使用する数値や文字列を格納するための任意の名前が付けられた領域です。変数の初期値は、明示的に値が代入されるまで、不定値となります。

4.2.1. 変数名と修飾子

下記に変数名の規則と修飾子について説明します。

- ✓ 変数名は、英字 (A~Z) で始まる必要があります。
- ✓ 変数名の 2 文字目以降は、英数字及びアンダースコア () が使用可能です。
- ✓ 変数名の最後には、下記の修飾子を付加し、変数の型を定義します。

%	整数型	: 2 バイト (-32,768~+32,767)
&	長整数型	: 4 バイト (-2,147,483,648~+2,147,483,647)
!	実数型	: 4 バイト
\$	文字列型	: 255 バイト
無し (デフォルト)	整数型	: 2 バイト (-32,768~+32,767)
- ✓ 変数名に Cipher BASIC の予約語は使えません。
- ✓ Cipher BASIC は、整数型、長整数型、実数型、文字列型の 4 種類の変数のみをサポートしています。また、最大 1000 個までの変数を BASIC プログラム内で使用することができます。
- ✓ 変数の大文字・小文字の区別はありません。TEMP1 と temp1 は、同じ変数名と解釈されます。

実数型について

内部では、全ての小数は、 2 進数で表現されています。しかし、これは真の値を実際に表現している訳ではありません。 2 進数における浮動小数点システムが制限された浮動小数点値しか表現できないことを知っておく必要があります。表現できない値に関しては、目的とする値に一番近い値が採用されます。例えば、 $0.1 (1/10)$ という小数 (分数) は、 10 進数では 0.1 と表現できるのに対し、同じ数を 2 進数で表現すると、次のような循環 2 進数になります。

0001100110011100110011 (以下、繰り返し)

この数値は、上記のように無限に繰り返され、有限量の領域では表現できません。よって、この数値が格納されるときには、目的とする値に一番近い値に丸められます。

REM 浮動小数点エラー

fnum1! = 99999.1

fnum2! = 99999.0

SET_PRECISION(4)

REM 99999.1000 ではなく 99999.16 と表示されます

PRINT fnum1!

REM 10 ではなく 10.1563 と表示されます

PRINT (fnum1!-fnum2!)*100

REM “99999.1-99999.0”と“0.1”の比較の結果は Not Equal と表示されます

IF (fnum1!-fnum2! <> 0.1) THEN

PRINT “Not equal”

ELSE

PRINT “Equal”

END IF

浮動小数点値を直接扱うのではなく、計算を行う前に実数を整数に変換し、計算終了後に必要に応じて、整数を実数に再変換することをお奨めします。例えば、 $1.82-1.8$ という計算の場合、 $182-180$ のように整数での計算に置き換え、計算終了後に結果を 100 で割り、最終計算結果 0.02 を導き出します。

浮動小数点値を表示・印字・計算すると、精度が落ちます。よって、算術計算や論理演算を行う場合は、代わりに整数や長整数を使用するようにします。どうしても浮動小数点値を表示する必要がある場合は、整数値又は長整数値を文字列に変換し、小数点を適当な位置に挿入してください。下記の例を参照ください。

num1& = 999991

num2& = 999992

num3& = (num1&-num2&)*100

REM 99999.1 と表示

PRINT (num1& ¥ 10);“.”;(num1& mod 10))

REM 10.0 と表示

PRINT (num3& ¥ 10);“.”;(num3& mod 10))

4.2.2. 配列変数

Cipher Basic は、2 次元迄の配列変数を使用することができます。配列変数の名前付けの規則は、「4.2.1. 変数名と修飾子」を参照ください。

✓ 配列変数の宣言

配列変数は、使用する前に必ず明示的に宣言を行う必要があります。

```
DIM IntegerA%(20) : 20 個の 1 次元要素を持つ整数型配列変数を宣言
DIM StringB$(100) : 100 個の 1 次元要素を持つ文字列型配列変数を宣言
DIM RealC!(10)    : 10 個の 1 次元要素を持つ実数型配列変数を宣言
DIM Tb(5,5)       : 5 x 5 個の 2 次元要素を持つ整数型配列変数を宣言
```

✓ 配列変数の参照方法

```
A(12)           : 1 次元配列変数、12 番目の要素を参照
T(2,5)          : 2 次元配列変数、2 番目 x 5 番目の要素を参照
X(i+1,y)        : 整数型変数を用いた配列要素の参照
```

- ✓ 配列要素は、1 から始まります。例えば、DIM X(10) の場合は、配列変数は、X(1)~X(10) となります。
- ✓ Cipher BASIC では、2 次元迄の配列を使用でき、各配列要素の最大は、32,767 となります。

4.3. 演算子

Cipher BASIC では、様々な演算子が利用可能です。ここでは、代入演算子・算術演算子・比較演算子など利用可能な演算子については説明します。

4.3.1. 代入演算子

代入演算子 = は、変数に値を代入します。下記に例を示します。

- ✓ Length% = 100
- ✓ PI! = 3.14159
- ✓ Company\$ = "Cipher BASIC"
- ✓ X = i + j +100

4.3.2. 算術演算子

下記の算術演算子が利用可能です。

演算子	機能	例
^	べき乗	A% = 9 ^ 3
-	負の数値	A% = -B%
*	乗算	A! = B! * C!
¥	除算 (整数、余り切り捨て)	A% = B! ¥ C!
/	除算 (実数)	A! = B! / C!
+	加算	A% = B% + C%
-	減算	A% = B% - C%
MOD	除算余り	A% = B% MOD C%

4.3.3. 比較演算子

比較演算子は、2つの値を比較し、その結果を "True" 又は "False" の何れかで返します。プログラムでは、この結果を利用して、処理の分岐などを行います。

下記の比較演算子が利用可能です。

演算子	機能	例
=	等しい	IF A% = B% THEN GOTO Label1
<>	等しくない	IF A% <> B% THEN GOTO Label1
><	等しくない	IF A% >< B% THEN GOTO Label1
>	より大きい	IF A% > B% THEN GOTO Label1
<	より小さい	IF A% < B% THEN GOTO Label1
>=	以上 (より大きい又は等しい)	IF A% >= B% THEN GOTO Label1
<=	以下 (より小さい又は等しい)	IF A% <= B% THEN GOTO Label1

4.3.4. 論理演算子

論理演算子は、比較演算式による複数条件評価やビット演算処理に使用され、式の評価結果は “True” 又は “False” の何れかで返されます。論理演算子は、算術演算子、比較演算子の処理後に実行されます。

下記の論理演算子が利用可能です。

演算子	機能	例
NOT	論理否定	IF NOT (A% = B%) THEN I+1
AND	論理積	IF (A% = B%) AND (C% = D%) THEN I+1
OR	論理和	IF (A% = B%) OR (C% = D%) THEN I+1
XOR	排他的論理和	IF (A% = B%) XOR (C% = D%) THEN I+1

4.3.5. 演算子の優先順位

下記に演算子の優先順位を示します。同一欄に列挙されている演算子は同じ優先順位となり、式中で同一優先順位の演算子が複数使用されている場合は、現れる順序で左から右に評価されます。

優先順位	演算子の種類	演算子
高い	算術演算子 - べき乗演算子	^
↓	算術演算子 - 乗算、除算演算子	*, ¥, / MOD
↓	算術演算子 - 加算、減算演算子	+, -
↓	比較演算子	=, <>, >, <, >=, <=
↓	論理演算子	AND, NOT, OR, XOR
低い	代入演算子	=


4.4. ラベル

Cipher BASIC では、プログラム中の特定位置 (行) にラベルをつけることができます。ラベル名には、整数値又は英数字が使用でき、英数字で命名されたラベル名には、必ず後ろにコロン (:) を付ける必要があります。下記に例を示します。

```
GOTO 100
...
100 PRINT "This is an integer label."
...
GOTO Label2
...
Label2: PRINT "This is a character string label."
```

- ✓ ラベルに使用できる整数値は、1~32,767 の範囲です。ラベルが整数値の場合は、後ろにコロン (:) を付ける必要はありません。
- ✓ ラベルに使用できる文字列 (英数字) は、49 文字迄です。49 文字を超えるラベル名は、切り詰められます。ラベルが文字列 (英数字) の場合は、後ろにコロン (:) を付ける必要があります。
- ✓ プログラム中で使用できるラベル数は、最大 1,000 です。

注意

 Cipher BASIC では、最大 12,000 行迄のプログラムが開発可能です。(トライアルバージョンは、1,000 行)

4.5. サブルーチン

Cipher BASIC では、他の BASIC 言語と同様にサブルーチンを記述することができます。各サブルーチンは、ユニーク(重複しない)なラベル名を持ち、GOSUB コマンドを使って呼び出すことができます。

下記に例を示します。

例 1)



```
ON KEY(1) GOSUB KeyIn
...
KeyIn:
    PRINT "F1 is pressed"
RETURN
```

例 2)

```
GOSUB Calc
...
ResultOfSubroutine:
PRINT "The result was "; A

Calc:
    A = B * C
RETURN ResultOfSubroutine
```

参考

-  RETURN ステートメントの後ろにラベル名を指定することで、サブルーチンの処理終了後、特定ラベル位置から処理を再開することができます。
 -  Cipher BASIC では、全ての変数がグローバル変数として扱われるため、引数の受け渡しやサブルーチン呼び出し前に変数へ値を代入する必要はありません。
-

サブルーチンは再帰型であるため、下記の例のように、サブルーチンが自身を再呼び出したり、サブルーチン内から別のサブルーチンを呼び出すこともできます。但し、複雑なサブルーチンの呼び出しは、プログラムの複雑にし、プログラミंगミスや暴走の要因となるため、お奨めできません。

```
PRINT "Please enter a number(1-13):"
INPUT N%
FactResult! = 1
Fact% = 1
GOSUB Factorial
PRINT N%, "!=" , FactResult

Loop:
    GOTO Loop

Factorial:
    IF Fact% < 1 THEN RETURN
    FactResult! = FactResult! * Fact%
    Fact% = Fact% -1
    GOSUB Factorial
RETURN
```

5. BASICリファレンス

本章 BASIC リファレンスでは、Cipher BASIC がサポートするコマンドについて、それぞれ例を交えながら説明を行っています。下記に説明の見方について説明します。

適用機種

使用できる各機種の機種名を記載しています。特に機種依存がない場合は、ALL と記載しています。

目的

使用目的を説明します。

構文

構文 (書式) を説明します。説明は、下記の規則に従って表記されています。

CAPS 大文字アルファベット

説明を行うコマンド名を意味します。実際にコーディングを行う場合は、大文字/小文字の区別は行われなため、何れを入力しても正しく動作します。

Italics イタリック書体

引数を意味します。

[] 角括弧

省略可能な引数を意味します。

{ } 波括弧

必要に応じて繰り返し指定できる引数を意味します。

| 垂直バー

排他的引数を意味します。プログラム中で、垂直バーで区切られた引数の何れかを指定します。

解説

コマンドの使用法・引数・戻り値の解説などを記載します。

使用例

プログラムの例を記載します。

関連項目

関連するコマンドを記載します。

5.1. 一般的なコマンド

ABS

適用機種	ALL
目的	引数で与えられた数値の絶対値を返します。
構文	REM Result%は結果を代入するための整数型変数です。 Result% = ABS(n)
解説	引数 <i>n</i> には、整数値 (整数型変数) 又は実数値 (実数型変数) を指定します。
使用例	TimeDifference% = ABS(Time1% - Time2%)

BIT_OPERATOR

適用機種	ALL
目的	ビット演算を行い結果を返します。
構文	、 Result%は結果を代入するための整数型変数です。 Result% = BIT_OPERATOR(operator%, a, b)
解説	引数 operator% には、ビット演算タイプを 1~3 の範囲で指定します。 1 : AND (論理積) 2 : OR (論理和) 3 : XOR (排他的論理和) 引数 <i>a</i> には、第 1 引数として整数値 (整数型変数) 又は実数値 (実数型変数) を指定します。 引数 <i>b</i> には、第 2 引数として整数値 (整数型変数) 又は実数値 (実数型変数) を指定します。
使用例	Result& = BIT_OPERATOR(2, 1100, 1000)

DIM

適用機種	ALL
目的	配列変数を宣言し、メモリ領域を確保します。
構文	DIM Array(range, {, range}), {, Array(range, {, range})}
解説	引数 range には、配列要素数を整数値 (整数型変数) で指定します。 配列変数の宣言を行うと、全ての配列要素は 0 又は空文字列で初期化されます。 2 次元までの配列変数が宣言可能です。
使用例	DIM Array1(10), ArrayB&(20), ArrayC\$(30, 10)

GOSUB

適用機種	ALL
目的	サブルーチンをコールします。
構文	GOSUB SubName
解説	引数 SubName には、コールしたいサブルーチン名 (ラベル名) を指定します。
使用例	GOSUB DoIt DoIt: PRINT "Now I've done it!" RETURN

GOTO

適用機種	ALL
目的	指定のラベルへ処理をジャンプします。
構文	<code>GOTO LineLabel</code>
解説	引数 <i>LineLabel</i> には、処理をジャンプしたいラベル名を指定します。
使用例	<pre>Loop: GOTO Loop</pre>

INT

適用機種	ALL
目的	引数で与えられた値に等しいか、それより小さい一番近い整数値を返します
構文	<code>` Result%は結果を代入するための整数型変数です。</code> <code>Result% = INT(n%)</code>
解説	引数 <i>n%</i> には、小数点を含む数値を指定します。
使用例	<pre>Result% = INT(-2.86) ` Result% = -3 Result% = INT(2.86) ` Result% = 2</pre>

REM

適用機種	ALL
目的	プログラム中にコメントを入れます。
構文	<code>REM remark</code> <code>` remark</code>
解説	コロンは、REM に続く文字列を全てコメントとして解釈し、無視します。REM は、アポストロフィー (') で代用することが可能です。
使用例	<pre>REM これはコメントです ` This is a comment</pre>

SET_PRECISION

適用機種	ALL
目的	実数の小数点以下の精度を設定します。
構文	<code>SET_PRECISION(n%)</code>
解説	引数 <i>n%</i> には、小数点以下の精度を表す 0~6 の値を指定します。デフォルトは、2 です。
使用例	<pre>PI! = 3.14159 PRINT "PI = ",PI! ` PI = 3.14 (デフォルト精度 2 桁) SET_PRECISION(6) PRINT "PI = ",PI! ` PI = 3.141590 (精度 6 桁) SET_PRECISION(3) PRINT "PI = ",PI! ` PI = 3.141 (精度 3 桁)</pre>

SGN

適用機種	ALL						
目的	引数で与えられた値を評価し、符号を数値で返します。						
構文	<code>` Result%は結果を代入するための整数型変数です。 Result% = SGN(n%)</code>						
解説	引数 $n\%$ には、整数値 (整数型変数) を指定します。 引数で与えられた値を評価し、下記の数値を返します。 <table><tr><td>1</td><td>: 0 より大きい (プラスの値)</td></tr><tr><td>0</td><td>: 0 である</td></tr><tr><td>-1</td><td>: 0 より小さい (マイナスの値)</td></tr></table>	1	: 0 より大きい (プラスの値)	0	: 0 である	-1	: 0 より小さい (マイナスの値)
1	: 0 より大きい (プラスの値)						
0	: 0 である						
-1	: 0 より小さい (マイナスの値)						
使用例	<table><tr><td><code>Result1% = SGN(100)</code></td><td><code>` Result1% = 1</code></td></tr><tr><td><code>Result2% = SGN(-1.5)</code></td><td><code>` Result1% = -1</code></td></tr></table>	<code>Result1% = SGN(100)</code>	<code>` Result1% = 1</code>	<code>Result2% = SGN(-1.5)</code>	<code>` Result1% = -1</code>		
<code>Result1% = SGN(100)</code>	<code>` Result1% = 1</code>						
<code>Result2% = SGN(-1.5)</code>	<code>` Result1% = -1</code>						

5.2. 条件処理コマンド

IF ... THEN ... [ELSE] END IF

適用機種	ALL
目的	条件に応じて、処理を行います。
構文	<pre>IF Condition THEN Action1 [ELSE Action2] END IF</pre>
解説	<p><i>Condition</i> には、比較演算子を含む条件式を指定します。</p> <p><i>Action1</i> には、<i>Condition</i> が真 (True) の場合に行う処理を記述します。</p> <p><i>Action2</i> には、<i>Condition</i> が偽 (False) の場合に行う処理を記述します。</p>
使用例	<pre>IF Data1% > Data2% THEN Temp% = Data1% ELSE Temp% = Data2%</pre>

IF ... THEN ... {ELSE IF ...} [ELSE] END IF

適用機種	ALL
目的	条件 (複数条件) に応じて、処理を行います。
構文	<pre>IF Condition1 THEN ActionBlock1 {ELSE IF Condition2 THEN ActionBlock2} [ELSE ActionBlockN] END IF</pre>
解説	<p><i>Condition1</i> 及び <i>Condition2</i> には、比較演算子を含む条件式を指定します。</p> <p><i>ActionBlock1</i> には、<i>Condition1</i> が真 (True) の場合に行う処理を記述します。</p> <p><i>ActionBlock2</i> には、<i>Condition2</i> が真 (True) の場合に行う処理を記述します。</p> <p><i>ActionBlockN</i> には、全ての <i>Condition</i> が偽 (False) の場合に行う処理を記述します。</p>
使用例	<pre>IF LEFT\$(String1\$,1) = "A" THEN PRINT "String1 is led by A." ELSE IF LEFT\$(String1\$,1) = "B" THEN PRINT "String1 is led by B." ELSE PRINT "String1 is not led by A nor B." END IF</pre>

ON ... GOSUB ...

適用機種	ALL
目的	値によってサブ ルーチンをコールします。
構文	ON <i>n%</i> GOSUB <i>SubName</i> {, <i>SubName</i> }
解説	引数 <i>n%</i> には、整数値 (整数型変数) を指定します。 値が 1 の場合は、1 つ目のサブ ルーチン、2 の場合は、2 つ目のサブ ルーチンという具合に、値によって指定されているサブ ルーチンをコールします。値が 0 の場合及び指定されているサブ ルーチンの数より大きい場合はサブ ルーチンはコールせず、次の行に処理を移します。
使用例	<pre>PRINT Input a number(1-9):" INPUT Num% CLS ON Num% GOSUB 100, 100, 100, 200, 200, 300, 400, 400, 400 ... 100 PRINT "Numer 1-3 is input" RETURN 200 PRINT "Numer 4-5 is input" RETURN 300 PRINT "Numer 6 is input" RETURN 400 PRINT "Numer 7-9 is input" RETURN ...</pre>

ON ... GOTO ...

適用機種	ALL
目的	値によって指定ラベルへ処理をジャンプします。
構文	ON <i>n%</i> GOTO <i>LineLabel</i> {, <i>LineLabel</i> }
解説	引数 <i>n%</i> には、整数値 (整数型変数) を指定します。 値が 1 の場合は、1 つ目のラベル、2 の場合は、2 つ目のラベルという具合に、値によって指定されているラベルへジャンプします。値が 0 の場合及び指定されているラベルの数より大きい場合はラベルへのジャンプは行わず、次の行に処理を移します。
使用例	<pre>PRINT Input a number(1-9):" INPUT Num% CLS ON Num% GOTO 100, 100, 100, 200, 200 ... 100 PRINT "Numer 1-3 is input" GOTO FIN 200 PRINT "Numer 4-5 is input" GOTO FIN FIN: PRINT "FIN" ...</pre>

5.3. ループ処理コマンド

Cipher Basic では、FOR...NEXT と WHILE...WEND の 2 つのループ処理コマンドが使用でき、10 層迄のネストが行えます。

EXIT

適用機種	ALL
目的	FOR...NEXT や WHILE...WEND ループ処理から抜けだします。
構文	EXIT
解説	EXIT コマンドはループコマンド内のどこにでも記述可能です。
使用例	<pre>DataCount% = TRANSACTION_COUNT FOR Counter% = 1 TO DataCount% Data\$ = GET_TRANSACTION_DATA\$(Counter%) HostCommand\$ = READ_COM\$(1) IF HostCommand\$ = "STOP" THEN EXIT WRITE_COM(1,Data\$) NEXT</pre>

FOR ... NEXT

適用機種	ALL
目的	FOR...NEXT で囲まれたコマンドブロックを指定回数実行します。
構文	<pre>FOR n% = startvalue TO endvalue [STEP step] [COMMAND BLOCK] NEXT</pre>
解説	<p><i>n%</i> は、ループカウンタとなる整数型変数です。</p> <p><i>startvalue</i> には、ループカウンタの初期値を指定します。</p> <p><i>endvalue</i> には、ループカウンタの最終値を指定します。</p> <p><i>step</i> には、ループカウンタの増分値を指定します。省略が可能で、デフォルト値は 1 です。マイナスの値を指定することもできます。</p> <p>指定された最終値に達するか、それを超えるとループ処理を終了し、NEXT に続くコマンドに処理が移ります。EXIT コマンドを実行した場合も、ループ処理を終了し、NEXT に続くコマンドに処理が移ります。</p>
使用例	<pre>DataCount% = TRANSACTION_COUNT FOR Counter% = 1 TO DataCount% Data\$ = GET_TRANSACTION_DATA\$(Counter%) WRITE_COM(1,Data\$) NEXT</pre>

WHILE ... WEND

適用機種	ALL
目的	WHILE...WEND で囲まれたコマンドブロックを条件が真 (True) の間実行します。
構文	<pre>WHILE condition [COMMAND BLOCK] WEND</pre>
解説	<p><i>condition</i> には、比較演算子を含む条件式を指定します。</p> <p><i>condition</i> で指定された条件式が偽 (False) になると、ループ処理を終了し、WEND に続くコマンドに処理が移ります。EXIT コマンドを実行した場合も、ループ処理を終了し、WEND に続くコマンドに処理が移ります。</p>
使用例	<pre>WHILE TRANSACTION_COUNT > 0 Data\$ = GET_TRANSACTION_DATA\$(1) WRITE_COM(1,Data\$) DEL_TRANSACTION_DATA(1) NEXT</pre>

5.4. 文字列処理コマンド

Cipher BASIC では、NULL から最大 250 バイトの可変長文字列を取り扱うことができます。

5.4.1. 文字列の結合

文字列の結合は、プラス(+) 演算子で行います。下記に例を示します。

```
A$ = "ABC"
Data$ = "123"+A$+"321"
PRINT Data$           \ Data$ = "123ABC321"
```

5.4.2. 文字列の比較

文字列の比較は、比較演算子で行います。

1 文字を比較する場合は、その文字の ASCII 値を元に比較が行われます。例えば、"B">"A"という評価式は、42hex>41hex と解釈され、真 (True) となります。

2 文字以上の文字列を比較する場合、先頭から順に ASCII 値を元に比較が行われ、結果が導かれます。例えば、"aaabaa">"aaaaaaaa"という評価式の場合、最初の 3 文字までは同じとなり、4 文字目の"b"が"a"より大きいため、結果は、真 (True) となります。

文字列の長さが同じで内容にも全く相違が無い場合は、2 つの文字列は等しい(イコール)という評価になりますが、文字列の長さが異なる場合は、長い方の文字列が大きいという評価になります。例えば、"aaaaaaaa">"aaaaa"と"abc"="abc"という評価式は、何れも真 (True) となります。

文字列の先頭及び後方にある空白(スペース)は、文字列を比較する上で重要な意味を持ちます。例えば、" ABC"は"ABC"より小さいと評価されます。これは、先頭の空白(20hex)が A(41hex)より小さいためです。また、逆に"ABC "は"ABC"より大きいと評価されます。

5.4.3. 文字列の長さ

LEN

適用機種 ALL

目的 文字列の長さを取得します。

構文 \ Result%は結果を代入するための整数型変数です。
Result% = LEN(x\$)

解説 引数 x\$には、文字列(文字列型変数)を指定します。
引数で与えられた文字列の長さ(桁数)を返します。スペースや制御文字などもカウントされます。

使用例 String1\$ = "ab c de"
Result% = LEN(String1\$) \ Result% = 7(スペースを含む)

5.4.4. 文字列の検索

INSTR

適用機種	ALL
目的	指定文字列を検索し、結果を返します。
構文	` Result%は結果を代入するための整数型変数です。 Result% = INSTR\$([n%],x\$,y\$)
解説	引数 n%に文字列検索を開始したい位置を 1~255 の範囲で指定します。省略した場合、1 桁目から検索されます。 引数 x\$には、検索対象となる元の文字列 (文字列型変数) を指定します。 引数 y\$には、検索を行う文字列 (文字列型変数) を指定します。 <u>x\$の中に y\$が見つかった場合</u> y\$が見つかった先頭位置が返されます。 <u>n%が x\$の長さより大きい又は x% が Null の場合 (y\$が見つからない)</u> 0 が返されます。 <u>y\$ が Null の場合</u> n%が返されます。n%が省略されている場合は、1 が返されます。
使用例	String1\$ = "11025John Thomas, Accounting Manager" String2\$ = ", " Name\$ = MID\$(String1\$,6,INSTR(String1\$,String2\$)-6) ` John Thomas

5.4.5. 文字列の抽出・編集

LEFT\$

適用機種	ALL
目的	文字列の左から指定桁数を抽出します。
構文	` Result\$は結果を代入するための文字列型変数です。 Result\$ = LEFT\$(x\$,n%)
解説	引数 x\$には、抽出対象となる元の文字列 (文字列型変数) を指定します。 引数 n%には、抽出したい桁数を 0~255 の範囲で指定します。 <u>n%が x\$の長さより大きい場合</u> x\$全てが返されます。 <u>n%が 0 の場合</u> 長さ 0 の Null 文字列が返されます。
使用例	String1\$ = "11025John Thomas, Accounting Manager" Id\$ = LEFT\$(String1\$,5) ` 11025

MID\$

適用機種	ALL
目的	文字列の指定開始位置から指定桁数を抽出します。
構文	、 Result\$は結果を代入するための文字列型変数です。 Result\$ = MID\$(x\$,n%,m%)
解説	引数 x\$ には、抽出対象となる元の文字列 (文字列型変数) を指定します。 引数 n% には、抽出を開始したい位置を 0～255 の範囲で指定します。 引数 m% には、抽出をしたい桁数を 0～255 の範囲で指定します。省略した場合は、指定開始位置以降全てを抽出します。 <u>m%が省略又は開始位置以降の残桁数が m%よりも小さい場合</u> 開始位置以降の全てが返されます。 <u>m%が 0 又は n%が文字列の桁数より大きい場合</u> 長さ 0 の Null 文字列が返されます。
使用例	String1\$ = "11025John Thomas, Accounting Manager" String2\$ = ", " Name\$ = MID\$(String1\$,6,INSTR(String1\$,String2\$)-6) ` John Thomas

RIGHT\$

適用機種	ALL
目的	文字列の右から指定桁数を抽出します。
構文	、 Result\$は結果を代入するための文字列型変数です。 Result\$ = RIGHT\$(x\$,n%)
解説	引数 x\$ には、抽出対象となる元の文字列 (文字列型変数) を指定します。 引数 n% には、抽出したい桁数を 0～255 の範囲で指定します。 <u>n%が x\$の長さより大きい場合</u> x\$全てが返されます。 <u>n%が 0 の場合</u> 長さ 0 の Null 文字列が返されます。
使用例	String1\$ = "11025John Thomas, Accounting Manager" Id\$ = RIGHT\$(String1\$,7) ` Manager

TRIM_LEFT\$

適用機種	ALL
目的	文字列の前方にある`-sを除去します。
構文	、 Result\$は結果を代入するための文字列型変数です。 Result\$ = TRIM_LEFT\$(x\$)
解説	引数 x\$ には、`-sを除去したい元の文字列 (文字列型変数) を指定します。
使用例	String1\$ = " Hello World!" Result\$ = TRIM_LEFT\$(String1\$) ` Hello World!

TRIM_RIGHT\$

適用機種	ALL
目的	文字列の後方にある \backslash - \backslash を除去します。
構文	\backslash Result\$は結果を代入するための文字列型変数です。 Result\$ = TRIM_RIGHT\$(x\$)
解説	引数 x\$には、 \backslash - \backslash を除去したい元の文字列 (文字列型変数) を指定します。
使用例	String1\$ = "Hello World! " Result\$ = TRIM_RIGHT\$(String1\$) \ Hello World!

5.4.6. 文字列の変換

ASC

適用機種	ALL
目的	文字 (文字列の 1 文字目) を ASCII コード (10 進数) に変換します。
構文	\backslash Result%は結果を代入するための整数型変数です。 Result% = ASC(x\$)
解説	引数 x\$には、変換を行いたい文字列 (文字列型変数) を指定します。
使用例	String1\$ = "John Thomas" Result% = ASC(String1\$) \ 74

CHR\$

適用機種	ALL
目的	数値 (ASCII コード) を文字に変換します。
構文	\backslash Result\$は結果を代入するための文字型変数です。 Result\$ = CHR\$(n%)
解説	引数 n%には、変換を行いたい整数値 (整数型変数) を指定します。
使用例	Result\$ = CHR\$(64) \ A

HEX\$

適用機種	ALL
目的	数値を 16 進数 (文字列) に変換します。
構文	\backslash Result\$は結果を代入するための文字型変数です。 Result\$ = HEX\$(n%)
解説	引数 n%には、変換を行いたい整数値を 0~2,147,483,647 の範囲で指定します。
使用例	Result\$ = HEX\$(140) \ 8C

LCASE\$

適用機種	ALL
目的	文字列を大文字から小文字に変換します。
構文	` Result\$は結果を代入するための文字型変数です。 Result\$ = LCASE\$(n%)
解説	引数 n%には、変換を行いたい文字列 (文字列型変数) を指定します。
使用例	Result\$ = LCASE\$("John Thomas") ` john thomas

OCT\$

適用機種	ALL
目的	数値を 8 進数 (文字列) に変換します。
構文	` Result\$は結果を代入するための文字型変数です。 Result\$ = OCT\$(n%)
解説	引数 n%には、変換を行いたい整数値を 0~2,147,483,647 の範囲で指定します。
使用例	Result\$ = OCT\$(24) ` 30

STR\$

適用機種	ALL
目的	数値を文字列に変換します。
構文	` Result\$は結果を代入するための文字型変数です。 Result\$ = STR\$(n%)
解説	引数 n%には、変換を行いたい整数値 (整数型変数) を指定します。
使用例	Result\$ = STR\$(123456) ` 123456

UCASE\$

適用機種	ALL
目的	文字列を小文字から大文字に変換します。
構文	` Result\$は結果を代入するための文字型変数です。 Result\$ = UCASE\$(x\$)
解説	引数 x\$には、変換を行いたい文字列 (文字列型変数) を指定します。
使用例	Result\$ = UCASE\$("John Thomas") ` JOHN THOMAS

VAL

適用機種	ALL
目的	文字列を数値に変換します。
構文	<code>` Result&は結果を代入するための長整数型変数 (又は整数型変数) です。 Result& = VAL(x\$)</code>
解説	<p>引数 <code>x\$</code> には、変換を行いたい文字列 (文字列型変数) を指定します。尚、指定した文字列の 1 文字目が数字でない場合は、0 を返します。</p> <p>VAL 関数は、先頭にある <code>^</code> や <code>%</code>、<code>!</code> を無視します。 取り扱える数値の範囲は、<code>-2,147,483,648~2,147,483,647</code> です。</p>
使用例	<pre>ON HOUR__SHARP GOSUB OnHourAlarm ... OnHourAlarm: Hour% = VAL(LEFT\$(TIME\$,2)) FOR Counter% = 1 TO Hour% BEEP(800,50) WAIT(200) NEXT RETURN</pre>

VALR

適用機種	ALL
目的	文字列を実数に変換します。
構文	<code>` Result!は結果を代入するための実数型変数です。 Result! = VAL(x\$)</code>
解説	<p>引数 <code>x\$</code> には、変換を行いたい文字列 (文字列型変数) を指定します。 変換後の精度は、SET_PRECISION 関数による設定に依存します。</p>
使用例	<pre>A! = VAL("1213.45") PRINT "A = ",A! ` A = 123.45</pre>

5.4.7. 文字列の生成

STRING\$

適用機種	ALL
目的	指定の文字 (又は ASCII コード (10 進数)) を指定数生成します。
構文	<code>` Result\$は結果を代入するための文字列型変数です。 Result\$ = STRING\$(n%, j%) Result\$ = STRING\$(n%, x\$)</code>
解説	<p>引数 <code>n%</code> には、繰り返し生成したい文字の数を 0~255 の範囲で指定します。 引数 <code>j%</code> には、生成したい文字の ASCII コード (10 進数) を指定します。 引数 <code>x\$</code> には、生成したい文字を指定します。</p>
使用例	<pre>IDX_LENGTH% = 20 ` 文字数が 20 桁になるよう^を後方に追加 Data\$ = Name\$ + STRING\$(IDX_LENGTH-LEN(Name\$)," ") ADD_RECORD(1,Data\$)</pre>

5.5. イベント処理コマンド

Cipher BASIC では、キーによるイベントやリアルタイムによるイベントなどに対応した幾つかの割り込みイベントコマンドが用意されています。イベント処理では、ホールドと割り込みの2通りのコーディング方法が利用できます。

イベントホールドでは、プログラム中の適切なポイントでイベントトリガとなる条件を常に監視する方法が取られます。例えば、下記の例では、キー入力を監視し、何らかのキー入力が行われるまでループ処理が繰り返されます。

```
LOOP:
    KeyData$ = INKEY
    IF KeyData$ = "" THEN GOTO LOOP
    ...
```

イベントホールドは、予期可能なイベント処理をコーディングする際には、使い勝手の良い手法です。しかし、予期できないイベントの場合は、ループ処理によるプログラム中断を行う必要がない割り込みイベントによる手法が有用となります。例えば、下記の例では、F1キーが押されると割り込みが発生し、いつでもKey_F1サブルーチンに処理が移ります。

```
ON KEY(1) GOSUB Key_F1
...
Key_F1:
    ...
RETURN
```

5.5.1. 割り込みイベント

OFF ALL

適用機種	ALL
目的	全ての割り込みイベントを無効にします。
構文	OFF ALL
解説	全ての割り込みイベントを無効にします。 再度、割り込みイベントを有効にしたい場合は、該当のON xxx GOSUB コマンドを実行します。
使用例	<pre>ON READER(1) GOSUB BcrData_1 ON KEY(1) GOSUB KeyData_1 ... IF BACKUP_BATTERY < BATTERY_LOW% THEN OFF ALL BEEP(2000, 30) CLS PRINT "Backup Battery LOW!!" Loop: GOTO Loop END IF</pre>

OFF COM

適用機種	ALL								
目的	COM 割り込みバートを終了します。								
構文	OFF COM (n%)								
解説	<p>指定 COM ポートの COM 割り込みバートを無効にします。</p> <p>再度、COM 割り込みバートを有効にしたい場合は、ON COM GOSUB コマンドを実行します。</p> <p>引数 n%には、COM ポート番号を指定します。各ターミナルで有効な COM ポート番号は、下記の通りです。</p> <table><thead><tr><th>ターミナル型式</th><th>COM ポート番号</th></tr></thead><tbody><tr><td>8000/8300 シリーズ</td><td>1~2</td></tr><tr><td>8200/8400/8700 シリーズ</td><td>1~2, 5</td></tr><tr><td>8500 シリーズ</td><td>1~4</td></tr></tbody></table>	ターミナル型式	COM ポート番号	8000/8300 シリーズ	1~2	8200/8400/8700 シリーズ	1~2, 5	8500 シリーズ	1~4
ターミナル型式	COM ポート番号								
8000/8300 シリーズ	1~2								
8200/8400/8700 シリーズ	1~2, 5								
8500 シリーズ	1~4								
使用例	<pre>ON COM(1) GOSUB HostCommand ... HostCommand_1: OFF COM(1) 、データ処理中は COM 割り込みバートを無効に設定 ... ON COM(1) GOSUB HostCommand RETURN</pre>								

OFF ESC

適用機種	ALL
目的	ESC 割り込みバートを無効にします。
構文	OFF ESC
解説	<p>ESC 割り込みバートを無効にします。</p> <p>再度、ESC 割り込みバートを有効にしたい場合は、ON ESC GOSUB コマンドを実行します。</p>
使用例	<pre>ON ESC GOSUB Key_Esc ... Key_Esc: OFF ESC ... ON ESC GOSUB Key_Esc RETURN</pre>

OFF HOUR_SHARP

適用機種	ALL
目的	HOUR_SHARP 割り込みバートを無効にします。
構文	OFF HOUR_SHARP
解説	<p>HOUR_SHARP 割り込みバートを無効にします。</p> <p>再度、HOUR_SHARP 割り込みバートを有効にしたい場合は、ON HOUR_SHARP GOSUB コマンドを実行します。</p>
使用例	OFF HOUR_SHARP

OFF KEY

適用機種	8500
目的	ファンクション-割り込みイベントを無効にします。
構文	OFF KEY (<i>n</i> %)
解説	<p>ファンクション-割り込みイベントを無効にします。</p> <p>再度、ファンクション-割り込みイベントを有効にしたい場合は、ON KEY GOSUB コマンドを実行します。引数 <i>n</i>%には、ファンクション-番号を 1~12 の範囲で指定します。</p>
使用例	<pre>ON KEY(1) GOSUB On_Shift ON KEY(2) GOSUB Off_Shift ... On_Shift: OFF KEY(1) Mode\$ = "IN" GOSUB Progress ON KEY(1) GOSUB On_Shift RETURN ...</pre>

OFF KEY

適用機種	8000/8200/8300/8400/8700
目的	キー割り込みイベントを無効にします。
構文	OFF KEY (<i>n</i> %) OFF KEY (256 + <i>keycode</i> %)
解説	<p>キー割り込みイベントを無効にします。</p> <p>再度、キー割り込みイベントを有効にしたい場合は、ON KEY GOSUB コマンドを実行します。引数 <i>n</i>%には、ファンクション-番号を 1~12 の範囲で指定します。これらは、それぞれファンクション- F1~F12 に対応します。その他のキーを指定する場合は、もう 1 つの構文を利用し、引数 <i>keycode</i>%に、キーコードを指定します。キーコードは、「補足 5. キーコードテーブル」を参照ください。</p>
使用例	<pre>ON KEY(1) GOSUB Key1Event ` F1 キー割り込みイベント ON KEY(256 + 144) GOSUB Key13Event ` F13 (キーコード 144) キー割り込みイベント ... Key1Event: OFF KEY(1) PRINT "KEY1 is pressed" ON KEY(1) GOSUB Key1Event RETURN Key13Event: OFF KEY(256 + 144) PRINT "KEY13 is pressed" ON KEY(256 + 144) GOSUB Key1Event RETURN ...</pre>

OFF MINUTE_SHARP

適用機種	ALL
目的	MINUTE_SHARP 割り込みイベントを無効にします。
構文	OFF MINUTE_SHARP
解説	MINUTE_SHARP 割り込みイベントを無効にします。 再度、MINUTE_SHARP 割り込みイベントを有効にしたい場合は、ON MINUTE_SHARP GOSUB ｺマﾝﾄを実行します。
使用例	OFF MINUTE_SHARP

OFF READER

適用機種	ALL
目的	リーダ 割り込みイベントを無効にします。
構文	OFF READER (<i>n</i> %)
解説	リーダ 割り込みイベントを無効にします。 再度、リーダ 割り込みイベントを有効にしたい場合は、ON READER GOSUB ｺマﾝﾄ を実行します。 引数 <i>n</i> %には、リーダ ポート番号を指定します。通常は、1 です。
使用例	<pre>ON READER(1) GOSUB BcrData1 ... BcrData1: OFF READER(1) BEEP(2000, 5) Data\$ = GET_READER_DATA\$(1) CLS PRINT Data\$...</pre>

OFF TCPIP

適用機種	ALL
目的	TCP/IP 割り込みイベントを無効にします。
構文	OFF TCPIP
解説	TCP/IP 割り込みイベントを無効にします。 再度、TCP/IP 割り込みイベントを有効にしたい場合は、ON TCPIP GOSUB ｺマﾝﾄ を実行します。
使用例	OFF TCPIP

OFF TIMER

適用機種	ALL
目的	タイマ-割り込みイベントを無効にします。
構文	OFF TIMER (<i>n</i> %)
解説	タイマ-割り込みイベントを無効にします。 再度、タイマ-割り込みイベントを有効にしたい場合は、ON TIMER GOSUB コマンドを実行します。 引数 <i>n</i> %には、タイマ- ID を 1~5 の範囲で指定します。
使用例	<pre>ON TIMER(1, 200) GOSUB ClearScreen ... ClearScreen: OFF TIMER(1) CLS ... RETURN</pre>

OFF TOUCH SCREEN

適用機種	8500/8700
目的	タッチスクリーン割り込みイベントを無効にします。
構文	OFF TOUCHSCREEN
解説	タッチスクリーン割り込みイベントを無効にします。 再度、タッチスクリーン割り込みイベントを有効にしたい場合は、ON TOUCHSCREEN GOSUB コマンドを実行します。
使用例	OFF TOUCHSCREEN

ON COM ... GOSUB ...

適用機種	ALL
目的	COM 割り込みイベントを有効にします。
構文	ON COM (<i>n</i> %) GOSUB <i>SubName</i>
解説	引数 <i>n</i> %には、COM ポート番号を指定します。 引数 <i>SubName</i> には、処理を移したいサブルーチン名を指定します。
使用例	<pre>ON COM(1) GOSUB HostCommand ... HostCommand: OFF COM(1) ... ON COM(1) GOSUB HostCommand RETURN</pre>

ON ESC GOSUB ...

適用機種	ALL
目的	ESCキー割り込みイベントを有効にします。
構文	ON ESC GOSUB <i>SubName</i>
解説	引数 <i>SubName</i> には、処理を移したいサブルーチン名を指定します。
使用例	<pre>ON ESC GOSUB ESC_Key ... ESC_Key: OFF ESC ... ON ESC GOSUB ESC_Key RETURN</pre>

ON HOUR_SHARP GOSUB ...

適用機種	ALL
目的	HOUR_SHARP 割り込みイベントを有効にします。
構文	ON HOUR_SHARP GOSUB <i>SubName</i>
解説	引数 <i>SubName</i> には、処理を移したいサブルーチン名を指定します。 この割り込みイベントは、タミカル内蔵のシステムクロックの時刻(時)が変わると発生します。
使用例	<pre>ON HOUR_SHARP GOSUB OnHourAlarm ... OnHourAlarm: CurrentTime\$ = TIME\$ Hour% = VAL(LEFT\$(CurrentTime\$, 2)) FOR I = TO Hour% BEEP(800, 10, 0, 10) WAIT(100) NEXT I ON HOUR_SHARP GOSUB OnHourAlarm RETURN</pre>

ON KEY ... GOSUB ...

適用機種	8500
目的	ファンクションキー割り込みイベントを有効にします。
構文	ON KEY(<i>n</i> %) GOSUB <i>SubName</i>
解説	引数 <i>n</i> % には、ファンクションキー番号を 1~12 の範囲で指定します。 引数 <i>SubName</i> には、処理を移したいサブルーチン名を指定します。 指定したファンクションキーが押下されると、サブルーチンをコールします。
使用例	<pre>ON KEY(1) GOSUB On_Shift ON KEY(2) GOSUB Off_Shift ... On_Shift: Mode\$ = "IN" RETURN Off_Shift: Mode\$ = "OUT" RETURN</pre>

ON KEY ... GOSUB ...

適用機種	8000/8200/8300/8400/8700
目的	キー割り込みイベントを有効にします。
構文	<pre>ON KEY(<i>n</i>%) GOSUB <i>SubName</i> ON KEY(256 + <i>n</i>%) GOSUB <i>SubName</i></pre>
解説	<p>引数 <i>n</i>%には、ファンクションキー番号を 1~12 の範囲で指定します。これらは、それぞれファンクションキー F1~F12 に対応します。その他のキーを指定する場合は、もう 1 つの構文を利用し、引数 <i>keycode</i>%に、キーコードを指定します。キーコードは、「補足 5. キーコードテーブル」を参照ください。</p> <p>引数 <i>SubName</i> には、処理を移したいサブルーチン名を指定します。</p> <p>指定したキーが押下されると、サブルーチンをコールします。</p>
使用例	<pre>ON KEY(1) GOSUB Key1Event ` F1 キー割り込みイベント ON KEY(256 + 144) GOSUB Key13Event ` F13 (キーコード 144) キー割り込みイベント ... Key1Event: OFF KEY(1) PRINT "KEY1 is pressed" ON KEY(1) GOSUB Key1Event RETURN Key13Event: OFF KEY(256 + 144) PRINT "KEY13 is pressed" ON KEY(256 + 144) GOSUB Key1Event RETURN ...</pre>

ON MINUTE_SHARP GOSUB ...

適用機種	ALL
目的	MINUTE_SHARP 割り込みイベントを有効にします。
構文	<pre>ON MINUTE_SHARP GOSUB <i>SubName</i></pre>
解説	<p>引数 <i>SubName</i> には、処理を移したいサブルーチン名を指定します。</p> <p>この割り込みイベントは、タミカ内蔵のシステムクロックの時刻 (分) が変わると発生します。</p>
使用例	<pre>ON MINUTE_SHARP GOSUB CheckTime ... CheckTime: CurrentTime\$ = TIME\$ Minute% = VAL(MID\$(CurrentTime\$, 3, 2)) IF Minute% = 30 THEN GOSUB HalfHourAlarm RETURN ... HalfHourAlarm: BEEP(800, 30) WAIT(100) RETURN</pre>

ON POWER_ON GOSUB ...

適用機種	ALL
目的	POWER_ON 割り込みイベントを有効にします。
構文	ON POWER_ON GOSUB <i>SubName</i>
解説	引数 <i>SubName</i> には、処理を移したいサブルーチン名を指定します。 この割り込みイベントは、ターミナルの電源を切した後、再度電源キーが押されると発生します。
使用例	<pre>ON POWER_ON GOSUB RESUME_ON ... Main1: ... PWR_INDEX1& = PWR_INDEX& PRINT "[POWER ON]", PWR_INDEX1& RETURN Main2: IF PWR_INDEX& > PWR_INDEX1& THEN GOTO Main1 END IF ... GOTO MAIN2 Resume_On: PWR_INDEX& = PWR_INDEX& + 1 WAIT(100) RETURN</pre>


ON READER GOSUB ...

適用機種	ALL
目的	リーダ割り込みイベントを有効にします。
構文	ON READER(<i>n</i> %) GOSUB <i>SubName</i>
解説	引数 <i>n</i> % には、リーダポート番号を指定します。通常、ターミナルの場合、1 となります。 引数 <i>SubName</i> には、処理を移したいサブルーチン名を指定します。 この割り込みイベントは、リーダポートでデータを受信すると発生します。
使用例	<pre>ON READER(1) GOSUB BcrData_1 ... BcrData_1: OFF READER(1) BEEP(2000, 5) Data\$ = GET_READER_DATA\$(1) ...</pre>

ON TCPIP GOSUB ...

適用機種	ALL
目的	TCP/IP 割り込みイベントを有効にします。
構文	ON TCPIP GOSUB <i>SubName</i>
解説	引数 <i>SubName</i> には、処理を移したいサブルーチ名を指定します。 この割り込みイベントは、TCP/IP 接続でデータを受信した場合、又は、エラーが発生した場合に発生します。
使用例	<pre>ON TCPIP GOSUB TCPIP_Trigger ... TCPIP_Trigger: MSG% = GET_TCPIP_MESSAGE ...</pre>

参考

 TCP/IP 接続ステータスの確認には、GET_TCPIP_MESSAGE イベントを使用します。

ON TIMER ... GOSUB ...

適用機種	ALL
目的	タイマ割り込みイベントを有効にします。
構文	ON TIMER(<i>n%</i> , <i>durtation%</i>) GOSUB <i>SubName</i>
解説	引数 <i>n%</i> には、使用するタイマ番号を指定します。タイマは、1~5 の 5 つが用意されています。 引数 <i>durtation%</i> には、タイマ割り込みイベントを発生させる間隔を 10 ミリ秒単位で指定します。 引数 <i>SubName</i> には、処理を移したいサブルーチ名を指定します。
使用例	<pre>ON TIMER(1, 200) GOSUB ClearScreen ` Timer(1) 2 秒 ... ClearScreen: OFF TIMER(1) CLS RETURN</pre>

ON TOUCHSCREEN ... GOSUB ...

適用機種	8500/8700
目的	タッチスクリーン割り込みイベントを有効にします。
構文	ON TOUCHSCREEN GOSUB <i>SubName</i>
解説	引数 <i>SubName</i> には、処理を移したいサブルーチ名を指定します。 この割り込みイベントは、タッチスクリーンで有効で、タッチスクリーンがタップされた時に発生します。
使用例	<pre>ON TOUCHSCREEN GOSUB Check_Fun ... Check_Fun: NO% = GET_SCREENITEM RETURN</pre>

5.5.2. 割り込みバートロックとアンロック

割り込みバートは、補して有効にすることが可能です。新しい割り込みバートが発生すると、通常、最新のバートが優先処理されることになります。

割り込みバート処理中に、他の割り込みバートを抑止したい場合に、ここで紹介する LOCK/UNLOCK コマンドを利用します。

LOCK

適用機種	ALL
目的	有効な全ての割り込みバートを UNLOCK コマンド が実行されるまで抑止します。
構文	LOCK
解説	有効な全ての割り込みバートを UNLOCK コマンド が実行されるまで抑止します。 下記の例では、リーダ 1 及びリーダ 2 からの受信データ処理中、LOCK コマンド で全ての割り込みバートを抑止し、処理終了後、UNLOCK コマンド で解除しています。
使用例	<pre>ON READER(1) GOSUB BcrData_1 ON REDAER(2) GOSUB BcrData_2 ... BcrData_1: LOCK BEEP(2000, 5) Data\$ = GET_READER_DATA\$(1) GOSUB AddNewData UNLOCK RETURN BcrData_2: LOCK BEEP(2000, 5) Data\$ = GET_READER_DATA\$(2) GOSUB AddNewData UNLOCK RETURN</pre>

UNLOCK

適用機種	ALL
目的	LOCK コマンド で抑止された全ての割り込みバートを解除します。
構文	UNLOCK
解説	LOCK コマンド で抑止された全ての割り込みバートを解除します。

5.6. システムコマンド

AUTO_OFF

適用機種	ALL
目的	オートオフ時間を設定します。
構文	AUTO_OFF (n%)
解説	引数 n%には、自動的に電源を切にする時間を 1 秒単位で指定します。 オートオフ時間で設定された時間、ターミナルの操作が行われないと、システムは自動的にターミナルの電源を切にします。オートオフ機能を無効にしたい場合は、オートオフ時間を 0 に設定します。
使用例	AUTO_OFF (30) 、 オートオフ時間 30 秒 AUTO_OFF (0) 、 オートオフ機能 無効
関連項目	POWER_ON, RESTART

CHANGE_SPEED


適用機種	8000/8300
目的	CPU スピードを設定します。
構文	CHANGE_SPEED (n%)
解説	引数 n%には、CPU スピードを 1~5 の範囲で指定します。

n%	意味
1	1/16 スピード
2	1/8 スピード
3	1/4 スピード
4	1/2 スピード
5	フルスピード

ターミナルが入力待ちなど負荷の少ない処理を実行している際に、CPU スピードを落とすことで消費電力を抑制することができます。

使用例	CHANGE_SPEED (3)
-----	------------------

参考

 WAIT コマンドの方が消費電力を抑制することができます。

IOPIN_STATUS

適用機種	8000/8200/8300/8400/8700
目的	I/Oピンのステータスを取得します。
構文	REM Result は結果を代入するための整数型変数です。 Result = ABS (n%)
解説	引数 n%には、取得したいステータスに対応する値を指定します。

n%	意味																													
0	IrDA 接続ステータスを取得します。IRDA_STATUS (0) マクロと同じです。																													
	<table><tr><th>戻り値</th><th>意味</th></tr><tr><td>0</td><td>IrDA 接続が無効</td></tr><tr><td>1</td><td>IrDA 接続が有効</td></tr></table>	戻り値	意味	0	IrDA 接続が無効	1	IrDA 接続が有効																							
	戻り値	意味																												
	0	IrDA 接続が無効																												
1	IrDA 接続が有効																													
8200/8400/8700 では、常に 1 が返されます。																														
1	データ転送が成功したかどうかを取得します。戻り値として、データを含むデータ桁数が返されます。																													
2	8200/8400/8700 でのみ使用可能 ターミナルがクレードル/ケーブル/AC アダプタに接続されているかを取得します。戻り値の各ビットの意味は、下記の表を参照ください。																													
	<table><tr><th>ビット</th><th>値</th><th>略称/意味</th></tr><tr><td rowspan="5">0~3</td><td>0x00</td><td>NO CRADLE/クレードルに接続されていません</td></tr><tr><td>0x01</td><td>MODEM CRADLE/モデムクレードルに接続されています</td></tr><tr><td>0x02</td><td>ETHERNET CRADLE/イーサネットクレードルに接続されています</td></tr><tr><td>0x03</td><td>GPRS CRADLE/GPRS・GSM クレードルに接続されています</td></tr><tr><td>0x04</td><td>CHARGER CRADLE/充電機能付通信クレードルに接続されています</td></tr><tr><td rowspan="2">4</td><td>0x00</td><td>RS232 CABLE DISCONNECTED/RS232 ケーブルが接続されていません</td></tr><tr><td>0x10</td><td>RS232 CABLE CONNECTED/RS232 ケーブルが接続されています</td></tr><tr><td rowspan="2"></td><td>0 00</td><td>USB CABLE DISCONNECTED/USB ケーブルが接続されていません</td></tr><tr><td>0x20</td><td>USB CABLE CONNECTED/USB ケーブルが接続されています</td></tr><tr><td rowspan="2">6</td><td>0x00</td><td>ADAPTER DISCONNECTED/AC アダプタ (DC5V) が接続されていません</td></tr><tr><td>0x40</td><td>ADAPTER CONNECTED/AC アダプタ (DC5V) が接続されています</td></tr></table>	ビット	値	略称/意味	0~3	0x00	NO CRADLE/クレードルに接続されていません	0x01	MODEM CRADLE/モデムクレードルに接続されています	0x02	ETHERNET CRADLE/イーサネットクレードルに接続されています	0x03	GPRS CRADLE/GPRS・GSM クレードルに接続されています	0x04	CHARGER CRADLE/充電機能付通信クレードルに接続されています	4	0x00	RS232 CABLE DISCONNECTED/RS232 ケーブルが接続されていません	0x10	RS232 CABLE CONNECTED/RS232 ケーブルが接続されています		0 00	USB CABLE DISCONNECTED/USB ケーブルが接続されていません	0x20	USB CABLE CONNECTED/USB ケーブルが接続されています	6	0x00	ADAPTER DISCONNECTED/AC アダプタ (DC5V) が接続されていません	0x40	ADAPTER CONNECTED/AC アダプタ (DC5V) が接続されています
	ビット	値	略称/意味																											
	0~3	0x00	NO CRADLE/クレードルに接続されていません																											
		0x01	MODEM CRADLE/モデムクレードルに接続されています																											
		0x02	ETHERNET CRADLE/イーサネットクレードルに接続されています																											
		0x03	GPRS CRADLE/GPRS・GSM クレードルに接続されています																											
		0x04	CHARGER CRADLE/充電機能付通信クレードルに接続されています																											
	4	0x00	RS232 CABLE DISCONNECTED/RS232 ケーブルが接続されていません																											
		0x10	RS232 CABLE CONNECTED/RS232 ケーブルが接続されています																											
		0 00	USB CABLE DISCONNECTED/USB ケーブルが接続されていません																											
		0x20	USB CABLE CONNECTED/USB ケーブルが接続されています																											
	6	0x00	ADAPTER DISCONNECTED/AC アダプタ (DC5V) が接続されていません																											
		0x40	ADAPTER CONNECTED/AC アダプタ (DC5V) が接続されています																											
3	8200/8400/8700 でのみ使用可能 ターミナルが大容量記憶装置デバイスとして動作しているかを取得します。戻り値の意味は、下記の表を参照ください。																													
	<table><tr><th>戻り値</th><th>略称/意味</th></tr><tr><td>0</td><td>USB 接続されていません</td></tr><tr><td>1</td><td>USB 接続されていますが、大容量記憶装置デバイスにはアクセスされていません</td></tr><tr><td>3</td><td>USB 接続されており、大容量記憶装置デバイスにはアクセス中です</td></tr></table>	戻り値	略称/意味	0	USB 接続されていません	1	USB 接続されていますが、大容量記憶装置デバイスにはアクセスされていません	3	USB 接続されており、大容量記憶装置デバイスにはアクセス中です																					
	戻り値	略称/意味																												
	0	USB 接続されていません																												
	1	USB 接続されていますが、大容量記憶装置デバイスにはアクセスされていません																												
3	USB 接続されており、大容量記憶装置デバイスにはアクセス中です																													
4	8000/8300 でのみ使用可能 ターミナルの充電ステータスを取得します。戻り値の意味は、下記の表を参照ください。																													
	<table><tr><th>戻り値</th><th>略称/意味</th></tr><tr><td>0</td><td>電源に接続されていません</td></tr><tr><td>1</td><td>充電中です</td></tr><tr><td>2</td><td>充電完了です</td></tr><tr><td>3</td><td>充電エラーが発生しています</td></tr></table>	戻り値	略称/意味	0	電源に接続されていません	1	充電中です	2	充電完了です	3	充電エラーが発生しています																			
	戻り値	略称/意味																												
	0	電源に接続されていません																												
	1	充電中です																												
	2	充電完了です																												
3	充電エラーが発生しています																													

使用例

```
U% = IOPIN_STATUS(2)

、 *** クレドールを検出 ***
V% = BIT_OPERATOR(1, U%, 15)
、 ビット 0~3 をチェック
IF V% = 2 THEN          、 イーサネットクレドール
    PRINT "Seated in Ethernet Cradle"
ENDIF

、 USB ケーブル接続を検出
V% = BIT_OPERATOR(1, U%, 32)
、 ビット 5 をチェック
IF V% = 32 THEN          、 32 = 0x20
    PRINT "USB Cable connected"
ENDIF
```

MENU

適用機種

ALL

目的

メニューを表示します。

構文

REM Result%は結果を代入するための整数型変数です。
Result% = MENU(Item\$)

解説

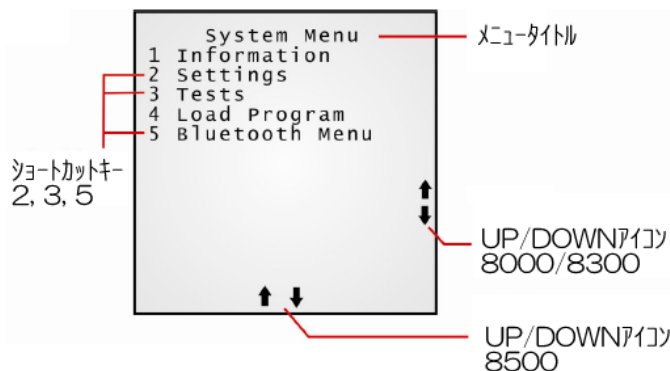
引数 Item\$には、メニュー表示したいメニュー項目(文字列)を CR(0x0d) で区切って指定します。
メニューの選択は、↑/↓キーで行い、Enter キーで決定します。ESC キーを押すと、メニューをキャンセルします。
また、下記の規則に従って、メニュータイトルやショートカットキーを定義することが可能です。

- & &に続く 1 文字がショートカットキーになります。
- @ @に続く文字列がメニュータイトルになります。メニュータイトルは、何番目に定義しても問題ありません。

戻り値は、メニュー項目番号で、ESC キーが押された場合は、0 が返されます。

使用例

```
MENU_STR$ = "1 Information" + CHR$(13)
MENU_STR$ = MENU_STR$ + "@System Menu" + CHR$(13)
MENU_STR$ = MENU_STR$ + "&2 Settings" + CHR$(13)
MENU_STR$ = MENU_STR$ + "&3 Tests" + CHR$(13)
MENU_STR$ = MENU_STR$ + "4 Load Program" + CHR$(13)
MENU_STR$ = MENU_STR$ + "&5 Bluetooth Menu" + CHR$(13)
...
S% = MENU(MENU_STR$)
```



参考

- メニューは、最大 32 項目まで定義できます。
- 1 行に表示可能なメニュー項目のバイト数は、下記の通りです(システムフォント使用時)。これを超える桁数を定義した場合、オーバーした文字列は次の行に表示されます。

ターミナル	バイト数
8000	16 バイト
8300	20 バイト
8200/8400/8500/8700	26 バイト

POWER_ON

適用機種	ALL
目的	リブート機能を設定します。
構文	POWER_ON (n%)
解説	引数 n%には、0 (リブート機能 無効) 又は 1 (リブート機能 有効/プログラムリスタート) を指定します。
使用例	POWER_ON (0) 、リブート機能 無効

RESTART

適用機種	ALL
目的	システムを再起動します。
構文	RESTART
解説	BASIC プログラムを終了し、システムを再起動します。
使用例	<pre>HostCommand = READ_COM\$(1) ... IF HostCommand\$ = "RESTART" THEN RESTART ELSE ...</pre>
関連項目	AUTO_OFF, POWER_ON

DEVICE_ID\$

適用機種	ALL
目的	ターミナルのシリアル番号を取得します。
構文	REM Result\$は結果を代入するための文字列型変数です。 Result\$ = DEVICE_ID\$
解説	ターミナルのシリアル番号を戻り値として返します。
使用例	PRINT "S/N:", DEVICE_ID\$
関連項目	SYSTEM_INFORMATION\$

GET_TARGET_MACHINE\$

適用機種	ALL
目的	ターミナルのモデル番号を取得します。
構文	REM Result\$は結果を代入するための文字列型変数です。 Result\$ = GET_TARGET_MACHINE\$
解説	ターミナルのモデル番号を戻り値として返します。
使用例	<pre>A\$ = GET_TARGET_MACHINE\$ IF A\$ = "8000" THEN PRINT "8000 series" ELSE IF A\$ = "8200" THEN PRINT "8200 series" ELSE ...</pre>

SYSTEM_INFORMATION\$

- 適用機種 ALL
- 目的 ターミナルのハードウェア/ソフトウェアに関する情報を取得します。
- 構文 REM Result\$は結果を代入するための文字列型変数です。
Result\$ = SYSTEM_INFORMATION\$(index%)
- 解説 引数 index%には、取得したい情報のインデックス番号を指定します。インデックス番号は下記の表を参照ください。

index%	意味
1	C ライブ ラリバ -ゾ ョソ
2	BASIC ランタイムバ -ゾ ョソ
3	カーネルバ -ゾ ョソ
4	ハードウェアバ -ゾ ョソ
5	製造日
6	シリアル番号
7	オリジナルシリアル番号
8	デバイスタイプ (次頁の表を参照)
9	RFID バ -ゾ ョソ
10	ブザー音量 8200 : Mute, Low, Medium, High 8400 : Low, Medium, High
11	USB 充電電流 : 500mA, 100mA
12	ブートローダーバ -ゾ ョソ (8200)
21	GPS ステータス
22	GPS スピード (Km/h)
23	GPS 緯度 (ddmm.mmmmmN 又は ddmm.mmmmmS)
24	GPS 経度 (dddmm.mmmmmE 又は dddmm.mmmmmW)
25	GPS SNR (Signal to Noise Ratio, 平均 dB)
26	GPS 衛星数 (見つかった衛星の数)
27	GPS 高度 (m)

- 使用例 LIBVER\$ = SYSTEM_INFORMATION\$(1)
PRINT "Library ver :", LIBVER\$

参考

-  USB 充電電流は、システムメニューからのみ変更可能です。
-  GPS スピード / 緯度 / 経度 / 高度は、GPS ステータスが 1 になるまで更新されません。

デバイスタイプ

SYSTEM_INFORMATION\$(8)を実行すると、デバイスタイプが“xxxx”フォーマットで返されます。

x	x	x	x
リーダ モジュール	無線モジュール	その他 8000:電池タイプ 8300/8500/8700 RFID モジュール	予備

8000 シリーズ		
デバイスタイプ 4桁	意味	
1桁目	0xxx	リーダ モジュール無し
	1xxx	CCD スキャナモジュール
	2xxx	レーザ スキャナモジュール
2桁目	x0xx	無線モジュール無し
	x4xx	802.11b/g モジュール
	x5xx	Bluetooth モジュール
	x6xx	音響コデックモジュール
3桁目	xx0x	アルカリ乾電池
	xx1x	リチウムイオン充電電池パック
4桁目	xxx0	予備

8200 シリーズ		
デバイスタイプ 4桁	意味	
1桁目	0xxx	リーダ モジュール無し
	1xxx	CCD スキャナモジュール
	2xxx	レーザ スキャナモジュール
	3xxx	2次元リーダ モジュール
2桁目	x0xx	N/A
	x5xx	Bluetooth モジュール
	x8xx	802.11b/g モジュール + Bluetooth モジュール
	x6xx	音響コデックモジュール
3桁目	xx0x	予備
4桁目	xxx0	予備

8300 シリーズ		
デバイスタイプ 4桁	意味	
1桁目	0xxx	リーダ モジュール無し
	1xxx	CCD スキャナモジュール
	2xxx	レーザ スキャナモジュール
	4xxx	ロングレンジレーザ スキャナモジュール
2桁目	x0xx	無線モジュール無し
	x1xx	433MHz 無線モジュール
	x2xx	2.4GHz 無線モジュール
	x4xx	802.11b/g モジュール
	x5xx	Bluetooth モジュール
	x6xx	音響コデックモジュール
	x8xx	802.11b/g モジュール + Bluetooth モジュール
3桁目	xx0x	RFID モジュール無し
	xx1x	RFID モジュール
4桁目	xxx0	予備

8500 シリーズ		
デバイスタイプ 4桁		意味
1桁目	0xxx	リーダ モジュール無し
	1xxx	CCD スキャナ モジュール
	2xxx	レーザ スキャナ モジュール
	3xxx	2次元リーダ モジュール
	4xxx	ロングレンジ レーザ スキャナ モジュール
	5xxx	エクストラロングレンジ レーザ スキャナ モジュール
2桁目	x0xx	N/A
	x1xx	433MHz 無線 モジュール
	x4xx	802.11b/g モジュール + Bluetooth モジュール
	x5xx	Bluetooth モジュール
	x8xx	802.11b/g モジュール + Bluetooth モジュール
3桁目	xx0x	RFID モジュール無し
	xx1x	RFID モジュール
4桁目	xxx0	予備

8700 シリーズ		
デバイスタイプ 4桁		意味
1桁目	0xxx	リーダ モジュール無し
	1xxx	CCD スキャナ モジュール
	2xxx	レーザ スキャナ モジュール
	3xxx	2次元リーダ モジュール
	4xxx	ロングレンジ レーザ スキャナ モジュール
2桁目	x0xx	N/A
	x3xx	3.5G + Bluetooth モジュール
	x4xx	802.11b/g モジュール + Bluetooth モジュール
	x5xx	Bluetooth モジュール
	x7xx	3.5G + 802.11b/g モジュール + Bluetooth モジュール
3桁目	xx0x	RFID モジュール無し
	xx1x	RFID モジュール
	xx2x	GPS モジュール
4桁目	xxx0	予備

VERSION

適用機種	ALL
目的	システムにバージョン情報を書き込みます。
構文	VERSION (a\$)
解説	引数 a\$には、書き込みたいバージョン情報を指定します。
使用例	VERSION("Cipher Basic 2.0")

SYSTEM_PASSWORD

適用機種	ALL
目的	システムメニューに入るためのパスワードを設定します。
構文	SYSTEM_PASSWORD (a\$)
解説	引数 a\$には、設定したいパスワードを指定します。
使用例	SYSTEM_PASSWORD("12345")

DOWNLOAD_BASIC

適用機種	ALL
目的	指定の通信ポートから BASIC プログラムを受信し、指定のファイル番号に格納します。
構文	REM Result%は結果を代入するための整数型変数です。 Result% = DOWNLOAD_BASIC\$(file%, port%)
解説	引数 file%には、BASIC プログラムを格納するファイル番号を指定します。下記の表を参照ください。

file%	意味
1 ~ 6	BASIC プログラムを格納するファイル番号 (トランザクションファイル)
18	SRAM に格納 (UPDATE_BASIC (18) コマンドと併用) 現状、8000/8200/8300/8400/8700 ターミナルで使用可能です。

引数 port%には、通信ポート番号を指定します。

port%	意味
1	高速 IR, IrDA, RS232
2	Bluetooth
5	USB バイナル COM (8200/8400/8700)

ダウンロード処理実行後、結果を戻り値として返します。

戻り値	意味
0	正常終了
-1	I/O : 無効なファイル番号
-2	I/O : 無効なポート番号
-3	I/O : 通信ポート応答なし
-4	I/O : BASIC プログラムヘッダの読み取りに失敗
-5	I/O : BASIC プログラムヘッダファイル(.ini)の読み取りに失敗
-6	I/O : BASIC プログラムボディファイル(.syn)の読み取りに失敗
-7	I/O : 書き込みI/O, SRAM に十分な空き容量がありません

使用例 Error_Code% = DOWNLOAD_BASIC(6, 1)

参考

- トランザクションファイルは、予め EMPTY_TRANSACTION_EX コマンドでクリアしておく必要があります。
- 通信ポートは、予め SET_COM 及び SET_COM_TYPE コマンドでパラメータ設定を行っておく必要があります。

UPDATE_BASIC

適用機種	ALL
目的	指定のプログラムを更新します。
構文	REM Result%は結果を代入するための整数型変数です。 Result% = UPDATE_BASIC\$(file%)
解説	引数 file%には、プログラムが格納されているファイル番号を指定します。 下記の表を参照ください。

file%	意味
1 ~ 6	アップローションプログラムが格納されているファイル番号 (トランザクションファイル)
18	FTP 経由又は DOWNLOAD_BASIC (18) で SRAM に格納したアップローションプログラム (.tkn) ✓ プログラムファイルは、処理完了後、削除されます。 現状、8000/8200/8300/8400/8700 ターミナルで使用可能です。
19	FTP 経由で SRAM に格納したランタイムプログラム (.bin) ✓ プログラムファイルは、処理完了後、削除されます。但し、ファイルシステムは、そのまま保持されます。 現状、8000/8200/8300/8400/8700 ターミナルで使用可能です。
20 ~ 39	SD カード に格納したアップローションプログラム (.tkn, .syn, .ini) ✓ .tkn ファイルが最優先順位です。 ✓ 処理終了後もプログラムファイルは、そのまま保持されます。
40 ~ 59	SD カード に格納したランタイムプログラム (.bin, .shx) ✓ .bin ファイルが最優先順位です。 ✓ 処理終了後もプログラムファイル及びファイルシステムは、そのまま保持されます。

ファイル番号で、SD カード を指定する場合、下記の手順・規則に従う必要があります。

手順 1	ファイル番号を表すプリフィックスをファイル名に付加します。 例えば、 echotest.ini → 25echotest.ini echotest.syn → 25echotest.syn にリネームします。
手順 2	リネームしたファイルを SD カード の ¥Program フォルダの下に配置します。
手順 3	UPDATE_BASIC (25) コマンド をコールします。システムは、¥Program フォルダ内の 25 で始まるファイル名の検索を開始します。 ✓ SD カード 内にアップローションプログラム (25*.tkn) が見つかったら、それを最優先とし、アクティブプログラムとします。 ✓ 同じプリフィックスを持つ複数のファイルが存在した場合 (例えば、25x.bin と 25a.bin)、FAT システムにより優先順位が決定され、先に見つかったファイルが採用されます。

更新処理実行後、結果を戻り値として返します。

戻り値	意味
-1	I/O : 無効なファイル番号
-2	I/O : 無効なファイルフォーマット
-8	I/O : 書き込みI/O, フラッシュROMに十分な空き容量がありません
-9	I/O : プログラムヘッダファイル(.ini)読み込み失敗
-10 (*)	I/O : プログラムオブジェクトファイル(.syn)読み込み失敗
-11	I/O : RAMサイズに合いません
-12 (*)	I/O : 容量不足・アドレス違反・フラッシュメモリの消去不良になどの原因により、新しいプログラムの書き込み失敗
-13 (*)	I/O : 新しいプログラムをフラッシュROMに書き込み終了後、ヘッダファイルの書き込み失敗
-14	I/O : SDカード内にファイルがありません
-15	I/O : SDカードが読み取れません
-16	I/O : SDカード内のファイル名が64バイトを超えています。

(*) 但し、BASICプログラムが上書き更新された場合は、戻り値は返りません。

使用例

```
Error_Code% = UPDATE_BASIC(3)
```

5.7. バージョリーダ

ターミナルは、型式により各種バージョリーダオプションを搭載しています。各ターミナル型式で用意されているリダオプションは、下記の通りです。

リダオプション		8000	8200	8300	8400	8500	8700
1D	CCD リニアイメージャ	X	X	X	X	X	X
	レーザースキャナ	X	X	X	X	X	X
	ロングレンジレーザースキャナ (LR)			X		X	X
	エクストラロングレンジレーザースキャナ (XLR)					X	
2D	2次元エリアイメージャ		X		X	X	X

参考

- 文字列変数は 255 バイト迄のデータしか取り扱えないため、255 バイトを超える二次元コードデータを取得する場合は、データが無くなるまで繰り返し GET_READER_DATA\$() フังก์ションをコールする必要があります。
- Cipher BASIC の DBF ファイルの最大レコード長は 250 バイトです。この制限により、250 バイトを超える二次元コードデータを 1 レコードに保存することはできません。

DISABLE READER

適用機種	ALL
目的	指定のリダポートを無効にします。
構文	DISABLE READER (n%)
解説	引数 n%には、リダポート番号を指定します。通常は、1 です。
使用例	DISABLE READER (1)

ENABLE READER

適用機種	ALL
目的	指定のリダポートを有効にします。
構文	ENABLE READER (n%)
解説	引数 n%には、リダポート番号を指定します。通常は、1 です。
使用例	<pre> ENABLE READER (1) ON READER (1) GOSUB Bcr_1 ... Bcr_1: Data\$ = GET_READER_DATA\$ (1) RETURN </pre>

GET_READER_DATA\$

適用機種	ALL
目的	指定のリーダーポートからデータを取得します。
構文	REM Result\$は結果を代入するための文字列型変数です。 Result\$ = GET_READER_DATA\$(n%)
解説	引数 n%には、リーダーポート番号を指定します。通常は、1 です。
使用例	<pre>ENABLE READER(1) ON READER(1) GOSUB Bcr_1 ... Bcr_1: Data\$ = GET_READER_DATA\$(1) RETURN</pre>

READER_CONFIG

適用機種	8200/8300/8400/8500/8700
目的	リーダー設定を有効にします。
構文	READER_CONFIG
解説	READER_SETTING マクロに続いて、このマクロをコールするとリーダー設定が有効になります。このマクロは、下記に列挙する一部のリーダー搭載スキャンションを採用したターミナルにのみ対応しています。その他のターミナルでは、READER_SETTING マクロをコールするだけで、設定が有効になります。 <ol style="list-style-type: none">2次元コードリーダー搭載モデル(8200/8400/8500/8700)ロングレンジレーザースキャナ搭載モデル(8300/8500/8700)エクストラロングレンジレーザースキャナ搭載モデル(8500)
使用例	<pre>READER_SETTING(5, 0) READER_SETTING(132, 0) READER_CONFIG 、リーダー搭載スキャンションに設定を送信 ... ON READER(1) GOSUB Bcr_1 ... Bcr_1: Data\$ = GET_READER_DATA\$(1) RETURN</pre>

CODE_TYPE

適用機種 ALL

目的 バコードタイプを取得します。

構文 REM Result%は結果を代入するための整数型変数です。
Result% = CODE_TYPE

解説 直前に読み取ったコードのバコードタイプを取得します。下記のバコードタイプ表を参照ください。

バコードタイプ表 1 - CCDリニアイメージャ, レザースキャナ			
戻り値		バコードタイプ	対応スキャンエンジン
10進数	ASCII		
63	?	COOP 2/5	CCDリニアイメージャ, レザースキャナ
64	@	ISBT 128	CCDリニアイメージャ, レザースキャナ
65	A	コード 39	CCDリニアイメージャ, レザースキャナ
66	B	コード 32 (イタリアンフォーマット)	CCDリニアイメージャ, レザースキャナ
67	C	CIP 39 (ロシアフォーマット)	CCDリニアイメージャ, レザースキャナ
68	D	インターストリップ 2/5	CCDリニアイメージャ, レザースキャナ
69	E	インターリーブド 2/5	CCDリニアイメージャ, レザースキャナ
70	F	マトリクス 2/5	CCDリニアイメージャ, レザースキャナ
71	G	コードバ (NW7)	CCDリニアイメージャ, レザースキャナ
72	H	コード 93	CCDリニアイメージャ, レザースキャナ
73	I	コード 128	CCDリニアイメージャ, レザースキャナ
74	J	UPC-E0/UPC-E1	CCDリニアイメージャ, レザースキャナ
75	K	UPC-E プド 2	CCDリニアイメージャ, レザースキャナ
76	L	UPC-E プド 5	CCDリニアイメージャ, レザースキャナ
77	M	EAN/JAN-8	CCDリニアイメージャ, レザースキャナ
78	N	EAN/JAN-8 プド 2	CCDリニアイメージャ, レザースキャナ
79	O	EAN/JAN-8 プド 5	CCDリニアイメージャ, レザースキャナ
80	P	EAN/JAN-13, UPC-A	CCDリニアイメージャ, レザースキャナ
81	Q	EAN/JAN-13 プド 2	CCDリニアイメージャ, レザースキャナ
82	R	EAN/JAN-13 プド 5	CCDリニアイメージャ, レザースキャナ
83	S	MSI	CCDリニアイメージャ, レザースキャナ
84	T	PLESSEY	CCDリニアイメージャ, レザースキャナ
85	U	GS1-128 (EAN-128)	CCDリニアイメージャ, レザースキャナ
86	V	予約	---
87	W	予約	---
88	X	予約	---
89	Y	予約	---
90	Z	Telepen	CCDリニアイメージャ, レザースキャナ
91	[GS1 Databar	CCDリニアイメージャ, レザースキャナ
92	¥	予約	---
93]	予約	---

バコードタイプ表 2 - ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ			
戻り値		バコードタイプ	対応スキャンエンジン
10進数	ASCII		
64	@	ISBT 128	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ
65	A	コード 39	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ
66	B	コード 32 (イタリアンフォーマット)	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ
67	C	N/A	---
68	D	N/A	---
69	E	インターリーブド 2/5	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ
70	F	マトリクス 2/5	2次元エリアイメージャ
71	G	コードバ (NW7)	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ
72	H	コード 93	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ
73	I	コード 128	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ
74	J	UPC-E0	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ
75	K	UPC-E プド 2	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ
76	L	UPC-E プド 5	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ
77	M	EAN/JAN-8	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ
78	N	EAN/JAN-8 プド 2	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ
79	O	EAN/JAN-8 プド 5	ロングレザースキャナ, エクストラロングレザースキャナ, 2次元エリアイメージャ

80	P	EAN/JAN-13	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
81	Q	EAN/JAN-13 プト 2	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
82	R	EAN/JAN-13 プト 5	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
83	S	MSI	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
84	T	N/A	---
85	U	GS1-128 (EAN-128)	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
86	V	N/A	---
87	W	N/A	---
88	X	N/A	---
89	Y	N/A	---
90	Z	N/A	---
91	[GS1 Databar Ominidirectional	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
92	¥	GS1 Databar Limited	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
93]	GS1 Databar Expanded	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
94	^	UPC-A	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
95	_	UPC-A プト 2	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
96	`	UPC-A プト 5	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ ロングレーザースキャナ
97	a	UPC-E1	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ ロングレーザースキャナ
98	b	UPC-E1 プト 2	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ ロングレーザースキャナ
99	c	UPC-E1 プト 5	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ ロングレーザースキャナ
100	d	TLC-39 (TCIF Linked Code 39)	2次元エリアイメージャ
101	e	Trioptic (Code 39)	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
102	f	Bookland (EAN)	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
103	g]-ト 11	ロングレーザースキャナ, 2次元エリアイメージャ
104	h]-ト 39 フルアスキー	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
105	i	IATA	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
106	j	インダストリアル 2/5 (デイスクリット 2/5)	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ
107	k	PDF417	2次元エリアイメージャ
108	l	MacroPDF417	2次元エリアイメージャ
109	m	Data Matrix	2次元エリアイメージャ
110	n	Maxicode	2次元エリアイメージャ
111	o	QR]-ト	2次元エリアイメージャ
112	p	US Postnet	2次元エリアイメージャ
113	q	US Planet	2次元エリアイメージャ
114	r	イギリス郵便]-ト	2次元エリアイメージャ
115	s	日本郵便]-ト (カスターバ]-ト)	2次元エリアイメージャ
116	t	オーストラリア郵便]-ト	2次元エリアイメージャ
117	u	オランダ郵便]-ト	2次元エリアイメージャ
118	v	コンバット]-ト	2次元エリアイメージャ
119	w	Macro PDF417	2次元エリアイメージャ
120	x	Macro MicroPDF417	2次元エリアイメージャ
121	y	Chinese 2/5	2次元エリアイメージャ
122	z	Aztec	2次元エリアイメージャ
123	{	MicroQR]-ト	2次元エリアイメージャ
124		USPS 4CB/One Code/Intelligent Mail	2次元エリアイメージャ
125	}	UPU FICS Postal	2次元エリアイメージャ
126	~	Coupon Code	ロングレーザースキャナ, イクストラロングレーザースキャナ, 2次元エリアイメージャ

使用例

```

...
CheckCodeType:
IF CODE_TYPE = 65 THEN
    BcrType$ = "Code 39"
ELSE IF CODE_TYPE = 66 THEN
    BcrType$ = "Italian Pharmacode"
...
END IF
PRINT "Code Type:", BcrType$
RETURN

```

関連項目

GET_READER_SETTING, READER_SETTING

GET_READER_SETTING

適用機種	ALL
目的	リーダのパラメータ設定値を取得します。
構文	REM A%は結果を代入するための整数型変数です。 A% = GET_READER_SETTING(n%)
解説	引数 n%には、取得したいリーダ設定値に対応する配列値を指定します。 リーダの設定値が戻り値として返されます。
使用例	Setting1% = GET_READER_SETTING(1) IF Setting1% = 1 THEN PRINT "Code 39 is Enabled" ELSE PRINT "Code 39 is Disabled" END IF
関連項目	CODE_TYPE

READER_SETTING

適用機種	ALL
目的	リーダのパラメータ設定を行います。
構文	READER_SETTING(n1%, n2%)
解説	引数 n1%には、設定したいリーダ設定値に対応する配列値を指定します。 引数 n2%には、リーダ設定値を指定します。
使用例	READER_SETTING(1, 1) ` Code 39 読み取り有り
関連項目	CODE_TYPE, READER_CONFIG

参考

- リニアイメージャ/レーザスキャニングは、「補足 1. スキャ配列」の「スキャ配列表 1」を参照してください。
- エリアイメージャ(2次元コード)/エキストラロングレンジレーザスキャニングは、「補足 1. スキャパラメータ」の「スキャ配列表 2」を参照してください。
- エリアイメージャ(2次元コード)/エキストラロングレンジレーザスキャニングの場合は、READER_SETTING コマンドでリーダ設定を行った後、READER_CONFIG コマンドをコールしなければリーダ設定は有効になりません。

5.8. RFIDリーダ /ライタ

8300/8500/8700 ターミナルは、型式により RFID リーダ /ライタオプションを搭載しています。下記に RFID タグの対応表を示します。

タグタイプ	UID のみ	リードページ	ライトページ
タグ Mifare ISO1443A			
Mifare Standard 1K	✓	✓	✓
Mifare Standard 4K	✓	✓	✓
Mifare Ultralight	✓	✓	✓
Mifare DESFire	✓	---	---
Mifare S50	✓	✓	✓
SLE44R35	✓	---	
SLE66R35	✓	✓	✓
タグ SR176			
SRIX 4K	✓	✓	✓
SR176	✓	✓	✓
タグ ISO15693			
ICODE SLI	✓	✓	✓
SRF55V02P	✓	---	---
SRF55V02S	✓	---	---
SRF55V10P	✓	---	---
TI Tag-it HF-I	✓	✓	✓
タグ ICODE			
ICODE	✓	✓	✓

参考

- ➡ プログラミングを行う前に RFID タグの仕様・特性について熟知する必要があります。
- ➡ 上記の RFID タグ対応表は、RFID モジュールバージョン 1.0 における結果です。

仮想COMポート

RFID リーダ /ライタモジュールは、仮想 COM ポート (COM4) に割り付けられています。よって、下記のコマンドにより、制御を行います。

- ✓ OPEN_COM(4) : RFID リーダ /ライタモジュールとの通信ポートをオープンします。
- ✓ CLOSE_COM(4) : RFID リーダ /ライタモジュールとの通信ポートをクローズします。
- ✓ A\$ = READ_COM\$(4) : RFID タグからの読取データを取得します。
- ✓ WRITE_COM(4) : RFID タグにデータを書き込みます。
- ✓ ON COM(4) GOSUB ... : 通信ポート 4 の割り込みイベントを有効にします。
- ✓ OFF COM(4) : 通信ポート 4 の割り込みイベントを無効にします。

パラメータとデータフォーマット

RFID タグからのデータ読み取り及び書き込みの際は、下記のパラメータを引数として指定する必要があります。

TagType&

ビット 31~6	ビット 5	ビット 4	ビット 3	ビット 2	ビット 1	ビット 0
予約	ISO14443B	SRI176	ISO14443A	ICODE	TAGIT	ISO15693

Start%

読み取り及び書き込みを開始する開始アドレスを指定します。

MaxLen%

読み取り時 : 1~255 の範囲で読取桁数を指定します。0 を指定すると、UID のみを読み取ります。
書き込み時 : 予約 (意味を持たないため、どんな数値を指定しても影響はありません。)

読取データフォーマット

読取データは、下記のフォーマットで返されます。

バイト 1	バイト 2~18	バイト 19~xx
タグタイプ タグ V ISO15693 T TAGIT I ICODE M Mifare S SR176 Z ISO14443B	UID (S/N)	データ

SET_RFID_READ

適用機種	8300/8500
目的	RFID タグ 読み取りのためのパラメータを設定します。
構文	SET_RFID_READ (TagType&, Start%, MaxLen%)
解説	RFID タグ 読み取りのためのパラメータを設定します。パラメータに関しては、上記「パラメータとデータフォーマット」を参照ください。パラメータが正しく設定されるまで、RFID タグ の読み取りは行えません。
使用例	SET_RFID_READ (1, 0, 20) \ ISO 15693 タグ のアドレス 0 から 20 バイトを読み取り ... A\$ = READ_COM\$(4)
関連項目	CLOSE_COM, OPEN_COM, READ_COM\$, WRITE_COM

ENABLE READER

適用機種	8300/8500
目的	RFID タグ 書き込みのためのパラメータを設定します。
構文	SET_RFID_WRITE (TagType&, Start%, MaxLen%)
解説	RFID タグ 書き込みのためのパラメータを設定します。パラメータに関しては、上記「パラメータとデータフォーマット」を参照ください。パラメータが正しく設定されるまで、RFID タグ への書き込みは行えません。
使用例	SET_RFID_WRITE (63, 6, 32) \ 全ビットタグ に対して、アドレス 6 からデータを書き込み ... \ 63 = 0x3F WRITE_COM(4, Wdata\$)
関連項目	CLOSE_COM, OPEN_COM, READ_COM\$, WRITE_COM

GET_RFID_KEY

適用機種 8300/8500

目的 特定 RFID タグ のメモリリット-を取得します。

構文 REM Result\$はメモリリット-を代入するための文字列型変数です。
Result\$ = GET_RFID_KEY(TagType%)

解説 Mifare Standard 1K/4K や SLE66R35 のような特定 RFID タグ のメモリリット-を取得します。
引数 TagType%には、下記のタグ タイプ を示す値を指定します。

TagType%	意味
1	ISO 15693
2	TAGIT
3	ICODE
4	Mifare ISO 14443A
5	SR176
6	ISO 14443B

使用例 Mkey\$ = GET_RFID_KEY(4) ` Mifare ISO 14443A タグ のメモリリット-を取得

SET_RFID_KEY

適用機種 8300/8500

目的 特定 RFID タグ のメモリリット-を設定します。

構文 SET_RFID_KEY(TagType%, keyString\$, KeyType%)

解説 Mifare Standard 1K/4K や SLE66R35 のような特定 RFID タグ のメモリリット-を設定します。
引数 TagType%には、下記のタグ タイプ を示す値を指定します。

TagType%	意味
1	ISO 15693
2	TAGIT
3	ICODE
4	Mifare ISO 14443A
5	SR176
6	ISO 14443B

引数 TagString\$には、設定したいメモリリット-を指定します。
引数 KeyType%には、下記のキータイプ を示す値を指定します。

KeyType%	意味
1	Key A
2	Key B

使用例 SET_RFID_KEY(4, "111111111111", 1) ` Mifare ISO 14443A タグ のメモリリット-を設定
` キータイプ Key A, キー 111111111111

5.9. キーボードインターフェイス

8300 ターミナルは、専用のキーボードインターフェイスケーブルを本体に接続することで、キーボードインターフェイスを通して、PC に直接データを送信することができます。

参考

他のターミナルモデルは、キーボードインターフェイスを搭載していません。下記の方法で代用が可能です。

- Bluetooth HID インターフェイス接続
- Bluetooth SPP インターフェイス接続 + キーボードウィッジユーティリティソフト RsWedge
- シリアルインターフェイス接続 + キーボードウィッジユーティリティソフト RsWedge

SEND_WEDGE

適用機種	8300
目的	キーボードインターフェイス経由でデータを送信します。
構文	SEND_WEDGE (DataString\$)
解説	引数 DataString\$ には、送信したいデータを指定します。
使用例	... DataString\$ = CHR\$(9) + "TESTING" + CHR\$(9) ' [TAB] TESTING [TAB] SEND_WEDGE (DataString\$)

SET_WEDGE

適用機種	8300
目的	キーボードインターフェイスの設定を行います。
構文	SET_WEDGE (WedgeSetting\$)
解説	引数 WedgeSetting\$ には、3 バイトの設定値を指定します。各バイトの意味は、下記を参照ください。

バイト	ビット	意味
1	7 ~ 0	別表の PC タイプ / キーボード タイプ 表を参照ください。
2	7	1 : CAPS ロック自動検出有効 0 : CAPS ロック自動検出無効
2	6	1 : CAPS ロックオフ 0 : CAPS ロックオン
2	5	1 : アルファベット大文字・小文字区別無し 0 : アルファベット大文字・小文字区別有り
2	4 ~ 3	11 : 数字上段ホジションキーボード 10 : 数字下段ホジションキーボード 00 : ノーマルキーボード
2	2 ~ 1	11 : SHIFT ロックキーボード 10 : CAPS ロックキーボード 00 : ノーマルキーボード
2	0	1 : 数字をテンキーボードとして送信 0 : 数字をフルキーボードとして送信
3	7 ~ 0	キャラクタ間送信デレイを 0~255 の範囲で指定します。 設定単位は、10 ミリ秒。

(*) 上記のキーボードインターフェイス設定は、全ての PC タイプ / キーボード タイプ で動作するものではありません。事前に動作検証をお願いします。

PCタイプ/キーボードタイプ表			
値	PCタイプ/キーボードタイプ	値	PCタイプ/キーボードタイプ
0	使用不可(データ送信されません)	21	PS55 002-81, 003-81
1	PCAT (US)	22	PS55 002-2, 003-2
2	PCAT (FR)	23	PS55 002-82, 003-82
3	PCAT (GR)	24	PS55 002-3, 003-3
4	PCAT (IT)	25	PS55 002-8A, 003-8A
5	PCAT (SV)	26	IBM 3477 TYPE 4 (日本語)
6	PCAT (NO)	27	PS2-30
7	PCAT (UK)	28	Memorex Telex 122 keys
8	PCAT (BE)	29	PCXT
9	PCAT (SP)	30	IBM 5550
10	PCAT (PO)	31	NEC 5200
11	PS55 A01-1	32	NEC 9800
12	PS55 A01-2 (日本語 106/109キー)	33	DEC VT220, 320, 420
13	PS55 A01-3	34	MAC ADB
14	PS55 001-1	35	日立 Elles
15	PS55 001-81	36	Wyse Enhance KBD (US)
16	PS55 001-2	37	NEC Astra
17	PS55 001-82	38	UNISYS TO-300
18	PS55 001-3	39	Televideo 965
19	PS55 001-8A	40	ADDS 1010
20	PS55 002-1, 003-1		

キーボードインターフェイスコード表									
	0	1	2	3	4	5	6	7	8
0		F2	SP	0	@	P	`	p	(0)
1	Insert	F3	!	1	A	Q	a	q	(1)
2	Delete	F4	"	2	B	R	b	r	(2)
3	Home	F5	#	3	C	S	c	s	(3)
4	Eend	F6	\$	4	D	T	d	t	(4)
5	↑	F7	%	5	E	U	e	u	(5)
6	↓	F8	&	6	F	V	f	v	(6)
7	←	F9	'	7	G	W	g	w	(7)
8	Bs	F10	(8	H	X	h	x	(8)
9	Tab	F11)	9	I	Y	i	y	(9)
A	LF	F12	*	:	J	Z	j	z	
B	→	Esc	+	;	K	[k	{	
C	PgUp	Exec	,	<	L	¥	l		
D	CR	CR*	-	=	M]	m	}	
E	PgDn		.	>	N	^	n	~	
F	F1		/	?	O	_	o	DLY	Enter*

1. DLY は、100 ミリ秒のデレイタイムを意味します。
2. Exec は、実行キー(右 ALT キー)を意味します。
3. CR*/Enter*は、キーボードの Enter キーを意味します。
4. (0)~(9)は、キーボードの数字キーを意味します。

スキャノード送信とコピネーション送信	
0x0C	特殊な意味を持つコードです。0x0C に続くコードは、スキャノードと解釈され、スキャノードがそのまま送信されます。
	0xC0 0x01 : 次のキャラクタを Shift キー付で送信します。
	0xC0 0x02 : 次のキャラクタを左 Ctrl キー付で送信します。
	0xC0 0x04 : 次のキャラクタを左 Alt キー付で送信します。
	0xC0 0x08 : 次のキャラクタを右 Ctrl キー付で送信します。
	0xC0 0x10 : 次のキャラクタを右 Alt キー付で送信します。
	0xC0 0x20 : 次のキャラクタを送信後、全てのコピネーションステータスをクリアします。
<p>例えば、[A][Ctrl+Insert][5][スキャノード 0x29][Tab][2][Shift+Ctrl+A][B][Alt+1][Alt+2+Break][Alt+1][Alt+3]は、0x41,0x01,0x35,0xC0,0x29,0x09,0x32,0xC3,0x41,0x42,0xC4,0x31,0xE4,0x32,0xC4,0x31,0xC4,0x33 となり、BASIC プログラムでコーディングすると下記のようにになります。</p> <pre> ... Data_1\$ = CHR\$(65) + CHR\$(194) + CHR\$(1) + CHR\$(53) + CHR\$(192) + CHR\$(41) Data_2\$ = CHR\$(9) + CHR\$(50) + CHR\$(195) + CHR\$(65) + CHR\$(66) Data_3\$ = CHR\$(196) + CHR\$(49) + CHR\$(228) + CHR\$(50) + CHR\$(196) + CHR\$(49) Data_4\$ = CHR\$(196) + CHR\$(51) DataString\$ = Data_1\$ + Data_2\$ + Data_3\$ + Data4\$ SEND_WEDGE(DataString\$) ... </pre>	

使用例

```

...
Wedge_1$ = CHR$(1)      \ PCAT (US)
Wedge_2$ = CHR$(1)      \ CAPS ロック自動検出無効, CAPS ロックオフ,
                          \ アルファベット大文字・小文字区別有り, ノーマルキーボード
                          \ 数字をテンキーボードとして送信
Wedge_3$ = CHR$(5)      \ キー間送信遅延 5 ミリ秒
WedgeSetting$ = Wedge_1$ + Wedge_2$ + Wedge_3$
SET_WEDGE(WedgeSetting$)
SEND_WEDGE(DataString$)
...

```

SEND_READY

適用機種

8300

目的

キーボードインターフェイス経由でデータ送信が可能かをチェックします。

構文

REM State%は結果を代入するための整数型変数です。
State% = WEDGE_READY

解説

キーボードインターフェイスが正しく接続され、データ送信が可能な状態かをチェックし、結果を戻り値として返します。

戻り値	意味
0	データ送信不可
1	データ送信可能

使用例

```

IF (WEDGE_READY = 1) THEN
...
SEND_WEDGE(DATA$)
...
END IF

```

5.10. ブザーコマンド

BEEP


適用機種	ALL
目的	指定の音色・間隔でブザーを鳴動します。
構文	BEEP(freq%, duration% {, freq%, duration%})
解説	引数 <i>Freq%</i> には、ブザー周波数を表す数値を指定します。下記を参照ください。

<i>freq%</i>	意味										
0 以上	1000 (1KHz) ~ 6000 (6KHz) の範囲でブザー周波数を指定します。 ブザー音を無しにしたい場合は、0 を指定します。										
-1	<u>8400 ヘルツ</u> -1 を指定して、ブザー音量の設定が行えます。 <i>duration%</i> には、1~3 を指定します。 <u>8200 ヘルツ</u> -1 を指定して、ブザー音量の設定が行えます。 <i>duration%</i> には、0~3 を指定します。 <table><tr><th>Duration%</th><th>意味</th></tr><tr><td>0</td><td>ゼロ音量</td></tr><tr><td>1</td><td>小音量</td></tr><tr><td>2</td><td>中音量</td></tr><tr><td>3</td><td>大音量</td></tr></table>	Duration%	意味	0	ゼロ音量	1	小音量	2	中音量	3	大音量
Duration%	意味										
0	ゼロ音量										
1	小音量										
2	中音量										
3	大音量										
-2	<u>8200 ヘルツ</u> -2 を指定することで、SD カードにある WAV ファイル再生が行えます。 <i>duration%</i> には、WAV ファイル名を示す値を指定します。例えば、1 と指定した場合、SD カードの WAV フォルダにある 1.wav が再生されます。										

使用例

```
...  
BcrData_1:  
    BEEP(-1, 1)           `ブザー小音量  
    BEEP(2000, 10, 0, 10, 2000, 10)  
    BEEP(-2, 1)           `A:¥WAV¥1.WAV を再生  
...  
RETURN
```

注意

 8200 ヘルツで再生可能な WAV ファイル仕様は、下記となります。WAV ファイルのサンプリングレートと MicroSD カード、WAV ファイルを作成するソフトの相性によっては、正しく再生されない場合があります。

フォーマット	: Windows 標準 WAV フォーマット
オーディオサンプリングレート	: 8/11/16/22/32/44Hz
チャンネル	: モノラル
オーディオサンプリングサイズ	: 16bit
オーディオ形式	: PCM

STOP BEEP

適用機種	ALL
目的	ブザーを停止します。
構文	STOP BEEP
解説	ブザーを停止します。
使用例	<pre>BEEP(2000, 0) ON KEY(1) GOSUB StopBeep PRINT "Press F1 to stop the buzzer." ... StopBeep: STOP BEEP RETURN</pre>

5.11. LEDコマンド

LED

適用機種	ALL
目的	LED を制御します。
構文	LED(<i>number%</i> , <i>mode%</i> , <i>duration%</i>)
解説	引数 <i>number%</i> には、LED 番号を指定します。下記を参照ください。

<i>number%</i>	意味
1	赤色 LED
2	緑色 LED
3	青色第 2LED (8200/8400/8700 が搭載している第 2 LED, デフォルトは無線通信ステータス表示用 LED)
4	緑色第 2LED (8200/8400/8700 が搭載している第 2 LED, デフォルトは無線通信ステータス表示用 LED)

引数 *Mode%*には、動作モードを指定します。下記を参照ください。

<i>mode%</i>	意味
0	指定時間 (<i>Duration%</i>) LED を消灯してから点灯させます。
1	指定時間 (<i>Duration%</i>) LED を点灯してから消灯させます。
2	指定時間 (<i>Duration%</i>) LED を点滅させます。点滅時間は、 <i>Duration%</i> の値の 2 倍になります。
240	8200/8400/8700 の第 2 LED は、デフォルト動作を行います。 <u>デフォルト動作</u> 青色 LED 高速点滅 : Bluetooth 接続待ち又は接続中 青色 LED 低速点滅 : Bluetooth 接続完了 緑色 LED 高速点滅 : WiFi 接続待ち又は接続中 緑色 LED 低速点滅 : WiFi 接続完了
241	8200/8400/8700 の第 2 LED は、デフォルト動作を中断し、ユーザー定義の動作を行います。下記の例を参照ください。 LED(3, 241, 0) 、青色第 2LED をユーザーが制御 LED(3, 240, 0) 、青色第 2LED をシステムが制御

引数 *Duration%*には、時間を 10 ミリ秒単位で指定します。0 を指定した場合は、指定の動作を継続します。

使用例	<pre>ON READER(1) GOSUB BcrData_1 ... BcrData_1: BEEP(2000, 5) LED(2, 1, 5) 、グッドリッド LED Data\$ = GET_READER_DATA\$(1) ...</pre>
-----	---

5.12. バイブレータマツド

VIBRATOR

適用機種	8200/8400/8500/8700
目的	バイブレータを制御します。
構文	VIBRATOR (<i>mode</i> %)
解説	引数 <i>mode</i> % には、動作モードを指定します。下記を参照ください。

<i>mode</i> %	意味
0	バイブレータをオにします。
1	バイブレータをオにします。 一度、バイブレータをオにすると、VIBRATOR(0) が実行されるまでオにはなりません。

使用例

```
ON READER(1) GOSUB BcrData_1
...
BcrData_1:
    VIRATOR(1)           、バイブレータ オ
    BEEP(2000, 5)
    LED(2, 1, 5)         、グッドリット LED
    Data$ = GET_READER_DATA$(1)
    WAIT(100)            、500 ミ秒 待ち
    VIBRAOR(0)           、バイブレータ オ
...
```

5.13. リアルタイムクロックコマンド

ターミナルは、バックアップ電池によって保護されたリアルタイムチップを内蔵しています。このリアルタイムチップは、ターミナルの電源が切された場合でも保持され、閏年にも対応しています。

DATE\$

適用機種

ALL

目的

日付 (年月日) を設定又は取得します。

構文

REM X\$は日付を設定するための文字列型変数です。
DATE\$ = X\$

解説

REM Y\$は取得した日付を代入するための文字列型変数です。
Y\$ = DATE\$

日付を設定する場合、X\$に yyyyymmdd フォーマットで日付を指定します。
日付と取得する場合、Y\$に yyyyymmdd フォーマットで日付が返されます。

yyyy	: 西暦年	例) 2012 = 西暦 2012 年
mm	: 月	例) 12 = 12 月
dd	: 日	例) 31 = 31 日

Cipher BASIC コマンド及びリアルタイムチップは、与えられた yyyyymmdd フォーマットデータが正しいかどうかのチェックは行いませんので、プログラム内で必要に応じたチェックを行ってください。

使用例

DATE\$ = "20120101"	、 2012 年 1 月 1 日に設定
Today\$ = DATE\$	、 現在の日付 (yyyyymmdd) を取得

DAY_OF_WEEK

適用機種

ALL

目的

曜日を取得します。

構文

REM A%は曜日に対応する数値を代入するための整数列型変数です。
A% = DAY_OF_WEEK

解説

曜日に対応する数値を戻り値として返します。下記を参照ください。

戻り値 A%	意味
1	月曜日
2	火曜日
3	水曜日
4	木曜日
5	金曜日
6	土曜日
7	日曜日

使用例

```
ON DAY_OF_WEEK GOSUB 100, 200, 300, 400, 500, 600, 700
...
100
    PRINT "月曜日"
RETURN

200
    PRINT "火曜日"
RETURN
```

TIME\$

適用機種 ALL

目的 時刻 (時分秒) を設定又は取得します。

構文 REM X\$は時刻を設定するための文字列型変数です。
TIME\$ = X\$

解説 REM Y\$は取得した時刻を代入するための文字列型変数です。
Y\$ = TIME\$

時刻を設定する場合、X\$に hhmmss フォーマットで時刻を指定します。
時刻と取得する場合、Y\$に hhmmss フォーマットで時刻が返されます。

hh	: 時	例) 15 = 15 時 (24 時間形式)
mm	: 分	例) 59 = 59 分
ss	: 秒	例) 31 = 31 秒

Cipher BASIC のバグ及び不具合は、与えられた hhmmss フォーマットが正しいかどうかのチェックは行いませんので、プログラム内で必要に応じたチェックを行ってください。

使用例	TIME\$ = "112500"	11:25.00 に設定
	CurrentTime\$ = TIME\$	現在の時刻 (hhmmss) を取得

TIMER

適用機種 ALL

目的 ターミナルを起動してからの秒数を取得します。

構文 REM A&はタイマー値 (秒数) を代入するための長整数型変数です。
A& = TIMER

解説 ターミナルを起動してからの秒数を戻り値として返します。

TIMER は読取専用のため値を代入することはできません。

使用例

```
StartTime& = TIMER
...
Loop:
    IF EndTime& <> TIMER
        TimerElapsed& = EndTime& -StartTime&
        CLS
        PRINT TimerElapsed&
        IF TimerElapsed& > 100 THEN GOTO NextStep
    END IF
    GOTO Loop
NextStep:
...
```

関連項目 OFF TIMER, ON TIMER GOSUB ...

WAIT

適用機種	ALL
目的	指定時間プログラムを停止します。
構文	WAIT(<i>duration</i> %)
解説	引数 <i>duration</i> %には、プログラム停止時間を指定します。設定単位は、5 ミリ秒です。 割り込み待ちのループ処理で WAIT 文を実行することで、消費電量を最小限に抑えることができます。CHANGE_SPEED 文で CPU の速度を落とすよりも効果的です。
使用例	<pre>ON COM(1) GOSUB RcvData While(1) WAIT(100) ' 500 ミリ秒待ち WEND RcvData: RETURN</pre>

5.14. バッテリーコメント

BACKUP_BATTERY

適用機種 ALL

目的 バックアップ電池の電圧を取得します。

構文 REM A%は戻り値を代入するための整数型変数です。

A% = BACKUP_BATTERY

解説 バックアップ電池の電圧を戻り値として返します。単位は、mV (ミリボルト) です。

バックアップ電池は、ターミナルの電源が切の状態でも SRAM 及びレジスタチップの内容が失われないよう、保護しています。バックアップ電池電圧が 2900mV を下回った場合は、バッテリー切れ状態で、注意が必要です。すぐにターミナルの充電をおこなってください。バックアップ電池が無くなると、SRAM 及びレジスタチップの内容は消滅します。

使用例

```
BATTERY_LOW% = 2900
CheckBackupBattery:
    IF BACKUP_BATTERY < BATTERY_LOW% THEN
        BEEP(2000, 30)
        CLS
        PRINT "バックアップ電池充電!!"
    LOOP:
        GOTO LOOP
    END IF
```

MAIN_BATTERY

適用機種 ALL

目的 メイン電池の電圧を取得します。

構文 REM A%は戻り値を代入するための整数型変数です。

A% = MAIN_BATTERY

解説 メイン電池の電圧を戻り値として返します。単位は、mV (ミリボルト) です。

メイン電池電圧が 3400mV (アルカリ乾電池の場合は、2200mV) を下回った場合は、バッテリー切れ状態です。すぐにターミナルの充電又は電池交換を行ってください。バッテリー切れ状態では、ターミナルは動作しますが、消費の高い一部の機能が正しく動作しない可能性があります。

使用例

```
BATTERY_LOW% = 3400
CheckMainBattery:
    IF MAIN_BATTERY < BATTERY_LOW% THEN
        BEEP(2000, 30)
        CLS
        PRINT "メイン電池を充電!!"
    LOOP:
        GOTO LOOP
    END IF
```

5.15. キーボードコマンド

CLR_KBD

適用機種	ALL
目的	キーボードをクリアします。
構文	CLR_KBD
解説	キーボードをクリアします。
使用例	<pre>CLR_KBD ON KEY(1) GOSUB KeyData_1 ...</pre>

INKEY\$

適用機種	ALL
目的	キーボードから 1 文字取得します。
構文	<pre>REM X\$は戻り値を代入するための文字列型変数です。 X\$ = INKEY\$</pre>
解説	キーボードから 1 文字を取得し、戻り値として返します。取得した 1 文字は、キーボードから削除されます。キーボードにデータが無い場合は、空の文字列を返します。
使用例	<pre>... PRINT "Intialize System(Y/N)?" Loop: KeyData\$ = INKEY\$ IF KeyData\$ = "" THEN GOTO Loop ELSE IF KeyData\$ = "Y" THEN GOTO Initialize ...</pre>

INPUT

適用機種	ALL
目的	キーボードからの入力を変数に代入します。
構文	INPUT <i>variable</i>
解説	<p>引数 <i>variable</i> には、入力する型に応じた変数を指定します。指定した変数に合ったデータを入力しなければいけません。</p> <p>Enter キーが押されると入力が確定し、指定の変数に入力値が代入されます。ESC キーが押された場合は、処理は終了し、入力値はクリアされます。</p>
使用例	<pre>INPUT String\$ 、 文字列を入力 PRINT String\$ INPUT Number% 、 整数を入力 PRINT Number%</pre>

INPUT_MODE

適用機種	ALL
目的	入力モードを設定します。
構文	INPUT_MODE (<i>mode</i> %)
解説	引数 <i>mode</i> %には、設定したい入力モードに対応する値を指定します。

<i>mode</i> %	意味
0	入力した文字をディスプレイに表示しません。
1	入力した文字をそのままディスプレイに表示します。(デフォルト)
2	入力した文字の代わりにアスタリスク(*)をディスプレイに表示します。パスワード入力を行う際などに利用されます。

使用例	LOCATE 1, 1 INPUT_MODE(1) INPUT Login\$ LOCATE 2,1 INPUT_MODE(2) INPUT Password\$
-----	--

KEY_CLICK

適用機種	ALL
目的	キークリック音を設定します。
構文	KEY_CLICK (<i>mode</i> %)
解説	引数 <i>mode</i> %には、設定したいキークリック音に対応する値を指定します。

<i>mode</i> %	意味
0	キークリック音を消にします。
1~5	キークリック音を消にします。1~5 には、それぞれ異なる音色が設定されています。

使用例	KEY_CLICK(0)	、キークリック音無し
-----	--------------	------------

PUTKEY

適用機種	8200/8400/8500/8700
目的	キーボードに 1 文字を書き込みます。
構文	PUKEY (<i>n</i> %)
解説	引数 <i>n</i> %には、書き込みたい文字に対応する ASCII コードを 10 進数で指定します。

このコマンドをコールすることで、キーボードからの入力をシミュレーションすることができます。例えば、タッチクリソ上にキーボードを描画し、その押されたキーに相当する文字をキーボードに書き込むことでキーボードをシミュレートできます。

使用例	PUTKEY (27) 、ESC キー (27) をキーボードに書き込み
-----	---

ALPHA_LOCK

適用機種	ALL
目的	入力モードと ALPHA ステートを設定します。
構文	ALPHA_LOCK (<i>mode</i> %)
解説	引数 <i>mode</i> %には、設定したい入力モードと ALPHA ステートに対応する値を指定します。

<i>mode</i> %	入力モード	ALPHA ステート
0	数字モード	□□無し
1	英字モード (大文字)	□□無し
2	数字モード	□□有り
3	英字モード (小文字)	□□無し
4	ファンクションモード (8000/8200 のみ)	□□無し
5	英字モード (大文字)	□□有り
6	英字モード (小文字)	□□有り
7	ファンクションモード (8000/8200 のみ)	□□有り

英字モードでは、1 秒以内に同じキーを繰り返し押すことで、キーに割り当てられている英字と数字を順番に表示していきます。1 秒以上経過するか異なるキーが押されると、表示中の文字が入力されます。例えば、「2ABC」と表示されているキーの場合、A→B→C→2→A→B... という順序で表示されます。

使用例	ALPHA_LOCK (1)
-----	----------------

GET_ALPHA_LOCK

適用機種	ALL
目的	入力モードと ALPHA ステートを取得します。このコマンドは、互換性維持のため、残されています。 通常は、次の GET_ALPHA_STATE コマンドをご利用ください。
構文	REM A%は戻り値を代入するための整数型変数です。 A% = GET_ALPHA_LOCK
解説	入力モードと ALPHA ステートを取得し、戻り値として返します。ALPHA キーに対応していないターミナルの場合は、-1 を返します。
使用例	ALPHA_LOCK% = GET_ALPHA_LOCK

GET_ALPHA_STATE

適用機種	ALL
目的	入力モードと ALPHA ステートを取得します。
構文	REM A%は戻り値を代入するための整数型変数です。 A% = GET_ALPHA_STATE
解説	入力モードと ALPHA ステートを取得し、戻り値として返します。ALPHA キーに対応していないターミナルの場合は、-1 を返します。

A%	入力モード	ALPHA ステート
0	数字モード	ロック無し
1	英字モード (大文字)	ロック無し
2	数字モード	ロック有り
3	英字モード (小文字)	ロック無し
4	ファンクションモード (8000/8200 のみ)	ロック無し
5	英字モード (大文字)	ロック有り
6	英字モード (小文字)	ロック有り
7	ファンクションモード (8000/8200 のみ)	ロック有り

使用例	ALPHA_LOCK% = GET_ALPHA_LOCK
-----	------------------------------

FUNCTION_TOGGLE

適用機種	8300/8400/8500/8700
目的	FUNCTION ｽｰｰﾄを設定します。
構文	FUNCTION_TOGGLE (<i>state</i> %)
解説	引数 <i>state</i> %には、設定したい FUNCTION ｽｰｰﾄに対応する値を指定します。

ﾀｰﾐナル機種	<i>state</i> %	説明
8300	0	オートﾚｼﾞ ﾕｰﾑﾓｰﾄﾞ + ﾏﾙﾁｷｰﾓｰﾄﾞ (ﾃﾞｨﾌｵﾙﾄ)
	1	ﾄﾞﾞﾞﾙﾓｰﾄﾞ + ﾏﾙﾁｷｰﾓｰﾄﾞ
8400 8500/8700 (44 ｷｰﾀｲﾌﾟ 2)	0	オートﾚｼﾞ ﾕｰﾑﾓｰﾄﾞ + ﾏﾙﾁｷｰﾓｰﾄﾞ (ﾃﾞｨﾌｵﾙﾄ)
	1	ﾄﾞﾞﾞﾙﾓｰﾄﾞ
	2	オートﾚｼﾞ ﾕｰﾑﾓｰﾄﾞ + ﾏﾙﾁｷｰﾓｰﾄﾞ + FN ﾆｰﾏﾙｷｰ
	3	ﾄﾞﾞﾞﾙﾓｰﾄﾞ + FN ﾆｰﾏﾙｷｰ
	4	ﾏﾙﾁｷｰﾓｰﾄﾞ
	6	ﾏﾙﾁｷｰﾓｰﾄﾞ + FN ﾆｰﾏﾙｷｰ
8500/8700 (24 ｷｰ) 8500 (44 ｷｰ ﾀｲﾌﾟ 1)	0	オートﾚｼﾞ ﾕｰﾑﾓｰﾄﾞ + ﾏﾙﾁｷｰﾓｰﾄﾞ (ﾃﾞｨﾌｵﾙﾄ)
	1	ﾄﾞﾞﾞﾙﾓｰﾄﾞ
	2	オートﾚｼﾞ ﾕｰﾑﾓｰﾄﾞ + ﾏﾙﾁｷｰﾓｰﾄﾞ + FN ﾆｰﾏﾙｷｰ
	3	ﾄﾞﾞﾞﾙﾓｰﾄﾞ + FN ﾆｰﾏﾙｷｰ
	4	-

オートﾚｼﾞ ﾕｰﾑﾓｰﾄﾞ

ﾌｧﾝｸｼｮﾝｷｰを押すと、ﾌｧﾝｸｼｮﾝﾓｰﾄﾞ が ｵﾝ になり、続く第 2 ｷｰを押すと、ﾌｧﾝｸｼｮﾝｷｰ入力が確定し、ﾌｧﾝｸｼｮﾝﾓｰﾄﾞ が ｵﾌ になります。ﾌｧﾝｸｼｮﾝﾓｰﾄﾞ が ｵﾝ になると、ﾃﾞｨｽﾌﾟﾚｲ下段に F アイｺﾝが表示されます。8300/8400/8700 シﾘｰｽﾞ ﾀｰﾐナルでは、再度ﾌｧﾝｸｼｮﾝｷｰを押すことで、ﾌｧﾝｸｼｮﾝﾓｰﾄﾞ を ｵﾌ にすることができます。

ﾄﾞﾞﾞﾙﾓｰﾄﾞ

ﾌｧﾝｸｼｮﾝｷｰを押すと、ﾌｧﾝｸｼｮﾝﾓｰﾄﾞ が ｵﾝ になり、再度押すと、 ｵﾌ になります。ﾌｧﾝｸｼｮﾝﾓｰﾄﾞ が ｵﾝ になると、ﾃﾞｨｽﾌﾟﾚｲ下段に F アイｺﾝが表示されます。

ﾏﾙﾁｷｰﾓｰﾄﾞ

ﾌｧﾝｸｼｮﾝｷｰを押した状態で、続く第 2 ｷｰを押すことでﾌｧﾝｸｼｮﾝｷｰを入力するﾓｰﾄﾞ です。

FN ﾆｰﾏﾙｷｰ

ﾌｧﾝｸｼｮﾝｷｰを通常ｷｰのひとつとして扱います。

使用例	FUNCTION_TOGGLE (0)	、 オートﾚｼﾞ ﾕｰﾑ + ﾏﾙﾁｷｰﾓｰﾄﾞ
-----	---------------------	--------------------------

5.16. ディスプレイマント

ターミナルは、それぞれ下記の解像度・座標を持つバックライト付グラフィックディスプレイを搭載しています。

ターミナル機種	解像度	左頂点座標 (x, y)	右上座標 (x, y)
8000	100 x 64ドット	(0, 0)	(99, 63)
8300	128 x 64ドット	(0, 0)	(127, 63)
8200/8400	160 x 160ドット	(0, 0)	(159, 159)
8500/8700	160 x 160ドット	(0, 0)	(159, 159)

ディスプレイコントラスト

コントラストレベル 1~8 の 8 段階調整が可能です。デフォルトは、レベル 5 です。

バックライト

デフォルトは、バックライトオフです。8000/8300/8500/8700 シリーズターミナルでは、「FN」+「Enter」キーでバックライトのオン/オフが行えます。8200/8400 シリーズターミナルは、バックライト専用キーを搭載しています。バックライトのデフォルトは、レベル 2 です。

BACK_LIGHT_DURATION

適用機種	ALL
目的	バックライトの点灯時間を設定します。
構文	BACK_LIGHT_DURATION (n%)
解説	引数 n% には、バックライト点灯時間を秒単位で指定します。
使用例	BACK_LIGHT_DURATION (20) 、バックライト点灯時間 20 秒

BACKLIT

適用機種	ALL
目的	バックライト動作モードを設定します。
構文	BACKLIT (state%)
解説	引数 state% には、設定したいバックライト動作モードに対応する数値を指定します。

ターミナル機種	state%	意味
8000/8300	0	バックライトオフ
	1	バックライトオン
8200/8400	0	バックライトオフ
	1	バックライトオン 超低輝度
	2	バックライトオン 低輝度
	3	バックライトオン 中輝度
	4	バックライトオン 高輝度
	16	バックライトオン シェイド 効果無し
	17	バックライトオン シェイド 最小効果
	18	バックライトオン シェイド 小効果
	19	バックライトオン シェイド 中効果
	20	バックライトオン シェイド 大効果
8500/8700	0	バックライトオフ
	1	バックライトオン 超低輝度
	2	バックライトオン 低輝度
	3	バックライトオン 中輝度
	4	バックライトオン 高輝度

使用例	BACK_LIT (1) 、8500/8700 バックライトオン 超低輝度
-----	--

LCD_CONTRAST

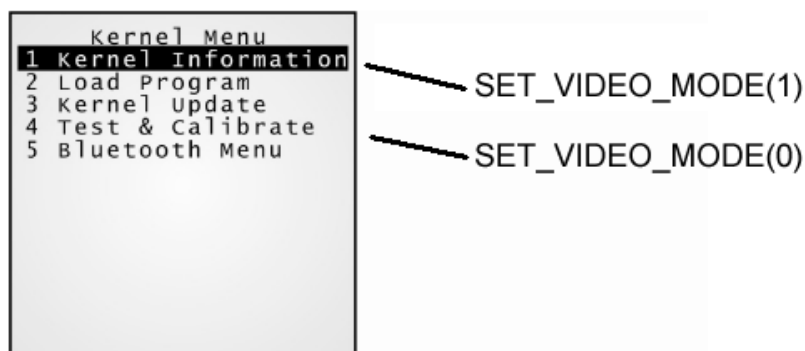
適用機種	ALL
目的	コントラストレベルを設定します。
構文	LCD_CONTRAST (<i>n%</i>)
解説	引数 <i>n%</i> には、コントラストレベルを 1 (低コントラスト) ~ 8 (高コントラスト) の範囲で指定します。
使用例	LCD_CONTRAST (4) 、コントラストレベル 4 (中コントラスト)

SET_VIDEO_MODE

適用機種	ALL
目的	ビデオモードを設定します。
構文	SET_VIDEO_MODE (<i>mode%</i>)
解説	引数 <i>mode%</i> には、設定したいビデオモードに対応する値を指定します。

<i>mode%</i>	意味
0	通常モード
1	反転モード (白黒反転)

使用例



CURSOR

適用機種	ALL
目的	カーソルの表示・非表示を設定します。
構文	CURSOR (<i>mode%</i>)
解説	引数 <i>mode%</i> には、カーソルの表示・非表示に対応する値を指定します。

<i>mode%</i>	意味
0	カーソル表示
1	カーソル非表示

使用例 CURSOR 、カーソル表示

CURSOR_X

適用機種	ALL
目的	現在のカーソル位置 (X 座標) を取得します。
構文	REM X%は戻り値を代入するための整数型変数です。 X% = CURSOR_X
解説	現在のカーソル位置 (X 座標) を戻り値として返します。
使用例	<pre>ON READER(1) GOSUB BcrData_1 ... Bcr_Data1: BEEEP(2000, 5) Data\$ = GET_READER_DATA\$(1) Pre_X% = CURSOR_X Pre_Y% = CURSOR_Y LOCATE 8, 1 PRINT Data\$ LOCATE Pre_Y%, Pre_X% RETURN</pre>

CURSOR_Y

適用機種	ALL
目的	現在のカーソル位置 (Y 座標) を取得します。
構文	REM Y%は戻り値を代入するための整数型変数です。 Y% = CURSOR_Y
解説	現在のカーソル位置 (Y 座標) を戻り値として返します。
使用例	<pre>ON READER(1) GOSUB BcrData_1 ... Bcr_Data1: BEEEP(2000, 5) Data\$ = GET_READER_DATA\$(1) Pre_X% = CURSOR_X Pre_Y% = CURSOR_Y LOCATE 8, 1 PRINT Data\$ LOCATE Pre_Y%, Pre_X% RETURN</pre>

LOCATE

適用機種	ALL
目的	カーソルを指定位置に移動します。
構文	LOCATE y%, x%
解説	引数 y%には、カーソルを移動したい Y 座標を指定し、引数 x%には、カーソルを移動したい X 座標を指定します。 指定できる値は、ターミナルの画面サイズ、使用するフォントサイズ及びアインマリヤを使用するか (ICON_ZONE_PRINT マット) の設定によります。8500 シリーズでは、ICON_ZONE_PRINT(0) が設定されていて、且つ 6x8 ドットフォントを使用している場合は、18 を超える Y 座標を指定することはできません。
使用例	<pre>LOCATE 1,1</pre> 、カーソルを左上頂点に移動します

FILL_RECT

適用機種	ALL
目的	指定の長方形エリアを塗りつぶします。
構文	<code>FILL_RECT(x%, y%, size_x%, size_y%)</code>
解説	引数 <i>x%</i> には、長方形エリアの開始 <i>x</i> 座標、引数 <i>y%</i> には、長方形エリアの開始 <i>y</i> 座標、引数 <i>size_x%</i> には、長方形エリアの幅、引数 <i>size_y%</i> には、長方形エリアの高さをそれぞれドット単位で指定します。
使用例	<code>FILL_RECT(1,1, 20, 10)</code>
関連項目	CLR_RECT

ICON_ZONE_PRINT

適用機種	ALL
目的	アイコンエリアを文字表示エリアとして使用するかを設定します。
構文	<code>ICON_ZONE_PRINT(status%)</code>
解説	引数 <i>status%</i> には、アイコンエリアを文字表示エリアとして使用するかを示す値を指定します。

<i>status%</i>	説明
0	アイコンエリアを文字表示エリアとして使用しない。(デフォルト)
1	アイコンエリアを文字表示エリアとして使用する。

アイコンエリアは、デフォルトでは、バッテリーアイコンや ALPHA アイコンなどのステータスアイコンが表示されるエリアで、PRINT コマンドによる文字表示は行えません。グラフィックコマンドによるアキスのみが許可されています。

ターミナル機種	ディスプレイ解像度	説明
8000	100 x 64 ドット	アイコンエリアは、ディスプレイ右端の 4x64 ドットの縦長エリアになります。幅 4 ドットは、1 文字を表示するにも小さすぎるため、アイコンエリアを文字表示エリアとした場合でも、表示できる文字桁数は変わりません。
8200/8400	160 x 160 ドット	アイコンエリアは、ディスプレイ下段の 160x16 ドットの横長エリアになります。アイコンエリアを文字表示エリアとした場合、6x8 フォントで 26 文字 x20 行、8x16 フォントで 20 文字 x10 行の表示が可能になります。
8300	128 x 64 ドット	アイコンエリアは、ディスプレイ右端の 8x64 ドットの縦長エリアになります。アイコンエリアを文字表示エリアとした場合、6x8 フォントで 21 文字 x8 行、8x16 フォントで 16 文字 x4 行の表示が可能になります。
8500/8700	160 x 160 ドット	アイコンエリアは、ディスプレイ下段の 160x8 (6x8 フォント) 又は 160x16 (8x16 フォント) ドットの横長エリアになります。アイコンエリアを文字表示エリアとした場合、6x8 フォントで 26 文字 x20 行、8x16 フォントで 20 文字 x10 行の表示が可能になります。

ICON_ZONE_PRINT(1) によって、アイコンエリアを文字表示エリアとした場合、CLS コマンドをコールすると、アイコンエリアを含む全ての文字がクリアされます。但し、アイコンエリアを文字表示エリアとした場合でも、バッテリーアイコンなどのステータスアイコンは、システムによって定期的に更新されるため、表示文字上に上書きされることがあります。

使用例	<code>ICON_ZONE_PRINT(1)</code>	アイコンエリアを文字表示エリアとして使用
関連項目	PRINT	

PRINT

適用機種	ALL
目的	データをディスプレイに表示します。
構文	PRINT <i>expression</i> [{, ;[<i>expression</i>]}]
解説	<p>引数 <i>expression</i> には、表示したい数値や文字列及び相当する変数や式を指定します。各引数は、コマ(,)又はセミコロン(;)で区切られます。下記のルールを参照ください。</p> <ul style="list-style-type: none"> ✓ コマ(,)で区切った場合、次のデータはスペース1文字が挿入された後に表示されます。 ✓ セミコロン(;)で区切った場合、次のデータは直後に表示されます。スペースは挿入されません。 ✓ 最後にコマ(,)又はセミコロン(;)が無い場合、全データ表示後、改行(キャリッジリターン)が行われます。
使用例	<pre>PRINT "ABC", "DEF" \ 表示結果 ABC DEF PRINT "ABC"; "DEF" \ 表示結果 ABCDEF A%=5 PRINT A, "正方形面積 "; A*A \ 表示結果 5 正方形面積 25</pre>
関連項目	CLS, ICON_ZONE_PRINT

WAIT_HOURLASS

適用機種	ALL
目的	処理中を示す回転砂時計を表示します。
構文	WAIT_HOURLASS(<i>x%</i> , <i>y%</i> , <i>type%</i>)
解説	<p>引数 <i>x%</i>と <i>y%</i>には、回転砂時計を表示したい座標(<i>x</i>,<i>y</i>)をドット単位で指定します。</p> <p>引数 <i>type%</i>には、回転砂時計の大きさを示す値を指定します。</p>

<i>type%</i>	説明
0	24 x 23ドット
1	8 x 8ドット

回転砂時計を更新するため、連続して、このコマンドをコールしてください。経過時間を示す5パターンの回転砂時計が表示されます。1周期には、約3秒弱を要します。

使用例	WAIT_HOURLASS(68, 68, 1) \ 座標(68,68)に24x23ドット砂時計を表示
-----	---

CLR_RECT

適用機種	ALL
目的	長方形エリアをクリアします。
構文	CLR_RECT(<i>x%</i> , <i>y%</i> , <i>size%</i> , <i>size_y%</i>)
解説	<p>引数 <i>x%</i>には、長方形エリアの開始 <i>x</i> 座標、引数 <i>y%</i>には、長方形エリアの開始 <i>y</i> 座標、引数 <i>size_x%</i>には、長方形エリアの幅、引数 <i>size_y%</i>には、長方形エリアの高さをそれぞれドット単位で指定します。</p>
使用例	CLR_RECT(1, 1, 20, 20)
関連項目	CLS, FILL_RECT

CLS

適用機種	ALL
目的	ディスプレイ全体をクリアします。
構文	CLS
解説	ディスプレイに表示されている全てをクリアし、カーソル位置を(1,1)にリセットします。
使用例	<pre>ON TIMER(1,200) GOSUB ClearScreen \ TIMER(1) = 2CLR 秒 ... ClearScreen: OFF TIMER(1) CLS RETURN</pre>
関連項目	CLR_RECT, PRINT

GET_IMAGE

適用機種	ALL
目的	長方形エリアのビットマップパターンを取得します。
構文	<pre>REM DataCount%は戻り値を代入するための整数型変数です。 DataCount% = GET_IMAGE(file_index%, x%, y%, size_x%, size_y%)</pre>
解説	<p>引数 <i>file_index%</i> には、取得したビットマップマップパターンを保存するトランザクションファイル番号を 1~6 の範囲で指定します。</p> <p>引数 <i>x%</i> には、長方形エリアの開始 x 座標、引数 <i>y%</i> には、長方形エリアの開始 y 座標、引数 <i>size_x%</i> には、長方形エリアの幅、引数 <i>size_y%</i> には、長方形エリアの高さをそれぞれドット単位で指定します。</p> <p>結果として、トランザクションファイルに保存されたデータ総数が戻り値として返されます。</p>
使用例	<pre>GET_IMAGE(3,12, 32, 60, 16)</pre>
関連項目	GET_TRANSACTION_DATA\$, GET_TRANSACTION_DATA_EX\$, SET_SIGNAREA

SHOW_IMAGE

適用機種 ALL

目的 長方形エリアにビットマップパターンを出力します。

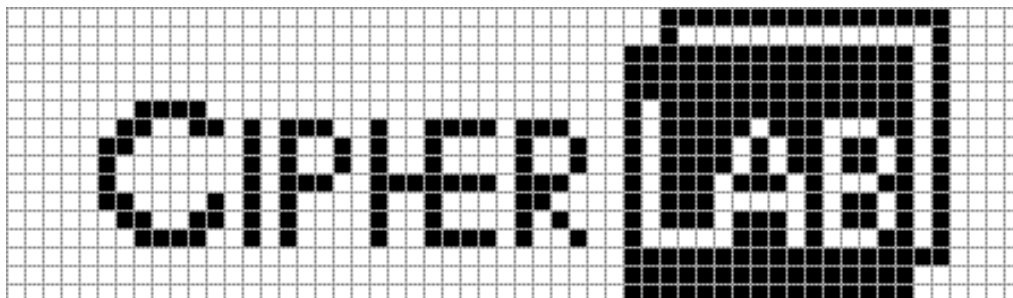
構文 REM DataCount%は戻り値を代入するための整数型変数です。
 SHOW_IMAGE(x%, y%, size_x%, size_y%, image\$)

解説 引数 x%には、長方形エリアの開始 x 座標、引数 y%には、長方形エリアの開始 y 座標、引数 size_x%には、長方形エリアの幅、引数 size_y%には、長方形エリアの高さをそれぞれドット単位で指定します。引数 image\$には、ビットマップパターンデータを指定します。

使用例 ` CipherLab 0J` の描画

```
icon_1$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(248)+chr$(255)+chr$(7)
icon_2$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(8)+chr$(0)+chr$(4)
icon_3$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(5)
icon_4$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(5)
icon_5$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(5)
icon_6$ = chr$(192)+chr$(3)+chr$(0)+chr$(0)+chr$(250)+chr$(255)+chr$(5)
icon_7$ = chr$(96)+chr$(214)+chr$(201)+chr$(59)+chr$(250)+chr$(142)+chr$(5)
icon_8$ = chr$(48)+chr$(80)+chr$(74)+chr$(72)+chr$(122)+chr$(109)+chr$(5)
icon_9$ = chr$(16)+chr$(80)+chr$(74)+chr$(72)+chr$(122)+chr$(109)+chr$(5)
icon_10$ = chr$(16)+chr$(208)+chr$(249)+chr$(59)+chr$(186)+chr$(139)+chr$(5)
icon_11$ = chr$(48)+chr$(84)+chr$(72)+chr$(24)+chr$(58)+chr$(104)+chr$(5)
icon_12$ = chr$(96)+chr$(86)+chr$(72)+chr$(40)+chr$(186)+chr$(107)+chr$(5)
icon_13$ = chr$(192)+chr$(83)+chr$(200)+chr$(75)+chr$(130)+chr$(139)+chr$(5)
icon_14$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(7)
icon_15$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(1)
icon_16$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(1)

show_image(2, 0, 56, 1, icon_1$)
show_image(2, 1, 56, 1, icon_2$)
show_image(2, 2, 56, 1, icon_3$)
show_image(2, 3, 56, 1, icon_4$)
show_image(2, 4, 56, 1, icon_5$)
show_image(2, 5, 56, 1, icon_6$)
show_image(2, 6, 56, 1, icon_7$)
show_image(2, 7, 56, 1, icon_8$)
show_image(2, 8, 56, 1, icon_9$)
show_image(2, 9, 56, 1, icon_10$)
show_image(2, 10, 56, 1, icon_11$)
show_image(2, 11, 56, 1, icon_12$)
show_image(2, 12, 56, 1, icon_13$)
show_image(2, 13, 56, 1, icon_14$)
show_image(2, 14, 56, 1, icon_15$)
show_image(2, 15, 56, 1, icon_16$)
...
```



CIRCLE

適用機種 ALL

目的 円形を描画します。







構文 CIRCLE(*cx*%, *cy*%, *r*%, *type*%, *mode*%)

解説 引数 *cx*%, *cy*%には、円形の中心座標 (*x*, *y*) を指定します。
引数 *r*%には、半径を指定します。
引数 *type*%には、円形の塗りつぶしを指定します。
引数 *mode*%には、ドットモードを指定します。

描画される図形は、引数 *type*%及び *mode*%の組み合わせにより異なります。下記の表を参照ください。

<i>type</i> %	説明	
0	SHAPE_NORMAL	塗りつぶし無し
1	SHAPE_FILL	塗りつぶし有り

<i>mode</i> %	説明	
1	DOT_MARK	通常ドット
0	DOT_CLEAR	クリアドット
-1	DOT_REVERSE	反転ドット

図形描画イメージ例			
<i>type</i> %	<i>mode</i> %		
	DOT_MARK (1)	DOT_CLEAR (0)	DOT_REVERSE (-1)
SHAPE_NORMAL (0)			
SHAPE_FILL (1)			

使用例 CIRCLE(80, 120, 8, 1, 1) \ 中心座標(8,120)半径8ドットの塗りつぶし円形

関連項目 CLS, LINE, PUT_PIXEL, RECTANGLE

LINE

適用機種 ALL

目的 線を描画します。

構文 LINE(*x1%*, *y1%*, *x2%*, *y2%*, *mode%*)

解説 引数 *x1%*, *y1%*には、線の始点座標(*x*,*y*)を指定します。
引数 *x2%*, *y2%*には、線の終点座標(*x*,*y*)を指定します。
引数 *mode%*には、ドットスタートを示す値を指定します。

<i>mode%</i>	説明	
1	DOT_MARK	通常ドット
0	DOT_CLEAR	クリアドット
-1	DOT_REVERSE	反転ドット

使用例 LINE(10, 10, 120, 10, 1) 、 横線
LINE(80,120, 10, 10, 1) 、 斜線

関連項目 CIRCLE, CLS, PUT_PIXEL, RECTANGLE

PUT_PIXEL

適用機種 ALL

目的 線を描画します。

構文 PIXEL(*x1%*, *y1%*, *mode%*)

解説 引数 *x1%*, *y1%*には、ドットを描画する座標(*x*,*y*)を指定します。
引数 *mode%*には、ドットスタートを示す値を指定します。

<i>mode%</i>	説明	
1	DOT_MARK	通常ドット
0	DOT_CLEAR	クリアドット
-1	DOT_REVERSE	反転ドット

使用例 PIXEL(80, 120, 1) 、 黒ドット 座標(80,120)

関連項目 CIRCLE, CLS, LINE, RECTANGLE

RECTANGLE

適用機種 ALL

目的 長方形を描画します。







構文 RECTANGLE (x1%, y1%, x2%, y2%, type%, mode%)

解説
引数 x1%, y1%には、長方形の左上頂点 (x, y) を指定します。
引数 x2%, y2%には、長方形の右下終点 (x, y) を指定します。
引数 type%には、長方形タイプを示す値を指定します。
引数 mode%には、ドットモードを示す値を指定します。

描画される図形イメージは、引数 type%及び mode%の組み合わせにより異なります。下記の表を参照ください。

type%	説明
0	SHAPE_NORMAL 塗りつぶし無し
1	SHAPE_FILL 塗りつぶし有り

mode%	説明
1	DOT_MARK 通常ドット
0	DOT_CLEAR クリアドット
-1	DOT_REVERSE 反転ドット

図形描画イメージ例			
type%	mode%		
	DOT_MARK (1)	DOT_CLEAR (0)	DOT_REVERSE (-1)
SHAPE_NORMAL (0)			
SHAPE_FILL (1)			

使用例 RECTANGLE (10, 20, 80, 100, 1, 1)

関連項目 CIRCLE, CLS, LINE, PUT_PIXEL

5.17. タッチスクリーンコマンド

8500/8700 シリーズ ターミナルは、タッチスクリーンを搭載しています。タッチスクリーンを利用することで、スタイラスで書いたサインイメージの取り込みやタッチスクリーン上に任意のボタンを描画して、キーボードのように扱うことが可能です。また、割り込みイベント (OFF TOUCHSCREEN, ON TOUCHSCREEN) にも対応します。

DISABLE_TOUCHSCREEN

適用機種	8500/8700
目的	タッチスクリーンを無効にします。
構文	DISABLE_TOUCHSCREEN
解説	タッチスクリーンを無効にします。有効にしたい場合は、ENABLE_TOUCHSCREEN コマンドをコールします。
使用例	DISABLE_TOUCHSCREEN 、タッチスクリーン無効

ENABLE_TOUCHSCREEN

適用機種	8500/8700
目的	タッチスクリーンを有効にします。
構文	ENABLE_TOUCHSCREEN
解説	タッチスクリーンを初期化し、有効にします。このコマンドがコールされるまで、タッチスクリーンは使用できません。
使用例	ENABLE_TOUCHSCREEN 、タッチスクリーン有効

GET_SCREENITEM

適用機種	8500/8700
目的	タップされたアイテムを検出し、アイテム番号を返します。
構文	REM A%は戻り値を代入するための整数型変数です。 A% = GET_SCREENITEM
解説	タップされたアイテムを検出し、アイテム番号を戻り値として返します。どのアイテムをタップされていない場合、0 が返されます。 このコマンドは、タップされたアイテムを検出するため、継続してコールしなければいけません。
使用例	A%=0 WHILE (A% = 0) 、タップ待ちループ A% = GET_SCREENITEM WEND PRINT "タップされたアイテム", A%

SET_SCREENITEMS

適用機種	8500/8700
目的	タッチスクリーンアイテムのディスプレイモードとサイズを設定します。
構文	SET_SCREENITEMS (mode%, total_item%, item\$)
解説	引数 mode% には、アイテムがタッチされた時のディスプレイモードを示す値を指定します。

mode%	説明	
0	ITEM_NORMAL	通常表示
1	ITEM_REVERSE	反転表示

引数 total_item% には、アイテムの総数を指定します。
引数 item\$ には、アイテムのサイズ情報を下記のルールに従って指定します。

item\$	説明
x%	左上頂点の x 座標
y%	左上頂点の y 座標
size_x%	アイテムの幅をドット単位で指定
size_y%	アイテムの高さをドット単位で指定
CHR\$(13)	アイテムの区切りキャリッジリターン

このコマンドは、あくまでもタッチスクリーンアイテムのサイズ (エリア) を指定するもので、アイテムのイメージ表示などとは行いません。実際のアイテムイメージは、SHOW_IMAGE コマンドでアイコンやホウライイメージを描画してください。

使用例	Itemstr\$ = CHR\$(5) + CHR\$(125) + CHR\$(70) + CHR\$(20) + CHR\$(13) Itemstr\$ = Itemstr\$ + CHR\$(85) + CHR\$(125) + CHR\$(70) + CHR\$(20) + CHR\$(13) SET_SCREENITEMS (1, 2, ITEMSTR\$)
関連項目	SHOW_IMAGE

SET_SIGNAREA

適用機種	8500/8700
目的	サインキャプチャエリアを指定します。
構文	SET_SIGNAREA (x1%, y1%, x2%, y2%)
解説	引数 x1%, y1% には、サインキャプチャエリアの左上頂点座標 (x, y) を指定します。 引数 x2%, y2% には、サインキャプチャエリアの右下終点座標 (x, y) を指定します。 サインキャプチャエリアの指定が終われば、1-ダミーはスタイラスを使ってエリア内で自由に描画ができます。但し、指定されたエリア以外は対象外です。
使用例	SET_SIGNAREA (8, 8, 150, 100)

5.17. フォントコメント

ターミナルには、システムフォントの他に日本語フォントを含むユーザ・フォントファイルを 1 つだけロードすることが可能です。
下記の表に各ターミナルのディスプレイサイズ及び各フォントで表示可能な桁数・行数を示します。

ターミナル機種	解像度	シグマ・ビットフォント		ダブル・ビットフォント (日本語)	
		6x8 フォント	8x16 フォント	12x12 フォント	16x16 フォント
8000	100 x 64 ドット	16 桁 x8 行	12 桁 x4 行	8 桁 x7 行	6 桁 x4 行
8300	128 x 64 ドット	20 桁 x8 行	15 桁 x4 行	10 桁 x5 行	7 桁 x4 行
8200/8400	160 x 160 ドット	26 桁 x18 行	20 桁 x9 行	13 桁 x12 行	10 桁 x9 行
8500/8700	160 x 160 ドット	26 桁 x19 行	20 桁 x9 行	13 桁 x12 行	10 桁 x9 行

GET_LANGUAGE

適用機種	ALL
目的	フォント及び言語設定を取得します。
構文	REM A%は戻り値を代入するための整数型変数です。 A% = GET_LANGUAGE
解説	フォント及び言語設定を取得し、戻り値として返します。

戻り値 A%	フォント	コード・ポイント
0	システムフォント	---
1	トラディショナル中国語フォント 16x16	Big-5
2	シンプル中国語フォント 12x12	GB 2-8
3	シンプル中国語フォント 16x16	GB 2-8
4	韓国語フォント	---
5	日本語フォント 16x16	---
6	ハングル語フォント	---
7	ポーランド語フォント	---
8	ロシア語フォント	---
9	トラディショナル中国語フォント 12x12	Big-5
10	予約	---
11	シンプル中国語フォント 12x12	GB 2-8
12	日本語フォント 12x12	---
16	英語フォント	MS-DOS 3-8 437
17	カタラン語	MS-DOS 3-8 863
18	ハングル語	MS-DOS 3-8 862
19	マルチンガル語 (ラテン I)	MS-DOS 3-8 850
20	ノルウェー語 (北欧)	MS-DOS 3-8 865
21	ポルトガル語	MS-DOS 3-8 860
22	ロシア語 (キル)	Windows 3-8 1251
23	ラテン II (スラブ語)	MS-DOS 3-8 852
24	中央ヨーロッパ ラテン II (ポーランド語)	Windows 3-8 1250
25	トルコ語	MS-DOS 3-8 857
26	ラテン II (スロバキア語)	---
27	Windows 1250	---
28	ISO-28592 (ラテン 2)	ISO 8859-2
29	IBM ラテン II	---
30	ギリシア語	MS-DOS 3-8 737
31	ラテン I	Windows 3-8 1252
32	ギリシア語	Windows 3-8 1253

使用例 Language% = GET_LANGUAGE

GET_LANGUAGE

適用機種 ALL

目的 マルチング-ジ フォントの使用言語を設定します。

構文 SET_LANGUAGE (n%)

解説 引数 n%には、設定したいフォントに対応する値を指定します。
マルチング-ジ フォントがダウンロードされていない場合、このコマンドはエラーとなります。

n%	フォント	コード値
16	英語フォント	MS-DOS コード値 437
17	カタ フランス語	MS-DOS コード値 863
18	ハブ ライ語	MS-DOS コード値 862
19	マルチンガ ル ラテン I	MS-DOS コード値 850
20	ノデ ィック (北欧)	MS-DOS コード値 865
21	ホ ルトガ ル語	MS-DOS コード値 860
22	ロシア語 (キル)	Windows コード値 1251
23	ラテン II (スラブ 語)	MS-DOS コード値 852
24	中央ヨーロッパ ラテン II (ホ -ランド 語)	Windows コード値 1250
25	トルコ語	MS-DOS コード値 857
26	ラテン II (スロバ イ語)	---
27	Windows 1250	---
28	ISO-28592 (ラテン 2)	ISO 8859-2
29	IBM ラテン II	---
30	ギリシア語	MS-DOS コード値 737
31	ラテン I	Windows コード値 1252
32	ギリシア語	Windows コード値 1253

使用例 SET_LANGUAGE (17) 、カタ フランス語

SELECT_FONT

適用機種 ALL

目的 フォントサイズを設定します。

構文 SELECT_FONT (font%)

解説 引数 font%には、設定したいフォントサイズに対応する値を指定します。

font%	説明 (ダブルバイト12ビットフォントがダウンロードされている場合)
1	6×8ビットフォント (かを除くシングルバイトキャラクタに対応)
2	8×16ビットフォント (かを含むシングルバイトキャラクタに対応)
3	予約
4	6×12ビットフォント (かを含むシングルバイトキャラクタに対応) 12×12ビットフォント (漢字などダブルバイトキャラクタに対応)
5	使用不可

font%	説明 (ダブルバイト16ビットフォントがダウンロードされている場合)
1	6×8ビットフォント (かを除くシングルバイトキャラクタに対応)
2	8×16ビットフォント (かを含むシングルバイトキャラクタに対応) 16×16ビットフォント (漢字などダブルバイトキャラクタに対応)
3	予約
4	6×12ビットフォント (かを含まないシングルバイトキャラクタに対応)
5	使用不可

使用例 SELECT_FONT (2)

5.18. メリコマソ

ターミナルが搭載しているフラッシュメモリ/ SRAM/ SD カード を操作するためのコマンド を説明します。各ターミナルの搭載メモリは、下記の通りです。

ターミナル機種	フラッシュメモリ	SRAM	SD カード
8000	2MB	2MB, 4MB	×
8200	8MB	4MB, 8MB	○
8300	2MB	2MB, 6MB, 10MB	×
8400	4MB	4MB, 16MB	○
8500	2MB	2MB, 7MB, 10MB	×
8700	8MB	4MB, 16MB	○

フラッシュメモリ

フラッシュメモリは、幾つかのメモリバンクに分割されており、各メモリバンクは 64KB となります。

ターミナル	メモリバンク
8000/8300/8500	2MB の場合、32 メモリバンクに分割
8400	4MB の場合、64 メモリバンクに分割
8200/8700	8MB の場合、128 メモリバンクに分割

8000/8300/8400/8500

カーネルが 2 バンクを占有し、システムがアプリケーション設定などのデータ保存のために 1 バンク (0xF60000~0xF6FFFF) を確保しています。残りのバンクは、ユーザプログラムやフォントファイルで使用可能です。フラッシュメモリは、不揮発性であるため、同一バンクへの書き込みを行う場合は、先に消去を行わなければいけません。また、メモリバンクは、さらに 256 セクタに分割され、各セクタは、255 バイト長となります。

注意

- 256 セクタまでを保存できます。フラッシュメモリは、バンク単位でしか消去が行えないため、0xF60000~0xF6FFFF に保存されたデータが書き込みの前に全て消去されます。
- 8400 シリーズターミナルでは、システムが将来使用のため 6 バンク (0xF00000~0xF5FFFF) を確保しています。

8200/8700

カーネルが 22 バンクを占有し、システムがアプリケーション設定などのデータ保存のために 1 バンク (0xF60000~0xF6FFFF) を確保しています。残りのバンクは、ユーザプログラムやフォントファイルで使用可能です。

プログラム	フラッシュメモリアドレス
ユーザプログラム	0XC00000~0xDFFFFF
カーネル	0xE00000~0xF5FFFF
ブートローダー	0xFF0000~0xFFFFF

SRAM

ファイルシステムは、ユーザデータを SRAM に保存します。SRAM は、バックアップバッテリーによってバックアップされていますが、バッテリーの消耗により、データを消失する可能性があります。特にターミナルを長期間使用しない場合は、その前に SRAM 内のデータを PC へ必ずアップロードするようにしてください。

MEMORY_INFORMATION

適用機種	ALL
目的	メモリ容量を取得します。
構文	REM R%は戻り値を代入するための整数型変数です。 R% = MEMORY_INFORMATION(n%)
解説	引数 n%には、取得したいメモリ情報に対応する値を 1~6 の範囲で指定します。範囲外の値が指定された場合は、戻り値として-1 が返され、指定されたメモリタイプが存在しない場合は、0 が返されます。

n%	説明	戻り値の単位
1	SRAM (ハートメモリ) 容量	キロバイト
2	SRAM (プログラムメモリ) 容量	キロバイト
3	SRAM 空き容量	キロバイト
4	フラッシュメモリ容量	キロバイト
5	SDカード容量	メガバイト
6	SDカード 空き容量	メガバイト

使用例	PRINT "Free memory = ", MEMORY_INFORMATION(3) ` SRAM 残りメモリ
関連項目	FREE_MEMORY, RAM_SIZE, ROM_SIZE, SD_SIZE, SE_FREE_MEMORY

FLASH_READ\$

適用機種	ALL
目的	フラッシュメモリ (メモリアドレス 0xF60000~0xF6FFFF) からデータを読み出します。
構文	REM A\$は戻り値を代入するための文字列型変数です。 A\$ = FLASH_READ(n%)
解説	引数 n%には、読み出したいバイト番号を 1~256 の範囲で指定します。 戻り値として、読み出したデータが返されます。
使用例	A\$ = FLASH_READ\$(3) ` バイト番号 3

FLASH_WRITE

適用機種	ALL
目的	フラッシュメモリ (メモリ 0xF60000~0xF6FFFF) にデータを書き込みます。
構文	REM A%は戻り値を代入するための整数型変数です。 A% = FLASH_WRITE (n%, a\$)
解説	引数 n%には、書き込みたいアドレス番号を 1~256 の範囲で指定します。 引数 a\$には、書き込みたいデータを指定します。 結果が戻り値として返されます。戻り値の意味は、下記を参照ください。

戻り値	説明
1	書き込み成功
-1	空き容量不足 (BASIC プログラムサイズが大き過ぎます)
-2	消去コマンドエラー
-3	引数 (アドレス番号) が範囲外
-4	書き込み異常 (消去が行われていないなど)

フラッシュメモリへの書き込みを行う場合は、最初にフラッシュメモリの消去を行う必要があります。書き込みを開始する前に、必ず下記のアドレス番号 0、データ "ERASE" の FLASH_WRITE コマンドをコールしてください。

```
Err% = FLASH_WRITE (0, "ERASE")
```

使用例	Err% = FLASH_WRITE (0, "ERASE")	、フラッシュメモリ消去
	Err% = FLASH_WRITE (1, "Data #1")	、アドレス 1 へ書き込み

ROM_SIZE

適用機種	ALL
目的	フラッシュメモリ容量を取得します。
構文	REM A%は戻り値を代入するための整数型変数です。 A% = ROM_SIZE
解説	フラッシュメモリ容量が戻り値として返されます。単位はバイトです。
使用例	PRINT "Flash size = ", ROM_SIZE
関連項目	MEMORY_INFORMATION (4)

FREE_MEMORY

適用機種	ALL
目的	SRAM の空き容量を取得します。
構文	REM A%は戻り値を代入するための整数型変数です。 A% = FREE_MEMORY
解説	SRAM の空き容量が戻り値として返されます。単位はバイトです。
使用例	PRINT "Free memory size = ", FREE_MEMORY
関連項目	MEMORY_INFORMATION(3)

RAM_SIZE

適用機種	ALL
目的	SRAM 容量を取得します。
構文	REM A%は戻り値を代入するための整数型変数です。 A% = RAM_SIZE
解説	SRAM 容量が戻り値として返されます。単位はバイトです。
使用例	PRINT "SRAM size = ", RAM_SIZE
関連項目	MEMORY_INFORMATION(1)

SD_FREE_MEMORY

適用機種	8200/8400/8700
目的	SD カード の空き容量を取得します。
構文	REM A%は戻り値を代入するための整数型変数です。 A% = SD_FREE_MEMORY
解説	SD カード の空き容量が戻り値として返されます。単位はバイトです。
使用例	PRINT "SD Free memory size = ", SD_FREE_MEMORY
関連項目	MEMORY_INFORMATION(6)

SD_SIZE

適用機種	8200/8400/8700
目的	SD カード 容量を取得します。
構文	REM A%は戻り値を代入するための整数型変数です。 A% = SD_SIZE
解説	SD カード 容量が戻り値として返されます。単位はバイトです。
使用例	PRINT "SD size = ", SD_SIZE
関連項目	MEMORY_INFORMATION(5)

5.19. ファイル操作コマンド

ファイルは、下記の 2 種類のファイル構造を継承しており、用途に応じて使い分けが可能です。

トランザクションファイル (DAT ファイル)

通常、トランザクションデータの保存に利用されるシーケンシャルファイルです。

DBF ファイル

インデックス付のデータファイルです。DBF ファイルは、データを保存する DBF ファイルとキーによるソートを実行するための IDX ファイルで構成されますが、プログラマは、IDX ファイルを意識することなく DBF ファイルを利用して、容易に検索・ソートを実行することができます。

5.19.1. トランザクションファイル (DAT ファイル)

トランザクションファイルは、FIFO ストラクチャを採用したシーケンシャルファイルです。通常、トランザクションデータの保存に利用されます。

参考

⚠️ コード長は、255 バイトまで定義可能です。

⚠️ Cipher Basic では、6 つまでのトランザクションファイルを定義可能です。

DEL_TRANSACTION_DATA

適用機種	ALL
目的	トランザクションファイル 1 のトランザクションデータを指定件数削除します。
構文	DEL_TRANSACTION_DATA (n%)
解説	引数 n%には、何件のトランザクションデータを削除するかを指定します。正の数値を指定した場合は、先頭 (一番古いデータ) から指定件数が削除され、負の数値を指定した場合は、後方 (一番新しいデータ) から指定件数が削除されます。
使用例	<pre>PRINT "Discard the latest transaction?(Y/N)" ... LOOP: KeyData\$ = INKEY\$ IF KeyData\$ = "" THEN GOTO LOOP ELSE IF KeyData\$ = "Y" THEN DEL_TRANSACTION_DATA (-1) END IF ...</pre>
関連項目	DEL_TRANSACTION_DATA_EX, EMPTY_TRANSACTION


DEL_TRANSACTION_DATA_EX

適用機種	ALL
目的	指定のトランザクションファイルのトランザクションデータを指定件数削除します。
構文	<code>DEL_TRANSACTION_DATA_EX(file%, n%)</code>
解説	<p>引数 <i>file%</i> には、トランザクションファイル番号を 1~6 の範囲で指定します。トランザクションファイル番号 1 を指定した場合、先の <code>DEL_TRANSACTION_DATA</code> コマンドの実行結果と同じになります。</p> <p>引数 <i>n%</i> には、何件のトランザクションデータを削除するかを指定します。正の数値を指定した場合は、先頭 (一番古いデータ) から指定件数が削除され、負の数値を指定した場合は、後方 (一番新しいデータ) から指定件数が削除されます。</p>
使用例	<pre>FileNo% = 5 PRINT "Discard the latest transaction?(Y/N)" ... LOOP: KeyData\$ = INKEY\$ IF KeyData\$ = "" THEN GOTO LOOP ELSE IF KeyData\$ = "Y" THEN DEL_TRANSACTION_DATA_EX(FileNo%, -1) END IF ...</pre>
関連項目	<code>DEL_TRANSACTION_DATA</code> , <code>EMPTY_TRANSACTION_EX</code>

EMPTY_TRANSACTION

適用機種	ALL
目的	トランザクションファイル 1 のトランザクションデータを全て削除します。
構文	<code>EMPTY_TRANSACTION</code>
解説	トランザクションファイル 1 のトランザクションデータを全て削除します。

注意


 このコマンドを無条件にプログラムの最初でコールすると、トランザクションデータが全て消去されます。バッテリー交換などを行った場合、プログラムが最初から実行されることとなりますので、コーディングには十分注意してください。

使用例	<pre>PRINT "Remove all the transaction data?(Y/N)" ... LOOP: KeyData\$ = INKEY\$ IF KeyData\$ = "" THEN GOTO LOOP ELSE IF KeyData\$ = "Y" THEN EMPTY_TRANSACTION END IF ...</pre>
関連項目	<code>DEL_TRANSACTION_DATA</code> , <code>EMPTY_TRANSACTION_EX</code>

EMPTY_TRANSACTION_EX

適用機種	ALL
目的	指定のトランザクションファイルのトランザクションデータを全て削除します。
構文	EMPTY_TRANSACTION_EX(<i>file</i> %)
解説	引数 <i>file</i> % には、トランザクションファイル番号を 1~6 の範囲で指定します。トランザクションファイル番号 1 を指定した場合、先の EMPTY_TRANSACTION マクロの実行結果と同じになります。

注意

 このマクロを無条件にプログラムの最初でコールすると、トランザクションデータが全て消去されます。バッテリー交換などを行った場合、プログラムが最初から実行されることとなりますので、コーディングには十分注意してください。

使用例

```
PRINT "Remove all the transaction data?(Y/N)"
...
LOOP:
    KeyData$ = INKEY$
    IF KeyData$ = "" THEN
        GOTO LOOP
    ELSE IF KeyData$ = "Y" THEN
        EMPTY_TRANSACTION(6)
    END IF
...
```

関連項目 DEL_TRANSACTION_DATA_EX, EMPTY_TRANSACTION

GET_TRANSACTION_DATA\$

適用機種	ALL
目的	トランザクションファイル 1 からトランザクションデータを読み出します。
構文	REM A\$ は戻り値を代入するための文字型変数です。 A\$ = GET_TRANSACTION_DATA(<i>n</i> %)
解説	引数 <i>n</i> % には、先頭 (一番古いデータ) から何番目のトランザクションデータを読み出すのかを指定します。読み出したトランザクションデータは、戻り値として返されます。

使用例

```
...
WHILE (TRANSACTION_COUNT > 0)
    TransactionData$ = GET_TRANSACTION_DATA$(1)
    WRITE_COM(1, TransactionData$)
    DEL_TRANSACTION_DATA(1)
WEND
```

関連項目 GET_TRANSACTION_DATA_EX\$, SAVE_TRANSACTION, UPDATE_TRANSACTION

GET_TRANSACTION_DATA_EX\$

適用機種	ALL
目的	指定のトランザクションファイルからトランザクションデータを読み出します。
構文	<code>GET_TRANSACTION_DATA_EX(file%, n%)</code>
解説	引数 <i>file%</i> には、トランザクションファイル番号を 1~6 の範囲で指定します。トランザクションファイル番号 1 を指定した場合、先の <code>GET_TRANSACTION_DATA\$</code> の実行結果と同じになります。 引数 <i>n%</i> には、先頭 (一番古いデータ) から何番目のトランザクションデータを読み出すのかを指定します。 読み出したトランザクションデータは、戻り値として返されます。
使用例	<pre>... WHILE (TRANSACTION_COUNT_EX(FileNo%) > 0) TransactionData\$ = GET_TRANSACTION_DATA\$(FileNo%, 1) WRITE_COM(1, TransactionData\$) DEL_TRANSACTION_DATA_EX(FileNo%) WEND</pre>
関連項目	<code>GET_TRANSACTION_DATA\$</code> , <code>SAVE_TRANSACTION_EX</code> , <code>UPDATE_TRANSACTION_EX</code>

SAVE_TRANSACTION

適用機種	ALL
目的	トランザクションファイル 1 にトランザクションデータを追加します。
構文	<code>SAVE_TRANSACTION(data\$)</code>
解説	引数 <i>data\$</i> には、追加したいトランザクションデータを指定します。
使用例	<pre>ON READER(1) GOSUB BcrData_1 ... Bcr_Data_1: Data\$ = GET_READER_DATA\$(1) PRINT Data\$ SAVE_TRANSACTION(Data\$) IF GET_FILE_ERROR <> 0 THEN PRINT "Transaction not saved" RETURN</pre>
関連項目	<code>GET_TRANSACTION_DATA\$</code> , <code>SAVE_TRANSACTION_EX</code> , <code>UPDATE_TRANSACTION</code> , <code>GET_FILE_ERROR</code>

SAVE_TRANSACTION_EX

適用機種	ALL
目的	指定のトランザクションファイルのトランザクションデータを追加します。
構文	<code>SAVE_TRANSACTION_EX(file%, data%)</code>
解説	引数 <i>file%</i> には、トランザクションファイル番号を 1~6 の範囲で指定します。トランザクションファイル番号 1 を指定した場合、先の <code>SAVE_TRANSACTION</code> マクロの実行結果と同じになります。 引数 <i>data%</i> には、追加したいトランザクションデータを指定します。
使用例	<pre>ON READER(1) GOSUB BcrData_1 ... Bcr_Data_1: Data\$ = GET_READER_DATA\$(1) PRINT Data\$ SAVE_TRANSACTION_EX(FileNo%, Data\$) IF GET_FILE_ERROR <> 0 THEN PRINT "Transaction not saved" RETURN</pre>
関連項目	<code>GET_TRANSACTION_DATA_EX\$, SAVE_TRANSACTION, UPDATE_TRANSACTION_EX, GET_FILE_ERROR</code>

TRANSACTION_COUNT

適用機種	ALL
目的	トランザクションファイル 1 に保存されているトランザクションデータ件数を取得します。
構文	REM A%は戻り値を代入するための整数型変数です。 <code>A% = TRANSACTION_COUNT</code>
解説	トランザクションファイル 1 に保存されているトランザクションデータ件数が、戻り値として返されます。
使用例	<pre>... DataCount% = TRANSACTION_COUNT CLS PRINT DataCount%, "Transaction data is saved."</pre>
関連項目	<code>TRANSACTION_COUNT_EX</code>

TRANSACTION_COUNT_EX

適用機種	ALL
目的	指定のトランザクションファイルに保存されているトランザクションデータ件数を取得します。
構文	REM A%は戻り値を代入するための整数型変数です。 <code>A% = TRANSACTION_COUNT_EX(file%)</code>
解説	引数 <i>file%</i> には、トランザクションファイル番号を 1~6 の範囲で指定します。指定したトランザクションファイルに保存されているトランザクションデータ件数が、戻り値として返されます。トランザクションファイル番号 1 を指定した場合、先の <code>SAVE_TRANSACTION</code> マクロの実行結果と同じになります。
使用例	<pre>... DataCount_1: DataCount% = TRANSACTION_COUNT_EX(1) CLS PRINT DataCount%, "Data in transaction file 1."</pre>
関連項目	<code>TRANSACTION_COUNT</code>

UPDATE_TRANSACTION

適用機種	ALL
目的	トランプ クイソファイル 1 のトランプ クイソデータを上書き更新します。
構文	UPDATE_TRANSACTION(<i>n</i> %, <i>data</i> %)
解説	引数 <i>n</i> %には、先頭 (一番古いデータ) から何番目のトランプ クイソデータを上書き更新するのかを指定します。 引数 <i>data</i> %には、上書き更新したいトランプ クイソデータを指定します。
使用例	... UpdateTransaction: UPDATE_TRANSACTION(Data%) RETURN
関連項目	GET_TRANSACTION_DATA\$, SAVE_TRANSACTION, UPDATE_TRANSACTION_EX

UPDATE_TRANSACTION_EX

適用機種	ALL
目的	指定のトランプ クイソファイルの先頭のトランプ クイソデータを上書き更新します。
構文	UPDATE_TRANSACTION_EX(<i>file</i> %, <i>n</i> %, <i>data</i> %)
解説	引数 <i>file</i> %には、トランプ クイソファイル番号を 1~6 の範囲で指定します。トランプ クイソファイル番号 1 を指定した場合、先の SAVE_TRANSACTION コマンドの実行結果と同じになります。 引数 <i>n</i> %には、先頭 (一番古いデータ) から何番目のトランプ クイソデータを上書き更新するのかを指定します。 引数 <i>data</i> %には、上書き更新したいトランプ クイソデータを指定します。
使用例	... UpdateTransaction: UPDATE_TRANSACTION_EX(1, Num%, Data%) RETURN
関連項目	GET_TRANSACTION_DATA_EX\$, SAVE_TRANSACTION_EX, UPDATE_TRANSACTION

5.19.2. DBFファイルとIDXファイル

DBF ファイルは、IDX ファイルと構成されるインデックス付データベースファイルです。このファイルを利用することで、キーを元に素早く検索を行い、目的のレコードへアクセスすることが可能になります。

DBF ファイル

DBF ファイルは、固定レコード長を持つ、実際のデータを保存するためのファイルです。

IDX ファイル

IDX ファイル(インデックスファイル)は、各 DBF ファイル(DBF#1~#5)に対して、指定されたインデックスキーに基づいてレコード番号を昇順に記録したファイルです。

【例】

5 レコードが保存されている DBF ファイルがあるとします。インデックスファイル 1 をセット 3・長さ 2、インデックスファイル 2 をセット 6・長さ 2 とした場合、DBF ファイルと各インデックスファイルの相対関係は、下記のようになります。

番号	DBF ファイル	インデックスファイル 1	インデックスファイル 2
1	0005-15	5 (01)	5 (05)
2	0007-13	3 (03)	3 (08)
3	0003-08	1 (05)	4 (10)
4	0009-10	2 (07)	2 (13)
5	0001-05	4 (09)	1 (15)

* () 内はインデックスキー

参考

- レコード長は、250 バイトまでに定義可能です。
- DBF ファイルは、5 つまで定義可能です。
- インデックスキーは、3 つまで定義可能です。
- SD カードに対応したターミナルであれば、SD カード上に DBF ファイルを作成することも可能です。

ADD_RECORD

適用機種	ALL
目的	指定の DBF ファイルにレコードを追加します。
構文	ADD_RECORD(<i>file%</i> , <i>data%</i>)
解説	引数 <i>file%</i> には、DBF ファイル番号を 1~5 の範囲で指定します。 引数 <i>data%</i> には、追加したいレコード(データ)を指定します。
使用例	ON COM(1) GOSUB HostCommand ... HostCommand: Cmd\$ = READ_COM\$(1) CmdIdentifier\$ = LEFT\$(Cmd\$, 1) DBFNum% = VAL(MID\$(Cmd\$, 2, 1)) CardID\$ = RIGHT\$(Cmd\$, LEN(Cmd\$) - 2) IF CmdIdentifier\$ = "+" THEN ADD_RECORD(DBFNum%, CardID\$) ELSE ...

DEL_RECORD

適用機種 ALL

目的 指定の DBF ファイルからレコードを削除します。

構文 DEL_RECORD(*file%* [, *index%*])

解説 引数 *file%* には、DBF ファイル番号を 1~5 の範囲で指定します。
引数 *index%* には、インデックスファイル番号 (IDX ファイル) を 1~3 の範囲で指定します。このオプションは省略可能で、省略した場合は、DBF ファイルのプライマリ・インデックスが適用されます。
このコマンドがコールされると、ファイルの特定のレコードが削除されます。

例えば、DBF ファイル 1 が「011-231」、「120-117」、「043-010」、「067-150」の 4 つのレコードから構成され、インデックス 1 及び 2 が下記のように定義されているとします。

インデックス 1 : 1 桁目から 3 桁
インデックス 2 : 5 桁目から 3 桁

DBF ファイル及びインデックスファイル、それぞれ下記のように最後のレコードファイル・インデックス (→の位置) がある場合、DEL_RECORD (1) は 067-150、DEL_RECORD (1, 1) は 120-117、DEL_RECORD (1, 2) は 011-231 を削除します。

	DBF ファイル 1		インデックスファイル 1		インデックスファイル 2
	011-231		011-231		043-010
	120-117		043-010		120-117
	043-010		067-150		067-150
→	067-150	→	120-117	→	011-231

使用例

```
ON COM(1) GOSUB HostCommand
...
HostCommand:
  Cmd$ = READ_COM$(1)
  CmdIdentifier$ = LEFT$(Cmd$, 1)
  DBFNum% = VAL(MID$(Cmd$, 2, 1))
  IDXNum% = VAL(MID$(Cmd$, 3, 1))
  CardID$ = RIGHT$(Cmd$, LEN(Cmd$) - 2)
  IF CmdIdentifier$ = "-" THEN
    DEL_RECORD(DBFNum%, IDXNum%)
  ELSE
    ...
```

EMPTY_FILE

適用機種	ALL
目的	指定の DBF ファイルから全レコードを削除します。
構文	<code>EMPTY_FILE (file%)</code>
解説	引数 <i>file%</i> には、DBF ファイル番号を 1~5 の範囲で指定します。
使用例	<pre>ON COM(1) GOSUB HostCommand ... HostCommand: Cmd\$ = READ_COM\$(1) CmdIdentifier\$ = LEFT\$(Cmd\$, 1) DBFNum% = VAL(MID\$(Cmd\$, 2, 1)) CardID\$ = RIGHT\$(Cmd\$, LEN(Cmd\$) - 2) IF CmdIdentifier\$ = "!" THEN EMPTY_FILE(DBFNum%) ELSE ...</pre>

FIND_RECORD

適用機種	ALL
目的	指定の DBF ファイルのレコードを検索します。
構文	<p>REM A%は戻り値を代入するための整数型変数です。</p> <p><code>A% = FIND_RECORD (file%, index%, key\$)</code></p>
解説	<p>引数 <i>file%</i> には、DBF ファイル番号を 1~5 の範囲で指定します。</p> <p>引数 <i>index%</i> には、インデックスファイル番号 (IDX ファイル) を 1~3 の範囲で指定します。</p> <p>引数 <i>key\$</i> には、検索したいキー (文字列) を指定します。</p> <p>検索の結果、該当レコードが見つかったとインデックスファイルのファイルポインタをそのレコード位置に移動させ、戻り値 1 を返します。該当レコードが無かった場合は、ファイルポインタをキーデータより上に位置する最初のレコード位置に移動させ、戻り値 0 を返します。</p>
使用例	<pre>ON COM(1) GOSUB HostCommand ... HostCommand: Cmd\$ = READ_COM\$(1) CmdIdentifier\$ = LEFT\$(Cmd\$, 1) DBFNum% = VAL(MID\$(Cmd\$, 2, 1)) IDXNum% = VAL(MID\$(Cmd\$, 3, 1)) CardID\$ = RIGHT\$(Cmd\$, LEN(Cmd\$) - 2) IF CmdIdentifier\$ = "?" THEN IF FIND_RECORD(DBFNum%, IDXNum%, VardID\$) = 1 THEN PRINT "Data is found in DBF.", DBFNum% ELSE PRINT "Data is not found in DBF.", DBFNum% END IF ELSE ...</pre>

GET_RECORD

適用機種 ALL

目的 指定の DBF ファイルのレコードを取得します。

構文 REM A\$は戻り値を代入するための文字列型変数です。
A\$ = GET_RECORD(*file%* [, *index%*])

解説 引数 *file%* には、DBF ファイル番号を 1~5 の範囲で指定します。
引数 *index%* には、インデックスファイル番号 (IDX ファイル) を 1~3 の範囲で指定します。このオプションは省略可能で、省略した場合は、DBF ファイルのオリジナルデータベースが適用されます。

このコマンドをコールすると、ファイル内の指定位置のレコードが戻り値として返されます。

使用例

```
ON COM(1) GOSUB BcrData_1
...
BcrData_1:
    BEEP(2000, 5)
    ID$ = GET_READER_DATA$(1)
    IF FIND_RECORD(DBFNum%, IDXNum%, ID$) = 1 THEN
        Data$ = GET_RECORD$(DBFNum%, IDXNum%)
        Item$ = MID$(Data$, LEN(Data$)-IDLeng%-ItemLeng%)
        Note$ = RIGHT$(Data$, LEN(Data$)-IDLeng%-ItemLeng%)
        LOCATE 1, 1
        PRINT "ID      :", Data$
        LOCATE 2, 1
        PRINT "Item   :", Item$
        LOCATE 3, 1
        PRINT "Note   :", Note$
    ELSE
        ...
```

GET_RECORD_NUMBER

適用機種 ALL

目的 指定の DBF ファイルのレコード番号を取得します。

構文 REM A%は戻り値を代入するための整数型変数です。
A% = GET_RECORD_NUMBER(*file%* [, *index%*])

解説 引数 *file%* には、DBF ファイル番号を 1~5 の範囲で指定します。
引数 *index%* には、インデックスファイル番号 (IDX ファイル) を 1~3 の範囲で指定します。このオプションは省略可能で、省略した場合は、DBF ファイルのオリジナルデータベースが適用されます。

このコマンドをコールすると、ファイル内の指定位置のレコード番号が戻り値として返されます。

使用例

```
A% = GET_RECORD_NUMBER(1, 1)
```

MOVE_TO

適用機種	ALL
目的	指定の DBF ファイルのファイルポインタを移動させます。
構文	<code>MOVE_TO(file%, index%, record_number%)</code>
解説	引数 <i>file%</i> には、DBF ファイル番号を 1~5 の範囲で指定します。 引数 <i>index%</i> には、インデックスファイル番号 (IDX ファイル) を 1~3 の範囲で指定します。このオプションは省略可能で、省略した場合は、DBF ファイルのオリジナルレコードが適用されます。 引数 <i>record_number%</i> には、移動させたいファイルポインタ位置を指定します。
使用例	<code>MOVE_TO(1, 1, 20)</code>

MOVE_TO_NEXT

適用機種	ALL
目的	指定の DBF ファイルのファイルポインタを 1 つ先に進めます。
構文	<code>MOVE_TO_NEXT(file%, index%)</code>
解説	引数 <i>file%</i> には、DBF ファイル番号を 1~5 の範囲で指定します。 引数 <i>index%</i> には、インデックスファイル番号 (IDX ファイル) を 1~3 の範囲で指定します。このオプションは省略可能で、省略した場合は、DBF ファイルのオリジナルレコードが適用されます。
使用例	<code>MOVE_TO_NEXT(1, 1)</code>

MOVE_TO_PREVIOUS

適用機種	ALL
目的	指定の DBF ファイルのファイルポインタを 1 つ前に戻します。
構文	<code>MOVE_TO_PREVIOUS(file%, index%)</code>
解説	引数 <i>file%</i> には、DBF ファイル番号を 1~5 の範囲で指定します。 引数 <i>index%</i> には、インデックスファイル番号 (IDX ファイル) を 1~3 の範囲で指定します。このオプションは省略可能で、省略した場合は、DBF ファイルのオリジナルレコードが適用されます。
使用例	<code>MOVE_TO_PREVIOUS(1, 1)</code>

RECORD_COUNT

適用機種	ALL
目的	指定の DBF ファイルのレコード数を取得します。
構文	REM A%は戻り値を代入するための整数型変数です。 <code>A% = RECORD_COUNT(file%)</code>
解説	引数 <i>file%</i> には、DBF ファイル番号を 1~5 の範囲で指定します。 このコマンドを実行すると、レコード数が戻り値として返されます。
使用例	<code>Total_Record% = RECORD_COUNT(1)</code>

UPDATE_RECORD

適用機種 ALL

目的 指定の DBF ファイルのレコードを上書き更新します。

構文 UPDATE_RECORD(*file%*, *index%*, *data%*)

解説 引数 *file%*には、DBF ファイル番号を 1~5 の範囲で指定します。
引数 *index%*には、インデックスファイル番号 (IDX ファイル) を 1~3 の範囲で指定します。
引数 *data%*には、上書き更新したいレコード (データ) を指定します。

このコマンドをコールすると、ファイル内の指定位置のレコードが上書き更新されます。

使用例

```
ON COM(1) GOSUB HostCommand
...
HostCommand:
    Cmd$ = READ_COM$(1)
    CmdIdentifier$ = LEFT$(Cmd$, 1)
    DBFNum% = VAL(MID$(Cmd$, 2, 1))
    IDXNum% = VAL(MID$(Cmd$, 3, 1))
    CardID$ = RIGHT$(Cmd$, LEN(Cmd$) - 2)
    IF CmdIdentifier$ = "&" THEN
        UPDATE_RECORD(DBFNum%, IDXNum%, CardID$)
    ELSE
        ...
```

5.19.3. ファイルエラーコード

ファイル操作コマンドに関するエラーコードは、GET_FILE_ERROR コマンドで取得可能です。正しくコマンドが処理された場合は、0 となり、エラーが発生した場合は、0 以外が返されます。

GET_FILE_ERROR

適用機種	ALL
目的	ファイル操作によるエラーコードを取得します。
構文	REM A%は戻り値を代入するための整数型変数です。 A% = GET_FILE_ERROR
解説	ファイル操作によるエラーコードが戻り値として返されます。エラーコードの意味は、下記の通りです。

A%	説明
0	エラー無し
10	十分な空き容量がありません

使用例

```
...  
ADD_RECORD(1, Data$)  
IF (GET_FILE_ERROR = 10) THEN  
    ErrorMessage$ = "No free file space."  
END IF
```

5.20. SDカード

8200/8400/8700 シリーズターミナルは MicroSD カードに対応しており、ユーザーアプリケーションからファイル操作コマンドで直接アクセスすることが可能です。また、USB ケーブルで接続することで、リムーバブルディスク(大容量記憶装置デバイス)として使用することも可能です。詳しくは、各ターミナルの取扱説明書を参照ください。

トランザクションファイル(DATファイル)

- トランザクションファイルは、SD カードのディレクトリ¥BasicRun 下に作成され、「5.19.1. トランザクションファイル(DAT ファイル)」にあるコマンドでアクセスできます。
- トランザクションファイルのサイズは、システムメニューより調整されます。ユーザーアプリケーションで DEL_TRANSACTION_DATA コマンドや DEL_TRANSACTION_DATA_EX コマンドを実行した場合、サイズはすぐにはリセットされません。ユーザーは、システムメニュー「System Menu」...「SD Card Menu」...「Access SD Card」...「Check File Size」より「A:¥BasicRun¥TXACTn.DAT (n=1~6)」のサイズをリセットしなければいけません。

DBFファイルとIDXファイル

- DBF ファイルは、SD カードのディレクトリ¥BasicRun 下に作成され、「5.19.2. DBF ファイルと IDX ファイル」にあるコマンドでアクセスできます。DBF ファイル本体の拡張子は、.DB0 となり、IDX ファイルの拡張子は、.DB1~.DB4 となります。

ファイル操作コマンドに関するエラーコードは、GTE_FILE_ERROR コマンドで取得可能です。エラー無くコマンドが処理された場合は、0 となり、エラーが発生した場合は、0 以外が返されます。

参考

- SD カードは、SRAM と比べて検索速度が非常に遅くなります。商品リストなど検索を必要とするリストファイルは、SD カードではなく、SRAM 上に作成することをお勧めします。

5.20.1. ファイルシステム

ファイルシステムとして、FAT12/FAT16/FAT32 を採用しています。SD カードのフォーマットは、アプリケーションプログラム(C言語)又はシステムメニュー「System Menu」...「SD Card Menu」...「Access SD Card」から行うことができます。下記のように FAT タイプによりカードの容量が決まります。

SD カード 容量	FAT 形式	セクタ数/クラスタ
32MB 以下	FAT12	32
1GB 以下	FAT16	32
2GB 以下	FAT16	64
8GB 以下	FAT32	8

5.20.2. デ ィ レ ク ト リ

SRAM と異なり、SD カード では、階層式ツリー構造によるディレクトリ/サブディレクトリが作成可能です。但し、一部のディレクトリ及びファイルはシステムにより予約されています。下記を参照ください。

予約済みディレクトリ	関係するアプリケーション/機能	説明																																																																															
¥Program	<div>✓ 「System Menu」...「Load Program」</div> <div>✓ 「Program Manager」...「Download」</div> <div>✓ 「Program Manager」...「Activate」</div> <div>✓ 「Kernel Menu」...「Load Program」</div> <div>✓ 「Kernel Menu」...「Kernel Update」</div> <div>✓ UPDATE BASIC ヲソツ</div>	<p>プログラムを保存するためのディレクトリです。</p> <div>✓ Cプログラム(.shx)</div> <div>✓ BASICプログラム(.syn/.ini)</div>																																																																															
¥BasicRun	BASIC ランタイム	<p>BASICプログラムからアクセスされるトランザクションファイル (DATファイル) と DBF ファイルを保存するためのディレクトリです。ファイル名は、下記のように決められています。</p> <table><tr><th colspan="3">トランザクションファイル名 (DAT ファイル)</th></tr><tr><td>トランザクションファイル #1</td><td colspan="2">TXACT1.DAT</td></tr><tr><td>トランザクションファイル #2</td><td colspan="2">TXACT2.DAT</td></tr><tr><td>トランザクションファイル #3</td><td colspan="2">TXACT3.DAT</td></tr><tr><td>トランザクションファイル #4</td><td colspan="2">TXACT4.DAT</td></tr><tr><td>トランザクションファイル #5</td><td colspan="2">TXACT5.DAT</td></tr><tr><td>トランザクションファイル #6</td><td colspan="2">TXACT6.DAT</td></tr><tr><th colspan="3">DBF ファイル名</th></tr><tr><td rowspan="5">DBF ファイル #1</td><td>レコード ファイル本体</td><td>F1.DB0</td></tr><tr><td>システムインデックス</td><td>F1.DB1</td></tr><tr><td>インデックスファイル #1</td><td>F1.DB2</td></tr><tr><td>インデックスファイル #2</td><td>F1.DB3</td></tr><tr><td>インデックスファイル #3</td><td>F1.DB4</td></tr><tr><td rowspan="5">DBF ファイル #2</td><td>レコード ファイル本体</td><td>F2.DB0</td></tr><tr><td>システムインデックス</td><td>F2.DB1</td></tr><tr><td>インデックスファイル #1</td><td>F2.DB2</td></tr><tr><td>インデックスファイル #2</td><td>F2.DB3</td></tr><tr><td>インデックスファイル #3</td><td>F2.DB4</td></tr><tr><td rowspan="5">DBF ファイル #3</td><td>レコード ファイル本体</td><td>F3.DB0</td></tr><tr><td>システムインデックス</td><td>F3.DB1</td></tr><tr><td>インデックスファイル #1</td><td>F3.DB2</td></tr><tr><td>インデックスファイル #2</td><td>F3.DB3</td></tr><tr><td>インデックスファイル #3</td><td>F3.DB4</td></tr><tr><td rowspan="5">DBF ファイル #4</td><td>レコード ファイル本体</td><td>F4.DB0</td></tr><tr><td>システムインデックス</td><td>F4.DB1</td></tr><tr><td>インデックスファイル #1</td><td>F4.DB2</td></tr><tr><td>インデックスファイル #2</td><td>F4.DB3</td></tr><tr><td>インデックスファイル #3</td><td>F4.DB4</td></tr><tr><td rowspan="5">DBF ファイル #5</td><td>レコード ファイル本体</td><td>F5.DB0</td></tr><tr><td>システムインデックス</td><td>F5.DB1</td></tr><tr><td>インデックスファイル #1</td><td>F5.DB2</td></tr><tr><td>インデックスファイル #2</td><td>F5.DB3</td></tr><tr><td>インデックスファイル #3</td><td>F5.DB4</td></tr></table>	トランザクションファイル名 (DAT ファイル)			トランザクションファイル #1	TXACT1.DAT		トランザクションファイル #2	TXACT2.DAT		トランザクションファイル #3	TXACT3.DAT		トランザクションファイル #4	TXACT4.DAT		トランザクションファイル #5	TXACT5.DAT		トランザクションファイル #6	TXACT6.DAT		DBF ファイル名			DBF ファイル #1	レコード ファイル本体	F1.DB0	システムインデックス	F1.DB1	インデックスファイル #1	F1.DB2	インデックスファイル #2	F1.DB3	インデックスファイル #3	F1.DB4	DBF ファイル #2	レコード ファイル本体	F2.DB0	システムインデックス	F2.DB1	インデックスファイル #1	F2.DB2	インデックスファイル #2	F2.DB3	インデックスファイル #3	F2.DB4	DBF ファイル #3	レコード ファイル本体	F3.DB0	システムインデックス	F3.DB1	インデックスファイル #1	F3.DB2	インデックスファイル #2	F3.DB3	インデックスファイル #3	F3.DB4	DBF ファイル #4	レコード ファイル本体	F4.DB0	システムインデックス	F4.DB1	インデックスファイル #1	F4.DB2	インデックスファイル #2	F4.DB3	インデックスファイル #3	F4.DB4	DBF ファイル #5	レコード ファイル本体	F5.DB0	システムインデックス	F5.DB1	インデックスファイル #1	F5.DB2	インデックスファイル #2	F5.DB3	インデックスファイル #3	F5.DB4
トランザクションファイル名 (DAT ファイル)																																																																																	
トランザクションファイル #1	TXACT1.DAT																																																																																
トランザクションファイル #2	TXACT2.DAT																																																																																
トランザクションファイル #3	TXACT3.DAT																																																																																
トランザクションファイル #4	TXACT4.DAT																																																																																
トランザクションファイル #5	TXACT5.DAT																																																																																
トランザクションファイル #6	TXACT6.DAT																																																																																
DBF ファイル名																																																																																	
DBF ファイル #1	レコード ファイル本体	F1.DB0																																																																															
	システムインデックス	F1.DB1																																																																															
	インデックスファイル #1	F1.DB2																																																																															
	インデックスファイル #2	F1.DB3																																																																															
	インデックスファイル #3	F1.DB4																																																																															
DBF ファイル #2	レコード ファイル本体	F2.DB0																																																																															
	システムインデックス	F2.DB1																																																																															
	インデックスファイル #1	F2.DB2																																																																															
	インデックスファイル #2	F2.DB3																																																																															
	インデックスファイル #3	F2.DB4																																																																															
DBF ファイル #3	レコード ファイル本体	F3.DB0																																																																															
	システムインデックス	F3.DB1																																																																															
	インデックスファイル #1	F3.DB2																																																																															
	インデックスファイル #2	F3.DB3																																																																															
	インデックスファイル #3	F3.DB4																																																																															
DBF ファイル #4	レコード ファイル本体	F4.DB0																																																																															
	システムインデックス	F4.DB1																																																																															
	インデックスファイル #1	F4.DB2																																																																															
	インデックスファイル #2	F4.DB3																																																																															
	インデックスファイル #3	F4.DB4																																																																															
DBF ファイル #5	レコード ファイル本体	F5.DB0																																																																															
	システムインデックス	F5.DB1																																																																															
	インデックスファイル #1	F5.DB2																																																																															
	インデックスファイル #2	F5.DB3																																																																															
	インデックスファイル #3	F5.DB4																																																																															
¥AG¥DBF ¥AG¥DAT ¥AG¥EXPORT ¥AG¥IMPORT	アプリケーション エレメント (AG) (日本では、ご希望のユーザー様にのみ配布)	アプリケーション エレメントで利用する DAT, DBF, ルックアップ ファイルを保存するフォルダです。																																																																															

5.20.3. ファイル名


ファイル名は、8.3 書式 (ファイル名.拡張子) を採用しています。ファイル名には、下記に示す記号以外が使用可能です。

“ * + , : ; < = > ? | []

- 8 文字を超えるファイル名は、自動的に 8 文字に丸められます。
- SD カード をムバブルディスク (大容量記憶装置デバイス) として使用する場合は、最大 255 文字のファイル名を使用することができます。例えば、PC で SD カード 上に [123456789.TXT] というファイルを作成した場合、ターミナルから直接そのファイルにアクセスすると、ファイル名は、「123456~1.txt」に丸められます。
- ASCII 文字以外を使用したファイル名をターミナル上で正しく表示したい場合は、そのファイル名に使用されている文字を含む適切なフォントファイルをターミナルにダウンロードしておく必要があります。
- ファイル名に大文字/小文字の区別はありません。下記のファイル名は、全て同じファイルとして認識されます。

```
DATAFILE.TXT  
DataFile.Txt  
datafile.txt
```

参考

 Cipher BASIC では、任意のファイル名の作成やアクセスは行えません。

補足 1. スキャ配列

スキャ配列表 1

配列 (N1%)	設定値 (N2%)	デフォルト値	スキャインジ
1	1: コド 39 読み取り有り 0: コド 39 読み取り無し	1	CCD/レーザ
2	1: イタリアマコード 読み取り有り 0: イタリアマコード 読み取り無し	0	CCD/レーザ
3	1: フランスマコード (CIP 39) 読み取り有り 0: フランスマコード (CIP 39) 読み取り無し	0	CCD/レーザ
4	1: イタ ストリアル 25 読み取り有り 0: イタ ストリアル 25 読み取り無し	1	CCD/レーザ
5	1: インターリーブド 25 読み取り有り 0: インターリーブド 25 読み取り無し	1	CCD/レーザ
6	1: マトリクス 25 読み取り有り 0: マトリクス 25 読み取り無し	0	CCD/レーザ
7	1: コダバ (NW7) 読み取り有り 0: コダバ (NW7) 読み取り無し	1	CCD/レーザ
8	1: コド 93 読み取り有り 0: コド 93 読み取り無し	1	CCD/レーザ
9	1: コド 128&EAN-128 読み取り有り 0: コド 128&EAN-128 読み取り無し	1	CCD/レーザ
10	1: UPC-E 読み取り有り 0: UPC-E 読み取り無し	1	CCD/レーザ
11	1: UPC-E プド 2 読み取り有り 0: UPC-E プド 2 読み取り無し	0	CCD/レーザ
12	1: UPC-E プド 5 読み取り有り 0: UPC-E プド 5 読み取り無し	0	CCD/レーザ
13	1: JAN/EAN-8 読み取り有り 0: JAN/EAN-8 読み取り無し	1	CCD/レーザ
14	1: JAN/EAN-8 プド 2 読み取り有り 0: JAN/EAN-8 プド 2 読み取り無し	0	CCD/レーザ
15	1: JAN/EAN-8 プド 5 読み取り有り 0: JAN/EAN-8 プド 5 読み取り無し	0	CCD/レーザ
16	1: UPC-A&JAN/EAN-13 読み取り有り 0: UPC-A&JAN/EAN-13 読み取り無し	1	CCD/レーザ
17	1: UPC-A&JAN/EAN-13 プド 2 読み取り有り 0: UPC-A&JAN/EAN-13 プド 2 読み取り無し	0	CCD/レーザ
18	1: UPC-A&JAN/EAN-13 プド 5 読み取り有り 0: UPC-A&JAN/EAN-13 プド 5 読み取り無し	0	CCD/レーザ
19	1: MSI 読み取り有り 0: MSI 読み取り無し	0	CCD/レーザ
20	1: Plessey 読み取り有り 0: Plessey 読み取り無し	0	CCD/レーザ
21	1: COOP 25 (NEC 25) 読み取り有り 0: COOP 25 (NEC 25) 読み取り無し	0	CCD/レーザ
22	1: コド 39 スタート/ストップ キャラク送信有り 0: コド 39 スタート/ストップ キャラク送信無し	0	CCD/レーザ
23	1: コド 39 チェックデジット検査有り 0: コド 39 チェックデジット検査無し	0	CCD/レーザ
24	1: コド 39 チェックデジット送信有り 0: コド 39 チェックデジット送信無し	1	CCD/レーザ

(*) レーザ=レーザスキャナ, CCD=リニアイメージャ, エクトラレーザ=ロングレンジレーザスキャナ/エクストラロングレンジレーザスキャナ, 2D=エリアイメージャ
(*) 8500 シリーズは、COOP 25 (NEC 25) をサポートしていません。

配列 (N1%)	設定値 (N2%)	デフォルト値	スキャンゾーン
25	1: コード 39 アルファベット読み取り有り 0: コード 39 アルファベット読み取り無し	0	CCD/レーザ
26	1: イタリアンコード チェック済 送信有り 0: イタリアンコード チェック済 送信無し	0	CCD/レーザ
27	1: フランスコード (CIP 39) チェック済 送信有り 0: フランスコード (CIP 39) チェック済 送信無し	0	CCD/レーザ
28	1: インターリーブド 25 チェック済 検査有り 0: インターリーブド 25 チェック済 検査無し	0	CCD/レーザ
29	1: インターリーブド 25 チェック済 送信有り 0: インターリーブド 25 チェック済 送信無し	1	CCD/レーザ
30	1: インダストリアル 25 チェック済 検査有り 0: インダストリアル 25 チェック済 検査無し	0	CCD/レーザ
31	1: インダストリアル 25 チェック済 送信有り 0: インダストリアル 25 チェック済 送信無し	1	CCD/レーザ
32	1: マトリクス 25 チェック済 検査有り 0: マトリクス 25 チェック済 検査無し	0	CCD/レーザ
33	1: マトリクス 25 チェック済 送信有り 0: マトリクス 25 チェック済 送信無し	1	CCD/レーザ
34	インターリーブド 25 スタート/ストップパターン 0: インダストリアル 25 スタート/ストップパターン 1: インターリーブド 25 スタート/ストップパターン 2: マトリクス 25 スタート/ストップパターン	1	CCD/レーザ
35	インダストリアル 25 スタート/ストップパターン 0: インダストリアル 25 スタート/ストップパターン 1: インターリーブド 25 スタート/ストップパターン 2: マトリクス 25 スタート/ストップパターン	0	CCD/レーザ
36	マトリクス 25 スタート/ストップパターン 0: インダストリアル 25 スタート/ストップパターン 1: インターリーブド 25 スタート/ストップパターン 2: マトリクス 25 スタート/ストップパターン	2	CCD/レーザ
37	コードバー (NW7) スタート/ストップキャラクタ 0: abcd/abcd 1: abcd/tn*e 2: ABCD/ABCD 3: ABCD/TN*E	0	CCD/レーザ
38	1: コードバー (NW7) スタート/ストップキャラクタ送信有り 0: コードバー (NW7) スタート/ストップキャラクタ送信無し	0	CCD/レーザ
39	MSI チェック済 検査 0: シングルモジ 10 10 1: ダブルモジ 10 10 2: モジ 10 11 & モジ 10 10	0	CCD/レーザ
40	MSI チェック済 送信 0: ラストチェック済 送信無し 1: 全チェック済 送信有り 2: 全チェック済 送信無し	1	CCD/レーザ
41	1: Plessey チェック済 送信有り 0: Plessey チェック済 送信無し	1	CCD/レーザ
42	1: Plessey→UK Plessey 変換有り 0: Plessey→UK Plessey 変換無し	1	CCD/レーザ
43	1: UPC-E→UPC-A 変換有り 0: UPC-E→UPC-A 変換無し	0	CCD/レーザ
44	1: UPC-A→UPC-E 変換有り 0: UPC-A→UPC-E 変換無し	1	CCD/レーザ

(*) レーザ=レーザスキャナ, CCD=リニアイメージャ, エクトラレーザ=ロングレンジレーザスキャナ/エクストラロングレンジレーザスキャナ, 2D=エリアイメージャ

配列 (N1%)	設定値 (N2%)	デフォルト値	スキャンレンジ
45	1: ISBN 変換有り 0: ISBN 変換無し	0	CCD/レーザ
46	1: ISSN 変換有り 0: ISSN 変換無し	0	CCD/レーザ
47	1: UPC-E チェックディット送信有り 0: UPC-E チェックディット送信無し	1	CCD/レーザ
48	1: UPC-A チェックディット検査有り 0: UPC-A チェックディット検査無し	1	CCD/レーザ
49	1: JAN/EAN-8 チェックディット送信有り 0: JAN/EAN-8 チェックディット送信無し	1	CCD/レーザ
50	1: JAN/EAN-13 チェックディット検査有り 0: JAN/EAN-13 チェックディット検査無し	1	CCD/レーザ
51	1: UPC-E システムバース送信有り 0: UPC-E システムバース送信無し	0	CCD/レーザ
52	1: UPC-A システムバース送信有り 0: UPC-A システムバース送信無し	1	CCD/レーザ
53	1: JAN/EAN-8→JAN/EAN-13 変換有り 0: JAN/EAN-8→JAN/EAN-13 変換無し	0	CCD/レーザ
54	予備	---	---
55	1: 反転バーコード読み取り有り 0: 反転バーコード読み取り無し	1	CCD/レーザ
56	0: 読取照合回数無し (リーダポート1) 1: 読取照合回数 1 回 (リーダポート1) 2: 読取照合回数 2 回 (リーダポート1) 3: 読取照合回数 3 回 (リーダポート1)	0	CCD/レーザ
57	0: 読取照合回数無し (リーダポート2) 1: 読取照合回数 1 回 (リーダポート2) 2: 読取照合回数 2 回 (リーダポート2) 3: 読取照合回数 3 回 (リーダポート2)	0	固定式ターミナル専用
58	1: インタリアル 25 最大桁数/最小桁数検査を使用 0: インタリアル 25 固定桁数 1/固定桁数 2 検査を使用	1	CCD/レーザ
59	インタリアル 25 最大桁数/固定桁数 1	64 (max.)	CCD/レーザ
60	インタリアル 25 最小桁数/固定桁数 2	1 (min.)	CCD/レーザ
61	1: インターリーブド 25 最大桁数/最小桁数検査を使用 0: インターリーブド 25 固定桁数 1/固定桁数 2 検査を使用	1	CCD/レーザ
62	インターリーブド 25 最大桁数/固定桁数 1	64 (max.)	CCD/レーザ
63	インターリーブド 25 最小桁数/固定桁数 2	1	CCD/レーザ
64	1: マトリクス 25 最大桁数/最小桁数検査を使用 0: マトリクス 25 固定桁数 1/固定桁数 2 検査を使用	1	CCD/レーザ
65	マトリクス 25 最大桁数/固定桁数 1	64 (max.)	CCD/レーザ
66	マトリクス 25 最小桁数/固定桁数 2	1 (min.)	CCD/レーザ
67	1: MSI 最大桁数/最小桁数検査を使用 0: MSI 固定桁数 1/固定桁数 2 検査を使用	1	CCD/レーザ
68	MSI 最大桁数/固定桁数 1	64 (max.)	CCD/レーザ
69	MSI 最小桁数/固定桁数 2	1 (min.)	CCD/レーザ

(*) レーザ=レーザスキャナ, CCD=リアイメージャ, エクスレーザ=ロングレンジレーザスキャナ/エクストラロングレンジレーザスキャナ, 2D=リアイメージャ

配列 (N1%)	設定値 (N2%)	デフォルト値	スキャンゾーン
70	読み取りモード (リーダポート1) 0: オートオフモード 1: コンティニアスモード 2: オートパワーオフモード 3: アルターネイトモード 4: モーメントリモード 5: リピートモード 6: レザモード 7: テストモード 8: イミングモード	6	CCD/レーザ
71	読み取りモード (リーダポート2) 0: オートオフモード 1: コンティニアスモード 2: オートパワーオフモード 3: アルターネイトモード 4: モーメントリモード 5: リピートモード 6: レザモード 7: テストモード 8: イミングモード	6	固定式ターミナル専用
72	読み取りタイムアウト (リーダポート1) イミング / レザ / オートオフ / オートパワーオフモードに適用 設定範囲 0~255 秒, 0 = 読み取りタイムアウト無し	3	CCD/レーザ
73	読み取りタイムアウト (リーダポート2) イミング / レザ / オートオフ / オートパワーオフモードに適用 設定範囲 0~255 秒, 0 = 読み取りタイムアウト無し	3	固定式ターミナル専用
74	1: GS1 Databar リミット 読み取り有り 0: GS1 Databar リミット 読み取り無し	0	CCD/レーザ
75	予備	---	---
76	1: GS1 Databar オムニレクショナル&イクスパンデット 読み取り有り 0: GS1 Databar オムニレクショナル&イクスパンデット 読み取り無し	0	CCD/レーザ
77	1: GS1 Databar オムニレクショナルコード ID 送信有り 0: GS1 Databar オムニレクショナルコード ID 送信無し	1	CCD/レーザ
78	1: GS1 Databar オムニレクショナルアプリケーション ID 送信有り 0: GS1 Databar オムニレクショナルアプリケーション ID 送信無し	1	CCD/レーザ
79	1: GS1 Databar オムニレクショナルチェックディジット送信有り 0: GS1 Databar オムニレクショナルチェックディジット送信無し	1	CCD/レーザ
80	1: GS1 Databar リミットコード ID 送信有り 0: GS1 Databar リミットコード ID 送信無し	1	CCD/レーザ
81	1: GS1 Databar リミットアプリケーション ID 送信有り 0: GS1 Databar リミットアプリケーション ID 送信無し	1	CCD/レーザ
82	1: GS1 Databar リミットチェックディジット送信有り 0: GS1 Databar リミットチェックディジット送信無し	1	CCD/レーザ
83	1: GS1 Databar イクスパンデットコード ID 送信有り 0: GS1 Databar イクスパンデットコード ID 送信無し	1	CCD/レーザ
84	1: Telpen 数字モード 0: Telpen ASCIIモード	0	CCD/レーザ
85	1: Telpen 読み取り有り 0: Telpen 読み取り無し	0	CCD/レーザ
86	1: UPC-E1 & UPC-E0 有効 0: UPC-E0 のみ有効	0	CCD/レーザ
87	1: GTIN 有効 0: GTIN 無効	0	CCD/レーザ
88 ~ 147	N/A	---	---

(*) レザ=レーザスキャナ, CCD=リアイメージャ, イクトラレーザ=ロングレンジレーザスキャナ/エクストラロングレンジレーザスキャナ, 2D=リアイメージャ

配列 (N1%)	設定値 (N2%)	デフォルト値	スキャンレンジ
148	1: UPC-E1 トリプルチェック有効 0: UPC-E1 トリプルチェック無効	0	CCD/レーザ
149	イメージタイムアウト 設定範囲 1~65535 (設定単位 5 ミリ秒) 0 = イメージ無し	200 (1 秒)	CCD/レーザ
150	配列 (N1%) 9 を 1=コード 128&EAN-128 読み取り有りに設定 0: コード 128&GS1-128 読み取り有り (旧フォーマット互換用) 1: GS1-128 のみ読み取り有り 2: コード 128 のみ読み取り有り 3: コード 128&GS1-128 読み取り有り	0	CCD/レーザ
151	配列 (N1%) 9 を 1=コード 128&EAN-128 読み取り有りに設定 1: GS1-128 コード ID 送信無し 0: GS1-128 コード ID 送信有り	0	CCD/レーザ
152	1: ISBT-128 読み取り有り 0: ISBT-128 読み取り無し	0	CCD/レーザ
153 ~ 170	N/A	---	---
171	1: COOP 25 (NEC 25) チェックサム検査有り 0: COOP 25 (NEC 25) チェックサム検査無し	0	CCD/レーザ
172	1: COOP 25 (NEC 25) チェックサム送信有り 0: COOP 25 (NEC 25) チェックサム送信無し	1	CCD/レーザ

(*) レーザ=レーザスキャナ, CCD=リアイメージャ, エキストラレーザ=ロングレンジレーザスキャナ/エキストラロングレンジレーザスキャナ, 2D=リアイメージャ

(*) 8500 シリーズは、COOP 25 (NEC 25) をサポートしていません。

スキャ配列表 2

配列 (N1%)	設定値 (N2%)	デフォルト値	スキャインジソン
1	1: コード 39 読み取り有り 0: コード 39 読み取り無し	1	2D/エクストラレーザ
2	1: イタリアファーマコード (コード 32) 読み取り有り 0: イタリアファーマコード (コード 32) 読み取り無し	0	2D/エクストラレーザ
3	N/A	---	---
4	N/A	---	---
5	1: インターリーブド 25 読み取り有り 0: インターリーブド 25 読み取り無し	1	2D/エクストラレーザ
6	1: マトリクス 25 読み取り有り 0: マトリクス 25 読み取り無し	0	8200/8400/8700-2D
7	1: コーダバ (NW7) 読み取り有り 0: コーダバ (NW7) 読み取り無し	1	2D/エクストラレーザ
8	1: コード 93 読み取り有り 0: コード 93 読み取り無し	1	2D/エクストラレーザ
9	1: コード 128 読み取り有り 0: コード 128 読み取り無し	1	2D/エクストラレーザ
10	1: UPC-E0 読み取り有り 0: UPC-E0 読み取り無し	1	2D/エクストラレーザ
11, 12	1: UPC/EAN/JAN ファミリーコード 2/5 読み取り有り (何れかを読み取る場合は、全てを 1 にセット) 0: UPC/EAN/JAN ファミリーコード 2/5 読み取り有り (何れも読み取らない場合は、全てを 0 にセット) (*) 配列 (N1%) 14, 15, 17, 18, 107, 19 を参照	0	2D/エクストラレーザ
13	1: JAN/EAN-8 読み取り有り 0: JAN/EAN-8 読み取り無し	1	2D/エクストラレーザ
14, 15	(*) 配列 (N1%) 11, 12 を参照	0	2D/エクストラレーザ
16	1: JAN/EAN-13 読み取り有り 0: JAN/EAN-13 読み取り無し	1	2D/エクストラレーザ
17, 18	(*) 配列 (N1%) 11, 12 を参照	0	2D/エクストラレーザ
19	1: MSI 読み取り有り 0: MSI 読み取り無し	0	2D/エクストラレーザ
20	N/A	---	---
21	予約	---	---
22	N/A	---	---
23	1: コード 39 チェックデジット検査有り 0: コード 39 チェックデジット検査無し	0	2D/エクストラレーザ
24	1: コード 39 チェックデジット送信有り 0: コード 39 チェックデジット送信無し	1	2D/エクストラレーザ
25	1: コード 39 フルアスキー読み取り有り 0: コード 39 フルアスキー読み取り無し	0	2D/エクストラレーザ
26	N/A	---	---
27	N/A	---	---
28	N/A	---	---
29	1: インターリーブド 25 チェックデジット送信有り 0: インターリーブド 25 チェックデジット送信無し	1	2D/エクストラレーザ
30	N/A	---	---
31	N/A	---	---

(*) レーザ=レーザスキャナ, CCD=リニアイメージャ, エクストラレーザ=ロングレンジレーザスキャナ/エクストラロングレンジレーザスキャナ, 2D=エリアイメージャ

配列 (N1%)	設定値 (N2%)	デフォルト値	スキャンエンジン
32	1: マトリクス 25 チェック データ 検査有り 0: マトリクス 25 チェック データ 検査無し	0	8200/8400/8700-2D
33	1: マトリクス 25 チェック データ 送信有り 0: マトリクス 25 チェック データ 送信無し	0	8200/8400/8700-2D
34	N/A	---	---
35	N/A	---	---
36	N/A	---	---
37	N/A	---	---
38	1: コーダバ - (NW7) スタート/ストップ キャラクタ 送信有り 0: コーダバ - (NW7) スタート/ストップ キャラクタ 送信無し	0	2D/エクストラレーザ
39	MSI チェック データ 検査 0: シングル 10 10 1: ダブル 10 10 2: 10 11 & 10 10	1	2D/エクストラレーザ
40	MSI チェック データ 送信 0: ラスト チェック データ 送信無し 1: 全 チェック データ 送信有り 2: 全 チェック データ 送信無し	0	2D/エクストラレーザ
41	N/A	---	---
42	N/A	---	---
43	1: UPC-E0→UPC-A 変換有り 0: UPC-E0→UPC-A 変換無し	0	2D/エクストラレーザ
44	N/A	---	---
45	N/A	---	---
46	N/A	---	---
47	1: UPC-E0 チェック データ 送信有り 0: UPC-E0 チェック データ 送信無し	1	2D/エクストラレーザ
48	1: UPC-A チェック データ 検査有り 0: UPC-A チェック データ 検査無し	1	2D/エクストラレーザ
49	N/A	---	---
50	N/A	---	---
51	1: UPC-E0 システム バ - 送信有り 0: UPC-E0 システム バ - 送信無し	1	2D/エクストラレーザ
52	1: UPC-A システム バ - 送信有り 0: UPC-A システム バ - 送信無し	1	2D/エクストラレーザ
53	1: JAN/EAN-8→JAN/EAN-13 変換有り 0: JAN/EAN-8→JAN/EAN-13 変換無し	1	2D/エクストラレーザ
54	予備	---	---
55	N/A	---	---
56	N/A	---	---
57	N/A	---	---
58	N/A	---	---
59	N/A	---	---
60	N/A	---	---
61	1: インタリープド 25 最大桁数/最小桁数検査を使用 0: インタリープド 25 固定桁数 1/固定桁数 2 検査を使用	0	2D/エクストラレーザ
62	インタリープド 25 最大桁数/固定桁数 1	0	2D/エクストラレーザ
63	インタリープド 25 最小桁数/固定桁数 2	0	2D/エクストラレーザ
64	1: マトリクス 25 最大桁数/最小桁数検査を使用 0: マトリクス 25 固定桁数 1/固定桁数 2 検査を使用	1	2D/エクストラレーザ
65	マトリクス 25 最大桁数/固定桁数 1	0	2D/エクストラレーザ
66	マトリクス 25 最小桁数/固定桁数 2	0	2D/エクストラレーザ

(*) レーザ = レーザ スキャナ, CCD = リニア イメージャ, エクストラレーザ = ロング レンジ レーザ スキャナ/エクストラロング レンジ レーザ スキャナ, 2D = エリア イメージャ

配列 (N1%)	設定値 (N2%)	デフォルト値	スキャンゾーン
67	1: MSI 最大桁数/最小桁数検査を使用 0: MSI 固定桁数 1/固定桁数 2 検査を使用	1	2D/エクストラレーザ
68	MSI 最大桁数/固定桁数 1	31 (max.)	2D/エクストラレーザ
69	MSI 最小桁数/固定桁数 2	3 (min.)	2D/エクストラレーザ
70	読み取りモード (リーダポート 1) 0: オートモード 1: コンティニアスモード 2: レーザモード 3: オルタネイトモード 4: レーザモード 5: レーザモード 6: レーザモード 7: テストモード 8: イミシグモード	6	2D/エクストラレーザ
71	N/A	---	---
72	N/A	---	---
73	N/A	---	---
74	N/A	---	---
75	N/A	---	---
76	N/A	---	---
77	N/A	---	---
78	N/A	---	---
79	N/A	---	---
80	N/A	---	---
81	N/A	---	---
82	N/A	---	---
83	N/A	---	---
84	N/A	---	---
85	N/A	---	---
86	N/A	---	---
87	N/A	---	---
88	1: コード 39 最大桁数/最小桁数検査を使用 0: コード 39 固定桁数 1/固定桁数 2 検査を使用	0	2D/エクストラレーザ
89	コード 39 最大桁数/固定桁数 1	0	2D/エクストラレーザ
90	コード 39 最小桁数/固定桁数 2	0	2D/エクストラレーザ
91	1: UPC-E1 システムバース送信有り 0: UPC-E1 システムバース送信無し	0	2D/エクストラレーザ
92	1: UPC-E1 チェックビット送信有り 0: UPC-E1 チェックビット送信無し	0	2D/エクストラレーザ
93	1: UCC.EAN コンボビット GS1-128 イミレーションモード 有効 0: UCC.EAN コンボビット GS1-128 イミレーションモード 無効	0	2D
94	1: TCIF Linked コード 39 読み取り有り 0: TCIF Linked コード 39 読み取り無し	1	2D
95	1: UPC-E1→UPC-A 変換有り 0: UPC-E1→UPC-A 変換無し	0	2D/エクストラレーザ
96	1: コード 11 読み取り有り 0: コード 11 読み取り無し	1	2D/8300-LR/8700-LR
97	配列 (N1%) 16 を 1=JAN/EAN13 読み取り有りに設定 1: Bookland EAN 読み取り有り 0: Bookland EAN 読み取り無し	0	2D/エクストラレーザ
98	1: インダストリアル 25 (ディスプレイ) 読み取り有り 0: インダストリアル 25 (ディスプレイ) 読み取り無し	1	2D/エクストラレーザ
99	1: ISBT128 読み取り有り 0: ISBT128 読み取り無し	1	2D/エクストラレーザ

(*) レーザ=レーザスキャナ, CCD=リアイメージャ, エクストラレーザ=ロングレンジレーザスキャナ/エクストラロングレンジレーザスキャナ, 2D=リアイメージャ

配列 (N1%)	設定値 (N2%)	デフォルト値	スキャンエンジン
100	1: Trioptic コド 39 読み取り有り 0: Trioptic コド 39 読み取り無し	0	2D/エクストラレーザ
101	1: GS1-128 読み取り有り 0: GS1-128 読み取り無し	1	2D/エクストラレーザ
102	1: GS1 Databar→UPC/EAN 変換有り 0: GS1 Databar→UPC/EAN 変換無し	0	2D/エクストラレーザ
103	1: GS1 Databar イクスパンデッド 読み取り有り 0: GS1 Databar イクスパンデッド 読み取り無し	1	2D/エクストラレーザ
104	1: GS1 Databar リミテッド 読み取り有り 0: GS1 Databar リミテッド 読み取り無し	1	2D/エクストラレーザ
105	1: GS1 Databar 拡張イックジョナル読み取り有り 0: GS1 Databar 拡張イックジョナル読み取り無し	1	2D/エクストラレーザ
106	1: UPC-A 読み取り有り 0: UPC-A 読み取り無し	1	2D/エクストラレーザ
107,109	1: UPC/EAN/JAN ファミリーアド カ 2/5 読み取り有り (何れかを読み取る場合は、全てを 1 にセット) 0: UPC/EAN/JAN ファミリーアド カ 2/5 読み取り有り (何れも読み取らない場合は、全てを 0 にセット) (*) 配列 (N1%) 11, 12, 14, 15, 17, 18 を参照	0	2D/エクストラレーザ
108	1: UPC-E1 読み取り有り 0: UPC-E1 読み取り無し	0	2D/エクストラレーザ
110	2: UPC コボット 自動識別 1: UPC 常にリク有り 0: UPC 常にリク無し	1	2D
111	1: コボット CC-A/B 読み取り有り 0: コボット CC-A/B 読み取り無し	0	2D*
112	1: コボット CC-C 読み取り有り 0: コボット CC-C 読み取り無し	0	2D*
113	1: コド 93 最大桁数/最小桁数検査を使用 0: コド 93 固定桁数 1/固定桁数 2 検査を使用	0	2D/エクストラレーザ
114	コド 93 最大桁数/固定桁数 1	0	2D/エクストラレーザ
115	コド 93 最小桁数/固定桁数 2	0	2D/エクストラレーザ
116	1: コド 11 最大桁数/最小桁数検査を使用 0: コド 11 固定桁数 1/固定桁数 2 検査を使用	0	2D/8300-LR/8700-LR
117	コド 11 最大桁数/固定桁数 1	0	2D/8300-LR/8700-LR
118	コド 11 最小桁数/固定桁数 2	0	2D/8300-LR/8700-LR
119	1: イタストリアル 25 (デイスクリット 25) 最大桁数/最小桁数検査を使用 0: イタストリアル 25 (デイスクリット 25) 固定桁数 1/固定桁数 2 検査を使用	0	2D/エクストラレーザ
120	イタストリアル 25 (デイスクリット 25) 最大桁数/固定桁数 1	0	2D/エクストラレーザ
121	イタストリアル 25 (デイスクリット 25) 最小桁数/固定桁数 2	0	2D/エクストラレーザ
122	1: コダバ (NW7) 最大桁数/最小桁数検査を使用 0: コダバ (NW7) 固定桁数 1/固定桁数 2 検査を使用	0	2D/エクストラレーザ
123	コダバ (NW7) 最大桁数/固定桁数 1	0	2D/エクストラレーザ
124	コダバ (NW7) 最小桁数/固定桁数 2	0	2D/エクストラレーザ
125	1: US Postal チックゲット送信有り 0: US Postal チックゲット送信無し	1	2D
126	1: Maxicode 読み取り有り 0: Maxicode 読み取り無し	1	2D
127	1: Data Matrix 読み取り有り 0: Data Matrix 読み取り無し	1	2D

(*) レザ=レーザスキャナ, CCD=リニアイメージャ, イクストラレーザ=ロングレンジレーザスキャナ/エクストラロングレンジレーザスキャナ, 2D=エリアイメージャ

配列 (N1%)	設定値 (N2%)	デフォルト値	スキャンインジック
128	1: QRコード 読み取り有り 0: QRコード 読み取り無し	1	2D
129	1: US Planet 読み取り有り 0: US Planet 読み取り無し	1	2D
130	1: US Postnet 読み取り有り 0: US Postnet 読み取り無し	1	2D
131	1: MicroPDF417 読み取り有り 0: MicroPDF417 読み取り無し	1	2D
132	1: PDF417 読み取り有り 0: PDF417 読み取り無し	1	2D
133	予約	--	--
134	1: 日本郵便コード (加付バーコード) 読み取り有り 0: 日本郵便コード (加付バーコード) 読み取り無し	1	2D
135	1: オーストラリア郵便コード 読み取り有り 0: オーストラリア郵便コード 読み取り無し	1	2D
136	1: フランス 郵便コード 読み取り有り 0: フランス 郵便コード 読み取り無し	1	2D
137	1: UK Postal チックゲージ 検査有り 0: UK Postal チックゲージ 検査無し	1	2D
138	1: UK Postal 読み取り有り 0: UK Postal 読み取り無し	1	2D
139	1: 複合コフイグ レーション UPC/EAN ファミリーコード あり無し/あり 2/あり 5 読み取り有り 0: 複合コフイグ レーション無し	0	2D/エクストラレーザ
140	2: インターリーブド 25 OPCC チックゲージ 検査有り 1: インターリーブド 25 USS チックゲージ 検査有り 0: インターリーブド 25 チックゲージ 検査無し	0	2D/エクストラレーザ
141	1: UPC-A システムバーコード & カントリーコード 送信有り 0: UPC-A システムバーコード & カントリーコード 送信無し	1	2D/エクストラレーザ
142	1: UPC-E0 システムバーコード & カントリーコード 送信有り 0: UPC-E0 システムバーコード & カントリーコード 送信無し	1	2D/エクストラレーザ
143	1: UPC-E1 システムバーコード & カントリーコード 送信有り 0: UPC-E1 システムバーコード & カントリーコード 送信無し	1	2D/エクストラレーザ
144	1: インターリーブド 25→EAN-13 変換有り 0: インターリーブド 25→EAN-13 変換無し	0	2D/エクストラレーザ
145	読み取りタイムアウト Iミツグ /レーザ /オートオフ/オートパワーオフモード に適用 設定範囲 0~255 秒, 0 = 読み取りタイムアウト無し	3	2D/エクストラレーザ
146	Macro PDF 送信/デコードモード 2: 順序に関係なく全てのシボ ルをリセットで送信 1: 全てのシボ ルをパ ッファシ、Macro PDF 完結後、送信 0: 全てのシボ ルをパ スルで送信	0	2D
147	1: Macro PDF イスケープ キャラクタ有効 0: Macro PDF イスケープ キャラクタ無効	0	2D
148	N/A	---	---
149	Iミツグ タイムアウト 設定範囲 1~65535 (設定単位 5 ミリ秒) 0 = Iミツグ 無し	200 (1 秒)	2D/エクストラレーザ
150	N/A	---	---
151	N/A	---	---
152	N/A	---	---

(*) レーザ =レーザ スキャナ, CCD=リニアイメージャ, エクストラレーザ =ロング レンジ レーザ スキャナ/エクストラロング レンジ レーザ スキャナ, 2D=エリアイメージャ

配列 (N1%)	設定値 (N2%)	デフォルト値	スキャンゾーン
153	フォーカスモード 2: スマートフォーカス 1: ニアフォーカス 0: ファーフォーカス	0	8500-2D
154	1: イミシグパターン有効 0: イミシグパターン無効	1	2D
155	1: 読み取り照明有効 0: 読み取り照明無効	1	2D
156	1: ピックリストモード 有効 0: ピックリストモード 無効	0	8200-2D, 8400-2D, 8700-2D
157	1 次元反転バーコード 読み取り 2: 通常/反転バーコード 自動読み取り 1: 反転バーコード のみ読み取り 0: 通常バーコード のみ読み取り	0	8200-2D, 8400-2D, 8700-2D
158	1: システムがサスペンド時、リーダーをスリープモードに移行 0: システムがサスペンド時、リーダーをパワーオンする	0	8200-2D, 8400-2D, 8700-2D
159	1: USPS 4CB/One Code/Intelligent Mail 読み取り有り 0: USPS 4CB/One Code/Intelligent Mail 読み取り無し	0	8200-2D, 8400-2D, 8700-2D
160	1: UPU FICS Postal 読み取り有り 0: UPU FICS Postal 読み取り無し	0	8200-2D, 8400-2D, 8700-2D
161	UPC/EAN-Bookland ISBN フォーマット 1: UPC/EAN-Bookland ISBN 13 0: UPC/EAN-Bookland ISBN 10	0	8200-2D, 8400-2D, 8700-2D
162	反転 Data Matrix 読み取り 2: 通常/反転 Data Matrix 自動読み取り 1: 反転 Data Matrix のみ読み取り 0: 通常 Data Matrix のみ読み取り	0	8200-2D, 8400-2D, 8700-2D
163	ミラード Data Matrix 読み取り 2: 通常/ミラード Data Matrix 自動読み取り 1: ミラード Data Matrix のみ読み取り 0: 通常 Data Matrix のみ読み取り	0	8200-2D, 8400-2D, 8700-2D
164	反転 QR コード 読み取り 2: 通常/反転 QR コード 自動読み取り 1: 反転 QR コード のみ読み取り 0: 通常 QR コード のみ読み取り	0	8200-2D, 8400-2D, 8700-2D
165	1: MicroQR コード 読み取り有り 0: MicroQR コード 読み取り無し	1	8200-2D, 8400-2D, 8700-2D
166	1: Aztec 読み取り有り 0: Aztec 読み取り無し	1	8200-2D, 8400-2D, 8700-2D
167	反転 Aztec 読み取り 2: 通常/反転 Aztec 自動読み取り 1: 反転 Aztec のみ読み取り 0: 通常 Aztec のみ読み取り	0	8200-2D, 8400-2D, 8700-2D
168	1: Coupon コード 読み取り有り 0: Coupon コード 読み取り無し	0	2D/エクストラレーザ
169	1: チャ付 25 読み取り有り 0: チャ付 25 読み取り無し	0	8200-2D, 8400-2D, 8700-2D
170	2: コード 11 2 チェックビット検査有り 1: コード 11 1 チェックビット検査有り 0: コード 11 チェックビット検査無し	0	2D/8300-LR/8700-LR
171 ~ 172	N/A	---	---

(*) レーザ=レーザスキャナ, CCD=リニアイメージャ, エクストラレーザ=ロングレンジレーザスキャナ/エクストラロングレンジレーザスキャナ, 2D=エリアイメージャ

補足 2. 木トマツド

ここで紹介する木トマツド をターミナルに送信することで、ブラウザ クリソソファイルやリアルタイムクロックの操作が可能になります。木トマツド の処理は、BASIC ランタイムエンジンがバックグラウンドで行っているため、ユーザーは BASIC プログラムを作成する必要はありません。

参考

木トマツド は、必ずキャリッジ リターン (CR=0Dhex) を終端に付加する必要があります。

CLEAR

適用機種	ALL
目的	指定のブラウザ クリソソファイルの全データを削除します。
構文	REM A\$は戻り値を代入するための文字列型変数です。 A\$ = CLEAR A\$ = CLEAR file%
解説	引数 file%には、ブラウザ クリソソファイル番号を 1~6 の範囲で指定します。省略した場合は、ブラウザ クリソソファイル番号 1 の全データを削除します。 マツド を実行すると、戻り値が返されます。

戻り値	説明
OK<CR>	処理完了
NAK<CR>	マツド が正しくありません

使用例	<pre>` Visual Basic サンプル MSComm1.CommPort = 1 MSComm1.Settings = "9600,n,8,1" MSComm1.InputLen = 1 ... MSComm1.PortOpen = True MSComm1.Output = "CLEAR3" & vbCr ` ブラウザ クリソソファイル番号 3 からデータを削除 ... Hst = MSComm1.Input If Hst="NAK" Then ... End If</pre>
-----	--

READ

適用機種 ALL

目的 指定のトラザ* クイソファイルから最初のデータを取得します。

構文 REM A\$は戻り値を代入するための文字列型変数です。

A\$ = READ

A\$ = READ file%

解説 引数 file%には、トラザ* クイソファイル番号を 1~6 の範囲で指定します。省略した場合は、トラザ* クイソファイル番号 1 からデータを取得します。

コマンドを実行すると、戻り値が返されます。

戻り値	説明
データ<CR>	取得したデータ
OVER<CR>	トラザ* クイソファイルにデータがありません
NAK<CR>	コマンドが正しくありません

使用例

```
` Visual Basic サンプル
MSComm1.CommPort = 1
MSComm1.Settings = "9600,n,8,1"
MSComm1.InputLen = 1
...
MSComm1.PortOpen = True
MSComm1.Output = "READ3" & vbCr      ` トラザ* クイソファイル番号 3 からデータを取得
...
Hst = MSComm1.Input
If Hst = "OVER" Then
    ...
Else If Hst = "NAK"
    ...
End If
```

REMOVE

適用機種 ALL

目的 指定のトランザクションファイルの最後に保存されたデータを削除します。

構文 REM A\$は戻り値を代入するための文字列型変数です。

A\$ = REMOVE

A\$ = REMOVE *file%*

解説 引数 *file%*には、トランザクションファイル番号を 1~6 の範囲で指定します。省略した場合は、トランザクションファイル番号 1 からデータを削除します。

コマンドを実行すると、戻り値が返されます。

戻り値	説明
データ<CR>	取得したデータ
OVER<CR>	トランザクションファイルにデータがありません
NAK<CR>	コマンドが正しくありません

使用例

```
` Visual Basic サンプル
MSComm1.CommPort = 1
MSComm1.Settings = "9600,n,8,1"
MSComm1.InputLen = 1
...
MSComm1.PortOpen = True
MSComm1.Output = "REMOVE3" & vbCr    ` トランザクションファイル番号 3 からデータを削除
...
Hst = MSComm1.Input
If Hst = "OVER" Then
    ...
Else If Hst = "NAK"
    ...
End If
```

TR

適用機種	ALL
目的	システム日付・時刻を取得します。
構文	REM A\$は戻り値を代入するための文字列型変数です。 A\$ = TR
解説	コマンドを実行すると、戻り値が返されます。

戻り値	説明
データ<CR>	取得したシステム日付・時刻データ (yyyymmddhhnnss)
NAK<CR>	コマンドが正しくありません

使用例

```
´ Visual Basic サンプル
MSComm1.CommPort = 1
MSComm1.Settings = "9600,n,8,1"
MSComm1.InputLen = 1
...
MSComm1.PortOpen = True
MSComm1.Output = "TR" & vbCr           ´ システム日付・時刻データを取得
...
Hst = MSComm1.Input
If Hst = "NAK" Then
    ...
End If
```

TW

適用機種	ALL
目的	日時・時刻を設定します。
構文	REM A\$は戻り値を代入するための文字列型変数です。 A\$ = TWyyyymmddhhnnss
解説	引数 <i>yyyymmddhhnnss</i> には、設定したい日時・時刻データを指定します。

<i>yyyy</i>	:	西暦 4 桁
<i>mm</i>	:	月 2 桁
<i>dd</i>	:	日 2 桁
<i>hh</i>	:	時 2 桁 (24 時間フォーマット)
<i>nn</i>	:	分 2 桁
<i>ss</i>	:	秒 2 桁

コマンドを実行すると、戻り値が返されます。

戻り値	説明
OK<CR>	設定完了
NAK<CR>	フォーマットが正しくありません

使用例

```

` Visual Basic サンプル
MSComm1.CommPort = 1
MSComm1.Settings = "9600,n,8,1"
MSComm1.InputLen = 1
...
MSComm1.PortOpen = True
MSComm1.Output = "TW20120215101010" & vbCr ` 2012/02/15 10:10:10 に設定
...
Hst = MSComm1.Input
If Hst = "NAK" Then
    ...
End If

```

補足 3. デバグコマンド

Cipher BASIC では、プログラムの診断、デバグを行うためのコマンドが用意されています。START_DEBUG コマンドを実行することで、指定の COM ポートを通して、プログラムの実行状況を逐次に列挙し、問題解決を容易にします。

START_DEBUG

適用機種	ALL
目的	デバグモードを開始します。
構文	START_DEBUG (port%, baudrate%, parity%, data%, handshake%)
解説	下記の表を参照して、各引数に適切な値を指定します。衝突を避けるため、プログラム中で使用している COM ポート以外を指定するようにしてください。

引数	値	説明
port%	1, 2, 5 の何れか	デバグメッセージを出力する COM ポートを指定
baudrate%	1: 115200bps 2: 76800bps 3: 57600bps 4: 38400bps 5: 19200bps 6: 9600bps 7: 4800bps 8: 2400bps	ボーレートを指定
parity%	1: 無し 2: 奇数 3: 偶数	パリティを指定
data%	1: 7ビット 2: 8ビット	データビットを指定
handshake%	1: 無し 2: CTS/RTS 3: XON/XOFF	ハンドシェイクを指定

	デバグモードでは、指定された COM ポートにプログラムの実行状況を示すデバグメッセージが出力されます。デバグメッセージの詳細については、次頁以降を参照ください。	
使用例	SET_COM_TYPE (1, 1)	、 RS232C, COM1
	START_DEBUG (1, 1, 1, 2, 1)	、 デバグモード開始 COM1, 115.2K/8/N/N

STOP_DEBUG

適用機種	ALL
目的	デバグモードを終了します。
構文	STOP_DEBUG
解説	デバグモードを終了します。
使用例	STOP_DEBUG、デバグモード終了

デバックメッセージ

デバックメッセージ	説明
ABS (n)	ABS ｺﾏﾝﾄﾞ を実行
ADD (n1%, n2%)	足し算を実行
ADD_RECORD (file%, data\$)	ADD_RECORD ｺﾏﾝﾄﾞ を実行
ALPHA_LOCK (status%)	ALPHA_LOCK ｺﾏﾝﾄﾞ を実行
AND	論理演算 AND を実行
ARY (n%)	n 要素の配列を宣言
ASC (x\$)	ASC ｺﾏﾝﾄﾞ を実行
ASGN (a)	変数に値 a を代入 (a は、整数、長整数、文字、文字列など)
AUTO_OFF (n%)	AUTO_OFF ｺﾏﾝﾄﾞ を実行 (n% は、設定時間)
BACK_LIGHT_DURATION (n%)	BACK_LIGHT_DURATION ｺﾏﾝﾄﾞ を実行 (n% は、設定時間)
BACKLIT (state%)	BACKLIT ｺﾏﾝﾄﾞ を実行
BACKUP_BATTERY	BACKUP_BATTERY ｺﾏﾝﾄﾞ を実行
BEEP (...)	BEEP ｺﾏﾝﾄﾞ を実行
BIT_OPERATOR (...)	BIT_OPERATOR ｺﾏﾝﾄﾞ を実行
BT_INQUIRY\$	BT_INQUIRY\$ ｺﾏﾝﾄﾞ を実行
BT_PAIRING (addr\$, type%)	BT_PAIRING ｺﾏﾝﾄﾞ を実行
CHANGE_SPEED (N%)	CHANGE_SPEED ｺﾏﾝﾄﾞ を実行 (n% は、設定値)
CHR\$ (N%)	CHR\$ ｺﾏﾝﾄﾞ を実行
CIRCLE (...)	CIRCLE ｺﾏﾝﾄﾞ を実行
CLOSE_COM (N%)	CLOSE_COM ｺﾏﾝﾄﾞ を実行 (n% は、COM ｺﾞｰﾄ番号)
CLR_KBD	CLR_KBD ｺﾏﾝﾄﾞ を実行
CLR_RECT (...)	CLR_RECT ｺﾏﾝﾄﾞ を実行
CLS	CLS ｺﾏﾝﾄﾞ を実行
CODE_TYPE	CODE_TYPE ｺﾏﾝﾄﾞ を実行
COM_DELIMITER (N%, C%)	COM_DELIMITER ｺﾏﾝﾄﾞ を実行
CURSORSX	CURSORSX ｺﾏﾝﾄﾞ を実行
CURSORY	CURSORY ｺﾏﾝﾄﾞ を実行
DATE\$	DATE\$ ｺﾏﾝﾄﾞ を実行 (リアルタイム取得)
DATE\$ (x\$)	DATE\$ ｺﾏﾝﾄﾞ を実行 (リアルタイム設定、x\$ は設定値)
DAY_OF_WEEK	DAY_OF_WEEK ｺﾏﾝﾄﾞ を実行
DEL_RECORD (file%[, index%])	DEL_RECORD ｺﾏﾝﾄﾞ を実行
DEL_TRANSACTION_DATA (n%)	DEL_TRANSACTION_DATA ｺﾏﾝﾄﾞ を実行
DEL_TRANSACTION_DATA_EX (file%, n%)	DEL_TRANSACTION_DATA_EX ｺﾏﾝﾄﾞ を実行
DISABLE_READER (n%)	DISABLE_READER ｺﾏﾝﾄﾞ を実行
DISABLE_TOUCHSCREEN	DISABLE_TOUCHSCREEN ｺﾏﾝﾄﾞ を実行
DIV (n1%, n2%)	割り算を実行
DNS_RESOLVER (a\$)	DNS_RESOLVER ｺﾏﾝﾄﾞ を実行
DOWNLOAD_BASIC (file%, port%)	DOWNLOAD_BASIC ｺﾏﾝﾄﾞ を実行
EMPTY_FILE (file%)	EMPTY_FILE ｺﾏﾝﾄﾞ を実行 (file% は、DBF ファイル番号)
EMPTY_TRANSACTION	EMPTY_TRANSACTION ｺﾏﾝﾄﾞ を実行
EMPTY_TRANSACTION_EX (file%)	EMPTY_TRANSACTION_EX ｺﾏﾝﾄﾞ を実行 (file% は、トランザクションファイル番号)
ENABLE_READER (n%)	ENABLE_READER ｺﾏﾝﾄﾞ を実行
ENABLE_TOUCHSCREEN	ENABLE_TOUCHSCREEN ｺﾏﾝﾄﾞ を実行
EQU? (n1%, n2%)	IF 文 (IF n1%=n2%) を実行
EVENT (0)	COM (1) イベントが発生
EVENT (1)	COM (2) イベントが発生
EVENT (2)	COM (3) イベントが発生
EVENT (3)	予約
EVENT (4)	予約
EVENT (5)	予約
EVENT (6)	予約
EVENT (7)	予約
EVENT (8)	予約
EVENT (9)	TIMER (1) イベントが発生
EVENT (10)	TIMER (2) イベントが発生

デバ`ック`メッセ-ジ`	説明
EVENT(11)	TIMER(3) ｲﾝﾀ`が発生
EVENT(12)	TIMER(4) ｲﾝﾀ`が発生
EVENT(13)	TIMER(5) ｲﾝﾀ`が発生
EVENT(14)	ON MINUTE EVENT ｲﾝﾀ`が発生
EVENT(15)	ON HOUR EVENT ｲﾝﾀ`が発生
EVENT(16)	ON READER(1) EVENT ｲﾝﾀ`が発生
EVENT(17)	ON READER(2) EVENT ｲﾝﾀ`が発生
EVENT(18)	FUNCTION(1) ｲﾝﾀ`が発生
EVENT(19)	FUNCTION(2) ｲﾝﾀ`が発生
EVENT(20)	TIMER(5) ｲﾝﾀ`が発生
EVENT(21)	ON MINUTE EVENT ｲﾝﾀ`が発生
EVENT(22)	ON HOUR EVENT ｲﾝﾀ`が発生
EVENT(23)	ON READER(1) EVENT ｲﾝﾀ`が発生
EVENT(24)	ON READER(2) EVENT ｲﾝﾀ`が発生
EVENT(25)	TIMER(4) ｲﾝﾀ`が発生
EVENT(26)	TIMER(5) ｲﾝﾀ`が発生
EVENT(27)	ON MINUTE EVENT ｲﾝﾀ`が発生
EVENT(28)	ON HOUR EVENT ｲﾝﾀ`が発生
EVENT(29)	ON READER(1) EVENT ｲﾝﾀ`が発生
EVENT(30)	ON READER(2) EVENT ｲﾝﾀ`が発生
EXP(n1%,n2%)	べき乗を実行
FALSE?(n%)	IF 文又は WHILE 文を実行
FILL_RECT(...)	FILL_RECT ｺﾏﾝﾄﾞ`を実行
FIND_RECORD(...)	FIND_RECORD ｺﾏﾝﾄﾞ`を実行
FLASH_READ\$(n%)	FLASH_READ\$ ｺﾏﾝﾄﾞ`を実行
FLASH_WRITE(n%,a\$)	FLASH_WRITE ｺﾏﾝﾄﾞ`を実行
FREE_MEMORY	FREE_MEMORY ｺﾏﾝﾄﾞ`を実行
FUNCTION_TOGGLE(status%)	FUNCTION_TOGGLE ｺﾏﾝﾄﾞ`を実行
GE?(n1%,n2%)	IF 文(IF n1%>=n2%)を実行
GET_ALPHA_LOCK	GET_ALPHA_LOCK ｺﾏﾝﾄﾞ`を実行
GET_ALPHA_STATE	GET_ALPHA_STATE ｺﾏﾝﾄﾞ`を実行
GET_CTS(n%)	GET_CTS ｺﾏﾝﾄﾞ`を実行 (n%は、COM ｷﾞ`-ﾄ番号)
GET_DEVICE_ID	GET_DEVICE_ID ｺﾏﾝﾄﾞ`を実行
GET_FILE_ERROR	GET_FILE_ERROR ｺﾏﾝﾄﾞ`を実行
GET_IMAGE	GET_IMAGE ｺﾏﾝﾄﾞ`を実行
GET_LANGUAGE	GET_LANGUAGE ｺﾏﾝﾄﾞ`を実行
GET_NET_PARAMETER\$(index%)	GET_NET_PARAMETERS\$ ｺﾏﾝﾄﾞ`を実行
GET_NET_STATUS(index%)	GET_NET_STATUS ｺﾏﾝﾄﾞ`を実行
GET_READER_DATA\$(n%)	GET_READER_DATA\$ ｺﾏﾝﾄﾞ`を実行 (n%は、ﾘｰﾀﾞ` ｷﾞ`-ﾄ番号)
GET_READER_SETTING(n%)	GET_READER_SETTING\$ ｺﾏﾝﾄﾞ`を実行
GET_RECORD\$(file%[,index%])	GET_RECORD\$ ｺﾏﾝﾄﾞ`を実行
GET_RECORD_NUMBER(file%[,index%])	GET_RECORD_NUMBER ｺﾏﾝﾄﾞ`を実行
GET_RFID_KEY(tagtype%)	GET_RFID_KEY ｺﾏﾝﾄﾞ`を実行
GET_SCREENITEM	GET_SCREEN_ITEM ｺﾏﾝﾄﾞ`を実行
GET_TARGET_MACHINE\$	GET_TARGET_MACHINE\$ ｺﾏﾝﾄﾞ`を実行
GET_TCPIP_MESSAGE	GET_TCPIP_MESSAGE ｺﾏﾝﾄﾞ`を実行
GET_TRANSACTION_DATA\$(n%)	GET_TRANSACTION_DATA\$ ｺﾏﾝﾄﾞ`を実行
GET_TRANSACTION_DATA_EX\$(file%,n%)	GET_TRANSACTION_DATA_EX\$ ｺﾏﾝﾄﾞ`を実行
GOSUB(n%)	GOSUB ｺﾏﾝﾄﾞ`を実行 (n%は、ﾌﾞﾛｯｸ`の先頭行番号)
GOTO(n%)	GOTO ｺﾏﾝﾄﾞ`を実行 (n%は、ｼﾞ` ｻｯﾌﾟ` 先の行番号)
GSM_CHANGE_PIN(old\$,new\$)	GSM_CHANGE_PIN ｺﾏﾝﾄﾞ`を実行
GSM_CHECK_PIN(pin\$)	GSM_CHECK_PIN ｺﾏﾝﾄﾞ`を実行
GSM_SET_PINLOCK(pin\$,mode%)	GSM_SET_PINLOCK ｺﾏﾝﾄﾞ`を実行
GT?(n1%,n2%)	IF 文(IF n1%>n2%)を実行
HEX\$(n%)	HEX\$ ｺﾏﾝﾄﾞ`を実行
ICON_ZONE_PRINT(status%)	ICON_ZONE_PRINT ｺﾏﾝﾄﾞ`を実行
INKEY\$(a\$)	INKEY\$ ｺﾏﾝﾄﾞ`を実行
INPUT	INPUT ｺﾏﾝﾄﾞ`を実行

デバ` ッグ` メッセ-ジ`	説明
INPUT_MODE (mode%)	INPUT_MODE `ヲ` 実行
INSTR ([n%], x\$, y\$)	INSTR `ヲ` 実行
INT (n%)	INT `ヲ` 実行
IOPIN_STATUS (n%)	IOPIN_STATUS `ヲ` 実行
IRDA_STATUS (n%)	IRDA_STATUS `ヲ` 実行
IRDA_TIMEOUT (n%)	IRDA_TIMEOUT `ヲ` 実行
KEY_CLICK (status%)	KEY_CLICK `ヲ` 実行
L (n%)	実行した行番号
LCASE\$ (x\$)	LCASE `ヲ` 実行
LCD_CONTRAST (n%)	LCD_CONTRAST `ヲ` 実行
LE? (n1%, n2%)	IF 文 (IF n1%<=n2%) を実行
LED (...)	LED `ヲ` 実行
LEFT\$ (x\$, n%)	LEFT\$ `ヲ` 実行
LEN (x\$)	LEN `ヲ` 実行
LINE (...)	LINE `ヲ` 実行
LOCATE (n1%, n2%)	LOCATE `ヲ` 実行
LOCK	LOCK `ヲ` 実行
LT? (n1%, n2%)	IF 文 (IF n1%<n2%) を実行
MAIN_BATTERY	MAIN_BATTERY `ヲ` 実行
MENU (item\$)	MENU `ヲ` 実行
MEMORY_INFORMATION (n%)	MEMORY_INFORMATION `ヲ` 実行
MID\$ (x\$, n% [, m%])	MID\$ `ヲ` 実行
MOD (n1%, n2%)	MOD `ヲ` 実行
MOVE_TO (file%, [, index%], record_no%)	MOVE_TO `ヲ` 実行
MOVE_TO_NEXT (file%, [, index%])	MOVE_TO_NEXT `ヲ` 実行
MOVE_TO_PREVIOUS (file%, [, index%])	MOVE_TO_PREVIOUS `ヲ` 実行
MUL (n1%, n2%)	掛け算を実行
NEG (n1%)	否定 (-) を実行
NEQ? (n1%, n2%)	IF 文 (IF n1%<>n2%) を実行
NCLOSE (n%)	NCLOSE `ヲ` 実行
NOT	論理演算子 NOT を実行
NREAD (n%)	NREAD `ヲ` 実行
NWRITE (n%, a\$)	NWRITE `ヲ` 実行
OCT\$ (n%)	OCT\$ `ヲ` 実行
OFF_ALL	OFF_ALL `ヲ` 実行
OFF_COM (n%)	OFF_COM `ヲ` 実行 (n%は、COM `ホ` -ト番号)
OFF_ESC	OFF_ESC `ヲ` 実行
OFF_HOUR_SHAPE	OFF_HOUR_SHAPE `ヲ` 実行
OFF_KEY (number%)	OFF_KEY `ヲ` 実行
OFF_MINUTE_SHAPE	OFF_MINUTE_SHARP `ヲ` 実行
OFF_READER (n%)	OFF_READER `ヲ` 実行 (n%は、リダ `ホ` -ト番号)
OFF_TCPIP	OFF_TCPIP `ヲ` 実行
OFF_TIMER (n%)	OFF_TIMER `ヲ` 実行 (n%は、タイマ-番号)
OFF_TOUCHSCREEN	OFF_TOUCHSCREEN `ヲ` 実行
ON_COM (n1%, n2%)	ON_COM GOSUB `ヲ` 実行 (n1%は、COM `ホ` -ト番号、n2%は、サブ `ル-` の先頭行番号)
ON_ESC (n%)	ON_ESC GOSUB `ヲ` 実行 (n%は、サブ `ル-` の先頭行番号)
ON_GOSUB (n%)	ON_GOSUB `ヲ` 実行 (n%は、サブ `ル-` の先頭行番号)
ON_GOTO (n%)	ON_GOTO `ヲ` 実行 (n%は、ジャン `プ` 先の行番号)
ON_HOUR_SHARP (n%)	ON_HOUR_SHARP GOSUB `ヲ` 実行 (n%は、サブ `ル-` の先頭行番号)
ON_KEY (n%)	ON_KEY GOSUB `ヲ` 実行 (n%は、サブ `ル-` の先頭行番号)
ON_MINUTE_SHARP (n%)	ON_MINUTE_SHARP GOSUB `ヲ` 実行 (n%は、サブ `ル-` の先頭行番号)
ON_POWER_ON (n%)	ON_POWER_ON GOSUB `ヲ` 実行 (n%は、サブ `ル-` の先頭行番号)
ON_READER (n1, n2%)	ON_READER GOSUB `ヲ` 実行 (n1%は、リダ `ホ` -ト番号、n2%は、サブ `ル-` の先頭行番号)
ON_TCPIP (n%)	ON_TCPIP GOSUB `ヲ` 実行 (n%は、サブ `ル-` の先頭行番号)
ON_TIMER (n1%, n2%)	ON_TIMER GOSUB `ヲ` 実行 (n1%は、タイマ-番号、n2%は、サブ `ル-` の先頭行番号)

デバックメッセージ	説明
ON_TOUCHSCREEN(n%)	ON TOUCHSCREEN GOSUB ｺマﾝﾄを実行 (n%は、ﾌﾞﾛｯｸの先頭行番号)
OPEN_COM(n%)	OPEN ｺマﾝﾄを実行 (n%は、COMﾎﾞｰﾄ番号)
OR	論理演算子 OR を実行
POWER_ON(n%)	POWER ON ｺマﾝﾄを実行
PRINT(a\$)	PRINT ｺマﾝﾄを実行
PUT_PIXEL(...)	PUT_PIXEL ｺマﾝﾄを実行
PUTKEY(n%)	PUTKEY ｺマﾝﾄを実行
RAM_SIZE	RAM_SIZE ｺマﾝﾄを実行
READ_COM\$(n%)	READ_COM\$ ｺマﾝﾄを実行 (n%は、COMﾎﾞｰﾄ番号)
READER_CONFIG	READER CONFIG ｺマﾝﾄを実行
READER_SETTING(n1%,n2%)	READER_SETTING ｺマﾝﾄを実行 (n1%は、設定番号、n2%は、設定値)
RECORD_COUNT(file%)	RECORD_COUNT ｺマﾝﾄを実行
RECTANGLE(...)	RECTANGLE ｺマﾝﾄを実行
RESTART	RESTART ｺマﾝﾄを実行
RETURN(n%)	RETURN ｺマﾝﾄを実行 (n%は、ﾘﾀｰﾝ先の行番号)
RIGHT\$(x\$,n%)	RIGHT\$ ｺマﾝﾄを実行
ROM_SIZE	ROM_SIZE ｺマﾝﾄを実行
SAVE_TRANSACTION(data\$)	SAVE_TRANSACTION ｺマﾝﾄを実行
SAVE_TRANSACTION_EX(file%,data\$)	SAVE_TRANSACTION EX ｺマﾝﾄを実行
SD_FREE_MEMORY	SD_FREE_MEMORY ｺマﾝﾄを実行
SD_SIZE	SD_SIZE ｺマﾝﾄを実行
SELECT_FONT(font%)	SELECT_FONT ｺマﾝﾄを実行
SEND_WEDGE(data\$)	SEND WEDGE ｺマﾝﾄを実行
SET_COM(...)	SET_COM ｺマﾝﾄを実行
SET_COM_TYPE(n%,type%)	SET_COM_TYPE ｺマﾝﾄを実行
SET_CURSOR(status%)	SET_CURSOR ｺマﾝﾄを実行
SET_LANGUAGE(n%)	SET LANGUAGE ｺマﾝﾄを実行
SET_NET_PARAMETER(index%,a\$)	SET NET PARAMETER ｺマﾝﾄを実行
SET_PRECISION(n%)	SET PRECISION ｺマﾝﾄを実行
SET_RFID_KEY(...)	SET RFID KEY ｺマﾝﾄを実行
SET_RFID_READ(...)	SET RFID READ ｺマﾝﾄを実行
SET_RFID_WRITE(...)	SET RFID WRITE ｺマﾝﾄを実行
SET_RTS(n1%,n2%)	SET_RTS ｺマﾝﾄを実行 (n1%は、COMﾎﾞｰﾄ番号、n2%は設定値)
SET_SCREENITEMS(...)	SET_SCREENITEMS ｺマﾝﾄを実行
SET_SIGNAREA(...)	SET_SIGNAREA ｺマﾝﾄを実行
SET_VIDEO_MODE(...)	SET VIDEO MODE ｺマﾝﾄを実行
SET_WEDGE(wedgesetting\$)	SET WEDGE ｺマﾝﾄを実行
SHOW_IMAGE(...)	SHOW_IMAGE ｺマﾝﾄを実行
SIGN(n%)	SIGN ｺマﾝﾄを実行
SOCKET_CAN_SEND(...)	SOCKET CAN SEND ｺマﾝﾄを実行
SOCKET_HAS_DATA(...)	SOCKET HAS DATA ｺマﾝﾄを実行
SOCKET_OPEN(n%)	SOCKET_OPEN ｺマﾝﾄを実行
START_TCPIP	START TCPIP ｺマﾝﾄを実行
STOP_BEEP	STOP BEEP ｺマﾝﾄを実行
STOP_TCPIP	STOP TCPIP ｺマﾝﾄを実行
STR\$(n%)	STR\$ ｺマﾝﾄを実行
STRING\$(...)	STRING\$ ｺマﾝﾄを実行
SUB(n1%,n2%)	引き算を実行
SYSTEM_INFORMATION(index%)	SYSTEM INFORMATION ｺマﾝﾄを実行
SYSTEM_PASSWORD(a\$)	SYSTEM PASSWORD ｺマﾝﾄを実行
T(n%)	ｽﾀｯｸﾊﾞｰﾙを表示 ﾌﾞﾛｯｸをｺｰﾙすると、ﾊﾞｰﾙが1増加し、ﾌﾞﾛｯｸからﾘﾀｰﾝすると1減少 ｽﾀｯｸｵｰﾊﾞｰﾌﾛｰｲﾗ-発生時のデバックに利用します
TCP_ERR_CODE	TCP_ERR_CODE ｺマﾝﾄを実行
TCP_OPEN(...)	TCP_OPEN ｺマﾝﾄを実行
TIME\$	TIME\$ ｺマﾝﾄを実行 (ｼｽﾃﾑ時刻の取得)

デバッグメッセージ	説明
TIME\$(x\$)	TIME\$コマンドを実行 (システム時刻の設定)
TIMER	TIMER コマンドを実行
TRANSACTION COUNT	TRANSACTION COUNT コマンドを実行
TRANSACTION COUNT_EX(file%)	TRANSACTION COUNT EX コマンドを実行
TRIM_LEFT\$(x\$)	TRIM_LEFT\$コマンドを実行
TRIM_RIGHT\$(x\$)	TRIM_RIGHT\$コマンドを実行
UCASE\$(x\$)	UCASE\$コマンドを実行
UNLOCK	UNLOCK コマンドを実行
UPDATE_BASIC(file%)	UPDATE_BASIC コマンドを実行
UPDATE_RECORD(...)	UPDATE_RECORD コマンドを実行
UPDATE_TRANSACTION(n%,data\$)	UPDATE_TRANSACTION コマンドを実行
UPDATE_TRANSACTION_EX(...)	UPDATE_TRANSACTION_EX コマンドを実行
VAL(x\$)	VAL コマンドを実行
VALR(x\$)	VALR コマンドを実行
VERSION(a\$)	VERSION コマンドを実行
VIBRATOR(x\$)	VIBRATOR コマンドを実行
WAIT(duration\$)	WAIT コマンドを実行
WAIT_HOURLASS(...)	WAIT HOUR GLASS コマンドを実行
WEDGE_READY	WEDGE_READY コマンドを実行
WRITE_COM(n%,a\$)	WRITE_COM コマンドを実行
XOR	論理演算子 XOR を実行

デバックメッセージ例

```
* L(7), T(0)
  ADD_RECORD(1,"10001      Justin      Jan
08300930113013001130150018002000")

* L(8), T(0)
* L(9), T(0)
  ASGN(2)

* L(10), T(0)
  ASGN(3)

* L(11), T(0)
  ASGN("CipherLab 510")

* L(12), T(0)
  ASGN("510AC_100.BAS")

* L(13), T(0)
  ...

* L(25), T(0)
  ARY(1)
  ASGN("OK Good Morning!")
  ...

* L(39), T(0)
  SET_COM(1,1,1,2,1)

* L(40), T(0)
  OPEN_COM(1)
  ...

* L(41), T(0)
  START_NETWORK
```

```
* L(42), T(0)
  ON_NET(316)

* L(43), T(0)
  ON_ENQUIRY(128)
  ...

* GOTO(68)
  L(68), T(0)

* L(69), T(0)
* L(70), T(0)
  GOTO(68)
  ...

* L(69), T(0)
  EVENT(16)

* L(79), T(1)
* L(80), T(1)
  OFF_READER(1)

* L(81), T(1)
  OFF_READER(2)

* L(82), T(1)
  CLS

* L(83), T(1)
  HIDE_CALENDAR

* L(84), T(1)
  BEEP(...)
```

補足 4. ランタイムエラー

エラーコード	説明
1	Unkown operator
2	Operand count mismatch
3	Type mismatch
4	Can't perform type conversion
5	No available temp string
6	Illegal operand
7	Not an L-value
8	Float error
9	Bad array subscript
10	Unkown function
11	Illegal function call
12	Return without GOSUB

補足 5. キーコードテーブル

Key Name						Key Code
8000	8200	8300	8400	8500	8700	
CLEAR	CLEAR	CLEAR	CLEAR	CLEAR	CLEAR	1
BS	BS	BS	BS	BS	BS	8
CR	CR	CR	CR	CR	CR	13
ESC	ESC	ESC	ESC	ESC	ESC	27
	SP		SP			32
#	#	#	#	#	#	35
\$	\$	\$	\$	\$	\$	36
%	%	%	%	%	%	37
	&		&	&	&	38
	((40
))			41
*	*	*	*	*	*	42
+	+	+	+	+	+	43
,	,	,	,	,	,	44
-	-	-	-	-	-	45
.	46
/	/	/	/	/	/	47
0	0	0	0	0	0	48
1	1	1	1	1	1	49
2	2	2	2	2	2	50
3	3	3	3	3	3	51
4	4	4	4	4	4	52
5	5	5	5	5	5	53
6	6	6	6	6	6	54
7	7	7	7	7	7	55
8	8	8	8	8	8	56
9	9	9	9	9	9	57
	:		:			58
;	;	;	;	;	;	59
	<		<	<	<	60
	=		=			61
	>		>	>	>	62

Key Name						Key Code
8000	8200	8300	8400	8500	8700	
A	A	A	A	A	A	65
B	B	B	B	B	B	66
C	C	C	C	C	C	67
D	D	D	D	D	D	68
E	E	E	E	E	E	69
F	F	F	F	F	F	70
G	G	G	G	G	G	71
H	H	H	H	H	H	72
I	I	I	I	I	I	73
J	J	J	J	J	J	74
K	K	K	K	K	K	75
L	L	L	L	L	L	76
M	M	M	M	M	M	77
N	N	N	N	N	N	78
O	O	O	O	O	O	79
P	P	P	P	P	P	80
Q	Q	Q	Q	Q	Q	81
R	R	R	R	R	R	82
S	S	S	S	S	S	83
T	T	T	T	T	T	84
U	U	U	U	U	U	85
V	V	V	V	V	V	86
W	W	W	W	W	W	87
X	X	X	X	X	X	88
Y	Y	Y	Y	Y	Y	89
Z	Z	Z	Z	Z	Z	90
				[[91
			\	\	\	92
]]	93
				^	^	94
a	a	a	a	a	a	97
b	b	b	b	b	b	98
c	c	c	c	c	c	99
d	d	d	d	d	d	100
e	e	e	e	e	e	101
f	f	f	f	f	f	102
g	g	g	g	g	g	103

Key Name						Key Code
8000	8200	8300	8400	8500	8700	
h	h	h	h	h	h	104
i	i	i	i	i	i	105
j	j	j	j	j	j	106
k	k	k	k	k	k	107
l	l	l	l	l	l	108
m	m	m	m	m	m	109
n	n	n	n	n	n	110
o	o	o	o	o	o	111
p	p	p	p	p	p	112
q	q	q	q	q	q	113
r	r	r	r	r	r	114
s	s	s	s	s	s	115
t	t	t	t	t	t	116
u	u	u	u	u	u	117
v	v	v	v	v	v	118
w	w	w	w	w	w	119
x	x	x	x	x	x	120
y	y	y	y	y	y	121
z	z	z	z	z	z	122
F1	F1	F1	F1	F1	F1	128
F2	F2	F2	F2	F2	F2	129
F3	F3	F3	F3	F3	F3	130
F4	F4	F4	F4	F4	F4	131
F5	F5	F5	F5	F5	F5	132
F6	F6	F6	F6	F6	F6	133
F7	F7	F7	F7	F7	F7	134
F8	F8	F8	F8	F8	F8	135
F9	F9	F9	F9	F9	F9	136
F0	F0	F0	F10	F10	F10	137
			F11	F11	F11	138
			F12	F12	F12	139
UP	UP	UP	UP	UP	UP	140
DOWN	DOWN	DOWN	DOWN	DOWN	DOWN	141
	LEFT	LEFT	LEFT	LEFT	LEFT	142
	RIGHT	RIGHT	RIGHT	RIGHT	RIGHT	143
		FP	F13	F13	F13	144
		FQ	F14	F14	F14	145

Key Name						Key Code
8000	8200	8300	8400	8500	8700	
		FR	F15	F15	F15	146
		FS	F16	F16	F16	147
		FT	F17	F17	F17	148
		FU	F18	F18	F18	149
		FV	F19	F19	F19	150
		FW	F20	F20	F20	151
		FX		F21	F21	152
		FY		F22	F22	153
		FZ		F23	F23	154
FESC	FESC	FESC	FESC	FESC	FESC	155
				F24	F24	156
			TAB	TAB	TAB	160
				INSERT	INSERT	161
			DEL	DEL	DEL	162
		FD				163
		FH				167
RCR (right CR)		RCR (right CR)				169
LCR (left CR)		LCR (left CR)				170
		FL				171
		FM				172
		FO				174