

Windows 用ターミナル ソフト

あじゃた～む

マクロ機能

操作説明書

(1. 6. 2. 2 版)

目次

1.	概要	1
1.1.	マクロテキストファイルの形式	1
1.2.	起動方法	1
2.	言語仕様	3
2.1.	変数名	3
2.2.	数値型変数のタイプ	3
2.3.	変数の定義	3
2.4.	変数のスコープ	4
2.5.	定義済定数	4
2.6.	数式	5
2.7.	数値定数	5
2.8.	文字列	5
2.9.	文字列の代入／比較	6
2.10.	配列の代入	6
2.11.	処理ブロック	7
2.12.	break / goto 文	7
2.13.	関数の実行	8
2.14.	書式文字列	9
2.15.	エスケープシーケンス	9
2.16.	コメント	9
2.17.	イベント	10
2.18.	C言語との相違点	11
3.	「あじやたへむ」との接続	12
4.	デバッグ機能	13
4.1.	トレース表示	13
4.2.	ステップトレース	14
5.	マクロテキストの解説	15
5.1.	イベントドリブン	15
5.2.	ボタン／チェックボックス操作	15
5.3.	シリアルポート送受信 (テキストデータ)	17
5.4.	シリアルポート送受信 (バイナリデータ)	18
5.5.	プロファイルへの記録	20
5.6.	右クリックイベントとポップアップメニュー	23
5.7.	タイムチャートグラフ表示とタイマ	25
5.8.	イベントドリブン	28
5.9.	サンプルマクロテキスト	29

1. 概要

C言語ライクな言語仕様によるマクロ・インタプリタ実行プログラムです。
プログラムファイル（マクロテキストファイル）を入力して、そのまま実行することができます。

このソフトウェアは、バイト文字（日本の場合はシフト J I S）対応のアプリケーションです。UNICODE は使用できません。

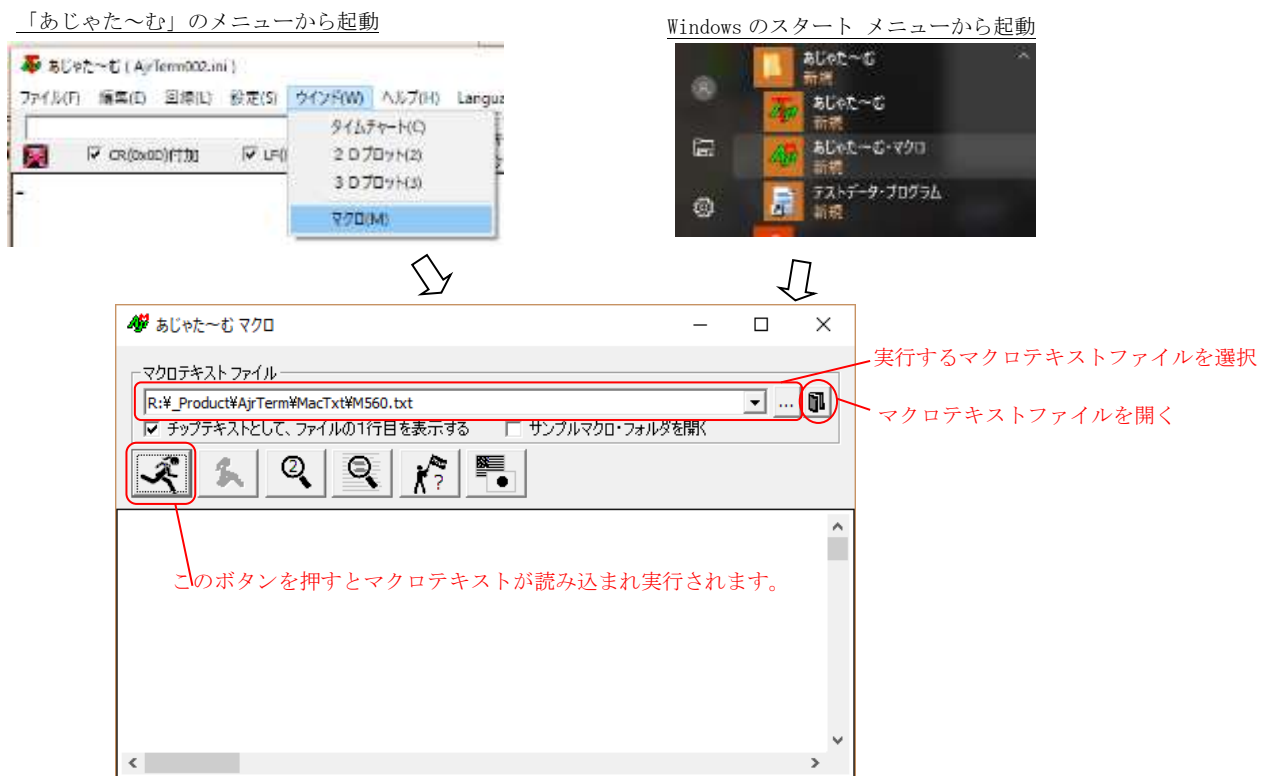
1.1. マクロテキストファイルの形式

マクロテキストファイルは、規定のバイト文字（日本の場合はシフト J I S）ファイルでなければなりません。

1.2. 起動方法

「あじやた〜む・マクロ」は、以下の方法で起動します。

- ・「あじやた〜む」の「ウインド」メニューから「マクロ」を選択する
- ・Windows のスタートメニューから、「あじやた〜む」→「あじやた〜む・マクロ」を選択する
- ・エクスプローラ等で AjrMacro32.exe / AjrMacro64.exe を実行する
- ・エクスプローラ等から、AjrMacro32.exe / AjrMacro64.exe のショートカットアイコンへマクロテキストファイルをドロップする



「チップテキストとして、ファイルの1行目を表示する」をチェックした場合、コンボボックスでのファイル選択中や、コンボボックスにカーソルを置いた際に表示されるチップテキストとして、当該ファイルの1行目が表示されます。

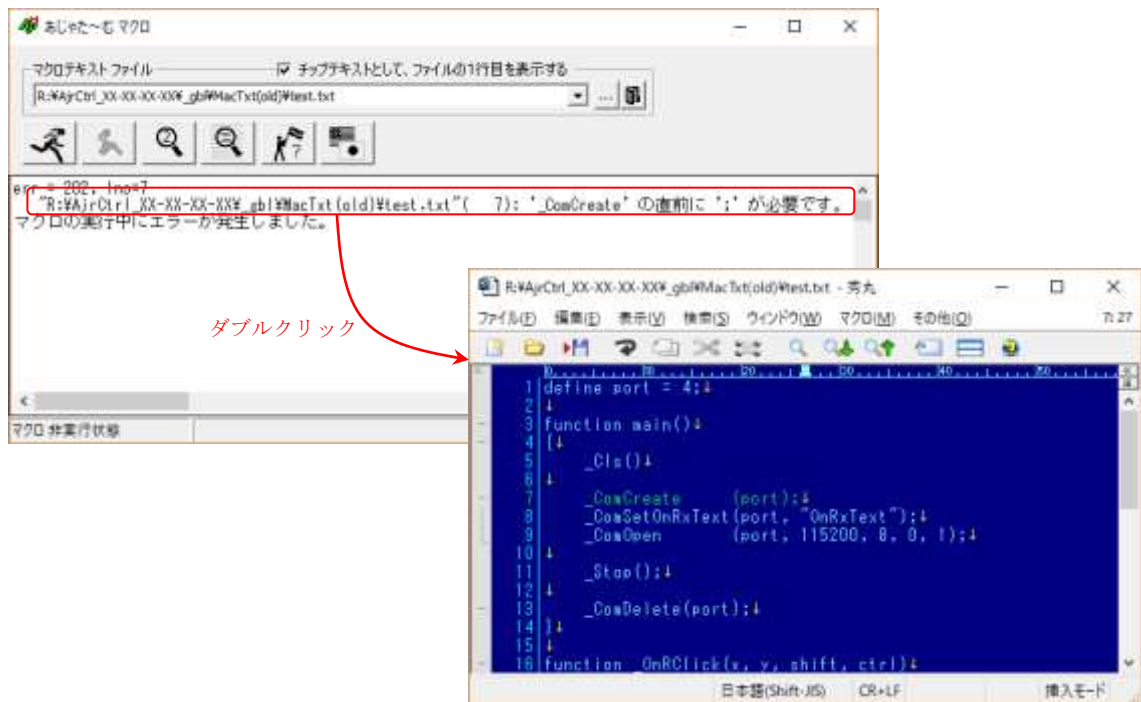
(ファイル1行目のテキストの両端から、タブ、復帰、改行、スラッシュ(/)や、アスタリスク(*)を除去したテキストを表示します)

「サンプルマクロ・フォルダを開く」をチェックした場合は、「...」ボタンでファイルを選択する際に、最初にサンプルマクロテキストのフォルダを開きます。

各ボタンの操作内容は、以下のとおりです。

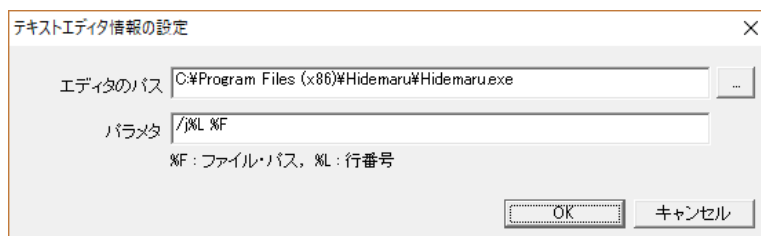
	マクロテキストファイルを読み込んで実行します。		ステップトレースウィンドを起動します
	マクロテキストの実行を中止します		バージョン表示や、説明ドキュメントの表示
	トレース表示ウィンドを起動します		言語設定

マクロテキストを読み込んで実行する際に、マクロテキストにエラーがある場合は、エラーメッセージが表示されます。このエラーメッセージ行をダブルクリックすると、テキストエディタで当該マクロテキストファイルを開きます。



テキストエディタの設定

ダブルクリックにより起動するテキストエディタを設定するには、エラーメッセージ行以外の部分をダブルクリックします。以下のダイアログが表示されるので、ここでテキストエディタの設定を行います。



「エディタのパス」には、起動するテキストエディタ・プログラム (.exe ファイル) のパスを設定します。パラメタには、コマンドラインで与えるパラメタを指定します。パラメタ中の「%F」はマクロテキストファイルのパス名に置き換えられ、「%L」は行番号に置き換えられます。

2. 言語仕様

本インタプリタは、C言語ライクな言語仕様となっていますが、C言語との相違点もあります。
(本文で記述していない事項に関しては、C言語と同じと考えてください)
以下、本インタプリタの言語仕様を述べます。

2.1. 変数名

変数名は、先頭が「半角英字、全角文字、アンダバー() or ドルマーク(\$)」で始まり、後続が「先頭文字の条件」+「半角数字」で構成します。

先頭が「\$」で始まる変数は文字変数、それ以外は数値変数となります。
文字変数は、規定のバイト文字(日本の場合はシフトJIS)で構成します。
C言語と同様に変数名の英字は、大文字/小文字を区別します。
以下の変数名は、いずれも有効な変数名です。

```
(例)  define var = 10, VAR, _abc09, $str1;
      define 東京都港区赤坂_10_2_102;
      define $私は誰でしょう = "文字変数です";
      define 3教科点数[3] = {82, 75, 96};
```

2.2. 数値型変数のタイプ

先頭がドルマーク(\$)以外の変数は、数値型変数として扱われますが、そのタイプは「符号付き整数」「符号無し整数」あるいは、「実数」となります。

どの型にするかは、マクロテキストの先頭で「option」文により指定します。

```
(例)  option signed; ----- 数値型を符号付き整数として扱う (option 指定無し時のデフォルト)
      option unsigned; ----- 数値型を符号無し整数として扱う
      option double; ----- 数値型を実数として扱う
```

整数はC言語の「signed long long」「unsigned long long」と同じです。(64ビット精度)

実数はC言語の「double」と同じです。(64ビット精度)

2.3. 変数の定義

変数は、「define」文により、以下の構文で定義します。

#	構文	内容	
1	define 変数名;	単純変数 (変数名[1]; と同じ)	単純変数
2	define 変数名 = 値;	初期値付き単純変数 (変数名[1] = 値; と同じ)	
3	define 変数名[要素数];	配列変数	配列変数
4	define 変数名[要素数] = 値;	初期値付き配列変数 (先頭要素のみ初期化)	
5	define 変数名[要素数] = {値1, 値2, ..., 値n};	初期値付き配列変数 (複数の要素を初期化)	
6	define 変数名[] = 値;	初期値付き配列変数 (要素数=1)	
7	define 変数名[] = {値1, 値2, ..., 値n};	初期値付き配列変数 (要素数=初期値の数)	

※ カンマ(,)で区切って複数の変数を定義可能です。(ex. define a, b=21, c[2]= {10, 20};)

※ 初期値が指定されていない場合、数値変数はゼロ(0)で、文字変数は空文字列("")で初期化されます。

※ 「要素数」や「初期値」は、一般の数式で指定可能です。

(ex. define var[a * 3 + b] = {a + b, a - b}; // a, b は先に定義されている必要があります)

※ 単純変数は、実際は要素数=1の配列となります。「define x;」と「define x[1];」は同じです)

※ 変数参照時に添え字([n])を付けない場合、変数名[0]と同じです。「var = 1;」と「var[0] = 1;」は同じです)

2.4. 変数のスコープ

C言語と同様に、関数の外で定義された変数は、グローバルなスコープを持ちます。(ソース全体から参照可能)

関数の中で定義された変数は、C言語の自動変数と同様に、当該関数内のスコープを持ちます。(当該関数内でのみ参照可能)

但し、関数内のブラケット({ . . . })内に定義した自動変数も、関数全体のスコープとなります。

グローバルな変数名と関数内の自動変数名が同じ場合、自動変数がアクセスされ、グローバルな変数はアクセスできません。

```
(例)  define gbl; // 広域変数 (いずれの関数からも参照可)
      define mult = 10; // 広域変数 (関数内で同名の自動変数が定義された場合は参照不可)
      function func(a, b)
      {
        define v1, v2; // 自動変数
        define mult = 20; // 自動変数 (広域変数と同名のため、この自動変数のみ参照可)
        if ( . . . ) {
          define inb = 1; // 「inb」も関数内全体のスコープを持つ自動変数となる
          . . .
        }
        v1 = inb; // 「inb」は上記の「if { . . . }」ブラケット内で定義されているが
        gbl = inb; // 関数内全体のスコープを持つため、参照可
        v2 = mult; // 自動変数の「mult」が参照され、v2=20 となります。
      }
```

関数内の「define」は、「自動変数を生成する実行命令」です。手続き上同じ「define」が2度実行されると、二重定義エラーとなります。以下のコードは「define a;」が2度実行されるため、実行時エラーとなります。

```
(例)  for (i = 0; i < 2; i++) {
      define a; // 2回目の実行で二重定義エラーとなる
      . . .
    }
```

※ 二重定義エラーを防止するために、自動変数は関数の先頭でまとめて定義しておくことをお勧めします。



2.5. 定義済定数

定義済定数一覧を以下に示します。

#	名称	値	備考
1	_TRUE	1	真値
2	_FALSE	0	偽値
3	_CR	0x0D	復帰コード
4	_LF	0x0A	改行コード
5	_SLOT_PORT	256	メールスロットのポート番号
6	_MB_OK	0	_MessageBox() の BoxType 引数
7	_MB_OKCANCEL	1	
8	_MB_RETRYCANCEL	5	
9	_MB_YESNO	4	
10	_MB_YESNOCANCEL	3	
11	_MB_ICONERROR	16	_MessageBox() の戻り値
12	_IDCANCEL	2	
13	_IDNO	7	
14	_IDOK	1	
15	_IDYES	6	_CalcCrc() の knd 引数
16	_CRC_ITL	0	
17	_CRC_ITR	1	
18	_CRC_16L	2	
19	_CRC_16R	3	
20	\$_CR	“¥r”	復帰
21	\$_LF	“¥n”	改行
22	\$_CRLF	“¥r¥n”	復帰・改行
23	\$_MyPath	マクロテキストのパス名部分	マクロのパス名部分 (「D:¥work¥Mac.txt」 → ” D:¥work¥”)
24	\$_MyName	マクロテキストファイル名	マクロのファイル名部分 (「D:¥work¥Mac.txt」 → ” Mac”)

2.6. 数式

数式の表現は、ほぼC言語と同じです。(但し、文字列の代入/比較を記述できる等、若干の違いがあります)
演算子の種類と優先順位は、以下の通りです。

#	演算子	名称	備考	優先度
1	(. . .)	優先計算部	「. . .」は数式	高   低
2	= ==, !=, <, >, <=, >=	文字列の代入 文字列の比較	文字列の代入/比較は高優先 ex. \$s[2]="AB" \$s < \$t	
3	++ -- +, -, ! ~	プリ・インクリメント, ポスト・インクリメント プリ・デクリメント, ポスト・デクリメント 単項演算子 (プラス, マイナス, 論理否定) 単項演算子 (ビット反転)	ex. ++x y++ ex. --x y-- ex. !x -10 実数の場合、使用不可	
4	*, /, %	乗算, 除算, 剰余算		
5	+, -	加算, 減算		
6	<<, >>	左シフト, 右シフト	実数の場合、2 ⁿ で乗算/除算	
7	<, >, <=, >=	大小比較	数値の比較	
8	==, !=	等値, 不等値比較		
9	&	ビット論理積	実数の場合、使用不可	
10	^	ビット排他論理和	実数の場合、使用不可	
11		ビット論理和	実数の場合、使用不可	
12	&&	論理積		
13		論理和		
14	? :	3項演算	ex, a == 1 ? 0 : 1	
15	= +=, -= *=, /=, %= &=, =, ^= <<=, >>=	代入 加減算 代入 乗除算, 剰余算 代入 ビット論理積, 論理和, 排他論理和 代入 左シフト, 右シフト 代入	数値型変数への(演算付)代入 実数の場合、使用不可 実数の場合、2 ⁿ で乗算/除算	

※ カンマ(,)演算子は使用できません。(但し、「for(初期化; 条件; 更新)」の初期化と更新はカンマで区切って複数指定可)

※ 「# (項番)」が同じものは、同一優先度です

※ 「#2」の文字列の代入(=)の場合、空文字列(“”)を代入した場合は「0」、それ以外は「1」となります。
文字列の比較(==, !=, <, >, <=, >=)は、結果が真(成立)の場合「1」、偽(不成立)の場合は「0」となります。

```
(例)  var %= 100;           $str = "ABC" ;
      var <= (a + (b[2] = 3));  $str >= $func(1, 2);
      var = (a++ + ++b);
      var = a << 2;
      (var = a + func(1, 2)) || b == 2;
```

●プリ・インクリメント/デクリメントや、ポスト・インクリメント/デクリメントについて

プリインクリメント/デクリメントやポストインクリメント/デクリメントは、「++」や「--」を変数の直前または、直後に記述し、変数を括弧で囲んだりしないでください。

```
ex.    ++a; -----OK
      b--; -----OK
      (c)++; --- エラー
```

2.7. 数値定数

数値定数の表現は、C言語と同じです。(123, -789, 3.14, 0.7G3 . . . 等)

尚、文字列 (文字型変数や、文字型関数も含む) も数値として扱うことができます。(先頭の8バイトまで)

例えば、「a = "ABC"」は、「a = 0x414243」と同じになります。

但し、文字式の場合は、文字式の評価結果となります。「a = "ABC" == "DEF"」は、「a = 0」となります)

2.8. 文字列

文字列は、連なる文字をダブルクォート(")、あるいは、アポストロフィ(')で囲みます。

ダブルクォート(")で囲んだ文字列は、C言語と同様の「¥」で始まるエスケープ文字が記述できます。

アポストロフィ(')で囲んだ文字列は、エスケープ文字を認識せずに、書いた内容がそのままの文字列となります。

```
ex.    "¥x1B[0mABC¥n" ----- ¥x1B) は1バイトの「0x1B)に、「¥n)は「0x0A)になります。
      'c:¥work¥subdir¥sample.txt' --- 書いた内容「c:¥work¥subdir¥sample.txt)がそのまま文字列となる
```

2.9. 文字列の代入／比較

数式中に文字列の代入や比較を記述することができます。

文字列の代入は、以下の構文で記述します。

```
$文字変数 = <" 文字列" / $文字変数 / $文字関数(・・・)>
```

文字列の代入結果は、空文字列(“”)を代入した場合 偽(=0)、空文字列以外を代入した場合 真(=1)です。

```
(例)  $s[1] = "ABC"
      $t = $func(1, 2)
      $u = $s[1]
```

文字列の比較は、以下の構文で記述します。

```
<" 文字列" / $文字変数 / $文字関数(・・・)> <比較演算子> <" 文字列" / $文字変数 / $文字関数(・・・)>
比較演算子: 「==」, 「!=」, 「<」, 「>」, 「<=」 or 「>=」
```

文字列は、英字の大文字と小文字を区別します。(“ABC” と “abc” は異なる文字列と判断します)

文字列の比較結果は、比較が成立する場合 真(=1)、成立しない場合 偽(=0)です。

```
(例)  $s[1] == "ABC"
      $t >= $fun(1, 2)
      $u < $s[1]
```

文字列の代入／比較は、数式中で高優先度で計算されます。(文字列代入／比較全体を括弧で囲んだのと同等です)

例えば、「if (1 - \$s == “ABC”)」は、「if (1 - (\$s == “ABC”))」と同じです。

2.10. 配列の代入

代入演算子「=」により、配列全体の代入が可能です。

配列の代入は、以下の構文で記述します。

```
変数 1 [] = 変数 2 []; ----- 変数 1 に変数 2 の全内容をコピーします。変数 1 の要素数は変数 2 と同じになります。
変数 1 [] = 関数(・・・); --- 関数は return 文で配列を返す(ex. return var[];)
```

配列の代入は、通常の数式の中では記述できません。上記構文により単独で記述してください。

```
(例)  a[] = b[]; ----- 数値型配列の代入 (配列全体をコピー)
      a[] = func(1, 2); ----- func は、return 文で配列を返す(ex. return n[]; ) .....※1
      $s[] = $t[]; ----- 文字列配列の代入 (配列全体をコピー)
      $s[] = $func(1, 2); --- $func は、return 文で配列を返す(ex. return $x[]; ) .....※1
```

※1: 配列全体を返す関数の例を以下に示します。(下記例は、数値型配列の例)

関数の戻り値に配列(添え字なしで「**変数[]**」のように記述)を指定した場合、変数には配列全体がコピーされます。

```
(例)  function func()
      {
        define arr[5] = {10, 20, 30, 40, 50};
        return arr[];
      }
      .....
      var[] = func(); // 変数「var」には配列「arr[5]」の内容がコピーされ、
                    // var[0] = 10, var[2] = 20, ... var[4] = 50 となります。
      .....

```

但し、関数の戻り値を受ける変数を通常の変数で記述した場合は、当該変数に配列の先頭の値(上記 [arr[0]])が設定されます。

上記例で、「var[] = func();」を「var[2] = func();」とした場合、var[2]には、arr[0]の内容(=10)が設定されます。

逆に、「var[] = func();」で、func() が配列を返さない(単一の値を返す)場合は戻り値が var[0]に設定されます。

関数の戻り値を受ける変数を通常の変数で記述した「var[2] = func()」のような記述は、通常の数式内でも記述できます。

2.11. 処理ブロック

「do {・・・}」(後置き while()なし) で処理ブロックを記述できます。

この処理ブロックは、繰り返し処理は行わず、単に「break」により抜けるブロックとなります。

以下の2つの構文は、同じ動作となります。

do ブロック (後置き while なし)

```
do {
    if (a == 10) break; // →①へジャンプ
    .....
}
①→
```

do ブロック (後置き while あり)

```
do {
    if (a == 10) break; // →①へジャンプ
    .....
} while(_FALSE);
①→
```

※ do ブロックの場合、ブラケット (“{” と “}”) は必須です。

その他の処理ブロック (ほとんどC言語と同じですが、**switch-case 文は使用できません**)

while ブロック (前置き while, ブラケットあり)

```
while (条件式) {
    [expression;]
    .....
}
```

while ブロック (前置き while, ブラケットなし)

```
while (条件式) [expression];
```

for ブロック (ブラケットあり)

```
for ([初期化式1][, 初期化式2] ...; [条件式]; [更新式1][, 更新式2] ...) {
    [expression;]
    .....
}
```

※ for ブロックの場合のみ、カンマ演算子(,)で区切って複数の「初期化式」「更新式」を記述できます。

for ブロック (ブラケットなし)

```
for ([初期化式1][, 初期化式2] ...; [条件式]; [更新式1][, 更新式2] ...) [expression];
```

if / else ブロック (ブラケットあり)

```
if (条件式) {
    [expression;]
    .....
}

[ else if (条件式) {
    [expression;]
    .....
}

○
○
○

[ else {
    [expression;]
    .....
} ]
```

if / else ブロック (ブラケットなし)

```
if (条件式) [expression] ;
[ else if (条件式) [expression] ; ]

○
○
○

[ else [expression] ; ]
```

※ C言語と同様に、「ブラケットあり」と「ブラケットなし」が混在してもOKです。

もちろん、各処理ブロックはネストすることができます。

2.12. break / goto 文

C言語と同様に break, goto 文を使用できます。

ex.

```
while (!fEnd) {
    for (i = 0; i < 10; i++) {
        .....
        if (fIllegalCondition) goto label;
        .....
    }
    if (fExitCondition) break;
}
label::
```

2.13. 関数の実行

C言語と異なり、関数が後方に記述されていてもプロトタイプを定義する必要はありません。(プロトタイプは記述できません)

マクロテキスト内で定義した関数を呼び出す場合、関数の引数の個数と、引数の型(数値型/文字型)は完全に一致していなければなりません。

```
(例)      . . .
          add(1, 2); ----- OK   (引数の個数と型が完全に一致, add()は後方に定義されていても参照可)
          add(1, 2, 3); ----- エラー (引数の個数が一致しない)
          add(1, "XYZ"); ----- エラー (第2引数の型が一致しない)
          . . .
          function add(a, b)
          {
            return a + b;
          }
```

引数に'var[]'のようにインデクスを指定しない配列の場合は、配列全体を渡します。

この場合、引数を受け取る関数でも、引数を'arg[]'のようにインデクスを指定しない配列としなければなりません。

```
(例)      define var[3] = {10, 20, 30};
          sub(var[]);
          . . .
          function sub(arg[]) {
            // arg[0]=10, arg[1]=20, arg[2]=30 が設定されています。
          }
```

関数に渡す引数は、(配列全体を渡す場合も含めて) 仮引数としてコピーされたデータとなります。

従って、関数内で引数の値を変更しても、呼び出し元の変数値は変更されません。

```
(例)      define var = 30;
          sub(var);
          . . .
          function sub(arg) {
            arg = 999; // 呼び出し元の引数 (var) の値は変更されません (30 のままです)
          }
```

<注> **関数の引数に文字式を含む数式**を指定する場合は、数式全体を括弧で囲むか、数式の先頭を数値項目としてください。

- func(\$a == "ABC" + 1); --- ×
- func((\$a == "ABC" + 1)); --- ○ (\$aが"ABC"の場合「func(2)」、\$aが"ABC"以外の場合「func(1)」となる)
- func(1 + \$a == "ABC"); --- ○ (\$aが"ABC"の場合「func(2)」、\$aが"ABC"以外の場合「func(1)」となる)

2.14. 書式文字列

「_Printf()」「\$_SPrintf()」「_Trace()」では、書式文字列により、文字列を生成します。
この書式はC言語(MS-VC)とほぼ同等ですが、以下の点が少々異なります。

- 1) 実数モードで、整数書式(書式の末尾が「d, i, u, x, or X」)の場合、対応する引数の数値を「long long型」に変換します。(※1)
- 2) 整数モードで、実数書式(書式の末尾が「f, e, E, g, or G」)の場合、対応する引数の数値を「double型」に変換します。(※1)
- 3) サイズ指定子「l」「ll」「l64」は指定する必要はありません。(整数書式の場合、自動的に付加します)
- 4) ワイド文字を意識した書式には対応していません。
大文字の「S / C」は小文字の「s / c」と同様に扱います。
「S, C, s, c」にサイズ指定子「l」や「h」は指定できません。
- 5) 生成する文字列は、半角換算で1023文字以内(1023バイト以内)でなければなりません。

※1: 数値の型に関係なく、整数値を扱う場合「%d」、「%x」や「%u」、実数値を扱う場合「%f」や「%g」を書式指定すればOKです。
また、桁数を指定した「%5d」、「%-5d」、「%04x」、「%*d」、「%7.2f」や「%.3f」といった書式も有効です。

2.15. エスケープシーケンス

画面表示のテキスト内に以下のエスケープシーケンスを含むことができます。

#	ESC シーケンス	内容
1	ESC・[0J	カーソル位置～最終行の右端までクリアー
2	ESC・[1J	先頭行の左端～カーソル位置までクリアー
3	ESC・[2J ESC・*	画面をクリアーし、カーソルをホームへ移動
4	ESC・[OK ESC・[K	カーソル位置～同行右端までをクリアー
5	ESC・[1K	カーソル行の左端～カーソル位置までをクリアー
6	ESC・[2K	カーソル行をクリアー
7	ESC・[s	カーソル位置を退避
8	ESC・[u	カーソル位置を回復
9	ESC・[>5l	カーソル表示
10	ESC・[>5h	カーソル非表示
11	ESC・[p1;pcH	カーソル位置設定 (p1は行位置(1～), pcは桁位置(1～))
12	ESC・[pnA	カーソルを上方向に移動 (pnは移動行数であり、省略時は1を仮定)
13	ESC・[pnB	カーソルを下方向に移動 (" ")
14	ESC・[pnC	カーソルを右方向に移動 (pnは移動桁数であり、省略時は1を仮定)
15	ESC・[pnD	カーソルを左方向に移動 (" ")
16	ESC・[pnM	カーソル行以降をスクロールアップ (pnはスクロール行数)
17	ESC・[pnL	カーソル行以降をスクロールダウン (" ")
18	ESC・[psm	描画属性設定 (ps=属性値、0=デフォルト属性、7=反転、30～37(文字色)=黒、赤、緑、黄、青、紫、水色、白、40～47=背景色)
19	ESC・D	カーソルを1行下へ移動
20	ESC・E	カーソルを1行下の左端へ移動
21	ESC・M	カーソルを1行上へ移動

[注] 「ESC・」はエスケープコード(0x1B)を意味します。(ex. _Printf(“%x1b[31mABC”); → “ABC”を赤色表示)

2.16. コメント

コメントは、C言語と同様に「/*」と「*/」でサンドイッチするか、「//」から行末までとなります。
「/*」と「*/」でサンドイッチする場合、複数行に渡っていてもOKです。
コメントのネストはできません。

2.17. イベント

マクロテキストの実行中に以下のイベントを処理することができます。

いずれのイベントも多重には発生しません。

イベントの実行中にさらにイベントが発生した場合や、イベントが禁止状態である場合にイベントが発生した場合の動作については、下表の「多重発生時」の欄を参照。

標準イベント処理関数

#	イベント 処理関数名	イベント	イベント 禁止/許可	多重 発生時	備考
1	<code>_OnKeyIn</code>	キー入力	<code>_DisStdEvt</code> <code>_EnaStdEvt</code>	全保留	
2	<code>_OnRClick</code>	右クリック	〃	1 回の み保留	Shift や Ctrl キーを押下しない場合は <code>_EnableStdRClk(FALSE)</code> 実行要
3	<code>_OnFileSearch</code>	ファイル検索フォルダ通知	〃	破棄	
4	<code>_OnTmcClose</code>	タイムチャートグラフ クローズ	〃	全保留	
5	<code>_OnButton</code>	ボタン押下	〃	全保留	
6	<code>_OnChkBox</code>	チェックボックス設定	〃	全保留	
7	<code>_OnTimer</code>	タイマ周期通知	〃	1 回の み保留	但し、タイマ停止時は、残留イベントを破棄
8	<code>_OnMenu</code>	ポップアップメニュー選択	〃	全保留	実際の関数名は「 <code>_PopupMenu()</code> 」で指定

COMポート通信イベント処理関数

#	イベント 処理関数名	イベント	イベント 禁止/許可	多重 発生時	備考
1	<code>_OnRxText</code>	テキスト受信通知	<code>_DisComEvt</code> <code>_EnaComEvt</code>	全保留	
2	<code>_OnRxCtrl</code>	制御コード受信通知	〃	全保留	
3	<code>_OnRxEsc</code>	E S Cシーケンス受信通知	〃	全保留	
4	<code>_OnRxPacket</code>	パケット受信通知	〃	全保留	

あじやた〜む接続イベント処理関数

#	イベント 処理関数名	イベント	イベント 禁止/許可	多重 発生時	備考
1	<code>_OnTermLink</code>	「あじやた〜む」接続	<code>_DisStdEvt</code> <code>_EnaStdEvt</code>	全保留	「あじやた〜む」からマクロ・ウインドを起 動した場合に発生
2	<code>_OnTermUnlink</code>	「あじやた〜む」切断	〃	全保留	「あじやた〜む」が終了したことを通知
3	<code>_OnTermOpened</code>	「あじやた〜む」上のポー トがオープンされた	〃	全保留	
4	<code>_OnTermClosed</code>	「あじやた〜む」上のポー トがクローズされた	〃	全保留	

※ `main()` 等、プログラム開始からの延長線で実行されるコードをメインコードといいます。

これに対し、上記イベントで実行されるコードをイベントコードといい、イベントコードは非同期に実行されます。

イベントは、メインコードの各ステップの間に割り込んで実行されます。

ステップとは、`if` 等の条件式や実行コードの式が単位となりますが、途中に関数の呼び出しがある場合は関数呼び出しの直前までとなります。また、`while/for` 等の制御構造の末尾(`;` や `}`)も1つの実行ステップとなります。

(ステップトレースウインドを起動すれば、ステップ実行の様子を見ることができます)

2.18. C言語との相違点

C言語との、およその相違点を以下にまとめます。

- 1) 「#」で始まるプリプロセス文 (#include, #if や#define 等) は使用できません。
- 2) switch-case 文は使用できません。
- 3) ポインタは使用できません。
- 4) 変数は define 文で定義します。
- 5) 変数のタイプは、数値型と文字型で、数値型は「符号付き整数」「符号無し整数」or「実数」に統一されます。
数値演算式の評価は、統一された型（「符号付き整数」「符号無し整数」or「実数」）で行われます。
- 6) 関数は function 文で定義し、引数は変数名だけを列記します。
- 7) 関数のプロトタイプを定義する必要はありません。
- 8) 構造体(struct)や共用体(union)は使用できません。
- 9) プログラム (マクロテキスト) は、1つのテキストファイルにしなければなりません。
- 10) 配列は、1次元配列のみ使用可能です。
- 11) 配列の要素数には、変数を含め一般の数式が使用できます。
- 12) 関数内で定義された自動変数は、(ブラケット「{...}」でネストしても) すべて関数内全体のスコープとなります。
- 13) 後置き while 「do {...} while(式);」の while は省略できます。
この場合「do {...}」は、単に break で抜けるための処理ブロックとなります。
- 14) マクロテキスト内で定義する関数では、可変個引数「...」は使用できません。
- 15) 文字式 (文字列の代入や比較) が使用できます。

3. 「あじやた〜む」との接続

「あじやた〜む」と接続し、「あじやた〜む」が接続しているシリアル回線を経由してデータを送受信することができます。「あじやた〜む」と接続するには、以下の手順を実行します。

- 1) 「あじやた〜む」の「ウインド」→「マクロ」メニューから、マクロウインドを起動します。
- 2) マクロテキスト内に、「あじやた〜む」接続通知イベント関数「_OnTermLink()」を作成しておき、以下のように処理します。

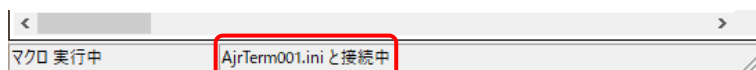
```
define bno;

function _OnTermLink($MySlot, $RmtSlot)
{
    // メモリブロック確保 (パケットデータ受信用)
    bno = _MemAlloc(1024);
    // ターミナルとの接続用メールスロット・ポート生成
    _ComCreate($MySlot);
    // 受信イベント関数設定
    _ComSetOnRxText (_SLOT_PORT, "OnText");
    _ComSetOnRxCtrl (_SLOT_PORT, "OnCtrl");
    _ComSetOnRxEsc (_SLOT_PORT, "OnEsc");
    _ComSetOnRxPacket(_SLOT_PORT, "OnPacket", bno);
    // ターミナルとの接続用メールスロット・ポートオープン
    _ComOpen($RmtSlot);
    . . . . .
}

```

_ComSetOnRxXXXX() は、必要なイベントについてだけ実行します。
例えば、テキストデータの受信だけが
必要な場合は、_ComSetOnRxText() だけを
実行します。

「あじやた〜む」と正常に接続した場合、ウインド下部に「AjrTermXXX.ini と接続中」と表示されます。



(AjrTermXXX.ini は、「あじやた〜む」のウインドタイトルに表示されている内容です)

これで、「あじやた〜む」上のシリアル回線にデータが受信された場合、「OnText()」「OnCtrl()」「OnEsc()」「OnPacket()」が実行され、それぞれ受信した「テキストデータ」「制御コード」「ESC シーケンス」「パケットデータ」が通知されます。

これらの関数は、マクロテキスト内に作成しておきます。

「あじやた〜む」上のシリアル回線にデータを送信するには、_ComSendText(), _ComSendBinary()や、_ComSendPacket()関数を実行します。(port 引数に「_SLOT_PORT」を指定します)

「あじやた〜む」と接続し、「あじやた〜む」上のシリアル回線で送受信する実際の例は、サンプルテキスト (MacTxt¥M200.txt) を参照してください。

「あじやた〜む」と接続する場合、「あじやた〜む」設定メニューで「受信テキストエンコード」と「送信テキストエンコード」を「規定のマルチバイト(SJIS)」に設定しておいてください。(シフト J I S コードでの送受信のみ可能)


4. デバッグ機能

デバッグ用の機能として、以下の内部関数があります。

内部関数

関数名	機能	備考
_VarDump	トレース表示ウインドへ全ての変数情報を表示します	
_Trace	トレース表示ウインドへ書式文字列を表示します。	
_Pause	全ての実行を一時停止します	
_StepTrace	ステップトレースの表示を許可/禁止します。	ステップトレースする範囲を限定できます

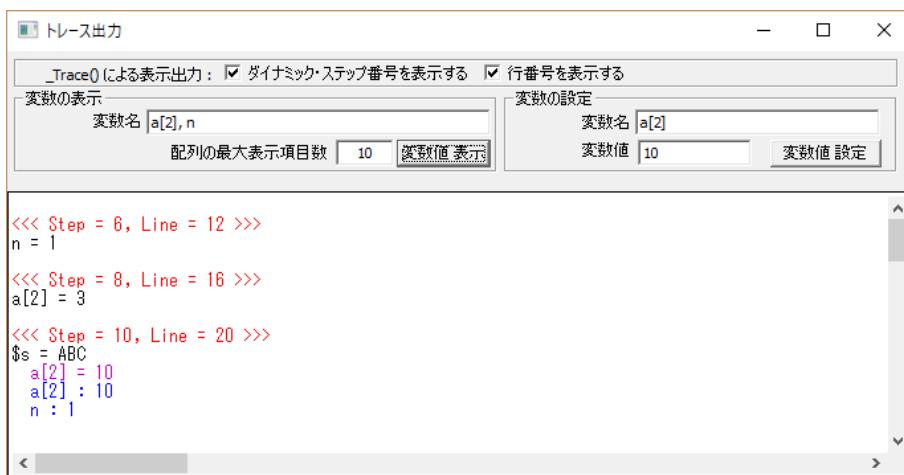
4.1. トレース表示

 ボタンを押すと、トレース表示ウインドが表示されます。

トレース表示ウインドは、内部関数「_Trace()」実行時にも自動的に表示されます。

トレース表示ウインドは、_Trace()内部関数による出力内容を表示します。

_Trace()内部関数の構文は、C言語の printf() と同じで、プログラムの内部情報を表示することができます。



2つのチェックボックス「動的ステップ番号を表示する」と「行番号を表示する」は、_Trace()による表示出力に、動的ステップ番号や行番号情報を付加します。(上記表示例中の「<<< Step = 6, Line = 12 >>>」等)

動的ステップ番号とは、実行開始時のステップを1とした、実行開始時からの動的な実行ステップ数を示します。

行番号は、マクロテキストファイルの行番号を示します。

また、内部関数「_Pause()」によりマクロテキストの実行を停止した状態で、変数の表示や設定が可能です。

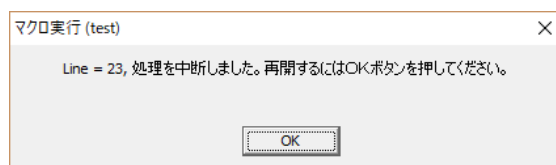
(他にも、ステップトレースウインドからブレークポイントの設定や、一時停止ボタン等による停止が可能です)

「変数の表示」中の「変数名」に、表示したい変数を入力し、「変数値表示」ボタンを押すと、トレース表示ウインドに当該変数の値が表示されます。複数の変数を表示する場合は、カンマ(,)で区切って変数名を入力します。

「配列の最大表示項目数」は、変数が配列である場合の表示する最大の配列要素数を指定します。

「変数の設定」中の「変数名」に設定したい変数名を、「変数値」に設定する値を入力し、「変数値設定」ボタンを押すと、当該変数に値が設定されます。変数の設定では、1つの変数名だけを入力してください。また、配列を指定する場合は、配列の要素を特定するように変数名を入力します。(ex. arr[3])


内部関数「_Pause()」では、以下のようなメッセージが表示されますので、「OK」ボタンを押すと実行を再開します。

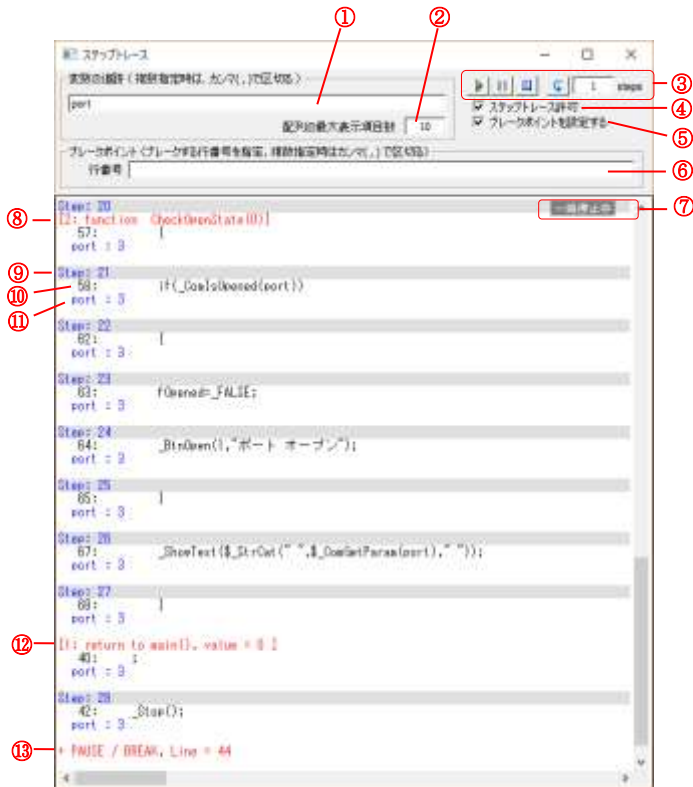








※ ステップトレース・ウインドを開いている場合は、上記メッセージは表示されません。

この場合は、ステップトレース・ウインドの実行継続ボタンを押すことにより実行を再開します。

4.2. ステップトレース

 ボタンを押すと、ステップトレースウインドが表示されます。但し、ステップトレース・ウインドはマクロテキストの実行中には開くことができず、また、マクロ実行中はウインドを閉じることができません。(つまり、マクロテキストを実行する前に開いておかなければなりません) ステップトレース・ウインドの外観は以下のとおりです。




#	内容
1	実行ステップ毎に表示する変数を指定
2	①で指定した変数が配列の場合、表示する最大項要素数
3	マクロテキストの実行操作ボタンとステップ数  : 実行継続 (マクロテキストの実行再開)  : 一時停止  : 停止 (マクロテキストの実行中止)  : ステップ実行 (指定ステップ数実行後に、再び一時停止状態となる) <input type="text" value="1"/> steps : ステップ実行するステップ数
4	ステップトレースの許可(チェック)/禁止(未チェック) ステップトレースを禁止した場合は、何も表示なくなります。 _StepTrace()内部関数でも同様の操作が可能です。
5	ブレークポイントの有効化(チェック)/無効化(未チェック), ⑥に入力した行番号でブレークします。
6	ブレークする行番号の入力(⑤)が未チェックの場合は表示されません)
7	実行状態の表示(実行中/一時停止中/停止中(マクロテキストが実行されていない状態))
8	関数の実行開始を示す表示(この例では、CheckOpenState()の実行が開始したことを示す) 先頭の数字「n:」は、関数のネストレベルを意味します。
9	ダイナミックステップ番号
10	マクロテキストの行番号と、実行ステップ命令の表示
11	①で指定した変数値の表示
12	関数からの戻りを示す表示(この例では、戻り値=0で、main()関数へ戻ったことを示す) 先頭の数字「n:」は、関数のネストレベルを意味します。
13	一時停止状態となったことを示す。以下の条件で一時停止状態となります。 <ul style="list-style-type: none"> ・Pause()が実行された ・⑥で指定した行番号(ブレークポイント)に達した ・一時停止ボタン() 押下 ・ステップ実行ボタン() によるステップ実行満了

5. マクロテキストの解説

「あじゃた〜む・マクロ」では、内部関数（C言語の `printf()` や `strcpy()` 等のような関数の集合）を呼び出すことにより、さまざまな機能を実現できます。

内部関数・全体の説明については「AjmQuick.pdf」を参照ください。

「AjmQuick.pdf」は、マクロウインドの  ボタンで表示されるポップアップメニューから、「クイックリファレンス」を選択すれば表示されます。（但し、.pdf ファイルに関連付けされたプログラム (Acrobat Reader 等) が必要です）

ここでは、いくつかの内部関数を使用したマクロテキストの例を示して、マクロの解説をします。

5.1. イベントドリブン

「あじゃた〜む・マクロ」によるプログラミングは、イベント駆動による処理を基本としています。

マクロは、引数無しの `main()` 関数から実行を開始します。

通常、`main()` 関数は、初期設定を行い「`_Stop()`」を実行します。「`_Stop()`」はメインコードの実行を停止しイベントの発生を待ちます。

イベントが発生すると、イベントに対応付けられた特定の関数が実行され、再びイベントの発生待ち状態となります。

いずれかのイベントで、「`_Restart()`」を実行すると、イベント待ち状態は解除され、「`_Stop()`」の後へ制御が移ります。

通常、「`_Stop()`」後には、後処理を記述しておきます。

5.2. ボタン/チェックボックス操作

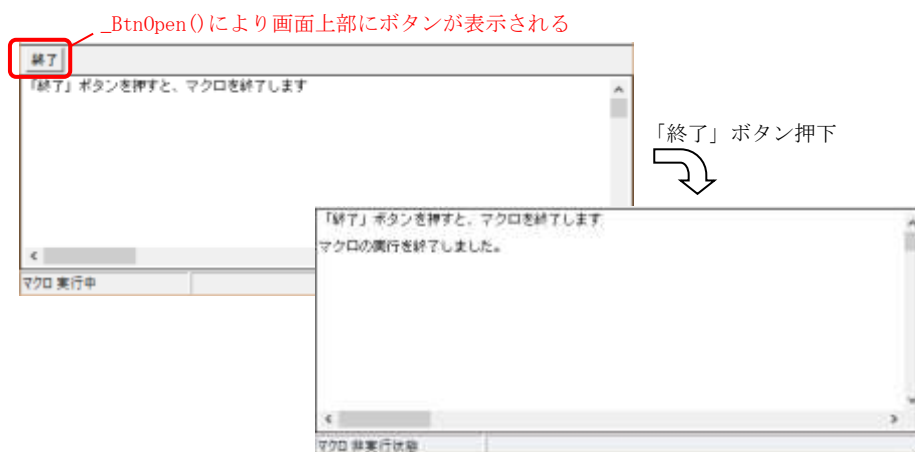
次に示すマクロテキストは、「終了」ボタンを表示し、ボタンが押されたら「`_OnButton()`」が実行され、処理を終了します。

```

1 : function main()
2 : {
3 :   _BtnOpen(0, "終了");
4 :   _Printf("「終了」ボタンを押すと、マクロを終了します\n");
5 :   _Stop();
6 :   _BtnClose(0);
7 : }
8 :
9 : function _OnButton(BtnNo)
10 : {
11 :   _Restart();
12 : }

```

実行画面は、以下のようになります。



6行目の「`_BtnClose()`」は、ボタンを削除しますが、マクロ終了時にはすべてのリソースが破棄されますので、記述しなくてもOKです。（オープンしたものは、使用後にクローズするという、プログラムのマナーはよく見えますが・・・）

ボタンやチェックボックスは、最大8個まで表示できます。

次に示すマクロでは、少々手の込んだ処理を行います。

ボタンを「送信」と「終了」の2つ表示し、さらに、1つのチェックボックス「改行を付加」を表示します。

「終了」ボタンを押すとマクロを終了します。

「送信」ボタンを押すと、現在時刻のテキストをCOMポートへ送信します。

さらに、チェックボックス「改行を付加」がチェックされている場合は、改行コードを送信します。

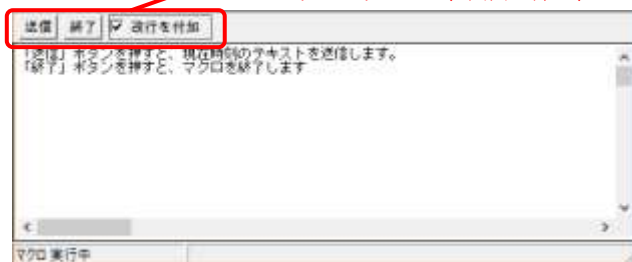
```

1 : /* M500 : ボタンとチェックボックス, COM送信 */
2 : function main()
3 : {
4 :     /* ボタン/チェックボックス表示 */
5 :     _BtnOpen (0, "送信");
6 :     _BtnOpen (1, "終了");
7 :
8 :     _ChkBoxOpen (0, "改行を付加");
9 :     _ChkBoxSetSts(0, _TRUE);
10 :
11 :     /* COMポート生成, オープン */
12 :     _ComCreate(4);
13 :     _ComOpen (4, 115200, 8, 0, 1);
14 :
15 :     _Cls();
16 :     _Printf("「送信」ボタンを押すと、現在時刻のテキストを送信します。¥n");
17 :     _Printf("「終了」ボタンを押すと、マクロを終了します¥n");
18 :
19 :     _Stop();
20 :
21 :     /* リソース破棄 */
22 :     _BtnClose(0); _BtnClose(1); _ChkBoxClose(0);
23 :     _ComClose(4); _ComDelete(4);
24 : }
25 :
26 :
27 : function _OnButton(BtnNo)
28 : {
29 :     if (BtnNo == 0) {
30 :         _ComSendText(4, $_DateAndTime());
31 :         if (_ChkBoxGetSts(0)) _ComSendText(4, $_LF);
32 :     }
33 :     else if (BtnNo == 1) _Restart();
34 : }

```

実行画面は、以下のようになります。

ボタンとチェックボックスは、画面上部に、ボタン0～7、チェックボックス0～7の順で表示します



5.3. シリアルポート送受信（テキストデータ）

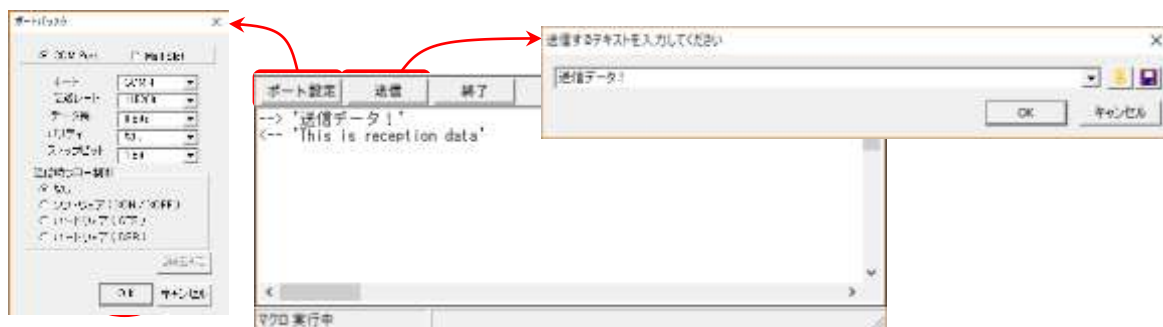
次に示すマクロは、シリアルポートを介して、テキストデータの送受信を行います。
シリアルポートの通信パラメータは、「ポート設定」ボタンでダイアログにより設定します。
送信するテキストは、ダイアログにより入力し、末尾に改行コードを付加して送信します。
受信したテキストデータは、そのまま表示します。

```

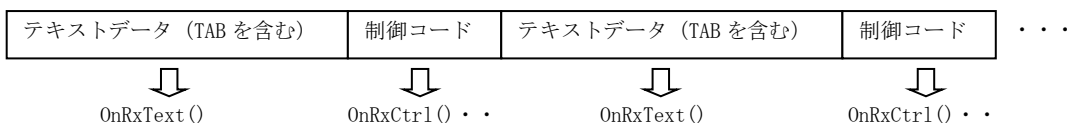
1 : // M510 : シリアルポートを介して、テキストデータの送受信
2 :
3 : define port = 0;
4 : define $InpTxt = "";
5 :
6 : function main()
7 : {
8 :     // ボタン表示
9 :     _BtnOpen(0, "ポート設定");
10 :    _BtnOpen(1, "送信");
11 :    _BtnOpen(2, "終了");
12 :
13 :    _BtnEnable(1, _FALSE);
14 :
15 :    _Cls();
16 :
17 :    _Stop();
18 :
19 :    /* リソース破棄 */
20 :    _BtnClose(0); _BtnClose(1); _BtnClose(2);
21 :    if (port != 0) {
22 :        _ComClose (port);
23 :        _ComDelete (port);
24 :    }
25 : }
26 :
27 : // ボタン・イベント処理
28 : function _OnButton (BtnNo)
29 : {
30 :     define $txt;
31 :
32 :     if (BtnNo == 0) { // ●「ポート設定」ボタン押下
33 :         if (port = _ComCreateByDialog(port)) {
34 :             _ComSetOnRxText (port, "OnRxText");
35 :             _ComSetOnRxCtrl (port, "OnRxCtrl");
36 :             if (!_ComIsOpened(port) && _ComIsOpenPossible(port)) {
37 :                 _ComOpen (port);
38 :             }
39 :             if (_ComIsOpened(port)) _BtnEnable(1, _TRUE);
40 :             else _BtnEnable(1, _FALSE);
41 :         }
42 :     }
43 :     else if (BtnNo == 1) { // ●「送信」ボタン押下
44 :         if ($InpTxt = $_Input("送信するテキストを入力してください", $InpTxt)) {
45 :             _ComSendText (port, $_StrCat($InpTxt, "\n"));
46 :             _Printf("--> '%s'\n", $InpTxt);
47 :         }
48 :     }
49 :     else if (BtnNo == 2) _Restart(); // ●「終了」ボタン押下
50 : }
51 :
52 : // テキスト受信イベント処理
53 : function OnRxText($Text)
54 : {
55 :     _Printf("<-- '%s'", $Text);
56 : }
57 :
58 : // 制御コード受信イベント処理
59 : function OnRxCtrl(ctrl)
60 : {
61 :     _Printf("%c", ctrl);
62 : }

```

実行画面は、以下のようになります。



- ・「ポート設定」ボタンが押された場合の処理 (行 32~41)
 _ComCreateByDialog() は、ダイアログによるシリアルポートのパラメタ設定を行い、指定されたパラメタでポートを生成します。
 (キャンセルボタンを押した場合は、何もせずに _FALSE を返します)
 _ComSetOnRxText() と _ComSetOnRxCtrl() は、それぞれ、「テキストを受信時の実行関数」「制御コード受信時の実行関数」を指定します。
 ポートがクローズ状態(!_ComIsOpened(port))で、ポートがオープン可能 (_ComIsOpenPossible(port)) ならば、ポートをオープンし、「送信」ボタンを操作可能にします。
- ・「送信」ボタンが押された場合の処理 (行 42~47)
 \$_Input() は、ダイアログにより文字列の入力を行い、入力した文字列を返します。(キャンセル時は空文字列を返す)
 文字列が入力されたら、_ComSendText() でポートへ文字列と改行コードを送信します。
- ・テキスト受信イベント (OnRxText()) と制御コード受信イベント (OnRxCtrl())
 一連の受信データストリームは、(TAB(0x09)を除く制御コード(0x00~0x1F, 0x7F)で分離された部分テキストを「OnRxText()」へ通知し、制御コードは「OnRxCtrl()」へ1バイトづつ通知します。



ここでは、受信したデータすべてをそのまま表示出力するために、この2つのイベントを受け付けています。

5.4. シリアルポート送受信 (バイナリデータ)

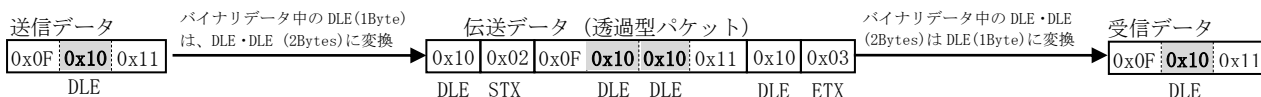
透過型パケットにより、バイナリデータの送受信ができます。

透過型パケットとは、DLE・STX で始まり、DLE・ETX で終了するバイトストリームです。(STX=0x02, ETX=0x03, DLE=0x10)



バイナリデータ中の、2つの連続する DLE は、1つの DLE に変換されます。

例えば、3バイトのバイナリデータ (0x0F, 0x10, 0x11) を送信する場合は、以下のようになります。



透過型パケットによりバイナリデータの送受信を行うには、_MemAlloc() で、バイナリデータ格納用のメモリブロックを確保しておく必要があります。

上記例の3バイトのバイナリデータを、透過型パケットで送信するには、以下のように記述します。

(シリアルポートはオープンされているものとします)

```

define bno:

bno = _MemAlloc(1024);          // メモリブロック確保 (最初に1回だけ実行)
...
_PutMemBlk(bno, 0, 3, 0x11100F); // メモリブロック (先頭3バイト) ヘデータ設定
_ComSendPacket(port, bno, 0, 3); // バイナリデータを透過型パケットで送信
...
_MemFree(bno);                // メモリブロック解放 (最後に1回だけ実行)

```

- ※ バイナリデータを透過型パケットではなく、単純なバイト列として送信する場合は、_ComSendBinary() を使用します。
 上記コード例では「_ComSendPacket(port, bno, 0, 3);」を「_ComSendBinary(port, bno, 0, 3);」とします。
 この場合は、単に3バイトのデータ (0x0F, 0x10, 0x11) が送信されます。

次に示すマクロは、先のテキスト送受信マクロ (M510) を、バイナリデータ (透過型パケット) の送受信に改造したものです。

「送信」ボタンを押すと 3 バイトのバイナリデータ (0x0F, 0x10, 0x11) を、透過型パケットとして送信します。

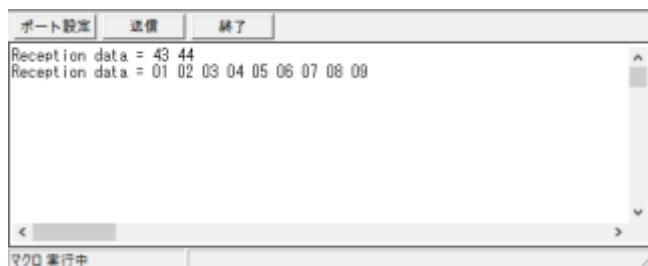
また、透過型パケットを受信すると、受信したバイナリデータを 16 進表示します。

```

1 : // M520 : シリアルポートを介して、バイナリ・パケット・データの送受信
2 :
3 : define bno_tx = 0;
4 : define bno_rx = 0;
5 : define port = 0;
6 : define $InpTxt = "";
7 :
8 : function main()
9 : {
10 : // メモリブロック確保
11 : bno_tx = _MemAlloc(1024);
12 : bno_rx = _MemAlloc(1024);
13 :
14 : // ボタン表示
15 : _BtnOpen(0, "ポート設定");
16 : _BtnOpen(1, "送信");
17 : _BtnOpen(2, "終了");
18 :
19 : _BtnEnable(1, _FALSE);
20 :
21 : _Cls();
22 :
23 : _Stop();
24 :
25 : /* リソース破棄 */
26 : _BtnClose(0); _BtnClose(1); _BtnClose(2);
27 : if (port != 0) {
28 : _ComClose(port);
29 : _ComDelete(port);
30 : }
31 : }
32 :
33 : // ボタン・イベント処理
34 : function _OnButton(BtnNo)
35 : {
36 : if (BtnNo == 0) { // ●「ポート設定」ボタン押下
37 : if (port = _ComCreateByDialog(port)) {
38 : _ComSetOnRxPacket(port, "OnRxPacket", bno_rx);
39 : if (!_ComIsOpened(port) && _ComIsOpenPossible(port)) {
40 : _ComOpen(port);
41 : }
42 : if (_ComIsOpened(port)) _BtnEnable(1, _TRUE);
43 : else _BtnEnable(1, _FALSE);
44 : }
45 : }
46 : else if (BtnNo == 1) { // ●「送信」ボタン押下
47 : // バイナリデータ (0x0F, 0x10, 0x11) を透過型パケットで送信
48 : _PutMemBlk(bno_tx, 0, 3, 0x11100F);
49 : _ComSendPacket(port, bno_tx, 0, 3);
50 : }
51 : }
52 : else if (BtnNo == 2) _Restart(); // ●「終了」ボタン押下
53 : }
54 :
55 : // パケット受信イベント処理
56 : function OnRxPacket(len)
57 : {
58 : define loc;
59 :
60 : _Printf("Reception data =");
61 : for (loc = 0; loc < len; loc++) {
62 : _Printf(" %02X", _GetMemBlk(bno_rx, loc, 1));
63 : }
64 : _Printf("\n");
65 : }
66 :

```

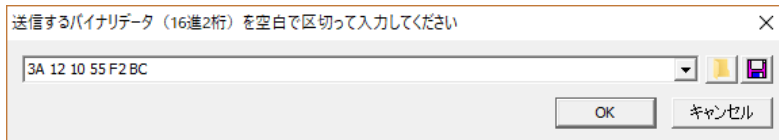
実行画面は、以下のようになります。



このサンプルマクロでは、3バイトの固定データしか送信しません。
行47~49を以下のように変更すれば、ダイアログボックスで任意のバイト列を入力し送信できます。

```
// バイナリデータの入力と、透過型パケット送信
$InpTxt = $_Input("送信するバイナリデータ (16進2桁) を空白で区切って入力してください");
if ($InpTxt != "") { // 文字列入力あり?
    if (bytes = _HexStrToBinary($InpTxt, bno_tx, 0)) { // 16進文字列→バイナリ変換
        _ComSendPacket(port, bno_tx, 0, bytes); // バイナリパケットデータ送信
    }
}
```

以下のダイアログボックスで、任意のバイト列を入力します。



行58~64の受信データのダンプ表示は、以下のように簡略化できます。

```
_DumpMemBlk(bno_rx, 0, len);
```

5.5. プロファイルへの記録

先のサンプルマクロ (M520) では、起動時に必ずダイアログでポート設定をしなければなりません。

前回の実行時に設定していたポート番号やオープン状態が分かれば、ダイアログを表示することなく、前回設定したポート番号でシリアルポートをオープンすることができます。

「あじやた〜む・マクロ」では、プロファイルへ情報を記録したり、読み出すことができます。

プロファイルへ数値を記録するには「_SetProfileVal()」を、記録した数値を読み出すには「_GetProfileVal()」を使用します。

また、記録するセクション名を指定して、記録する情報をセクションで整理することもできます。

記録するセクション名は、「_SetProfileSect()」で指定します。

※ プロファイルの実体は、レジストリです。

レジストリのパスは「HKEY_CURRENT_USER¥Software¥AjrCtl~~XX~~¥AjrMacro~~XX~~<マクロテキスト名>¥<セクション名>」となります。

(「~~XX~~」は、AjrMacro32.exe の場合「32」、AjrMacro64.exe の場合「64」となります)

次のマクロは、プロファイルへの記録により、シリアルポートのポート番号とオープン状態を永続的に維持します。

```
1 : define port = 0;
2 : define fOpened = _FALSE;
3 :
4 : function main()
5 : {
6 :     // ボタン表示
7 :     _BtnOpen(0, "ポート設定"); _BtnEnable(0, _TRUE );
8 :     _BtnOpen(1, "ポートクローズ"); _BtnEnable(1, _FALSE);
9 :     _BtnOpen(2, "終了"); _BtnEnable(2, _TRUE );
10 :
11 :     _Cls();
12 :
13 :     // ポート番号, オープン状態読み出し
14 :     port = _GetProfileVal("PORT", port );
15 :     fOpened = _GetProfileVal("fOpened", _FALSE);
16 :
17 :     // オープン状態の永続化
18 :     if (fOpened) {
19 :         if (_ComIsOpenPossible(port)) {
20 :             _ComOpen(port);
21 :             _BtnEnable(1, fOpened = _TRUE);
22 :             _Printf("ポート %d をオープンしました。¥n", port);
23 :         }
24 :     }
25 :
26 :     _Stop();
27 :
28 :     // ポート番号, オープン状態記録
29 :     _SetProfileVal("PORT", port );
30 :     _SetProfileVal("fOpened", fOpened);
31 : }
32 :
```

つづき

```

33 : // ボタン・イベント処理
34 : function _OnButton(BtnNo)
35 : {
36 :     if (BtnNo == 0) { // ● 「ポート設定」 ボタン押下
37 :         if (port = _ComCreateByDialog(port)) {
38 :             if (!_ComIsOpened(port) && _ComIsOpenPossible(port)) {
39 :                 _ComOpen(port);
40 :                 _BtnEnable(1, fOpened = _TRUE);
41 :             }
42 :         }
43 :     }
44 :     else if (BtnNo == 1) { // ● 「ポートクローズ」 ボタン押下
45 :         _ComClose(port);
46 :         _BtnEnable(1, fOpened = _FALSE);
47 :     }
48 :     else if (BtnNo == 2) _Restart(); // ● 「終了」 ボタン押下
49 : }

```

シリアルポート送受信マクロの最終版

次のサンプルマクロは、ポート番号とオープン状態を永続化し、テキストとバイナリパケットの送受信を行います。

```

1 : // M530 : ポート状態の永続化と、テキスト/バイナリの送受信
2 :
3 : define bno_tx = 0;
4 : define bno_rx = 0;
5 : define port = 0;
6 : define fOpened = _FALSE;
7 : define $InpTxt = "";
8 :
9 : function main()
10 : {
11 :     // メモリブロック確保
12 :     bno_tx = _MemAlloc(1024);
13 :     bno_rx = _MemAlloc(1024);
14 :
15 :     // ボタン表示
16 :     _BtnOpen(0, "ポート設定"); _BtnEnable(0, _TRUE);
17 :     _BtnOpen(1, "ポートクローズ"); _BtnEnable(1, _FALSE);
18 :     _BtnOpen(2, "テキスト送信"); _BtnEnable(2, _FALSE);
19 :     _BtnOpen(3, "バイナリ送信"); _BtnEnable(3, _FALSE);
20 :     _BtnOpen(4, "終了"); _BtnEnable(4, _TRUE);
21 :
22 :     _Cls();
23 :
24 :     // ポート番号, オープン状態読み出し
25 :     port = _GetProfileVal("PORT", 0);
26 :     fOpened = _GetProfileVal("fOpened", _FALSE);
27 :
28 :     // シリアルポートの永続化
29 :     if (port != 0) {
30 :         _ComCreate(port);
31 :         if (fOpened) OpenPort();
32 :     }
33 :
34 :     _Stop();
35 :
36 :     // ポート番号, オープン状態記録
37 :     _SetProfileVal("PORT", port);
38 :     _SetProfileVal("fOpened", fOpened);
39 :
40 :     /* リソース破棄 */
41 :     _BtnClose(0); _BtnClose(1); _BtnClose(2); _BtnClose(3);
42 :     if (port != 0) { _ComClose(port); _ComDelete(port); }
43 :     _MemFree(bno_tx);
44 :     _MemFree(bno_rx);
45 : }
46 :

```

つづく

```

47 : // ★ボタン・イベント処理
48 : function _OnButton(BtnNo)
49 : {
50 :     define bytes;
51 :
52 :     if (BtnNo == 0) { // ●「ポート設定」ボタン押下
53 :         if (port = _ComCreateByDialog(port)) {
54 :             _ComSetOnRxText (port, "OnRxText");
55 :             _ComSetOnRxPacket(port, "OnRxPacket", bno_rx);
56 :             OpenPort();
57 :         }
58 :     }
59 :     else if (BtnNo == 1) { // ●「ポートクローズ」ボタン押下
60 :         _ComClose(port);
61 :         fOpened = _FALSE;
62 :         _BtnEnable(1, _FALSE); _BtnEnable(2, _FALSE); _BtnEnable(3, _FALSE);
63 :     }
64 :     else if (BtnNo == 2) { // ●「テキスト送信」ボタン押下
65 :         // テキストの入力と送信
66 :         if ($InpTxt = $_Input("送信するテキストを入力してください", $InpTxt)) {
67 :             _ComSendText(port, $_StrCat($InpTxt, "\n"));
68 :         }
69 :     }
70 :     else if (BtnNo == 3) { // ●「バイナリ送信」ボタン押下
71 :         // バイナリデータの入力と、透過型パケット送信
72 :         $InpTxt = $_Input("送信するバイナリデータ (16進2桁) を空白で区切って入力してください");
73 :         if ($InpTxt != "") { // 文字列入力あり?
74 :             if (bytes = _HexStrToBinary($InpTxt, bno_tx, 0)) { // 16進文字列→バイナリ変換
75 :                 _ComSendPacket(port, bno_tx, 0, bytes); // バイナリパケットデータ送信
76 :             }
77 :         }
78 :     }
79 :     else if (BtnNo == 4) _Restart(); // ●「終了」ボタン押下
80 : }
81 :
82 : // ★テキスト受信イベント処理
83 : function OnRxText($Text)
84 : {
85 :     _Printf("%s", $Text);
86 :     _Printf("\n");
87 : }
88 :
89 : // ★パケット受信イベント処理
90 : function OnRxPacket(len)
91 : {
92 :     _DumpMemBlk(bno_rx, 0, len);
93 : }
94 :
95 : // ★シリアルポートのオープン
96 : function OpenPort()
97 : {
98 :     if (!_ComIsOpened(port) && _ComIsOpenPossible(port)) {
99 :         _ComOpen(port);
100 :         fOpened = _TRUE;
101 :     }
102 :     if (fOpened) {_BtnEnable(1, _TRUE); _BtnEnable(2, _TRUE); _BtnEnable(3, _TRUE);}
103 :     else {_BtnEnable(1, _FALSE); _BtnEnable(2, _FALSE); _BtnEnable(3, _FALSE);}
104 : }

```


5.6. 右クリックイベントとポップアップメニュー

「あじやた〜む・マクロ」では、右クリックを検知したり、ポップアップメニューを表示できます。Shift/Ctrl キーを押しながら、右クリックすると、イベント処理関数「_OnRClick()」が実行されます。「_OnRClick()」には、以下の4つの引数が渡されます。

- x, y - 文字クリックした位置 (マクロウインド左上を原点としたピクセル位置)
- shift - Shift キー押下フラグ (_TRUE:押下)
- ctrl - Ctrl キー押下フラグ (_TRUE:押下)

※ Shift/Ctrl キーを押さなくても右クリックイベントを発生させるには、「_EnableStdRClk(_FALSE)」を実行しておきます。但し、「_EnableStdRClk(_FALSE)」を実行した場合は、標準の右クリックによるポップアップメニューは表示されなくなります。

以下のマクロは、Shift/Ctrl+右クリックするとポップアップメニューが表示され「テキスト送信」「終了」を選択できます。

「テキスト送信」を選択すると現在時刻の文字列を送信し、「終了」を選択するとマクロを終了します。

また、受信したテキストデータを表示します。

```

1 : // M550 : 右クリックとポップアップメニュー
2 : define port = 4; // COM4
3 :
4 : function main()
5 : {
6 :   _Cls();
7 :   _Printf("Shift/Ctrl+右クリックで、ポップアップメニューが表示されます。%n");
8 :   _Printf("ポップアップメニュー表示中は、受信テキストの表示が停止します。%n");
9 :
10 :   _ComCreate (port);
11 :   _ComSetOnRxText(port, "OnRxText");
12 :   _ComOpen (port, 115200, 8, 0, 1);
13 :
14 :   _Stop();
15 :
16 :   _ComDelete(port);
17 : }
18 :
19 : function _OnRClick(x, y, shift, ctrl)
20 : {
21 :   define id;
22 :   define $menu[] = { "テキスト送信",
23 :                     "終了"};
24 :
25 :   id = _PopupMenu($menu[], x, y);
26 :
27 :   if (id == 0) { // ●テキスト送信
28 :     _ComSendText(port, $_StrCat($_DateAndTime(), "%n"));
29 :   }
30 :   else if (id == 1) { // ●終了
31 :     _Restart();
32 :   }
33 : }
34 :
35 : function OnRxText($Text)
36 : {
37 :   _Printf("%s", $Text);
38 :   _Printf("%n");
39 : }

```

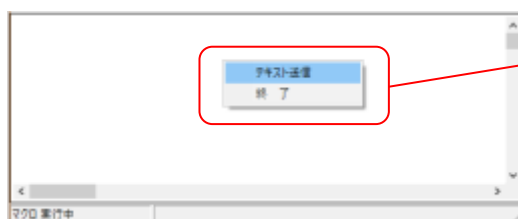
このサンプルマクロでは、イベント「_OnRClick()」でポップアップメニューを表示しています。

イベント処理中は他のイベント処理が実行されないので、「_PopupMenu()」の実行が終了するまで (つまり、ポップアップメニューが選択されるか、キャンセルされるまで) 他のイベントが実行されません。

この為、「OnRxText()」が実行されず、受信テキストの表示が止まってしまいます。(受信テキストの表示を止めない方法は次ページ)

ただ、「OnRxText()」イベントは全て保留されていますので、ポップアップメニューが終了すると、保留されていた「OnRxText()」イベントが一気に実行されます。

実行画面は、以下のようになります。



Shift/Ctrl+右クリックでポップアップメニューが表示される

受信テキストの表示を止めない場合は、_PopupMenu() の第 4 引数にメニューが選択された場合に実行するイベント関数を指定します。このイベント関数には、メニュー項目が選択された場合、引数として選択されたメニューのインデクス値 (0～) が渡されます。また、メニューがキャンセルされた (他の部分をクリックした) 場合、このイベント関数は実行されません。

_PopupMenu() の第 4 引数にイベント関数を指定した場合、_PopupMenu() の戻り値は意味を持ちません。(不定値が帰ります)

ポップアップメニューで、メニュー選択イベントを発生させるには、以下のようにします。

```

    . . . . .
    _PopupMenu($menu[], x, y, "OnMenu");
    . . . . .

function OnMenu(id)
{
    // id に選択したメニュー項目のインデクス (0～) が設定されています
}

```

以下のマクロは、右クリックするとポップアップメニューが表示され「テキスト送信」「終了」を選択できます。

「テキスト送信」を選択すると現在時刻の文字列を送信し、「終了」を選択するとマクロを終了します。

また、(ポップアップメニューの表示中でも) 受信したテキストデータを表示します。

```

1 : // M551 : 右クリックとポップアップメニュー (改良版)
2 : define port = 4;
3 :
4 : function main()
5 : {
6 :     _Cls();
7 :     _Printf("右クリックで、ポップアップメニューが表示されます。¥n");
8 :     _Printf("ポップアップメニュー表示中も、受信テキストが表示されます。¥n");
9 :
10 :    _EnableStdRClk(FALSE);
11 :
12 :    _ComCreate (port);
13 :    _ComSetOnRxText(port, "OnRxText");
14 :    _ComOpen (port, 115200, 8, 0, 1);
15 :
16 :    _Stop();
17 :
18 :    _ComDelete(port);
19 : }
20 :
21 : function _OnRClick(x, y, shift, ctrl)
22 : {
23 :     define $menu[] = { "テキスト送信",
24 :                       "終 了"};
25 :
26 :     _PopupMenu($menu[], x, y, "OnMenu");
27 :
28 : }
29 :
30 : function OnMenu(id)
31 : {
32 :     if (id == 0) { // ●テキスト送信
33 :         _ComSendText(port, $_StrCat($_DateAndTime(), "¥n"));
34 :     }
35 :     else if (id == 1) { // ●終了
36 :         _Restart();
37 :     }
38 : }
39 :
40 : function OnRxText($Text)
41 : {
42 :     _Printf("%s¥n", $Text);
43 : }

```

5.7. タイムチャートグラフ表示とタイマ

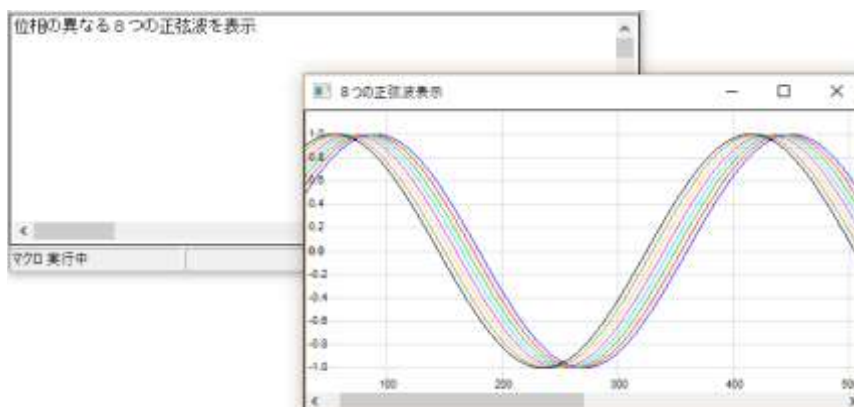
「あじやた〜む・マクロ」では、タイムチャート（時間経過とともに値が変化の様子）のグラフを表示することができます。以下のマクロは、位相が異なる8つの正弦波をグラフ表示します。データは、100ms 毎に更新します。

```

1 : // M560 : 位相の異なる8つの正弦波を表示
2 : option double;
3 :
4 : define t = 0;
5 :
6 : function main()
7 : {
8 :     define dat[8];
9 :
10 :     _Cls();
11 :     _Printf("位相の異なる8つの正弦波を表示\n");
12 :
13 :     _TmcOpen("8つの正弦波表示");
14 :     _TmcSetItems(8);
15 :     _TimerStart(1, 100);
16 :
17 :     _Stop();
18 :
19 :     _TmcClose();
20 :
21 : }
22 :
23 : // タイマ処理
24 : function _OnTimer(tid)
25 : {
26 :     define i, dt;
27 :     define dat[8];
28 :
29 :     if (tid == 1) { // タイマ1?
30 :         for (i = 0, dt = 0; i < 8; i++, dt += 5) {
31 :             dat[i] = _Sin(t + dt);
32 :         }
33 :         _TmcPutData(dat[]);
34 :         t += 1.0;
35 :     }
36 : }
37 :
38 : // タイムチャートウインドクローズ時の処理
39 : function _OnTmcClose()
40 : {
41 :     _Restart();
42 : }

```

実行画面は以下のようになります。

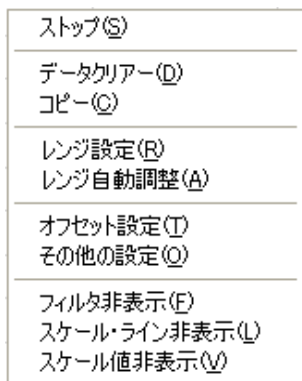


ここでは、単に正弦波を表示していますが、シリアルポートからデータを受信しそのデータを順次表示することにより、センサ等の出力波形を描いたりできます。

タイムチャートウインドでは、次に示す操作ができます。

ポップアップメニュー

グラフ上で右クリックすると、以下のポップアップメニューが表示されます。



- | | |
|------------|--|
| ストップ | : グラフ表示を停止します。次回は「スタート」メニューとなります。 |
| データクリアー | : バッファリングされているデータを全て破棄します。 |
| コピー | : グラフ表示内容 (ビットマップ) をクリップボードへコピーします。 |
| フィルタ非表示 | : コントロール左上のフィルタ (チェックボックス) を非表示にします。次回は「フィルタ表示」メニューに変わります。 |
| スケールライン非表示 | : 目盛り線 (薄いグレーの線) を非表示にします。次回は「スケールライン表示」メニューに変わります。 |
| スケール値非表示 | : 目盛り数値を非表示にします。次回は「スケール値表示」メニューに変わります。 |

[レンジ設定]「レンジ自動調整」「オフセット設定」「その他の設定」に関しては、すぐ後で説明します。

レンジ設定

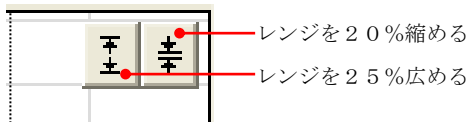
ポップアップメニューで「レンジ設定」を選択すると、以下のダイアログボックスが表示されます。



ここで、レンジ値を入力し、「OK」ボタンを押すと、グラフのレンジが設定されます。「Cancel」ボタンを押すと設定を中止します。

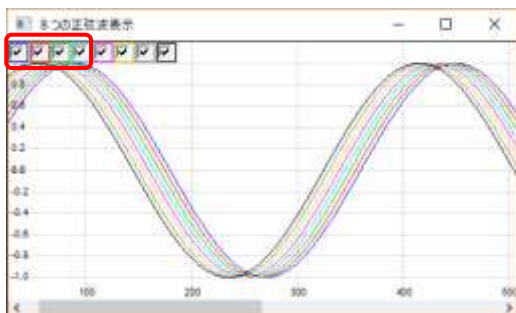
ワンタッチでレンジ設定

マウスマウスカーソルをタイムチャートグラフの右上隅に置くと、2つのボタンが表示されます。これらのボタンで、レンジを25%広めたり、20%縮めたりすることができます。

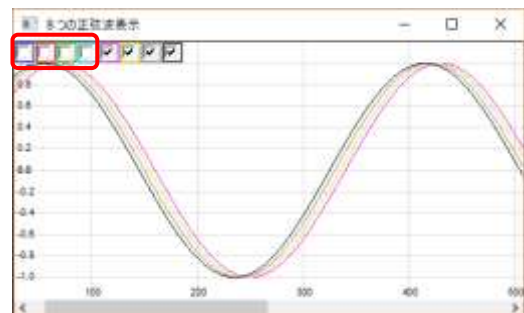


フィルタ機能

マウスマウスカーソルをタイムチャートグラフの左上隅に置くと、チェックボックスが表示されます。ウインド左上のチェックボックスは、表示フィルタであり、チェックを外すと当該表示色のデータは非表示となります。

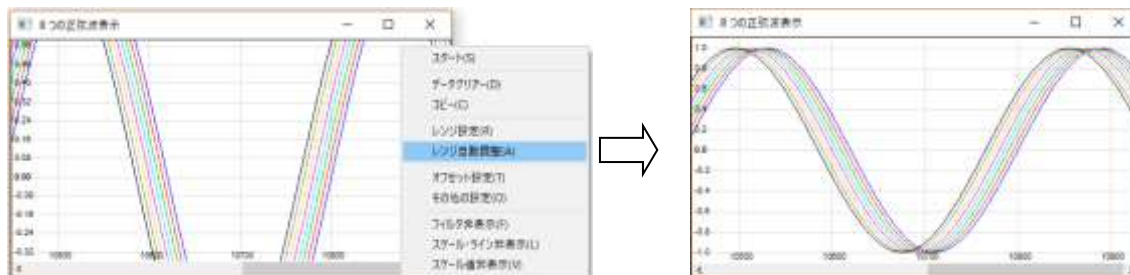


青緑



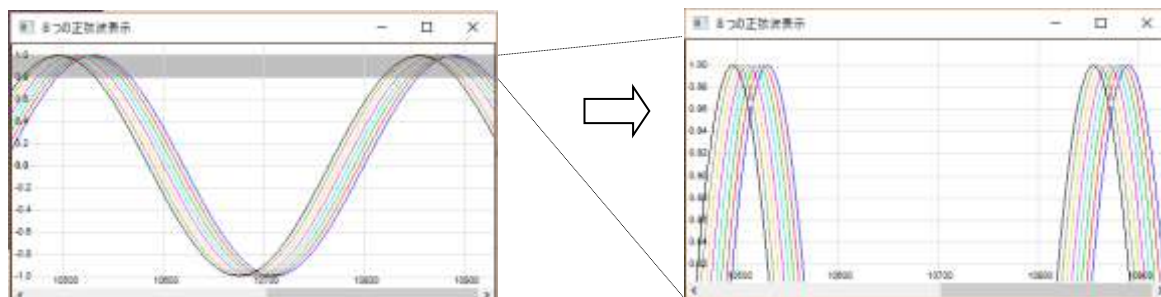
レンジ自動調整

ポップアップメニューで「レンジ自動調整」を選択すると、(フィルタで非表示となっているデータ項目を除く) 全てのデータから最小値と最大値を算出し、 $\pm 5\%$ のマージンを持ってレンジ設定を行います。



ドラッグ操作によるレンジ設定

CTRL キーを押しながら、マウス左ボタンで、レンジ設定したい部分をドラッグすることにより、レンジの設定を行うことができます。



レンジ設定する部分を、CTRL キーを押しながらマウスでドラッグ
(ドラッグされている部分はグレー表示されます)

CTRL キーを押したまま、マウス左ボタンを離すと、
ドラッグした部分がレンジ設定されます。

グラフの上端/下端を越えた部分までドラッグしても、当該ドラッグ範囲がレンジとして設定されます。

CTRL キーを先に離して、マウス左ボタンを離した場合は、レンジ設定は行われません。

オフセット設定

ポップアップメニューで「オフセット設定」を選択すると、以下のダイアログが表示されます。



各0～7の項目は、表示されているデータ項目に対応します。(外枠の表示色がグラフ表示色と同じになっています)

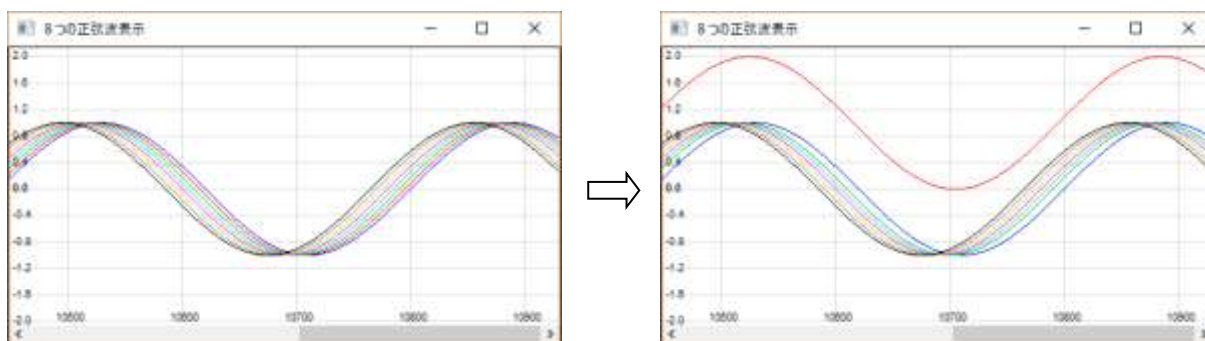
ここで、値を設定すると、当該データ項目のデータ値に、設定値を加算した値でグラフが表示されます。

値の設定に追従して設定したオフセット値がグラフに反映されます。

「OK」ボタンを押すと設定内容が確定されます。「Cancel」ボタンを押すと設定内容は破棄され、元のオフセット値に戻ります。

「リセット」ボタンを押すと、全てのオフセット値が「0」に設定されます。

以下の例は、赤色表示のデータに、オフセット値として「+1.0」を設定したものです。



5.8. イベントドリブン

ここまでは、イベントドリブンを使用した(_Stop()とRestart()を使用した)マクロで説明してきましたが、イベントドリブンを使用しない方法でもマクロを記述できます。

以下の2つのマクロは、1秒毎に現在時刻のテキスト送信処理を行います。

最初のマクロはイベントドリブンを使用した処理で、2つ目のマクロはイベントドリブンを使用しない処理となっています。

イベントドリブンを使用した場合

```

1 : // M570 : 1秒毎に現在時刻のテキスト送信 (イベントドリブン)
2 :
3 : define port = 4;
4 :
5 : function main()
6 : {
7 :     define dat[8];
8 :
9 :     _Cls();
10 :    _Printf("1秒毎に現在時刻のテキストを送信します。%n");
11 :    _BtnOpen(0, "終了");
12 :
13 :    _ComCreate (port);
14 :    _ComOpen (port, 115200, 8, 0, 1);
15 :    _TimerStart(1, 1000);
16 :    _Stop();
17 :    _ComDelete (port);
18 : }
19 :
20 : function _OnTimer(tid)
21 : {
22 :     _ComSendText(port, $_StrCat($_DateAndTime(), "%n"));
23 : }
24 :
25 : function _OnButton(BtnNo)
26 : {
27 :     _Restart();
28 : }

```

イベントドリブンを使用しない場合

```

1 : // M571 : 1秒毎に現在時刻のテキスト送信 (イベントドリブン未使用)
2 :
3 : define port = 4,
4 :     fEnd = _FALSE;
5 :
6 : function main()
7 : {
8 :     define dat[8];
9 :
10 :    _Cls();
11 :    _Printf("1秒毎に現在時刻のテキストを送信します。%n");
12 :    _BtnOpen(0, "終了");
13 :
14 :    _ComCreate (port);
15 :    _ComOpen (port, 115200, 8, 0, 1);
16 :    while (!fEnd) {
17 :        _ComSendText(port, $_StrCat($_DateAndTime(), "%n"));
18 :        _msWait(1000);
19 :    }
20 :    _ComDelete (port);
21 : }
22 :
23 : function _OnButton(BtnNo)
24 : {
25 :     fEnd = _TRUE;
26 : }

```

5.9. サンプルマクロテキスト

サンプル・マクロテキスト一覧を以下に示します。

#	マクロ・テキスト ファイル	内容
1	M010. txt	右クリックとポップアップメニュー, ユーザイベント処理
2	M020. txt	ロケート&プリント
3	M021. txt	CRC計算
4	M030. txt	スクリーン情報
5	M040. txt	配列操作
6	M050. txt	メモリブロック
7	M051. txt	メモリブロック (整数)
8	M052. txt	メモリブロック (実数)
9	M060. txt	ファイルのコピー&ベリファイ
10	M061. txt	テキストファイルを読み出し、指定した文字列を含む行を表示する
11	M062. txt	テキストファイルを読み出し、100ms/行の速度でCOMポートへ送信
12	M063. txt	指定フォルダ下のファイル検索
13	M064. txt	ファイルパスの分解, 組み立て
14	M070. txt	シリアル送受信とイベント
15	M071. txt	XMODEM送受信
16	M072. txt	YMODEM送受信
17	M073. txt	XMODEM送信
18	M074. txt	XMODEM受信
19	M075. txt	YMODEM送信 (フォルダ下の全ファイル)
20	M076. txt	YMODEM送信 (フォルダ下の全「.C」ファイル)
21	M077. txt	YMODEM送信 (単一ファイル)
22	M078. txt	YMODEM受信 (サブフォルダを作成しない)
23	M079. txt	YMODEM受信 (サブフォルダを作成する)
24	M080. txt	COMポート信号読み出し/設定
25	M090. txt	時間/ウェイト
26	M100. txt	算術演算
27	M101. txt	3点から円の中心を求める
28	M110. txt	ボタンとチェックボックスのサンプル
29	M120. txt	三角関数のチャート図を表示
30	M130. txt	文字列操作
31	M140. txt	フォルダ作成, パスチェック
32	M150. txt	プロファイル (数値) の読み出し/書き込み
33	M160. txt	チェックボックスの生成/消去
34	M200. txt	あじやた〜むとの接続
35	M500. txt	ボタンとチェックボックス, COM送信
36	M510. txt	シリアルポートを介して、テキストデータの送受信
37	M520. txt	シリアルポートを介して、バイナリ・パケット・データの送受信
38	M530. txt	ポート状態の永続化と、テキスト/バイナリの送受信
39	M550. txt	右クリックとポップアップメニュー
40	M551. txt	右クリックとポップアップメニュー (改良版)
41	M560. txt	位相の異なる8つの正弦波を表示
42	M570. txt	1秒毎に現在時刻のテキスト送信 (イベントドリブン)
43	M571. txt	1秒毎に現在時刻のテキスト送信 (イベントドリブン未使用)