

The terminal software for the Windows

AJATERM-MACRO

**MACRO FUNCTION
OPERATION MANUAL**

(Ver 1.6.3.0)

Attention

This manual is a Englishversion manual created by general Japanese to English translation software.
I think there are parts that are difficult to read, but please understand.

本書は日本語版説明書を、一般の 日本語→英語 翻訳ソフトで作成したものです。
読みづらい部分も多分にあると思いますが、ご了承ください。

— CONTENT —

1.	Overview.....	1
1.1.	Macro text file format.....	1
1.2.	Starting.....	1
2.	Language specification.....	3
2.1.	Variable name.....	3
2.2.	Type of numeric variable.....	3
2.3.	Definition of variables.....	3
2.4.	Scope of variable.....	4
2.5.	Defined constants.....	4
2.6.	Formula.....	5
2.7.	Numerical constant.....	5
2.8.	String.....	5
2.9.	Assign / Compare Strings.....	6
2.10.	Array assignment.....	6
2.11.	Processing block.....	7
2.12.	break / goto statement.....	7
2.13.	Function execution.....	8
2.14.	Format string.....	9
2.15.	Escape sequence.....	9
2.16.	Comment.....	9
2.17.	Event.....	10
2.18.	Differences from C language.....	11
3.	Connection with AJATERM.....	12
4.	Debug function.....	13
4.1.	Trace display.....	13
4.2.	Step trace.....	14
5.	Explanation of macro text.....	15
5.1.	Event driven.....	15
5.2.	Button / check box operation.....	15
5.3.	Serial port transmission / reception (text data).....	17
5.4.	Serial port transmission / reception (binary data).....	18
5.5.	Record to Profile.....	20
5.6.	Right click event and popup menu.....	23
5.7.	Time chart Graph display and timer.....	25
5.8.	Event driven.....	28

1. Overview

It is a macro / interpreter execution program with C language-like language specification.
You can input a program file (macro text file) and execute it as it is.

1.1. Macro text file format

The macro text file must be an ASCII file / shift JIS code file.

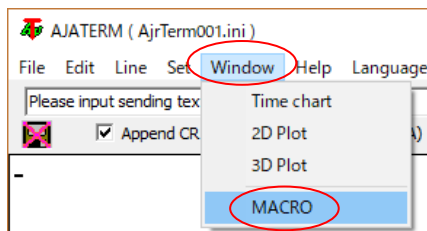
(Because other countries' multi-byte also follows the locale setting, I think that it will work normally, but the current situation is not tested)

1.2. Starting

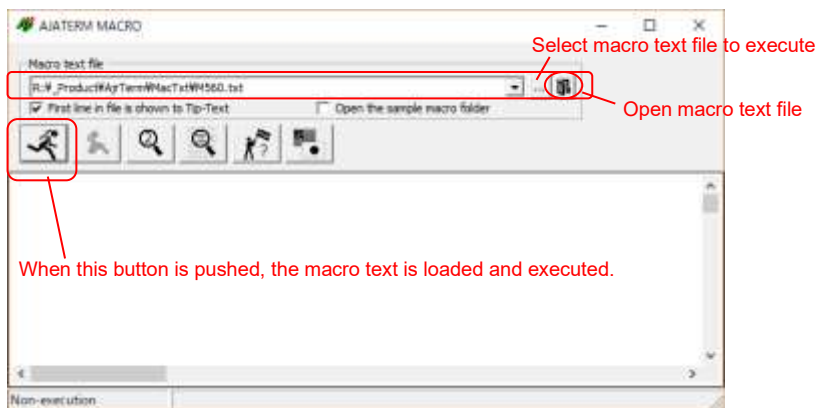
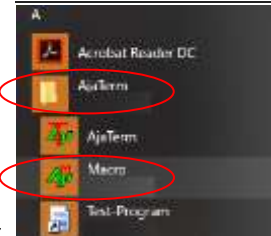
AJATERM - MACRO is started in the following way.

- Select "Macro" from the "Window" menu of AJATERM
- From the Windows start menu, select AJATERM → AJATERM-MACRO
- Execute AjrMacro32.exe / AjrMacro64.exe with Explorer etc.
- Drop the macro text file from the explorer etc. to the shortcut icon of AjrMacro32.exe / AjrMacro64.exe

Launch from the menu of AJATERM



Launch from the Windows start menu



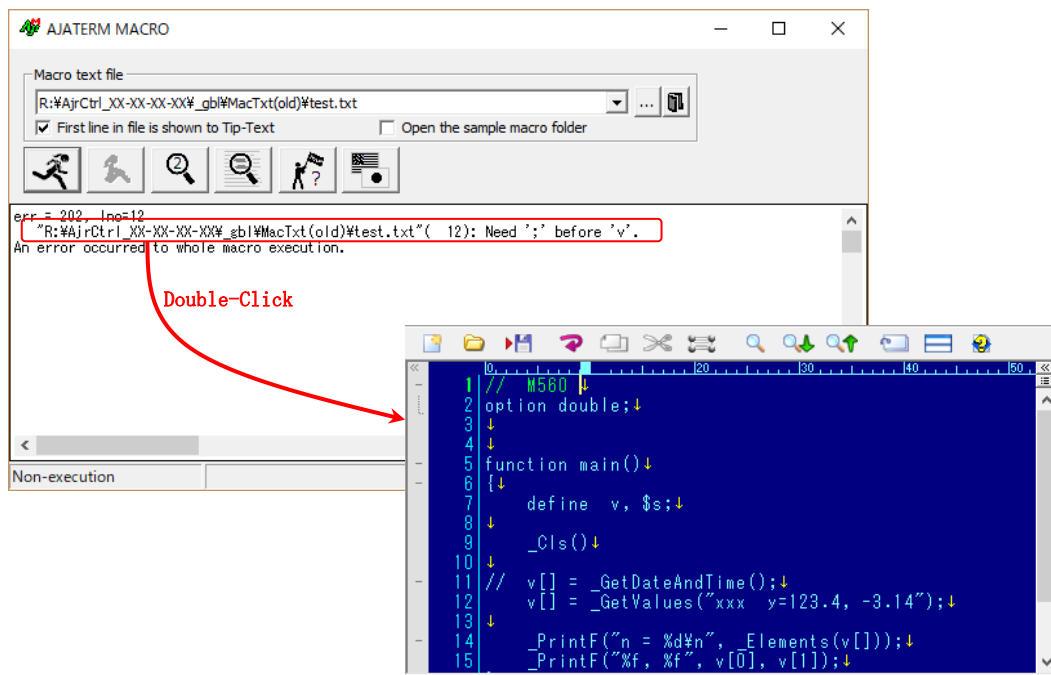
When "First line in file is shown to Tip-Text" is checked, the first line of the file is displayed as the chip text displayed when selecting the file in the combo box or placing the cursor in the combo box. (From the both ends of the text in the first line of the file, display text with tabs, carriage return, line feed, slash (/), and asterisk (*) removed)

When "Open the sample macro folder" is checked, when selecting a file with "..." button, first open the sample macro text folder.

The operation contents of each button are as follows.

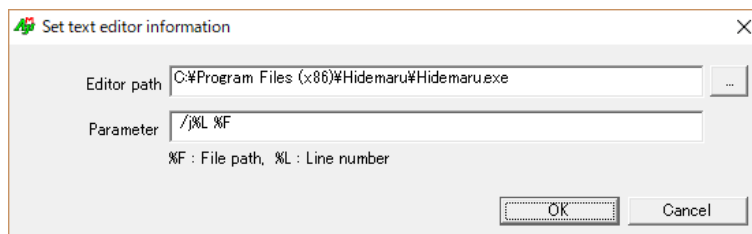
	Read the macro text file and execute it.		Start the step trace window
	Cancel execution of macro text.		Version display and description document display
	Activate the trace display window		Language settings

When reading and executing the macro text, if there is an error in the macro text, an error message will be displayed.
Double-click this error message line to open the macro text file with a text editor.



Setting up the text editor

To set up a text editor activated by double-clicking, double-click the part other than the error message line.
The following dialog will be displayed, so we will set up the text editor here.



In "Editor path", set the path of the text editor / program (. Exe file) to be started up.

For the parameter, specify the parameter given on the command line.

"% F" in the parameter is replaced with the path name of the macro text file and "% L" is replaced with the line number.

2. Language specification

This interpreter is C language-like language specification, but there are also differences from C language.

(For matters not described in the text, consider it the same as C language)

The following is the language specification of this interpreter.

2.1. Variable name

Variable names start with "Half width alphabet, double byte character, underscore (_) or dollar mark (\$)", followed by "Condition of first character" + "Half size numeric character".

Variables whose heads begin with "\$" are character variables, and others are numeric variables.

As with C language, alphabetic characters of variable names are case sensitive.

The following variable names are valid variable names.

```
(Ex.) define var = 10, VAR, _abc09, $str1;
      define $WhoAml = "It is a character variable";
      define Scores [3] = {82, 75, 96};
      define 東京都港区赤坂_10_2_102;
```

2.2. Type of numeric variable

A variable other than the dollar sign (\$) at the beginning is treated as a numeric variable, but its type is "signed integer" "unsigned integer" Or, it becomes "real number".

Which type you want to use is specified by the "option" statement at the beginning of the macro text.

```
(Ex.) option signed; ----- Treat numeric types as signed integers (default when no option is specified)
      option unsigned; ---- Treat numeric types as unsigned integers
      option double; ----- Treat numeric types as real numbers
```

The integer is the same as "signed long long" or "unsigned long long" in C language. (64 bit precision)

The real number is the same as "double" in C language. (64 bit precision)

2.3. Definition of variables

Variables are defined by the "define" statement with the following syntax.

#	syntax	Content	
1	define variable;	Simple variable (same as variable[1];)	Simple variable
2	define variable = value;	Simple variable with initial value (same as variable [1] = value;)	
3	define variable [n];	Array variable	Array variable
4	define variable [n] = value;	Array variable with initial value (Initialize only the first element)	
5	define variable [n] = {value-1, value-2, . . . , value-n};	Array variable with initial value (Initialize multiple elements)	
6	define variable [] = value;	Array variable with initial value (number of elements = 1)	
7	define variable [] = { value-1, value-2, . . . , value-n };	Array variable with initial value (number of elements = number of initial values)	

※ Multiple variables can be defined, separated by commas (.). (ex. define a, b=21, c[2]= {10, 20};)

※ If an initial value is not specified, the numeric variable is zero (0) and the character variable is initialized with the empty string ("").

※ "Number of elements(n)" and "Initial value(value/value-n)" can be specified by general mathematical formulas.

(ex. define var[a * 3 + b] = {a + b, a - b}; // a, b must be defined earlier)

※ A simple variable is actually an array with the number of elements = 1. ("Define x;" and "define x [1];" are the same)

※ It is the same as the variable name [0] unless you add a suffix ([n]) when referring to variables. ("Var = 1;" and "var [0] = 1;" are the same)

2.4. Scope of variable

Like the C language, variables defined outside the function have global scope. (It can be referred to from the whole source)

A variable defined in a function has a scope in the function like an automatic variable in C language. (It can be referred only within the function)

However, the automatic variable defined in the bracket ({...}) in the function is also the scope of the whole function.

If the global variable name and the automatic variable name in the function are the same, the automatic variable is accessed and the global variable can not be accessed.

```
(Ex.)  define gbl;           // Global variable (can be referenced from any function)
       define mult = 10;    // Global variable (can not be referenced if variable of the same name is defined in function)
       function func(a, b)
       {
           define v1, v2;    // Automatic variable
           define mult = 20; // Automatic variable
                               // (Because it has the same name as global variable, only this automatic variable can be referenced)

           if ( . . . ) {
               define inb = 1; // 'Inb' is also an automatic variable with the scope of the whole function
               . . .
           }

           v1 = inb;          // "Inb" is defined in the above "if {...}" Bracket, but since it has the entire scope within
           gbl = inb;         // the function, it can be referred to
           v2 = mult;         // The automatic variable "mult" is referenced, and v2 = 20.
       }
```

"define" in the function is "execution instruction to generate automatic variable".

If the same "define" is executed twice in the procedure, a double definition error will result.

In the following code, "define a;" is executed twice, resulting in a runtime error.

```
(Ex.)  for (i = 0; i < 2; i++) {
           define a;        // A double definition error occurs in the second execution
           . . .
       }
```

※ To avoid double definition errors, we recommend that you define automatic variables at the beginning of functions.

2.5. Defined constants

The defined constants list is shown below.

#	Name	Value	Note
1	_TRUE	1	True value.
2	_FALSE	0	False value.
3	_CR	0x0D	Return code
4	_LF	0x0A	Line feed code
5	_SLOT_PORT	256	Port number of the mail slot
6	_MB_OK	0	BoxType argument of _MsgBox ()
7	_MB_OKCANCEL	1	
8	_MB_RETRYCANCEL	5	
9	_MB_YESNO	4	
10	_MB_YESNOCANCEL	3	
11	_MB_ICONERROR	16	Return value of _MsgBox ()
12	_IDCANCEL	2	
13	_IDNO	7	
14	_IDOK	1	
15	_IDYES	6	The "knd" argument of _ CalcCrc ()
16	_CRC_ITL	0	
17	_CRC_ITR	1	
18	_CRC_16L	2	
19	_CRC_16R	3	Return
20	\$_CR	"¥r"	
21	\$_LF	"¥n"	
22	\$_CRLF	"¥r¥n"	
23	\$_MyPath	Path name part of macro text	Macro path name part ("D: ¥ work ¥ Mac.txt" → "D: ¥ work ¥")
24	\$_MyName	Macro text file name	Macro file name part ("D: ¥ work ¥ Mac.txt" → "Mac")

2.6. Formula

Expression of mathematical expressions is almost same as C language.

(However, there are some differences, such as character string assignment / comparison can be described etc.)

The types of operators and their priorities are as follows.

#	operator	Name	Note	Priority
1	(. . .)	優先計算部	「 . . . 」は数式	High ↑ ↓ Low
2	= ==, !=, <, >, <=, >=	Assigning a string String comparison	String assignment / comparison has high priority ex. \$ s [2] = "AB" \$ s <\$ t	
3	++ -- +, -, ! ~	Pre-increment, Post-Increment Pre-decrement, Post -decrement Unary operator (plus, minus, logical negation) Unary operator (bit inversion)	ex. ++x y++ ex. --x y-- ex, !x -10 In case of real number, it can not be used	
4	*, /, %	Multiplication, division, remainder calculation		
5	+, -	Addition , subtraction		
6	<<, >>	Left shift, right shift	For real numbers, multiply / divide by 2 ⁿ	
7	<, >, <=, >=	Comparison of magnitude	Comparison of numerical values	
8	==, !=	Equality, inequality comparison		
9	&	Bitwise AND	In case of real number, it can not be used	
10	^	Bitwise exclusive OR		
11		Bitwise OR		
12	&&	Logical AND		
13		Logical OR		
14	? :	Ternary operation	ex, a == 1 ? 0 : 1	
15	= +=, -= *=, /=, %= &=, =, ^= <<=, >>=	Substitution Addition / subtraction substitution Multiplication and division Bit logical product, logical sum, exclusive OR Left shift, right shift assignment	Assignment (with calculation) to numeric variable It can not be used in real number For real numbers, multiply / divide by 2 ⁿ	

※ The comma (,) operator can not be used. (However, multiple initialization and updating of "for (initialization; condition; update)" can be specified separated by a comma)

※ Those with the same "# (item number)" have the same priority

※ In the case of substitution (=) of the character string of "# 2", the value when substituting the empty string ("") is "0", otherwise the value is "1".

The comparison of character strings (==, !=, <, >, <=, >=) is "1" if the result is true (established), "0" if it is false (not established).

```
(Ex.)  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;
```

●About pre / increment / decrement, post increment / decrement

For preincrement / decrement or post - increment / decrement, write "++" or "--" immediately before or after the variable, and do not enclose the variable in parentheses.

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

2.7. Numerical constant

Representation of numerical constants is the same as in C language. (123, - 789, 3.14, 0.7 G3 ... etc.)

In addition, character strings (including character type variables and functions) can also be treated as numerical values. (Up to the first 8 bytes)

For example, "a =" ABC "" is the same as "a = 0x414243".

However, if it is a character expression, it will be the evaluation result of the character expression. (「A =" ABC "==" DEF 」 becomes 「a = 0」)

2.8. String

In a character string, enclose characters in double quotes (") or apostrophes (').

Escape characters beginning with "\" similar to the C language can be described as a character string enclosed in double quotes (").

The character string enclosed in apostrophe (') does not recognize the escape character, and the written content becomes the character string as it is.

```
ex.      "¥x1B[0mABC¥n" ----- "¥ X1B" becomes 1 byte "0x1B", "¥ n" becomes "0x0A".
        'c:¥work¥subdir¥sample.txt' - The description 'c: ¥ work ¥ subdir ¥ sample.txt' becomes a character string as it is.
```

2.9. Assign / Compare Strings

You can write a character string assignment or comparison in a mathematical expression.

The substitution of a character string is described with the following syntax.

$$\text{\$StringVariable} = \text{<"String" / \$StringVariable / \$StringFunction(>}$$

The substitution result of the character string is false (= 0) when substituting the empty string (""), and true (= 1) when assigning something other than the empty string.

(Ex.) $\text{\$s[1]} = \text{"ABC"}$
 $\text{\$t} = \text{\$func(1, 2)}$
 $\text{\$u} = \text{\$s[1]}$

Compare character strings with the following syntax.

$$\text{<"String" / \$StringVariable / \$StringFunction(> <Comparison operator> <"String" / \$StringVariable / \$StringFunction(>}$$

Comparison operator : "==" , "!=" , "<" , ">" , "<=" or ">="

Strings are case-sensitive alphabetic characters. (We judge that "ABC" and "abc" are different character strings)

The comparison result of the character string is true (= 1) when the comparison is established, and false (= 0) when the comparison is not established.

(Ex.) $\text{\$s[1]} == \text{"ABC"}$
 $\text{\$t} >= \text{\$fun(1, 2)}$
 $\text{\$u} < \text{\$s[1]}$

String assignment / comparison is calculated with high priority in the formula. (It is equivalent to enclosing the character string assignment / comparison in parentheses). For example, "if (1 - \$ s == "ABC ") " is the same as "if (1 - (\$ s == "ABC "))".

2.10. Array assignment

With the assignment operator '=', assignment of the entire array is possible.

Assignment of an array is described with the following syntax.

$\text{Var 1 []} = \text{Var 2 []}$; --- Copy the entire contents of Var 2 to Var 1. The number of elements of Var1 is the same as Var2.
 $\text{Var 1 []} = \text{Func ()}$; --- The function returns an array with a return statement (ex. Return var []);

Array assignment can not be described in normal mathematical formulas. Please describe it by itself with the above syntax.

(Ex.) $\text{a[]} = \text{b[]}$; ----- Assign numeric type array (copy entire array)
 $\text{a[]} = \text{func(1, 2)}$; ----- func returns an array with a return statement (ex. return n []); · · · ※1
 $\text{\$s[]} = \text{\$t[]}$; ----- Substitution of string array (copy entire array)
 $\text{\$s[]} = \text{\$func(1, 2)}$; --- \$ func returns an array with a return statement (ex. return \$ x []); · · · ※1

※1 : An example of a function that returns the whole array is shown below. (Examples of numerical type array in the following example)

If an array (written as "variable []" without subscript) is specified as the return value of the function, the entire array is copied to the variable.

(Ex.)

```
function func()
{
    define arr[5] = {10, 20, 30, 40, 50};
    return arr[];
}
. . . . .
var[] = func();    // The contents of the array "arr [5]" are copied to the variable "var".
                  // and var [0] = 10, var [2] = 20, ... var [4] = 50
. . . . .
```

However, if a variable that receives a return value of a function is described as an ordinary variable, the value at the head of the array (the above arr[0]) is set to that variable.

In the above example, if "var[] = func();" is set to "var[2] = func();", the content (= 10) of arr[0] is set to var[2] I will.

Conversely, if "var[] = func();", func() does not return an array (returns a single value) the return value is set to var[0].

A description such as "var[2] = func()" in which a variable that receives a return value of a function is described by an ordinary variable can also be described in a normal mathematical expression.

2.11. Processing block.

You can describe a processing block with "do {...}" (last of while () none).

This processing block is a block which can be omitted simply by "break" without repeating processing.

The following two syntaxes have the same behavior.

do block (no later while)

```
do {
    if (a == 10) break; // →Jump to ①
    . . . . .
}
①→
```

do block (after setting while)

```
do {
    if (a == 10) break; // →Jump to ①
    . . . . .
} while (FALSE);
①→
```

※ For do block, brackets ("{" and "}") are mandatory.

Other processing block (It is almost the same as C, but you can not use switch-case statements)

while block (prefix while, with bracket)

```
while (Conditional expression) {
    [expression;]
    . . . . .
}
```

while block (prefix while, no bracket)

```
while (Conditional expression) [expression;]
. . . . .
```

for block (with bracket)

```
for ([Initialization expression 1][, Initialization expression 2] . . . ;
    [Conditional expression];
    [Update formula 1][, Update formula 2] . . . ) {
    [expression;]
    . . . . .
}
```

※For for-blocks only, you can write multiple "initialization expressions" and "update expressions" separated by comma operators (,).

for block (no bracket)

```
for ([Initialization expression 1][, Initialization expression 2] . . . ;
    [Conditional expression];
    [Update formula 1][, Update formula 2] . . . ) [expression;]
. . . . .
```

if / else block (with bracket)

```
if (Conditional expression) {
    [expression;]
    . . . . .
}

[ else if (Conditional expression) {
    [expression;]
    . . . . .
}

    ○
    ○
    ○

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

if / else block (no bracket)

```
if (Conditional expression) [expression] ;
[ else if (Conditional expression) [expression] ; ]
    ○
    ○
    ○

[ else [expression] ; ]
```

※ As with C language, it is OK even if "with bracket" and "without bracket" are mixed.

Of course, each processing block can be nested.

2.12. break / goto statement

You can use break and goto statements like C language.

ex.

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

2.13. Function execution

Unlike the C language, it is not necessary to define a prototype even if the function is described behind. (Prototype can not be described)

When calling a function defined in a macro text, the number of arguments and the argument type (numeric / character type) must match exactly.

```
(例)      . . .
          add(1, 2); ----- OK   (The number and type of the argument completely match.)
          add(1, 2, 3); ----- Error (The number of arguments does not match)
          add(1, "XYZ" ); ----- Error (The type of the second argument does not match)
          . . .
          function add(a, b)
          {
              return a + b;
          }
```

For an array that does not specify an index, such as 'var []' as an argument, pass the entire array.

In this case, even for functions that accept arguments, you must make the argument an array that does not specify an index like 'arg []'.

```
(例)      define var[3] = {10, 20, 30};
          sub(var[]);
          . . .
          function sub(arg[]) {
              // arg [0] = 10, arg [1] = 20, arg [2] = 30 are set.
          }
```

Arguments to be passed to the function are data copied as dummy arguments (including the case of passing the whole array).

Therefore, even if you change the value of the argument in the function, the variable value of the caller is not changed.

```
(例)      define var = 30;
          sub(var);
          . . .
          function sub(arg) {
              arg = 999; // The value of the caller's argument (var) is not changed (it is still 30)
          }
```

<Note> When specifying an expression that includes a character expression as a function argument, enclose the whole formula in parentheses, or make the beginning of the formula a numeric item.

- func(\$a == "ABC" + 1); --- Description mistake
- func((\$a == "ABC" + 1)); --- OK ("func (2)" when \$a is "ABC" and "func (1)" when \$a is other than "ABC")
- func(1 + \$a == "ABC"); --- OK ("func (2)" when \$a is "ABC" and "func (1)" when \$a is other than "ABC")

2.14. Format string

"_Printf ()" "\$ _ SPrintf ()" "_ Trace ()" generates a character string by format string.

This format is almost the same as C language (MS-VC), but the following points differ slightly.

- 1) In real mode, if the integer format (the end of the form is "d, i, u, x, or X"), convert the numeric value of the corresponding argument to "long long type". (※ 1)
- 2) In integer mode, if the real number format (the format ends with "f, e, E, g, or G"), convert the numeric value of the corresponding argument to "double type". (※ 1)
- 3) It is not necessary to specify the size specifier "l" "ll" "l64". (In the case of integer format, it is added automatically)
- 4) It does not correspond to the format conscious of the wide character.
The capital letter "S / C" is handled in the same way as lowercase "s / c".
The size specifier "l or h" can not be specified for "S, C, s, c".
- 5) The character string to be generated must be within 1023 characters (within 1023 bytes) in half size conversion.

※ 1 : Regardless of the type of numeric value, specify "% d", "% x" or "% u" when handling integer values and specify "% f" or "% g" when handling real numbers.

In addition, formats such as "% 5d", "% - 5d", "% 04x", "% * d", "% 7.2f" and "% .3f" which specify the number of digits are valid.

2.15. Escape sequence

The following escape sequence can be included in the screen display text.

#	ESC Sequence	内 容
1	ESC · [0J	Clear from the cursor position to the right end of the last line
2	ESC · [1J	Clear from the left end of the first line to the cursor position
3	ESC · [2J ESC · *	Clear the screen and move the cursor to the home
4	ESC · [OK ESC · [K	Cursor position ~ Clear until the right end of the line
5	ESC · [1K	Clear from the left end of the cursor line to the cursor position
6	ESC · [2K	Clear cursor line
7	ESC · [s	Save cursor position
8	ESC · [u	Recover cursor position
9	ESC · [>5l	Cursor display
10	ESC · [>5h	Hide cursor
11	ESC · [p1;pcH	Cursor position setting (pl is row position (1 to), pc is digit position (1 to))
12	ESC · [pnA	Moves the cursor upward (pn is the number of lines to be moved, assuming 1 when omitted)
13	ESC · [pnB	Move the cursor downward (pn is same as above)
14	ESC · [pnC	Move the cursor to the right (pn is same as above)
15	ESC · [pnD	Move the cursor to the left (pn is same as above)
16	ESC · [pnM	Scroll up after the cursor line (pn is the number of scroll lines)
17	ESC · [pnL	Scroll down after the cursor line (pn is same as above)
18	ESC · [psm	Drawing attribute setting (Ps = attribute value, 0 = default attribute, 7 = inversion, 30-37 (character color) = black, red, green, yellow, blue, purple, light blue, white, 40 to 47 = background color)
19	ESC · D	Move cursor down one line
20	ESC · E	Move the cursor one line down to the left edge
21	ESC · M	Move cursor up one line

[Note] "ESC ·" means escape code (0x1B). (Ex. _ Print F ("% x 1 b [31 m ABC"); ◇ "ABC" is displayed in red)

2.16. Comment

As with C, comments are sandwiched between "/*" and "*/", or from "/*" to the end of the line.

In sandwiching with "/*" and "*/", it is okay to cross over several lines.

Comments can not be nested.

2.17. Event

The following events can be handled during macro text execution.

Neither event occurs in multiple.

Standard event processing function

#	Event Function	Event	Enable/ Disable	Multi. occ.	Note
1	<code>_OnKeyIn</code>	Key input	<code>_DisStdEvt</code> <code>_EnaStdEvt</code>	Hold all	
2	<code>_OnRClick</code>	right click	Same as above	Hold once	If you do not press Shift or Ctrl key <code>_EnableStdRClick (_FALSE)</code> Execution required
3	<code>_OnFileSearch</code>	File search folder notification	Same as above	Discard	
4	<code>_OnTmcClose</code>	Time chart graph Close	Same as above	Hold all	
5	<code>_OnButton</code>	Press button	Same as above	Hold all	
6	<code>_OnChkBox</code>	Checkbox setting	Same as above	Hold all	
7	<code>_OnTimer</code>	Timer period notification	Same as above	Hold once	However, when the timer is stopped, the remaining event is discarded
8	<code>_OnMenu</code>	Select pop-up menu	Same as above	Hold all	The actual function name is specified by " <code>_PopupMenu ()</code> "

COM port communication event processing function

#	Event Function	Event	Enable/ Disable	Multi. occ.	Note
1	<code>_OnRxText</code>	Text reception notification	<code>_DisComEvt</code> <code>_EnaComEvt</code>	Hold all	
2	<code>_OnRxCtrl</code>	Control code reception notification	Same as above	Hold all	
3	<code>_OnRxEsc</code>	ESC sequence reception notification	Same as above	Hold all	
4	<code>_OnRxPacket</code>	Packet reception notification	Same as above	Hold all	

AJATERM connection event processing function

#	Event Function	Event	Enable/ Disable	Multi. occ.	Note
1	<code>_OnTermLink</code>	AJATERM connection	<code>_DisStdEvt</code> <code>_EnaStdEvt</code>	Hold all	Occurs when macro window is started from AJAXTERM
2	<code>_OnTermUnlink</code>	AJATERM disconnection	Same as above	Hold all	Notify that AJATERM has ended
3	<code>_OnTermOpened</code>	The port on AJATERM was opened	Same as above	Hold all	
4	<code>_OnTermClosed</code>	The port on AJATERM was closed	Same as above	Hold all	

※The main code, etc., which is executed by the extended line from the beginning of the program such as `main ()` is called main code.

On the other hand, the code executed in the above event is called an event code, and the event code is executed asynchronously.

The event is executed by interrupting each step of the main code.

A step is a conditional expression such as `if` or an expression of an execution code as a unit, but if there is a function call in the middle, it will be up to just before the function call. Also, the end of the control structure such as `while` / `for` (`';` or `'}`) is also one execution step.

(Activate the step trace window, you can see the state of the step execution)

2.18. Differences from C language

The approximate differences from C language are summarized below.

- 1) You can not use preprocessing statements (#include, #if, #define etc) beginning with "#".
- 2) You can not use switch-case statements.
- 3) Pointers can not be used.
- 4) Variables are defined with a "define" statement.
- 5) Variable types are numeric and character types, and numeric types are unified as "signed integer", "unsigned integer" or "real number".
The evaluation of mathematical expressions is done in a unified type ("signed integer", "unsigned integer" or "real number").
- 6) Functions are defined by "function" statements, and arguments are listed only in variable names.
- 7) It is not necessary to define a function prototype.
- 8) Structs (structs) and unions (unions) can not be used.
- 9) The program (macro text) must be a single text file.
- 10) Only one-dimensional arrays can be used for arrays.
- 11) General equations, including variables, can be used for the number of elements in the array.
- 12) All automatic variables defined in the function (even if nested with the bracket "{..}") will be the entire scope of the function.
- 13) You can omit "while" in "do-while".
In this case "do {...}" is simply a processing block to break out.
- 14) For functions defined in macro text, the variable argument "..." can not be used.
- 15) You can use character expressions (character string assignment and comparison).

3. Connection with AJATERM

You can connect to AJATERM and send and receive data via AJATERM's connected serial line.

To connect with AJATERM, follow the steps below.

- 1) Launch the macro window from the "Window" → "Macro" menu of AJATERM.
- 2) Create the AJATERM connection notification event function "_OnTermLink ()" in the macro text and proceed as follows.

```
define bno;

function _OnTermLink($MySlot, $RmtSlot)
{
    // Memory block allocation (for packet data reception)
    bno = _MemAlloc(1024);
    // Mail slot / port generation for connection with terminal
    _ComCreate($MySlot);
    // Receive event function setting
    _ComSetOnRxText (_SLOT_PORT, "OnText");
    _ComSetOnRxCtrl (_SLOT_PORT, "OnCtrl");
    _ComSetOnRxEsc (_SLOT_PORT, "OnEsc");
    _ComSetOnRxPacket(_SLOT_PORT, "OnPacket", bno);
    // Mail slot / port open for connection with terminal
    _ComOpen($RmtSlot);
    . . . . .
}
```

_ComSetOnRxXXXX() executes only for necessary events.
For example, if it is only necessary to receive text data, only _ComSetOnRxText () is executed.

If you connect normally, "Connecting with AjrTermXXX.ini" is displayed at the bottom of the window.
("AjrTermXXX.ini" is the content displayed in the AJATERM window title)

When data is received on the serial line on AJATERM, "OnText ()", "OnCtrl ()", "OnEsc ()" and "OnPacket ()" are executed and the received "text data", "control code" "ESC sequence" "packet data" is notified.

These functions are created in the macro text.

To send data to the serial line on "Aja-tamu", execute the _ComSendText (), _ComSendBinary (), and _ComSendPacket () functions.
(Specify "_SLOT_PORT" as the port argument)

Please refer to the sample text (MacTxt ¥ M200.txt) for an actual example of connecting to AJATERM and sending and receiving on a serial line on AJATERM.

「When connecting to AJATERM, set "Received text encode" and "Sending text encode" to "Default multi-byte" in the AJATERM set-menu. .. (Only transmission / reception using Default multi-byte code is possible)

4. Debug function

The following internal functions are available as a debugging function.

Internal function

Name	Function	Note
_VarDump	Display all variable information in the trace display window	
_Trace	Display format string in the trace display window.	
_Pause	Pause all execution	
_StepTrace	Enable / disable display of step trace.	You can limit the range to step trace

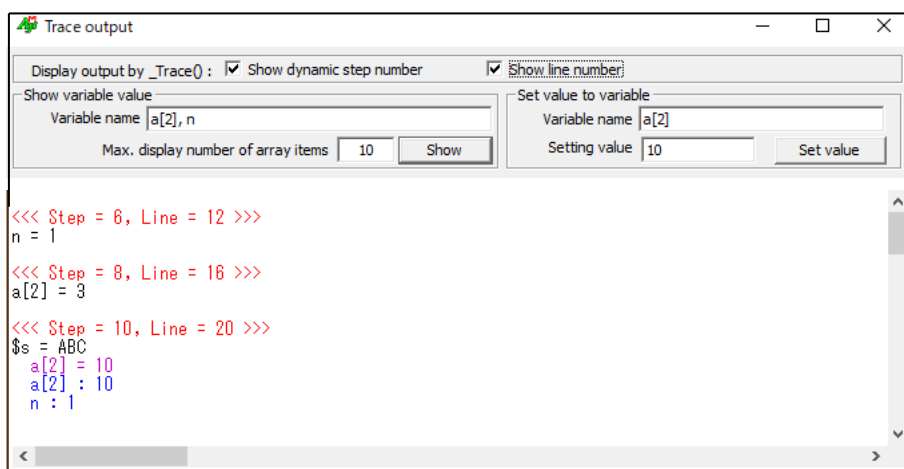
4.1. Trace display

When you press the  button, the trace display window is displayed.

The trace display window is also displayed automatically when the internal function "_Trace ()" is executed.

The trace display window displays the output contents of the "_Trace()" internal function.

The syntax of the "_Trace()" internal function is the same as that of C printf (), and you can display internal information of the program.



Two check boxes "Display dynamic step number" and "Display line number" add dynamic step number and line number information to display output by _Trace (). ("<<< Step = 6, Line = 12 >>>" etc. in the above display example)

The dynamic step number indicates the number of dynamic execution steps from the start of execution, with the step at the start of execution set to 1.

The line number indicates the line number of the macro text file.

In addition, variables can be displayed and set with execution of macro text stopped by internal function "_Pause ()".

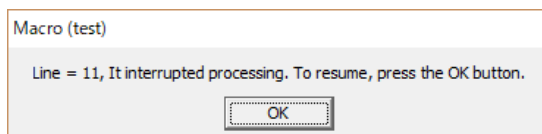
(In addition, it is possible to set a breakpoint from the step trace window or stop by a pause button etc.)

When you enter the variable you want to display in "Variable name" in "Show variable value" and press "Show" button, the value of the variable is displayed in the trace display window. To display multiple variables, enter a variable name separated by a comma (,).

"Max. display number of array items" specifies the maximum number of array elements to display when the variable is an array.

Enter the value to be set to "Setting value" for the variable name you want to set to "Variable name" in "Set value to variable", and press the "Set value" button to set the value to the variable. In "Variable name", enter only one variable name. In addition, when specifying an array, enter the variable name so as to identify the elements of the array. (ex. arr[3])

In internal function "_Pause ()", the following message will be displayed, so press "OK" button to resume execution.



※ If the step trace window is open, the above message will not be displayed.

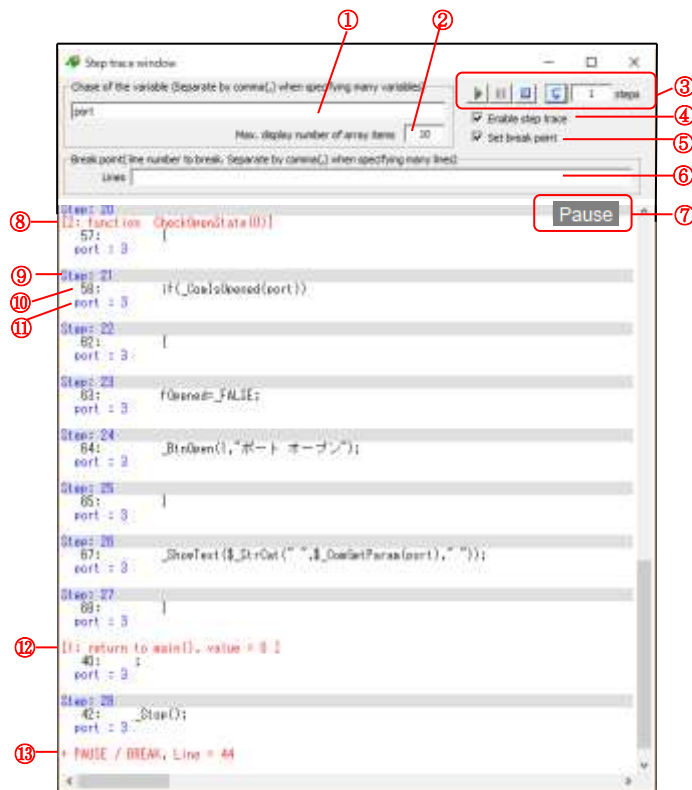
In this case, resume execution by pressing the Continue execution button in the step trace window.

4.2. Step trace

When you press the  button, the step trace window is displayed.

However, the step trace window can not be opened while executing the macro text, and the window can not be closed while the macro is running. (That is, it must be open before running the macro text)

The appearance of the step trace window is as follows.



#	Description
1	Specify variables to be displayed for each execution step
2	If the variable specified in ① is an array, the maximum number of term items to display
3	Macro text execution operation button and number of steps  : Execution continuation (Restart execution of macro text)  : pause  : Stop (Cancel execution of macro text)  : Step execution (After pausing the specified number of steps, paused state again) 1 steps : Number of steps to execute
4	Allow (check) / prohibit (not checked) step trace If step tracing is prohibited, nothing will be displayed. The same operation is possible with the <code>_StepTrace ()</code> internal function.
5	It breaks with the line number entered in (6) Enable (check) / invalidate (not checked) breakpoint.
6	Input of the line number to be broken (It is not displayed when ⑤ is not checked)
7	Displaying the execution status (Execution / Pause/ Stop (macro text is not running))
8	A display indicating the start of function execution (in this example, it indicates that execution of <code>CheckOpenState ()</code> started) The leading number "n:" means the nesting level of the function.
9	Dynamic step number
10	Line number of macro text and display of execution step instruction
11	Display of the variable value specified in ①
12	A display indicating the return from the function (in this example, return value = 0, indicating returning to the main () function) The leading number "n:" means the nesting level of the function.
13	Indicates that it is temporarily stopped. It is paused under the following conditions. •Pause () was executed •The line number (breakpoint) specified in ⑥ has been reached •Pause button () pressed •Step execution by step execution button () expiration

5. Explanation of macro text

In AJATERM - MACRO, various functions can be realized by calling internal functions (a set of functions such as C printf () and strcpy () etc).

Refer to "AjmQuickE.pdf" for the internal function · the whole description.

"AjmQuickE.pdf" is displayed by selecting "Quick Reference" from the pop-up menu displayed with the  button.

(However, the program (Acrobat Reader etc.) associated with the .pdf file is necessary)

Here, we show an example of macro text using some internal functions and explain the macro.

5.1. Event driven

Programming with AJATERM - MACRO is based on event driven processing.

The macro starts execution from the main () function without arguments.

Normally, the main () function performs initial setting and executes "_ Stop ()".

"_ Stop ()" stops execution of the main code and waits for the occurrence of the event.

When an event occurs, the specific function associated with the event is executed, and the event is waiting for the occurrence of the event again.

When executing "_ Restart ()" in any event, the event wait state is canceled and control is transferred to after "_ Stop ()".

Normally, postprocessing is described after "_ Stop ()".

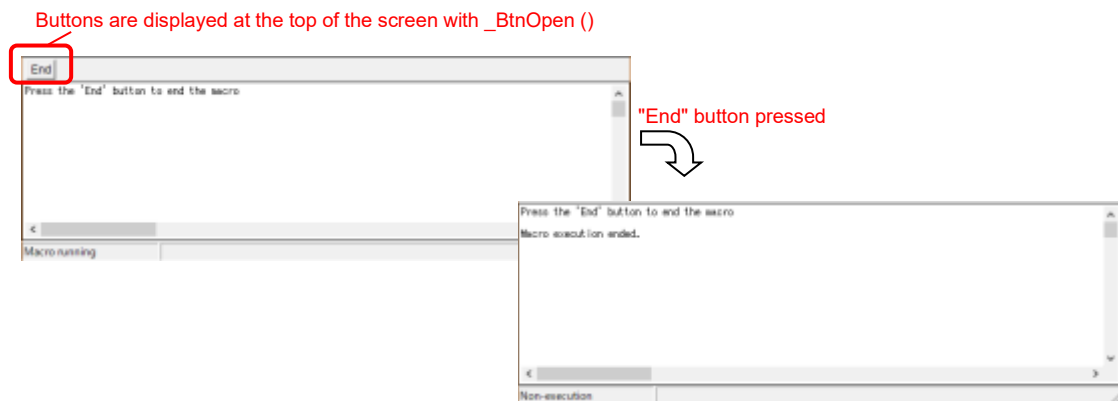
5.2. Button / check box operation

The following macro text displays the "End" button, "_ OnButton ()" is executed when the button is pressed, and the process ends.

```

1 : function main()
2 : {
3 :   _BtnOpen(0, "End");
4 :   _Printf("Press the 'End' button to end the macro\n");
5 :   _Stop();
6 :   _BtnClose(0);
7 : }
8 :
9 : function _OnButton(BtnNo)
10 : {
11 :   _Restart();
12 : }
```

The execution screen is as follows.



"_ BtnClose()" on line 6 deletes the button, but since all resources are discarded at the end of the macro, it is OK without describing it. (Although the manner of the program is good)

Up to 8 buttons and check boxes can be displayed.

In the macros shown below, we do a somewhat elaborate process.

Two buttons are displayed, "Send" and "End", and one check box "Add line break" is displayed.

Click "End" button to end the macro.

Press the "Send" button to send the text of the current time to the COM port.

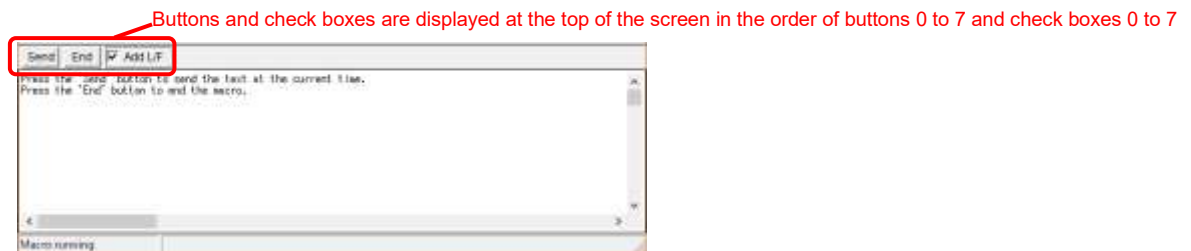
In addition, if the check box "Add line break" is checked, a line feed code will be sent.

```

1 : /* M500 : Button and check box, COM transmission */
2 : function main()
3 : {
4 :     /* Button / check box display */
5 :     _BtnOpen    (0, "Send");
6 :     _BtnOpen    (1, "End");
7 :
8 :     _ChkBoxOpen (0, "Add L/F");
9 :     _ChkBoxSetSts(0, _TRUE);
10 :
11 :     /* COM port generation, open */
12 :     _ComCreate(4);
13 :     _ComOpen  (4, 115200, 8, 0, 1);
14 :
15 :     _Cls();
16 :     _Printf("Press the 'Send' button to send the text at the current time.\n");
17 :     _Printf("Press the 'End' button to end the macro.\n");
18 :
19 :     _Stop();
20 :
21 :     /* Resource destruction */
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 : }

```

The execution screen is as follows.



5.3. Serial port transmission / reception (text data)

The following macros send and receive text data via the serial port.

Communication parameters of the serial port are set by dialog with "PortSetting" button.

The text to be sent is entered by dialogue, with a line feed code added at the end and sent.

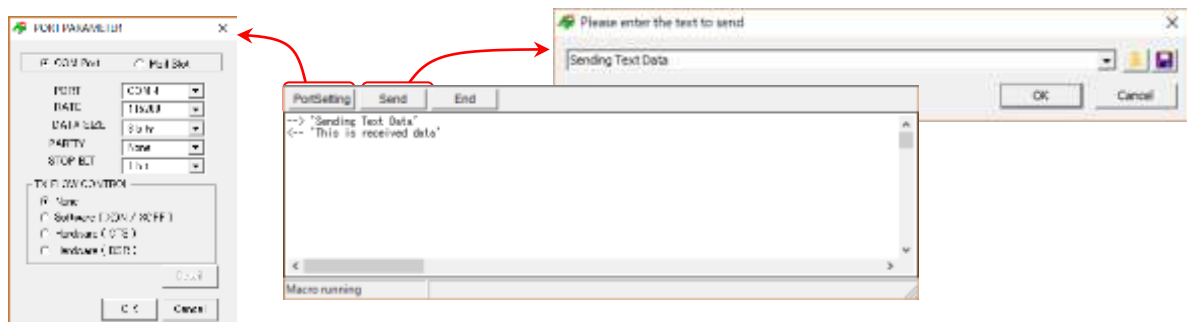
Received text data is displayed as it is.

```

1 : // M510 : Send and receive text data via serial port
2 :
3 : define port = 0;
4 : define $InpTxt = "";
5 :
6 : function main()
7 : {
8 :     // Button display
9 :     _BtnOpen(0, "PortSetting");
10 :    _BtnOpen(1, "Send");
11 :    _BtnOpen(2, "End");
12 :
13 :    _BtnEnable(1, _FALSE);
14 :
15 :    _Cls();
16 :
17 :    _Stop();
18 :
19 :    /* Resource destruction */
20 :    _BtnClose(0); _BtnClose(1); _BtnClose(2);
21 :    if (port != 0) {
22 :        _ComClose(port);
23 :        _ComDelete(port);
24 :    }
25 : }
26 :
27 : // Button Event processing
28 : function _OnButton(BtnNo)
29 : {
30 :     define $txt;
31 :
32 :     if (BtnNo == 0) { // - "Port setting" button pressed?
33 :         if (port = _ComCreateByDialog(port)) {
34 :             _ComSetOnRxText(port, "OnRxText");
35 :             _ComSetOnRxCtrl(port, "OnRxCtrl");
36 :             if (!_ComIsOpened(port) && _ComIsOpenPossible(port)) {
37 :                 _ComOpen(port);
38 :                 _BtnEnable(1, _TRUE);
39 :             }
40 :         }
41 :     }
42 :     else if (BtnNo == 1) { // - "Send" button pressed?
43 :         if ($InpTxt = $_Input("Please enter the text to send", $InpTxt)) {
44 :             _ComSendText(port, $_StrCat($InpTxt, "\n"));
45 :             _Printf("--> '%s'\n", $InpTxt);
46 :         }
47 :     }
48 :     else if (BtnNo == 2) _Restart(); // - "End" button pressed?
49 : }
50 :
51 : // Text reception event processing
52 : function OnRxText($Text)
53 : {
54 :     _Printf("<-- '%s'", $Text);
55 : }
56 :
57 : // Control code reception event processing
58 : function OnRxCtrl(ctrl)
59 : {
60 :     _Printf("%c", ctrl);
61 : }

```

The execution screen is as follows.



- Processing when the "port setting" button is pressed (lines 32 to 41)

_ComCreateByDialog () sets the parameters of the serial port by dialog, and generates the port with the specified parameter.

(If you press the "Cancel" button, _FALSE is returned without doing anything)

_ComSetOnRxText () and _ComSetOnRxCtrl () specify "Execution function when receiving text" and "Execution function when receiving control code", respectively.

If the port is in the closed state (! _ComIsOpened(port)) and the port is openable (_ComIsOpenPossible(port)), open the port and enable the "send" button.

- Processing when the "Send" button is pressed (lines 42 to 47)

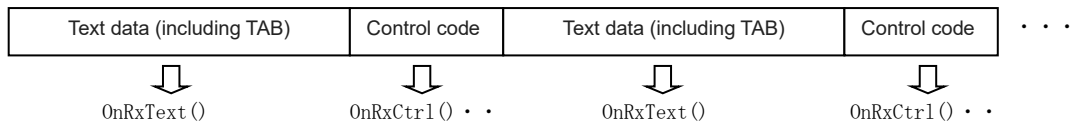
\$_Input () inputs a character string by dialog, and returns the input character string. (When canceling, return empty string)

When a string is entered, _ComSendText() sends a string and a line feed code to the port.

- Text reception event (OnRxText()) and control code reception event (OnRxCtrl())

Partial text (separated by control code (0x00 to 0x1F, 0x7F) excluding TAB (0x09)) notify to "OnRxText ()".

The control code notify to "OnRxCtrl ()" byte by byte.

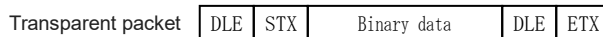


In this case, in order to display and output all the received text as it is, these two events are accepted.

5.4. Serial port transmission / reception (binary data)

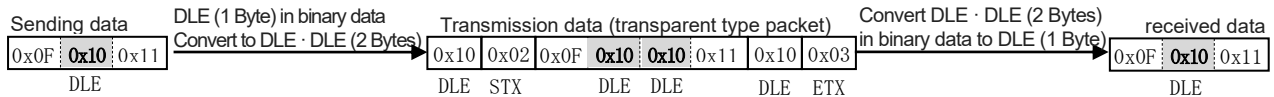
Binary data can be sent and received with transparent type packets.

Transparent type packet is a byte stream starting with DLE · STX and ending with DLE · ETX. (STX = 0x02, ETX = 0x03, DLE = 0x10)



Two consecutive DLEs in binary data are converted to one DLE.

For example, to transmit 3 bytes of binary data (0x0F, 0x10, 0x11), it is as follows.



In order to transmit and receive binary data with transparent type packets, it is necessary to reserve a memory block for storing binary data with _MemAlloc ().

To send 3 bytes of binary data in the above example as transparent type packet, write it as follows.

(Assume that the serial port is open)

```
define bno;

bno = _MemAlloc (1024);           // Memory block allocation (executed only once at the beginning)
. . .
_PutMemBlk (bno, 0, 3, 0x1100F); // Data setting to memory block (first 3 bytes)
_ComSendPacket (port, bno, 0, 3); // Send binary data as transparent type packet
. . .
_MmemFree (bno);                 // Memory block release (last executed only once)
```

※ Use _ComSendBinary() to send binary data as a simple byte sequence rather than a transparent type packet.

In the code example above, change "_ComSendPacket(port, bno, 0, 3);" to "_ComSendBinary(port, bno, 0, 3);"

In this case, 3 bytes of data (0x0F, 0x10, 0x11) are simply transmitted.

The macro shown below is a modified version of the above text transmission / reception macro (M510) for transmission and reception of binary data (transparent type packet).

When you press the "Send" button, 3 bytes of binary data (0x0F, 0x10, 0x11) are transmitted as transparent type packets.

Also, when receiving a transparent packet, it displays the received binary data in hexadecimal.

```

1 : // M520 : Send and receive binary packet data via serial port
2 :
3 : define bno_tx = 0;
4 : define bno_rx = 0;
5 : define port = 0;
6 : define $InpTxt = "";
7 :
8 : function main()
9 : {
10 :    // Memory block allocation
11 :    bno_tx = _MemAlloc(1024);
12 :    bno_rx = _MemAlloc(1024);
13 :
14 :    // Button display
15 :    _BtnOpen(0, "PortSetting");
16 :    _BtnOpen(1, "Send");
17 :    _BtnOpen(2, "End");
18 :
19 :    _BtnEnable(1, _FALSE);
20 :
21 :    _Cls();
22 :
23 :    _Stop();
24 :
25 :    /* Resource destruction */
26 :    _BtnClose(0); _BtnClose(1); _BtnClose(2);
27 :    if (port != 0) {
28 :        _ComClose(port);
29 :        _ComDelete(port);
30 :    }
31 : }
32 :
33 : // Button Event processing
34 : function _OnButton(BtnNo)
35 : {
36 :     if (BtnNo == 0) { // - "Port setting" button pressed?
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) { // - "Send" button pressed?
47 :         // Transmit binary data (0x0F, 0x10, 0x11) as transparent type packet
48 :         _PutMemBlk(bno_tx, 0, 3, 0x11100F);
49 :         _ComSendPacket(port, bno_tx, 0, 3);
50 :     }
51 :     else if (BtnNo == 2) _Restart(); // - "End" button pressed?
52 : }
53 :
54 :
55 : // Packet reception event processing
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 : }

```

The execution screen is as follows.

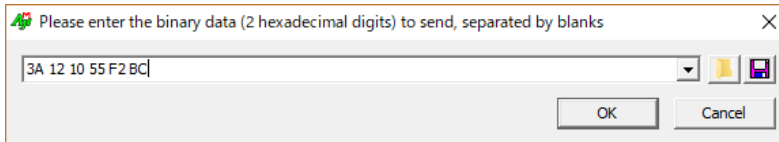


In this sample macro, only 3 bytes of fixed data is transmitted.

You can enter and send any byte sequence in the dialog box by changing lines 47-49 as follows.

```
// Binary data input and transparent packet transmission
$InpTxt = $_Input("Please enter the binary data (2 hexadecimal digits) to send, separated by blanks");
if ($InpTxt != "") { // With character string input?
    if (bytes = _HexStrToBinary($InpTxt, bno_tx, 0)) { // Hexadecimal character string → binary conversion
        _ComSendPacket(port, bno_tx, 0, bytes); // Binary packet data transmission
    }
}
```

In the dialog box below, enter any byte sequence.



Dump display of received data on lines 58 to 64 can be simplified as follows.

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

5.5. Record to Profile

In the sample macro (M520) earlier, you had to set the port in the dialog at startup.

If you know the port number and open status that was set at the time of the last execution, you can open the serial port with the previously set port number without displaying the dialog.

With AJATERM - MACRO, information can be recorded to and read from the profile.

Use `_SetProfileVal()` to record the number in the profile, and `_GetProfileVal()` to read the recorded number.

Also, you can specify the section name to be recorded and organize the information to be recorded in the section.

The section name to be recorded is specified with `"_ SetProfileSect()"`.

※ The entity of the profile is the registry.

The registry path is "HKEY_CURRENT_USER / Software / AjrCtlXX / AjrMacroXX / <macro text name> / <section name>".

("XX" will be "32" for AjrMacro32.exe, "64" for AjrMacro64.exe)

The next macro keeps the port number and open state of the serial port permanently by recording to the profile.

```
1 : define port = 0;
2 : define fOpened = _FALSE;
3 :
4 : function main()
5 : {
6 :     // Button display
7 :     _BtnOpen(0, "PortSetting"); _BtnEnable(0, _TRUE );
8 :     _BtnOpen(1, "PortClose");   _BtnEnable(1, _FALSE);
9 :     _BtnOpen(2, "End");         _BtnEnable(2, _TRUE );
10 :
11 :     _Cls();
12 :
13 :     // Port number, open status read
14 :     port = _GetProfileVal("PORT", port );
15 :     fOpened = _GetProfileVal("fOpened", _FALSE);
16 :
17 :     // Open persistence
18 :     if (fOpened) {
19 :         if (_ComIsOpenPossible(port)) {
20 :             _ComOpen(port);
21 :             _BtnEnable(1, fOpened = _TRUE);
22 :             _Printf("Opened port %d.\n", port);
23 :         }
24 :     }
25 :
26 :     _Stop();
27 :
28 :     // Port number, open state record
29 :     _SetProfileVal("PORT", port );
30 :     _SetProfileVal("fOpened", fOpened);
31 : }
32 :
```

つづく

つづき

```

33 : // Button Event processing
34 : function _OnButton(BtnNo)
35 : {
36 :     if (BtnNo == 0) { // ●"Port setting" button pressed
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) { // ●"Port close" button pressed
45 :         _ComClose(port);
46 :         _BtnEnable(1, fOpened = _FALSE);
47 :     }
48 :     else if (BtnNo == 2) _Restart(); // ●"End" button pressed
49 : }

```

Final version of serial port send / receive macro

The next sample macro persists the port number and open state, and sends and receives text and binary packets.

```

1 : // M530 : Port state persistence, text / binary transmission and reception
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 :     // Memory block allocation
12 :     bno_tx = _MemAlloc(1024);
13 :     bno_rx = _MemAlloc(1024);
14 :
15 :     // Button display
16 :     _BtnOpen(0, "PortSetting"); _BtnEnable(0, _TRUE);
17 :     _BtnOpen(1, "PortClose"); _BtnEnable(1, _FALSE);
18 :     _BtnOpen(2, "SendText"); _BtnEnable(2, _FALSE);
19 :     _BtnOpen(3, "SendBinary"); _BtnEnable(3, _FALSE);
20 :     _BtnOpen(4, "End"); _BtnEnable(4, _TRUE);
21 :
22 :     _Cls();
23 :
24 :     // Port number, open status read
25 :     port = _GetProfileVal("PORT", 0);
26 :     fOpened = _GetProfileVal("fOpened", _FALSE);
27 :
28 :     // Persisting the serial port
29 :     if (port != 0) {
30 :         _ComCreate(port);
31 :         if (fOpened) OpenPort();
32 :     }
33 :
34 :     _Stop();
35 :
36 :     // Port number, open state record
37 :     _SetProfileVal("PORT", port);
38 :     _SetProfileVal("fOpened", fOpened);
39 :
40 :     /* Resource destruction */
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 :

```

```

47 : // Button event processing
48 : function _OnButton(BtnNo)
49 : {
50 :     define bytes;
51 :
52 :     if (BtnNo == 0) { // - "Port setting" button pressed?
53 :         if (port = _ComCreateByDialog(port)) {
54 :             _ComSetOnRxText (port, "OnRxText");
55 :             _ComSetOnRxPacket(port, "OnRxPacket", bno_rx);
56 :             OpenPort();
57 :         }
58 :     }
59 :     else if (BtnNo == 1) { // - "Port close" button pressed?
60 :         _ComClose(port);
61 :         fOpened = _FALSE;
62 :         _BtnEnable(1, _FALSE); _BtnEnable(2, _FALSE); _BtnEnable(3, _FALSE);
63 :     }
64 :     else if (BtnNo == 2) { // - "Text transmission" button pressed?
65 :         // Enter and send text
66 :         if ($InpTxt = $_Input("Please enter the text to send", $InpTxt)) {
67 :             _ComSendText(port, $_StrCat($InpTxt, "\n"));
68 :         }
69 :     }
70 :     else if (BtnNo == 3) { // - "Binary transmission" button pressed?
71 :         // Binary data input and transparent packet transmission
72 :         $InpTxt = $_Input("Please enter the binary data (2 hexadecimal digits) to send, separated by blanks");
73 :         if ($InpTxt != "") { // With character string input?
74 :             if (bytes = _HexStrToBinary($InpTxt, bno_tx, 0)) { // Hexadecimal character string → binary conversion
75 :                 _ComSendPacket(port, bno_tx, 0, bytes); // Binary packet data transmission
76 :             }
77 :         }
78 :     }
79 :     else if (BtnNo == 4) _Restart(); // - "End" button pressed?
80 : }
81 :
82 : // Text reception event processing
83 : function OnRxText($Text)
84 : {
85 :     _Printf("%s", $Text);
86 :     _Printf("\n");
87 : }
88 :
89 : // Packet reception event processing
90 : function OnRxPacket(len)
91 : {
92 :     _DumpMemBlk(bno_rx, 0, len);
93 : }
94 :
95 : // Open serial port
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. Right click event and popup menu

AJATERM - MACRO can detect right click and display pop - up menu.

Right-click while pressing Shift / Ctrl key, the event processing function "_OnRClick()" is executed.

The following four arguments are passed to "_OnRClick()".

- x, y - Character clicked position (pixel position with the origin of the macro window upper left)
- shift - Shift key pressed flag (_TRUE: pressed)
- ctrl - Shift key pressed flag (_TRUE: pressed)

※ To generate a right click event without pressing the Shift / Ctrl key, execute "_EnableStdRClick(_FALSE)".

However, when "_EnableStdRClick(_FALSE)" is executed, the standard right-click popup menu will not be displayed.

In the following macros, Shift / Ctrl + Right click displays a popup menu and you can select "send text" or "exit".

Select "Send text" to send the character string at the current time, and select "End" to end the macro.

Also displays received text data.

```

1 : // M550 : Right click and popup menu
2 : define port = 4;
3 :
4 : function main()
5 : {
6 :     _Cls();
7 :     _Printf("Shift / Ctrl + Right-click to display the pop-up menu.¥n");
8 :     _Printf("While the popup menu is displayed, the display of the received text stops.¥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 $menu[] = { "Send text",
23 :                       "End"};
24 :
25 :     id = _PopupMenu($menu[], x, y);
26 :
27 :     if (id == 0) {                // - Send text
28 :         _ComSendText(port, $_StrCat($_DateAndTime(), "¥n"));
29 :     }
30 :     else if (id == 1) {           // - End
31 :         _Restart();
32 :     }
33 : }
34 :
35 : function OnRxText($Text)
36 : {
37 :     _Printf("%s", $Text);
38 :     _Printf("¥n");
39 : }

```

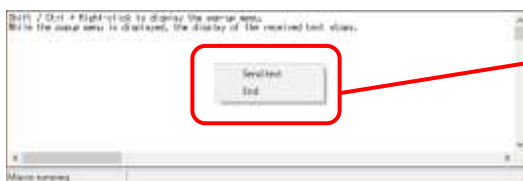
In this sample macro, a popup menu is displayed with the event "_OnRClick ()".

Since other event processing is not executed during event processing, other events will not be executed until execution of "_ _PopupMenu()" is completed (that is, a popup menu is selected or canceled).

For this reason, "OnRxText ()" is not executed and the display of received text stops. (For the method not to stop displaying the received text, see the next page)

However, since all "OnRxText()" events are held, when the popup menu is over, the suspended "OnRxText()" event will be executed at once.

The execution screen is as follows.



Shift / Ctrl + right click to display popup menu

If you do not want to stop displaying the received text, specify the event function to be executed when the menu is selected as the fourth argument of `_PopupMenu()`.

In this event function, when a menu item is selected, the index value (0 to n) of the menu selected as an argument is passed.

When an event function is specified as the fourth argument of `_PopupMenu()`, the return value of `_PopupMenu()` has no meaning. (Undefined value returns)

To generate a menu selection event from the popup menu, do as follows.

```

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

function OnMenu(id)
{
    // The index (0 to n) of the menu item selected for id is set
}

```

When right clicking on the following macro, a popup menu is displayed and you can select "send text" or "exit".

Select "Send text" to send the character string at the current time, and select "Exit" to end the macro.

Also, it displays the received text data even while displaying the popup menu.

```

1 : // M551 : Right click and popup menu (improved version)
2 : define port = 4;
3 :
4 : function main()
5 : {
6 :     _Cls();
7 :     _Printf("Right-click to display the pop-up menu.\n");
8 :     _Printf("While the popup menu is displayed, the received text is displayed.\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 $menu[] = { "Send text",
24 :                       "End" };
25 :
26 :     _PopupMenu($menu[], x, y, "OnMenu");
27 :
28 : }
29 :
30 : function OnMenu(id)
31 : {
32 :     if (id == 0) { // - Send text
33 :         _ComSendText(port, $_StrCat($_DateAndTime(), "\n"));
34 :     }
35 :     else if (id == 1) { // - End
36 :         _Restart();
37 :     }
38 : }
39 :
40 : function OnRxText($Text)
41 : {
42 :     _Printf("%s\n", $Text);
43 : }

```

5.7. Time chart Graph display and timer

With AJATERM - MACRO, you can display a graph of the time chart (how the value changes over time).

The following macros graphically display eight sine waves with different phases.

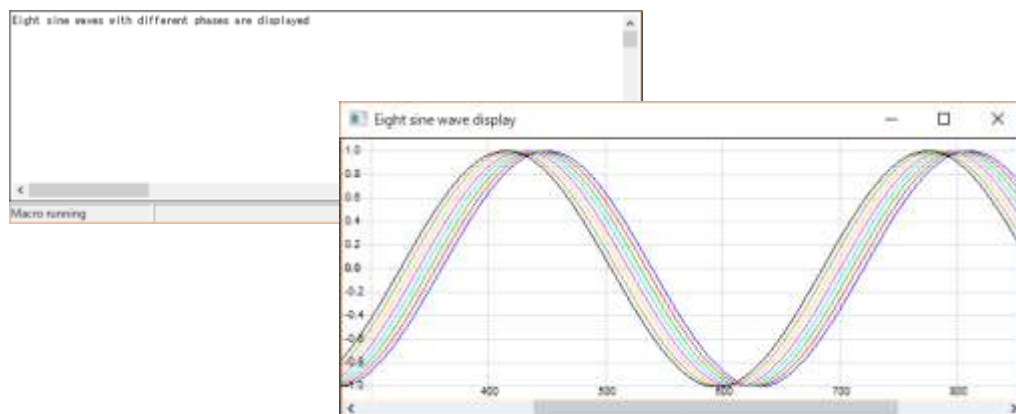
Data is updated every 100 ms.

```

1 : // M560 : Eight sine waves with different phases are displayed
2 : option double;
3 :
4 : define t    = 0;
5 :
6 : function main()
7 : {
8 :     define dat[8];
9 :
10 :    _Cls();
11 :    _Printf("Eight sine waves with different phases are displayed\n");
12 :
13 :    _TmcOpen("Eight sine wave display");
14 :    _TmcSetItems(8);
15 :    _TimerStart(1, 100);
16 :
17 :    _Stop();
18 :
19 :    _TmcClose();
20 :
21 : }
22 :
23 : // Timer procedure
24 : function _OnTimer(tid)
25 : {
26 :     define i, dt;
27 :     define dat[8];
28 :
29 :     if (tid == 1) { // Timer-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 : // Processing when chart window closes
39 : function _OnTmcClose()
40 : {
41 :     _Restart();
42 : }

```

The execution screen is as follows.

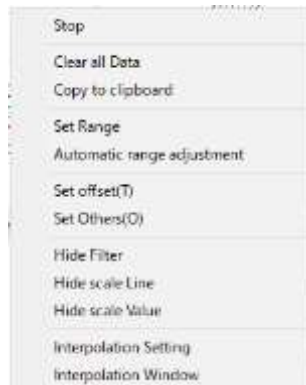


Although sine waves are simply displayed here, you can draw output waveforms of sensors etc. by receiving data from the serial port and sequentially displaying the data.

In the time chart window, you can perform the following operations.

Pop-up menu

Right click on the graph, the following popup menu will be displayed.

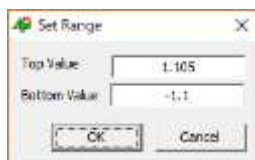


- | | | |
|-------------------|---|--|
| Stop | : | Stop the graph display. Next time will be the "Start" menu. |
| Clear all data | : | Discard all buffered data. |
| Copy to clipboard | : | Copy the graph display contents (bitmap) to the clipboard. |
| Hide filter | : | Hide the filter (check box) in the upper left of the control.
The "Show filter" menu will change the next time. |
| Hide scale line | : | Hide the scale lines (light gray lines).
Next time will change to "Show scale line" menu. |
| Hide scale value | : | 目盛り数値を非表示にします。
The display will change to "Show scale value" next time. |

"Set Range", "Automatic range adjustment", "Set offset", "SetOther" will be explained shortly.

Set Range

When "Set Range" is selected from the pop-up menu, the following dialog box will be displayed.



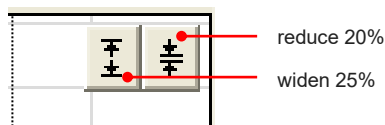
If you enter the range value here and press the "OK" button, the range of the graph is set.

Press the "Cancel" button to cancel the setting.

Range setting with one touch

When placing the mouse cursor in the upper right corner of the time chart graph, two buttons are displayed.

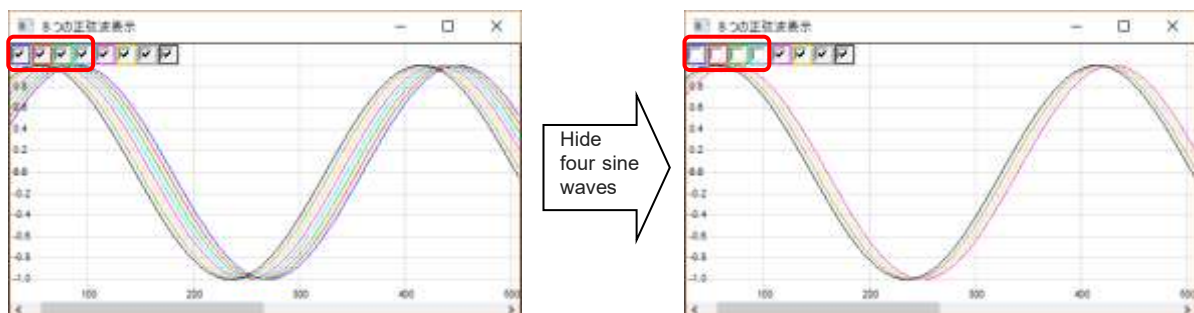
With these buttons, you can widen the range by 25% or reduce it by 20%.



Filter

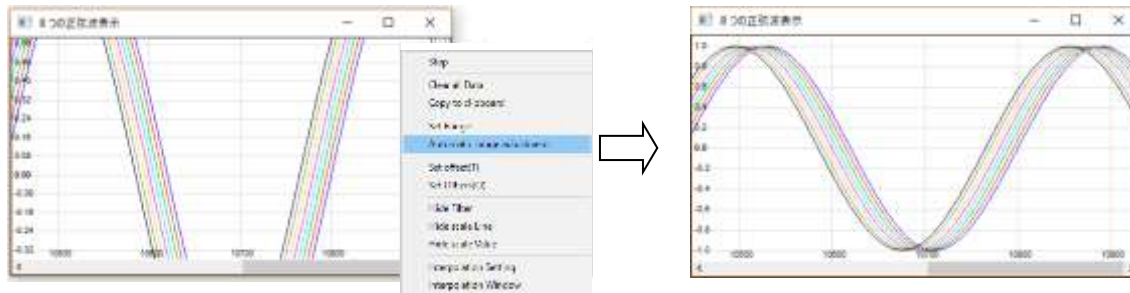
When placing the mouse cursor in the upper left corner of the time chart graph, a check box will be displayed.

The check box on the upper left corner of the window is a display filter, and if you uncheck it, the data for that display color will be hidden.



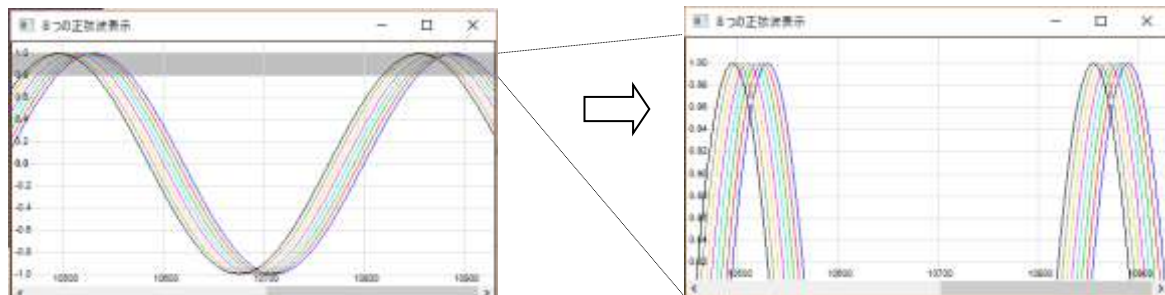
Automatic range adjustment

When "Automatic range adjustment" is selected from the pop-up menu, minimum value and maximum value are calculated from all data (excluding data items which are not displayed in filter), and range setting with $\pm 5\%$ margin I will do.



Range setting by drag operation

While holding down the CTRL key, you can set the range by dragging the part you want to set the range with the left mouse button.



While holding down the CTRL key, drag the range setting part with the mouse. (The part being dragged is grayed out.)

Hold down the CTRL key and release the left mouse button, the range is set for the dragged part.

Even if you drag to the part beyond the top / bottom of the graph, the drag range is set as the range.

If you release the CTRL key first and release the left mouse button, the range setting will not be performed.

Set offset

When you select "Offset setting" from the popup menu, the following dialog will be displayed.



The items 0 to 7 correspond to the displayed data items. (The display color of the outer frame is the same as the graph display color)

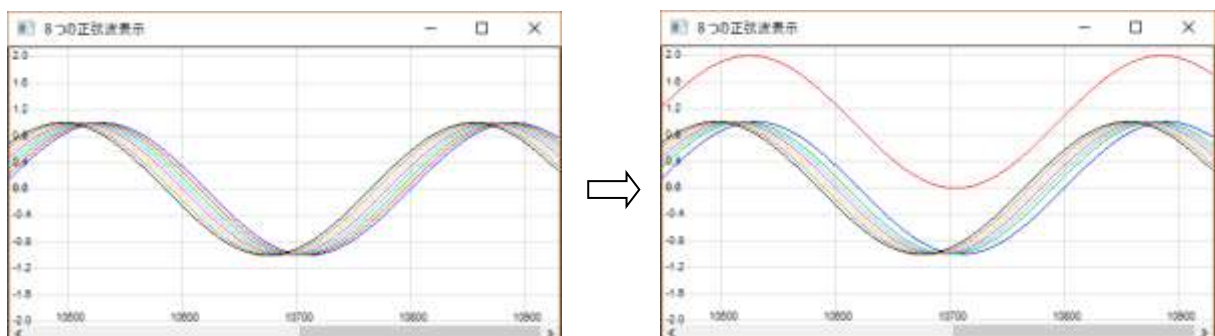
If you set a value here, the graph is displayed with the data value of the relevant data item plus the setting value.

The offset value set following the value setting is reflected in the graph.

Press the "OK" button to confirm the setting. When you press the "Cancel" button, the setting contents are discarded and the original offset value is restored.

When you press the "Reset" button, all offset values are set to "0".

In the following example, "+1.0" is set as the offset value in the data of red display.



5.8. Event driven

Up to this point, macros using event-driven (using `_Stop()` and `_Restart()`) have been described, but macros can be written using a method that does not use event-driven.

The following two macros perform text transmission processing at the current time every second.

The first macro is processing using event driven, and the second macro is processing that does not use event driven.

When using event driven

```

1 : // M570 : Text transmission at the current time every 1 second (event driven)
2 :
3 : define port = 4;
4 :
5 : function main()
6 : {
7 :     define dat[8];
8 :
9 :     _Cls();
10 :    _Printf("Send text at the current time every second.\n");
11 :    _BtnOpen(0, "End");
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 : }

```

When event-driven is not used

```

1 : // M571 : Text transmission at the current time every 1 second (event-driven not used)
2 :
3 : define port = 4,
4 :         fEnd = _FALSE;
5 :
6 : function main()
7 : {
8 :     define dat[8];
9 :
10 :    _Cls();
11 :    _Printf("Send text at the current time every second.\n");
12 :    _BtnOpen(0, "End");
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 : }

```