Windows

C language static analysis tool
SCANA

User's Manual

(Ver. 1. 6. 0. 1)

— CONTENTS —

1. OVERVIEW

This application (hereinafter referred to as SCANA (Static C-language ANAlizer)) is a static analysis tool of C language source program.

We already target C language programs that have been successfully built.

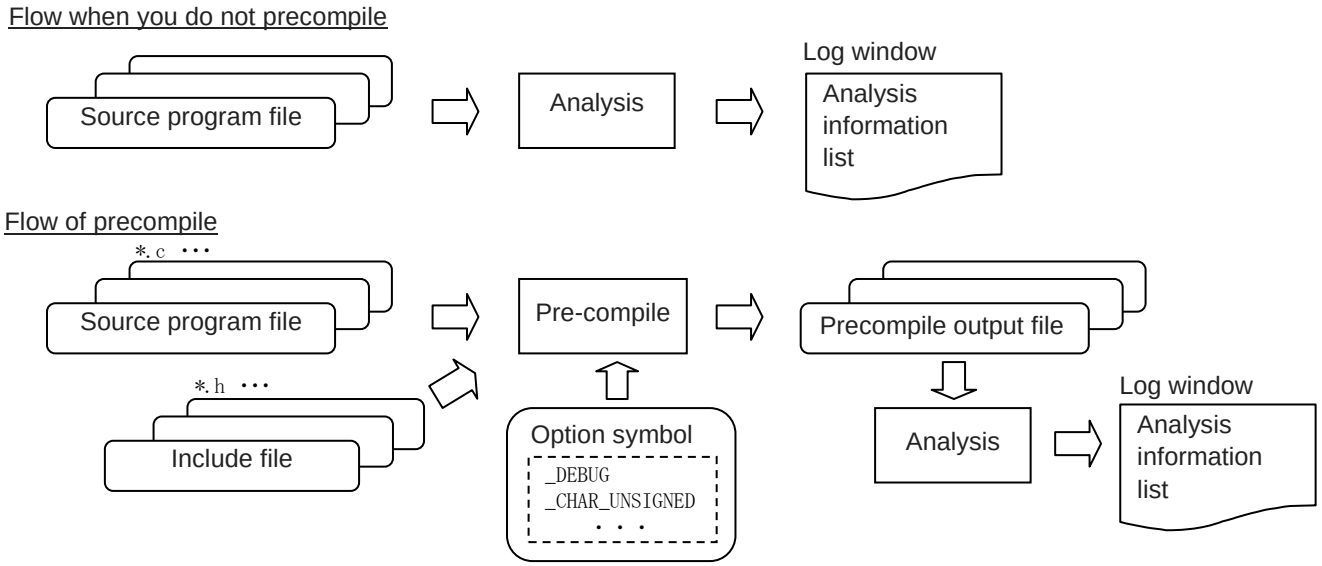
SCANA inputs the C language source program group and performs the following analysis processing.

- Precompile. (resolution of preprocessing statement - resolution of condition compilation (#if, #ifdef, # ifdef ...) or macro (#define))
- Extraction of option symbols ("_DEBUG" etc. of "# ifdef _ DEBUG") used at precompilation and listing of reference points.
- List of type names (typedef name, structure name, union name, enumeration name) and type name definition
- Extraction of macro names used at the time of precompilation and listing of definitions / references.
- A list of function names and a list of function definition points.
- Display function callingstructure.
- List of symbols such as variable names and list of symbol reference points.

1.1. Pre-Compile

When precompile (solution of preprocessing statement - condition compiling (#if, #ifdef, #ifndef · ...) and macro (#define) call) is performed, input the source file precompiled and output and analyze to hold.

When precompiling, it is necessary to give an appropriate include path and an option symbol for conditional compilation.



※ When precompiling, please specify only include path of own program.

Giving an include path of the system may make it difficult to read the precompiled output source due to excessive macro resolution, or if the include file of the system is huge, analysis may become impossible due to insufficient memory.

If conditional compilation (#ifdef, #ifndef, #if, #elif, #else, #endif) is included in the source file or include file, only the part where the condition is "true" is output, and the part of the "false" condition (However, #ifdef, #ifndef, #if, #elif, #else, #endif are output even in false conditions)

In #ifdef, #ifndef, #if, #elif, #else, the operation result of the condition is displayed as "TRUE" or "FALSE".

In #endif, the line number of the corresponding #ifdef, #ifndef, #if is displayed.

(Exsample)

Source program

```
 1 : #define A 10
 2 :
 3 : void func()
 4 : {
 5 :     int     x;
 6 :
 7 :     #if   A == 1
 8 :         x = 10;
 9 :     #elif A == 2
10 :         x = 20;
11 :     #elif A == 3
12 :         x = 30;
13 :     #elif A == 4
14 :         x = 40;
15 :     #elif A == 5
16 :         x = 50;
17 :     #elif A == 6
18 :         x = 60;
19 :     #else
20 :         x = 99;
21 :     #endif
22 : }
```

Pre-Compile result

```
/* "D:¥temp¥a1¥a1.c" */
/* Include-Nest; Macro-Nest; SourceFile; LineNumber; */

/* 0;  0; a1.c;     1; */ /*  #define A  10                          */
/* 0;  0; a1.c;     2; */
/* 0;  0; a1.c;     3; */     void func()
/* 0;  0; a1.c;     4; */     {
/* 0;  0; a1.c;     5; */         int x;
/* 0;  0; a1.c;     6; */
/* 0;  0; a1.c;     7; */ //      #if A==1                       // --> FALSE
/* 0;  0; a1.c;     8; */
/* 0;  0; a1.c;     9; */ //      #elif A==2                     // --> FALSE
/* 0;  0; a1.c;    10; */
/* 0;  0; a1.c;    11; */ //      #elif A==3                     // --> FALSE
/* 0;  0; a1.c;    12; */
/* 0;  0; a1.c;    13; */ //      #elif A==4                     // --> FALSE
/* 0;  0; a1.c;    14; */
/* 0;  0; a1.c;    15; */ //      #elif A==5                     // --> FALSE
/* 0;  0; a1.c;    16; */
/* 0;  0; a1.c;    17; */ //      #elif A==6                     // --> FALSE
/* 0;  0; a1.c;    18; */
/* 0;  0; a1.c;    19; */ //      #else                          // --> TRUE
/* 0;  0; a1.c;    20; */            x=99;
/* 0;  0; a1.c;    21; */ //      #endif                         // --> 7
/* 0;  0; a1.c;    22; */     }
```

Boolean value of #if,#elif,#else

Corresponding #if line number

1.2. SYMBOL

The names of variable names, function names, type names, macro names ... etc. are collectively referred to as "symbols".

Symbol means the token of the following condition.

1) First character is alphabet, underscore (_), or dollar mark ($) · · · (* 1)
2) Subsequent characters are alphabet, underscore (_), dollar mark ($), or numbers (0 to 9)
3) As a special case, even when the above conditions are followed by a period (.) And a number (0 to 9) are also used as symbols ... (* 2)

* 1: Although it is a violation under strict C language rules, some processing systems allow dollar marks.
* 2: It is a violation under strict C language rules, but depending on the processing system, port description such as "P2.1" or "P10.0" is allowed.

However, an array variable treats a character string to which '[]' is added like 'arr []' as a variable.

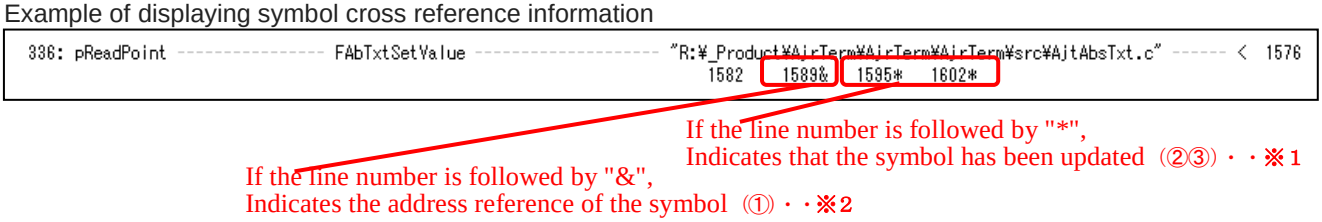
Reserved names in C language ("if", "int", "static", "while", etc.) are not regarded as symbols.

1.3. SYMBOL DETECTION

It detects symbols from the source program and displays a list of symbols and cross-reference information of symbols. You can also set filters when extracting symbols

In the display of the reference position(file name, line number) of the symbol, if the symbol (variable) has been updated, it is indicated "*" at tail of the line number is updated. (* 1)

Example of displaying symbol cross reference information



AjtAbsTxt.c

```
1589    hCtk = AjcCtkCreate(AJCTKFLG_INCSPACE, cbCtkAbTxtSetVal, (UX)&pReadPoint);
                                                                              ①
1590
1591    do {
1592      //----- 数値語句解析先頭位置設定 ----------
1593      if (pAbstInfo->PfxStr[0] != 0) {          //  数値直前
1594        if (p = MAjcStrStr(pLine, pAbstInfo->PfxStr)) {   //    数値
1595          pReadPoint = p + MAjcStrLen(pAbstInfo->PfxStr);   //
1596        }
                    ②
1597        else {                                  //    数値
1598          break;                                //
1599        }
1600      }
1601      else {                                    //  数値直前
1602        pReadPoint = pLine;                     //    先頭
1603      }
                ③
```

※1 The symbol "*" indicating that the symbol has been updated is a simple one and is detected under the following conditions.

・The words immediately before / after the symbol are "++" or "-"
・The word immediately after the symbol is “=” “+ =” “-=” “* =” “/ =” “% =” “& =” “| =” “^ =” “>> =” or “<<

If the symbol is enclosed in parentheses or is a structure member, etc., the symbol update may not be recognized.
The following example does not recognize symbol (var) updates.

```
++p->var;        (var) += 10;        memset(&var, 0, sizeof var);
```

※2 The symbol "&", which indicates that the symbol is addressed, is simple and is detected under the following conditions.

・"&" Just before the symbol
・The two symbols before the symbol are "{", "," "(" or ")"

Therefore, other than address reference, it may be considered as address reference.
In the following example, "&" is added even though it is not an address reference.

```
a = (b + c) & d;        //    "d" is considered as an address reference because it is arranged in the order ")" "&" "symbol"
```

1.4. DETECTION OF FUNCTION DEFINITION

Detects and extracts function definitions from the specified source program group.
When the words are arranged in the following order, the definition of the function is recognized.

Symbol	(	・・・	)	{	・・・	}

"Symbol" is recognized as a function name and "{" to "}" is recognized as the inside (function body) of the function.
If parentheses ('(・・・) ' or '{・・・}') are nested, it is regarded as an arbitrary phrase group until nesting ends.
The function name of the function definition detected in the source program is called "internal function".
In contrast, a function not defined in the source program is called an "external function".
Function macros (macros with arguments) are also handled as external functions (if precompilation does not expand macros).
You can also set filters when extracting internal function definitions.

1.5. FUNCTION CALL DETECTION

When words and phrases are arranged in the following order inside the function (function body), it is recognized as a function call.

Symbol	(

At this time, "symbol" is recognized as the calling function name.
Macro calls of function macros (macros with arguments) are also recognized as function calls.
If the symbol matches the internal function name (that is, it is defined in the source program), it is recognized as a call to the internal function.
If the symbol does not match the internal function name, it is recognized as a call to the external function.

＜Caution＞
In the case of a dynamic call with a function pointer etc., the variable name of the function pointer is recognized as the function name.
In the following cases, "pFunc" is recognized as the calling function name.

```
int  TopFunc(int flag, void (*pFunc)(int x, int y))
{
    ・・・
    pFunc'(10, 20);
    ・・・
}
```

```
void  cbFunc(int x, int y)
{
    ・・・
}

void  SubFunc(void)
{
    TopFunc(0x1000, cbFunc);
}
```

In the example above, calling TopFunc() from SubFunc() will pass the function pointer of cbFunc() as an argument and actually call cbFunc() from TopFunc(), but this is not recognized .

1.6. TOP LEVEL FUNCTION

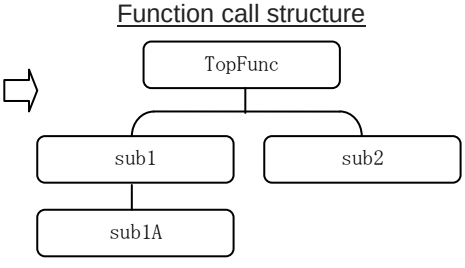
The top level function means a function located at the vertex of the function call structure.
For example, "TopFunc ()" is the highest level function in the sample source as shown below.

Sample Source

```
int  TopFunc(int arg1, int arg2)
{
    ・・・
    sub1();
    sub2();
    ・・・
}

void  sub1(void)
{
    sub1A();
}
```

```
void  sub2(void)
{
    ・・・
}

void  sub1A(void)
{
    ・・・
}
```

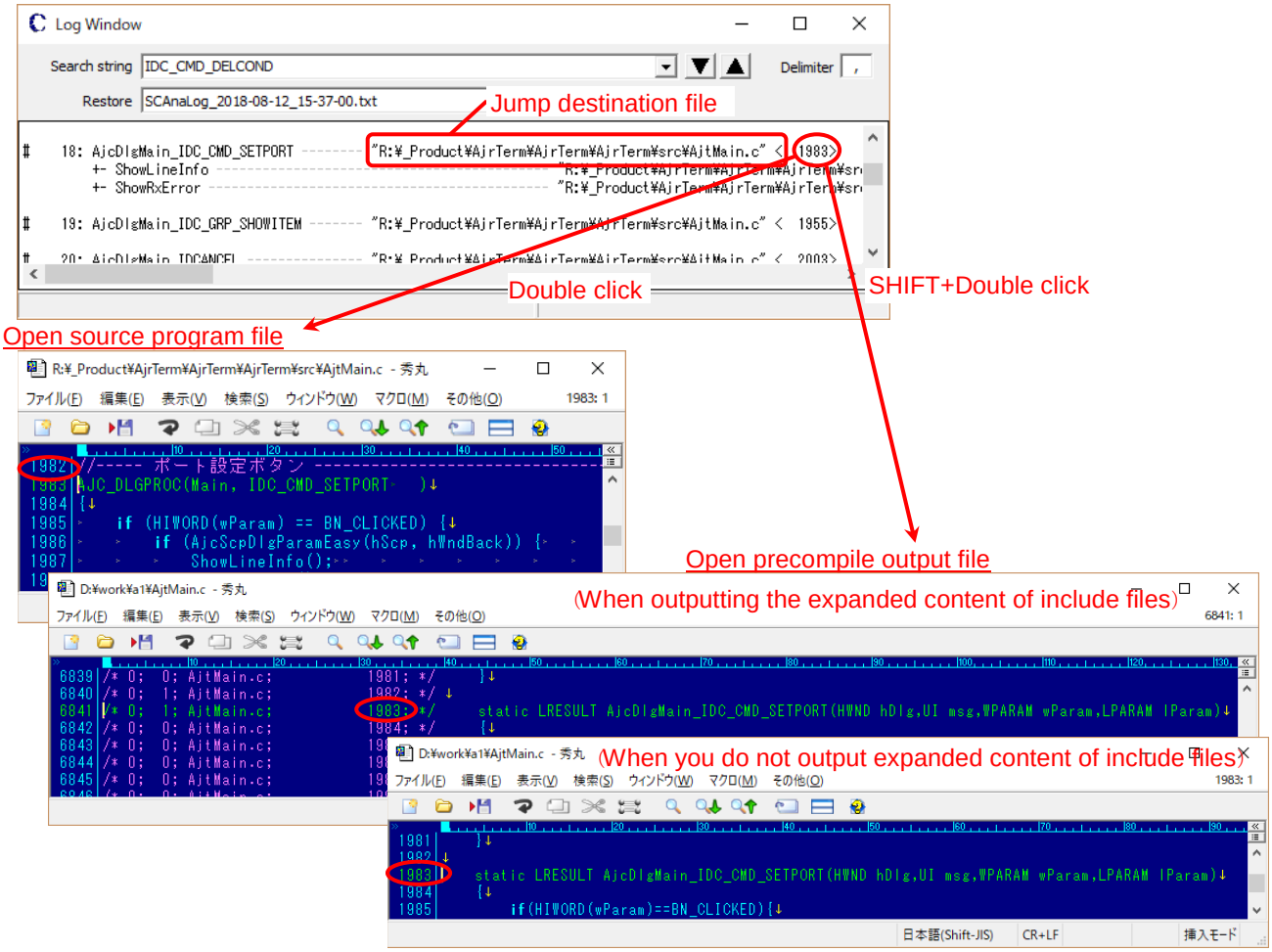
Function call structure



Functions that are not called from any of the source programs are automatically recognized as "top level functions"
Functions starting with a specific character string can be additionally specified as "top level functions", or individually "top level functions" can be specified.

1.7. TAG JUMP

Double-click on the log window to open the tag jump destination file.
If you double-click while holding down the SHIFT key, open the precompile output file.



1.7.1. Tag jump destination path name

The part sandwiched by the double quote (") of the double clicked line becomes the path name of the tag jump destination file.

If there is no part sandwiched by double quotes (") on the double clicked line, search backward to find the path.

If the SHIFT key is pressed, the path name found is changed to the path name of the precompiled result.

1.7.2. Tag jump line number

When double-clicking on the numeric part, the number is regarded as the line number.

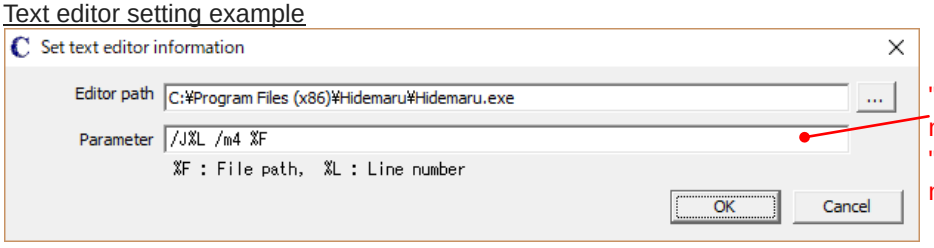
When double-clicking on a part other than a number, the part sandwiched by '<' and '>' of the double clicked line is regarded as the line number.

If there is no sandwiched part between '<' and '>' in the double clicked line, the line number = 1.

1.7.3. Setting up the text editor

In order to validate the line number of the tag jump destination, it is necessary to set the text editor.
From the "Set" → " Text Editor setting" menu, set the command line for starting the text editor as follows.

Text editor setting example

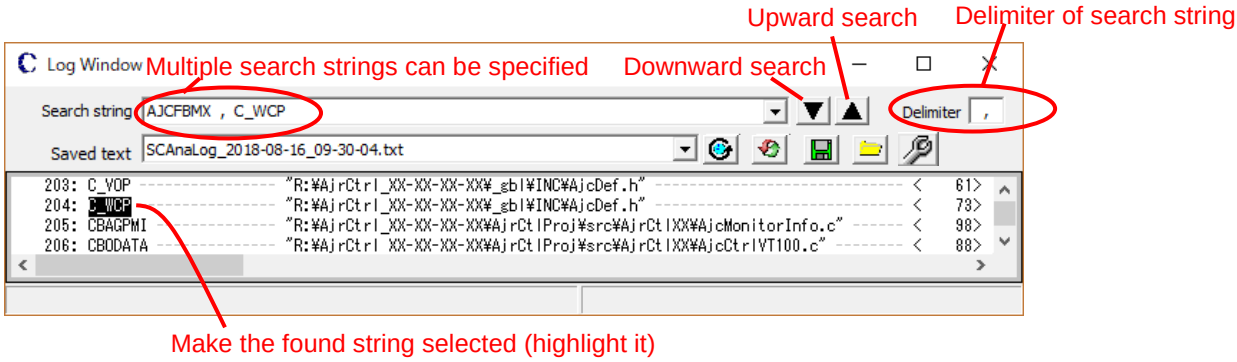


"%F" is converted to the path name of the tag jump destination.
"%L" will be converted to line number.

If you have not set the text editor, simply open the tag jump destination file.
（It is the same behavior as double-clicking on the file in Windows Explorer）

1.8. Search string

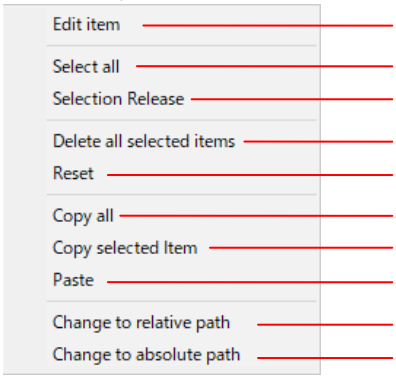
You can search for a string on the search result log window.
To search for multiple character strings, separate them with delimiters.
Specify the delimiter in the upper right corner of the log window.
If you specify a delimiter, the separated spaces before and after each search string are ignored.
When a string is found, it makes the string selected state (highlighted).



1.9. Listbox operation

Right click on the setting list box such as the source file and include path, the operation menu of the list box is displayed.

List box operation menu

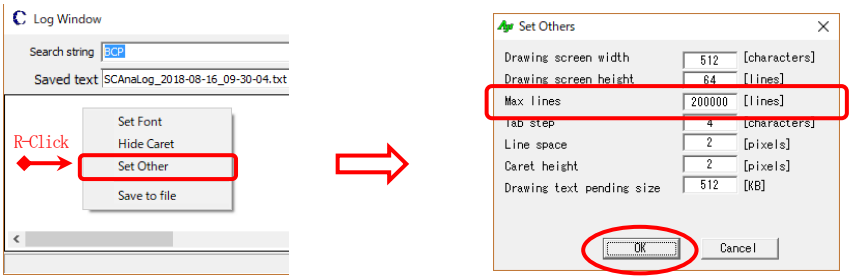


- Edit item — Edit the selected item (Add item if there is no selection item)
- Select all — Make all items selected
- Selection Release — Cancels selection of all items(Alternatively, double click on the list box)
- Delete all selected items — Delete the selected item
- Reset — Delete all items
- Copy all — Copy all items to clipboard (* 1)
- Copy selected Item — Copies the selected item to the clipboard (* 1)
- Paste — Add items from the clipboard (* 1)
- Change to relative path — Convert the selected item to relative path (* 2)
- Change to absolute path — Convert the selected item to an absolute path (* 2)

* 1: Each list box item corresponds to one line of text in the clipboard.
* 2: It is a relative path from "base folder". This menu will not be displayed if "base folder" is not set or in a list box that does not relate to the path name.

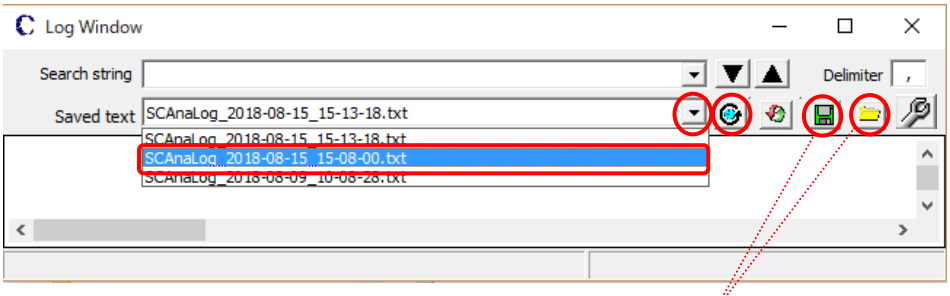
1.10. Maximum number of log lines

The maximum number of lines when logging the analysis result is set to 200,000 lines by default.
If it is insufficient at 200,000 lines, right click on the log window, change "max lines" from the "Set Other" menu and press the "OK" button.



1.11. Save analysis result log

Press the button to save the analysis result log. Press the button to open the analysis result log folder.
To display the saved logs in the past, select the text file to redisplay from "save text" in the log window and press the button.
button sets the text encoding for saving / reading text with or button.

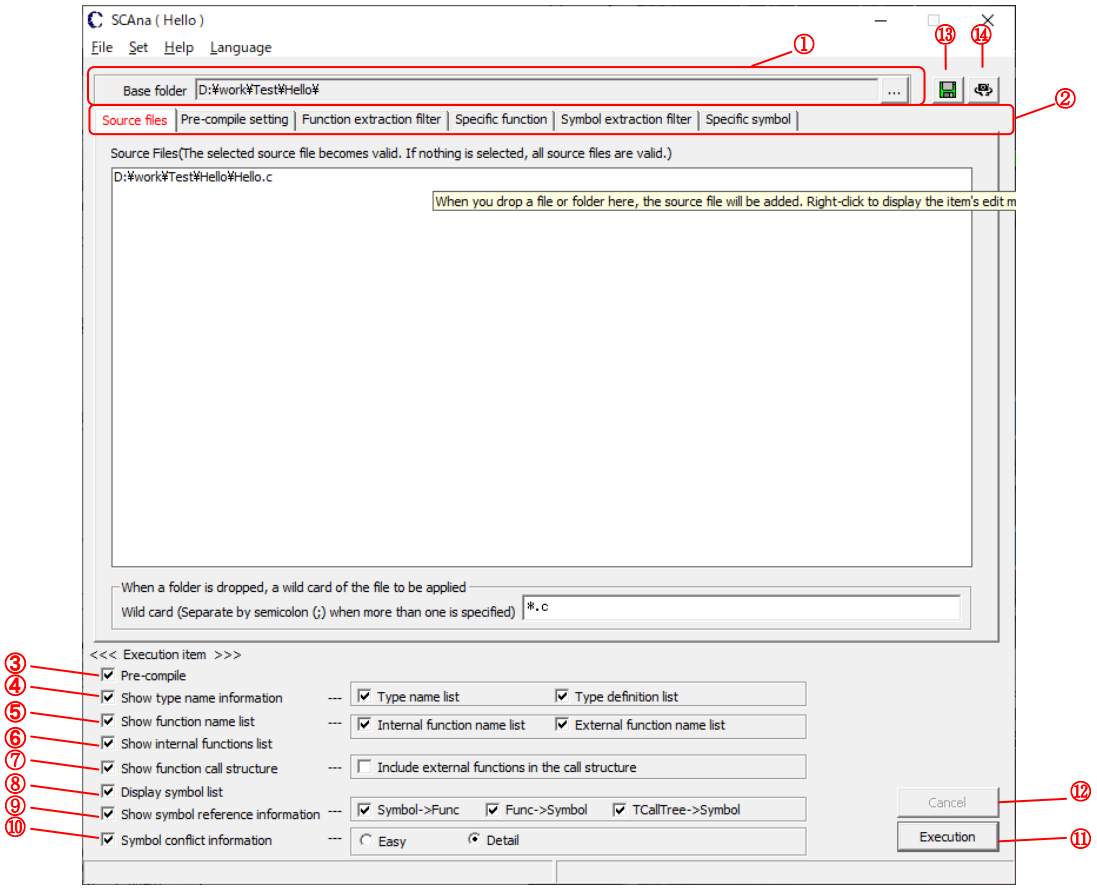


Pressing the button deletes all logs saved in the past.

When saving with the button, the date and time saved in the file name is included.
To change the saved file name, open the folder with button and change directly in Explorer.
At this time, please do not change "SCAnaLog_" at the head.

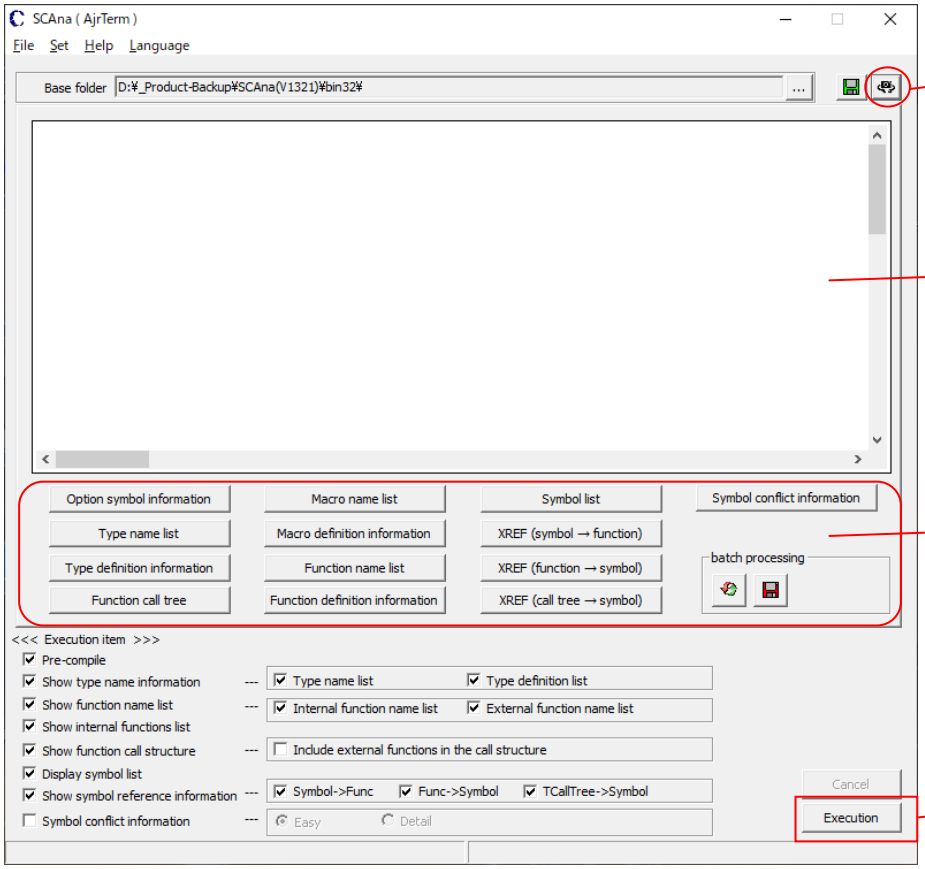
2. MAIN WINDOW

When SCAN is started, the following main window will be displayed.

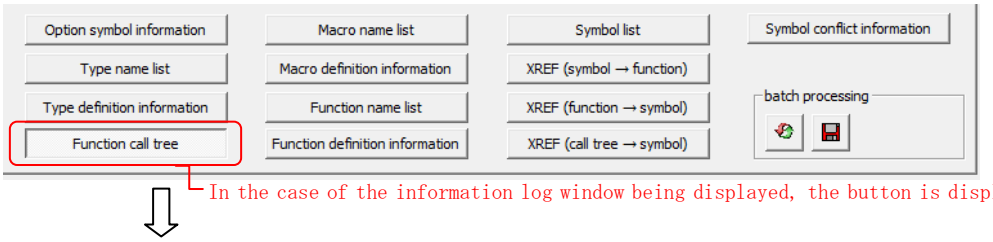


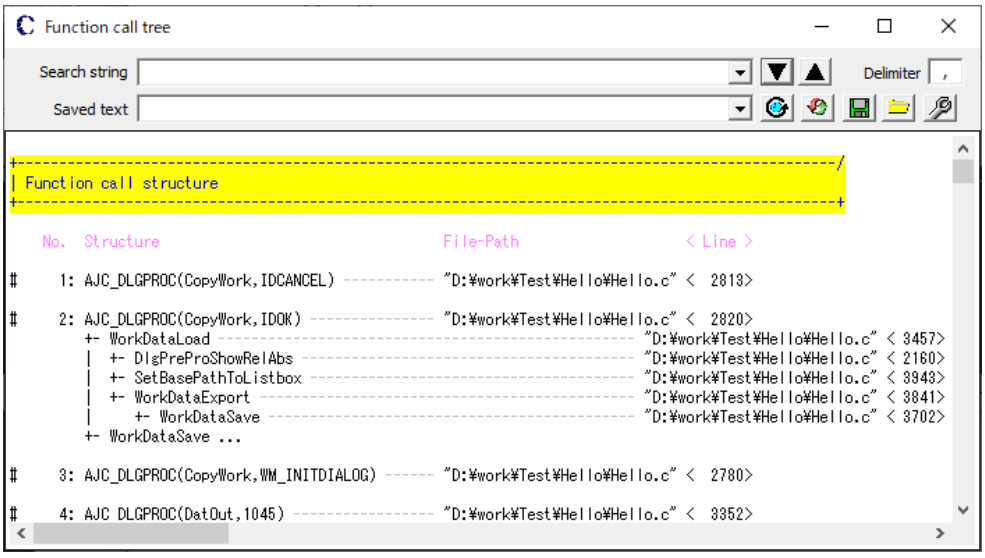
#	内　容
①	Specify the source file or the top-level folder of the include file. It is the base position for automatically searching include files from this base folder, or changing path names to relative paths.
②	Select the tab and make various settings. These will be explained shortly.
③	Check if you want to pre-compile.
④	Check to display type name ･Type name list　：　Display type names list only ･Type definition list　：　Display type name and definition position(file name and line number)
⑤	Check to display a list of function names. ･ Internal function name list **:** Check this to display a list of internal function names ･ External function name list **:** Check this to display a list of external function names.
⑥	Check this to display a list of collected internal functions.
⑦	Check this to display the function call structure. ･ Include external functions in the call structure**:** Include calls to functions not defined in the source program.
⑧	Check this to display the collected symbol list.
⑨	Check this to display cross-reference information of symbols. ･ Symbol -> Function**:** Lists in which function and where each symbol is referenced. ･ Function → Symbol**:** Lists the symbols referenced by each function. ･ Upper-level function → Symbol**:** Lists the symbols referenced by the top-level function (including the calling subfunction)
⑩	Under the top-level function, list conflicting symbols.
⑪	Start analyzing C language source program. If the license has not been set and the trial period has already been exceeded, it will be grayed out. (That is, it can not be executed)
⑫	If you are analyzing C language source program, press Cancel button to abort processing.
⑬	Save current settings
⑭	Switches between the various setting screens (above screen) and the log screen.

When the “Execute” button is pressed or the screen is switched with the  button, the following screen appears.



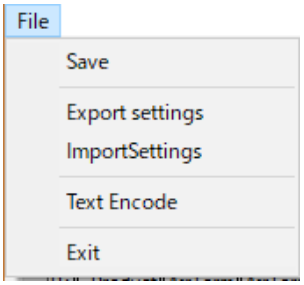
*1： For example, when the “Function Call Structure” button is clicked, the following analysis result log window is displayed / hidden.





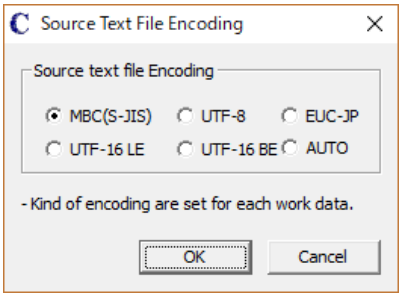
2.1. MENU

2.1.1. File Menu



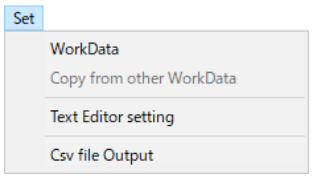
Item	Content
Save	Save the current work name settings The same operation can be performed with the button ( ) in the upper right of the window.
Export setting	Export the settings to a file
Import setting	Import setting contents from file
Text Encode	Encode settings of input / output files, such as source files (* 1)
Exit	Quit the program

* 1: Use the dialog below to set the encoding of the source text file.



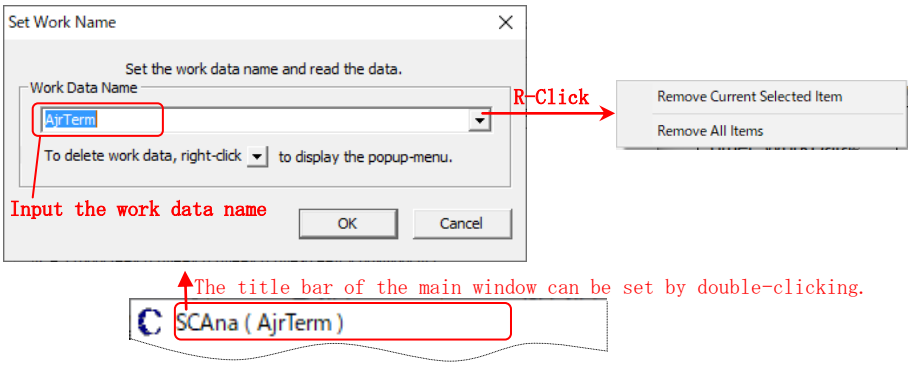
Set the text encoding of source file and include file.
However, if BOM is set in the input text file, BOM takes precedence.
When AUTO is set, text encoding is automatically determined.
The precompile output file is the same format as the source file.

2.1.2. Set Menu

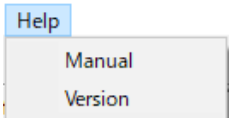


Item	Content
Work Data	Set the work data name. (*1) Work data is a name that summarizes each setting content into one. The same operation can be performed by double-clicking the title bar of the main window.
Copy from other WorkData	Copy the setting contents from other work data.
Text Editor setting	Set information of the text editor to be activated when tag-jump.

*1 : Set the work data name in the following dialog box.

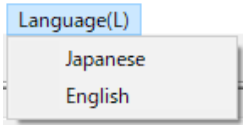


2.1.3. **Help Menu**



Item	Content
Manual	Display SCANA's user's manual (this document)
Version	Show version of SCANA

2.1.4. **Language Menu**

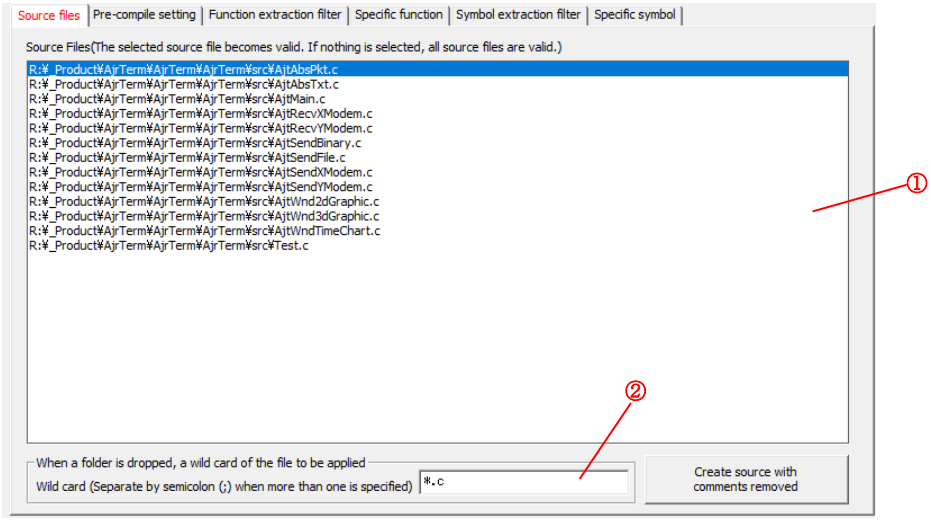


Item	Content
Japanese	All menus and text etc. are displayed in Japanese
English	All menus and text etc. are displayed in English

* The language setting will be effective from the next startup

3. SOURCE FILE SETTING

When you select the "Source files" tab in the main window, the following screen will be displayed.



Here, set the C language source file to be analyzed.

If any source file is selected, the selected source file will be analyzed.
If neither source file is selected, all source files are analyzed.
Double-click (on any list box item) to deselect all items.

3.1. Setting of source file

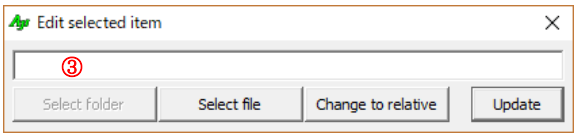
Use one of the following methods to set the source file.

1) Drop the source file from Explorer into the list box of ①. (Convert to relative path when pressing Ctrl key)

2) From the Explorer, drop the folder containing the source file to the list box ①.
(Convert to relative path when pressing Ctrl key)

In this case, the file specified by "wild card" in ② is set from the dropped folder and its subfolder group.

3) Right-click the list box in ① and set the source file from the "Edit item" menu with the following dialog.



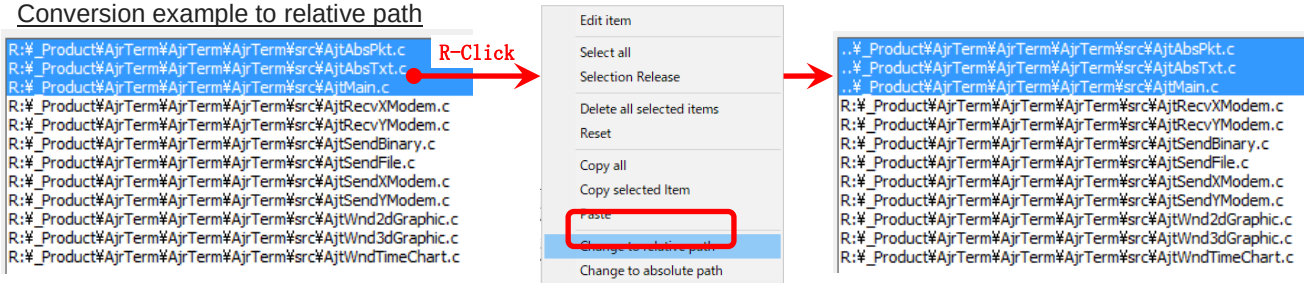
When you press the "Select file" button, you can select the file by the file selection dialog.
Pressing the "Convert to relative path" button converts the selected source file to a relative path from "base folder".
It is OK even if you input the path name of the file directly in ③.
Press the "Update" button to add the set item.

3.2. Convert to Relative Path / Absolute Path

After selecting the item on the list box, right click and select "Convert selected item to relative path" / "Convert selected item to absolute path" menu, You can convert the path name of the source file to relative path or absolute path.

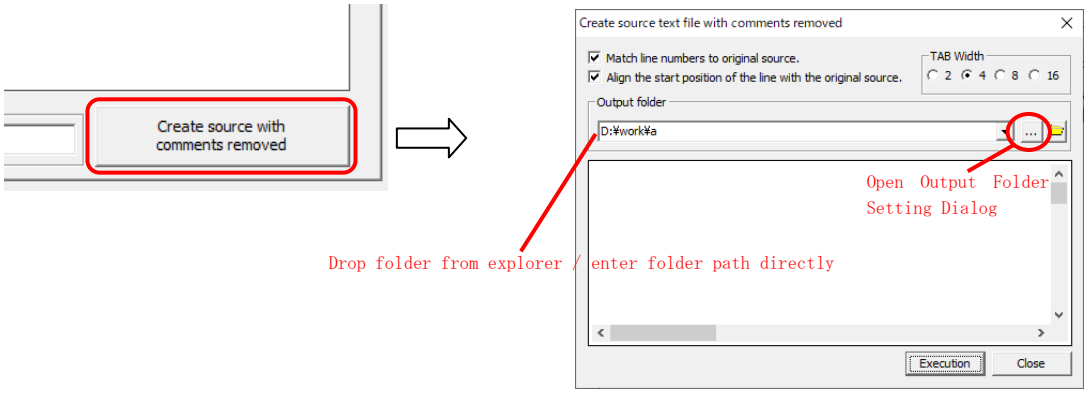
The relative path is relative to the folder set in "base folder".

Conversion example to relative path



3.3. Creating source text without comments

When you press the "Create source with comments removed" button at the bottom right of the source file setting screen, the following screen is displayed. You can create source file text with comments removed and words restructured.



Set the output folder and click the "Execution" button to create source text in the output folder with the comments removed from the source files set in the "Source files" tab.

If you check "Match line numbers to original source", line feed will be inserted and the line numbers will match the original source.

If you check "Align the start position of the line with the original source", the first column position of the line will match the original source.

For Exsample

Original source

```
1 : //-------------------------------------------//
2 : //  Window Procedure                         //
3 : //-------------------------------------------//
4 : AJC_WNDPROC(Back,  WM_CREATE     )
5 : {
6 :     PWRKLISTBOX     pW;
7 :     BOOL            rc = -1;
8 :     RECT            r;
9 :     int             sty,  exs;
10 :
11 :    do {
12 :        //----- Generate work space --------//
13 :        if ((pW = (PWRKLISTBOX)malloc(100) == NULL)
14 :            break;
15 :        memset(pW,  0,  100);
16 :        //----- Set my instance ID --------//
17 :        pW->InstID = INST_ID;
18 :    } while(0);
19 : }
```

Match line numbers to original source.
Align the start position of the line with the original source.

```
1 : AJC_WNDPROC(Back,WM_CREATE)
2 : {
3 : PWRKLISTBOX pW;
4 : BOOL rc=-1;
5 : RECT r;
6 : int sty,exs;
7 : do{
8 : if((pW=(PWRKLISTBOX)malloc(100)==NULL)
9 : break;
10 : memset(pW,0,100);
11 : pW->InstID=INST_ID;
12 : }while(0);
13 : }
```

Match line numbers to original source.
Align the start position of the line with the original source.

```
1 :
2 :
3 :
4 : AJC_WNDPROC(Back,WM_CREATE)
5 : {
6 : PWRKLISTBOX pW;
7 : BOOL rc=-1;
8 : RECT r;
9 : int sty,exs;
10 :
11 : do{
12 :
13 : if((pW=(PWRKLISTBOX)malloc(100)==NULL)
14 : break;
15 : memset(pW,0,100);
16 :
17 : pW->InstID=INST_ID;
18 : }while(0);
19 : }
```

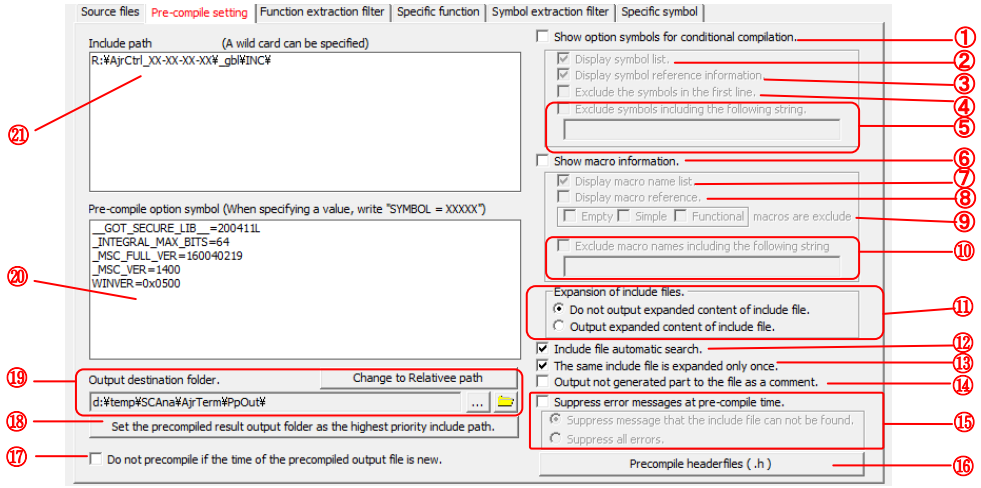
Match line numbers to original source.
Align the start position of the line with the original source.

```
1 :
2 :
3 :
4 : AJC_WNDPROC(Back,WM_CREATE)
5 : {
6 :     PWRKLISTBOX pW;
7 :     BOOL rc=-1;
8 :     RECT r;
9 :     int sty,exs;
10 :
11 :    do{
12 :
13 :        if((pW=(PWRKLISTBOX)malloc(100)==NULL)
14 :            break;
15 :        memset(pW,0,100);
16 :
17 :        pW->InstID=INST_ID;
18 :    }while(0);
19 : }
```

* By removing the comments and reconstructing the token (place a single blank only where white space is needed), you can avoid inconsistencies due to differences in the number of comments and blanks in the SourceCompare tool.

4. PRE-COMPILE SETTING AND OPTION-SYMBOL / MACRO INFORMATION

When you select the "precompile setting" tab in the main window, the following screen will be displayed.



Here, we will make precompile related settings.

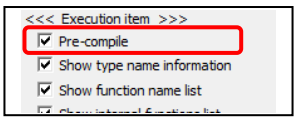
（Precompilation means resolving preprocessing statements such as "# include", "# define" and "#if".）

The setting contents are as follows。

#	Content
①	The symbols used in condition compilation (#if, #elif, #ifdef, #ifndef) are displayed. In case of unchecked, setting of ②～⑤ is invalid.
②	Display a list of symbol names.
③	The symbol and its reference point (file, line number) are displayed.
④	Exclude the symbols (#if, #elif, #ifdef, #ifndef) used in conditional compilation of the first line of the file.
⑤	Exclude symbols that contain a specific string. To specify more than one specific character string, separate them with a semicolon (;).
⑥	Displays a list of macros and references. In case of unchecked, setting of ⑦～⑩ is invalid.
⑦	Display a list of macro names.
⑧	It displays the macro and its reference point (file, line number).
⑨	Specify macros to exclude from display. ・Empty：Exclude empty macros (macros without contents only with macro name, (ex. "#Define MACNAME") ・Simple：Exclude simple macros (macros without arguments, ex "#define MACNAME 123"). ・Functional：Exclude functional macros (macro with arguments, ex '#define MACNAME (ARG, ...) ...').
⑩	Exclude macro names containing specific strings. To specify more than one specific character string, separate them with a semicolon (;).
⑪	Specify whether to include the contents of the include file in the output source file of the precompile result. When "Output expanded content of include file" is selected, include the contents of included file.
⑫	Automatically search include files from “Include pathes” and "base folder" including subfolders.
⑬	Do not read the same include file that has already been expanded.
⑭	A false condition is generated by conditional compilation, and the part that is not generated is also output to the file as a comment. (Add "//- " to the beginning of the line)
⑮	Suppresses precompile error messages. Choose whether to "suppress messages that include files can not be found" or "suppress all errors".

#	Content
⑯	Precompile header file (.h file)
⑰	If checked, check the file timestamps and do not precompile if the precompiled output file is newer than the source file. ※ Since the include file is not read by #include, the information such as macro name and type name defined in the include file is not output.
⑱	Set the precompile output destination folder as an include path. This include path has the highest priority and takes precedence over includes (#include "XXX.h") in the local folder.
⑲	Specify the output destination of the precompiled file. Precompile output files are created with the same file name in the same folder. (It does not create subfolders)
⑳	Specify an option symbol for precompilation. It is set from right-click popup menu. To set a value for a symbol, specify it in the format of "S*YMBOL = value*".
㉑	Specify an include path by one of the following methods. ・Drop a folder on this part. (If you hold down the Ctrl key while dropping it, it will be converted to a relative path) ・Set from right-click popup menu. If the check box (⑫) (include file automatic search) is checked, a wildcard can be specified as an include path. (Ex. "c:Program Files¥Microsoft*.*")

To precompile, check "Precompile" in the main window.

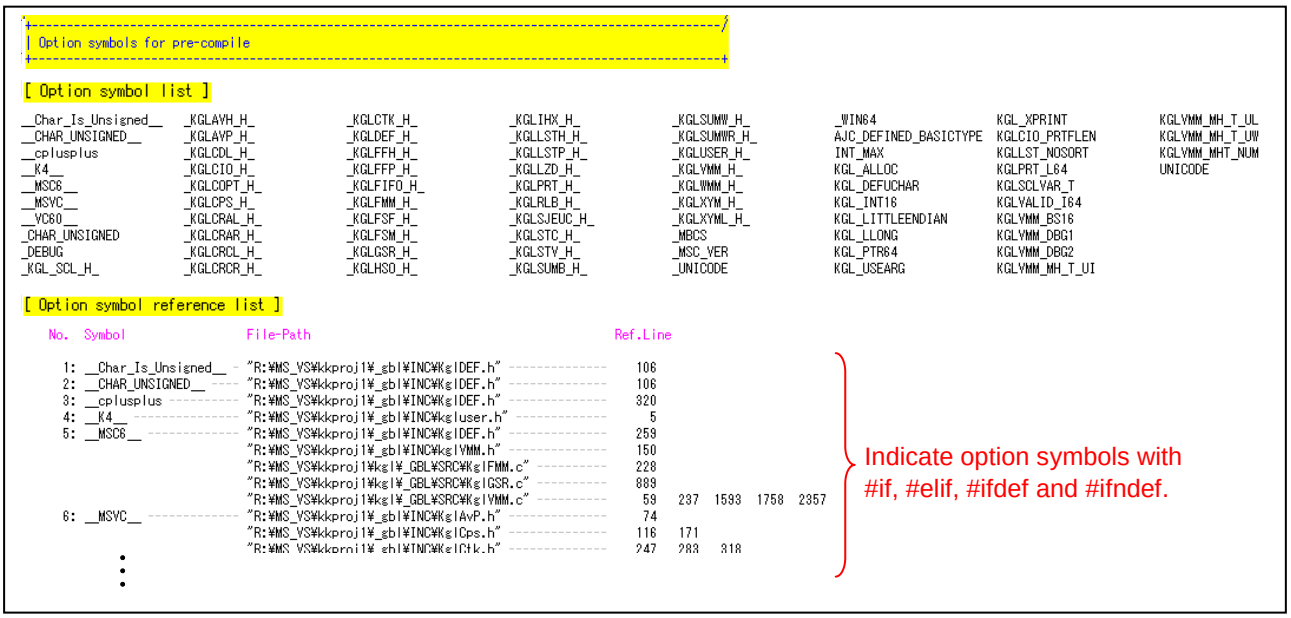


When "Check option symbol for condition compilation" is checked, "#if" "# elif" "# ifdef" "# ifdef" Display the referenced option symbol.

When "Display symbol list" is checked, a list of option symbol names is displayed.

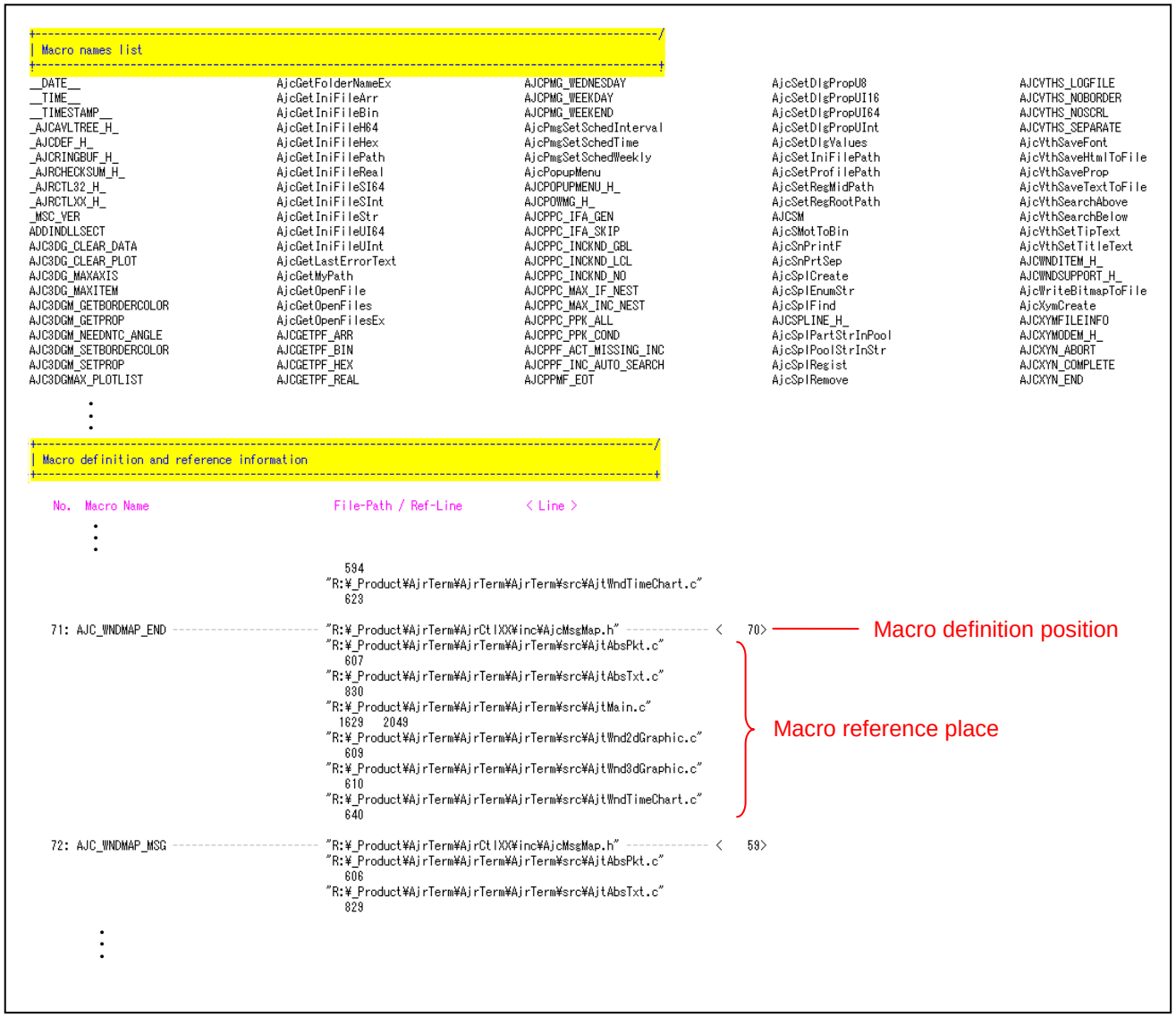
When "Display symbol reference information" is checked, the reference position (file name, line number) of the option symbol is listed.

An example display of option symbol information is shown below.



When "Display macro information" is checked, information on macros detected at precompile time is displayed.
When "Display macro name list" is checked, a list of macro name names is displayed.
When "Display reference information of macro" is checked, the macro definition / reference place (file name, line number) is listed up.

An example of displaying macro information is shown below.

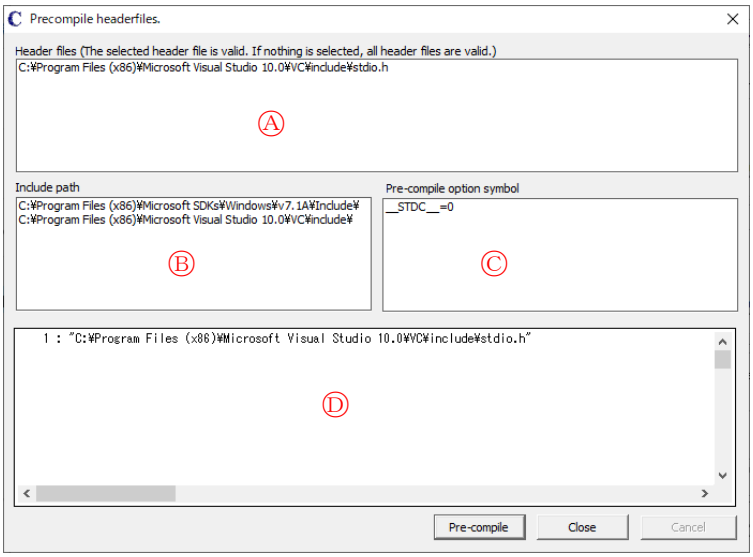


4.1. Header file precompilation

If the header file is huge and this header file is commonly included, it will take time to precompile the source files.

By precompiling the header file in advance, the precompile time of the source file can be shortened.
Follow the steps below to precompile the header file.

1) Press "Header (.h) file compilation" in ⑯ to display the following screen.



2) Make the following settings.
- Set the header file to be precompiled in part Ⓐ.(You can drop the header file)
- Set the include path required for precompilation in part Ⓑ. (You can drop the folder)
- In the part Ⓒ, specify the compile options required for precompilation. (Right-click to display the edit menu)

3) Click the “Precompile” button to execute precompilation.（If there is an error, it will be displayed in red in part Ⓓ）

4) Click the “Close” button to close the precompile screen of the header file.

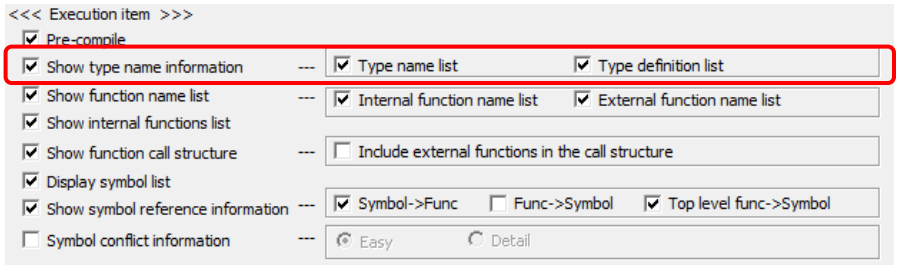
5) Press ⑯'s "Set the precompile result output folder as the highest priority include path" button, and ⑰ precompile output destination.

Set the folder as the highest priority include path.（The folder is added to the "include path"）

* Even if you manually add the ⑰ precompile output destination to the include path, it will not be the highest priority include path.

5. TYPE NAME INFORMATION

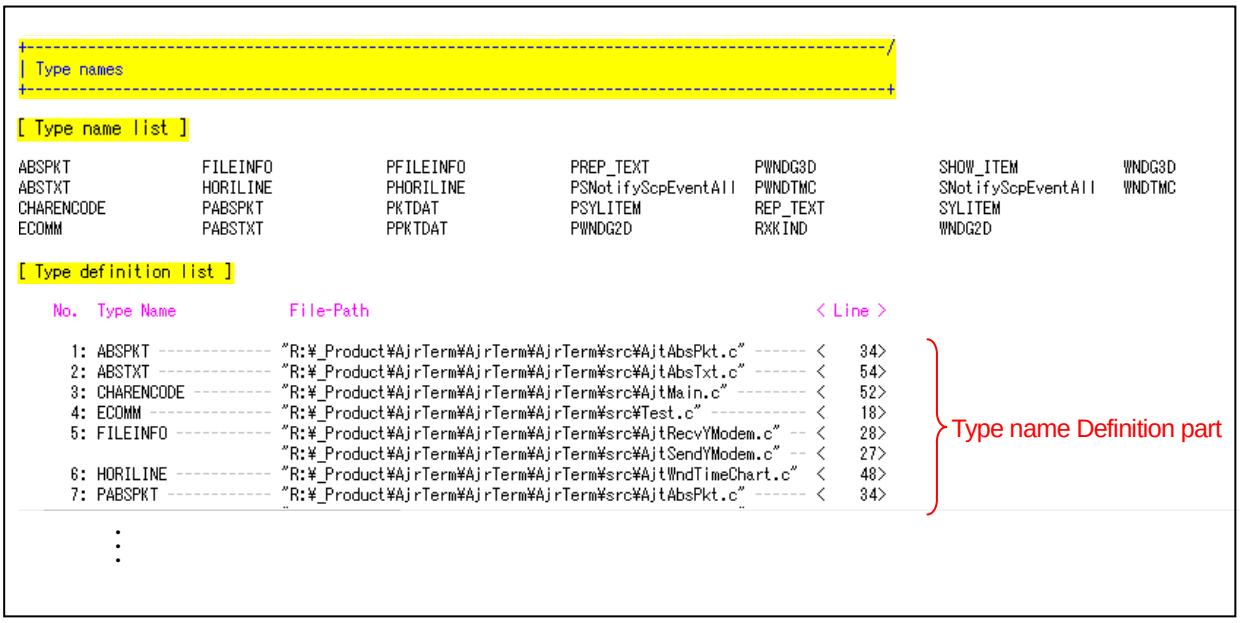
If you check "Show type name information" in the main window, information on the type name will be displayed.
When "Type name list" is checked, a list of type names is displayed.
When "Type definition list" is checked, the definition position of the type name is displayed.



Type names also include structure name (struct), union name (union), enumeration name (enum) in addition to the name defined by "typedef".

When you execute precompile, extract the precompile output source and the type name defined in the include file.
If you do not want to precompile, extract the type name defined in the source program file.

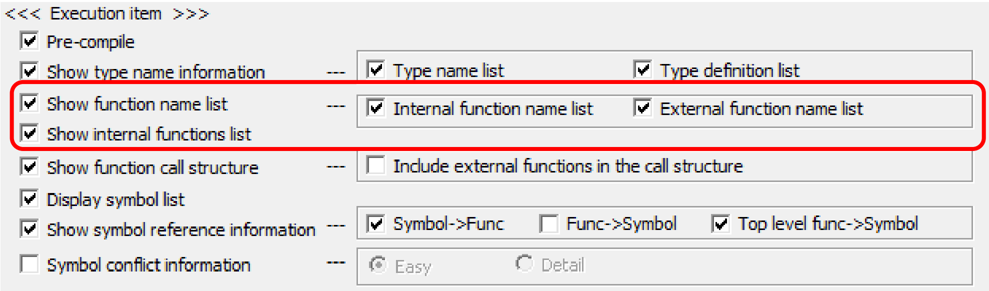
An example display of type name information is shown below.



6. FUNCTION NAME INFORMATION

If you check "Display a list of function names" in the main window, a list of function names will be displayed. You can select "Internal function name list" and "External function name list" for function name list display. (You can also select 2)

When "Show internal function list" is checked, the definition position of the internal function, the number of instructions in the function, the number of keywords such as if and while, etc. are tabulated and displayed.

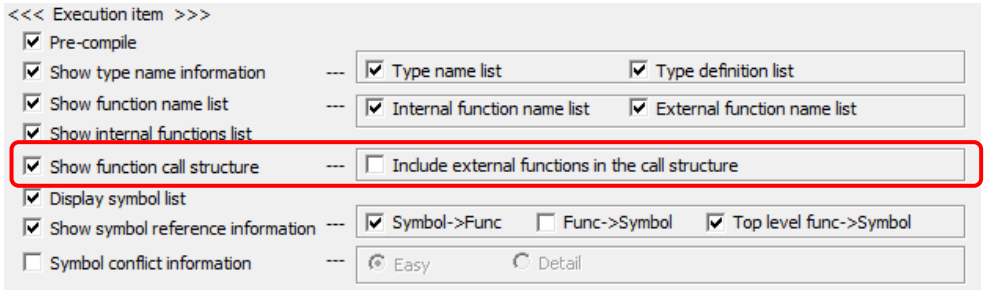


An example display of function name information is shown below.

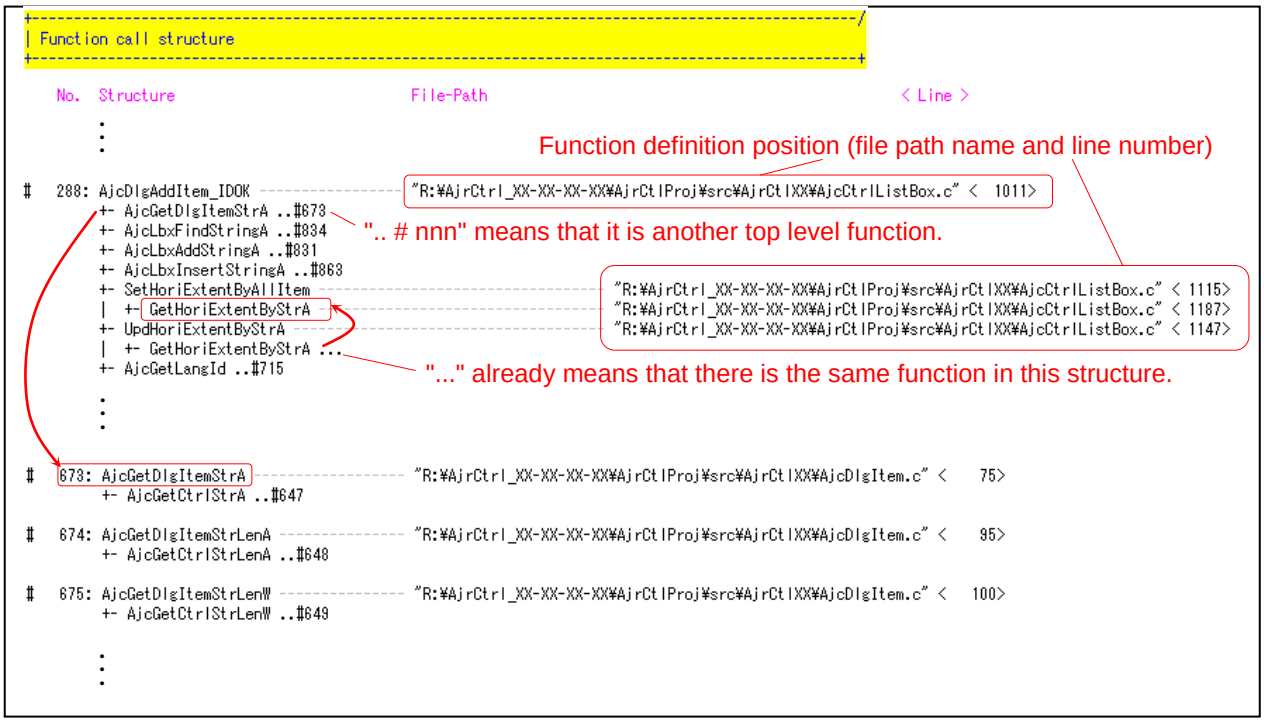
```
+--------------------------------------------------------------------------------------/
| Func names
+--------------------------------------------------------------------------------------+

[ Internal function name list ]
AJC_DLGPROC(FileInput,IDCANCEL)        AJC_DLGPROC(MyDlg,WM_SCP_TXEMPTY)     AJC_WNDPROC(Back,ID_WND_3DPLOT)        FuncRecvXModemIsOpened
AJC_DLGPROC(FileInput,IDOK)            AJC_DLGPROC(MyDlg,WM_SIZE)            AJC_WNDPROC(Back,ID_WND_MACRO)         FuncRecvYModem
AJC_DLGPROC(FileInput,WM_INITDIALOG)   AJC_DLGPROC(MyDlg,WM_TIMER)           AJC_WNDPROC(Back,ID_WND_TIMECHART)     FuncRecvYModemIsBusy
AJC_DLGPROC(FileInput,WM_TIMER)        AJC_DLGPROC(Relay,IDC_CMD_CREATE)     AJC_WNDPROC(Back,IDC_VT100)            FuncRecvYModemIsOpened
AJC_DLGPROC(Main,IDC_CMD_CONNECT)      AJC_DLGPROC(Relay,IDC_CMD_OPEN)       AJC_WNDPROC(Back,WM_CREATE)            FuncSendBinary
AJC_DLGPROC(Main,IDC_CMD_EASY)         AJC_DLGPROC(Relay,IDC_CMD_SETPORT)    AJC_WNDPROC(Back,WM_DESTROY)           FuncSendBinaryIsBusy
AJC_DLGPROC(Main,IDC_CMD_OPEN)         AJC_DLGPROC(Relay,IDCANCEL)           AJC_WNDPROC(Back,WM_DEVICECHANGE)      FuncSendFile
AJC_DLGPROC(Main,IDC_CMD_SEND)         AJC_DLGPROC(Relay,IDOK)               AJC_WNDPROC(Back,WM_ENDSESSION)        FuncSendFileIsBusy
          :

[ External function name list ]
__max          AjcGetDlgItemCboIx    AjcRngPurge          AjcSin                  CreateFile            MAjcSetDlgPropChk
__min          AjcGetDlgItemChk      AjcRngPutData        AjcSnPrintF             CreateMutex           MAjcSetDlgPropHex
_findclose     AjcGetDlgItemH64      AjcSaveDlgProfiles   AjcStrCnvRevPath        CreatePen             MAjcSetDlgPropReal
_findfirst     AjcGetDlgItemHex      AjcSaveWndPos        AjcStrRetRevPath        CreatePopupMenu       MAjcSetDlgPropStr
_findnext      AjcGetDlgItemReal     AjcSaveWndRect       AjcSysTimeToTime1970    CreateSolidBrush      MAjcSetDlgPropUInt
_ismbblead     AjcGetDlgItemSI64     AjcSbcComboBox       AjcTchEnableHLine       CreateStatusWindow    MAjcSetWindowLong
_ismbbtrail    AjcGetDlgItemSInt     AjcSbcLoadItems      AjcTchGetDroppedFile    CreateWindowEx        MAjcStrCat
_makepath      AjcGetDlgItemStr      AjcSbcRadioBtns      AjcTchGetProp           DeleteDC              MAjcStrCmp
_splitpath     AjcGetDlgItemStrLen   AjcSbcSaveItems      AjcTchLoadPropEx        DeleteFile            MAjcStrCpy
_stati64       AjcGetDlgItemUI64     AjcScpClose          AjcTchPurge             DeleteObject          MAjcStrLen
          :

+------------------------------------------------------------------------------------------/
| Function definition information list
+------------------------------------------------------------------------------------------+
    No.  Func.name                      File-Path                                                   < Line >  Stmt   if  else  while  for  switch  case

#    1: AjcDlgFileInput_IDCANCEL ------ "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWnd2dGraphic.c" < 1128>    5    1    1     0     0     0      0
#    2:    〃                           "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWnd3dGraphic.c" < 1113>    5    1    1     0     0     0      0
#    3:    〃                           "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWndTimeChart.c" < 1325>    5    1    1     0     0     0      0
#    4: AjcDlgFileInput_IDOK ---------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWnd2dGraphic.c" < 1085>   21    3    2     1     0     0      0
#    5:    〃                           "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWnd3dGraphic.c" < 1070>   21    3    2     1     0     0      0
#    6:    〃                           "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWndTimeChart.c" < 1282>   21    3    2     1     0     0      0
#    7: AjcDlgFileInput_WM_INITDIALOG - "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWnd2dGraphic.c" < 1030>   13    0    0     0     0     0      0
#    8:    〃                           "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWnd3dGraphic.c" < 1015>   13    0    0     0     0     0      0
#    9:    〃                           "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWndTimeChart.c" < 1227>   13    0    0     0     0     0      0
#   10: AjcDlgFileInput_WM_TIMER ------ "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWnd2dGraphic.c" < 1064>   11    1    1     0     0     0      0
#   11:    〃                           "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWnd3dGraphic.c" < 1049>   11    1    1     0     0     0      0
#   12:    〃                           "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWndTimeChart.c" < 1261>   11    1    1     0     0     0      0
#   13: AjcDlgFileInputProc ----------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWnd2dGraphic.c" < 1142>    3    1    0     0     0     0      0
#   14:    〃                           "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWnd3dGraphic.c" < 1127>    3    1    0     0     0     0      0
#   15:    〃                           "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWndTimeChart.c" < 1339>    3    1    0     0     0     0      0
#   16: AjcDlgMain_IDC_CMD_CONNECT ---- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" ------- < 1994>    3    2    1     0     0     0      0
#   17: AjcDlgMain_IDC_CMD_EASY ------- "R:¥_Product¥AjrTerm¥AjrTerm¥src¥Test.c" ------------------- <  210>    2    1    0     0     0     0      0
          :
```

7. FUNCTION CALL STRUCTURE

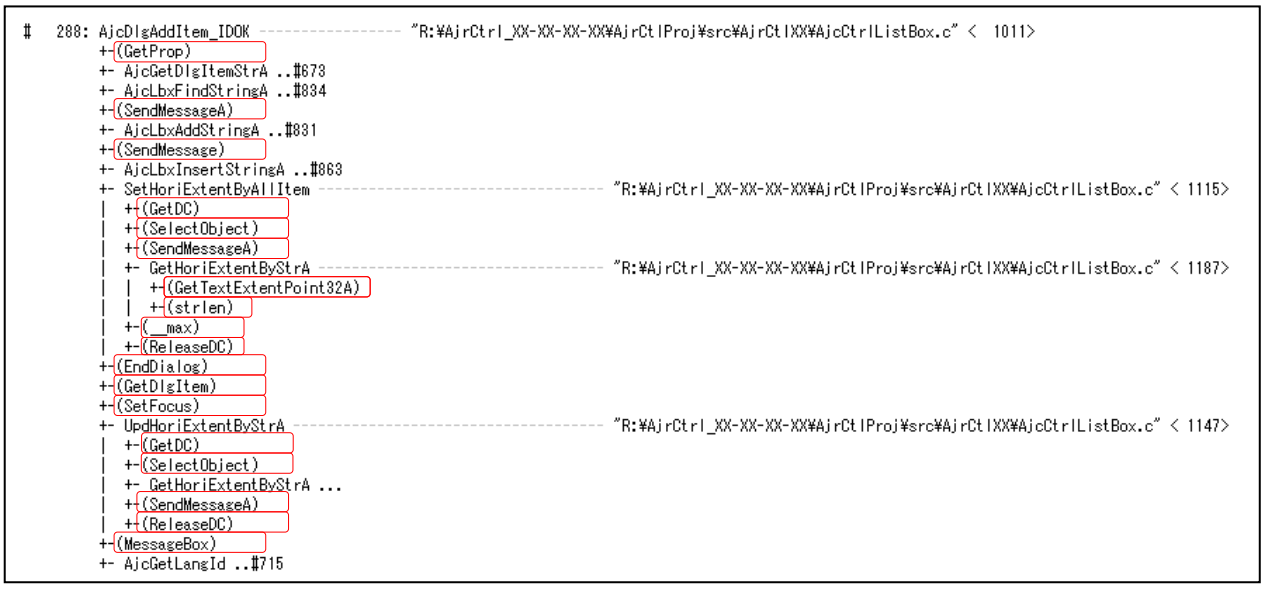
By checking "Show function call structure" in the main window, analyze and display function call structure.
If "Include external functions in the call structure" is checked, include external functions in function call structure.



An example display of the function call structure is shown below.

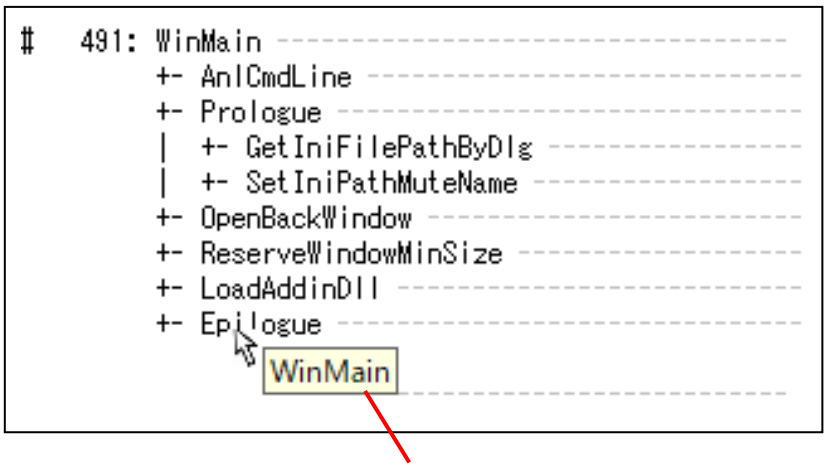


When external functions are included in the calling structure, external functions are displayed in parentheses.



7.1. Tooltip display of top function name

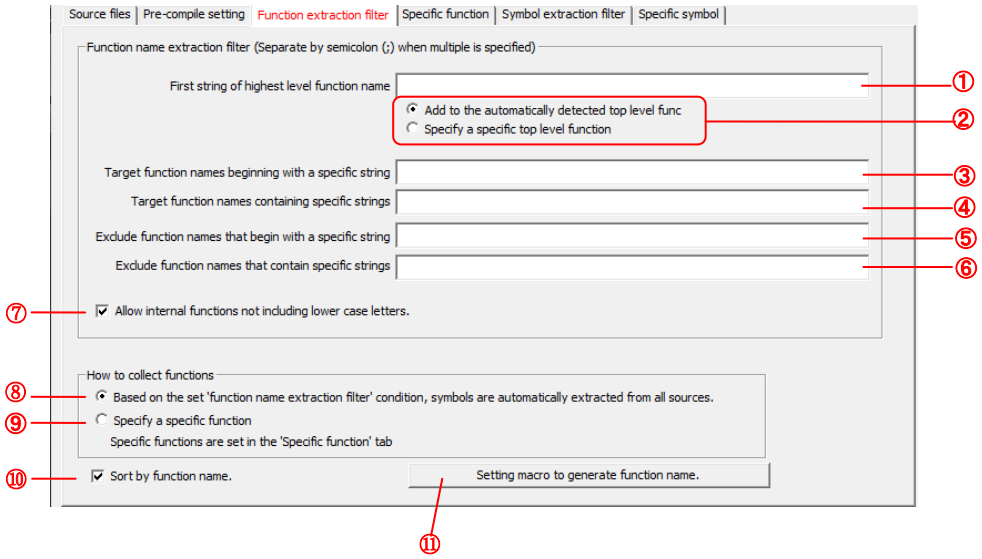
When the cursor is placed on a function name, the top-level function name of the subfunction is displayed as tooltip text. （If the function is k-linked in multiple call trees, multiple top-level functions are displayed.）



Display the top-level function name “WinMain” of Epilogue() with tooltip.

8. FUNCTION NAME EXTRACTION FILTER

When you select the "Function Name Extraction Filter" tab of the main window, the following windows are displayed and you can limit the name of the function to be collected.



Here, specify the condition of the function name to be collected.

The setting contents are as follows.

#	Content
①	Treat function names beginning with the specified character string as top level functions. (* 1)
②	Specifies a specific top level function group. ・Add to the automatically detected top level function Add the function group specified by ① to the automatically detected top level function. ・Specify a specific top level function We create a function call structure for only the function group specified by ①.
③	Collect function names starting with the specified character string.
④	Collects the function name including the specified character string.
⑤	Excludes function names beginning with the specified string.
⑥	Excludes the function name including the specified character string.
⑦	Include function names that do not contain lowercase letters ('a' - 'z') in the collection target. If unchecked, function names that do not contain lowercase letters ('a' to 'z') are excluded.
⑧	Activate the extraction filter condition of the function set in ①～⑦.
⑨	Invalidate the extraction filter condition of the function set in ①～⑦, and specify the extraction condition of the function with "Specific function ".
⑩	Sort in ascending order of function name and output "internal function list" and "function call structure".
⑪	Setting macro to generate function name (This will be explained shortly).

To specify multiple strings in item ①, ③～⑥, separate them with a semicolon (;) and specify them.
Spaces before and after the specified character string are ignored.

* 1: Functions that have not been called from any source program are automatically recognized as top level functions. "Function call structure" outputs the call structure in under of the top-level function, but by specifying another top-level function, the call structure is also output for that function.

Setting macro to generate function name

Suppose you have a function macro (AJC_WNDPROC) that generates a function name as follows.

```
#define AJC_WNDPROC(NAME, MSG) static LRESULT AjcWnd##NAME##_##MSG(HWND hwnd, UI msg, WPARAM wParam, LPARAM lParam)

AJC_WNDPROC(Main, WM_CREATE)  ----- ①
{
        . . . . .
}

AJC_WNDPROC(Main, WM_SIZE)    ----- ②
{
        . . . . .
}
```

The actual function name is generated by the AJC_WNDPROC () macro, and the actual function is ...

①= 'static LRESULT AjcWndMain_WM_CREATE(HWND hwnd,UI msg,WPARAM wParam,LPARAM lParam) '

②= ' static LRESULT AjcWndMain_WM_SIZE(HWND hwnd,UI msg,WPARAM wParam,LPARAM lParam) '

will be expanded .

If you do not precompile (or if a valid header file folder is not specified), the AJC_WNDPROC () macro will not be expanded and will interpret "AJC_WNDPROC" as a function name.

If there is the same function definition at several places like this, it will not be recognized even if you look at the reference.

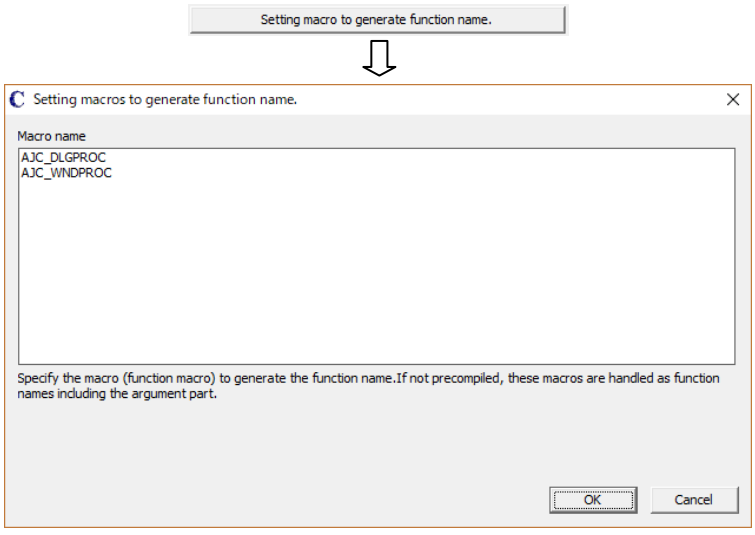
In such a case, it can be set to treat it as a function name including the argument part.

①= ' AJC_WNDPROC(Main,WM_CREATE) '

②= 'AJC_WNDPROC(Main,WM_SIZE) '

as a function name.

To set it to treat it as a function name including the argument part, set the macro name with "Setting macro to generate function name" button.



In the list box, set a macro to generate a function name like "AJC_WNDPROC" above.

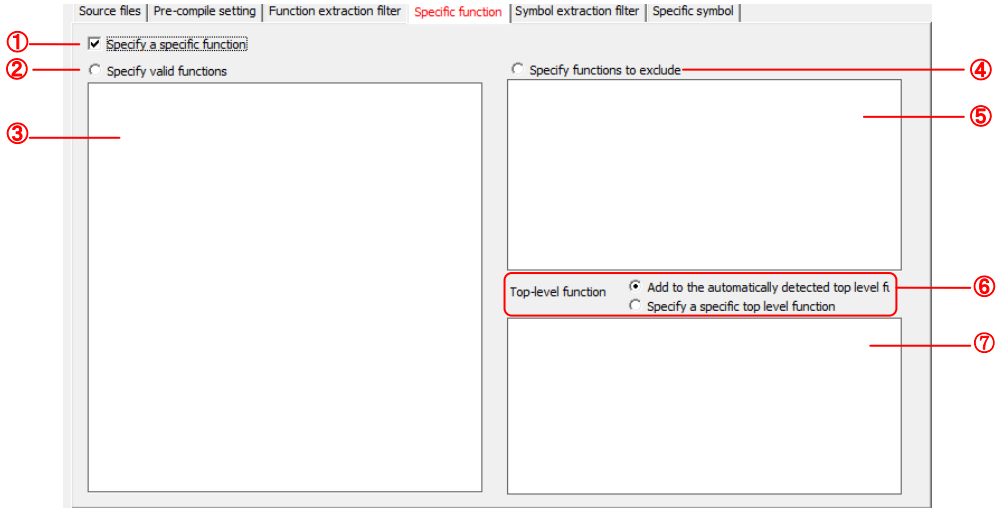
Note that "AJC_DLGPROC" and "AJC_WNDPROC" are set by default, so delete them if they are not convenient.

However, when precompiling and expanding these macros, it becomes the (original) function name generated by the macro, so it is not necessary to set the macro name.

8.1. SPECIFICATION OF SPECIFIC FUNCTION NAME

You can individually specify functions to be collected and top-level functions on the “Specific function” tab of the main window.

The setting on the "Function Name Extraction Filter" tab restricts the name of the function to be collected roughly, whereas on the "Specify Specific Function Name" tab, specify the name of each function directly.



The setting contents are as follows.

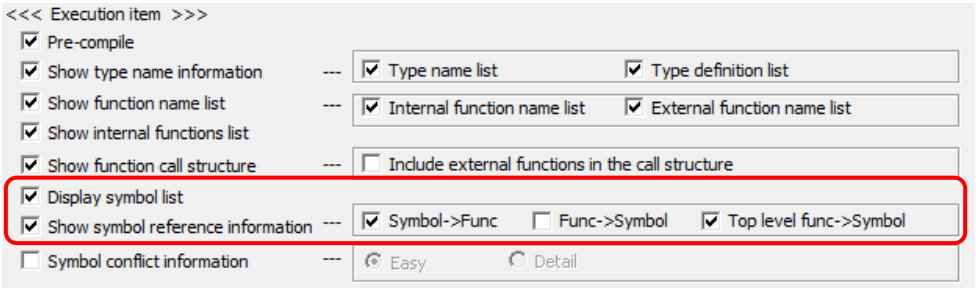
#	Content
①	Check to specify a specific function name. （「Similar to checking "Specify a specific function " on the "Function extraction filter" tab）
②	Only specific function names are targeted. （Create function calling structure and cross-references of symbols only with the function group specified by ③）
③	When ② is checked, a specific function group to be targeted is enumerated. If ② is checked and nothing is specified, it becomes "no target function".
④	Exclude specific functions. （Exclude the function group specified by ⑤ from the function calling structure and cross-reference of symbols）
⑤	When ④ is checked, a specific function group to be excluded is enumerated. If ④ is checked and nothing is specified, all functions are covered.
⑥	Specifies a specific top level function group. ・Add to the automatically detected top level function Add the function group specified by ⑦ to the automatically detected top level function. ・Specify a specific top level function We create a function call structure for only the function group specified by ⑦.
⑦	Enumerate the specific top level functions in ⑥.

Only one of ② and ④ can be set.

⑦ specifies the name of the function to treat as the top level function.

9. SYMBOL CROSS REFERENCE INFORMATION

By checking "Display symbol list" in the main window, a list of collected symbols will be displayed.
When "Show symbol reference information" is checked, cross-reference information of symbols is displayed.

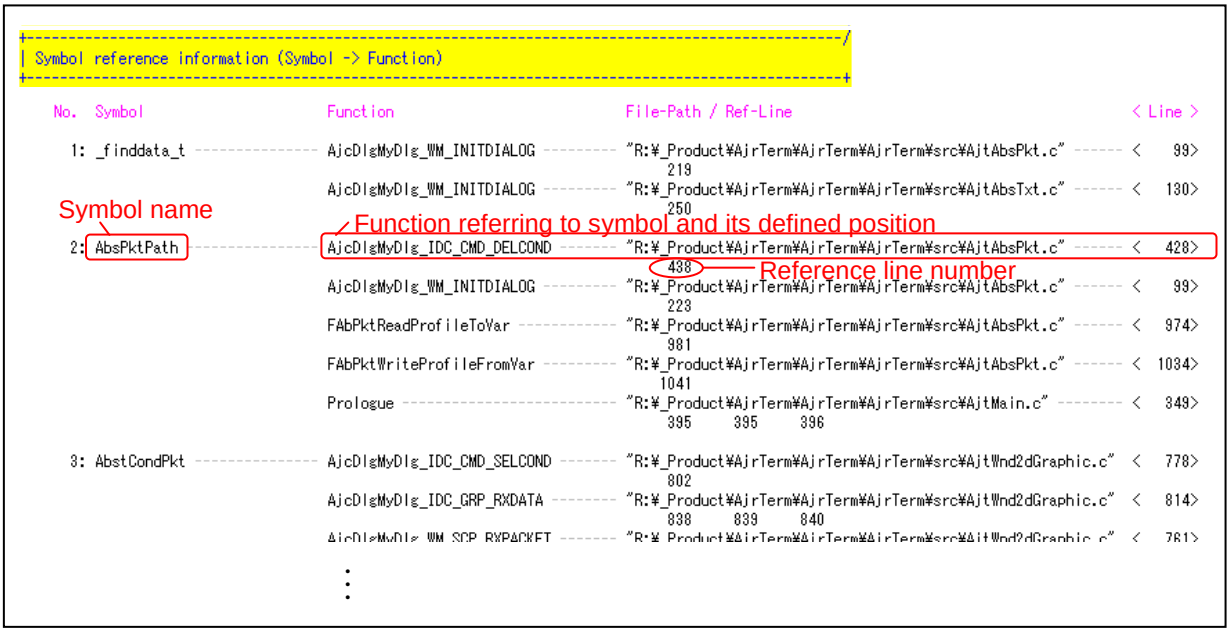


Symbol cross reference information has the following three patterns.

#	Item	Content
1	Symbol -> Func	Lists the symbols and indicates the function referring to that symbol.
2	Func -> Symbol	List the function and indicate the symbol referenced in the function.
3	Top level func -> Symbol	Lists the functions of the highest level and shows the symbols used in the function including the called subfunction. That is, it shows the symbol to use when calling the top-level function.

9.1. Symbol -> Func

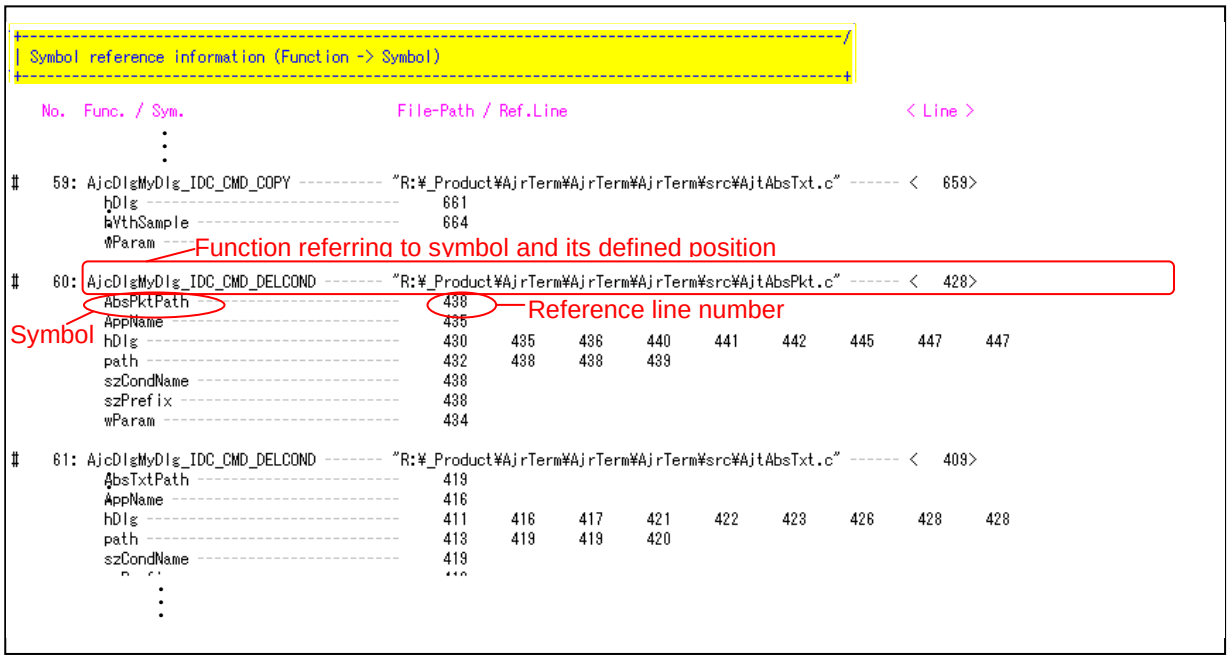
Lists the symbols and indicates the function referring to that symbol.
An example display is shown below.



9.2. Func -> Symbol

List the function and indicate the symbol referenced in the function.

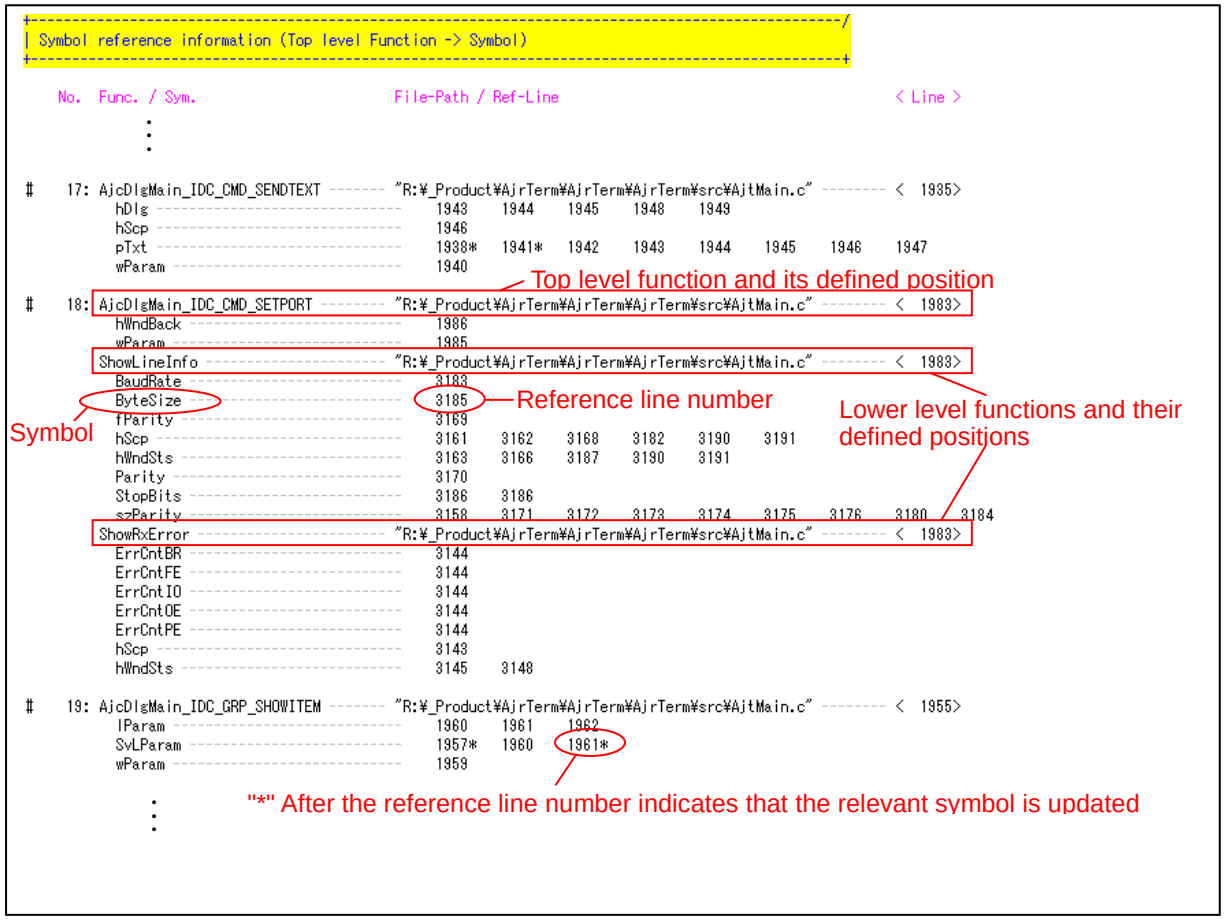
An example display is shown below.



9.3. Top Lvel Func -> Symbol

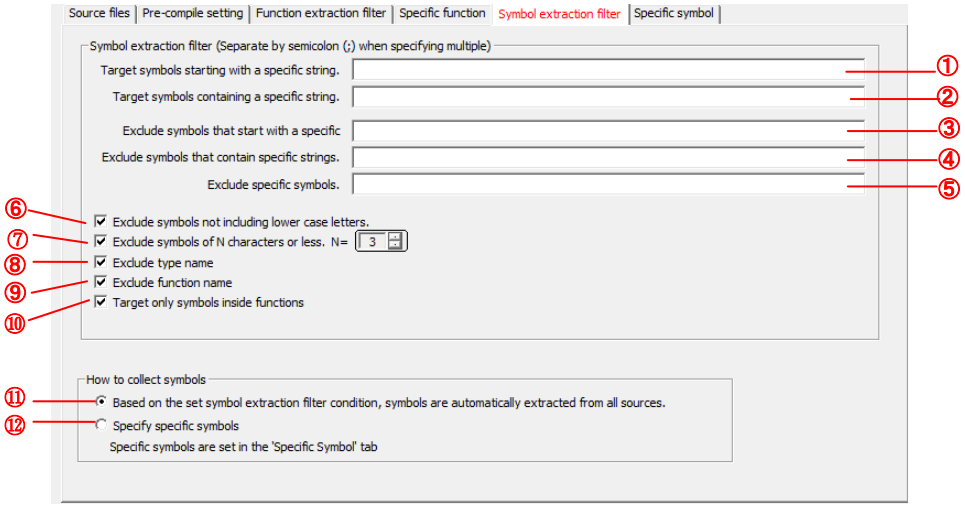
Lists the functions of the highest level and shows the symbols used in the function including the called subfunction. That is, it shows the symbol to use when calling the top-level function.

An example display is shown below.



10. SYMBOL EXTRACTION FILTER

When you select the "Symbol extraction filter" tab in the main window, the following screen will be displayed.



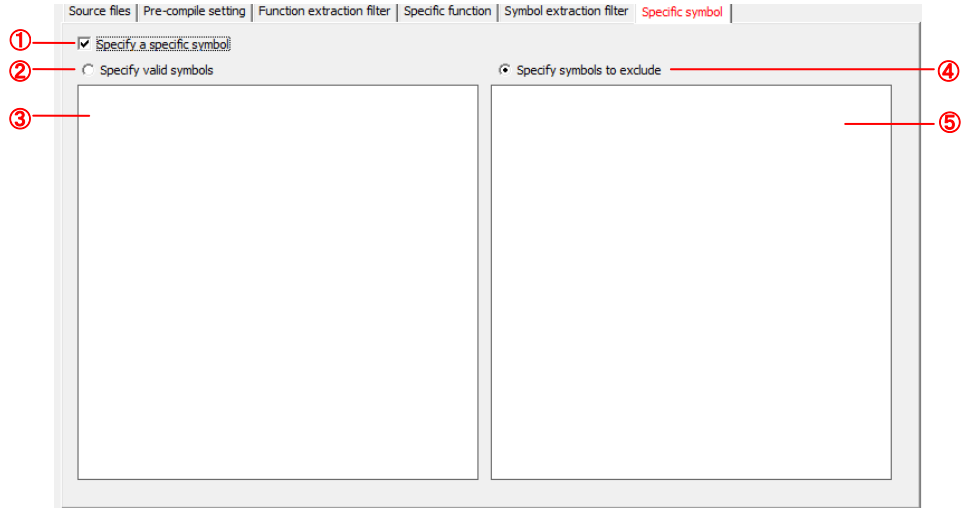
The setting contents are as follows.

#	Content
①	Collects symbols starting with the specified string.
②	Collects symbols containing the specified string.
③	Excludes symbols beginning with the specified string.
④	Exclude symbols containing the specified string.
⑤	Excludes the specified symbol group
⑥	Exclude symbols that do not contain lower case letters ('a' ~ 'z').
⑦	Exclude symbols with N characters or less.
⑧	Exclude type name.
⑨	Exclude the function name.
⑩	Only the symbol reference inside the function is targeted.
⑪	Extract symbols from all input source files according to the condition of ① ～ ⑩
⑫	Specify a specific symbol. Specific symbols are set by the "Specific Symbol Specification" tab.

To specify multiple strings in item ①～⑤, separate them with a semicolon (;) and specify them.
Spaces before and after the specified string are ignored.

10.1. SPECIFIC SYMBOL

When you select the "Specific symbol" tab in the main window, the following screen will be displayed.



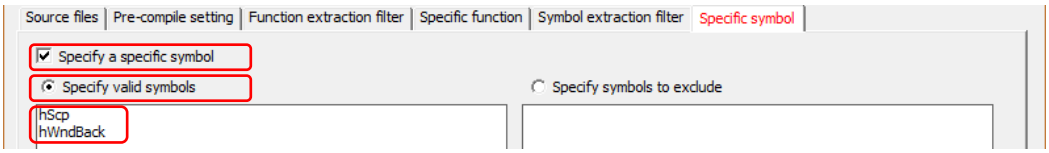
The setting contents are as follows.

#	Content
①	Check to specify a specific function name. （Same as checking "Specify specific symbol" on the "Symbol extraction filter" tab）
②	Target a specific symbol group. （Create cross-references of symbols only with the symbol group specified by ③）
③	When ② is checked, a specific symbol group to be targeted is enumerated. If ② is checked and nothing is specified, it becomes "no target symbol".
④	Exclude certain symbol groups. （Exclude the symbol group specified by ⑤ from cross reference of symbols）
⑤	When ④ is checked, a specific symbol group to be excluded is enumerated. If ④ is checked and nothing is specified, all symbols will be covered.

Only one of ② and ④ can be set.

By specifying a specific symbol, you can display only the reference information of the specified symbol in "Symbol cross reference".

For example, when displaying the reference information of the symbols "hScp" and "hWndBack" in all the subfunctions under the Top level function, check “Specify a specific symbol” and “Specify valid symbols”, and add "hScp" and "hWndBack" to ListBox



By pressing the "execution" button, we restricted it to the specified symbol. Output reference information of symbols.

```
+-------------------------------------------------------------------------------------------------/
| Symbol reference information (Top level Function -> Symbol)
+-------------------------------------------------------------------------------------------------+

    No.  Func. / Sym.                    File-Path / Ref-Line                              < Line >

          . . . . .
#    21: AjcDlgMain_IDC_CMD_SETPORT -------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" -------- <  2038>
          hScp ------------------------------    2041
          hWndBack --------------------------    2041
        ShowLineInfo ---------------------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" -------- <  3276>
          hScp ------------------------------    3282    3283    3289    3303    3311    3312
        ShowRxError ----------------------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" -------- <  3260>
          hScp ------------------------------    3264
           . . . . .
```

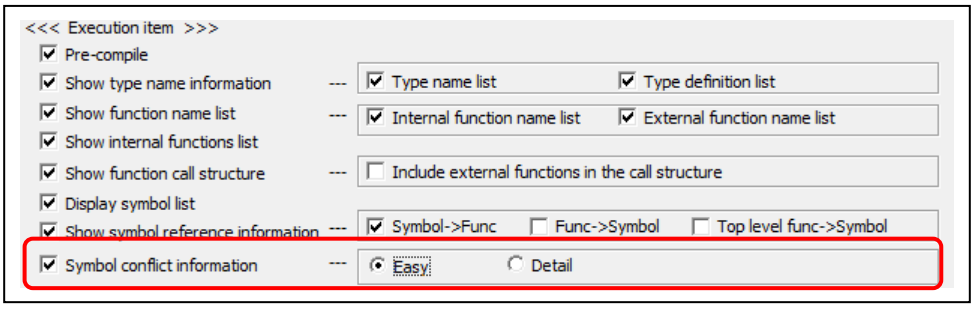
From the above output, you can see that "hScp" and "hWndBack" are referenced by the top level function "AjcDlgMain_IDC_CMD_SETPORT" , and "hScp" is referenced by its subfunction "ShowLineInfo" and “ShowRxError” .

The calling structure of the function "AjcDlgMain_IDC_CMD_SETPORT" is as follows.

```
+-------------------------------------------------------------------------------------------------/
| Function call structure
+-------------------------------------------------------------------------------------------------+

    No.  Structure                       File-Path                                        < Line >

          . . . . .

#    21: AjcDlgMain_IDC_CMD_SETPORT -------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" <  2038>
        +- ShowLineInfo ------------------------------------------ "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" < 3276>
        +- ShowRxError ------------------------------------------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" < 3260>

          . . . . .
```

11. SYMBOL COMFLICT INFORMATION

Checking "Show symbol conflict information" in the main window will list up the symbols referenced / updated in multiple top level function call structures under each top level function call structure.



For example, if you have the following function call structure, list the symbols referenced in multiple function structures under the four top-level functions "ist_ERI9", "ist_RXI9", "ist_TXI9" and "MainProc".

```
ist_ERI9
 +- AjrRingPutByte

 ist_RXI9
 +- AjrRingPutByte

 ist_TXI9
 +- AjrRingGetByte

 MainProc
 +- MainInit
 |  +- InitEepRom
 |  +- DebugInit
 |     +- AjrRingInit
 |     +- DebugSendPacket
 +- MainLoop
    +- DebugPeriod
       +- DebugPutS
       |  +- AjrRingPutData
       +- DebugPrintF
```

※ The top-level function means a function (interrupt handler, task, etc.) that has not been called from within the program.
However, functions that are dynamically called by function pointers and unused functions are also recognized as top-level functions.

In the example on the left, "MainProc" means main processing and "ist_XXXX" means an interrupt handler.

Display symbol conflict information (Easy)

```
+-------------------------------------------------------------------------------------------/
| Symbol conflict information (Symbol information conflicting between under the top level function group)
+-------------------------------------------------------------------------------------------+

  1 : RxRing   -    #46   MainProc  "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrMain.c" - <   212>
               -    #41   ist_ERI9  "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   540>
               -    #42   ist_RXI9  "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   555>

  2 : TxRing   -    #46   MainProc  "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrMain.c" - <   212>
               -    #43   ist_TXI9  "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   569>

  3 : fTxBusy  *4   #46   MainProc  "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrMain.c" - <   212>
               *1   #43   ist_TXI9  "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   569>
```

"fTxBusy" means that it is referenced from two function call structures "MainProc" and "ist_TXI9".
However, symbol update information is recognized only when updating symbols statically.
If you are updating dynamically with a pointer etc., you can not recognize symbol updates.
If the symbol has not been updated (or if the update can not be recognized), "-" is displayed.

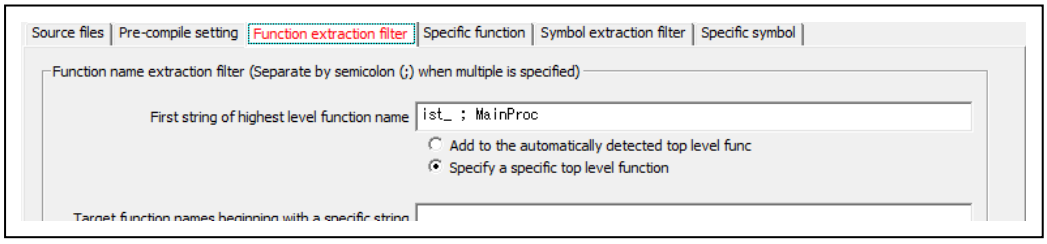
When "Detail " is checked, the function name (under the top level function) that references / updates the symbol is also displayed.

```
+-------------------------------------------------------------------------------------------/
| Symbol conflict information (Symbol information conflicting between under the top level function group)
+-------------------------------------------------------------------------------------------+

  1 : RxRing  -    #46   MainProc ------- "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrMain.c" - <   212>
              -          DebugInit ------ "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <    88>
              -          DebugPeriod ---- "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   125>
              -    #41   ist_ERI9 ------- "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   540>
              -    #42   ist_RXI9 ------- "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   555>

  2 : TxRing  -    #46   MainProc ------- "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrMain.c" - <   212>
              -          DebugInit ------ "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <    88>
              -          DebugPrintF ---- "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   340>
              -          DebugPutS ------ "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   317>
              -          DebugSendPacket  "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   239>
              -    #43   ist_TXI9 ------- "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   569>

  3 : fTxBusy -    #46   MainProc ------- "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrMain.c" - <   212>
              *1         DebugInit ------ "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <    88>
              *1         DebugPrintF ---- "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   340>
              *1         DebugPutS ------ "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   317>
              *1         DebugSendPacket  "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   239>
              *1   #43   ist_TXI9 ------- "D:¥work¥AjrFW¥AjrMaster¥Src¥AjrDebug.c" <   569>
```

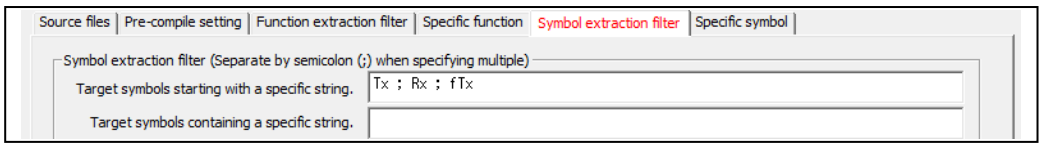
Indicates that "fTxBusy" is referenced / updated by the four functions "DebugInit", "DebugPrintF", "DebugPutS" and "DebugSendPacket" under "MainProc".

※ I think that it is necessary to set the function name and the symbol name so as to exclude unnecessary names in "function extraction filter", "specific function ", "symbol extraction filter" and "specific symbol".
In the output example shown above, the following settings are made.

Function extraction filter



Symbol extraction filter



13. EXECUTABLE WINDOWS VERSION

SCANA can run on 32-bit platform (x86) and 64-bit platform (x64) in all versions (Windows 7, Windows 8, Windows 8.1, Windows 10) for Windows 7 and later for PC.
Executable files are as follows.

Type	Execution file	Note
Windows（32Bit）	SCAna32.exe	
Windows（64Bit）	SCAna64.exe	Executable only on 64Bit Windows

14. DISCLAIMER

Please pay attention to the following points when using the program.

· The copyright of SCANA belongs to the creator.
・Please use it in the user's responsibility in any case.
We can not assume any responsibility for operation results.