Windows 用

C言語 静的解析ツール
SCANA

説明書

（ 1.6.0.1版 ）

－　目次　－

1. 概　要

このアプリケーション（以降ＳＣＡＮＡ(Static C-language ANAlizer)と言う）は、Ｃ言語ソースプログラムの静的解析ツールです。
既に、正常にビルドされているＣ言語プログラム群を対象とします。

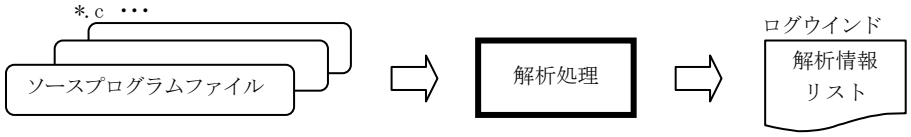
Ｃ言語ソースプログラム群を入力し、以下の解析処理を行います。

・プリコンパイル（プリプロセス文の解決 – 条件コンパイル(#if, #ifdef, #ifndef・・)やマクロ(#define)呼び出しの解決）
・プリコンパイル時に使用されているオプションシンボル(「#ifdef _DEBUG」の「_DEBUG」等)の一覧と、参照箇所のリストアップ
・プリコンパイル時に使用されているマクロ名の一覧と、参照箇所のリストアップ
・タイプ名（typedef 名，構造体名，共用体名，列挙名）とタイプ名定義箇所のリストアップ
・関数名一覧と、関数定義箇所のリストアップ
・関数の呼び出し構造の表示
・変数名等、シンボルの一覧と、参照箇所のリストアップ

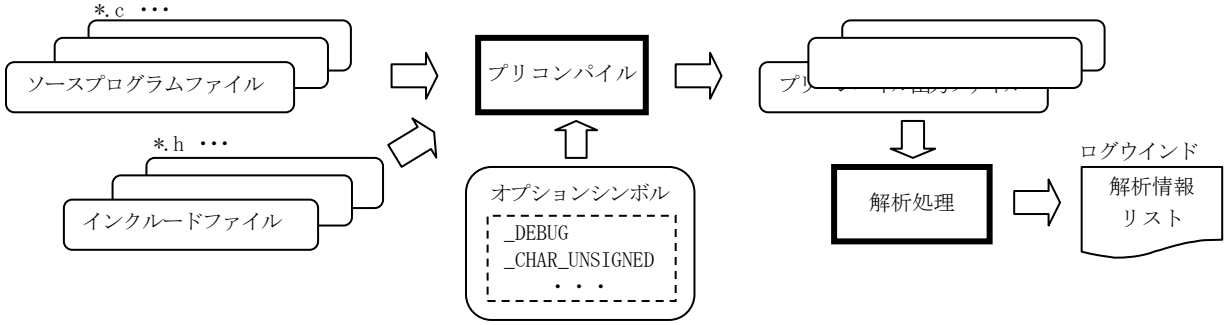
1.1. プリコンパイル

プリコンパイル（プリプロセス文の解決 – 条件コンパイル(#if, #ifdef, #ifndef・・)やマクロ(#define)呼び出しの解決）を行った場合は、プリコンパイル出力したソースファイルを入力して解析を行います。
プリコンパイルを行う場合は、適切なインクルードパスや、条件コンパイルのオプションシンボルを与える必要があります。

プリコンパイルしない場合の流れ



プリコンパイルする場合の流れ



ソースファイルやインクルードファイルに条件コンパイル(#ifdef, #ifndef, #if, #elif, #else, #endif)がある場合、条件が「真」となった部分のみ出力し、「偽」条件の部分は出力しません。（但し、#ifdef, #ifndef, #if, #elif, #else, #endif は偽条件中でも出力します）
#ifdef, #ifndef, #if, #elif, #else では、条件の演算結果を「TRUE」or「FALSE」で表示します。
#endif では、対応する #ifdef, #ifndef, #if の行番号を表示します。

（例）　ソースプログラム

```
 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 : }
```

プリコンパイル結果

```
/* "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; */     }
```

#if,#elif,#else の真偽値

対応する
#if の行番号

1.2. シンボル

変数名，関数名，タイプ名，マクロ名・・等の名称を総称して「シンボル」として扱います。
シンボルは、以下の条件の字句を意味します。

1）先頭文字はアルファベット，アンダバー(_) 、あるいは、ドルマーク($)　・・・(※１)
2）後続の文字はアルファベット，アンダバー(_)，ドルマーク($)、あるいは数字(0～9)
3）特例として、上記条件にピリオド(.)と数字(0～9)が続く場合もシンボルとする　・・・(※２)

※１：厳密なＣ言語のルールでは違反だが、処理系によってはドルマークを許容しているものもある。
※２：厳密なＣ言語のルールでは違反だが、処理系によっては「P2.1」や「P10.0」等のポート記述を許容している。

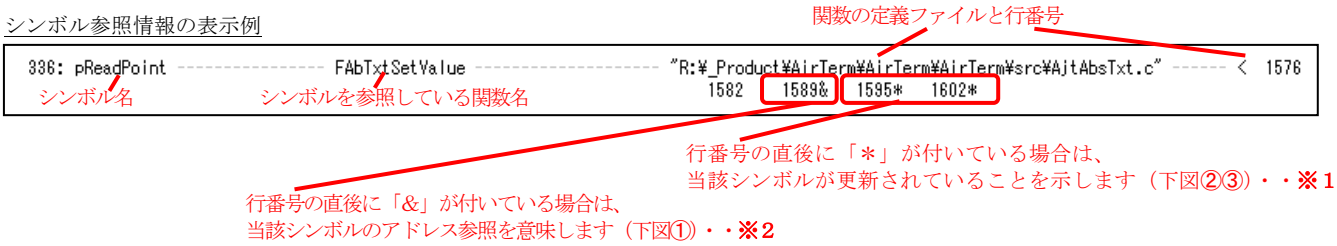
但し、配列変数は、「 arr[] 」のように '[]" を付加した文字列を変数として扱います。
C 言語の予約名（「if」「int」「static」や「while」・・等）は、シンボルとはみなしません。

1.3. シンボルの検出

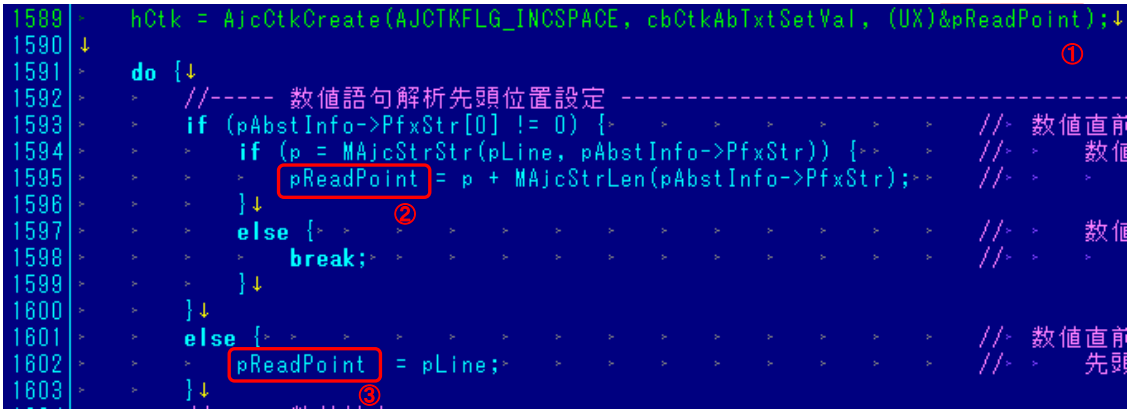
ソースプログラムからシンボルを検出し、シンボルの一覧や、シンボルの参照情報の表示を行います。
シンボルを抽出する際にフィルタを設定することもできます。(「シンボル抽出フィルタ」,「特定シンボルの指定」 を参照)

シンボルの参照箇所（ファイル名，行番号）の表示では、当該シンボル（変数）が更新されている場合は、行番号に「*」を付加して更新されていることを表示します。(**※１**)

シンボル参照情報の表示例



AjtAbsTxt.c



※１　シンボルが更新されていることを示す「＊」は、簡易的なものであり、以下の条件で検出します。

・シンボルの直前／直後の語句が「++」or「--」
・シンボルの直後の語句が「=」「+=」「-=」「*=」「/=」「%=」「&=」「|=」「^=」「>>=」　or「<<=」

シンボルが括弧で囲まれたり、構造体メンバである場合等は、シンボルの更新を認識できない場合があります。
以下の例では、シンボル(var)の更新を認識できません。

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

※２　シンボルがアドレス参照されていることを示す「&」は、簡易的なものであり、以下の条件で検出します。

・シンボルの直前が「&」
・シンボルの２つ前が「{ 」「, 」「(」 or 「)」

従って、アドレス参照以外でも、アドレス参照と見なされる場合があります。
以下の例では、アドレス参照でないにもかかわらず「&」が付加されます。

```
a = (b + c) & d;     //   「)」「&」「シンボル」の順で並んでいるので「 d 」はアドレス参照と見なされる
```

1.4. 関数定義の検出

指定されたソースプログラム群から、関数の定義を検出し抽出します。
語句が以下の順で並んでいる場合に、関数の定義を認識します。

シンボル	(	・・・	)	{	・・・	}

この時、「シンボル」を関数名として認識し、「{ 」から「 }」までを、関数の内部（関数本体）として認識します。
「・・・」は、任意の語句群を意味しますが、括弧（ ’(・・)’ や ‘{・・}’ ）がネストしている場合は、ネストが終了するまでを任意の語句群とみなします。

ソースプログラム中で検出した関数定義の関数名は、「内部関数」と言うことにします。
これに対し、ソースプログラム中で定義されていない関数を「外部関数」と言うことにします。
尚、関数型マクロ（引数付きのマクロ）も（プリコンパイルでマクロが展開されない場合は）外部関数として扱われます。

内部関数の定義を抽出する際にフィルタを設定することもできます。(「関数名抽出フィルタ」,「特定関数名の指定）を参照)

1.5. 関数呼び出しの検出

関数の内部（関数本体）中で、語句が以下の順で並んでいる場合に、関数の呼び出しとして認識します。

シンボル	(

この時、「シンボル」を呼び出し関数名として認識します。
関数マクロ（引数付きのマクロ）のマクロコールも関数の呼び出しとして認識します。
シンボルが内部関数名と一致する（つまり、ソースプログラム内で定義されている）場合、内部関数の呼び出しとして認識します。
シンボルが内部関数名と一致しない場合は、外部関数の呼び出しとして認識します。

＜注意＞
関数ポインタ等による動的な呼び出しの場合は、関数ポインタの変数名が関数名として認識されます。
以下のような場合は、「pFunc」が呼び出し関数名として認識されます。

```
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);
}
```

上記のような例の場合は、SubFunc()から TopFunc()をコールする際に、cbFunc()の関数ポインタを引数として渡しており、実際は TopFunc()から cbFunc()が呼び出されますが、このような動的な呼び出しは認識できません。

1.6. 最上位レベルの関数

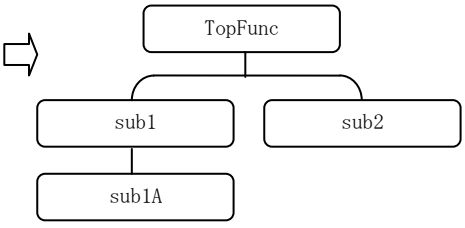
最上位レベルの関数とは、関数の呼び出し構造の頂点に位置する関数を意味します。
例えば、以下のようなサンプルソースの場合、「TopFunc()」が最上位レベルの関数となります。

サンプルソース

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

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

```
void  sub2(void)
{
    ・・・
}

void  sub1A(void)
{
    ・・・
}
```

関数の呼び出し構造



ソースプログラム中のいずれからも（静的に）呼び出されていない関数は自動的に「最上位レベルの関数」として認識しますが、特定の文字列で始まる関数を「最上位レベルの関数」として追加指定したり、個別に「最上位レベルの関数」を指定することもできます

1.7. 最上位レベル関数の指定

以下のサンプルプログラムは、コマンドラインの第１引数を判断し、cmdA()～cmdC()をコールします。

a1.c

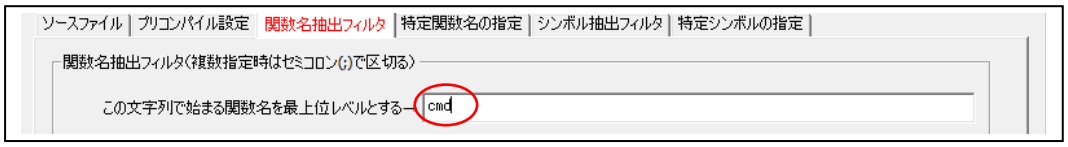
```
 1 : #include <string.h>
 2 :
 3 : static int cnt = 0;
 4 :
 5 : void sub1(void)
 6 : {
 7 : }
 8 :
 9 : void sub2(void)
10 : {
11 :     if (cnt++ < 10) {
12 :         sub2();
13 :     }
14 : }
15 :
16 : void cmdA(void)
17 : {
18 :     sub1();
19 : }
20 :
21 : void cmdB(void)
22 : {
23 :     cnt = 0;
24 :     sub2();
25 : }
26 :
27 : void cmdC(void)
28 : {
29 : }
30 :
31 : int main(int argc, char *argv[])
32 : {
33 :     if (argc >= 2) {
34 :         if      (strcmp(argv[1], "A") == 0) cmdA();
35 :         else if (strcmp(argv[1], "B") == 0) cmdB();
36 :         else if (strcmp(argv[1], "C") == 0) cmdC();
37 :     }
38 :     return 0;
39 : }
40 :
```

このサンプルプログラムでは、cmdA()～cmdC()を静的に呼び出しているので、(どこからもコールされていない)最上位関数は「main()」だけであり、関数の呼び出し構造は以下のように表示されます。

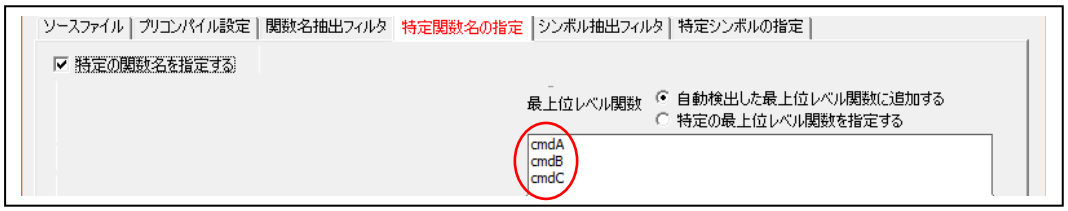
```
+-------------------------------------------------------------------------------/
| 関数の呼び出し構造
+-------------------------------------------------------------------------------+

    No.  Structure         File-Path          < Line >

#     4: main ------------ "D:¥work¥a1¥a1.c" <    31>
         +- cmdA ---------------------------- "D:¥work¥a1¥a1.c" <   16>
         |  +- sub1 -------------------------- "D:¥work¥a1¥a1.c" <    5>
         +- cmdB ---------------------------- "D:¥work¥a1¥a1.c" <   21>
         |  +- sub2 -------------------------- "D:¥work¥a1¥a1.c" <    9>
         |     +- sub2 ...
         +- cmdC ---------------------------- "D:¥work¥a1¥a1.c" <   27>
```

ここで、cmdA()～cmdC()を最上位関数として指定してみます。

cmdA()～cmdC()を最上位関数とするには、「関数名抽出フィルタ」タブ中の「この文字列で始まる関数名を最上位レベルとする」の部分に「cmd」と入力するか、あるいは、「特定関数名の指定」タブ中の「最上位レベル関数」として「cmdA」「cmdB」「cmdC」を入力します。



あるいは



これで、cmdA()～cmdC()は最上位レベル関数とみなされて、関数の呼び出し構造は以下のように表示されます。

```
+-------------------------------------------------------------------------------/
| 関数の呼び出し構造
+-------------------------------------------------------------------------------+

    No.  Structure         File-Path          < Line >

#     1: cmdA ------------ "D:¥work¥a1¥a1.c" <    16>
         +- sub1 ---------------------------- "D:¥work¥a1¥a1.c" <    5>

#     2: cmdB ------------ "D:¥work¥a1¥a1.c" <    21>
         +- sub2 ---------------------------- "D:¥work¥a1¥a1.c" <    9>
            +- sub2 ...

#     3: cmdC ------------ "D:¥work¥a1¥a1.c" <    27>

#     4: main ------------ "D:¥work¥a1¥a1.c" <    31>
         +- cmdA ..#1
         +- cmdB ..#2
         +- cmdC ..#3
```

尚、次のようなサンプルプログラムでは、cmdA()～cmdC()を動的に呼び出している為、cmdA()～cmdC()の呼び出しが認識されません。
従って、cmdA()～cmdC()は（いずれからも呼び出されていない）最上位レベル関数として認識されます。

a1.c

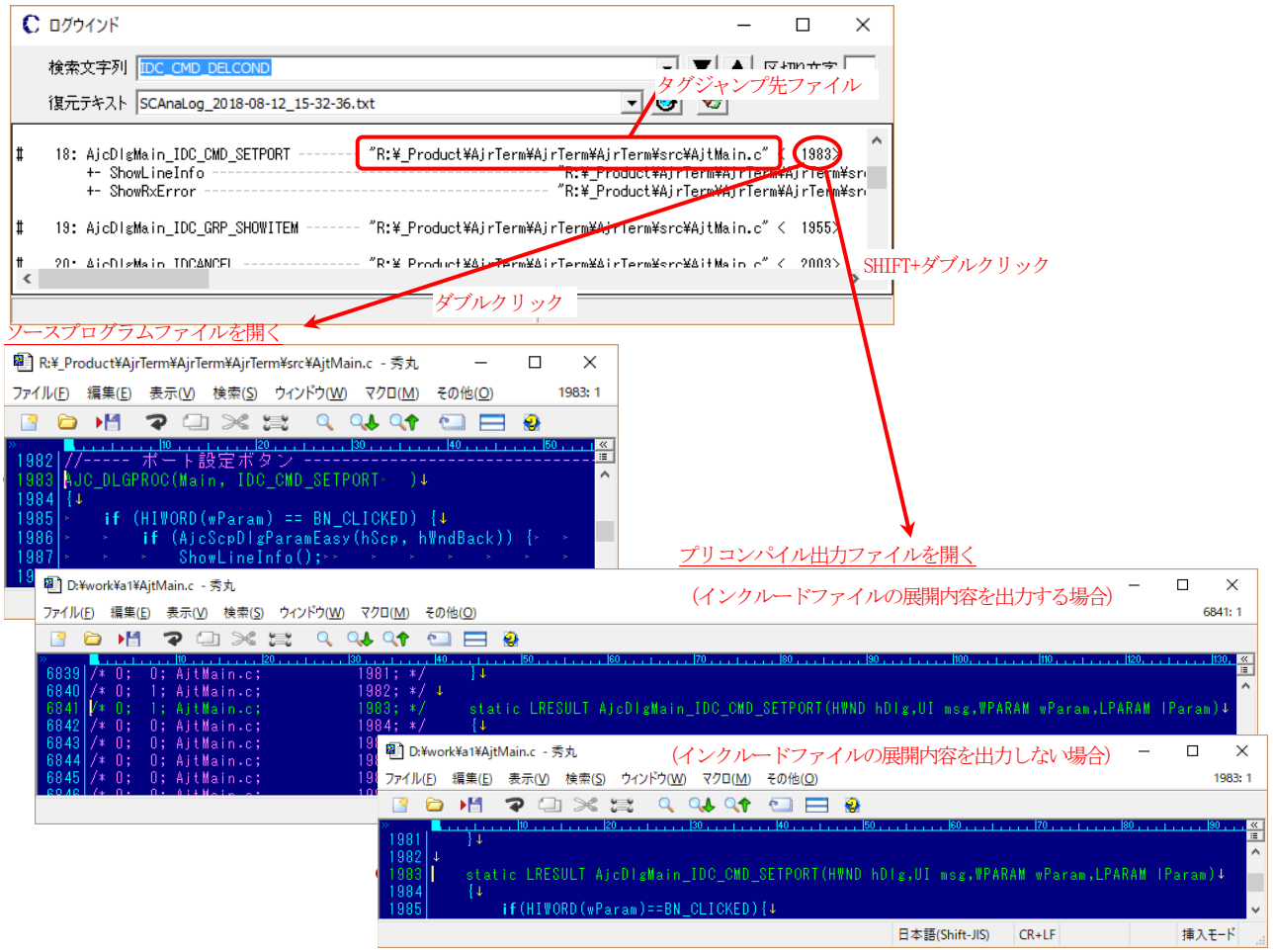
```
 1 : #include <string.h>
 2 :
 3 : static int cnt = 0;
 4 :
 5 : void sub1(void)
 6 : {
 7 : }
 8 :
 9 : void sub2(void)
10 : {
11 :     if (cnt++ < 10) {
12 :         sub2();
13 :     }
14 : }
15 :
16 : void cmdA(void)
17 : {
18 :     sub1();
19 : }
20 :
21 : void cmdB(void)
22 : {
23 :     cnt = 0;
24 :     sub2();
25 : }
26 :
27 : void cmdC(void)
28 : {
29 : }
30 :
31 : typedef struct {
32 :     char *pCmd;
33 :     void (*f)(void);
34 : } FUNC_TBL;
35 :
36 : static FUNC_TBL ft[] = {
37 :     {"A", cmdA},
38 :     {"B", cmdB},
39 :     {"C", cmdC},
40 : };
41 : #define N   ((sizeof ft) / sizeof ft[0])
42 :
43 : int main(int argc, char *argv[])
44 : {
45 :     int i;
46 :
47 :     if (argc >= 2) {
48 :         for (i = 0; i < N; i++) {
49 :             if (strcmp(argv[1], ft[i].pCmd) == 0) {
50 :                 ft[i].f();
51 :             }
52 :         }
53 :     }
54 :     return 0;
55 : }
56 :
```

この場合、cmdA()～cmdC()は最上位レベル関数として扱われますので、関数の呼び出し構造は以下のように表示されます。
但し、main()から、cmdA()～cmdC()の呼び出しは、(動的な呼び出しは認識できない為) 呼び出し構造に表れません。

```
+-------------------------------------------------------------------------------------------------/
| 関数の呼び出し構造
+-------------------------------------------------------------------------------------------------+

    No.  Structure          File-Path          < Line >

#     1: cmdA ------------ "D:¥work¥a1¥a1.c" <    16>
         +- sub1 ---------------------------------- "D:¥work¥a1¥a1.c" <     5>

#     2: cmdB ------------ "D:¥work¥a1¥a1.c" <    21>
         +- sub2 ---------------------------------- "D:¥work¥a1¥a1.c" <     9>
            +- sub2 ...

#     3: cmdC ------------ "D:¥work¥a1¥a1.c" <    27>

#     4: main ------------ "D:¥work¥a1¥a1.c" <    43>
```

1.8. タグジャンプ

ログウインド上でダブルクリックすると、タグジャンプ先のファイルを開きます。
SHIFT キーを押しながらダブルクリックした場合は、(プリコンパイル出力したファイルがあれば)プリコンパイル出力ファイルを開きます。



1.8.1. タグジャンプ先パス名

ダブルクリックした行のダブルクォート(“)でサンドイッチした部分がタグジャンプ先ファイルのパス名となります。
ダブルクリックした行にダブルクォート(“)でサンドイッチした部分が無い場合は、上方向に遡ってパスを検索します。
尚、SHIFT キーが押されている場合は、見つかったパス名を、プリコンパイル結果のパス名に変更します。

1.8.2. タグジャンプ先行番号

数字部分をダブルクリックした場合は、当該数字を行番号とみなします。
数字以外の部分をダブルクリックした場合は、ダブルクリックした行の ’<’ と ’>’ でサンドイッチした部分を行番号とみなします。
ダブルクリックした行に、’<’ と ’>’ でサンドイッチした部分が無い場合は、行番号＝１とします。

1.8.3. テキストエディタの設定

タグジャンプ先の行番号を有効とするには、テキストエディタの設定を行う必要があります。
「設定」→「テキストエディタの設定」メニューから、以下ように、テキストエディタ起動用コマンドラインを設定します。

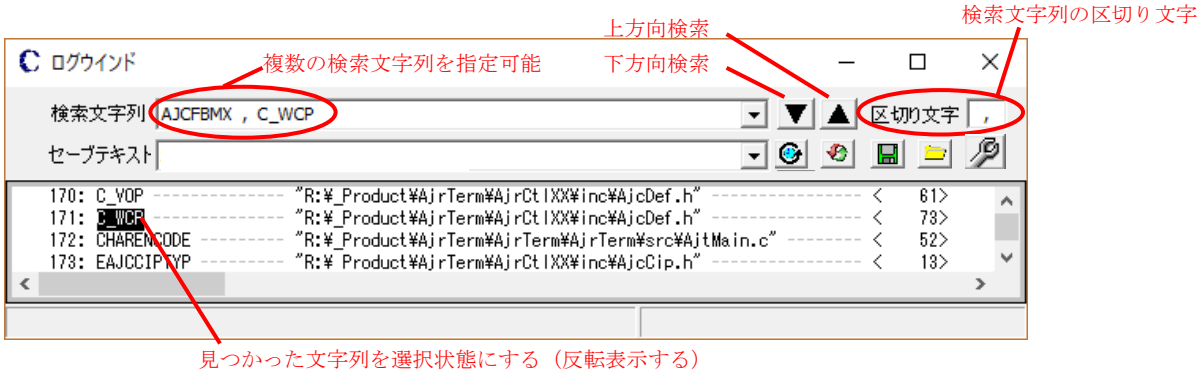
テキストエディタの設定例

「パラメタ」中の「%F」はタグジャンプ先パス名に変換され、「%L」は行番号に変換されます。

テキストエディタの設定を行っていない場合は、単にタグジャンプ先のファイルを開きます。
（エクスプローラで当該ファイルをダブルクリックしたのと同じ動作となります）

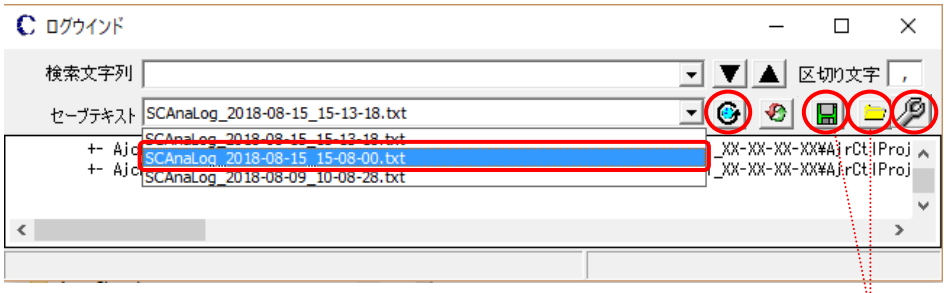
1.9. 文字列の検索

ログウインド上で、文字列を検索できます。
複数の文字列を検索する場合は、区切り文字（区切り文字はログウインド右上で指定する）で区切って指定します。
区切り文字を指定した場合、分離された、各検索文字列の前後の空白は無視されます。(”ABC,XYZ”と　“ ABC , XYZ “ は同じ)
文字列が見つかった場合は、当該文字列を選択状態（反転表示）にします。

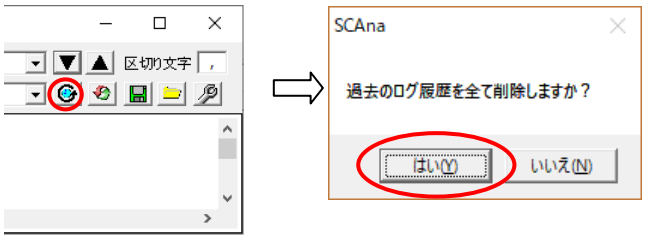


1.10. 解析結果ログのセーブ

ボタンを押すと解析結果ログをセーブします。 ボタンを押すと、解析結果ログのフォルダを開きます。
過去にセーブしたログを表示するには、ログウインドの「セーブテキスト」から再表示するテキストファイルを選択し、 ボタンを押します。
ボタンは、 や ボタンでテキストをセーブ／読み出しする際のテキストエンコードを設定します。(全ログウインドで共通)



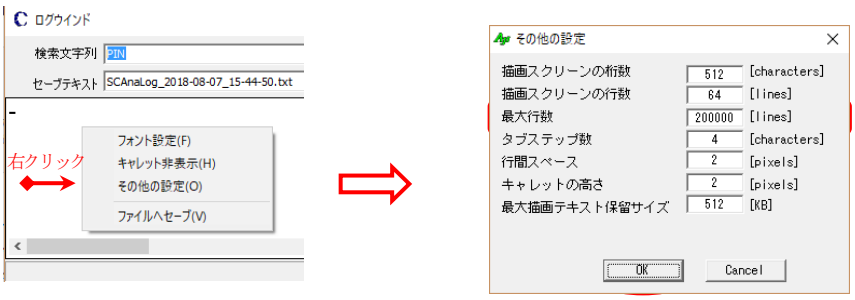
ボタンを押すと、過去にセーブしたログを全て消去します。



ボタンでセーブした場合、ファイル名にセーブした日時が含まれます。
セーブしたファイル名を変更する場合は、 ボタンでフォルダを開き、エクスプローラで直接変更してください。
この時、先頭の「 SCAnaLog_ 」は変更しないでください。
ファイル名は降順となっていますので、日時の後に特有の名前を追加すれば順序も維持できます。

1.11. ログの最大行数

解析結果をログ出力する際の最大行数は、デフォルトで 200,000 行に設定されています。
200,000 行で不足する場合は、ログウインドを右クリックし、「その他の設定」メニューから「最大行数」を変更し「ＯＫ」ボタンを押してください。



1.12. リストボックス操作

ソースファイルやインクルードパス等の設定用リストボックスを右クリックすると、リストボックスの操作メニューが表示されます。

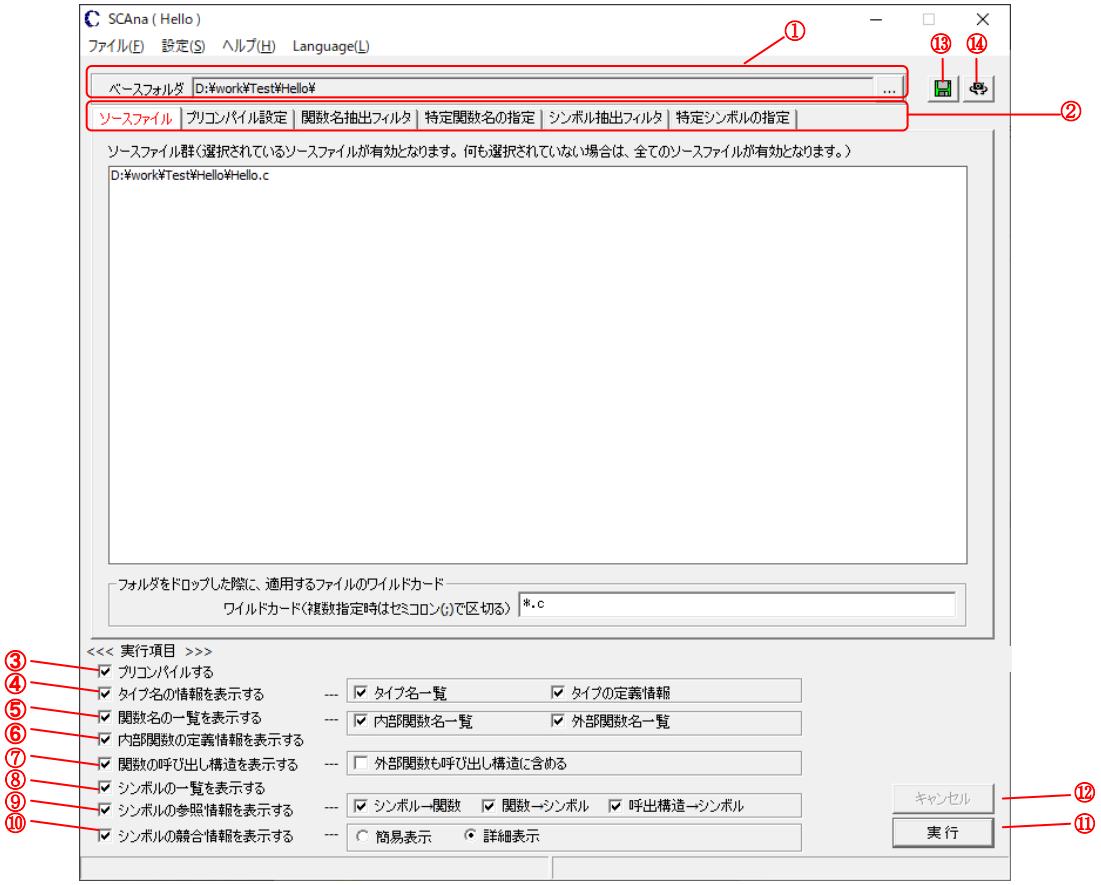
リストボックス操作メニュー

- 選択されている項目を編集します（選択項目が無い場合は、項目追加）
- 全ての項目を選択状態にします
- 全ての項目の選択状態を解除します（リストボックスをダブルクリックでも可）
- 選択されている項目を削除します
- 全ての項目を削除します
- 全ての項目をクリップボードにコピーします（※１）
- 選択されている項目をクリップボードにコピーします（※１）
- クリップボードから項目の追加を行います（※１）
- 選択されている項目を相対パスに変換します（※２）
- 選択されている項目を絶対パスに変換します(※２)

※1：各リストボックス項目が、クリップボード中のテキスト１行と対応します。
※２：「ベースフォルダ」からの相対パスとなります。「ベースフォルダ」が設定されていない場合や、パス名とは関係のないリストボックスでは、このメニューは表示されません。

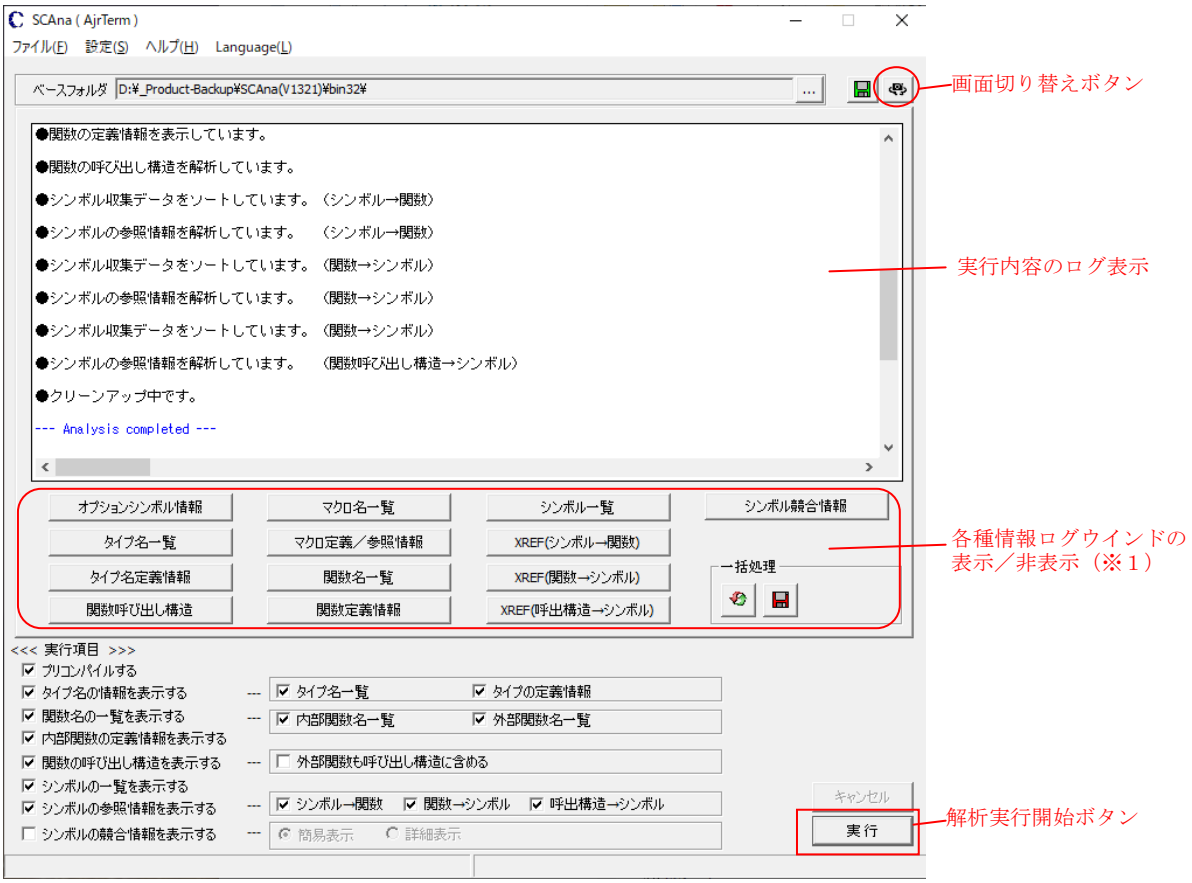
2. メインウインド

SCANA を起動すると以下のメインウインドが表示されます。



#	内　容
①	ソースファイルや、インクルードファイルの最上位フォルダを指定します。 インクルードファイルを、このベースフォルダ下から自動的に検索したり、また、パス名を相対パスに変更する際のベース位置となります。
②	タブを選択し、各種設定を行います。　これらについてはすぐ後で記述します。
③	プリコンパイルを行う場合にチェックします
④	タイプ名（typedef で定義されている名称や、構造体／共用体／列挙体の名称）を表示する場合にチェックします ・タイプ名一覧　：　タイプの名称だけを表示 ・タイプ定義情報：　タイプの名称と、定義箇所（ファイル名と行番号）を表示 ※ タイプ名を表示する場合は、プリコンパイルを実行する必要があります。
⑤	関数名の一覧を表示する場合にチェックします。 ・内部関数名一覧　：　内部関数名（ソースプログラム内で定義されている関数）の一覧を表示する場合にチェックします。 ・外部関数名一覧　：　外部関数名（ソースプログラム内で定義されていない関数）の一覧を表示する場合にチェックします。
⑥	収集した内部関数（ソースプログラム内で定義されている関数）の一覧を表示する場合にチェックします。
⑦	関数の呼び出し構造（ネスト構造）を表示する場合にチェックします。 ・外部関数も呼び出し構造に含める：　ソースプログラム内で定義されていない関数の呼び出しも含めます。
⑧	収集したシンボル一覧を表示する場合にチェックします。
⑨	シンボルの参照情報を表示する場合にチェックします。 ・シンボル→関数　　：各シンボルがどの関数のどの場所で参照されているかをリストアップします。 ・関数→シンボル　　：各関数で参照しているシンボルをリストアップします。 ・上位関数→シンボル：最上位関数（と呼び出しているサブ関数も含む）で参照しているシンボルをリストアップします
⑩	最上位関数下で、競合するシンボルをリストアップします。
⑪	C 言語ソースプログラムの解析を開始します。
⑫	C 言語ソースプログラムの解析処理中の場合、キャンセルボタンを押すと処理を中止します。
⑬	現在の設定値をセーブします
⑭	各種設定画面（上記画面）とログ画面を切り替えます。

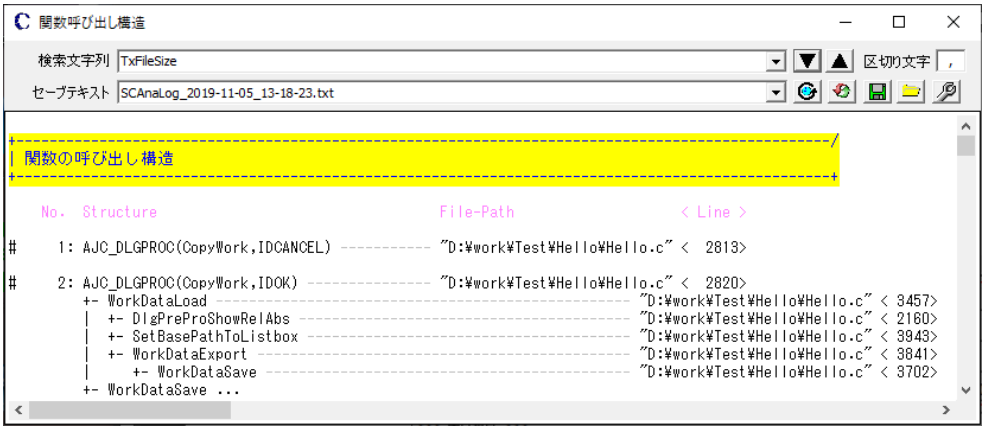
「実行」ボタンを押した場合や、「  」ボタンで画面を切り替えた場合は、以下のような画面となります。



※１：例えば、「関数呼び出し構造」ボタンを押すと、以下のような解析結果のログウインドが表示／非表示されます。

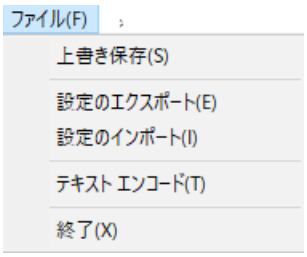
表示中の情報ログウインドの場合、ボタンを窪んだ形で表示します。

⇩



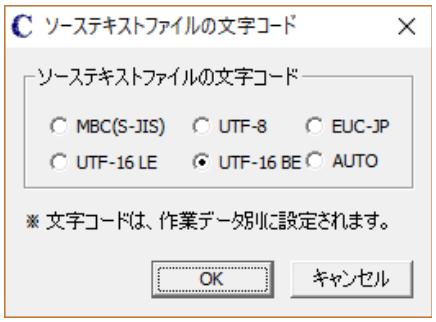
2.1. メニュー

2.1.1. ファイルメニュー



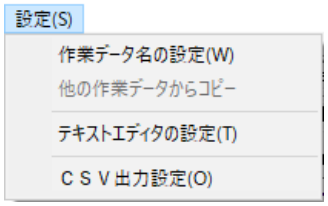
項目	内容
上書き保存	現作業名の設定内容を上書き保存します メインウインド右上の 🖫 ボタンでも同様の操作ができます
設定のエクスポート	現作業名の設定内容をファイルへエクスポートします
設定のインポート	ファイルから設定内容を現作業名へインポートします
テキストエンコード	ソースファイル等、入出力ファイルのエンコード設定（※１）
終了	プログラムを終了します

※１：以下のダイアログにより、ソーステキストファイルのエンコードを設定します。



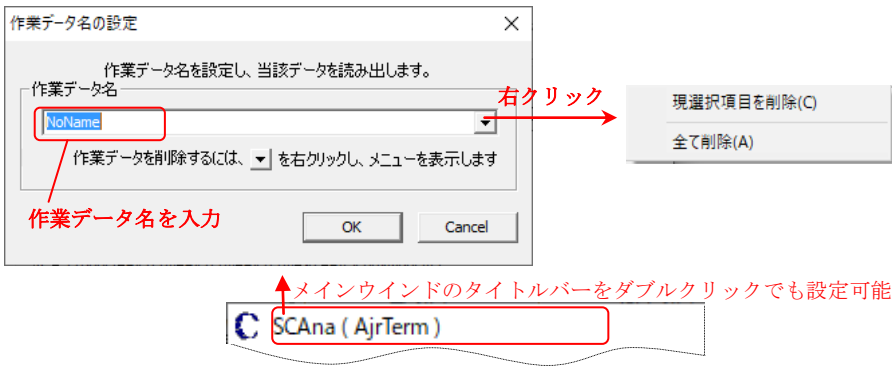
ソースファイル，インクルードファイルのテキストエンコードを設定します。
但し、入力テキストファイルにＢＯＭが設定されている場合はＢＯＭが優先されます。
AUTO を設定した場合は、テキストエンコードを自動的に判別します。
尚、プリコンパイル出力ファイルは、ソースファイルと同じ形式のファイルとなります。

2.1.2. 設定メニュー

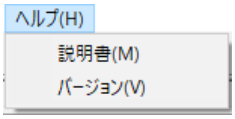


項目	内容
作業データ名の設定	作業データ名を設定します。　（※１） 作業データ名は、メインウインドのタイトルバーに表示されます。 作業データとは、各設定内容を１つにまとめた名称です。 メインウインドのタイトルバーをダブルクリックしても同様の操作ができます。
他の作業データからコピー	他の作業データから、設定内容をコピーします。
テキストエディタの設定	タグジャンプする際に起動するテキストエディタの情報を設定します。
ＣＳＶ出力設定	解析結果データをＣＳＶファイルとして出力するかを設定します。

※１：以下のダイアログにより作業データ名を設定します。

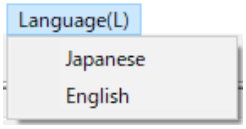


2.1.3. ヘルプメニュー



項目	内容
説明書	ＳＣＡＮＡの操作説明書（本書）を表示します
バージョン	ＳＣＡＮＡのバージョンを表示します

2.1.4. Language メニュー

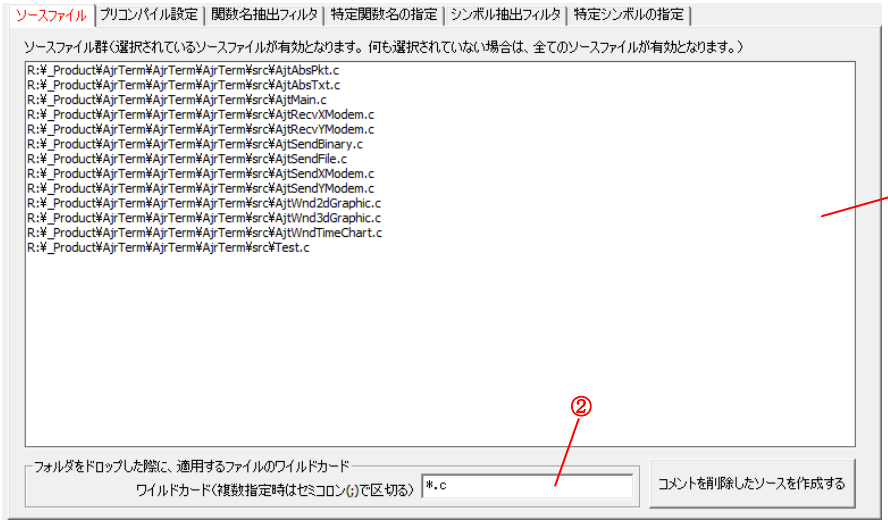


項目	内容
Japanese	メニューやテキスト等を全て日本語で表示します
English	メニューやテキスト等を全て英語で表示します

※言語設定は、次回起動時から有効となります

3. ソースファイルの設定

メインウインドで「ソースファイル」タブを選択すると、以下の画面が表示されます。



ここで、解析対象とするＣ言語ソースファイルを設定します。

いずれかのソースファイルが選択されている場合は、選択されているソースファイルが解析対象となります。
いずれのソースファイルも選択されていない場合は、全てのソースファイルが解析対象となります。
（いずれかのリストボックス項目上で）ダブルクリックすると、全項目の選択が解除されます。

3.1. ソースファイルの設定

以下のいずれかの方法で、ソースファイルを設定します。

１）エクスプローラからソースファイルを ① のリストボックスへドロップする。(Ctrl キー押下時は相対パスに変換します)

２）エクスプローラから、ソースファイルを含むフォルダを ① のリストボックスへドロップする。(Ctrl キー押下時は相対パスに変換します)
　この場合、ドロップしたフォルダとそのサブフォルダ群から、② の「ワイルドカード」で指定されたファイルが設定されます。

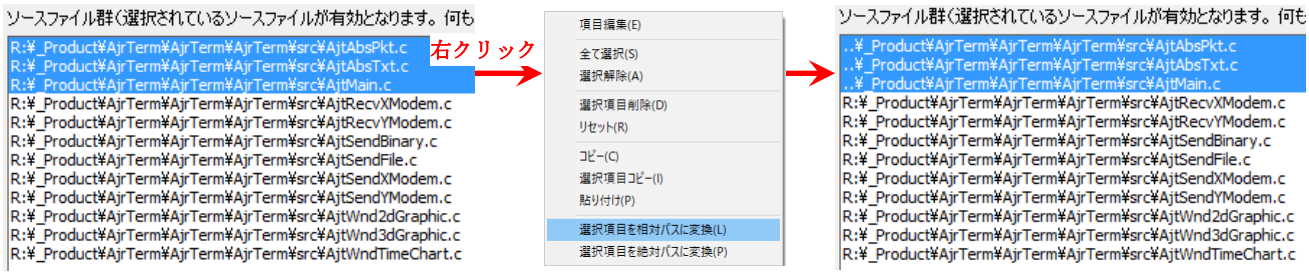
３）① のリストボックスを右クリック→「項目追加」メニューから以下のダイアログでソースファイルを設定する。

「ファイル選択」ボタンを押すと、ファイル選択用のダイアログによりファイルを選択します。
「相対パスに変換」ボタンを押すと、選択したソースファイルを、「ベースフォルダ」からの相対パスに変換します。
③ の部分に、直接ファイルのパス名を入力しても OK です。
「追加」ボタンを押すと設定した項目を追加します。

3.2. 相対パス／絶対パスへの変換

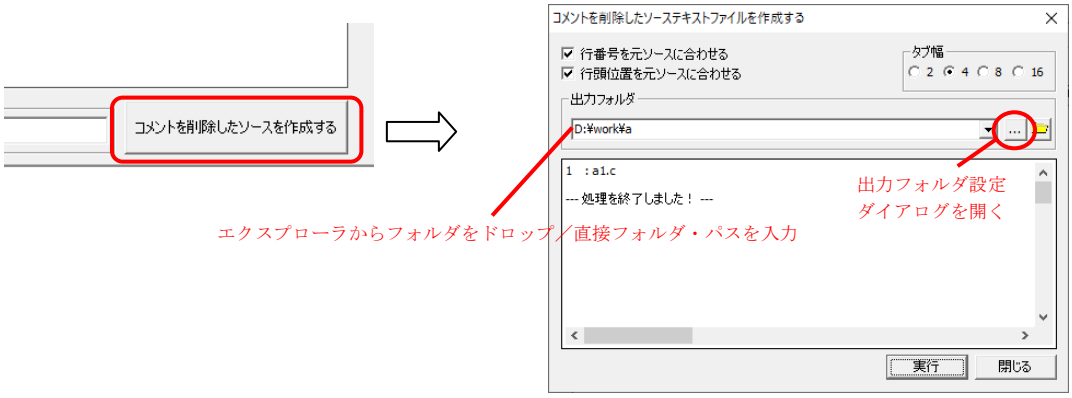
リストボックス上の項目を選択し、右クリック→「選択項目を相対パスに変換」／「選択項目を絶対パスに変換」メニューにより、ソースファイルのパス名を、相対パスや絶対パスに変換できます。
相対パスは、「ベースフォルダ」で設定されているフォルダからの相対位置となります。

相対パスへの変換例



3.3.　コメントを除去したソーステキストの作成

ソースファイルの設定画面の右下にある「コメントを削除したソースを作成する」ボタンを押すと、以下の画面が表示され、コメントを除去し、語句を再構成したソースファイルテキストを作成できます。



ここで、出力フォルダを設定し、「実行」ボタンを押すと、「ソースファイル」タブで設定されているソースファイルからコメントを除去したソーステキストを出力フォルダ内に作成します。

「行番号をソースに合わせる」をチェックすると、改行を挿入し、元のソースと行番号を一致させます。

「行頭位置を元ソースに合わせる」をチェックすると、行の先頭桁位置を元のソースと一致させます。

変換例

元ソース

```
 1 : //-------------------------------------------//
 2 : //  ウインドプロシージャ                     //
 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 :         //----- ウインドワーク生成 ---------//
13 :         if ((pW = (PWRKLISTBOX)malloc(100) == NULL)
14 :             break;
15 :         memset(pW, 0, 100);
16 :         //----- インスタンス識別ＩＤ設定 ---//
17 :         pW->InstID = INST_ID;
18 :     } while(0);
19 : }
```

☐ 行番号を元ソースに合わせる
☐ 行頭位置を元ソースに合わせる

```
 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 : }
```

☑ 行番号を元ソースに合わせる
☐ 行頭位置を元ソースに合わせる

```
 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 : }
```

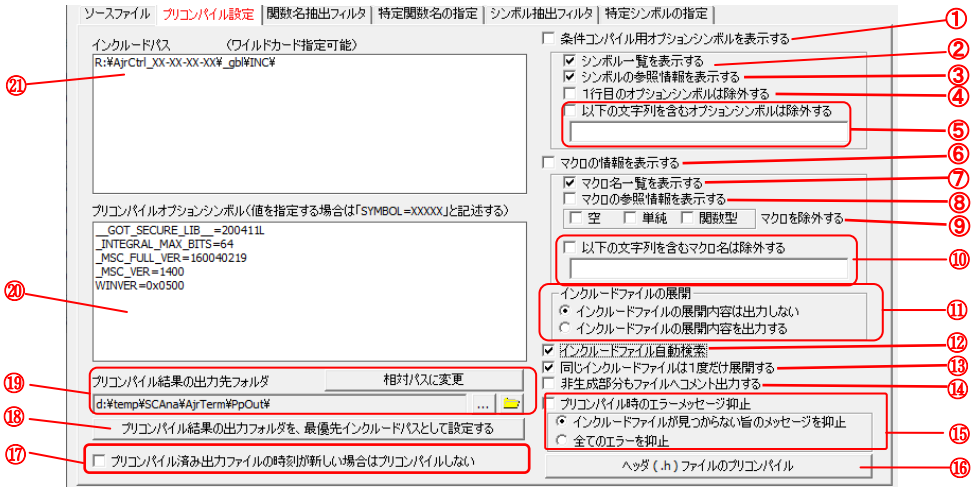
☑ 行番号を元ソースに合わせる
☑ 行頭位置を元ソースに合わせる

```
 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 : }
```

※コメントを除去し、語句を再構成する（空白が必要な個所にだけ１個の空白を置く）ことにより、ソースコンペア・ツールでコメントや空白の個数の相違による不一致を避けることができます。

4.　プリコンパイルの設定とオプションシンボル／マクロ情報の表示

メインウインドで「プリコンパイル設定」タブを選択すると、以下の画面が表示されます。



ここで、プリコンパイルに関する設定をします。

（プリコンパイルとは、「#include」「#define」や「#if」等のプリプロセス文を解決することを意味します。つまり、「#include」で指定されたインクルードファイルを読み出し、「#define」で定義されたマクロを呼び出している場合はマクロ展開したコードに変換し、「#if」「#ifdef」「#ifndef」「#elif」「#else」や「#endif」による条件コンパイルは、「偽条件」となる部分のコードをスキップします。
ちなみに、「#if」や「#elif」でシンボル未定義等により演算が不能な場合は「偽条件」となります。）

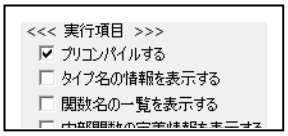
設定する内容は、以下のとおりです。

#	内　容
①	条件コンパイルで使用しているシンボル（#if, #elif, #ifdef, #ifndef で使用しているシンボル）を表示します。 未チェックの場合は、② ～ ⑤ の設定は無効となります。
②	シンボル名の一覧を表示します。
③	シンボルとその参照箇所（ファイル，行番号）を表示します。
④	ファイルの１行目の条件コンパイルで使用しているシンボル（#if, #elif, #ifdef, #ifndef で使用しているシンボル）を除外します。
⑤	特定の文字列を含むシンボルを除外します。特定の文字列を複数指定する場合はセミコロン(;)で区切ります。
⑥	マクロの一覧や参照箇所を表示します。 未チェックの場合は、、⑦～⑩ の設定は無効となります。
⑦	マクロ名の一覧を表示します。
⑧	マクロとその参照箇所（ファイル，行番号）を表示します。
⑨	表示から除外するマクロを指定します。 ・空　　：空マクロ（マクロ名だけで内容が無いマクロ， ex. 「#define　MACNAME」）を除外します ・単純　：単純マクロ（引数のないマクロ， ex 「#define　MACNAME　123」）を除外します。 ・関数型：関数型マクロ（引数付きマクロ， ex 「#define　MACNAME(ARG, ･･･)　･･･」）を除外します。
⑩	特定の文字列を含むマクロ名を除外します。特定の文字列を複数指定する場合はセミコロン(;)で区切ります。
⑪	プリコンパイル結果の出力ソースファイルに、インクルードファイルの内容を含めるか否かを指定します。 「インクルードファイルの展開内容を出力する」を選択した場合は、インクルードしたファイルの内容を含めます。
⑫	サブフォルダも含め「インクルードパス群」と「ベースフォルダ」下からインクルードファイルを自動検索します。 チェックした場合、㉑のインクルードパスにワイルドカードが指定可能となります。
⑬	既に展開したインクルードファイルと同じインクルードファイルは読み込まないようにします
⑭	条件コンパイルで偽条件となり、生成されない部分もコメントとしてファイルに出力します。（行頭に「//- 」付加）
⑮	プリコンパイル時のエラーメッセージを抑止します。「インクルードファイルが見つからない旨のメッセージを抑止」するか、「すべてのエラーを抑止」するかを選択します。

つづく

#	内　容
⑯	ヘッダファイル（ .h ファイル ）のプリコンパイルを行う
⑰	チェックした場合、ファイルのタイムスタンプをチェックし、プリコンパイル済の出力ファイルが、ソースファイルより新しい場合にはプリコンパイルを実行せずに、プリコンパイル済のソースを入力し解析します。 ※#include によるインクルードファイルの読み込みは行われない為、インクルードファイル内で定義されているマクロ名やタイプ名等の情報は出力されません。
⑱	⑰のプリコンパイル出力先フォルダを、インクルードパスとして設定します。 このインクルードパスは、最優先のパスとなり、自フォルダ内のインクルード（#include “*XXX*.h”）よりも優先されます。
⑲	プリコンパイルしたファイルの出力先を指定します。 「相対パスに変更」／「絶対パスに変更」ボタンを押すと、ベースフォルダを基準にして、パスの相対／絶対を変更します。 プリコンパイル出力ファイルは、同一フォルダ中に同じファイル名で作成します。（サブフォルダは作成しません）
⑳	プリコンパイル用オプションシンボルを指定します。 右クリックによるポップアップメニューから設定します。 シンボルに値を設定する場合は、「*SYMBOL=値*」の形式で指定します。
㉑	以下のいずれかの方法で、インクルードパスを指定します。 ・この部分にフォルダをドロップする。（Ctrl キーを押しながらドロップした場合は、相対パスに変換されます） ・右クリックによるポップアップメニューから設定します。 ・リストボックス上の項目を選択し、右クリック→「選択項目を相対パスに変換」／「選択項目を絶対パスに変換」メニューにより、インクルードパス名を、相対パスや絶対パスに変換できます。 **※ ⑫ のチェックボックス（インクルードファイル自動検索）をチェックした場合、インクルードパスとしてワイルドカードが指定可能となります。（ex. 「c:Program Files¥Microsoft*.*」）**

プリコンパイルを実行するには、メインウインドで「プリコンパイルする」をチェックします。

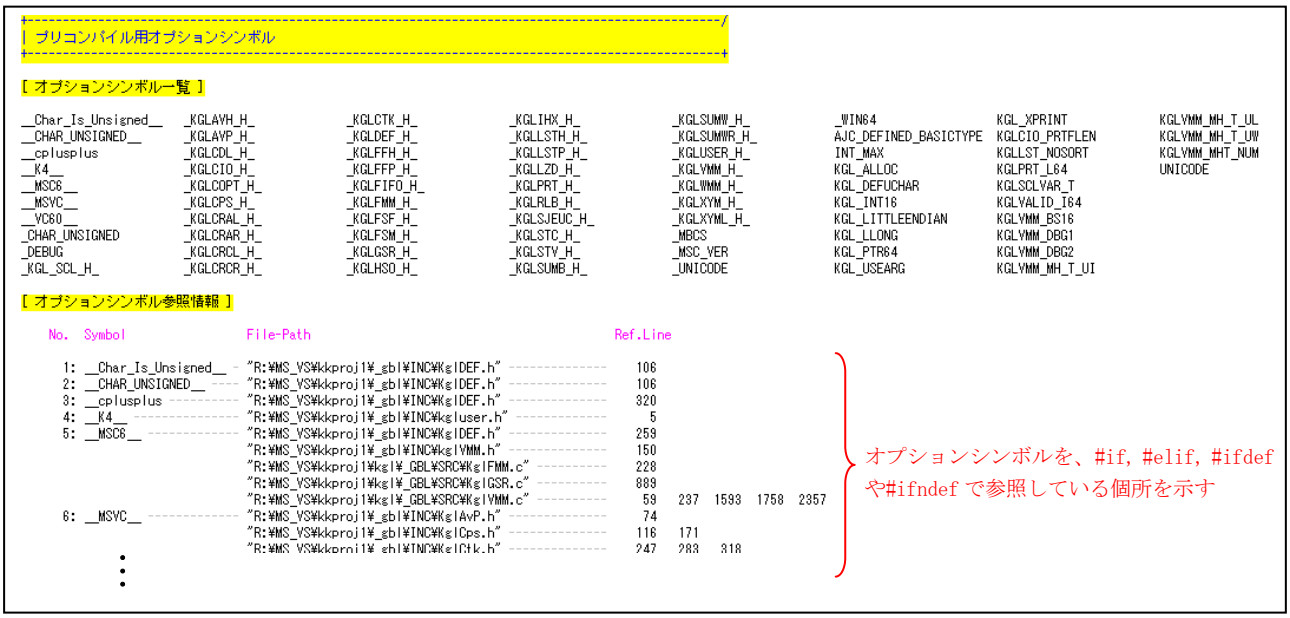


①の「条件コンパイル用オプションシンボルを表示する」をチェックすると、プリコンパイル時に「#if」「#elif」「#ifdef」「#ifndef」で参照したオプションシンボルを表示します。

「シンボル一覧を表示する」をチェックすると、オプションシンボル名の一覧を表示します。

「シンボルの参照情報を表示する」をチェックすると、オプションシンボルの参照箇所（ファイル名，行番号）をリストアップします。

オプションシンボル情報の表示例を以下に示します。



⑥の「マクロの情報を表示する」をチェックすると、プリコンパイル時に検出したマクロの情報を表示します。
「マクロ名一覧を表示する」をチェックすると、マクロ名名の一覧を表示します。
「マクロの参照情報を表示する」をチェックすると、マクロのの定義／参照箇所（ファイル名，行番号）をリストアップします。

マクロ情報の表示例を以下に示します。

```
+-------------------------------------------------------------------------------/
| マクロ名一覧
+-------------------------------------------------------------------------------+
__DATE__                        AjcGetFolderNameEx              AJCPMG_WEDNESDAY                AjcSetDlgPropU8                 AJCVTHS_LOGFILE
__TIME__                        AjcGetIniFileArr                AJCPMG_WEEKDAY                  AjcSetDlgPropUI16               AJCVTHS_NOBORDER
__TIMESTAMP__                   AjcGetIniFileBin                AJCPMG_WEEKEND                  AjcSetDlgPropUI64               AJCVTHS_NOSCRL
_AJCAVLTREE_H_                  AjcGetIniFileH64                AjcPmgSetSchedInterval          AjcSetDlgPropUInt               AJCVTHS_SEPARATE
_AJCDEF_H_                      AjcGetIniFileHex                AjcPmgSetSchedTime              AjcSetDlgValues                 AjcVthSaveFont
_AJCRINGBUF_H_                  AjcGetIniFilePath               AjcPmgSetSchedWeekly            AjcSetIniFilePath               AjcVthSaveHtmlToFile
_AJRCHECKSUM_H_                 AjcGetIniFileReal               AjcPopupMenu                    AjcSetProfilePath               AjcVthSaveProp
_AJRCTL32_H_                    AjcGetIniFileSI64               AJCPOPUPMENU_H_                 AjcSetRegMidPath                AjcVthSaveTextToFile
_AJRCTLXX_H_                    AjcGetIniFileSInt               AJCPOWMG_H_                     AjcSetRegRootPath               AjcVthSearchAbove
_MSC_VER                        AjcGetIniFileStr                AJCPPC_IFA_GEN                  AJCSM                           AjcVthSearchBelow
ADDINDLLSECT                    AjcGetIniFileUI64               AJCPPC_IFA_SKIP                 AjcSMotToBin                    AjcVthSetTipText
AJC3DG_CLEAR_DATA               AjcGetIniFileUInt               AJCPPC_INCKND_GBL               AjcSnPrintF                     AjcVthSetTitleText
AJC3DG_CLEAR_PLOT               AjcGetLastErrorText             AJCPPC_INCKND_LCL               AjcSnPrtSep                     AJCWNDITEM_H_
AJC3DG_MAXAXIS                  AjcGetMyPath                    AJCPPC_INCKND_NO                AjcSplCreate                    AJCWNDSUPPORT_H_
AJC3DG_MAXITEM                  AjcGetOpenFile                  AJCPPC_MAX_IF_NEST              AjcSplEnumStr                   AjcWriteBitmapToFile
AJC3DGM_GETBORDERCOLOR          AjcGetOpenFiles                 AJCPPC_MAX_INC_NEST             AjcSplFind                      AjcXymCreate
AJC3DGM_GETPROP                 AjcGetOpenFilesEx               AJCPPC_PPK_ALL                  AJCSPLINE_H_                    AJCXYMFILEINFO
AJC3DGM_NEEDNTC_ANGLE           AJCGETPF_ARR                    AJCPPC_PPK_COND                 AjcSplPartStrInPool             AJCXYMODEM_H_
AJC3DGM_SETBORDERCOLOR          AJCGETPF_BIN                    AJCPPF_ACT_MISSING_INC          AjcSplPoolStrInStr              AJCXYN_ABORT
AJC3DGM_SETPROP                 AJCGETPF_HEX                    AJCPPF_INC_AUTO_SEARCH          AjcSplRegist                    AJCXYN_COMPLETE
AJC3DGMAX_PLOTLIST              AJCGETPF_REAL                   AJCPPMF_EOT                     AjcSplRemove                    AJCXYN_END
        :
        :
+-------------------------------------------------------------------------------/
| マクロ名定義，参照情報
+-------------------------------------------------------------------------------+
   No.  Macro Name                  File-Path / Ref-Line          < Line >
        :
        :
                                      594
                                   "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWndTimeChart.c"
                                      623
   71: AJC_WNDMAP_END ------------ "R:¥_Product¥AjrTerm¥AjrCtlXX¥inc¥AjcMsgMap.h" ------------ <    70>          マクロ定義箇所
                                   "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtAbsPkt.c"
                                      607
                                   "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtAbsTxt.c"
                                      830
                                   "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c"
                                     1629   2049
                                   "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWnd2dGraphic.c"
                                      609
                                   "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWnd3dGraphic.c"
                                      610
                                   "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtWndTimeChart.c"
                                      640
   72: AJC_WNDMAP_MSG ------------ "R:¥_Product¥AjrTerm¥AjrCtlXX¥inc¥AjcMsgMap.h" ------------ <    59>
                                   "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtAbsPkt.c"
                                      606
                                   "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtAbsTxt.c"
                                      829
        :
        :
```

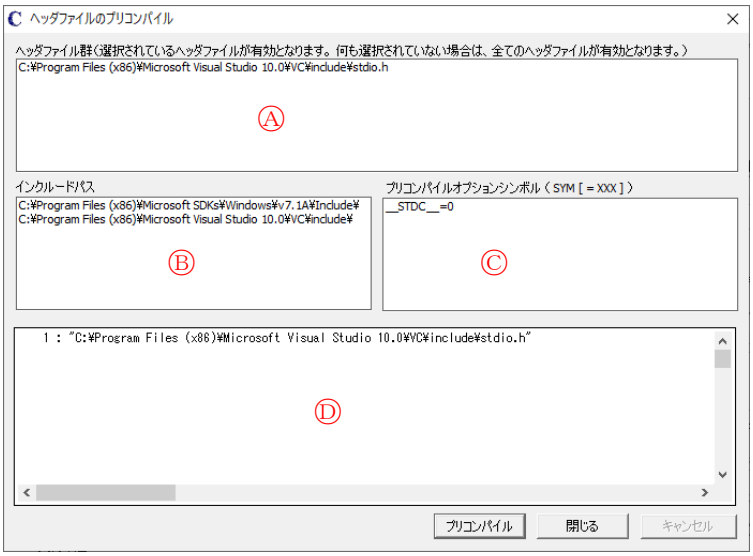
マクロ参照箇所

4.1. ヘッダファイルのプリコンパイル

ヘッダファイルが巨大な場合で、このヘッダファイルが共通的にインクルードされている場合、ソースファイル群のプリコンパイルに時間がかかります。

ヘッダファイルをあらかじめプリコンパイルしておくことにより、ソースファイルのプリコンパイル時間を短くすることができます。
ヘッダファイルのプリコンパイルは、以下の手順で実行します。

1）⑯ の「ヘッダ（ .h ）ファイルのプリコンパイル」ボタンを押すと以下の画面が表示されます。



2）以下の設定をします。

- Ⓐの部分にプリコンパイルするヘッダファイルを設定します。(ヘッダファイルをドロップでも可)
- Ⓑの部分にプリコンパイルに必要な、インクルードパスを設定します。(フォルダをドロップでも可)
- Ⓒの部分に、プリコンパイルに必要なコンパイルオプションを指定します。(右クリックで編集メニュー表示)

3）「プリコンパイル」ボタンを押して、プリコンパイルを実行します。(エラーがある場合Ⓓの部分に**赤色**で表示されます)

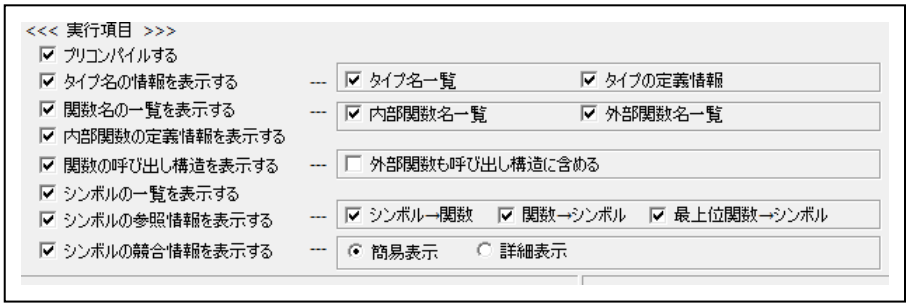
4）「閉じる」ボタンを押して、ヘッダファイルのプリコンパイル画面をクローズします。

5） ⑱の「プリコンパイルの出力フォルダを、最優先インクルードパスとして設定する」ボタンを押して、⑰のプリコンパイル出力先フォルダを、最優先インクルードパスとして設定します。(「インクルードパス」に当該フォルダが追加されます)

※ 手動で「インクルードパスに」⑰のプリコンパイル出力先を追加しても最優先インクルードパスとはなりません。

5. タイプ名情報の表示

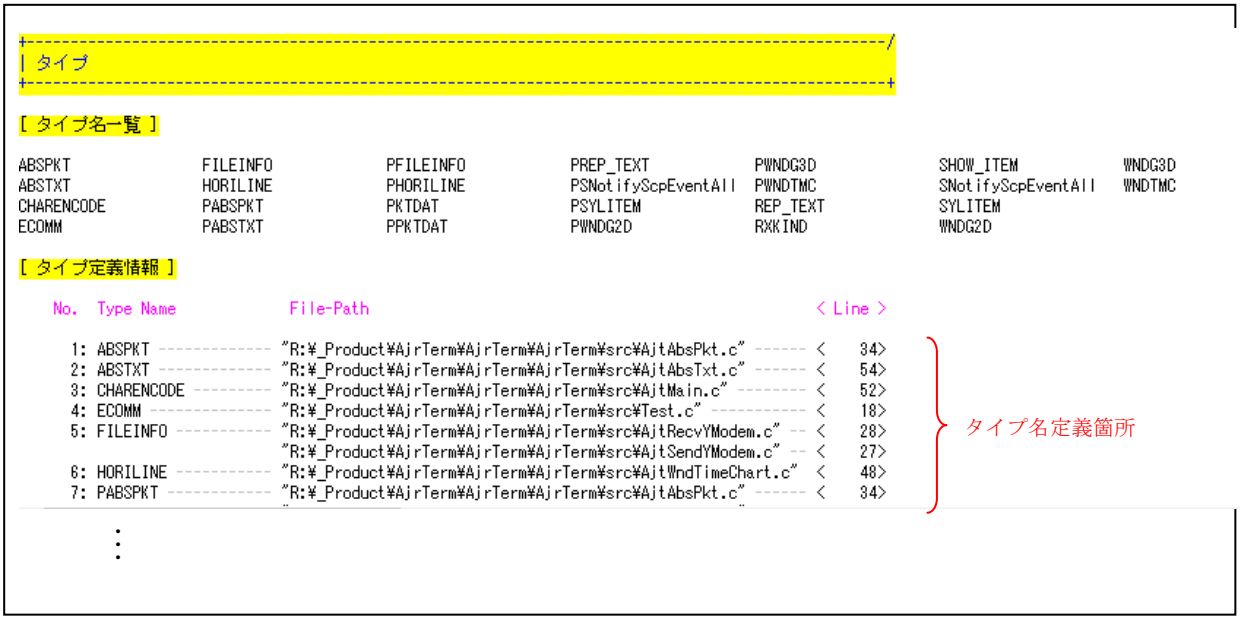
メインウインドで「タイプ名の情報を表示する」をチェックすると、タイプ名に関する情報を表示します。
「タイプ名一覧」をチェックすると、タイプ名の一覧を表示します。
「タイプの定義情報」をチェックすると、タイプ名の定義箇所を表示します。



タイプ名とは、「typedef」で定義された名称の他に、構造体名(struct)，共用体名(union)，列挙名(enum)も含みます。

プリコンパイルを実行した場合は、プリコンパイル出力したソースと、インクルードファイル中で定義されているタイプ名を抽出します。
プリコンパイルを実行しない場合は、ソースプログラムファイル中で定義されているタイプ名を抽出します。

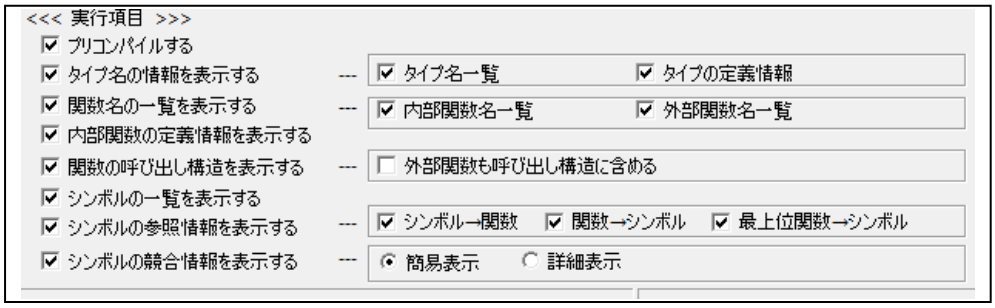
タイプ名情報の表示例を以下に示します。



6. 関数名情報表示

メインウインドで「関数名の一覧を表示する」をチェックすると関数名の一覧を表示します。
関数名の一覧表示は「内部関数名一覧」と「外部関数名一覧」を選択できます。（２つ選択も可）

「内部関数の定義情報を表示する」をチェックすると、内部関数の定義箇所と、関数内の命令数，if や while 等のキーワード個数を集計表示します。

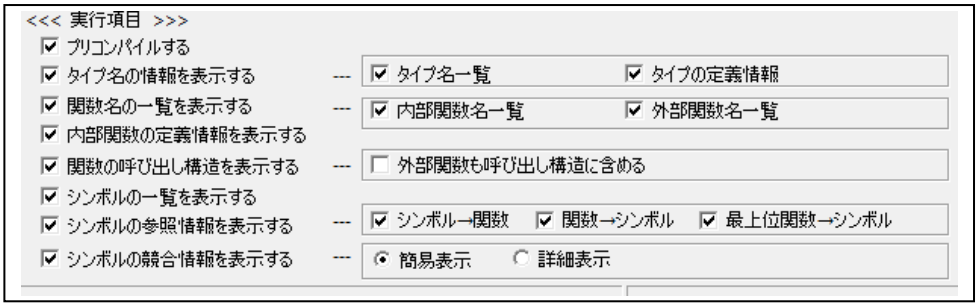


関数名情報の表示例を以下に示します。

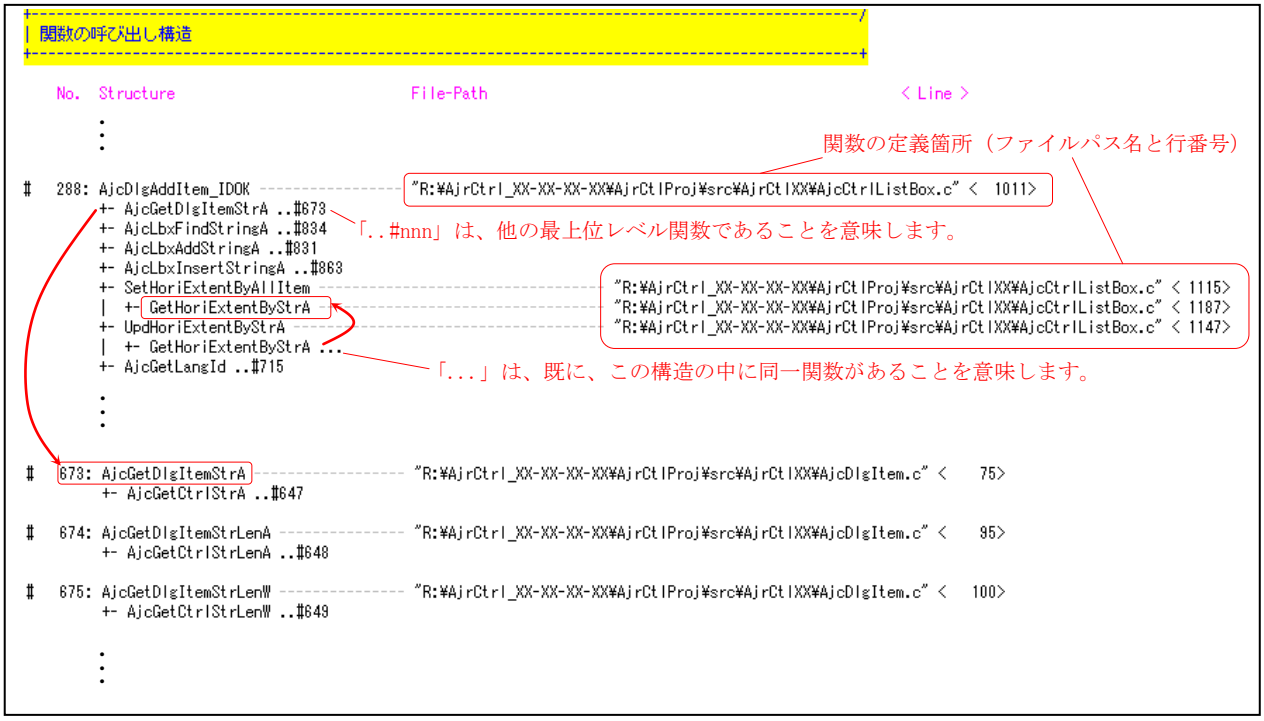
```
+-------------------------------------------------------------------------------+
| 関数名一覧
+-------------------------------------------------------------------------------+

[ 内部関数名一覧]

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
          :
          :
          :

[ 外部関数名一覧]

__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
          :
          :
          :

+-------------------------------------------------------------------------------+
| 内部関数定義情報
+-------------------------------------------------------------------------------+

    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¥AjrTerm¥src¥Test.c" -----------  <   210>       2       1       0       0       0       0       0
          :
          :
          :
```

7. 関数の呼び出し構造の表示

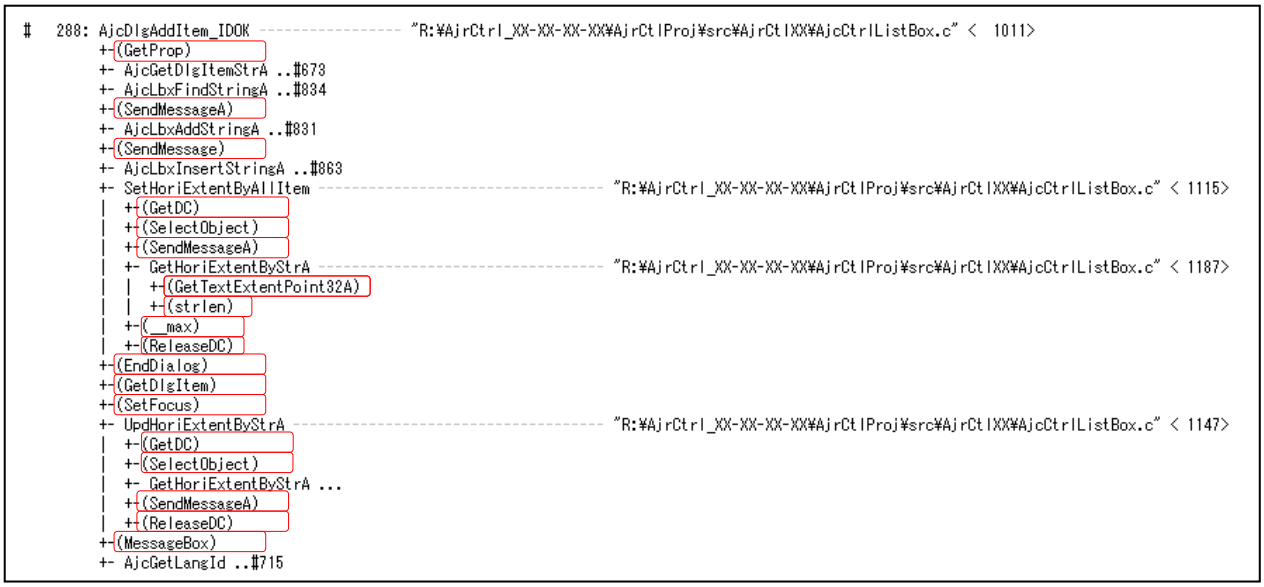
メインウインドで「関数の呼び出し構造を表示する」をチェックすると、関数呼び出し構造を解析し表示します。
「外部関数も呼び出し構造に含める」をチェックすると、関数の呼び出し構造の中に、外部関数も含めます。



関数の呼び出し構造の表示例を以下に示します。

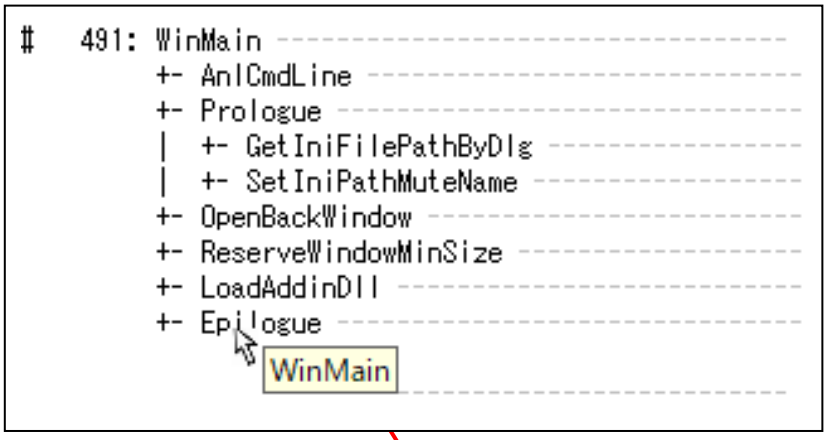


外部関数を呼び出し構造に含めた場合、外部関数は括弧で囲んで表示します。



7.1. 最上位関数名をツールチップ表示

関数名にカーソルを置くと、ツールチップテキストとして、当該サブ関数の最上位関数名を表示します。
（当該関数を複数のコールツリーでk-るしている場合は、複数の最上位関数が表示されます）



Epilogue()の最上位関数名「WinMain」をツールチップで表示

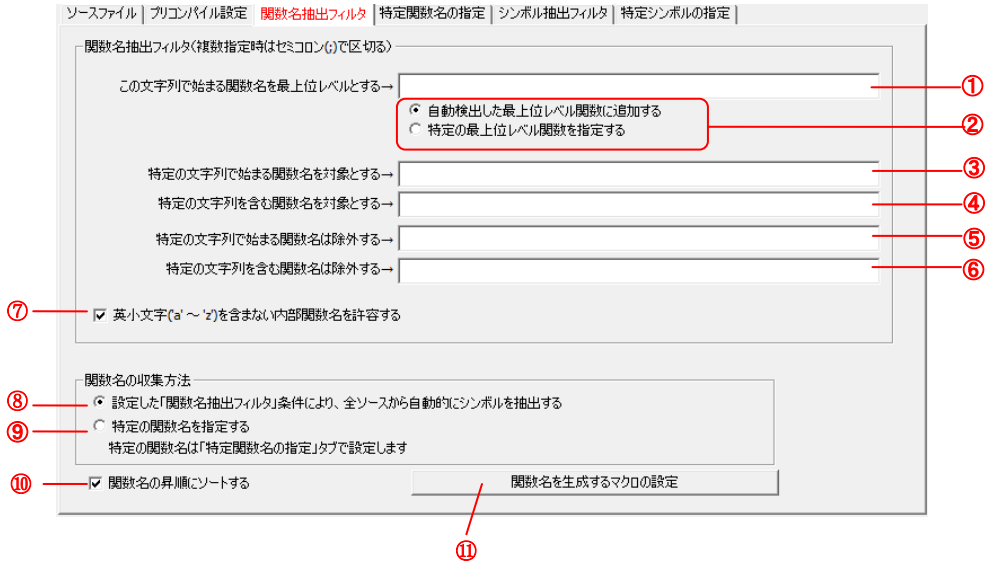
最上位関数のツールチップテキストは数秒経過するか、カーソルをログウインド外に移動すると消えます。
新たな。サブ関数にカーソルを置く場合、一度カーソルをログウインド外に移動してから、目的のサブ関数にカーソルを置いて下さい。

8. 関数名抽出フィルタ

「関数名抽出フィルタ」タブ、あるいは、「特定関数名の指定」タブで、対象とする関数を指定することができます。
「関数名の収集方法」のラジオボタンで「設定した関数名抽出フィルタにより・・」をチェックした場合は「関数名抽出フィルタ」が有効となり、「特定の関数名を指定する」をチェックした場合は「特定関数名の指定」が有効となります。

収集する関数名を指定（あるいは除外）した場合は、対象外となった関数は、関数の呼び出し構造やシンボルの参照情報内に表示されなくなります。

「関数名抽出フィルタ」タブでは、以下のウインドが表示されて、収集する関数名を制限することができます。



ここで、収集する関数名の条件を指定します。
設定する内容は、以下のとおりです。

#	内　容
①	ここで指定した文字列で始まる関数名を最上位関数として扱います。（※1）
②	特定の最上位レベル関数群を指定します。 ・自動検出した最上位レベル関数に追加する 　自動検出した最上位レベル関数に①で指定した関数群を追加します。 ・特定の最上位レベル関数を指定する 　①で指定した関数群だけについて関数の呼び出し構造が作成されます。
③	ここで指定した文字列で始まる関数名を収集します。
④	ここで指定した文字列を含む関数名を収集します。
⑤	ここで指定した文字列で始まる関数名を除外します。
⑥	ここで指定した文字列を含む関数名を除外します。
⑦	英小文字（'a'～'z'）を含まない関数名を収集対象に含めます。 未チェックの場合は、英小文字（'a'～'z'）を含まない関数名は除外されます。
⑧	①～⑦で設定した関数の抽出フィルタ条件を有効にします。
⑨	①～⑦で設定した関数の抽出フィルタ条件を無効とし、「特定の関数名設定」で関数の抽出条件を指定します。
⑩	関数名の昇順にソートして「内部関数一覧」や「関数の呼び出し構造」を出力します。
⑪	関数名を生成するマクロの設定（これについては、すぐ後で説明します）

①，③～⑥の項目で、複数の文字列を指定する場合は、セミコロン(;)で区切って指定します。
指定した文字列の前後の空白は無視されます。

※1：ソースプログラム内のいずれからも呼び出されていない関数は、自動的に最上位関数として扱います。
「関数の呼び出し構造」は、最上位関数下における呼び出し構造を出力しますが、ソースプログラムの他の箇所から呼び出されている関数でも、最上位関数として指定することにより、当該関数についても呼び出し構造が出力されます。

関数名を生成するマクロの設定

以下のように関数名を生成する関数マクロ（AJC_WNDPROC）があるとします。

```
#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
{
        ・・・・・
}
```

実際の関数名はAJC_WNDPROC()マクロで生成され、実際の関数は・・・

①では「 static LRESULT **AjcWndMain_WM_CREATE**(HWND hwnd,UI msg,WPARAM wParam,LPARAM lParam) 」

②では「 static LRESULT **AjcWndMain_WM_SIZE**(HWND hwnd,UI msg,WPARAM wParam,LPARAM lParam) 」

と展開されます。

プリコンパイルを行わない場合（あるいは、有効なヘッダファイルのフォルダが指定されていない場合）AJC_WNDPROC()マクロは展開されず、「AJC_WNDPROC」を関数名として解釈してしまいます。

このように複数個所で同じ関数定義があるとリファレンスを見ても識別が付かなくなります。

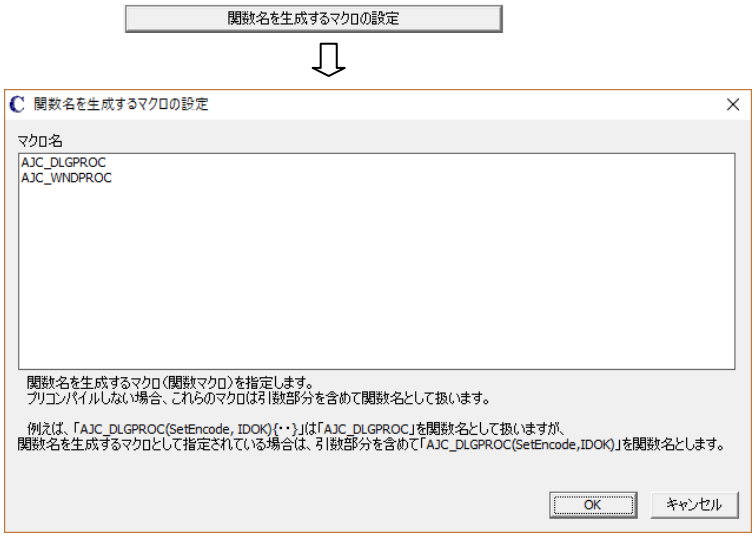
このような場合、引数部分を含めて関数名として扱うように設定することができます。

①では「 AJC_WNDPROC(Main,WM_CREATE) 」

②では「 AJC_WNDPROC(Main,WM_SIZE) 」

を関数名として扱うようにします。

引数部分を含めて関数名として扱うように設定するには、「関数名を生成するマクロの設定」ボタンで、マクロ名を設定します。



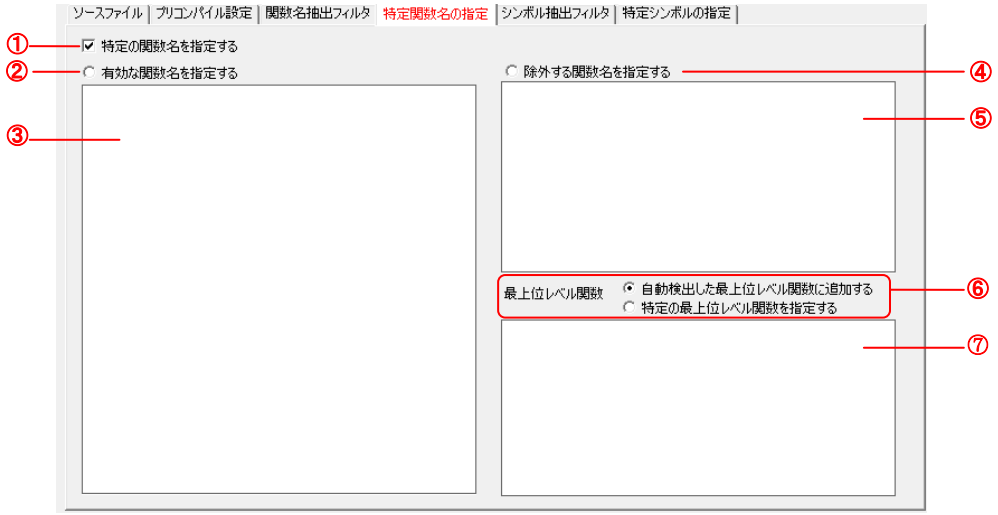
リストボックスに、上記「AJC_WNDPROC」のように関数名を生成するマクロを設定します。

尚、「AJC_DLGPROC」と「AJC_WNDPROC」の２つはデフォルトで設定されていますので、不都合な場合は削除してください。

但し、プリコンパイルを行って、これらのマクロを展開した場合は、マクロにより生成された（本来の）関数名となるので、マクロ名を設定する必要はありません。

8.1. 特定関数名の指定

メインウインドの「特定関数名の指定」タブで、収集する関数や、トップレベル関数を個々に指定することができます。
「関数名抽出フィルタ」タブの設定は、大雑把に収集する関数名を制限するのに対し、「特定関数名の指定」タブでは直接個々の関数名を指定します。



設定内容は、以下のとおりです。

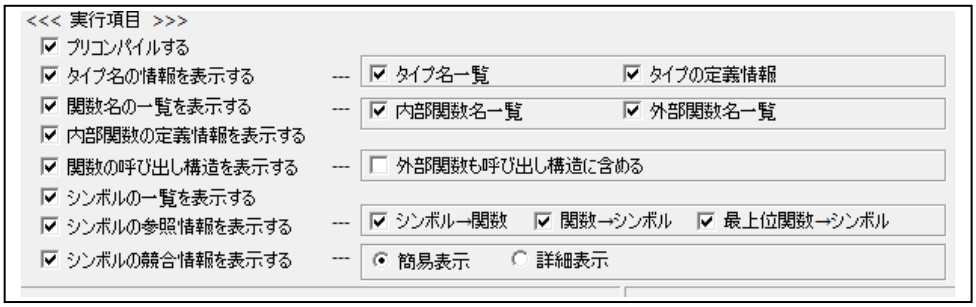
#	内　容
①	このタブで特定の関数名を指定する場合にチェックします。 （「関数名抽出フィルタ」タブで、「特定の関数名を指定する」をチェックしたのと同様）
②	特定の関数名だけを対象とします。 （③で指定した関数群だけで、関数の呼び出し構造やシンボルの相互参照を作成します）
③	②をチェックした場合に、対象とする特定の関数群を列挙します。 ②をチェックし、何も指定しない場合は「対象関数なし」となります。
④	特定の関数群を除外します。 （⑤で指定した関数群を、関数の呼び出し構造やシンボルの相互参照から除外します）
⑤	④をチェックした場合に、除外する特定の関数群を列挙します。 ④をチェックし、何も指定しない場合は、全ての関数が対象となります。
⑥	特定の最上位レベル関数群を指定します。 ・自動検出した最上位レベル関数に追加する 自動検出した最上位レベル関数に⑦で指定した関数群を追加します。 ・特定の最上位レベル関数を指定する ⑦で指定した関数群だけについて関数の呼び出し構造が作成されます。
⑦	⑥の、特定の最上位レベルの関数群を列挙します。

②と④は、いずれか一方だけ設定可能です。

⑦は、最上位レベルの関数として扱う関数名を指定します。

9. シンボルのクロスレファレンス表示

メインウインドで「シンボルの一覧を表示する」をチェックすると、収集したシンボルを一覧表示します。
「シンボルの参照情報を表示する」をチェックすると、シンボルの相互参照情報を表示します。

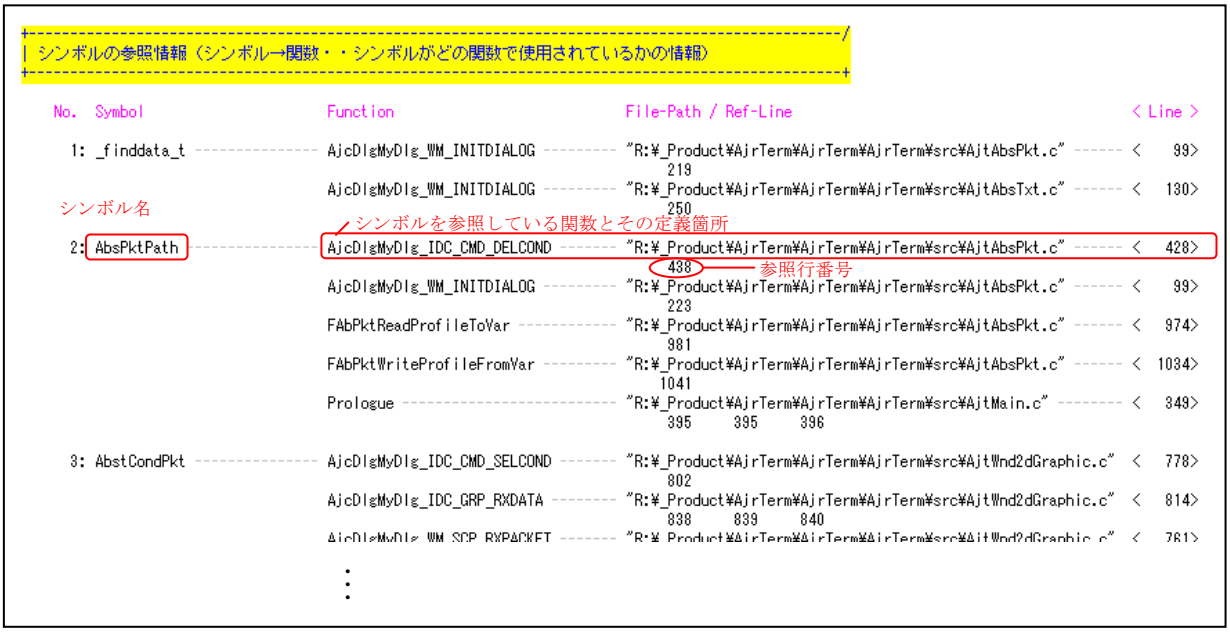


シンボルの相互参照情報には、以下の３つのパターンがあります。

#	項目	内容
1	シンボル→関数	シンボルをリストアップし、当該シンボルを使用している関数を示します。
2	関数→シンボル	関数をリストアップし、当該関数で使用しているシンボルを示します。
3	最上位関数→シンボル	最上位レベルの関数をリストアップし、呼び出しているサブ関数を含めた関数で使用しているシンボルを示します。つまり、当該最上位関数を呼んだ際に使用するシンボルを示します。

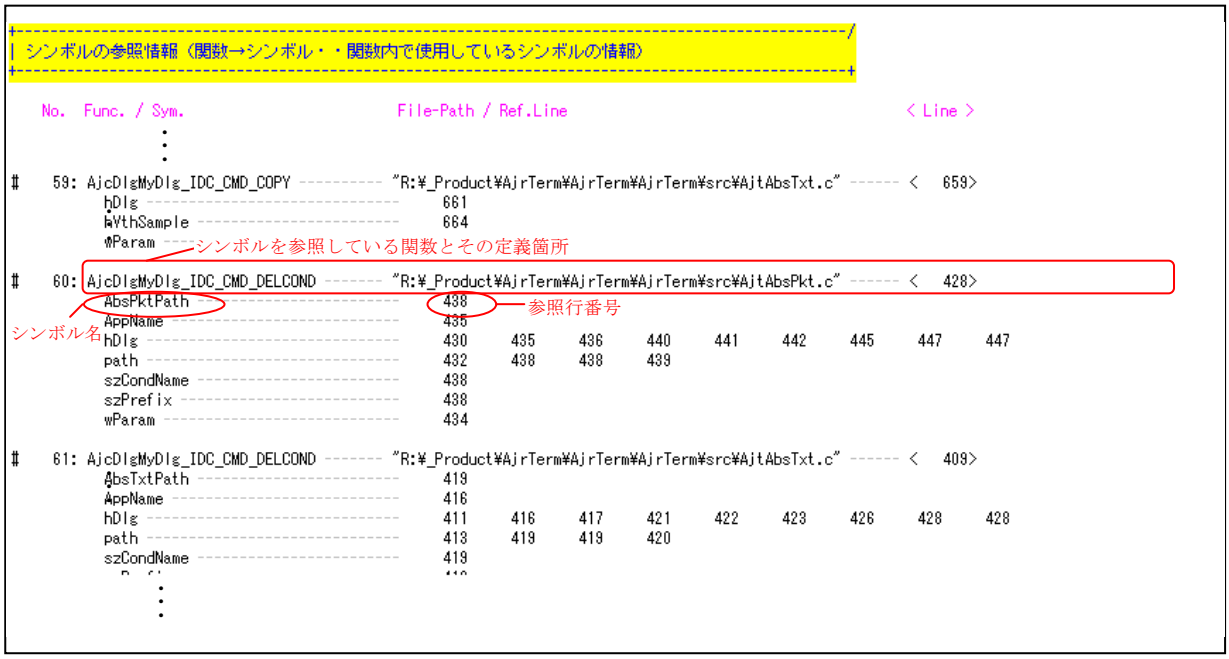
9.1. シンボル → 関数

シンボルをリストアップし、当該シンボルを使用している関数と参照行番号を示します。
「シンボル → 関数」の表示例を以下に示します。



9.2. 関数 → シンボル

関数単位で、シンボルの相互参照を表示します。
「関数 → シンボル」の表示例を以下に示します。



9.3. 呼び出し構造 → シンボル

関数の呼び出し構造の単位で、シンボルの相互参照を表示します。
つまり、当該最上位レベルの関数呼び出し構造に含まれる、全ての関数についてシンボルの相互参照を表示します。
「最上位関数 → シンボル」の表示例を以下に示します。

関数呼び出し構造リスト

```
+-------------------------------------------------------------------------------------------/
| 関数の呼び出し構造
+-------------------------------------------------------------------------------------------+

    No.  Structure                          File-Path                                    < Line >
           :
           :

#    21: AjcDlgMain_IDC_CMD_SETPORT -------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" <  1983>
         +- ShowLineInfo ------------------------------------------ "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" < 3155>
         +- ShowRxError ------------------------------------------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" < 3139>

           :
           :
```

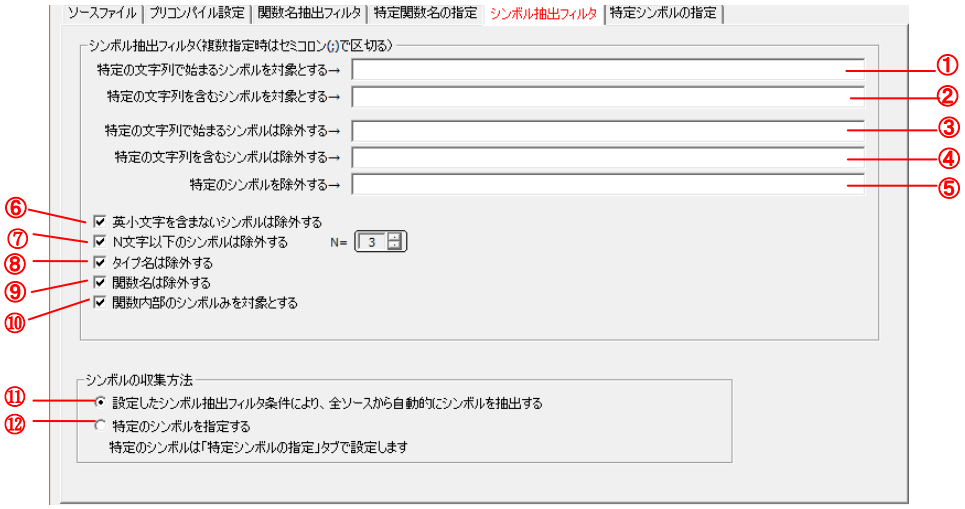
AjcDlgMain_IDC_CMD_SETPORT()の関数呼び出し構造で示される全関数について
シンボルの相互参照を表示

シンボル相互参照リスト

```
+-------------------------------------------------------------------------------------------/
| シンボルの参照情報（上位関数→シンボル・・上位関数下で使用しているシンボルの情報）
+-------------------------------------------------------------------------------------------+

    No.  Func. / Sym.                       File-Path / Ref-Line                         < Line >
           :
           :                                                  最上位レベル関数名とその定義箇所
#    21: AjcDlgMain_IDC_CMD_SETPORT -------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" -------- <  1983>
           hScp ------------------------------ 1986
           hWndBack -------------------------- 1986
           TRUE ------------------------------ 1991
           wParam ---------------------------- 1985
         ShowLineInfo ----------------------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" --------- < 3155>
           BaudRate -------------------------- 3183
           ByteSize -------------------------- 3185 ― 参照行番号
           fParity ---- シンボル名 ----------- 3169
           hScp ------------------------------ 3161   3162   3168   3182   3190   3191
           hWndSts --------------------------- 3163   3166   3187   3190   3191
           LPARAM ---------------------------- 3163   3166   3187   3190   3191   下位レベル関数名とその定義箇所
           NULL ------------------------------ 3168
           Parity ---------------------------- 3170
           SB_SETTEXT ------------------------ 3163   3166   3187   3190   3191
           StopBits -------------------------- 3186   3186
           szParity -------------------------- 3158   3171   3172   3173   3174   3175   3176   3180   3184
         ShowRxError ------------------------ "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" --------- < 3139>
           ErrCntBR -------------------------- 3144
           ErrCntFE -------------------------- 3144
           ErrCntIO -------------------------- 3144
           ErrCntOE -------------------------- 3144
           ErrCntPE -------------------------- 3144
           hScp ------------------------------ 3143
           hWndSts --------------------------- 3145   3148
           LPARAM ---------------------------- 3145   3148
           SB_SETTEXT ------------------------ 3145   3148
             :
             :
```

10. シンボル抽出フィルタ

メインウインドで「シンボル抽出フィルタ」タブを選択すると、以下の画面が表示されます。



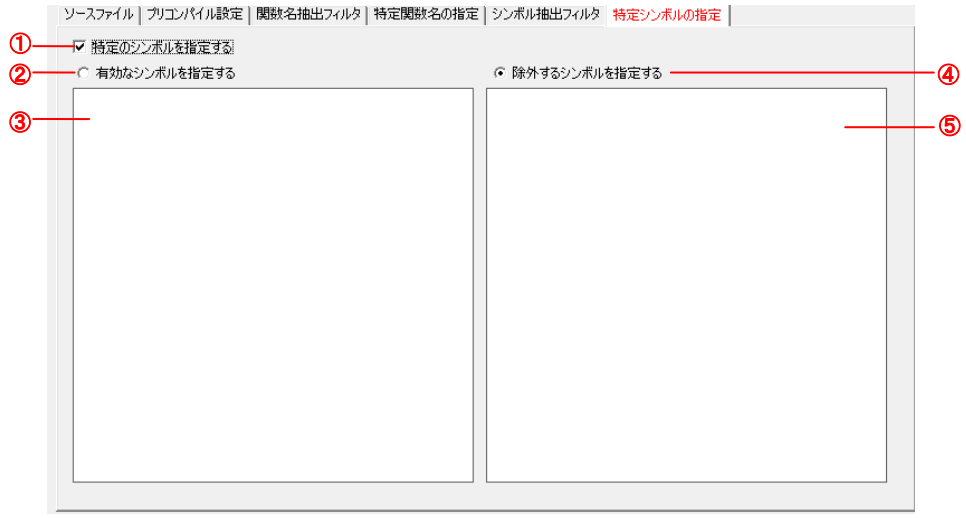
設定する内容は、以下のとおりです。

#	内　容
①	ここで指定した文字列で始まるシンボルを収集します。
②	ここで指定した文字列を含むシンボルを収集します。
③	ここで指定した文字列で始まるシンボルを除外します。
④	ここで指定した文字列を含むるシンボルを除外します。
⑤	ここで指定したシンボル群を除外します
⑥	英小文字（'a'～'z'）が含まれないシンボルは除外します。
⑦	Ｎ文字以下のシンボルを除外します。
⑧	タイプ名（typedef で定義した名称，構造体名(struct)，共用体名(union)や列挙名(enum) ）を除外します。
⑨	関数名を除外します。
⑩	関数内部のシンボル参照だけを対象とします。
⑪	入力した全ソースファイルから、① ～ ⑩ の条件に従い、シンボルの抽出を行います。
⑫	特定のシンボルを指定します。特定のシンボルは「特定シンボルの指定」タブで設定します。

①～⑤の項目で、複数の文字列を指定する場合は、セミコロン(;)で区切って指定します。
指定した文字列の前後の空白は無視されます。

10.1. 特定シンボル設定

メインウインドで「特定シンボルの設定」タブを選択すると、以下の画面が表示されます。



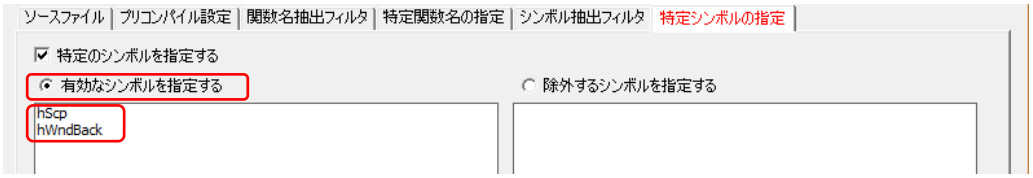
設定内容は、以下のとおりです。

#	内　容
①	このタブで特定の関数名を指定する場合にチェックします。 （「シンボル抽出フィルタ」タブで、「特定のシンボルを指定する」をチェックしたのと同様）
②	特定のシンボル群を対象とします。 （③で指定したシンボル群だけで、シンボルの相互参照を作成します）
③	②をチェックした場合に、対象とする特定のシンボル群を列挙します。 ②をチェックし、何も指定しない場合は「対象シンボルなし」となります。
④	特定のシンボル群を除外します。 （⑤で指定したシンボル群を、シンボルの相互参照から除外します）
⑤	④をチェックした場合に、除外する特定のシンボル群を列挙します。 ④をチェックし、何も指定しない場合は、全てのシンボルが対象となります。

②と④は、いずれか一方だけ設定可能です。

特定のシンボルを指定することにより、「シンボルのクロスリファレンス」に、指定したシンボルの参照情報だけを表示できます。

例えば、最上位関数下の全てのサブ関数において、シンボル「 hScp 」と「 hWndBack 」の参照情報を表示する場合、「特定シンボルの設定」で、「特定のシンボルを指定する」と「有効なシンボルを指定する」をチェックし、当該リストボックスに「 hScp 」と「 hWndBack 」を追加します。



「実行」ボタンを押すと、指定したシンボルに限定した。シンボルの参照情報を出力します。

```
+-------------------------------------------------------------------------------------------/
| シンボルの参照情報（上位関数→シンボル・・上位関数下で使用しているシンボルの情報）
+-------------------------------------------------------------------------------------------+

    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
           ・・・・・
```

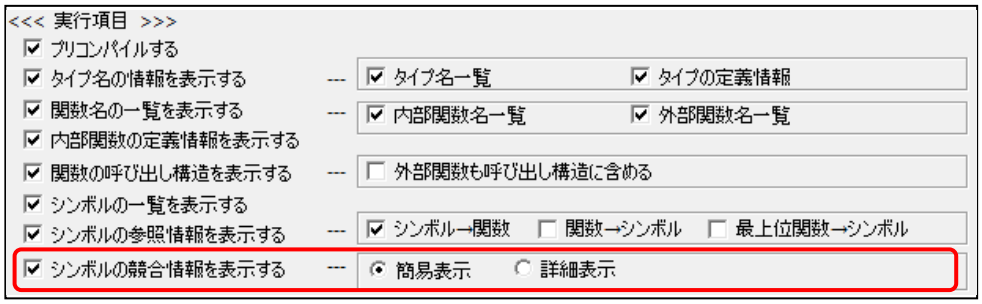
上記出力から、最上位関数「AjcDlgMain_IDC_CMD_SETPORT」で「 hScp 」と「 hWndBack 」を参照し、そのサブ関数「ShowLineInfo」と「ShowRxError」で「 hScp 」が参照されていることが分かります。

尚、関数「AjcDlgMain_IDC_CMD_SETPORT」の呼び出し構造は、以下のようになっています。

```
+-------------------------------------------------------------------------------------------/
| 関数の呼び出し構造
+-------------------------------------------------------------------------------------------+

    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. シンボルの競合情報

メインウインドで「シンボルの競合情報を表示する」をチェックすると、各最上位レベル下の関数呼び出し構造において、複数の最上位レベル下の関数呼び出し構造で参照／更新されているシンボルをリストアップします。



例えば、次のような関数の呼び出し構造がある場合、４つの最上位関数「ist_ERI9」「ist_RXI9」「ist_TXI9」「MainProc」下の複数の関数構造で参照されているシンボルをリストアップします。

```
ist_ERI9
 +- AjrRingPutByte

 ist_RXI9
 +- AjrRingPutByte

 ist_TXI9
 +- AjrRingGetByte

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

※ 最上位関数は、プログラム内からのいずれからも呼び出されていない関数（割り込みハンドラや、タスク等）を意味します。
但し、関数ポインタで動的に呼び出している関数や、未使用の関数も最上位関数として認識されます。

左記の例では、「MainProc」は、メイン処理を、「ist_*XXXX*」は割り込みハンドラを意味しています。

シンボルの競合情報を表示（簡易表示）

```
+-------------------------------------------------------------------------------------------------/
| シンボル競合情報（上位関数とそのサブ関数群のグループ間で競合するシンボル情報）
+-------------------------------------------------------------------------------------------------+

  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」は、「MainProc」と「ist_TXI9」の２つの関数呼び出し構造から参照されていることを意味します。
「*n」は、当該シンボルを更新している個所の個数を示します。
但し、シンボルの更新情報は、静的に（明示的に）シンボルを更新している場合だけを認識します。
ポインタ等で、動的に更新している場合は、シンボルの更新を認識できません。
シンボルの更新が無い場合（あるいは更新を認識できない場合）は「- 」を表示します。

さらに「詳細表示」をチェックした場合は、シンボルを参照／更新している（最上位関数下の）関数名も表示します。

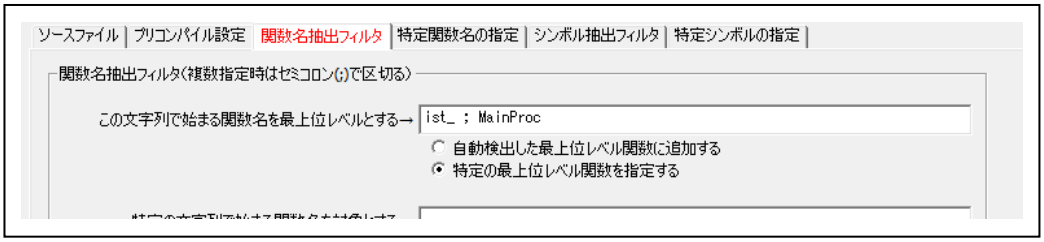
```
+-------------------------------------------------------------------------------------------------/
| シンボル競合情報（上位関数とそのサブ関数群のグループ間で競合するシンボル情報）
+-------------------------------------------------------------------------------------------------+

  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>
```

「MainProc」下の「DebugInit」「DebugPrintF」「DebugPutS」と「DebugSendPacket」の４つの関数で「fTxBusy」を参照／更新していることを示します。

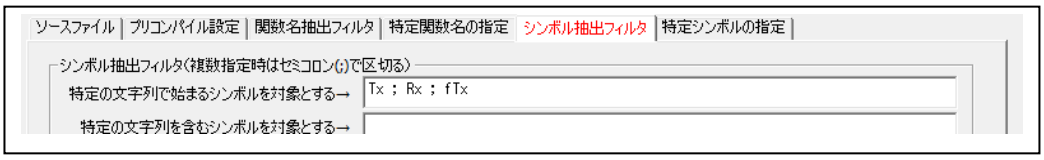
※関数名やシンボル名は、「関数名抽出フィルタ」「特定関数名の指定」「シンボル抽出フィルタ」「特定シンボルの指定」で必要な名称だけを指定することができます。

上記でで示した出力例では、以下のような設定を行っています。

関数名抽出フィルタの設定

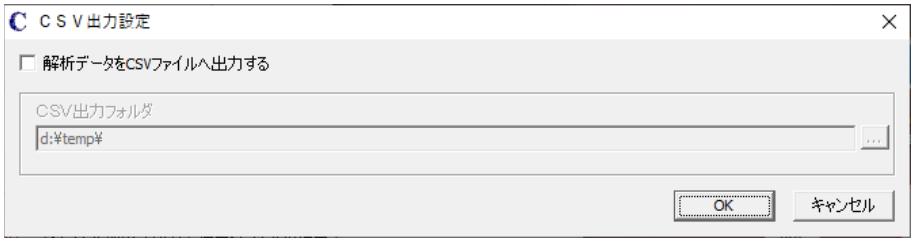


シンボル抽出フィルタの設定



12. ＣＳＶ出力

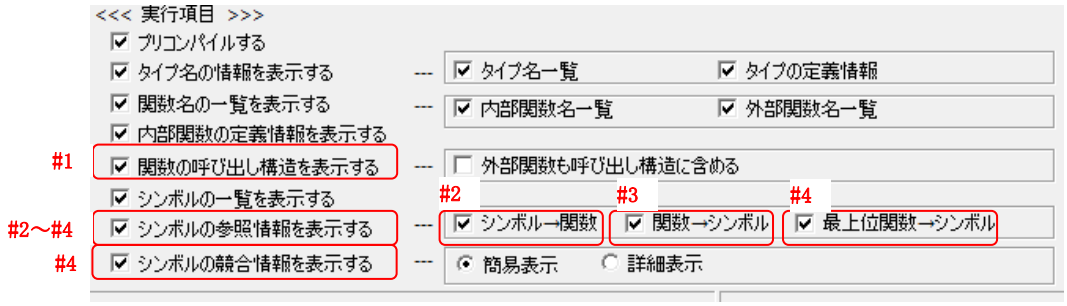
「設定」→「ＣＳＶ出力設定」メニューで、以下のダイアログボックスが表示されます。



ここで、「解析データをＣＳＶファイルへ出力する」をチェックすると、データをＣＳＶファイルへ出力します。
「ＣＳＶ出力フォルダ」は出力先のフォルダのパスを設定します。
ＣＳＶファイルの名称は固定で、以下の通りです。

#	CSV ファイル名	内容
1	CallInfo.csv	関数の定義と呼び出し情報
2	XRefBySym.csv	相互参照情報（シンボル → 関数）
3	XRefByFunc.csv	相互参照情報（関数 → シンボル）
4	XRefByTree.csv	相互参照情報（関数の呼び出し構造 → シンボル）

これらのＣＳＶファイルは、メインウインドで以下の赤い部分をチェックした場合に出力されます。



CSV ファイルに出力される情報は以下の通りです。(各情報がカンマ (,) で区切られて出力されます)

関数の定義と呼び出し情報 - CallInfo.csv

#	内容	
1	順序番号	1, 2, 3 ・・・
2	最上位関数フラグ	1：最上位関数, 0：その他
3	関数が定義されているファイルのパス名	
4	関数名	
5	関数名が定義されている位置 (行番号)	関数名の先頭位置
6	〃 (桁位置)	
7	関数の開始位置 (行番号)	関数の始まりを示す ‘{‘
8	〃 (桁位置)	
9	関数の終了位置 (行番号)	関数の終わりを示す ‘}‘
10	〃 (桁位置)	
11	関数が呼び出しているサブ関数名	呼び出している関数が無い場合、 この項目はありません
12	・・・(呼び出している関数の個数だけ続く)	

相互参照情報 (シンボル → 関数) - XRefBySym.csv

#	内容	
1	シンボル名	
2	シンボルを参照している関数のパス名	
3	シンボルを参照している関数名	
4	シンボルを参照している位置 (行番号)	
5	〃 (桁位置)	
6	シンボルの更新フラグ	1：更新, 0：参照

相互参照情報 (関数 → シンボル) - XRefByFunc.csv

#	内容	
1	関数名	
2	シンボルを参照している関数のパス名	
3	シンボル名	
4	シンボルを参照している位置 (行番号)	
5	〃 (桁位置)	
6	シンボルの更新フラグ	1：更新, 0：参照

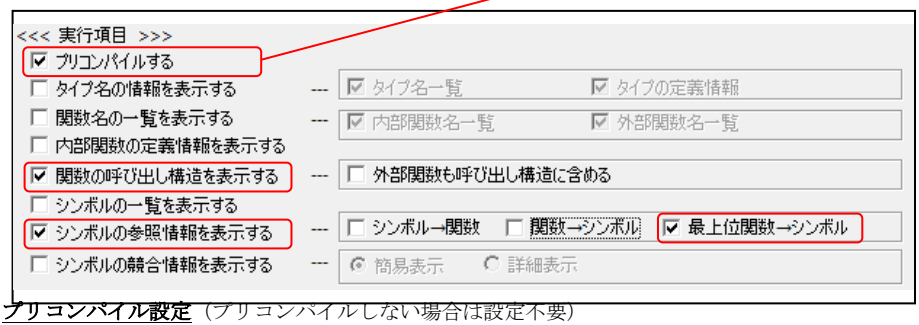
相互参照情報 (関数の呼び出し構造 → シンボル) - XRefByTree.csv

#	内容	
1	最上位関数の番号	ランダムな昇順値「#nnn」
2	最上位関数名	
3	シンボルを参照しているサブ関数名	
4	シンボルを参照しているサブ関数のパス名	サブ関数が最上位レベルの関数である場合は、「..#nnn」で最上位関数の番号を示し、以降の情報は無い
5	シンボル名	
6	シンボルを参照している位置 (行番号)	
7	〃 (桁位置)	
8	シンボルの更新フラグ	1：更新, 0：参照

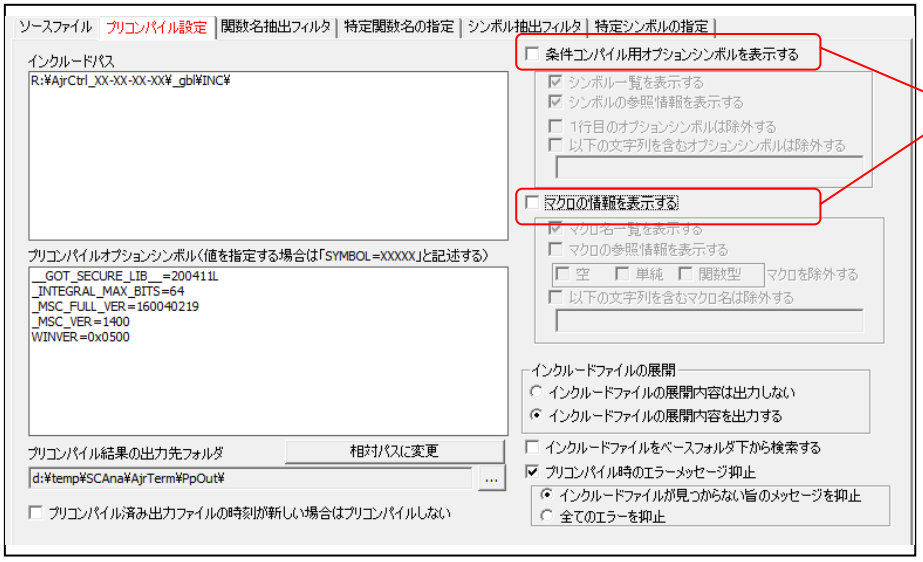
13. 使用例

13.1. 特定の関数とそのサブ関数で使用しているシンボルを調べる

出力設定（以下の項目をチェック）

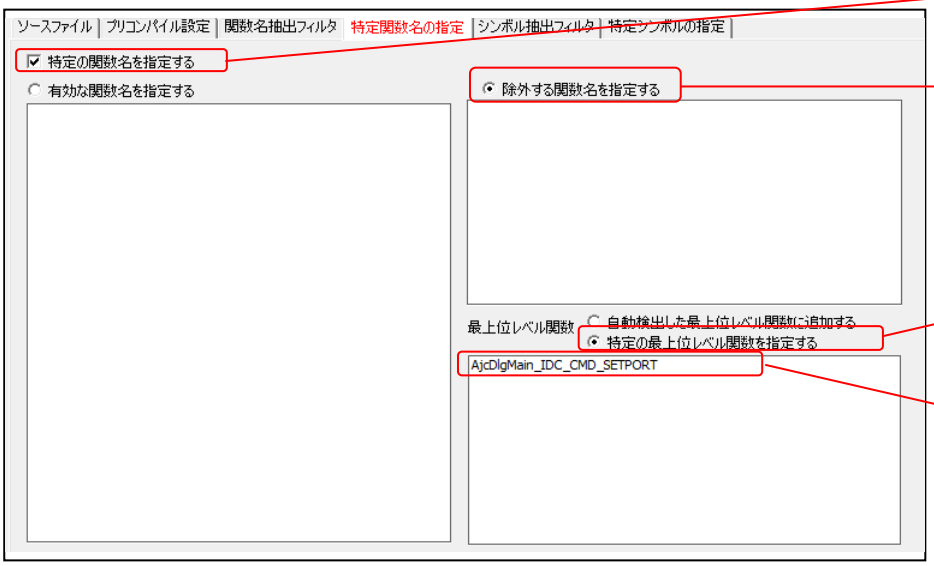


プリコンパイル設定（プリコンパイルしない場合は設定不要）



必要のない出力項目は抑止する

特定関数名の指定



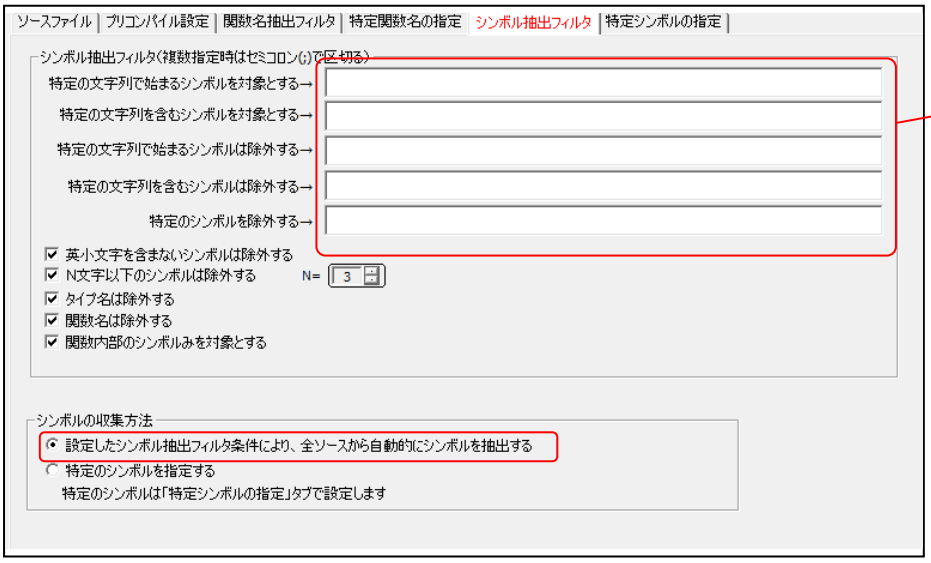
「特定の関数名を指定する」をチェックする

除外する関数名を指定する
（特に除外する関数が無い場合は何も指定しない）

「特定の最上位レベル関数を指定する」をチェックする

調査する関数を指定する。
（複数指定可）

シンボル抽出フィルタ



有効とするシンボルや除外するシンボルがあれば指定する。
（本例では何も指定しない）

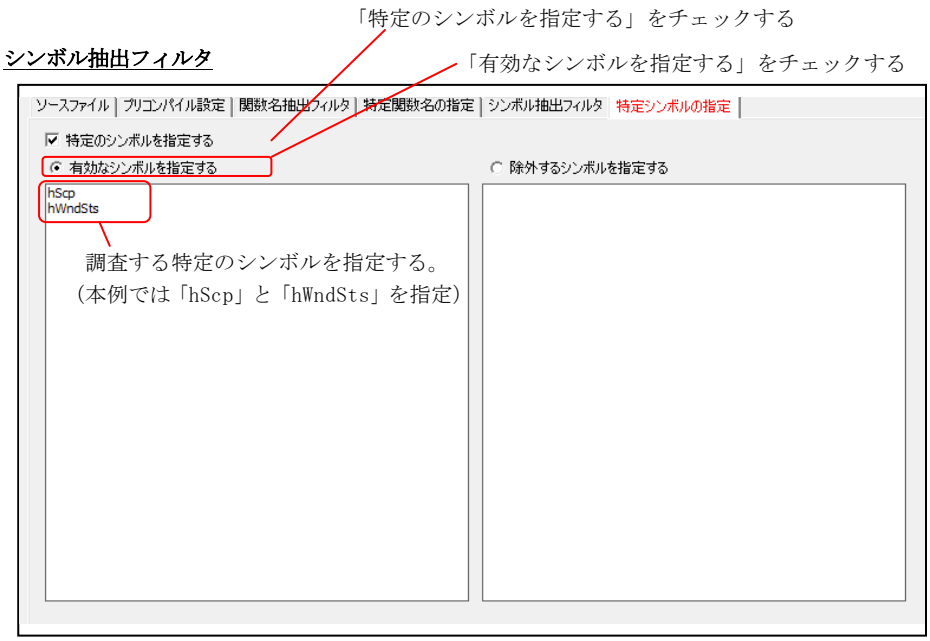
実行結果（指定した関数「AjcDlgMain_IDC_CMD_SETPORT()」の呼び出し構造と、その構造で使用しているシンボルを表示）

```
+-------------------------------------------------------------------------------------------------/
| 関数の呼び出し構造
+-------------------------------------------------------------------------------------------------+

    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>

+-------------------------------------------------------------------------------------------------/
| シンボルの参照情報（上位関数→シンボル・・上位関数下で使用しているシンボルの情報）
+-------------------------------------------------------------------------------------------------+

    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
          wParam ----------------------------    2040
        ShowLineInfo ----------------------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" -------- <  3276>
          BaudRate --------------------------    3304
          ByteSize --------------------------    3306
          fParity ---------------------------    3290
          hScp ------------------------------    3282    3283    3289    3303    3311    3312
          hWndSts ---------------------------    3284    3287    3308    3311    3312
          Parity ----------------------------    3291
          StopBits --------------------------    3307    3307
          szParity --------------------------    3279    3292    3293    3294    3295    3296    3297    3301    3305
        ShowRxError ------------------------ "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" -------- <  3260>
          ErrCntBR --------------------------    3265
          ErrCntFE --------------------------    3265
          ErrCntIO --------------------------    3265
          ErrCntOE --------------------------    3265
          ErrCntPE --------------------------    3265
          hScp ------------------------------    3264
          hWndSts ---------------------------    3266    3269
```

13.2. さらにシンボルを特定する

前述の例（特定の関数とそのサブ関数で使用しているシンボルを調べる）で、さらに特定のシンボルを指定します。

シンボル抽出フィルタ



実行結果（指定したシンボル（「hScp」「hWndSts」）だけについて、参照情報を表示）

```
+-------------------------------------------------------------------------------------------------/
| 関数の呼び出し構造
+-------------------------------------------------------------------------------------------------+

    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>

+-------------------------------------------------------------------------------------------------/
| シンボルの参照情報（上位関数→シンボル・・上位関数下で使用しているシンボルの情報）
+-------------------------------------------------------------------------------------------------+

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


#    21: AjcDlgMain_IDC_CMD_SETPORT -------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" -------- <  2038>
           hScp ------------------------------    2041
         ShowLineInfo ---------------------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" -------- <  3276>
           hScp ------------------------------    3282    3283    3289    3303    3311    3312
           hWndSts ---------------------------    3284    3287    3308    3311    3312
         ShowRxError ----------------------- "R:¥_Product¥AjrTerm¥AjrTerm¥AjrTerm¥src¥AjtMain.c" -------- <  3260>
           hScp ------------------------------    3264
           hWndSts ---------------------------    3266    3269
```

14. 実行可能な Windows バージョン

ＳＣＡＮＡは、ＰＣ用 Windows 7 以降の全バージョン(Windows7, Windows8, Windows8.1, Windows10・・)における 32 ビット・プラットフォーム(x86)と、64 ビット・プラットフォーム(x64)で動作可能です。
実行ファイルは、以下のとおりです。

種別	実行ファイル	備考
Windows (32Bit)	SCAna32.exe	32Bit/64Bit Windows のいずれでも実行可能
Windows (64Bit)	SCAna64.exe	64Bit Windows でのみ実行可能

15. 免責事項

プログラムの使用にあたっては、以下の点にご注意ください。

・ＳＣＡＮＡの著作権は、製作者に帰属します。
・ＳＣＡＮＡは、どのような場合においても利用者の責任において利用してください。
運用結果については、一切責任を負いかねます。

16. 問い合わせ先

本ソフトウェアに関するお問い合わせは、件名の先頭を「Ajara:」として、以下のメールアドレスに送付してください。

xxxajarakojara@kk.email.ne.jpxxx

[注] 先頭と末尾の「xxx」は削除してください。
「@」は、全角となっていますので、半角に訂正してください。

17. 変更履歴

17.1. Ver 1.3.2.0

1） プリコンパイルで、#if / #elif 条件式のマクロ展開形を表示するようにし、原文をコメント表示するようにした。

入力

```
#define  A  5
#if  defined(X) && A == 1
int b = XXX;
#endif
```

出力（変更前）

```
// #define A  5
// #if defined(X)&&A==1          // --> FALSE
//- int b=XXX;
// #endif                        // --> 2
```

出力（変更後）

```
// #define A  5
// #if  0 && 5 == 1 /* (#if defined(X) && A == 1) */  // --> FALSE
//- int b = XXX;
// #endif                        // --> 2
```

2） プリコンパイル出力が見やすいように、トークン間に空白を挿入するようにした。

入力

```
if (a == 1 && b == 2 && !(c & 1)) {
```

出力（変更前）

```
if(a==1&&b== 2&&!(c&1)){
```

出力（変更後）

```
if (a == 1 && b == 2 && !(c & 1)) {
```

3）プリコンパイルで、インクルードファイルのマクロ指定に対応しました。

入力

```
#define  INC  <stdio.h>
#include INC
```

出力（変更前）

```
//  #define INC <stdio.h>
//  #include INC
```

#include 構文エラー

出力（変更後）

```
//  #define INC <stdio.h>
//  #include <stdio.h>
```

4）プリコンパイルで、 #undef で削除したマクロ表示が消える不具合修正（#undef したマクロがコメント出力されない）

入力

```
#define XXX 123
int b = XXX;
#undef  XXX
```

出力（変更前）

```

    int b = 123;
//  #undef  XXX
```

出力（変更後）

```
//  #define XXX 123
    int b = 123;
//  #undef  XXX
```

5）プリコンパイルで、非生成部分の処理系依存プリプロセス（#pragma や#error 等）が表示されない不具合修正

入力

```
#if 1
      #pragma ABC
#else
      #pragma DEF
#endif
```

出力（変更前）

```
//  #if  1 /* (#if 1) */ // --> TRUE
        #pragma  ABC
//  #else                 // --> FALSE
//
//  #endif                // --> 1
```

出力（変更後）

```
//  #if  1 /* (#if 1) */ // --> TRUE
        #pragma ABC
//  #else                 // --> FALSE
//-     #pragma DEF
//  #endif                // --> 1
```

6）プリコンパイルで、処理系依存プリプロセス（#pragma や#error 等）のオペランドをマクロで指定し、展開形が原文より長くなる場合バッファオーバーランとなる不具合修正 (#pragma のオペランドがマクロでない場合は問題なし。これにより発生する現象は不定)

```
#define X warning(disable:1234)
#pragma X
```

「X」→「warning(disable:1234)」に変換されることにより、変換後のテキストが原文よりも長くなる場合にバッファオーバランが発生

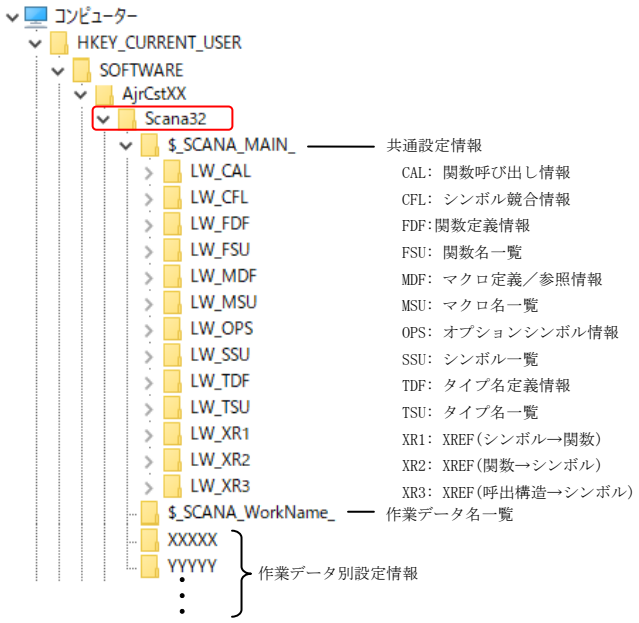
17.2. Ver 1.4.0.0

フリープログラム化（ライセンス設定に関する記述を削除）

17.3. Ver 1.6.0.0

ライブラリ変更（→AjrCstXX(V1600(UNICODE 版)))
保守の為、ライブラリを最新の UNICODE 版に変更も、本アプリはバイト文字アプリのままで、機能的な変更は無い。

バージョン 1.6.0.0 以降は、設定な用をレジストリの「HKEY_CURRENT_USER¥SOFTWARE¥**AjrCstXX**¥Scana32/64・・」に記録します。



※バージョン 1.6.0.0 より以前の場合、「HKEY_CURRENT_USER¥SOFTWARE¥**AjrCt132/64**¥Scana32/64」に記録していました。
バージョン 1.6.0.0 以降のＳＣＡＮＡを起動した場合、最初１回だけ旧設定情報を、新しい設定情報へコンバートします。
つまり、バージョン 1.6.0.0 以降のＳＣＡＮＡを起動した場合、最初は旧設定内容が引き継がれますが、その後は旧バージョンのＳＣＡＮＡと新バージョンのＳＣＡＮＡの設定内容は別々となります。