PharmaSUG2011 - Paper AD19

Symbol Table Generator
(New and Improved)

Jim Johnson, JKL Consulting, North Wales, PA

ABSTRACT

In Seattle at the PharmaSUG 2000 meeting the Symbol Table Generator was first presented. The original program was simple, and so was the output: a line numbered listing of selected program followed by a symbol table which is a cross reference of all the variables and keywords used in the program. In the New and Improved version comment limitations have been overcome and the strengths of Proc Report have been employed. The line numbered listing is much more intelligent and can provide up to three different types of program nesting. Other new features include optional suppression of common keywords from the symbol table, tab processing, simple error checking, and even suppressing the symbol table altogether.

INTRODUCTION

Have you ever written a lengthy program, printed it, and tried to find where in the program that a particular variable or two were used? Do you remember sitting there with your printout, highlighter, and program editor, repeatedly using the editor’s find feature, then trying to locate the reference in the printed copy so you can highlight it? Or, maybe instead of tediously looking for all the references, you simply used the editor’s change feature and changed all occurrences of a variable to another spelling. Then when you ran the program you found that you had changed things that you did not expect like TITLE statements, data set names, or keywords. What you really need is a Symbol Table.

When something is 'new and improved', which is it? If it's new, then there has never been anything before it. If it's improved, then there must have been something before it and couldn't be new. With this version of the Symbol Table Generator, it is both new and improved. It existed previously and has been improved, and there are a lot of new features that never existed before making it look and act like a completely new product.

“There are a hundred different ways to do everything in SAS®.” The techniques shown in this paper are only one way to do things. There probably are many more. These techniques have been chosen to demonstrate specific functions of SAS and / or the Symbol Table Generator, you are encouraged to explore other means of doing the same thing.

All examples given are based on the Microsoft® Windows® environment using SAS system for Personal Computers® version 9.2.

The remainder of this paper will discuss the New and Improved Symbol Table Generator as if it had never existed before.

WHAT IS A TOKEN?

The *SAS Macro Language Reference* defines a token as follows:

> When the SAS System processes a program, a component called the word scanner reads the program, character by character, and groups the characters into words. These words are referred to as *TOKENS*.

Tokens, therefore, are the individual words (keywords, variables, operators, and constants) that exist in your program.

WHAT IS A SYMBOL TABLE?

A symbol table is a cross reference of tokens, or symbols, providing the line numbers where each is used. The example below tells you that the token *print* (which may be a variable, keyword, or text literal) is used on lines 10, 22, and 266. Now you know exactly where to go in your program to review your code.

```
alpha   5, 17, 155, 210
beta    6, 22, 211, 234, 301
print   10, 22, 266
sort    45, 101
```

TYPES OF COMMENTS
There are three types of comments supported by the SAS System.

ASTERISK - SEMICOLON
These comments start with an asterisk (*) and end with a semicolon (;). The program fragement below shows several examples of asterisk-semicolon comments. The first line shows a comment that starts and ends on one line. The second line starts a comment that ends on the third line. The fourth line contains two SAS program statements, the first is not commented, the second is. The last line of code is not commented. SAS knows that there is no comment in the fifth line, though it contains an asterisk and semicolon, because SAS interprets the asterisk *in context*. SAS knows that use of the asterisk is not meant to be a comment.

```
* a = means + 1 ;
* proc print
data = final;
cnt + 1; * index = cnt * 2;
raise = salary * 0.05;
```

SLASH STAR – STAR SLASH
These comments start with a slash star (/*) and end with a star slash (*/). The code below shows the same examples of code using the slash star – star slash comments.

```
/* a = means + 1; */
/* proc print
data = final; */
cnt + 1; /* index = cnt * 2; */
```

MACRO COMMENTS
These comments start with a percent sign asterisk (%*) and end with a semicolon (;). The Symbol Table Generator treats these comments exactly the same way as asterisk – semicolon comments. Any references in this paper to asterisk – semicolon comments applies equally to macro comments.

COMMENT USAGE PERMUTATIONS
Comment types can be mixed. Asterisk – semicolon comments can contain slash star – star slash comments, and slash star – star slash comments can contain asterisk – semicolon comments. Examples of these are shown below.

```
* a = a + 1 /* add 1 to a */ ;
/*a = a + 1; *add 1 to a; */
```

On any one line of program code, there are four possible comment scenarios for each type of comment:

1. Comment starts and stops on the given line
2. Comment starts but does not stop on the given line
3. Comment stops but does not start on the given line
4. Comment is ongoing, neither starts nor stops on the given line

Each of these possibilities is handled by the Symbol Table Generator.

WHY IS THIS IMPORTANT?
Since we all use ample comments in our code, it would be desirable to exclude those comments from the symbol table. Therefore, the Symbol Table Generator ignores all comments. Below are three lines of code. The first will be tokenized, the second will be ignored, and third line will tokenize the first statement and ignore the second. SAS understands the second statement on the third line is a comment, because it interprets the asterisk *in context*. The Symbol Table Generator is sophisticated enough to also recognize the use of the asterisk.

```
raise = salary * 0.05;
*a = mean + 1;
cnt + 1; * index = cnt * 2;
```

NESTING TYPES
BASE *DO / END* NESTING
The Symbol Table Generator can also interpret *do*s and *end*s in context. It can tell the difference between a *do* used as a keyword and a *do* used as a variable. It can also tell if an *end* is connected to a *do* or other keywords such as a base SAS *select* statement or an SQL procedure *case* statement, or if it is just another variable.

MACRO *%DO / %END* NESTING
As with base *do* / *end* nesting, the Symbol Table Generator can also track %*do* and %*end* nesting.

MACRO NESTING
Another feature similar to *do* / *end* nesting is macro nesting. This is the nesting of *%macro* and *%mend* statements to help you keep track of how deeply your macros are nested.

```
  Line  nesting
Number  B  M  L  Original Code
-----------------------------------------------------------------------------------------------
  1740  3  1  1                  start_sub = start_sub + col_segmentlength;
  1741  3  1  1               end;
  1742  2  1  1            end;
  1743  1  1  1        %end;
  1744  1  0  1
  1745  1  0  1      end;
  1746  0  0  1
  1747  0  0  1  %mend;
  1748  0  0  0
```

Nesting levels show base (B) *do* / *end* nesting, macro (M) %*do* / %*end* nesting, and macro (L) nesting. In the example above, notice that the B level, the base *do* / *end* nesting level reduces when each *end* is encountered, and the M level, the macro %*do* / %*end* nesting, reduces when each *%end* is encountered, and the L level, the *%macro* nesting level reduces when each *%mend* is encountered.

HOW DOES THE SYMBOL TABLE GENERATOR WORK?

The Symbol Table Generator is invoked by a simple call:

```
%symbol(fully qualified name of a SAS program);

%symbol(c:\sas\program\test.sas);
```

A line numbered program listing will be generated. Every page will have a header indicating the name of the program processed, the date and time the listing was produced, and a page number. Lines of program code that are greater than the LINESIZE provided in the SAS session (less space for the line number and nesting) will wrap to the next line.

The input program is then scanned, and comments are detected and removed.

The remaining code is then broken into tokens. Tokens are separated by common delimiters
}[{]!@#$^*)(+-=/:;\|'?><,.~" *space*
Note that included in that list are the single and double quote and semicolon.

EXCLUDED from the list are the numerals 0-9, allowing for variables like b6, date1, or table2a. Numeric constants like 6, 5.1, 2.54 are programmatically eliminated from the symbol table. Also EXCLUDED from the delimiter list are the & and % which allows for macro commands and macro variables. The underscore (_) is also excluded from the delimiter list since it is a valid component of SAS variable names.

The tokens are collected in a SAS data set along with the line number on which it is used. The data set contains one observation for each token and line number. If a token appears twice on a line of code, then there will be two observations on the data set. This data set is processed into a smaller set which contains one observation per token and **all** the line numbers on which it appears. Then the final SAS data set is printed.

FEATURES

SUPPRESS COMMON KEYWORDS FROM SYMBOL TABLE
The larger the program, the more common keywords like *if, then, else, do, end, put,* and *data* will be used. These common keywords are tokenized with the rest of the program and reported in the Symbol Table just like all other tokens. With larger programs, the Symbol Table may go on for a full page or more reporting every place your program used the keyword *if*. By suppressing the common keywords from the Symbol Table, it can make it easier for you to scan through the table and certainly will shorten the report for printing.

This feature comes with drawbacks. The Symbol Table Generator has no way of knowing if you have used symbols like *if* or *where*, or most of the other common keywords, as keywords or variables. While the program can tell if certain keywords such as *do, end,* and *data* have been used as keywords or variables, it would be impossible to determine the context of all the common keywords used in SAS. To write a program that dynamic would pretty much mean writing a program as large as the SAS system itself. So, use this feature with caution.

EMBEDDED TABS CHANGED TO SPACES
Many programmers use the tab key to indent their program code for readability. It is also possible to set preferences in the SAS enhanced editor to automatically indent the next line of code and to use tabs or spaces for that indentation. As a result, many programs contain tab characters. The Symbol Table Generator reports these tabs as unprintable characters, and depending on the printer attached to the computer, they may appear as rectangles or a strange print alignment of the code and symbol table. The tabs will also be interpreted as part of the following symbol causing it to sort away from the rest of the same symbols.

The code below was written using tabs.

```
data _null_;
    do a = 1 to 10;
        do b = 1 to 5;
            put a=   b=;
            end;
        end;
run;
```

The tabs are not visible, so the code has been shown again demonstrating where the tabs have been used.

```
data _null_;
<tab>do a = 1 to 10;
<tab><tab>do b = 1 to 5;
<tab><tab><tab>put a=   b=;
<tab><tab><tab>end;
<tab><tab>end;
run;
```

Below is an example of the line listing output with the tabs in place. In this example, the tabs, now unprintable characters, have essentially been suppressed, making the program code appear to have no indentation.

```
  Line nesting
Number  B   M  Original Code
     1  0   0  data _null_;
     2  0   0  do a = 1 to 10;
     3  1   0  do b = 1 to 5;
     4  2   0  put a=   b=;
     5  2   0  end;
     6  1   0  end;
     7  0   0  run;
```

The symbol table represents the unprintable characters slightly differently, causing the line numbers to appear out of their column seemingly related to the number of tabs preceding the symbol. Notice that the symbols do not appear in alphabetic order and that the symbols *do* and *end* are listed twice. This is all because the tabs have been attached to the symbols and that ***<tab>****do* is different than ***<tab><tab>****do.*

```
Symbol                   Line Numbers
end                      5

put                      4

do                        3

end                       6

do                         2

a                           2,4

b                           3,4

data                        1

run                         7

to                          2,3

_null_                      1
```

The Symbol Table Generator automatically detects and converts tabs into four spaces. Four spaces is the default size of a tab in SAS. While this value can be set to anything you like in the enhanced editor, the Symbol Table Generator has no idea what setting was used for the generation of the program code it is analyzing. Therefore, it simply uses the default value of four spaces. This will align the printed code more suitably than using the tabs did, and the symbols preceded by a tab now appear the same as those that are not, thus keeping all the same symbols clustered together in the Symbol Table.

The Symbol Table Generator will produce the output below from the sample source file.

```
  Line nesting
Number  B   M  Original Code
     1  0   0  data _null_;
     2  0   0      do a = 1 to 10;
     3  1   0          do b = 1 to 5;
     4  2   0              put a=   b=;
     5  2   0              end;
     6  1   0          end;
     7  0   0  run;


Symbol                 Line Numbers
a                      2,4

b                      3,4

data                   1

do                     2,3

end                    5,6

put                    4

run                    7

to                     2,3

_null_                 1
```

SUPPRESS SYMBOL TABLE
There may be times when the Symbol Table is not desired. For instance, very large programs can produce very large Symbol Tables that may be of little use, or if the nesting levels are being used to track down missing or extra *do*s or *end*s, then the Symbol Table would be of no use at all. The Symbol Table Generator allows you to turn the Symbol Table off and just generate the line numbered listing of the original program code.

NESTING OPTIONS
The nesting described earlier can be controlled with one of four settings available for report generation. You can request no nesting, base *do / end* nesting only, base *do / end* and macro *%do / %end* nesting, or base *do / end*, macro *%do / %end* nesting, and macro nesting.

CHECKS IF FILE EXISTS
The Symbol Table Generator checks that your file exists, and if it does not, indicates that it does not exist with an error message and then aborts in a controlled manner.

INTERPRETING IN CONTEXT
The SAS System interprets your program code *in context*. That is, SAS can tell by the way you are using a keyword or operator whether it is being used as a keyword or variable, or in the case of operators, if they are being used as mathematic or logical operators or as special symbols like comments or concatenation.

The Symbol Table Generator can interpret only certain symbols as keywords or variables. The list is limited and directly related to the calculation of program nesting. The symbols that can be interpreted in context are *data, do, end, select,* and *set*. Each of these has a unique relationship to the determination of nesting. SAS has many other keywords that can also be used as variables, but they do not affect the computation of *do / end* nesting.

The keyword *do* can be used in at least these following statements:

```
do a = 1 to 5;
do while (…);
do until (…);
do over (…);
… then do;
… else do;
when (…) do;  (Base SAS select or SQL procedure case statements)
otherwise do;
```

Identifying whether a *do* is being used as a keyword or variable is not as straight forward as you might think. The examples above represent basically two forms of *do* as a keyword. The first is a *do* at the beginning of a statement, that is, a *do* following a semicolon. The second is a *do* being used at the end of a statement, that is, a *do* preceding a semicolon.

The examples below demonstrate the complications of devising an algorithm by introducing the use of *do* as a variable. These are all valid statements.

```
do do = 1 to 5;
do a = 1 to do;
do while (do = 1);
… then do + 1;
… else 1 + do;
data do;
set do;
do = 1 + do;
```

The same issues exist determining the use of *end* as a keyword or variable.

```
do end = 1 to 5;
do = 1 + end;
end + 1;
set end end=end;
```

Beyond determining if *do* and *end* are used as keywords or variables, there are a number of circumstances that require special handling to keep track of *do* / *end* nesting. These are times when *end* is used without a *do* and are directly related to the keywords *data, select,* and *set*.

***DATA* AND *SET* STATEMENTS**

Data and *set* have to be interpreted in context because *do* and *end* can be used as data set names and do not affect the nesting levels. In addition, *set* offers the *end=* option which does not affect the nesting level.

```
data do;
    set end end=do;
    if do then do;
```

DATA STEP *SELECT* STATEMENT

In the data step the keyword *end* closes the *select* statement and does not affect nesting.

```
data _null_;
   set dsn;
   select (variable);
      when (0) variable + 1;
      otherwise;
      end;
```

For this situation, the Symbol Table Generator first detects the *select* and checks whether it is being used as a keyword or variable. A *select* followed by an open parenthesis or preceded and followed by a semicolon is a keyword. Once it has been determined that *select* is being used as a keyword, the program seeks a corresponding *end* keyword. It is possible to have one or more *do* / *end* combinations within the *select* / *end* construct. The algorithm for tracking *select / end* combinations is different from that which does the *do* / *end* nesting, so no additional complications are created.

```
data _null_;
   set dsn;
   select (variable);
      when (0) do;
```

```
            variable + 1;
            end;
         otherwise;
         end;
```

SQL PROCEDURE *CASE* STATEMENT
In the SQL procedure the keyword *end* closes the *case* statement and does not affect nesting.

```
proc sql;
  select 1 as one
        ,2 as two
        ,case answer_code
          when '1' then 'yes  '
          when '2' then 'no   '
          else 'maybe'
          end as answer_text;
```

The Symbol Table Generator still checks if *case* has been used as a keyword by checking for an open SQL procedure *select* statement. If *case* is being used as a keyword then the program knows there will be an associated *end* keyword to come.

TEMPLATE PROCEDURE *DEFINE* AND *EDIT* STATEMENTS
In the TEMPLATE procedure the keyword *end* closes the *define* statement and the *edit* statement.

```
proc template;
   define style style0;
      replace fonts /
         'FixedFont' = ("Courier",2)
         'BatchFixedFont' = ("SAS Monospace, Courier",2)
         'FixedHeadingFont' = ("Courier",2)
         'FixedStrongFont' = ("Courier",2,Bold)
         'FixedEmphasisFont' = ("Courier",2,Italic)
   end;
```

The Symbol Table Generator is able to tell that *define* or *edit* have been used as a keyword if the TEMPLATE procedure has been started and a step boundary (*data, proc*, or *run* statements) have not been encountered, therefore it should expect an associated *end* keyword.

WHAT ABOUT… ?

What about TITLE and FOOTNOTE statements?
TITLE and FOOTNOTE statements will be tokenized as if they were standard SAS code. The TITLE and FOOTNOTE statements were not excluded from processing since it is possible for them to contain variables.

What about mixed case?
Variables and keywords can appear in a program in upper, lower, or mixed case. Each will be tokenized and accumulated and reported separately because the difference in case may have some meaning or importance to the programmer. In the sort phase of the program an upper case form of the symbol is used. The mixed case form of the symbol will be used to print in the Symbol Table. Thus, it is possible to see what appears to be the same variable listed more than once with different line numbers. Below is an example where the same symbol appears in three different text cases and they are listed adjacent to one another in the symbol table.

Symbol	Line Numbers
LINE	442
Line	447,455,464,474,523
line	174,175,208,446,447,454,455,463,464,473,474,496,501

What about spaces? Or the lack thereof?
Use of spaces is important to the Symbol Table Generator. The base SAS statement **IF A & B** will tokenize, alphabetically, to A, B, and IF. The equivalent base SAS statement **IF A&B** will tokenize, alphabetically, to A&B and IF.

What about the use of /* or */ in the code?
If a line of code such as **IF STRING = '/*' OR STRING = '*/' THEN…** were encountered, it would cause a problem. The Symbol Table Generator would try to find the comment text that is associated with the apparent comment symbol and eliminate it. It was decided that the chances of this happening were very slight and not worth the time it would take to program around it. Therefore is it NOT handled in this version of the Symbol Table Generator.

PRACTICAL USES

DOCUMENTATION TOOL

The Symbol Table Generator can be used as a documentation tool. Validated programs could be run through the Symbol Table Generator and the printed output could be stored with the other program documentation. Future programmers could browse the printed output using the symbol table and perhaps other documentation, such as flow diagrams, as a guide to help trace the use or calculation of a variable.

DEBUGGING TOOL

When the Symbol Table Generator runs against a program with balanced *do / end* nesting, the nesting levels will all be zero at the end of the program listing. The example below shows the output from a balanced program because line 23 show all nesting levels at zero. Line 2 has a base nesting (B) level of zero because the *do* statement is opening a new level and the code following the *do* is at the new level. The closing *end* will have a nesting level that matches the opening *do*. Therefore, the nesting level which begins at line 2 will end when the nesting level returns to zero, at line 22. The nesting level which begins at line 15 will end when the nesting level returns to 2, at line 17. The nesting level which begins at line 3 will end when the nesting level returns to 1, at line 21.

```
  Line nesting
Number  B   M  Original Code
     1  0   0  data _null_;
     2  0   0    do a = 1 to 10;
     3  1   0      do b = 1 to a;
     4  2   0        do c = 1 to b;
     5  3   0          output;
     6  2   0          end;
     7  2   0
     8  2   0        if b = 5 then do;
     9  3   0           d = b;
    10  2   0           end;
    11  2   0        else do;
    12  3   0           d = a;
    13  2   0           end;
    14  2   0        select (a);
    15  2   0          when (1) do;
    16  3   0            d = a;
    17  2   0            end;
    18  2   0          when (5) d = 0;
    19  2   0          otherwise d = .;
    20  2   0          end;  /** select statement **/
    21  1   0        end;
    22  0   0      end;
    23  0   0
```

"THERE WAS 1 UNCLOSED DO BLOCK"

The example below is the same code from above, but line 13 has been commented, thus eliminating an *end* statement. When this program runs SAS will print the message *"There was 1 unclosed DO block"* and abort. The fact that line 23 shows a positive non-zero base nesting (B) level indicates that there are more *do* statement than *end* statements. The nesting level cannot determine whether there is a missing *end* or an extra *do* in the code. The message from SAS suggests you are missing an *end* statement, however only the programmer can say if the code is missing an *end* or has an extra *do*.

```
  Line nesting
Number  B   M  Original Code
     1  0   0  data _null_;
     2  0   0    do a = 1 to 10;
     3  1   0      do b = 1 to a;
     4  2   0        do c = 1 to b;
     5  3   0          output;
     6  2   0          end;
     7  2   0
     8  2   0        if b = 5 then do;
     9  3   0           d = b;
    10  2   0           end;
    11  2   0        else do;
    12  3   0           d = a;
    13  3   0  ***      end;
    14  3   0        select (a);
    15  3   0          when (1) do;
    16  4   0            d = a;
    17  3   0            end;
    18  3   0          when (5) d = 0;
    19  3   0          otherwise d = .;
    20  3   0          end;  /** select statement **/
    21  2   0        end;
    22  1   0      end;
    23  1   0
```

"NO MATCHING DO/SELECT STATEMENT"

In the example below a *do* statement has been commented at line 3. When this program runs SAS will print the message *"No matching do/select statement"* and abort. This time the base nesting (B) level at line 23 is negative, telling you that there are more *end* statements than there are *do* statements. The nesting level cannot pinpoint the erroneous code because it does not know if you provided an extra *end* or have a missing *do*. The message from SAS suggests you are missing a *do* statement, however only the programmer can say if the code is missing a *do* or has an extra *end*.

```
  Line nesting
Number  B   M  Original Code
     1  0   0  data _null_;
     2  0   0    do a = 1 to 10;
     3  1   0  *** do b = 1 to a;
     4  1   0        do c = 1 to b;
     5  2   0          output;
     6  1   0          end;
     7  1   0
     8  1   0        if b = 5 then do;
     9  2   0           d = b;
    10  1   0           end;
    11  1   0        else do;
    12  2   0           d = a;
    13  1   0           end;
    14  1   0        select (a);
    15  1   0          when (1) do;
    16  2   0            d = a;
    17  1   0            end;
    18  1   0          when (5) d = 0;
    19  1   0          otherwise d = .;
    20  1   0          end;  /** select statement **/
    21  0   0        end;
    22 -1   0      end;
    23 -1   0
```

TESTING

The testing of the Symbol Table Generator has been extensive, however, the testing cannot be considered all encompassing. The testing suite includes programs written by at least half a dozen programmers with varying levels of experience and using styles vastly different from my own, and each other. The test suite demonstrates several of the limitations and restrictions discussed in the next section.

The test suite includes a program over 120 lines long made up almost exclusively of comments to prove that all comment combinations can be identified and ignored. This test of comments not only tests that comments are ignored, but also that active program code leading, trailing, or between comments is properly recognized and represented in the symbol table.

Other tests check the use of the SAS keywords *data, set, do, end,* and *select* used as both keywords and variables alone and in various combinations. Other tests check processes that use *end* as a keyword without a *do* keyword.

KNOWN LIMITATIONS / RESTRICTIONS

There are a few known limitations or restrictions in the Symbol Table Generator. Given the complexity of the code, there are probably more that are not yet known.

MACROS

The Symbol Table Generator does not compile code as the SAS processor does, therefore it does not properly interpret the use of macros. Macros are simply code substitution. That is, macro code is substituted into the job stream where a macro is called. The Symbol Table Generator does not attempt the code substitution.

MACROS THAT CREATE CODE FRAGMENTS
Macros are often used to generate code fragments. Below is an extreme case of a macro being used to generate a code fragment. This macro only generates the keyword *end* when called. The keyword *do* exists in the base code, but each *do* is ended by calling the macro. The program below will function perfectly, but the Symbol Table Generator will not report the *do / end* nesting accurately.

```
  Line nesting
Number  B   M  Original Code
-------------------------------------------------------------------
     1  0   0  %macro e;
     2  0   0  end;
     3 -1   0  %mend;
     4 -1   0
     5 -1   0  data one;
     6 -1   0    do a = 1 to 10;
     7  0   0      do b = 1 to 10;
     8  1   0        do c = 1 to 10;
     9  2   0          output;
    10  2   0          %e;
    11  2   0        %e;
    12  2   0      %e;
    13  2   0  run;
```

MACROS THAT BYPASS PROGRAM CODE
A common method of bypassing a large section of code in a program is to enclose it in a macro and never call the macro. The SAS system will compile the macro as it would any other macro, but SAS does not know or care if the macro is ever actually called. This is an effective way of bypassing blocks of code that contain slash-star star-slash comments.

Using this technique of bypassing code can deactivate *do*s or *end*s in your code; however the Symbol Table Generator does not interpret the macro and may get lost. Below is an example that uses this technique. Though this program will execute and probably run correctly, the Symbol Table Generator will parse the code that is intended to be skipped, counting *do*s and *end*s, and will detect unbalanced *do*s / *end*s.

```
  Line nesting
Number  B   M  Original Code
-------------------------------------------------------------------
  1051  0   0      else do;
  1052  1   0        if undersc = ' '
  1053  1   0          then do;
  1054  2   0            dd = ' ';
  1055  2   0          end;
  1056  1   0          else do;
  1057  2   0
  1058  2   0  %macro skip1;
  1059  2   0            if indexc(undersc,'/-') ^= 0
  1060  2   0              then do;
  1061  3   0                dd = substr(undersc,1,2);
  1062  3   0                if rank(dd) < 48 | rank(dd) > 57
  1063  3   0                  then dd = '  ';
  1064  3   0              end;
  1065  2   0              else do;
  1066  3   0                dd = substr(undersc,7,2);
  1067  3   0              end;
  1068  2   0            end;
  1069  1   0  %mend skip1;
  1070  1   0
  1071  1   0            if indexc(undersc,'/-') ^= 0
  1072  1   0              then do;
  1073  2   0                dd = scan(undersc,1,'/-');
  1074  2   0                if dd in ('UNK','DD')
  1075  2   0                  then dd = '  ';
  1076  2   0              end;
  1077  1   0              else do;
  1078  2   0                dd = substr(undersc,7,2);
  1079  2   0              end;
  1080  1   0            end;
  1081  0   0
  1082  0   0      end;
  1083 -1   0
```

“UNREASONABLE” COMBINATION OF *DO*S AND *END*S IN TEMPLATE PROCEDURE

Using *do* or *end* in “unreasonable” ways will cause the base *do / end* nesting to produce unexpected results. In the example below, though *do* and *end* are being used in a perfectly legal fashion, its acceptability may be in question. It seems reasonable to believe that more meaningful names for the styles could be used.

```
  Line nesting
Number  B   M  Original Code
     1  0   0      proc template;
     2  0   0         define style end;
     3  0   0            replace fonts /
     4  0   0               'FixedFont' = ("Courier",2)
     5  0   0               'BatchFixedFont' = ("SAS Monospace, Courier",2)
     6  0   0               'FixedHeadingFont' = ("Courier",2)
     7  0   0               'FixedStrongFont' = ("Courier",2,Bold)
     8  0   0               'FixedEmphasisFont' = ("Courier",2,Italic);
     9  0   0         end;
    10 -1   0      run;
    11 -1   0
    12 -1   0      proc template;
    13 -1   0         define style do;
    14  0   0
    15  0   0            parent = end;
    16  0   0            replace fonts /
    17  0   0               'FixedFont' = ("Courier",2)
    18  0   0               'BatchFixedFont' = ("SAS Monospace, Courier",2)
    19  0   0               'FixedHeadingFont' = ("Courier",2)
    20  0   0               'FixedStrongFont' = ("Courier",2,Bold)
    21  0   0               'FixedEmphasisFont' = ("Courier",2,Italic);
    22  0   0         end;
    23 -1   0      run;
    24 -1   0
```

SPECIFICATIONS

The parameters of the Symbol Table Generator are

```
%macro symbol(
    dsn           /* REQUIRED, fully qualified program name to process  */
   ,suppress = n  /* Defaulted, suppress common keywords?  Y|N          */
   ,nesting  = 2  /* Defaulted, nesting level desired.  0|1|2|3         */
   ,symbol   = y  /* Defaulted, provide symbol table?  Y|N              */
);
```

- **DSN**, required positional parameter, the fully qualified file name for the Symbol Table Generator to process.
- **SUPPRESS=,** optional keyword parameter, valid values Y | N, default = N, indicates whether or not to suppress the common keywords from the symbol table.
- **NESTING=,** optional keyword parameter, valid values 0 | 1 | 2 | 3, default = 2, indicates the level of nesting to provide in the line numbered listing.
  - 0 = no nesting
  - 1 = base DO/END nesting
  - 2 = base and macro DO/END nesting
  - 3 = base and macro DO/END and macro nesting
- **SYMBOL=,** optional keyword parameter, valid values Y | N, default = Y, indicates whether or not to produce the symbol table.

SO, WHAT DO YOU GET?

Appendix 1 shows a page from the line numbered listing with the code exactly as it appears in the provided program, including comments.

Appendix 2 shows a sample of the Symbol Table. The tokens, or symbols, are in the first column, and the second column contains all the line numbers on which the symbol is referenced.

DISCLAIMER

The New and Improved Symbol Table Generator has been tested against many programs, large and small, efficient or not, conveniently human readable or not, written by many different programmers, experienced or not, technical or not. Many different programming styles have been encountered. However, this in no way means that all scenarios have been tested or accounted for.

RECOMMENDED READING

Johnson, Jim (May 2000), **"Symbol Table Generator"**, *Proceedings of the 2000 Conference of the Pharmaceutical Industry SAS Users Group,* Cary, NC: SAS Institute, Inc., 415-420. This paper is available on the Lex Jansen website at http://www.lexjansen.com/pharmasug/2000/techtech/tt07.pdf.

TRADEMARKS

ABOUT THE AUTHOR

Jim Johnson has been programming with SAS in the Pharmaceutical Industry since 1986. He has presented at many local, regional, and national conferences and has been teaching in the SAS Certificate Program at Philadelphia University since its inception in 1997. Jim has a reputation as a "problem solver" and efficiency enthusiast. His recent work includes large SAS systems, writing programs that write programs, standard macro programming, advanced validation and documentation skills, and an SDTM compliance verification system.

Jim Johnson
jimmy2960@verizon.net

Appendix 1

```
                                          LINE LISTING for c:\symbol.sas                                                1
                                                                                    08:39 Thursday, February 10, 2011

  Line nesting
Number  B   M  Original Code
----------------------------------------------------------------------------------------------------------------------
     1  0   0  *****************************************************************************************************;
     2  0   0  *                                                                                                   *;
     3  0   0  *                                                                                                   *;
     4  0   0  *                                                                                                   *;
     5  0   0  *****************************************************************************************************;
     6  0   0                                                    /*                                                */
     7  0   0  %macro symbol(dsn, suppress=n, nesting=2, symbol=y);
     8  0   0                                                    /*                                                */
     9  0   0  %macro count_do;                                  /*                                                */
    10  0   0     if data and leading = ';'                      /*                                                */
    11  0   0       then data = 0;                               /*                                                */
    12  0   0                                                    /*                                                */
    13  0   0     if set and leading = ';'                       /*                                                */
    14  0   0       then set = 0;                                /*                                                */
    15  0   0                                                    /*                                                */
    16  0   0     if template = 1 and                            /* If PROC TEMPLATE is active ...                 */
    17  0   0        define   = 1 and                            /*  ... and it has an open DEFINE statement ...   */
    18  0   0        symbol2  = 'END'                            /*  ... then do not count the END.                */
    19  0   0       then do;                                     /*                                                */
    20  1   0          define   = 0;                             /*                                                */
    21  0   0          end;                                      /*                                                */
    22  0   0                                                    /*                                                */
    23  0   0       else do; /** not part of proc template **/   /*                                                */
    24  1   0          if symbol2 = '%DO' then do;               /*                                                */
    25  2   0             if indexc(trailing,'''"') = 0 and      /*                                                */
    26  2   0                indexc(leading, '''"') = 0          /*                                                */
    27  2   0               then m1 = 1;                         /*                                                */
    28  1   0             end;                                   /*                                                */
    29  1   0                                                    /*                                                */
    30  1   0          if symbol2 = '%END'  then do;             /*                                                */
    31  2   0             if indexc(trailing,'''"') = 0 or       /*                                                */
    32  2   0                indexc(leading, '''"') = 0          /*                                                */
    33  2   0               then m0 = m0 - 1;                    /*                                                */
    34  1   0             end;                                   /*                                                */
    35  1   0                                                    /*                                                */
    36  1   0          if symbol2 = 'DO' then do;                /*                                                */
    37  2   0             if not data and                        /*                                                */
    38  2   0                not set                             /*                                                */
    39  2   0               then do;                             /*                                                */
    40  3   0                 if when                            /*                                                */
    41  3   0                   then delims = '*+-/:,=("''';     /*                                                */
    42  3   0                   else delims = '*+-/:,=()"''';    /*                                                */
    43  3   0               * ?? why this limited number of delimiters ?? ;
    44  3   0               * close paren removed from leading for data step select like ... when (0) do ...  ;
    45  3   0               * colon added to leading 20080331 for something like ... err: do not match ...  ;
    46  3   0                  if (indexc(trailing,'*+-/,=$()''"') = 0 and
    47  3   0                      indexc(leading, delims        ) = 0   and
    48  3   0                      (trailing ^= ' ' or leading  ^= ' '))
    49  3   0                  OR (trailing = ' ' and leading = ' ' and last_sym in ('THEN' 'ELSE'))
    50  3   0                    then d1 = 1;                    /*                                                */
    51  3   0                  if select then dos + 1;           /*                                                */
    52  2   0                  end;                              /*                                                */
    53  1   0             end; /** symbol2 = DO **/              /*                                                */
    54  1   0                                                    /*                                                */
    55  1   0          if symbol2 = 'END'  then do;              /*                                                */
    56  2   0             if (select = 0 OR (select = 1 and dos > 0)) and
    57  2   0                (not data)                                and
    58  2   0                (not set)                           /*                                                */
    59  2   0               then do;                             /*                                                */
    60  3   0                  if indexc(trailing,'*+-/,=()''"') = 0 and
    61  3   0                     indexc(leading, '*+-/,=()''"') = 0 and
    62  3   0                      (trailing ^= ' ' or leading  ^= ' ')
    63  3   0                    then do;                        /*                                                */
    64  4   0                      d0 = d0 - 1;                  /*                                                */
    65  3   0                      end;                          /*                                                */
    66  2   0                  end;                              /*                                                */
    67  1   0             end; /** symbol2 = 'END' **/           /*                                                */
    68  1   0                                                    /*                                                */
    69  1   0          if symbol in ('&' '%')                    /* if word is & or %                              */
    70  1   0            then;                                   /*   then do NOTHING! DO NOT DELETE!              */
    71  1   0            else do;                                /*   else output record.                          */
    72  2   0               count + 1;                           /*                                                */
    73  2   0               output __symb__;                     /*                                                */
    74  2   0               last_sym = symbol2;                  /*                                                */
```

Appendix 2

```
                                           SYMBOL TABLE for c:\symbol.sas                                       3
                                                                              08:39 Thursday, February 10, 2011

Symbol                Line Numbers
-----------------------------------------------------------------------------------------------------------------
c_end                 293,296,297,300,305,319,322,329

d                     175,184,421,424,426,454,456,463,465,473,475

d0                    64,64,184,421,433,433

d1                    50,184,433,434

DATA                  112,339,488

data                  10,11,37,57,116,118,119,146,174,199,445,453,462,472,484,495,499,520,522

datasets              529

date                  517

datetime              152,206

DEFINE                348

define                17,20,197,344,350,447,448,455,456,457,464,465,466,467,474,475,476,477,478,522,523

delete                262,281,311,333,490,504,532

delimiter             182,190,191,365,381,384,390,394,402,406,411,417

delims                41,42,47

display               447,448,455,456,457,464,465,466,467,474,475,476,477,478,523

DO                    36,488

do                    19,23,24,30,36,39,55,59,63,71,91,97,151,155,161,205,212,220,241,245,258,269,278,296,306,309,
                      320,330,342,357,366,372,378,380,382,387,399,401,413,425,511

does                  139

dos                   51,56,85,132,132

edge                  170

EDIT                  349

ELSE                  49,488

else                  23,42,71,215,245,249,265,274,313,316,325,360,399

END                   18,55,86,131,488

end                   21,28,34,52,53,65,66,67,75,77,94,100,153,164,165,171,176,207,217,223,250,251,263,272,282,301,
                      312,314,323,334,345,368,374,389,395,397,405,407,408,409,419,429,500,514

file                  177

fileexist             137

filename              139

first                 505

flow                  448,457,467,478,522,523

for                   442,518

format                447,455,456,464,465,466,474,475,476,477

headline              445,453,462,472,520

IF                    488

if                    10,13,16,24,25,30,31,36,37,40,46,51,55,56,60,69,79,84,89,96,102,107,112,118,121,126,130,150,
                      154,205,213,238,243,246,254,267,276,304,308,318,327,338,346,348,358,362,370,377,379,381,383,
                      386,400,403,411,424,486,504,505,508,510

in                    49,69,81,103,108,109,148,486
```