Take an In-Depth Look at the %EVAL Function

Beilei Xu, Merck &Co., Inc., Rahway, NJ
Lei Zhang, Merck &Co., Inc., Rahway, NJ

ABSTRACT

%EVAL is a widely used yet frequently misunderstood SAS® macro language function due to its seemingly simple form. However, when its actual argument is a complex macro expression interlaced with special characters, mixed arithmetic and logical operators, or macro quotation functions, its usage and result become elusive and problematic. In this paper, we aim to explore and demystify the intricate part of %EVAL. First, we explain the mechanism that the SAS macro processor uses to invoke %EVAL, and point out the subtle difference between the implicit and explicit uses of %EVAL calls. Then, we reveal a series of hidden rules that %EVAL uses to manipulate its actual macro argument embedded with different types of special characters. Examples are provided to illustrate those rules and exceptions on a case-by-case basis.

INTRODUCTION

The %EVAL function evaluates integer arithmetic or logical expressions. %EVAL operates by converting its argument from a character value to a numeric or logical expression. Then it performs the evaluation. Finally, %EVAL converts the result back to a character value and returns that value.

All parts of the macro language that evaluate expressions (for example, %IF and %DO statements) call %EVAL to evaluate the expressions. Unlike other SAS macro functions, the %EVAL function can be called explicitly and implicitly. It is often overlooked due to its seemingly simple implicit form. For most serious users, the %EVAL function frequently poses challenges. It appears elusive and problematic when its argument is a macro expression laced with special characters, mixed arithmetic and logical operators, or macro quoting functions. Consider the following simple example. The code produces the LOG shown in the box below.

```
* Example 1;
%macro notruth(x);
      %if &x=TRUE %then %put TRUE;
      %else %put FALSE;
%mend;

%notruth(TRUE)          TRUE
```

In this example, the %IF statement calls the %EVAL function implicitly to evaluate the logical expression &X=TRUE, where the macro parameter "X" is resolved as TRUE by the macro call. Since TRUE=TRUE, the condition is true, it writes TRUE in the LOG. However, you wouldn't expect that each of the following macro calls also produces TRUE in the LOG.

```
%notruth(FALSE <);                  TRUE

%notruth(1 OR FALSE);               TRUE

%notruth(FALSE NE TRUE | TRUE);     TRUE

%notruth(LE OR );                   TRUE

%notruth(%str(     TRUE));          TRUE
```

Vice versa, we don't expect the following call would produce FALSE in the LOG.

```
%notruth(%quote(    TRUE))          FALSE
```

The complexity or the surprise in this example is mainly because what the actual argument resolves into changed the evaluation operation. Also, macro quotation functions have different effects on the expression. This paper aims to explore and demystify the intricate part of the %EVAL function. It discusses how macro processor handles a %EVAL function call, and how differently it treats explicit and implicit calls. We also summarize a series of rules which the %EVAL function uses to manipulate special characters and mixed arithmetic and logical expressions. Plenty of examples are presented to illustrate those hidden rules and exceptions.

INVOCATION OF %EVAL FUNCTION

A macro expression is any combination of text, macro variables, macro functions, or macro calls. When %EVAL is explicitly invoked to evaluate its argument: a macro expression, it internally performs the following two steps during execution time.

1. Resolution phase: Rescan the value of the actual argument to resolve the special macro symbols % and &. Normally, these two symbols indicate references to macro variables and macro calls that must be resolved prior to the evaluation;
2. Evaluation phase: Parse the resolved text from step 1 into arithmetic or logical operators and operands, and perform evaluation on this parsed expression.

Example 2 shows the common warning and error messages associated with %EVAL function during these two different phases.

```
* Example 2;
%macro example2;
      %if %eval(&ABC + %XYZ) %then %put TRUE;
%mend example2;

%example2
```

```
WARNING: Apparent symbolic reference ABC not resolved.
WARNING: Apparent invocation of macro XYZ not resolved.
WARNING: Apparent symbolic reference ABC not resolved.
WARNING: Apparent invocation of macro XYZ not resolved.
ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is
required. The condition was: &ABC + %XYZ
ERROR: %EVAL function has no expression to evaluate, or %IF statement has no condition.
ERROR: The macro EXAMPLE2 will stop executing.
```

The above four warnings come from the resolution phase. Notice the warnings are repeated. The first set is generated when the macro processor tried to resolve &ABC and invoke %XYZ, but it failed. So it gave warning messages, and passed &ABC+%XYZ to %EVAL as its actual argument. Then %EVAL rescanned this expression and tried to resolve the symbols again for evaluation. Unfortunately, %EVAL was not able to resolve the symbols either. It generates the second set of the warning messages.

The error messages are from the evaluation phase. %EVAL was left with the unresolved expression: (&ABC+%XYZ) and tried to evaluate it. The operands are character texts, but the operator "+" expects numeric operands. It causes %EVAL to stop evaluating this arithmetic expression.

However, the macro processor treats implicit %EVAL calls differently. Example 3 shows a %IF statement implicitly calling %EVAL to evaluate the same expression.

```
* Example 3;
%macro example3;
      %if (&ABC + %XYZ) %then %put TRUE;
%mend example3;

%example3
```

```
WARNING: Apparent symbolic reference ABC not resolved.
WARNING: Apparent invocation of macro XYZ not resolved.
ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is
required. The condition was: (&ABC + %XYZ)
ERROR: The macro EXAMPLE3 will stop executing.
```

Note it only generates one set of warning messages, as compared to two sets in the explicit %EVAL call in Example 2. In this implicit %EVAL call, the macro expression was not identified as the argument of an implicit %EVAL call until the statement was executed. The normal argument passing mechanism was not triggered. The macro expression was directly passed to the %EVAL function as an actual argument. And it caused the warning and error messages when %EVAL tried to resolve and evaluate the expression.

Therefore, a macro expression, as an actual argument of an explicit %EVAL call, will be scanned and resolved twice; while as an actual argument of an implicit %EVAL call, it will only be scanned and resolved once.

INTERACTION WITH SPECIAL CHARACTERS

Once %EVAL is invoked, whether implicitly or explicitly, its argument is scanned, resolved, and passed to %EVAL to evaluate. Then %EVAL determines whether it performs an integer arithmetic evaluation or a logical comparison based on the actual resolved argument.

When all operands in the argument can be interpreted as integers, the expression is treated as arithmetic. If at least one operand cannot be interpreted as numeric, the expression is treated as logical. When %EVAL encounters an arithmetic operation containing non-integer values or characters, it displays an error message about finding a character operand where a numeric operand is required. If a division operation results in a fraction, the fraction is truncated to an integer. %SYSEVALF is the function to deal with non-integer floating point arithmetic and comparisons.

%EVAL can have multiple arithmetic or logical expressions or mixed expressions as an argument. The order in which operations are performed in the macro language is the same as in the DATA step. As in the DATA step, operations within parentheses are performed first. Confusions and exceptions often rise from the interaction among macro expression resolutions, text quotations and %EVAL's special requirements for integer arithmetic or logical evaluation. In this section, we will explore some rules that %EVAL function uses to deal with special characters, giving many examples.

1. Blank(s)

Blanks are normally separators between operators and operands in an arithmetic or logical expression. The following rules apply when blanks are not used as separators.

- Leading and trailing blanks in %STR and %NRSTR functions (compilation time quoting functions) will be stripped off during the evaluation;
- Leading and trailing blanks in %NRBQUOTE and %BQUOTE functions (execution time quoting functions) will not be stripped off, and they can not be trimmed.

Example 4 examines these two rules explicitly. %EVAL performs logical comparisons between two strings. It returns a value that is evaluated as true or false. In the macro language, any numeric value other than 0 is true and a value of 0 is false.

```
* Example 4;
%put %eval(%str( A   ) EQ A);                          1

%put %eval(%nrstr( A   ) EQ A);                        1

%put %eval(%nrbquote( A   ) EQ A);                     0

%put %eval(%nrbquote( A   ) EQ %nrbquote(A));                  0

%put %eval(%qtrim(%nrbquote( A   )) EQ %nrbquote(A));          0

%put %eval(%nrbquote(%trim( A   )) EQ %nrbquote(A));           1

%put %eval(%nrbquote(%trim(%str( A   ))) EQ %nrbquote(A));     0
```

Just like blanks in the DATA step, all blanks are equal no matter how many blanks there are. The following three statements all write 1 in the log.

```
%put %eval(%nrbquote(    ) EQ %nrstr());          1

%put %eval(%nrbquote(    ) EQ %nrstr( ));         1

%put %eval(%nrbquote(    ) EQ %nrstr(  ));        1
```

2. Parentheses: ( )

Matched parentheses act as grouping characters as well as separators. They will change the evaluation order. The expression within parentheses will be evaluated first. Compare the different results from expressions with or without parentheses in the following example.

```
* Example 5;
%put %eval((A>B)+(C>D));
```

```
0
```

```
%put %eval(A>B+C>D);
```

```
ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is
required. The condition was: A>B+C>D
```

3. Period: .

In macro expressions, a period (.) can be used as a separator for macro variables. For example, in &x.y, the period separates macro variable &X from the regular text Y. %EVAL only accepts operands in arithmetic expressions that represent integers (in standard or hexadecimal form), operands that contain a period character cause an error when they are part of an integer arithmetic expression. However, if a period appears in a resolved logical expression, it will be treated as a regular character. For example if the SAS version is 8.2, the expression %eval(&sysver > 10) is evaluated as true.

The following examples show correct and incorrect usage of period, respectively:

```
* Example 6;
%put %eval(10.0+20.0);
```

```
ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is
required. The condition was: 10.0+20.0
```

```
%put %eval( 10 > .);
```

```
1
```

```
%put %eval( A > .);
```

```
1
```

Now consider the following examples for different results of using %EVAL and %SYSEVALF.

```
%put %eval(10+.);
```

```
ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is
required. The condition was: 10+.
```

```
%put %sysevalf(10+.);
```

```
NOTE: Missing values were generated as a result of performing an operation on missing values during
%SYSEVALF expression evaluation.
.
```

%EVAL treats the period as a character value, which causes an error in an arithmetic operation. However, %SYSEVAL treats the period as a missing value and returns a missing value without an error.

4. Percentage sign: %

As a single character, the percent sign (%) will be treated as an ordinary character. Here is an example:

```
* Example 7;
%macro abc;
    %if % = %str(%) %then %put TRUE;
    %else %put FALSE;
%mend abc;

%abc          TRUE
```

If a percentage sign (%) follows non-blank characters, it triggers a macro call during the resolution of the macro argument. If the macro call can not be executed, SAS will issue warning messages. For example,

```
* Example 8;
%macro abc;
   %if %macrocall = %str(%%macrocall) %then %put TRUE;
   %else %put FALSE;
%mend abc;

%abc
```

```
WARNING: Apparent invocation of macro MACROCALL not resolved.
WARNING: Apparent invocation of macro MACROCALL not resolved.
TRUE
```

Notice that %EVAL still performs the evaluation after the resolution warning.

5. Single or Double Quotes: ‘, “

It is always required to mask unbalanced quotes (‘, “) with macro quoting functions such as %NRSTR, or %NRBQUOTE, but you don’t have to mask balanced quotes (‘’, “”). The following is an example of balanced double quotes.

```
* Example 9;
%macro abc;
   %if "A" < "B" %then %put TRUE;
   %else %put FALSE;
%mend abc;

%abc          TRUE
```

Unlike quotes in the DATA step where encompassing quotes are not part of the value, quotes in macro expressions are part of the value and are treated just like any other regular characters. For example:

```
%put %eval("0" = '0');   0
```

Consider comparing a string with or without quotes: as the following two examples show, a string with quotes and without quotes returns different values in a macro expression.

```
%put %eval("0" = 0);     0

%put %eval(" " =  );     0
```

6. Operators for minimum, maximum, and concatenation: ><, <>, and ||

In the DATA step, >< and <> can be used to obtain minimum and maximum values of two operands. However, the %EVAL function does not recognize these two operators. Neither does %EVAL recognize the concatenation operator: ||. Example 10 shows the error message in LOG by using <> in the evaluation.

```
* Example 10;
%macro abc;
   %if  A <> B %then %put TRUE;
   %else %put FALSE;
%mend abc;

%abc
```

```
ERROR: A character operand was found in the %EVAL function or %IF condition where a    numeric operand
is required. The condition was: A <> B
ERROR: The macro ABC will stop executing.
```

```
* Example 11;
%macro abc;
   %if  %eval(C le (A <> B)) %then %put TRUE;
   %else %put FALSE;
%mend abc;

%abc                          FALSE
```

In Example 11, <> are treated as regular characters. So string C is compared to String A <> B. Since "C" is greater than "A" according to the sorting order, so the macro returns "FALSE".

The %EVAL function treats <>, >< and || as regular characters. If you accidentally insert a space between the symbols, they will become logical comparison symbols.

For example, we insert a space between <> in example 10.

```
%macro abc;
   %if  A< >B %then %put TRUE;
   %else %put FALSE;
%mend abc;

%abc

FALSE
```

We don't get any error messages; instead we get an evaluation result, "FALSE". With the space between "<" and ">", the expression became two logical expressions: A< (blank) >B. Operators < and > have the same evaluation precedence, so the evaluation is from left to right in this expression. First, A< (blank) is evaluated and returns 0. And then 0 is passed to the next evaluation, 0>B, which also returns 0, for a false condition.

7. Unary arithmetic operators: + and –

When + or - is used as positive and negative prefix, the right operand must be numeric, otherwise, an error message will be issued.

```
* Example 12;
%put %eval(-A);
```

```
ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is
required. The condition was: -A
```

```
%put %eval(-9);                -9
```

Here is another example to show the interaction with compilation and execution quoting functions. The %NRSTR function is a compilation time macro function, so it trims the trailing blank after 9 and returns -9. With the %NRBQUOTE function, the trailing blank is masked and is not trimmed, so "9 " becomes a character operand and causes the error.

```
* Example 13;

%put %eval(%nrstr(-9 ));          -9

%put %eval(%nrbquote(-9  ));
```

```
ERROR: A character operand was found in the %Eval function or %IF
       condition where a numeric operand is required. The condition was: -9
```

8. Binary arithmetic operators: + , –, /, *, **

When +, -, /, * and ** are used as arithmetic operators, %EVAL expects numeric operands; otherwise it will produce error messages.

9. Binary logical operators: AND (&), OR (|)

When & or | is used as the logical AND or OR operator, blanks are not required between their operands; while blanks are required for AND and OR. These two logical operators also expect numeric operands. When character operands are detected, an error message is issued.

```
* Example 14;
```

```
%put %eval(1|A);
```

```
ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is
required. The condition was: 1|A
```

When character operands are used with &, and there is no space between them, it will trigger the macro variable resolution. If the macro variable is not resolved, SAS displays warning messages and continues to evaluate the expression as a logical expression. For example,

```
* Example 15;
%put %eval(1&A=B);
```

```
WARNING: Apparent symbolic reference A not resolved.
WARNING: Apparent symbolic reference A not resolved.
0
```

In this expression, &A triggers the macro variable resolution. After producing the warning messages of unresolved macro variable A, it tries to evaluate the expression anyway. Since the evaluation for operator = precedes the logical operator &, A=B is evaluated first and it returns the value of 0. Then it evaluates the expression: 1&0, which is the intersection of 1 and 0. Apparently, the result is 0.

10. Binary logical operators: LT, LE, EQ, NE, GT, GE, <, <=, =, ^=, >, >=

Blanks are not required for the operands <, <=, =, ^=, >, >=, but are required for their mnemonics. For these logical comparisons, operands can be numeric and/or character. If both operands are numeric, numeric comparison will be performed. If one of the operands is character, character comparison is performed. Example 16 shows the different comparisons %EVAL performs just by changing one operand.

```
* Example 16;
%put %eval( (500 + 20) LE %str(1000));     1

%put %eval( (500 + 20) LE %str(1000.));     0
```

A compound logical expression as in 10<&X<20, may or may not be the equivalent to a DATA step compound expression (depending on what the macro expression resolves to). To be safe, write the connecting operator explicitly, as in the expression 10<&X AND &X<20.

In summary, when dealing with a complex argument of %EVAL, understanding how the macro processor works with %EVAL, what roles each operator or special character plays, and the order in which operations are performed helps to better interpret the LOG messages and the evaluation results.

CONCLUSION

We have highlighted some rules governing %EVAL function. Understanding them helps us have confidence in dealing with the %EVAL function. The subtle difference between explicit and implicit %EVAL calls, the difference between masked text at compilation (by %STR or %NRSTR) and at execution (by %QUOTE, %NRQUOTE, %BQUOTE, or %NRBQUOTE), and some tricky situations with special characters that we have discussed should be kept in mind when we write our macros.

ACKNOWLEDGMENTS

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Beilei Xu
Merck & Co., Inc.
P.O. Box 2000
Rahway, NJ 07065
Work Phone: (732) 594-9980
Fax: (732) 594-6075
Email: beilei_xu@merck.com

Lei Zhang
Merck & Co., Inc.
P.O. Box 2000
Rahway, NJ 07065
Work Phone: (732) 594-9865
Fax: (732) 594-6075
Email: lei_zhang4@merck.com