Paper CC-09

# If and %IF You Don't Understand

Ian Whitlock, Kennett Square, PA

## Abstract

Many beginning SAS® Software macro programmers are confused about the difference between the DATA step IF and a %IF in a DATA step generated by a macro, and when to use which. Simple examples are used and discussed so that you may never be confused about this issue again.

This presentation is appropriate to all operating systems and SAS products where code is written.

## Introduction

The SAS DATA step language provides an IF statement and the SAS MACRO language provides a %IF statement. You might say, "Two different languages two different statements." Although this is an answer and a correct one, it doesn't really answer the confusion between the two statements or when you should use one as opposed to the other.

On the other hand, it does focus on the problem that the two languages are designed to do different things and that the source of confusion may be to recognize the different objectives of the two languages. Many beginning macro programmers make the mistake of thinking that macro is a super SAS and hence fail to appreciate the difference and consequently make many mistakes.

There are four times that are important to distinguish when a SAS program with macros is executed.

1) Macro compile time when the code between the %MACRO and %MEND statements is read and stored.[1]
2) Macro execution time when the macro instructions are executed to produce SAS code.
3) SAS compile time when the DATA step code is compiled and procedure calls parsed.
4) SAS execution time when a step boundary is encountered the prepared step executes.

With the study of times we see that the %IF statement makes a decision when the SAS code is being generated and compiled. It is essentially a decision about how to generate code or which code to generate. The IF statement actually executes when the DATA step is executing.

The problems below assume a macro environment in which a DATA step is generated.

## Example 1

Suppose we are generating a DATA step to read one of two SAS data sets. On Monday through Thursday mornings (at 1:00 am) a set LIB.DETAILS is prepared for our program to process during the day. On Friday morning a set LIB.SUMMARY is to be processed. First we try a direct IF statement so the code might be:

```
data analysis_prep ;
  if 2 <= weekday(today()) < 6 then
```

[1] There is no macro compile time for open code macro instructions, e.g. %LET or %PUT, outside of macros.

```
    set lib.details ;
  else
    set lib.summary ;
  ....
run ;
```

This step doesn't work. Why? The compiler sees two SET statements. When the compiler sees a SET statement it reads the header to find out what the variables are. On Monday through Thursday the second SET statement cannot be compiled because LIB.SUMMARY doesn't exist. On Fridays the first SET statement fails because LIB.DETAILS doesn't exist.

The IF statement guarantees that only the correct set would be read if the DATA step could be compiled, but it cannot be compiled because both sets must be available at compile time. This is a rather curious situation that is worth pondering whether you ever write macro code or not.

So now we try a %IF statement.

```
data analysis_prep ;
  %if 2 <= %sysfunc(weekday(&sysdate)) and %sysfunc(weekday(&sysdate)) < 6
  %then %do ;
    set lib.details ;
  %end ;
  %else %do ;
    set lib.summary ;
  %end ;
  ....
run ;
```

First note that we had to change the %IF condition quite a bit. Why? One, we needed to use the function WEEKDAY during the compilation of the DATA step and that means we have to tell the macro facility to use the function rather than just generate the code for SAS to use it. Second the form, A <= B < C, is a SAS idiom for, A <= B and B < C, which is not available in %IF[2].

The code works because only one SET statement is generated. Consequently the SAS compiler sees only the correct data set name. Hence the DATA step compiles and executes when there are no other problems.

The style indicated, is harder to read than either plain SAS code or plain macro instructions because the two language are intertwined. It is possible to develop a better style. Consider the following code.

```
%local data ;
%if 2 <= %sysfunc(weekday(&sysdate)) and %sysfunc(weekday(&sysdate)) < 6
%then %let data = lib.details ;
%else %let data = lib.summary ;

data analysis_prep ;
  set &data ;
  ....
run ;
```

---

[2] %IF calls %EVAL to do the evaluation of the condition, so it is really %EVAL that does not understand the idiom. The SAS idiom, a < b < c, means the combined condition, a < b and b < c. (Any set of consistent comparison operators may be used in place of the < symbol.)

Here the two languages are separated. Now the DATA step is as easy to read as the remaining code allows and the macro instructions with the %IF are also relatively clear.

So should we conclude that two languages are necessary and this is the essential difference between IF and %IF? No, consider the following solution using IF and no macro for this part of the problem.

```
filename set_stmt temp ;     /* uses WORK to store file during execution of the program */
data _null_ ;                /* DATA step to write the SET statement */
  file set_stmt ;
  if 2 <= weekday(today()) < 6 then
    put "set lib.details ;" ;
  else
    put "set lib.summary ;" ;
run ;

data analysis_prep ;
  %inc set_stmt / source2 ;
  ....
run ;
```

This brings us back to the fact that the difference is not so much one of language as one of timing. The DATA step to write the SET statement must precede the compilation of the step that uses it. (Do you understand why?[3]) Since SAS steps are compiled and executed before the next step is considered for compilation, the objective can be achieved as a two-step process without any macro[4].

However, the introduction of a macro variable might improve the readability os the second step.

```
%local data ;
data _null_ ;    /* Store data set name in a macro variable */
  if 2 <= weekday(today()) < 6 then
    call symput ("data", "lib.details") ;
  else
    call symput ("data", "lib.summary") ;
run ;

/* value of macro variable is available in next step and after */
data analysis_prep ;
  set &data ;
  ....
run ;
```

The big advantage of macro instructions over something like writing an include file is that the instructions can appear in the same step because macro instructions do macro execute during SAS compilation. If you don't understand, reread the paragraphs about timing in the introduction. The advantage is particularly true when a step is far more complex than simply requiring variability of one data set name.

---

[3] A DATA step cannot begin execution until that step has finished compilation. Consequently any executable line that changes or affects the compilation of code below that line must be in a later step.

[4] The %INCLUDE (or %INC) statement uses a %-sign, but it is a SAS statement that predates the introduction of the new macro language.

### Example 2

Now let's look at the reverse problem. Suppose we have one file of students in either the 4th, 8th, or 12th grade and we have to process information about them for various subjects. The 4th graders in Texas may be studied for reading while 8th graders in Oregon may be studied for science, etc. We have to write a DATA step to process the data and different things must be done for students in different grades and subjects. The question is should we use a %IF statement or an IF statement.

Let's consider %IF first. In outline we have

```
data elementary_results ;
  set students (where = (grade in (4, 8))) ;
  %if grade = 4 %then
  %do ;
    ....
  %end ;
  %else
  %do ;
    ....
  %end ;
run ;
```

What is wrong with this plan? There is a purely logical mistake - how can the code we generate depend on which record we are reading? Remember the code is compiled before any data is considered and before any code is executed. We really need both blocks of code generated.

If the above logical mistake doesn't ring bells for you, then let's consider the condition, grade = 4. Remember that macro instructions process text[5]. Since letters are present, the comparison must be a character one. The expression on the left begins with a "g" and the one on the right with the character 4. These are not the same hence the condition is always false.

We are forced to consider the following code because it is not a question of which code to generate. In this case, we need both blocks of code which is the opposite of Example 1.

```
data elementary_results ;
  set students (where = (grade in (4, 8))) ;
  if grade = 4 then
  do ;
    ....
  end ;
  else
  do ;
    ....
  end ;
run ;
```

[5] Unless there are special circumstances where numeric expressions are expected and the text is a numeric expression. For example, the condition, 2 + 3 = 5, will evaluate to true because, 2 + 3, is a numeric expression and the condition is to compare it with 5. In contrast, the word "GRADE" is not a numeric expression; it is not a macro variable (no &); and it is just text, i.e. five consecutive letters.

Does that mean there is no place for %IF in this DATA step? No, the two blocks of code indicated by a sequence of dots may be generated by macros with many %IF decisions in them. However, that would depend on requirements not specified here and would not depend on the grade level of a student observation read.

### Example 3

In general, if the decision depends on values coming from SAS data (including foreign sources, e.g. Excel) then the decision should probably be based on an IF statement. If the decision depends on values coming from parameters or macro variables then the decision should probably be based on a %IF statement. A more precise way to look at this is: if the conditions of the decision don't change during the step then use a %IF statement. If they do, then use an IF statement.

What if both conditions happen? For example, we get a large data set where sometimes we want only the observations added today and sometimes we want the whole data set. Say we want today's data when the parameter SUBSET is 1 and the whole data set when SUBSET is 0.

Consider the following subsetting IF statement.

```
if &subset = 0 or date_added = today() ;
```

This works, but it means the DATA step always has a subsetting condition. It is just that the condition is sometimes always true. This should be a little disturbing to the purist. A better solution would be to view it as a decision of whether to generate a subsetting IF statement or not. In this view we get

```
%if &subset = 0 %then
%do ;
  if date_added = today() ;
%end ;
```

This is more precise because it breaks the problem into its respective parts, but at a cost of writing more code[6]. I think most experienced macro programmers would choose the second solution, but I can sympathize with the beginner who asks, "What is the harm?".

The same sort problem often arises in generating an IF statement of the form shown below

```
%local i ;
if
  %do i = 1 %to &n ;
    &&cond&i and
  %end ;
  1
then ... ;
```

or with a variable to hold the conditions. For example:

```
%local i big_condition ;
%let big_condition = 1 ;
```

---

[6] The two languages are also intertwined, but they may be separated using the same technique as used in Example 1 - introduce a local macro variable, and use a %IF statement to decide on the value. However, this introduces a quoting problem to hide (or mask) the semicolon of the SAS statement unless a little macro is used to hold the subsetting IF statement.

```
%do i = 1 %to &n ;
  %let big_condition = &big_condition and &&cond&i ;
%end ;

if &big_condition then ... ;
```

It is left as an exercise to find all the variations and then write code without a beginning or ending 1 by the use of a %IF statement in the %DO-loop.

## Decisions

We have considered IF and %IF statements in the context of times during the execution of a SAS program. However, the subject is decisions. The statements are just the programming tools to make the decisions. When should decisions be made? And why? That is the real problem. In addition to the times already mentioned, going backward we might add the time we write the program, the time the specifications are made, and the time the problem is analyzed; and going forward the time reports are read, or the time the next program to be executed is written. So there are many times at which a decision can be made. From the programmers point of view the cheapest decisions are the early ones, those that decide to not handle the problem in this program or to let the user decide (when the programmer doesn't know how to write the condition for the decision) via a user interface.

Choosing the best time is often a compromise. In general the earlier a decision can be made the cheaper it will be in both programming time and execution time. However, early decisions make a program less flexible. You can think of early decisions as building a smaller and smaller box. That is why they are cheaper, but breaking out of the box, when a wrong decision is made, can be quite costly in programming time because you are essentially writing the wrong program.

Much of the flexibility of SAS comes from its step orientation, the compile and execute a step rhythm as shown in the non-macro solution discussed near the end of Example 1. Adding macro makes this flexibility more easily available. Sometimes it means the program can write the program so that the program can make the decision rather than the programmer.

## Conclusion

Much of the confusion about IF versus %IF is due to confusion about the role of the two languages. I hope that the material presented here will help clarify the problem for some people. The last section discussed more generally the role of decisions in the programming process. Although the programmer may not be able to control the decisions outside his/her program s/he should still be aware of how these decisions affect the programming process.

## Contact Information

Ian Whitlock
149 Kendal Drive
Kennett Square, PA, 19348

iw1sas@gmail.com