

MISSING VALUES: Everything You Ever Wanted to Know

Malachy J. Foley, Chapel Hill, NC

ABSTRACT

Many people know about the 28 different missing values for SAS® numerical data. However, few people know about the many different missing values for character data. This paper reviews all the different types of missing values, their sort order, the difficulties they can cause the SAS programmer and how to avoid those difficulties. It also discusses the MISSING system option, and how to use the various missing values in input, output, comparisons, and in PROC FREQ.

INTRODUCTION

Missing values are one of the most basic concepts in SAS. Yet, they are far trickier than one might expect.

For example, aside from the blank and the null character, there are some four other values that SAS recognizes as missing character values. Sometimes these extra values can be useful, as in titles. Other times these values can be devastating.

This paper is for anyone who uses SAS. It looks at both character and numerical missing values, and how both are treated on input, storage, and output. This paper should give the reader a basic understanding of, and an appreciation of the nuances and mysteries of missing values.

NUMERIC MISSING VALUES

The symbol usually used to represent a missing value for a numerical variable is the period or dot. Aside from the dot, there are 27 special missing values SAS can store in numerical variables. They are the dot-underscore (._), and dot-letter (.A thru .Z). Note that these special values are case insensitive. That is, .A=.a .B=.b .C=.c etc.

The special values are available in SAS to distinguish among different types of missings. For example, in a response to a multiple-choice question you might have a missing value because the respondent does not know the answer, or is not sure of the answer, or refuses to answer, or is missing for another reason. In this example, there are 4 different kinds of missings. All four can be distinguished in SAS. For instance, they could be coded as .D .S .R and .M respectively.

SAS can store the 28 missing values in a numerical variable. In addition to these 28 values, SAS sometimes recognizes a blank as a numeric missing on input. Furthermore, SAS can output the dot as almost any character. The following sections detail these and other features.

SORT ORDER OF STORED NUMERICAL MISSING VALUES

The following exhibit shows a SAS program and the corresponding SAS LOG. This program demonstrates the sort order of the 28 missing values SAS stores in numerical variables.

Exhibit 1: Sort Order of Missing Values

```
-----  
DATA _NULL_;  
  PUT "SORT ORDER OF NUM MISSING VALUES";  
  IF .<. THEN PUT "<." "@;  
  IF .>. THEN PUT ">." "@;  
  IF .<.A THEN PUT "<.A" "@;  
  IF .>.A THEN PUT ">.A" "@;
```

```

IF .A<.Z THEN PUT ".A<.Z" "@;
IF .A<.Z THEN PUT ".A<.Z" "@;
IF .Z=.Z THEN PUT ".Z=.Z" "@;

x=-1*1e300;
IF .Z<x THEN PUT ".Z<" x;
RUN;
-----
SAS LOG FILE
-----
SORT ORDER OF NUM MISSING VALUES
._<. .<.A .A<.Z .A<.Z .Z=.Z .Z<-1E300
-----

```

This exhibit illustrates that the different missing values are not equal. What's more, it shows that the dot-underscore is the lowest valued missing value. After the dot-underscore, comes the dot, and then the dot-A. The highest missing value is the dot-Z.

CHECKING FOR MISSING NUMERIC VALUES

The sort order outlined in the previous section has consequences in checking for missing values. Often the SAS programmer uses the following SAS code to check for a missing numeric value:

```
IF VALUE=. THEN PUT "**** Value is missing";
```

While in most instances the above code works as intended, there are occasions where it may not catch some missing values. The above statement assumes that only a dot is present, and none of the other 27 missing numeric values, are present in your data. In exhibit 1, it was shown that the dot-Z is the highest missing value. So, a better, more inclusive way to check for a missing numeric values is:

```
IF VALUE <=.Z THEN PUT "**** Value is missing";
```

The latter IF statement checks for all 28 possible missing values.

It is good programming practice to always use the <=.Z in checking for a missing numeric value. Here are some reasons why. First, it will identify any stray missing values that inadvertently creep into the data. Second, the <=.Z check will work even if the specifications for your data set change. Finally, it allows you to copy your program (code) to another application.

MISSING VALUES IN LOGICAL EXPRESSIONS

The next exhibit investigates how numerical missing values are treated in logical expressions.

```

Exhibit 2: Missing & Logical Expressions
-----
DATA NULL_;
IF NOT (.A) THEN PUT ".A= FALSE";
IF NOT (.) THEN PUT ". = FALSE";
IF NOT 0 THEN PUT "0 = FALSE";
IF . = 0 THEN PUT
    ". = 0 in logical expressions";
ELSE PUT
    ". not = 0 in logical expressions";
IF . = .A THEN PUT
    ". = .A in logical expressions";
ELSE PUT
    ". not = .A in logical expressions";
RUN;
-----
SAS LOG FILE
-----

```

```

-----
.A= FALSE
.= FALSE
0 = FALSE
. not = 0 in logical expressions
. not = .A in logical expressions
-----

```

Notice that dot-A, dot, and zero are all considered false in a logical expression. Yet, these three values are not equal to each other. This is as it should be, but is sometimes counterintuitive.

NUMERICAL MISSING VALUES ON OUTPUT

On output, the usual missing value prints as a dot. The special missing values print as the symbol that follows the dot. So, a dot-underscore outputs as an underscore, and a dot-letter outputs as the corresponding upper-case letter. Consequently, each of the numerical missing values print as one character.

The following code and results demonstrates how the special missing values are printed.

Exhibit 3: Special Numeric Missing Values

CODE	SAS LST FILE
DATA TEST;	MISS
MISS=.e; OUTPUT;	E
MISS=. ; OUTPUT;	.
MISS=.A; OUTPUT;	A
MISS=.B; OUTPUT;	B
MISS=.B; OUTPUT;	B
MISS=.B; OUTPUT;	B
MISS=.C; OUTPUT;	C
MISS=. ; OUTPUT;	-
MISS=. ; OUTPUT;	.
MISS=.A; OUTPUT;	A
MISS=.a; OUTPUT;	A
MISS=.c; OUTPUT;	C
MISS=.d; OUTPUT;	D
MISS= 5; OUTPUT;	5
MISS=61; OUTPUT;	61
RUN;	
PROC PRINT NOOBS; RUN;	

Please note that, except for the dot, SAS requires two characters to denote a missing value in SAS code. Also in code, SAS accepts small letters as well as capital letters to indicate a special missing value. Meanwhile on output, SAS displays only one character and it is always upper-case and right-justified.

Furthermore, if you don't like the dot as the representation of the normal missing value, you can change it. Just specify the character you want in a MISSING system option. In the next exhibit, the dot is change to an asterisk utilizing this option.

Exhibit 4: MISSING System Option

CODE	SAS LST FILE
OPTIONS MISSING='*';	MISS
PROC PRINT	E
DATA=TEST (OBS=4)	*
NOOBS	A
;	B
RUN;	

Here PROC PRINT prints the first few records of the data set (TEST) from exhibit 3. Observe how the MISSING='*' option makes an asterisk, instead of a dot, print on the second record. Rather than an asterisk, any character can be requested, including lower-case letters.

The MISSING system option requires a single character. Multiple characters like OPTIONS MISSING='**' produce an error message.

Exercise care in selecting a character to replace the dot. For example, it is possible to choose a letter as the character to replace the dot. Such a specification could cause confusion with the corresponding dot-letter. For instance, if you make OPTIONS MISSING='A', a dot would print just like a dot-A, producing ambiguous results.

Exhibit 4 shows how the MISSING system option functions with PROC PRINT. This option also works with most outputs, including PROC FREQ (see the next section).

PROC FREQ & SPECIAL NUMERICAL MISSING VALUES

To further understand how the special numerical missing values works, consider the following code and output. This program inputs the TEST data set created in Exhibit 3.

```
Exhibit 5: Missing Values & PROC FREQ
-----
PROC FREQ DATA=TEST;
  TABLES MISS /NOPERCENT NOROW NOCOL;
RUN;
-----
```

```
-----
              SAS LST FILE
-----
MISS      Frequency      Cumulative
-----
      5              1              1
     61              1              2

Frequency Missing = 13
-----
```

Note how in the above example all the different missing values show up as missing with no distinction. On the other hand, look at the following.

```
Exhibit 6: MISSING Option in TABLE State.
-----
PROC FREQ DATA=TEST;
  TABLES MISS
    /NOPERCENT NOROW NOCOL MISSING;
RUN;
-----
```

```
-----
              SAS LST FILE
-----
MISS      Frequency      Cumulative
-----
      .              2              2
      A              3              5
      B              3              8
      C              2              10
      D              1              11
      E              1              12
      5              1              13
      61             1              14
      61             1              15
-----
```

Exhibit 6 is the same as exhibit 5 except the MISSING option in the TABLE statement. With this option, SAS includes the different missing values inside the frequency table.

The next exhibit adds the `OPTIONS MISSING='*'` statement to the previous exhibit. This statement makes an asterisk print instead of a dot.

```
Exhibit 7: System MISSING Option & FREQ
-----
OPTIONS MISSING='*';
PROC FREQ DATA=TEST;
  TABLES MISS
    /NOPERCENT NOROW NOCOL MISSING;
RUN;
-----
```

SAS LST FILE		
MISS	Frequency	Cumulative Frequency
*	2	2
A	3	5
B	3	8
C	2	10
D	1	11
E	1	12
	1	13
5	1	14
61	1	15

INPUT & NUMERIC MISSING VALUES

On input, you can read any of the special numerical missing values provided you specify it in a `MISSING` statement. For example, to input the letters D, S, R, and M as missing values, you can use the program that follows.

```
Exhibit 8: Special Numeric Missing Values
-----
CODE
-----
DATA OUT;
  INPUT ID $ 1-3          SAS LST FILE
        NUM 6-7;
        MISSING D S R M _;
  CARDS;
A01
A02 3
A03 .
A04 D
A05 2
A07 r
A08
A12 S
A14 M
A22 1
A25 4
A26 .S
;
RUN;
PROC PRINT NOOBS;RUN;
-----
```

In this case, the underscore and dot were added to demonstrate their use as well. The dot is always a missing numerical value and requires no specification in the `MISSING` statement.

SAS recognizes both one or two-character special missing values on input. An example of this feature is the S for ID=A12 and the Dot-S for ID=A26 in exhibit 8.

Also in the exhibit SAS accepts a blank as a valid missing value on numeric input. SAS converts the blank to a dot and stores the dot in the corresponding numerical variable (see ID=A08). The blank is automatically considered a missing numeric data if the position of the numeric value is given in the `INPUT` statement. In the previous

example, the location of the NUM value in the input line was coded as columns 6 and 7 in the INPUT statement. However, if instead of INPUT ID \$ 1-3 NUM 6-7, the input statement was INPUT ID \$ NUM, the blank for ID=A08 would have caused reading problems. So, a blank is read as a missing numeric value when you do not use LIST INPUT.

On input, the MISSING systems option does not work. Thus, the only missing values you can input, without trickery, are the 28 normal missings and the blank.

Remember that the special missing values are denoted by two characters in the SAS code, while they are denoted by one or two characters on input and only one character on output.

SPECIAL CHARACTER MISSING VALUES

Until now, this paper explored only numerical missing values. This section will look at missing character values.

SAS officially recognizes only one missing character value. It is the blank. To code the blank missing value use double or single quotes separated by any number of blanks, including zero blanks. Thus, if you want to set the character variable C to blank, all of the following work.

Exhibit 9: Setting a Char Variable to Blank

```
-----
C=" ";
C="  ";
C="   ";
C=' ' ;
C='  ' ;
C='   ' ;
-----
```

Unofficially, SAS may view some non-blank characters as missing. For instance, SAS almost always acknowledges the null character (ASCII code 0) and the special blank character (ASCII code 255) as missing.

Depending on your specific SAS/computer/Operating system setup, you may have still other characters which SAS perceives as missing. The following example illustrates how on one computer configuration there are 6 characters that SAS recognizes as missing.

Exhibit 10: 6 Missing Character Values

```
-----
FILENAME IN 'C:\ascii.inp';
DATA NULL ;
  LENGTH REC $80 DEC $4 CODE $1;
  INFILE IN LENGTH=RECLN MISSEVER;
  INPUT @;
  INPUT @1 REC $VARYING80. RECLN;
  DEC=SUBSTR(REC,1,4);
  CODE=SUBSTR(REC,24,1);
  If CODE = " " THEN
    PUT "*** Missing " code= dec;
RUN;
-----
```

```
-----
                        SAS LOG FILE
-----
*** Missing CODE=  DEC=0
*** Missing CODE=  DEC=9
*** Missing CODE=  DEC=10
*** Missing CODE=  DEC=13
*** Missing CODE=  DEC=32
*** Missing CODE=  DEC=255
-----
```

```
NOTE: The data set WORK.ONE has
      255 observations and 2 variables
-----
```

The input file for this exhibit is a flat ASCII file. It contains all of the ASCII characters except the end-of-file character (^Z). Columns 1-4 of the input file had the ASCII code in the decimal number system written in characters. Column 24 contained the actual ASCII character. However, the input file could have been a SAS data set that was derived from many sources from outside of SAS.

The SAS program in exhibit 10 found the three "normal" missing character values. Namely, it found the blank (Dec=32), the null (Dec=0), and the special blank (Dec=255). In addition to the normal missings, the program found 3 other missing values. They were the Tab (Dec=9), the line feed (Dec=10), and carriage return (Dec=13).

These extra missing values can work for you or against you.

One place where the special missing values can work against you is when they inadvertently and unknowingly get into one of your files. For instance, when you sort on a key field, you expect the records with a missing key field to be at the beginning of your data (because missing has the lowest sort value). But when you have special missing values in your key field, missing values can show up at the beginning, the middle, and the end of your file. Programmers tell horror stories about this situation happening to them. One defense against this problem is to check for missing values in your data sets.

One place where special missing values work for you is in titles. For example, in many computer/printer setups, the special blank (Dec 255) prints as a blank but SAS sees it as a not-blank character. Decimal 255 is equal to hexadecimal FF. These facts come in handy when you want to have indented row titles in PROC TABULATE. Normally, PROC TABULATE will left-justify formats, automatically deleting all leading blanks.

The following exhibit shows one way to force indentation (leading blanks) using the special blank. Here the special blank character is created in the NULL Data Step, then used in the PROC FORMAT. The double quotes must be used in the PROC for the macro variable blk to work.

```
Exhibit 11: Using the special Blank
-----
DATA NULL;
  *CALL SYMPUT('blk','A0'X);
  CALL SYMPUT('blk','FF'X);
RUN;

PROC FORMAT;
  VALUE trtfmt
    0 = "&blk      Treatment A"
    1 = "&blk      Treatment B"
    2 = "&blk      Overall  "
RUN;
-----
```

In this exhibit, the indentation following the blk character is maintained. The reason for this is that the first blank after the quote is a special blank and not recognized by SAS as a blank. Remember that you have to use double quotes for macro variables to resolve. (Single quotes will not work.)

In exhibit 11, the special blank was created using macro variables. The special blank can also be created in the SAS editor (and many other editors) by a special keying sequence. Namely, you can key this blank by turning the Num Lock on, then hold down the right ALT key while keying the 255 on the keypad. Note that this procedure does not work with the left ALT key nor with the number keys located at the top of the keyboard.

Finally, the FF trick does not work on all systems. If the FF value does not work on your system, you may want to try A0 (Decimal 160). This value works on many configurations. (More information will be given in the presentation.)

CONCLUSION

There are two kinds of SAS variables: numeric and character. Each has its own set of missing values.

Numeric variables can store 28 different missing values. These are the dot, the dot-underscore, and the 26 dot-letters.

On all inputs, except LIST inputs, SAS recognizes the blank as a numeric missing and converts it to a dot. Also, SAS recognizes the 27 numeric missing values on input provided that the values being read are specified in a MISSING statement. With this statement, the dot is optional before the letter or underscore. If the dot is missing, SAS converts the letter or underscore to a dot-letter or dot-underscore for storage.

On numeric output, all missing values are displayed as a single character. Thus, a dot-underscore prints as an underscore, and a dot-letter prints as a letter. The letter values are always upper-case. The dot can print as itself or any other character, including an underscore or a letter, via the MISSING system option.

In code, special numeric missing values are denoted by two case-insensitive characters. On the other hand, they are denoted by one or two case-insensitive characters on input, and only one upper-case character on output.

In checking for a missing numeric value, it is best to always check for all possible missing values by using `<=.Z` as in the following IF statement.

```
IF VALUE <=.Z THEN PUT "*** Value is missing";
```

SAS officially recognizes only one missing character value. It is the blank. Unofficially, SAS almost always accepts the null character (ASCII code 0) and the special blank character (ASCII code 255) as missing. Depending on your specific SAS/computer/operating system setup, you may have other characters which SAS recognizes as missing (see exhibit 10). This paper shows how these extra character missing values can work against and for you. Exhibit 11 demonstrates how to use a special missing value to force indentations in titles and formats.

TRADEMARKS

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

ABOUT THE AUTHOR

Mal Foley is a software engineer. His career includes being an international computing consultant, a university professor, and the CEO of his own computing company. Recently, Mal has taken a hiatus from his position of many years as senior SAS programmer/analyst with the Department of Biostatistics at the University of North Carolina, Chapel Hill. He actively participates in user groups and will chair the 2005 Pharmaceutical Industry SAS Users' Group. Currently, Mal is an independent SAS trainer and provides in-house SAS seminars around the country.

AUTHOR CONTACT

The author welcomes comments, questions, corrections and suggestions.

Malachy J. Foley
2502 Foxwood Dr.
Chapel Hill, NC 27514

Email: FOLEY@unc.edu