

Beginner Beware: Hidden Hazards in SAS® Coding

Alissa Wise, South Carolina Department of Education

ABSTRACT

New SAS® programmers rely on errors, warnings, and notes to discover coding issues. However, it is important to note that some coding issues may be hiding in plain sight. Herein are a few examples of these issues including incomplete comparisons and inadvertently truncating variables with the IMPORT procedure. The explanations provided are meant to assist new SAS® programmers navigate these hazards so that results are clean and programs run more efficiently.

INTRODUCTION

This paper highlights six hazards of which beginners may not be aware. Examples provide useful ways to code in order to produce clean results with improved efficiency.

The topics covered include the following:

1. The COMPARE procedure – Avoid dropping variables from the comparison.
2. The IMPORT procedure – Avoid truncating variable length.
3. The FORMAT procedure – Avoid dropping significant leading zeros from output.
4. The DATA step MERGE STATEMENT – Avoid incorrect merge results.
5. The LENGTH function versus the LENGTHN function – Avoid misuse of missing values.
6. AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) CHARACTERS – Avoid issues due to special characters.

1. PROC COMPARE – AVOID DROPPING VARIABLES FROM THE COMPARISON

PROC COMPARE is a useful procedure for determining differences between datasets. However, variables can be dropped from the comparison if they are not referenced correctly. The **students** variable in table *one* is supposed to be equivalent to the **headcount** variable in table *two*.

<i>one</i>				<i>two</i>			
Obs	day	block	students	Obs	day	block	headcount
1	A	3	28	1	A	1	24
2	B	4	22	2	A	3	28
3	A	7	19	3	B	4	22
4	B	6	24	4	A	7	19
5	B	8	23	5	B	6	42
6	A	1	24	6	B	8	23

The following code steps through this process:

```
PROC COMPARE base=one compare=two;
run;
```

The PROC COMPARE results indicate that both datasets contain the same number of variables and observations. The last line in the **Observation Summary** section offers the message programmers like to see “NOTE: No unequal values were found. All values compared are exactly equal”. Before accepting this last line, notice the **Variables Summary** section where “Number of Variables in Common: 2”.

```

The COMPARE Procedure
Comparison of WORK.ONE with WORK.TWO
(Method=EXACT)

```

Variables Summary

Number of Variables in Common: 2.

.

NOTE: No unequal values were found. All values compared are exactly equal.

This PROC COMPARE is only comparing two of the three variables. The differing names – **students** and **headcount** – prevent a complete comparison. Two correction options are offered.

OPTION 1: USING VAR AND WITH OPTIONS IN PROC COMPARE

The VAR and WITH options in PROC COMPARE allow comparison even if variable names differ. All variables are included in the PROC COMPARE with one unequal comparison:

```

PROC COMPARE base=one
compare=two;
var day block students;
with day block headcount;
run;

```

```

The COMPARE Procedure
Comparison of WORK.ONE with WORK.TWO
(Method=EXACT)

Variables Summary
Number of Observations in Common: 6
.

Values Comparison Summary
Number of Variables Compared with All Observations Equal: 2.
Number of Variables Compared with Some Observations Unequal: 1.
Total Number of Values which Compare Unequal: 1.

Value Comparison Results for Variables

```

Obs	Base	Compare	Diff.	% Diff
5	24.0000	42.0000	18.0000	75.0000

OPTION 2: RENAME VARIABLES

By adding a RENAME statement to the DATA step, variables with differing names can be compared without additional options to the PROC COMPARE. Do not forget to sort both of the datasets with the same BY criteria before running the PROC COMPARE as the COMPARE procedure is comparing by observation. By using the RENAME and PROC COMPARE, all variables are included in the PROC COMPARE with one unequal comparison:

```

Data two;
set two;
rename headcount=students;

PROC COMPARE base=one
compare=two;
run;

```

```

The COMPARE Procedure
Comparison of WORK.ONE with WORK.TWO
(Method=EXACT)

Variables Summary
Number of Variables in Common: 3.
.

Values Comparison Summary
Number of Variables Compared with All Observations Equal: 2.
Number of Variables Compared with Some Observations Unequal: 1.
Total Number of Values which Compare Unequal: 1.

Value Comparison Results for Variables

```

Obs	Base	Compare	Diff.	% Diff
5	24.0000	42.0000	18.0000	75.0000

2. PROC IMPORT – AVOID TRUNCATING VARIABLE LENGTH

There are numerous options available for use with PROC IMPORT. One in particular, GUESSINGROWS, can truncate imported data if used incorrectly.

Here, the results show that GUESSINGROWS has been incorrectly set for the given data in the **teacher** variable:

```
PROC IMPORT DATAFILE="C:\Three.csv"
OUT=Three_TRUNCATED
REPLACE;
GETNAMES=YES;
GUESSINGROWS=5;
RUN;
```

Obs	Teacher	Day	Block	Students
1	Thomas	A	1	22
2	Thomas	A	3	26
3	Thomas	A	7	25
4	Fox	A	1	22
5	Fox	A	3	26
6	Fox	A	7	25
7	Willia	A	3	21
8	Willia	A	5	23
9	Willia	A	7	25

The GUESSINGROWS value tells SAS® to examine up to and including that row for the length of each variable. In the **teacher** variable, 'Thomas' and 'Fox' are in the first five rows which are being examined. Beginning with the seventh row, 'Williamson' appears; but, GUESSINGROWS has truncated it to 'Willia'.

SET GUESSINGROWS TO AN ADEQUATE VALUE

To avoid truncation, set GUESSINGROWS to one of the following: the number of rows in the dataset, 2147483647 (for Base SAS® 9.3 or later), or MAX (which is equivalent to 2147483647 for Base SAS® 9.3 or later). Warning! The larger the GUESSINGROWS, the longer it will take for the code to run.

Revisit the previous example with GUESSINGROWS equal to MAX. "Williamson" is not truncated:

```
PROC IMPORT
DATAFILE="C:\Three.csv"
OUT=Three_CORRECT
REPLACE;
GETNAMES=YES;
GUESSINGROWS=max;
RUN;
```

Obs	Teacher	Day	Block	Students
1	Thomas	A	1	22
2	Thomas	A	3	26
3	Thomas	A	7	25
4	Fox	A	1	22
5	Fox	A	3	26
6	Fox	A	7	25
7	Williamson	A	3	21
8	Williamson	A	5	23
9	Williamson	A	7	25

3. PROC FORMAT – AVOID DROPPING SIGNIFICANT LEADING ZEROS FROM OUTPUT

If data are numeric, leading zeros will not show in the output. With some data, this is not acceptable. In order to preserve the leading zeros in the output, these values are best stored as character variables. For the example here, **schoolid** is entered as a 2-digit number (see code at right). However, '07' appears as '7' in the output. To avoid dropping the significant leading zeros from the output, two options are provided. In both options, the PUT function is used to convert **schoolid** from numeric to character.

```
data four;
input schoolid 1-3 teacher $ 4-16;
datalines;
11 Thomas
07 Fox
11 Williamson
11 Smith
07 Jones
;
run;
```

OPTION 1: THE PICTURE FORMAT

The PICTURE FORMAT produces the variable **schoolid2**. The '99' contains the exact number of spaces required for the length of the data. The example here requires 2-digits. Another example is United States (US) zip codes; they require 5-digits coded as '99999'.

OPTION 2: THE ZW.D FORMAT

The Zw.d FORMAT produces the variable **schoolid3**. The w.d portion of the code indicates the number of digits required. The w value indicates the length of the data; whereas, the d value indicates the number of digits to the right of the decimal. US zip codes require z5. to format the data correctly.

In the table below, **schoolid** shows the value as numeric without significant zeros. **Schoolid2** and **schoolid3** show the same values as character with the leading zeros retained:

```
PROC FORMAT;
picture lead low-high='99';

Data four;
set four;
schoolid2=put(schoolid,lead.);
schoolid3=put(schoolid,z2.);
run;
```

Obs	schoolid	teacher	schoolid2	schoolid3
1	11	Thomas	11	11
2	7	Fox	07	07
3	11	Williamson	11	11
4	11	Smith	11	11
5	7	Jones	07	07

4. DATA STEP MERGE STATEMENT – AVOID INCORRECT MERGE RESULTS

When using the MERGE statement in the DATA step, proceed with caution. A BY statement may not be required; but if it is, the SORT procedure is also required. A look at merging without the BY statement, with the BY statement but no PROC SORT, and with the BY statement and PROC SORT follow.

The two tables to merge are seen here:

Five

Obs	schoolid	teacher	DOB	monthDOB	dayDOB	yearDOB
1	7	Fox	05/26/1970	5	26	1970
2	7	Jones	02/02/1990	2	2	1990
3	11	Smith	10/15/1981	10	15	1981
4	11	Thomas	10/02/1961	10	2	1961
5	11	Williamson	11/22/1973	11	22	1973

Six

Obs	schoolid	teacher	grade
1	7	Fox	8
2	7	Jones	7
3	11	Smith	8
4	11	Thomas	7
5	11	Williamson	8

WITHOUT THE BY STATEMENT

Without the BY statement, the merge will be completed based on order alone. In some cases, this may be acceptable. However, for the given example, the resulting table has inaccurate results. For example, Fox's DOB is 5/26/1970. However, the merge has incorrectly assigned Fox's DOB to be 10/2/1961.

```
Data combine56_NOBY_NOSORT;
merge five six;
run;
```

Obs	schoolid	teacher	DOB	monthDOB	dayDOB	yearDOB	grade
1	7	Fox	10/02/1961	10	2	1961	8
2	7	Jones	05/26/1970	5	26	1970	7
3	11	Smith	11/22/1973	11	22	1973	8
4	11	Thomas	10/15/1981	10	15	1981	7
5	11	Williamson	02/02/1990	2	2	1990	8

WITH THE BY STATEMENT BUT NO PROC SORT

If the BY statement is used without sorting, the merge fails. An ERROR and WARNING appear in the log:

```
Data combine56_NOSORT;
merge five six;
by schoolid teacher;
run;
```

```
ERROR: BY variables are not properly sorted
on data set WORK.FIVE.
WARNING: The data set
WORK.COMBINE56_ERROR_NOSORT may be
incomplete. When this step was stopped
there were 3 observations and 7 variables.
```

WITH BOTH THE BY STATEMENT AND PROC SORT

Including the BY statement with PROC SORT yields the desired results for merging tables *five* and *six*:

```
PROC SORT data=five;
by schoolid teacher;

PROC SORT data=six;
by schoolid teacher;

Data combine56_BY_SORT;
merge five six;
by schoolid teacher;
run;
```

Obs	schoolid	teacher	DOB	monthDOB	dayDOB	yearDOB	grade
1	7	Fox	05/26/1970	5	26	1970	8
2	7	Jones	02/02/1990	2	2	1990	7
3	11	Smith	10/15/1981	10	15	1981	8
4	11	Thomas	10/02/1961	10	2	1961	7
5	11	Williamson	11/22/1973	11	22	1973	8

5. LENGTH AND LENGTHN FUNCTIONS – AVOID MISUSE OF MISSING VALUES

Another source of error can be introduced if the LENGTH or LENGTHN functions are misused. If a value is missing, LENGTH returns a value of 1; whereas, LENGTHN returns a value of 0. To avoid error, first determine how missing values are to be treated. Next, select LENGTH or LENGTHN accordingly:

```
Data seven;
set seven;
length=length(x);
lengthN=lengthn(x);
run;
```

Obs	x	length	lengthN
1	12 abcd	7	7
2	1234abcdefg	11	11
3		1	0

6. ASCII CHARACTERS – AVOID ISSUES DUE TO SPECIAL CHARACTERS

Special characters should be preserved to avoid issues with tasks such as matching. SAS® does allow the use of ASCII characters. The *ASCII Table, ASCII Codes: American Standard Code for Information Interchange* website provides the “Extended ASCII Characters” chart which provides the correct key sequence for special characters.

In table *eight*, ASCII characters are preserved. In table *nine*, they are not.

eight

Obs	English	Spanish	French
1	spring	la primavera	le printemps
2	summer	el verano	l'été
3	fall	el otoño	l'automne
4	winter	el invierno	l'hiver

nine

Obs	English	Spanish	French
1	spring	la primavera	le printemps
2	summer	el verano	lete
3	fall	el otono	l'automne
4	winter	el invierno	l'hiver

A PROC COMPARE indicates that values do not match if ASCII characters appear in one dataset but not the other.

```
PROC COMPARE base=eight compare=nine;
run;
```

The COMPARE Procedure
Comparison of WORK.EIGHT with WORK.NINE
(Method=EXACT)

Value Comparison Results for Variables

Obs	Base Value Spanish	Compare Value Spanish
3	el otoño	el otono

Obs	Base Value French	Compare Value French
2	l'été	lete

CONCLUSION

As a beginner programmer, the hazards described in this paper create errors that often go unnoticed. For each one, a solution is provided so that errors are avoided. Clean results with increased efficiency are key to producing quality work.

REFERENCES

“Extended ASCII Characters”. ASCII Table, ASCII Codes: American Standard Code for Information Interchange. Retrieved September 21, 2017. Available at <http://www.theasciicode.com.ar/>.

“Usage Note 46530: Maximum value for GUESSINGROWS= value for PROC IMPORT and Number of Rows to Guess for Import Wizard when reading a comma, tab, or delimited file”. SAS® Support Knowledge Base. Retrieved September 21, 2017. Available at <http://support.sas.com/kb/46/530.html>.

ACKNOWLEDGMENTS

The author thanks Dr. Imelda Go for her support and valuable work-related SAS® training.

CONTACT INFORMATION

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

Alissa Wise
South Carolina Department of Education
1429 Senate Street
Columbia, SC 29201
awise@ed.sc.gov

TRADEMARK NOTICE

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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