

Paper 085-2008

When PROC APPEND May Make More Sense Than the DATA STEP

David W. Carr, ICON Clinical Research, Redwood City, CA

ABSTRACT

Virtually all SAS programmers (with apologies to diehard SQL codeslingers) tend to use a simple SET statement in the DATA STEP when concatenating two or more datasets in their programs. Use of the SET statement is generally the most logical and practical approach because the required code is typically very succinct and the process in most instances involves concatenation of only two datasets. However for those occasions when the SAS programming job either requires concatenation of a large number of datasets or concatenation of very large datasets (regardless of the number), the APPEND procedure is oftentimes a more sensible alternative. The purpose of this paper is to present examples of cases in which PROC APPEND may be a better choice than the SET statement in the DATA STEP and to provide some justification for why the APPEND procedure is a viable option. This presentation is based on SAS version 8.2 or above, is not limited to any particular operating system, and is intended for intermediate SAS programmers who have some familiarity with PROC APPEND and SAS Macro.

KEYWORDS: SAS, PROC APPEND, DATA STEP, SET Statement, SAS Macro

INTRODUCTION

Many SAS programmers would agree that, in most cases, use of the SET statement in a DATA STEP is the best and simplest (not to mention the most familiar) method for concatenating two or more datasets into one comprehensive SAS dataset. Similar results can be obtained in the SQL procedure via use of the UNION operator. There are times however when the use of PROC APPEND may be the most feasible (and economical) approach to concatenating multiple datasets, particularly if the job involves either many input datasets or very large datasets. In both scenarios, the APPEND procedure can be used in open SAS code and/or in conjunction with the SAS macro facility in order to achieve desired results. The SAS code required to invoke PROC APPEND is very terse and processing of the procedure quite often takes less run time (sometimes much less) than its DATA STEP and PROC SQL counterparts. Some examples of appropriate usage of the APPEND procedure, along with relevant discussion points, are presented in the following sections.

EXAMPLE 1: A LARGE NUMBER OF DATASETS IN OPEN SAS CODE

Periodically a SAS job may involve the reading in of many datasets, all or part of which at some point will become part of a single larger comprehensive dataset. Such a task may be an excellent opportunity for the programmer to employ PROC APPEND, particularly if it is necessary to perform data manipulation on each individual input dataset prior to concatenation. Consider the following sample SAS code where 10 input datasets are used to eventually create one SAS dataset:

```
data dset1;
  set samplib1.dset1;
  SAS statements ...
run;

data dset2;
  set samplib1.dset2;
  SAS statements ...
run;

. . .

data dset10;
  set samplib2.dset10;
  SAS statements ...
run;
```

The programmer can concatenate these input datasets in any number of ways. One way would be to use a SET statement in the DATA STEP immediately after each new input dataset is read in and data manipulation is performed.

```

data dset1;
  set samplib1.dset1;
  SAS statements ...
run;

data dset2;
  set samplib1.dset2;
  SAS statements ...
run;

data alldata;
  set dset1
    dset2;
run;

data dset3;
  set samplib1.dset3;
  SAS statements ...
run;

data alldata;
  set alldata
    dset3;
run;

. . .

```

Another method would be to concatenate all the input datasets at one time using a SET statement.

```

data dset1;
  set samplib1.dset1;
  SAS statements ...
run;

. . .

data dset10;
  set samplib2.dset10;
  SAS statements ...
run;

data alldata;
  set dset1
    dset2
    dset3
    dset4
    dset5
    dset6
    dset7
    dset8
    dset9
    dset10
    ;
run;

```

And the APPEND procedure could also be used in this case.

```

data dset1;
  set samplib1.dset1;
  SAS statements ...
run;

```

```

proc append base=alldata data=dset1;
run;

data dset2;
  set samplib1.dset2;
  SAS statements ...
run;

proc append base=alldata data=dset2;
run;

. . .

```

TIP: In the sample code provided above, it could be beneficial to create a small macro containing the PROC APPEND code that simply calls the work dataset following each DATA STEP. Such a macro could potentially save the programmer keystrokes in the long run.

```

%macro append(dsn);

  proc append base=alldata data=&dsn;
  run;

%mend append;

data dset1;
  set samplib1.dset1;
  SAS statements ...
run;

%append(dset1);

```

NOTE: If the work dataset ALLDATA does not already exist, the dataset is automatically created by SAS.

EXAMPLE 1 DISCUSSION

The first method demonstrated above is largely foolproof for obtaining desired results but may not be the optimal way to accomplish stacking of datasets if processing efficiency (run time) is a consideration. This is because when executing the SET statement, SAS has to physically read in both datasets in order to create the output dataset. Consequently, run time is affected by the ever-increasing size of the comprehensive dataset being constructed (in this case, ALLDATA).

The second method, in which all the input datasets are read in a single DATA STEP, is most likely a better alternative since it requires less processing time due to the fact that the input datasets are in effect read in only once. The single DATA STEP/SET statement method is in fact the best means (better too than the APPEND procedure) to concatenate datasets in this example if the data subsequently require more manipulation that can be accomplished within the same DATA STEP. One disadvantage to this method is that the programmer may have to scroll back up through the program in order to manually identify the names of all the input datasets to be read in the SET statement, particularly if the dataset names aren't sequential (i.e. DSET1, DSET2, ... DSET10).

PROC APPEND may be the best choice to concatenate the input datasets if (1) processing time is a consideration and (2) no further data manipulation is required (or no other necessary data manipulation can be performed afterward without first employing PROC SORT or another SAS procedure). The APPEND procedure can improve processing time substantially because SAS only reads in the dataset being appended (i.e. the dataset identified by the syntax 'DATA=') and in effect attaches that dataset to the end of the base dataset (i.e. the dataset identified by the syntax 'BASE='). Run time efficiency is probably the single greatest advantage that PROC APPEND has to offer in most instances.

EXAMPLE 2: A LARGE NUMBER OF DATASETS IN MACRO CODE

Similar to example 1, PROC APPEND can also be used to create a large dataset from several input datasets when used within a SAS macro. The code for such a macro might look something like this:

```

%macro appdsn(lib=,dsn=,cond=%str());

  data &dsn;
    length charcd $40;
    set &lib..&dsn (keep=membid vbtext pttext flag:);
    where &cond;

    %if %upcase(&lib)=SAMPLIB1 %then
      %do;
        source=1;
        charcd=vbtext;
      %end;
    %else
      %do;
        source=2;
        charcd=pttext;
      %end;
  run;

  proc append base=alldata data=&dsn;
  run;

%mend appdsn;

%appdsn(lib=samplib1,dsn=dset1,cond=flag3>.);
%appdsn(lib=samplib1,dsn=dset2,cond=flag2>.);
. . .
%appdsn(lib=samplib2,dsn=dset10,cond=flag5>.);

```

EXAMPLE 2 DISCUSSION

Again, as with example 1, PROC APPEND is most useful here because some unique processing of each input dataset is done prior to concatenation onto the single comprehensive dataset (and we assume no other data manipulation will subsequently be required). Another advantage granted to the programmer in this scenario is dynamic concatenation of SAS data (versus setting all the data together in a SET statement after the macro execution). This code may best demonstrate how PROC APPEND can be used with SAS macro language to gain efficiency in both code writing (fewer keystrokes) and program processing time.

EXAMPLE 3: VERY LARGE INPUT DATASETS

Another instance where PROC APPEND can be used to the programmer's advantage involves SAS jobs that require building a single dataset from very large input datasets. Assume now that we are to build a final dataset from two input datasets, DSN1 and DSN2, each of which contains about 20 million records. The syntax for accomplishing this task in PROC APPEND would be very simple:

```

proc append base=dsn1 data=dsn2;
run;

```

EXAMPLE 3 DISCUSSION

Although requiring much less code than the previous 2 examples, conceptually this final example perhaps best demonstrates the power of PROC APPEND in terms of run time efficiency. The APPEND procedure in this case may process as much as three times more quickly than a simple SET statement in a SAS DATA STEP. Again, this is due to the fact that the procedure only actually processes (or reads in) the DSN2 dataset for the append in contrast to the SET statement which must read in and process both datasets DSN1 and DSN2. This may have significant implications for programmers working in the health care/claims and financial industries where processing of millions of records of data is fairly routine.

THERE'S ALMOST ALWAYS A CATCH

As with many other ways of doing things within SAS, there do exist some costs and considerations to be addressed when using the APPEND procedure. The primary concern for the programmer in using PROC APPEND is trying to ensure that the datasets involved in the procedure have the same variables and variable attributes. For instance, suppose we are appending two datasets, DSN1 and DSN2:

```
proc append base=dsn1 data=dsn2;
run;
```

Suppose also that the two datasets have the exact same variables but the common variable CHARVAR has a length of \$35 in the DSN1 dataset and \$40 in DSN2. As a result, SAS would return the following message in the LOG:

```
WARNING: Variable charvar has different lengths on BASE and DATA files (BASE 35
DATA 40).
ERROR: No appending done because of anomalies listed above. Use FORCE option
to append these files.
```

As the LOG message implies, no appending is done and use of the FORCE option is necessary to append the input files. The consequent required syntax to correct this issue would look like this:

```
proc append base=dsn1 data=dsn2 force;
run;
```

There are two items that still should be noted here. First, even though the above code will produce a successful concatenation of datasets, SAS may return the following message in the LOG file:

```
WARNING: Variable charvar has different lengths on BASE and DATA files (BASE 35
DATA 40).
NOTE: FORCE is specified, so dropping/truncating will occur.
```

This could be an issue for the programmer if he/she works in an environment where ERROR and WARNING messages in the LOG file are unacceptable in production runs.

The second point to be made here is that when encountering differing attributes for the same variable in the APPEND procedure, SAS accepts the attributes from the variable in the base dataset (in this case, DSN1). From the previous example, this would mean that the length of CHARVAR in the output dataset will be \$35 since that is the length that exists in DSN1. Thus, values for CHARVAR from DSN2 that are greater than 35 characters in length are truncated.

So what about instances in PROC APPEND where some variables exist in one dataset but not in the other? Assume now that we are appending datasets DSN3 and DSN4:

```
proc append base=dsn3 data=dsn4;
run;
```

Assume also that the two datasets have the same variables and variable attributes with the exception that dataset DSN3 has a variable named TESTFLG which does not exist in DSN4. Although successful concatenation of these datasets will occur, SAS may produce the following message in the program's LOG file:

```
WARNING: Variable testflg was not found on DATA file.
```

Again, this could potentially be problematic in SAS environments where WARNING messages in the LOG are forbidden.

USEFUL TIPS AND OTHER CONSIDERATIONS FOR USING PROC APPEND

To reiterate, if it is possible for the programmer to (1) read in all input datasets with a SET statement without any prior data manipulation and (2) perform any subsequent necessary data manipulation within the same DATA STEP, then DATA STEP processing with the SET statement will almost always be the best method for creating a comprehensive dataset from several input datasets. If however one of the scenarios previously described should present itself to the programmer, the following are some tips and/or points to keep in mind when using the APPEND procedure:

- Strive to ensure that both datasets have the same exact variables and variable attributes to avoid a 'messy' log (i.e. avoid using the FORCE option if at all possible).
- It is always best to explicitly define the dataset name using the 'DATA=' option in the PROC APPEND statement. If 'DATA=xxxx' is not specified, SAS uses the dataset most recently created.

- Generally speaking, the use of PROC APPEND is probably best restricted to working with WORK datasets (i.e. avoid using PROC APPEND with permanent datasets).
- To improve processing efficiency in PROC APPEND, try to designate the larger of the two datasets as the BASE dataset if possible.

CONCLUSION

In summary, there are occasions when the use of PROC APPEND may be a more sensible approach to concatenating datasets, particularly if the SAS job involves either many input datasets or very large input datasets. The procedure requires very concise code, can be used effectively in concert with SAS macro code, and can save the programmer significant processing time if utilized astutely. Programmers employing the APPEND procedure are best advised to use care in ensuring that any datasets used by the procedure have identical variables and variable attributes. Probably the greatest advantage offered by PROC APPEND is the potential for a substantial improvement in run time efficiency, and this could have important implications for industries in which processing of very large SAS datasets is commonplace. However, as with other SAS methodologies, any gains in processing efficiency must always be weighed against other programming considerations.

ACKNOWLEDGEMENTS

The author would like to extend a special thanks to Jackie Lane, Stephen Hunt, Jasmin Fredette, and Brian Fairfield-Carter for their input to and review of this paper, as well as for their ongoing support for and contributions to progress, creativity, and innovation.

CONTACT INFORMATION

The author can be contacted at:

David W. Carr
ICON Clinical Research
555 Twin Dolphin Drive, Suite 400
Redwood City, CA 94065
Work Phone: 215.616.4966
E-mail: carrd@iconus.com