Paper 084-2011

PROC COPY and PROC APPEND: A Developer's Secrets

Diane Olson, SAS Institute Inc., Cary, NC

ABSTRACT

This presentation gives you the inside tricks about PROC COPY and PROC APPEND that the developer is dying to tell you. Want a clone of your data set (attributes and data) written to the same library? You can! Want to copy your data set using data set options? You can! Need a data set converted to a particular data representation and encoding? You can! Make the most of these basic, necessary procedures by applying the tricks you'll learn.

INTRODUCTION

The COPY and APPEND procedures are basic tools; by understanding the way they work internally, your SAS® programs can benefit. This paper contains facts not previously documented about these two procedures. It isn't that these facts were meant to be secrets; I was surprised to learn that they weren't commonly known. While talking with an experienced SAS programmer at a SAS Global Forum, I discovered that he did not know these facts and was excited to hear about them. I wanted to be sure everyone else had the same opportunity.

TERMS AND STRUCTURE

To understand the answers to the questions posed in the abstract, you need to understand some basic terms and the structure in PROC COPY and PROC APPEND. This paper assumes that you are working with the BASE engine. Other engines might or might not comply with the same rules.

ATTRIBUTES

When we talk about the attributes of a data set, what do we really mean? Basically, we mean all the items that are displayed when you look at the output from the CONTENTS procedure. They are the data that describe the observational data. These attributes, along with the observations, make up the entirety of the data set. Change one attribute, and you might have different results when processing the data set or you might experience performance problems you didn't have before. Examples of these attributes that we'll discuss include indexes, integrity constraints (general and referential), compression, sort indicator, audit trails, data representation, encoding, and engine. There are many more attributes, but we'll focus on these.

In the world of PROC COPY, a true copy of a data set means exactly that — take this data set in this library and create a new data set in another library that contains exactly the same attributes and observations. Exactly the same observations are in the output data set with one slight difference: all the deleted observations are removed and not copied to the output data set.

Because PROC COPY does not allow data set options, all the input data set's attributes are copied to the output data set. There are two exceptions; for integrity constraints, the CONSTRAINT=YES option must be in effect for the constraints to be copied. Also, the DATECOPY option must be supplied if you wish to keep the same created and modified dates on the output data set.

A true copy is something that we often need, but we also need all the other ways of copying a data set. This means allowing the attributes on the output data set to be changed. Since data set options are not allowed, PROC COPY accomplishes this by using the LIBNAME and/or system options for some attributes when the NOCLONE option is used.

THE CLONE/NOCLONE OPTIONS

PROC COPY limits the attributes of the output data set that can be changed. With the CLONE option in effect, only one attribute can be changed; when NOCLONE is set, six other attributes are also allowed to vary. CLONE is the default value. When it is set, all the attributes of the input data set are used to create the output data set. One attribute might be changed during this operation: the engine associated with the output data set. This is determined by the engine that was assigned to the OUT= libref used in the PROC COPY statement.

Six other attributes (as well as the output engine) can be changed when the NOCLONE option is in effect. These include BUFSIZE, COMPRESS, DATA REPRESENTATION, ENCODING, REUSE, and POINTOBS. When NOCLONE is used, the values of these six attributes are taken from the OUT= libref or from system options if the libref options were not set or are not available. In the example code below, BUFSIZE is set as a system option; it is not one of the options available in the LIBNAME statement. COMPRESS is a LIBNAME option, so its value would be used over the COMPRESS system option.

```
/* This example assumes that the input data set does not have compression and has a
different buffer size than 4096. */
options bufsize=4096;
libname outlib v9 'physical location'
  compress=yes;
proc copy
  in=mylib out=outlib;
```

```
run;
```

That allows a bit of flexibility, but there are a lot more options in the SAS system you might want to change on the output data set.

BLOCK I/O

PROC COPY does not allow you to use data set options. The only attributes we can change with PROC COPY are the engine or the six options affected by NOCLONE. When we use NOCLONE, all six attributes are set with the LIBNAME or system options – none can be inherited from the input data set.

You know that you can get exactly the data set you want if you use the DATA step SET statement. So why not do that? Two reasons: First, you might want all the rest of the attributes on the data set to be passed to the output data set without having to individually specify them. Second, you might want the performance gains that come with block I/O.

The COPY and APPEND procedures can make use of block I/O in order to move observations between the input and output data sets; especially for large data sets, this is quite a performance boost. Instead of moving one observation at a time, entire blocks of data can be added to the output data set at once.

Block I/O is used whenever rows of observations can be taken from the input data set and plugged right into the output data set without changes. You can imagine how certain attributes can make this impossible; if you are copying from one data representation to another, for example, the observations have to go through transcoding, so can't be merely moved from the input data set. Other attributes that we will discuss later also affect usage of block I/O.

STRUCTURE OF PROCEDURES

One important thing to know about PROC COPY, PROC APPEND (in some cases), and PROC MIGRATE is that they share a code base. Some of their duties are very similar, so the code is exactly the same, with some "ifs" thrown in for the peculiarities of each procedure.

With this in mind, let's find out how to accomplish the goals set out in the abstract.

CREATE A CLONE IN THE SAME LIBRARY

PROC COPY does not have syntax that would allow you to make a copy of a data set in the same library, as there is no syntax to specify a new data set name. However, when the output data set is non-existent in a PROC APPEND, the procedure uses the same code base as PROC COPY. PROC APPEND has the capability of naming an input and output data set. Using this capability, it is simple to make a copy of your data set in the same library:

```
/* MYLIB.NEWDATA must not exist before this step. */
proc append data=mylib.mydata
  base=mylib.newdata;
run;
```

Make sure you see the following note in the log:

```
NOTE: BASE data set does not exist. DATA file is being copied to BASE file.
```

The engines are the same since we are using the same libref, and we haven't used any data set options that could require records to be read individually, so block I/O is used. The common code that is used to append to a non-existent data set is the NOCLONE COPY code. Deleted observations in the input data set are not copied to the output data set. You need to be aware of the LIBNAME and system options setting for the options mentioned in the CLONE/NOCLONE section above. Sorted data sets retain the sort indicator on the output data set, including linguistically collated data sets. Indexes and general integrity constraints are added to the output data set; however, referential integrity constraints are not added. When the input data set has referential integrity constraints, you see a note like the following:

```
NOTE: Referential constraints for MYLIB.MYDATA.DATA were not copied.
```

Audit trails are not added to the output data set; PROC MIGRATE is the only method of moving an audit trail forward.

COPY WITH DATA SET OPTIONS

As we've already mentioned, PROC COPY does not allow you to use data set options. But there are times when you'd like to have a copy of the data set with just some slight tweaks. Of course, now you know the answer! PROC APPEND allows you to use data set options, and appending to a non-existent data set allows you to do a NOCLONE COPY. Using the combination, you can copy with data set options.

Perhaps you'd like to have a data set that is encrypted so that you can protect the contents. Furthermore, you'd like to subset the contents of the new data set.

```
/* MYLIB.NEWDATA must not exist before this step. */
proc append
  data=mylib.mydata( where=(name contains 'C'))
  base=mylib.newdata( pw=quiet encrypt=yes);
run;
```

Again, you are looking for this note in the log:

```
NOTE: BASE data set does not exist. DATA file is being copied to BASE file.
```

Remember that the NOCLONE COPY code was used, so the six options discussed in the CLONE/NOCLONE section reflect the value of either the LIBNAME or the system option, not that of the input data set. You can change those using data set options, which would override the NOCLONE values.

In the case above, note that we have used a WHERE data set option to subset the data being added to the output data set. Because individual analysis of each observation is required to determine if it fits the WHERE clause, block I/O would not be used.

COPY TO NEW DATA REPRESENTATION AND/OR ENCODING

Of course, now you're thinking that you know how to do this one! You can use the same trick with PROC APPEND to a non-existent data set and use data set options, or even set the LIBNAME options for OUTREP (for data representation) and/or ENCODING. You can do that, but then you also have to think about the other options that NOCLONE affects.

In SAS 9.3, there is a preproduction option for PROC COPY called OVERRIDE that allows you to set the data representation or encoding (or both) and not affect the other attributes on the output data set.

The OVERRIDE= option in the COPY statement of the DATASETS procedure is preproduction in SAS 9.3.

OVERRIDE=(<OUTREP=value><ENCODING=value>)

> overrides specified output data set options OUTREP and/or ENCODING copied from the input data set. The OVERRIDE option is a preproduction feature.
>
> Restriction: The OVERRIDE option cannot be specified if the NOCLONE option is used.
>
> Tip: When copying a data set stored in another host data representation or encoding, the default (or CLONE) behavior of COPY is to preserve the other host data representation or encoding in the new copy of the data set. By specifying OVERRIDE=(OUTREP=SESSION ENCODING=SESSION) in the COPY statement, the new copy of the data set is created in the host data representation and encoding of the SAS session executing the PROC COPY.

OUTREP and ENCODING are important options because these allow you to deliver data to someone when you do not share the same operating system or language. Delivering the data in their session data representation and encoding allows them full access to the data and no transcoding through Cross Environment Data Access (CEDA). Assume we are using a WINDOWS_32 data representation with WLATIN1 encoding, but we want to send a data set to someone using HP_UX_64 and WLATIN2 encoding:

```
proc copy in=mylib out=outlib
  override=(outrep=hp_ux_64 encoding=wlatin2);
  select mydata;
run;
```

Note that the OVERRIDE option allows you to specify the keyword SESSION that would determine the data representation and the encoding of the current SAS session. This allows you to convert a data set to native access when you are seeing messages about CEDA being used, such as:

```
NOTE: Data file MYLIB.MYDATA.DATA is in a format that is native to another host or the
file encoding does not match the session encoding. Cross Environment Data Access is used,
which might require additional CPU resources and might reduce performance.
```

EFFICIENCY

As mentioned above in the BLOCK I/O section, any time you can either append or copy and use block I/O instead of record I/O, you increase the performance of that step. Knowing the structure of your data combined with rules for using block I/O can guarantee that increase. There are, of course, times when you need the features that turn off the block I/O, and then you know

to expect slower performance.

Situations where block I/O cannot be used are summarized in the list below. (In the terminology here, for a PROC APPEND step, the DATA= data set is the input data set and the BASE= data set the output data set. For a COPY step, the IN= data set is the input data set and the OUT= data set is the output data set.)

- The input and output engines are not the same. Note that this is a general rule, but there are exceptions. For example, Version 7 BASE engine and Version 9 BASE engine might have the same layout and might allow block I/O between the two engines.
- The variables in the input and output data sets are not the same. This applies to PROC APPEND. Keep in mind that using the DROP and KEEP options can cause this difference.
- The variable attributes, such as length or format, are not the same on the input and output data set.
- There is a WHERE clause present.
- Member-level locking on either the input or output data set is not available.
- The OBS or FIRSTOBS option is in effect.
- Referential integrity constraints exist on the output data set.
- Cross Environment Data Access (CEDA) is being used on either the input or output data set.
- Either the input or output data set is compressed.
- There is an unsuspended AUDIT trail on the output data set.

Another efficiency factor to consider is the size of the data sets involved. If you can use the larger of the two as the BASE data in an APPEND, fewer records to move via record or block I/O is faster. Of course, this isn't a consideration in a COPY step.

OPTION MSGLEVEL=I

Informational messages are written to the log when OPTIONS MSGLEVEL=I is set and the PROC APPEND or PROC COPY step is run. You might not want this turned on in production, but it is extremely helpful during development. Messages are output that indicate when block I/O is not being used, as well as the reasons why not. Other information about processing is also output. Some of this information is pertinent to the SAS code developer; some of it is used by developers at SAS to debug problems in the field.

In SAS 9.3, some messages have been added for PROC APPEND when MSGLEVEL= I is set. Here is an example of the log during a PROC APPEND when indexes exist on the BASE data set:

```
data mylib.mydata(index=(i x));
  do i = 1 to 1000;x=i;
   output;
  end;
run;
NOTE: The data set MYLIB.MYDATA has 1000 observations and 2 variables.

data mylib.mydata2; set mylib.mydata; run;
NOTE: The data set MYLIB.MYDATA2 has 1000 observations and 2 variables.

proc append base=mylib.mydata
  data=mylib.mydata2; run;
NOTE: Appending MYLIB.MYDATA2 to MYLIB.MYDATA.
NOTE: 1000 observations added.
NOTE: The data set MYLIB.MYDATA has 2000 observations and 2 variables.

options msglevel=i;

proc append base=MYLIB.MYDATA
  data=MYLIB.MYDATA2; run;
NOTE: Appending MYLIB.MYDATA2 to MYLIB.MYDATA.

INFO: Engine's fast-append process in use.
INFO: 12:47:03 Reading DATA file, updating BASE file
INFO: Starting data set size is 10 pages
INFO: Engine's block-read method is in use.
INFO: Engine's block-write  method is in use.
INFO: Ending data set size is 14 pages, 4 added.
INFO: 00:00:00 Elapsed time
INFO: 12:47:03 Finished updating BASE file

INFO: Starting index file size is 23 pages.

INFO: 12:47:03 Updating index i
```

```
INFO: Ending index file size is 25 pages, 2 added.
INFO: 00:00:00 Elapsed time
INFO: 12:47:03 Finished updating index i

INFO: 12:47:03 Updating index x
INFO: Ending index file size is 27 pages, 2 added.
INFO: 00:00:00 Elapsed time
INFO: 12:47:03 Finished updating index x

NOTE: 1000 observations added.
NOTE: The data set MYLIB.MYDATA has 3000 observations and 2 variables.
```

When questioned about the 00:00:00 elapsed time, the BASE engine developer said, “It’s not called fast-append for nothing!” If you increase the number of observations that you are appending, however, you will see the elapsed time increase.

In this example, you can also see that block I/O is not an all-or-nothing situation. It is possible to use block reads even if you can’t use block writes. The messages indicate whether you are using block I/O for the input and the output data sets.

APPENDING TO A ZERO-OBSERVATION DATA SET

A number of customers I have spoken with create a master data set with attributes set on it before adding observations. This is especially true when they wish to use integrity constraints to filter the values of variables to be added to the data set. A zero-observation data set is not treated as a non-existent data set would be. None of the cloning is true for the attributes from the input data set. Usually that is what you desire; if your input data set is sorted, you would like that to retain the sort indicator. Use the GETSORT option in this situation to retain the sort indicator of the data. This only works for appending to zero-observation data sets. Note that this works for linguistic collation sort information as well.

```
proc append
  data=mylib.mydata base=mylib.zero_obs
  getsort;
run;
```

As an aside on creating zero-observation data sets, keep in mind that even when an observation is deleted, that doesn’t mean the observation is physically removed from the data set. It can still exist, though it is marked as deleted. If you are using zero-observation data sets, it is much more efficient to create a new data set with the attributes from a pre-existing data set rather than deleting all the observations. An easy way to do that is to use PROC APPEND with the OBS=0 data set option on the input data set. While general integrity constraints are added, referential integrity constraints are not.

```
/* Note that at this point, mylib.zero_obs does not exist. */
proc append data=mylib.mydata(obs=0)
  base=mylib.zero_obs;
run;
```

If you need to add referential integrity constraints or you choose to add observations to your data set first, you can obtain the indexes and integrity constraints from an existing data set and apply them on your new data set. To do this, make use of the RECEATE data that is produced in PROC CONTENTS OUT2=data set; the OUT2=data set contains the syntax to recreate indexes and integrity constraints from the input data set.

```
proc contents noprint data=mylib.mydata
  out=contents_out
  out2=contents_out2;
  run;

proc sql noprint;
  select recreate into :recreatem
  separated by " " from contents_out2;
  quit;

proc datasets lib=mylib nolist;
  modify zero_obs;
  &recreatem;
  quit;
```

If you wish to filter your data by creating a unique index or unique integrity constraint, the data must be added after the PROC DATASETS step. If you know your data conforms to the rules you place on it via indexes and integrity constraints, then you can add the data before the PROC DATASETS step.

CONCLUSION

If you need to copy or append your data sets, there is a way to get any result that you desire using either the COPY or APPEND procedure. Knowing more about what attributes are on your input data set as well as the facts about when block I/O is used gives you the ability to plan for the best results. Now the secret is out!

ACKNOWLEDGMENTS

Thanks to Maggie Marcum for the OVERRIDE documentation and her review. Thanks to Art Jensen for his SAS code in the APPENDING TO A ZERO-OBSERVATION DATA SET section along with all his sage advice. Thanks to a great team for the review and for supporting these procedures – Art Jensen, Miguel Bamberger, Doug Ragsdale, and Claire Bonney.

CONTACT INFORMATION

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

Diane Olson
SAS institute Inc.
SAS Campus Drive
Cary NC 27513
E-mail: Diane.Olson@sas.com