P's Porridge: What's In Those Brown Manuals?

Jack Hamilton

First Health,, West Sacramento, California
JackHamilton@FirstHealth.com

Overview

The SAS System has a documentation problem. Although it is generally well documented, information about any particular product or feature may be scattered over many different manuals.

The Institute compounds this problem by constantly upgrading the product, making their documents obsolescent almost as soon as they're out the door. As problems go, this is a good one to have, but it's easy to miss useful new features.

Documentation on new features usually comes in Changes and Enhancements documents, which have brown binders. They originally had names of the form ***SAS Technical Report P-nnn: Changes and Enhancements to xxx***, but recently the Institute has dropped the first part of the name. Nevertheless, these manuals are widely referred to as Tech Reports.

The changes and enhancements documents published since the original version 6 manuals are several inches thick. It's not possible to describe them all in a few minutes, so I will concentrate on a few new features which I think are the most useful.

Caveats and Cautions

A complete list of current manuals is available from SAS Institute's Book Sales Department at +1 (800) 727-3228, or on the World Wide Web at

<http://www.sas.com>

Not all enhancements presented here will work on all platforms and releases.

Release numbers in SAS documentation seem to be only guidelines, not rules. Some enhancements are documented for 6.09 on the Alpha platform, but are actually present in 6.08. Others are documented for 6.07, but aren't in 6.08 yet.

Manuals Used In Writing This Paper

SAS Language Reference, Version 6, First Edition; Document number 56076

SAS Technical Report P-204: SAS Software: Changes and Enhancements, Release 6.06; Document number 59121

SAS Technical Report P-222: SAS Software: Changes and Enhancements, Release 6.07; Document number 59139

SAS Technical Report P-242: SAS Software: Changes and Enhancements, Release 6.08; Document number 59159

SAS Software: Changes and Enhancements, Release 6.10; Document number 55120

SAS Software: Changes and Enhancements, Release 6.11; Document number 55300

What's New for the 6.09 Enhanced Release of SAS Software; Document number 55569

What's Hot in Release 6.12; <http://www.sas.com/service/techsup/wn612/wn612.pdf>

Interesting Features by SAS Release

Some of the new features documented in the brown manuals are shown below. Within each release, they're not ordered in any particular way.

A Bonus from the Base Manual

Column Pointer Controls

This feature is described in the base Language Reference Manual, but I didn't know about it until I read through the manuals while preparing this paper.

There's a new way to do formatted input: move the input pointer to the next character after a specified string. Suppose you had input lines like these:

```
SFO RAIL BART MUNI .7 BART .2
SAC RAIL STD STD .9 AMTRAK .1
YOL BUS YT AMTRAK .1 YT .9
```

You want to read the city, primary carrier, and percentage of traffic for the primary carrier. First, read the city and primary carrier, then locate the primary carrier in the remainder of the input line. After that, it's easy to read the value:

```
input @1 city $3.
      @5 type
         carrier
      @(trim(carrier)) percent;
```

If you have a fixed string, you can just put it in quotes after the at-sign.

Release 6.06 (P-204)

These features are actually included in the base manuals, but seem to be little known, so I'm including them. They are changes from 5.18 behavior.

TEMPORARY arrays

The _TEMPORARY_ array is a mechanism for creating variables which are not stored in the PDV. Access is faster than to regular variables, and there is no danger of name collision with one of your datastep variables.

Because these variables are not referred to by name, only by subscript, and are not stored in the PDV, they cannot be written to an output dataset.

```
array xyz{3} _temporary_ (7 11 13);
first = a*test{1};
```

FILENAME= and FILEVAR= options in the FILE statement

The FILENAME= option allows you to find the complete OS filename of a file.

The FILEVAR= option allows you to dynamically change the name of the file you're using. You could, for example, calculate the name of the file based on your data. Here's a sample job log, with some parts of the output deleted to save space:

```
61 data _null_;
62 infile cards;
63 input indate mmddyy8.
64 count 4.;
65 fv = 'd' ||
66 put(indate, date7.);
67 file dummy filevar=fv
68 filename=fn;
69 put count;
70 cards;
NOTE: The file DUMMY is:
      FILENAME=P:\d07JUN55,
      RECFM=V,LRECL=256
NOTE: The file DUMMY is:
      FILENAME=P:\d14FEB23,
      RECFM=V,LRECL=256
73 run;
```

Field alignment with the PUT statement

The -L, -C, and -R options allow you to left, center, or right-justify the output from a format. For example:

```
put x dollar7.2-c;
```

would center the value of X in the outout field.

INPUT and PUT functions allow ? and ??

If the value you want to convert with the INPUT or PUT function can't be converted correctly, you will ordinarily see an error message in the SAS log. Use of ? or ?? prevents these messages.

```
s = input(datestr, ?? mmddyy.);
```

New host system informats such as $ASCII.

It is often necessary to read data using a different encoding (ASCII vs. EBCDIC). New formats allow you to read data written in formats other than the ones native to your system.

The IN operator

The IN operator is a quick way to test whether a value is in a list of other values. For example:

```
if title in ('Dr', 'Sir', 'Col');
```

Unfortunately, the values in the list must be constants when the IN operator is used in the data step.

A BY statement applies to only the preceding SET, MERGE, or UPDATE statement.

In 5.18, you could use only one SET, MERGE or UPDATE statement in a data step that contained a BY statement, or bad things would sometimes happen. This restriction has been lifted.

```
data new;
if _n_ = 1 then
   set total(keep=gtot);
set detail
     (keep=group btot amt);
 by group;
bpercent = btot/gtot;
dpercent = amt/gtot;
<other code>
run;
```

Release 6.07 (P-222)

Multi-unit intervals in INTxx functions

INTCK and INTNX are little-known and much-misunderstood functions which allow you to calculate the interval between two dates or times, or advance a certain interval from a given date or time. New options make these functions much more useful. You can now shift the starting point of an interval to something meaningful. For example, if you used a quarter system where the quarters started in February, May, August, and November, and you wanted to know when the next quarter started after February 15, you could use this code:

```
data _null_;
nextqtr = intnx('month3.2',
                '15feb96'd, 1);
put nextqtr=date7.;
run;
```

which would print

```
NEXTQTR=01MAY96
```

INPUTC, INPUTN, PUTC, PUTN functions

These functions allow you to decide on a format at runtime, rather than hardcoding it..

```
data _null_;
input @1 fmt $9.
      @11 number $10.;
outnum = putn(number, fmt);
put @1   number  $10.
    @10  fmt     $11.
    @20  outnum;
cards;
dollar8.2 123.45
comma6.0  678.9
binary10  11
;;;; run;
```

would print

```
123.45   dollar8.2 $123.45
678.9    comma6.0  679
111      binary10. 0001101111
```

Dataset passwords

Version 5 of SAS allowed password protection of datasets. Apparently, many people missed this feature, because it's back.

```
data salary (read=green write=red
                  alter=yellow);
```

I suspect that the encryption methods used are of the "protect against your little sister" variety rather than the "protect against big government agencies" variety, and in any case haven't been subjected to public scrutiny, so don't depend on SAS passwords for real security.

SORTEDBY dataset option

If a dataset is sorted, but wasn't sorted by SAS, you can tell SAS about it. Some operations will be performed more quickly as a result.

```
proc sql;
 select *
  from  x.y (sortedby=usage)
  group by usage;
```

Data step views

Just as SQL views are a shortcut method of telling SAS how to process data using SQL, datastep views are a shortcut method of telling SAS how to process data using datastep code. There are several benefits to using datastep views. The data step doesn't have to be compiled every time you use it. You can hide the intricacies of your code from other users. You can make coding changes transparent. And you can sometimes greatly reduce CPU time and I/O by using a datastep view and a PROC rather than a regular data step followed by a PROC.

The following data step will create a view named NEW. When it is accessed, a real dataset (not a view) named work.errors will be created.

```
data new errors / view=new;
set old;
if date = . then
   output new;
else
   output errors;
run;
```

The documentation says that output views are not currently supported. I suppose this means that someday they will be.

KEY= option in SET and MODIFY statements

You can go directly to an observation in a SAS dataset if it is indexed and you know the key value for the observation. This has the potential to replace a lot of messy coding now done with formats or sorts and merges.

```
set claims
    (keep=claimno patient billed);
set patients
    (keep=patient lastname)
    key=patient;
```

When you use the KEY= option, be sure to use the _IORC_ variable to check whether the lookup succeeded.

DLM= and DSD options in the INFILE statement

SAS has long suffered from the inability to read text files which other programs can read with ease. These new options make it much easier to read files which contain tab- or comma-delimited data.

This statement will read a comma-delimited file, where text strings may optionally be inside quotes:

```
infile x dlm=',' dsd;
```

There's no corresponding facility for writing a comma-delimited file, but it's straightforward to do using the PUT statement in conjunction with the QUOTE function.

DSD implies DLM=',', which is probably the most common case. To read a tab-delimited file, use

```
dlm='09'x dsd
```

Version 6.12 of SAS contains an Import/Export wizard which makes it much easier to read and write external files.

BY values usable in titles

You can now include the current BY value in the title created by a TITLE statement. This means that you can have much better looking titles. Alas, this applies only to output from PROC's, not to output from data steps.

```
options nobyline;
title "Payments for region #BYVAL1";
proc print;
 by region;  pageby region;
run;
```

In this example, #BYVAL1 means "show the current value of the first variable in the BY list". You can also Insert the name of a BY variable with #BYVAR, or get complete BY variable information with #BYLINE.

CALL EXECUTE

CALL EXECUTE allows you to write a SAS program from within SAS, without resorting to writing an external file and reading it back in (which there is no general mechanism to do in SAS).

The functionality that this provides may be replaced by the CATALOG engine in 6.12.

FMTSEARCH= system option

The FMTSEARCH option allows you to tell SAS where to look for formats. You may specify a one-level or two-level name. FMTSEARCH= makes it much easier to maintain department or company format libraries.

```
options
   fmtsearch=(myfmts deptfmts
              compfmts myfmts.old);
```

Use existing formats in PROC FORMAT

You can now use existing formats inside your own format definitions. This means, for example, that you can redefine the output of specific values without having to duplicate all of SAS's extensive format library for other values. Here's an example from the manual:

```
proc format;
 value status
  low-'01MAR1990'd = [date7.]
  other = 'OVERDUE';
```

The QUOTE function

The QUOTE function adds double quotes to the outside ends of a character expression. If the expression contains internal double quotes, they will be doubled.

The following code would write a tiny comma-delimited file:

```
QA = quote(a);
QC = quote(c);
put QA "," b "," QC;
```

CALCULATED component in SQL Queries

Sometimes it is necessary to use the same calculated value two or more times in a single SQL statement. For example, suppose you want calculate a percentage of savings, and also select only those observations with low savings. The SQL code might look like this:

```
proc sql;
 select *,
        billamt-discamt as savings
 where  billamt-discamt < 100;
```

The CALCULATED component allows you to say "use the variable I've already calculated", instead of repeating the calculation:

```
proc sql;
 select *,
        billamt-discamt as savings
 where  calculated savings
          < 100;
```

Release 6.08 (P-242)

ERRORCHECK= system option

ERRORCHECK=STRICT causes your program to go into syntax-check mode if there is an error in a LIBNAME or FILENAME statement, and to abend if a %INCLUDE statement fails.

MODIFY, REPLACE, and REMOVE statements

(This replaces the description in P-222)

The MODIFY statement allows you to modify a SAS dataset in place, without creating a copy (which is ordinarily done, invisibly to you, when you change an existing dataset).

Although there are dangers to this technique (on some operating systems, the dataset may be damaged if your program abends), it can greatly increase processing speed and reduce the amount of disk space needed.

The REPLACE statement, used after MODIFY, updates the current observation. The OUTPUT statement writes a new observation to the end. REPLACE is the default action when used after MODIFY (the usual datastep default is OUTPUT). The REMOVE statement marks the observation as deleted.

INDEX= dataset option

(This replaces the description in P-222)

When indexed datasets were first announced, you had to use PROC DATASETS or PROC SQL to create the index.

You can now create indexes while creating the dataset. Here are some examples:

```
data new (index=(claimno));

data new (index=(claimno claimdt));

data new (index=(c=(claimno
                    claimdt)));
```

Indexes make some operations substantially faster, but increase the time and disk space needed to create a dataset.

Release 6.09E (Document 55569)

The Data Step Debugger

This is an important new feature, perhaps the most significant in 6.11. It's beyond the scope of this paper.

The SORTEDBY= dataset option is supported for views and stored datastep programs.

Self-explanatory; see the explanation of the SORTEDBY option under 6.07, above.

New options on the %PUT statement.

New options on the %PUT statement let you print some or all macro variables.

`%PUT _ALL_` will print all currently defined macro variables.

`%PUT _AUTOMATIC_` will print all automatic macro variables (because these variables vary by operating system and installed products, it has been difficult to track them all down in the past. This should help).

`%PUT _GLOBAL_` will print all user-defined global variables.

`%PUT _LOCAL_` will print all user-created macro variables available in the currently executing macro.

`%PUT _USER_` will print all user-created macros.

Writing multiple host variables in PROC SQL

The SEPARATED BY component of the INTO clause lets you write values from several observations into a single macro variable.

```
select name
into :names
separated by ', '
from userlist;
%put &names.;
```

might print

```
Jan June Joan John
```

You can also specify multiple macro variables with a hyphen or THROUGH; each will be set with the value of a different row:

```
select name
into  :name1-:name4
from   userlist;
%put &name1 &name2 &name3 &name4;
```

would print the same result as the earlier example.

Storing MPRINT Output in an External File

You can store the output from the MPRINT option into an external file, suitable for later use standalone. To do this, create an MPRINT fileref pointing to the new file, and specify the MPRINT and RESERVEDB1 system options:

```
   options mprint reservedb1;
   filename mprint 'test.sas';
```

This will be useful for debugging macros.

Release 6.12 (What's Hot)

The %SYSFUNC macro function

This new macro function gives you the ability to execute data step functions inside a macro.

The %SYSFUNC macro also has the ability to apply a specified format to the result of the function.. For example, to put the current date into a title using the WORDDATE format, you could use the statement

```
   title "%sysfunc(date(), worddate.)";
```

Release 6.11 (Document 55300)

FILENAME now allows catalog, ftp, and socket files.

Catalog access provides an easy way to read and write data or programs stored in SAS catalogs; ftp and socket access provide another way to look at data on remote machines connected to your network.

```
   filename prog catalog

'sasuser.progs.readfmt.source';
   %include prog;
   filename claims ftp
            'claims.dat'
            host='ftp.hccompare.com';
```

Note: Not available in SAS for OpenVMS 6.09 TS048.

New dictionary tables: MACROS, TITLES

The DICTIONARY.TITLES table allows you to retrieve information about all the titles and footnotes that are currently defined. This table can be used outside of PROC SQL through a permanent SQL view, SASHELP.VTITLE.

The DICTIONARY.MACROS table allows you to look at the names and definitions of all macro variables. SASHELP.VMACRO is the SQL view.

Note: Not available in SAS for OpenVMS 6.09 TS048.

VMS-Specific Enhancements in TS048

The text of the online help for these items is **highlighted.**

PROC VAXTOAXP

PROC VAXTOAXP is a new procedure that converts VAX data to AXP format, converting 2 byte data to 3 bytes. It is only intended for VAX data sets, not catalogs, indexes or other types of SAS files.

This procedure is used on a DEC Alpha to read datasets which were created on a Dec VAX.

Increasing Allocation Sizes

Previous versions of SAS for OpenVMS defaulted to allocating 33 disk blocks to a file when it was created. When the file needed to be extended, it was extended in increments of 32 blocks. With this change, 300 blocks are initially allocated to the file. This potentially reduces the number of extends required for the file. When the file is extended, it is now done so in increments of 96 blocks. The unused disk blocks are deallocated and freed back to the file system at close time. These changes reduce the number of times a file needs to be extended, potentially creates a more contiguous file, and reduces subsequent access time.

Two new logical names are supported. SAS$ALQ and SAS$DEQ. These logical names define the number of disk blocks to be initially allocated for SAS files at creation (SAS$ALQ) and the number of blocks to be allocated when the file needs to be extended (SAS$DEQ). These correspond directly to the ALQ= and DEQ= options which have previously existed and are documented elsewhere. The order of precedence is as follows: 1) if an ALQ/DEQ option is specified for a specific data set that is honored; 2) if ALQ/DEQ is specified on a LIBNAME statement it affects all the SAS files created in that library; 3) if the logicals SASALQ/SASDEQ are defined these values are used; and 4) if none of the above are specified the system default values are used.

Asynchronous Read-Ahead and Write-Behind

There are two new data set options to control asynchronous read-ahead and write-behind for OpenVMS (RAH/WBH). RAH may have a value of "YES", "NO", or "LOG" ("LOG" is the same as "YES", but enables the logging of information to the SAS log). WBH may have values of "YES" or "NO". When these options are enabled asynchronous read-ahead and/or write-behind is in effect.

Asynchronous read-ahead means that requests to read pages from the file are asynchronously queued such that when a page is requested, additional pages may be requested from the operating system asynchronously. When the application actually requests those additional pages, they should already be available.

Likewise, asynchronous write-behind means that when a page is written to disk it is asynchronously queued, and control immediately returns to the application.

When the next page is written, SAS ensures that the previously queued page has completed its write before queuing the current page. In this way, control is returned to the application sooner without having to wait for the I/O operation to complete each time.

This option must be specified for each dataset; unlike some options, it cannot be used in the LIBNAME statement. It cannot be used in a view. An example is:

```
   set dataware.penult (rah=yes)
```

A few tests on large datasets showed no decrease in I/O or CPU time, but a substantial decrease in elapsed time. For example, the time needed to read 7 million records decreased from 15 minutes with RAH=NO to 9 minutes with RAH=YES.

The LOGMULTREAD System Option

A new configuration option LOGMULTREAD allows the user to open the session log file for shared read access. This option can be set on the command line or in a configuration file.

Turning the option on will result in performance degradation since data is written to the log with greater frequency.

LOGMULTREAD is off by default.

You would use this option in a long-running batch job if you wanted to check on its progress.