Paper CS-043

Data Set Options: Beyond DROP, KEEP, RENAME, and WHERE

Sarah Woodruff, Westat, Rockville, MD
Ed Heaton, Westat, Rockville, MD

Abstract

While the `DROP=`, `KEEP=`, `RENAME=()`, and `WHERE=()` data set options are extremely useful and well known, there are numerous other data set options that can greatly enhance computer efficiency and programmer productivity. However, finding them and the documentation can be daunting because not all options are a part of Base SAS® and their descriptions exist in various locations.

This paper will provide a quick reference to the documentation for data set options and then go on to discuss a handful of data set options that deserve special consideration. Some of these are Base SAS data set options and some are provided with SAS/ACCESS® software. Our discussion of the documentation will cover both that which is available through the **Help** menu in the SAS® Display Manager and what is available online via support.sas.com.

Keywords: `label=`, `genMax=`, `genNum=`, `compress=`, `dbLabel=`, `dbSasLabel=`, `dbSasType=()`, `dbType=()`, `sasDateFmt=()`

Introduction

Data set options control the way data comes into our steps and the way data goes out.

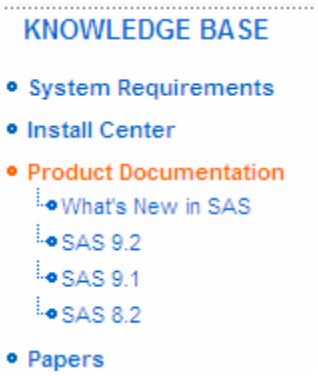
We are all familiar with the big four: `drop=`, `keep=`, `rename=()`, and `where=()`. Did you know that SAS processes these in alphabetical order? That's really all we're going to say about these – for now. First, we are going to look at some data set options that might be less familiar. Our goal is to heighten your awareness of data set options as solutions to your programming tasks. We will do this by presenting a few options that you might have not used before. We will also show the use of some options that you might have employed but in ways you might not have considered.

Data set options are listed in several places in the SAS documentation. We will also attempt to make these easier to find.

We will be discussing documentation in general for each of the data set options. Primary sources for documentation are the **Help** menu in the SAS Display Manager and at **support.sas.com**.

SAS OnlineDoc® Documentation

Currently, you can reach the product documentation at http://support.sas.com under *KNOWLEDGE BASE* in the tree on the left side of the web page.



Click on *Product Documentation* and select SAS® *9.2*, SAS® *9.1*, or SAS® *8.2*.

If you select SAS *9.1*, you will be taken to a web page that allows you to choose between SAS *9.1.3* or either of the earlier versions of SAS 9.1. Choose *SAS OnlineDoc 9.1.3 for the Web* or one of its earlier equivalents if you are looking for something, as *Documentation for SAS 9.1.3 in PDF* is for printing.
Under SAS *8.2*, you can choose *SAS OnlineDoc 8*, or any of a number of technical reports.

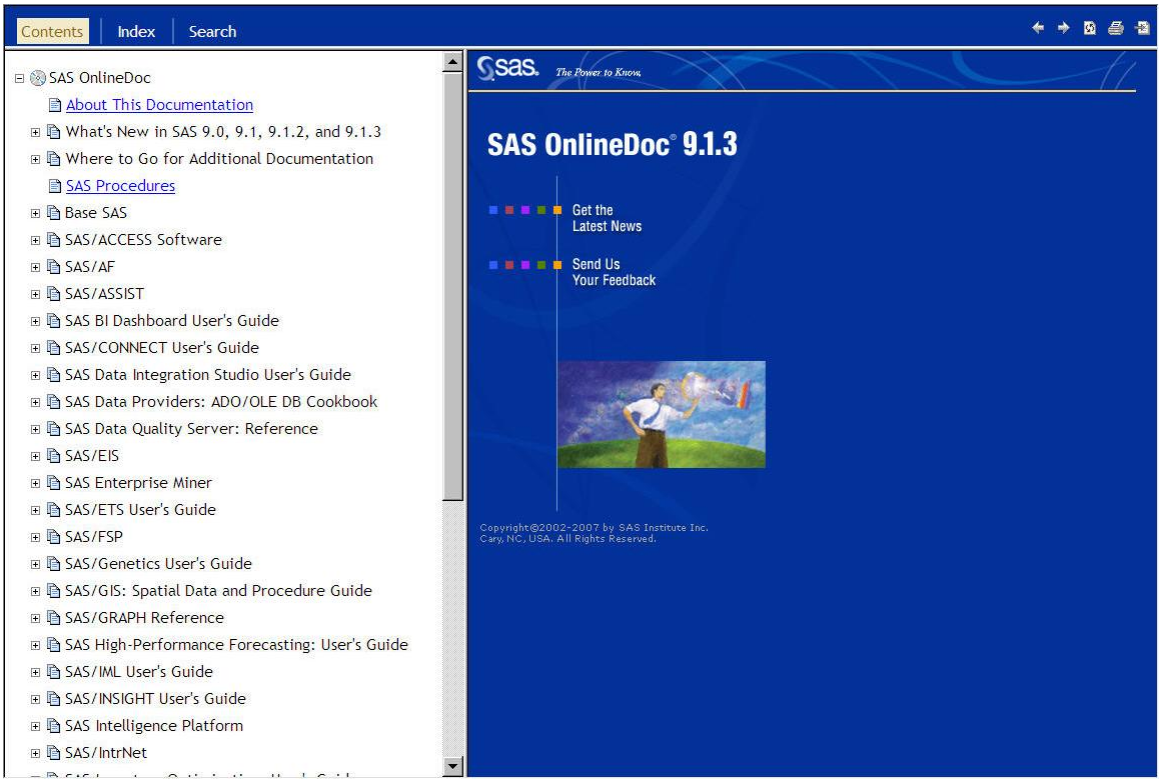
SAS 9.2

Start at http://support.sas.com/cdlsearch?ct=80000 or negotiate there as described above. You might want to create a shortcut to this location, or put it in your **Favorites** menu.

Click on **more..** under **SAS Reference** and you will find **Data Set Options**.

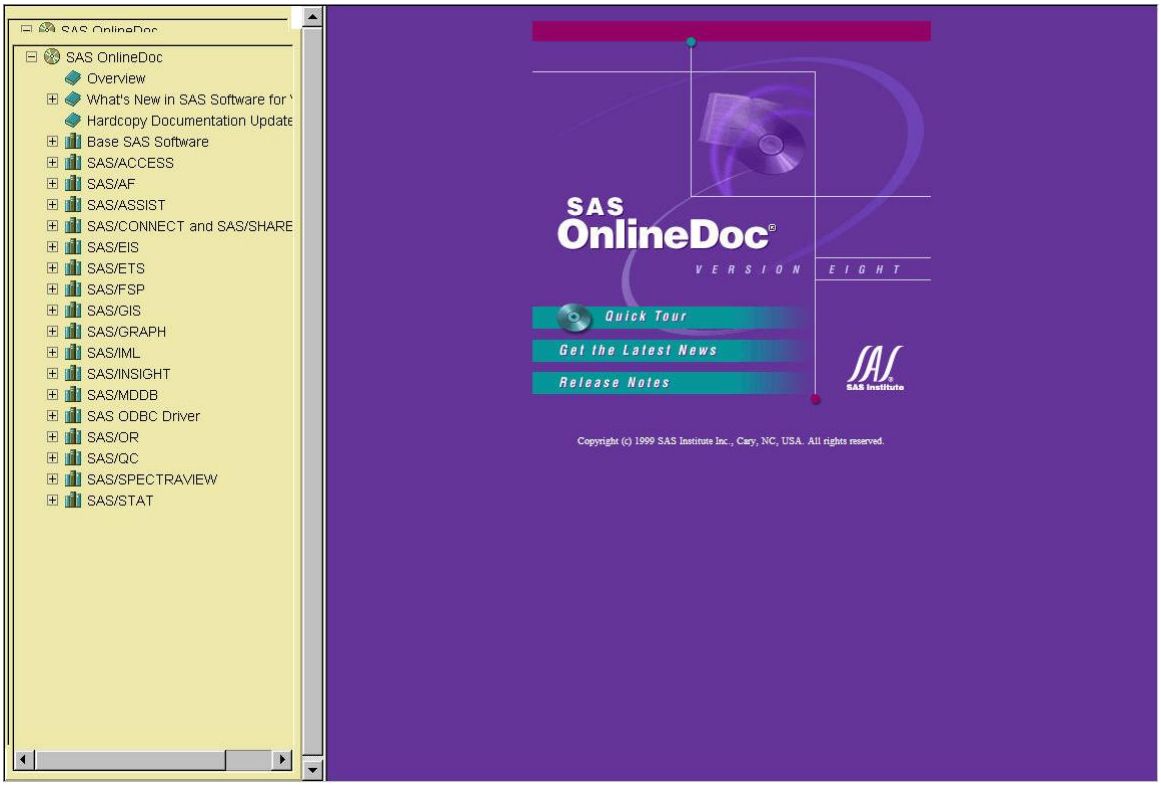
SAS 9.1.3

Either follow the links as described above or go directly to http://support.sas.com/onlinedoc/913/docMainpage.jsp. You might want a shortcut to this page.



SAS 8.2

For SAS 8.2, the main interface looks like this.



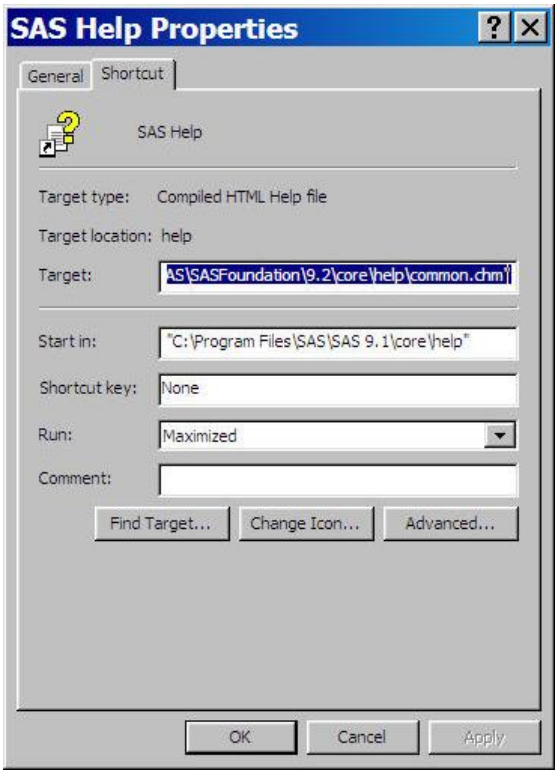
The URL for this location is http://v8doc.sas.com/sashtml.

SAS Help and Documentation

If you have the SAS® Display Manager open, you will probably want to use the documentation that is available under the **Help** menu. You can get there quickly even without SAS open if you create a shortcut to a file called **common.chm**. This is a compiled HTML Help file. You might have to search for it. We found ours under **C:\Program Files\SAS\SASFoundation\9.2\core\help**. When we double-click on this file, we get SAS Documentation.

We found this so handy that we put the shortcut in our tool bar.

Of course, we don't like it coming up so small. So, we changed the properties to run maximized.



You should be able to find **common.chm** if you are running SAS 9.1.3, too. SAS 8.2 has a different structure for their documentation.

For SAS 9.1.3, a general inventory listing of the data set options can be found by starting with *SAS Products*, then clicking on *Base SAS*, then *SAS Language Dictionary*, then *Dictionary of Language Elements* where you will find *SAS Data Set Options*.

Index and Search

Throughout this paper, we give specific direction to the documentation for the data set options we discuss. However, it is possible to find information on them via the Index or Search options available in SAS Help. When using Index, adding the phrase, *data set option*, in the search box will limit the results to the most relevant and provide the most targeted information. Use of Search can often return a multitude of results when only entering the name of the option so try including “data set option”, being sure to utilize the quotes, in order to view the most relevant results.

Data Set Options

label=

Have you ever found or inherited a SAS data set and wondered where it came from or what it's all about? Did it have a data set label?

You probably know of the `label=` data set option. Do you use it?

Data set labels can now be up to 256 characters, so they can contain a lot of information. For permanent data sets, think about recording things like the program that created it, the project, where the data came from, etc.

The data set label is documentation for your SAS data set and can also serve as documentation for your code. Just as `Title` statements can serve the added role of program documentation, so can data set labels. If your data sets are labeled, you can search for key words in the label.

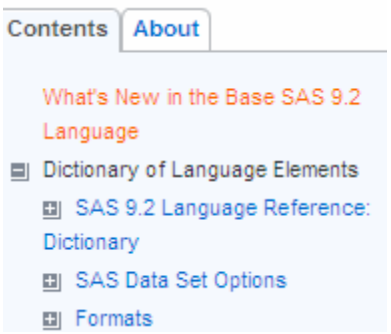
```
LibName sasLib ".\Library\" ;
Proc sql ;
    Select MemName
       from dictionary.tables
        where (
            ( libName eq 'SASLIB' )
          & ( lowCase(memLabel) contains 'special' )
          & ( upCase(memLabel) contains 'SESUG' )
        )
    ;
Quit ;
```

If you want to search several libraries except those supplied by SAS, you can use the following clause.

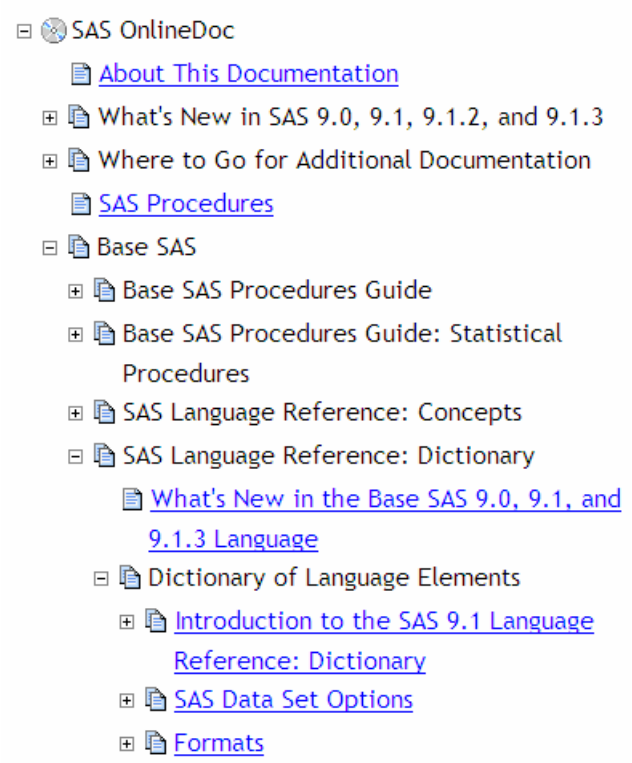
```
( libName not in ('SASHELP','SASUSER','MAPS','WORK') )
```

Documentation

support.sas.com:



SAS 9.2: Start at http://support.sas.com/cdlsearch?ct=80005&qm=3&la=en and select *SAS® 9.2 Language Reference Dictionary*. Click on *Dictionary of Language* elements in the tree on the left side of the window and you will see *SAS Data Set Options*.



SAS 9.1.3: In the tree in the left pane, select *Base SAS*, then *SAS Language Reference: Dictionary*, then *Dictionary of Language Elements*, and then *SAS Data Set Options*. Here you will find *LABEL= Data Set Option*.

SAS 8.2: In the tree in the left pane of the **Help** interface, select *Base SAS Software*, then *SAS Language Reference Dictionary*; then *Dictionary of Language Elements*, and then *Data Set Options*. Here you will find *LABEL=.*

SAS Help:

SAS 9.2: Start with *SAS Products*. Go to *Base SAS*, then *SAS 9.2 Language Reference Dictionary*, then *Dictionary of Language Elements*, and then *Data Set Options*. Here you will find *LABEL= Data Set Option*.

SAS 9.1.3: Start with *SAS Products*. Go to *Base SAS*, then *SAS Language Dictionary*, then *Dictionary of Language Elements*, and then *SAS Data Set Options*. Here you will find *LABEL= Data Set Option*.

SAS 8.2: If you're still running SAS 8.2, you can find the options by slightly modifying the directions for versions of SAS 9.

genMax= and genNum=

Have you ever overwritten a data set with disastrous consequences? Usually, we can recreate these data sets, but it might take considerable work. The `genMax=` data set option can save the day.

The `genMax=` data set option sets up generation data sets. So, when we update a data set, the original is kept with a generation number. The `genMax=` option tells SAS how many old copies to keep.

Suppose we have a data set that has rows in a particular order where the order is important – e.g., a data set with abbreviations and names for the states and territories.

ST	State
AL	Alabama
AK	Alaska
AZ	Arizona
AR	Arkansas
CA	California
CO	Colorado
⋮	⋮
WI	Wisconsin
WY	Wyoming
AS	American Samoa
PQ	Canal Zone
EQ	Canton/Enderbury Is
FM	Fed State Micronesia
⋮	⋮
VI	Virgin Islands
WQ	Wake Island

Now, suppose we need to merge this file and we sort it by **ST**, but accidentally neglect to use the `out=` option to sort into another data set. If we look at our output, we see that *Alaska* comes before *Alabama* and *American Samoa* follows *Arkansas*. If we were to resort by `State`, we would see that *Alaska* now follows *Alabama*, but *American Samoa* comes next. We can't get the data back in the original order without adding some other variable that identifies whether the entry is a state or a territory.

If our data set had the generation feature turned on, we could simply delete the new base version and revert to the previous version.

```
LibName foo ".\" ;
Title1 "Generations" ;
Proc data sets library=foo noList ;
    Modify StateAbbr( genMax=3 ) ;
    Run ;
Quit ;
Title2 "Original Sorted" ;
Proc sort data=foo.StateAbbr ;
    By ST ;
Run ;
Proc print data=foo.StateAbbr( obs=6 ) ;
Run ;
```

We refer to previous versions of our data set with the **genNum=** data set option, which can use either absolute or relative referencing. When we sorted **foo.StateAbbr** above, SAS saved the original as **stateabbr#001.sas7bdat**. The current version is **stateabbr.sas7bdat**. If we modify **foo.StateAbbr** again, SAS will save the data set first as **stateabbr#002.sas7bdat**. If we save it again, SAS will delete **stateabbr#001.sas7bdat** and create **stateabbr#003.sas7bdat**. The numbers will progress to 999 before starting over again at 000.

We can specify a saved data set by its number. We can also specify it with a relative reference. That is, **genNum=-1** is the most recently saved copy and **genNum=-2** is the one that was saved before that.

```
Title2 "Pre-sort Version as genNum=-1" ;
Proc print data=foo.StateAbbr(
    obs=6
    genNum=-1
 ) ;
Run ;
```

In **Proc datasets**, **genNum=revert** (when combined with the **Delete** statement) will revert back to the original data set – if the original still exists.

```
Title2 "Reclaimed Original" ;
Proc datasets library=foo genNum=revert ;
    Delete StateAbbr ;
    Run ;
Quit ;
Proc print data=foo.StateAbbr( obs=6 ) ;
Run ;
```

If the original data set does not exist, you can use a **Data** step to get back to the prior version.

```
Title2 "Revert to Previous Version" ;
Data foo.StateAbbr ;
    Set foo.StateAbbr( genNum=-1 ) ;
Run ;
Proc print data=foo.StateAbbr( obs=6 ) ;
Run ;
```

We're sure you can think of other places where it would be good to have previous copies of your data – all maintained by SAS.

Documentation

The documentation for the **genMax=** and **genNum=** data set options can be found in the *SAS Language Reference Dictionary* under Data Set Options. That's the same place you found the documentation for **label=**.

In the SAS® OnlineDoc, you can find out more about generation data sets in *SAS Language Reference: Concepts*. For SAS 9.2, click on *SAS® 9.2 Language Reference: Concepts*. In the tree on the left, select *SAS Files Concepts*, then *SAS Data Files*, then *Understanding Generation Data sets*.

For SAS 9.1.3, click on *SAS Data Files*, then *SAS Files Concepts*, then *SAS Data Files*, and then *Understanding Generation Data sets*.

The locations are similar for SAS Help.

compress=

Have you ever considered compressing your data sets? SAS provides two compression algorithms: **compress=char** and **compress=binary**.

The **compress=char** data set option tells SAS to compress your data set using the *Run Length Encoding* (RLE) algorithm while **compress=binary** requests the *Ross Data Compression* (RDC) algorithm. RDC compression adds 12 bytes of compression information to each row of your data set – twice that if you are running on a 64-bit host. However, if your data sets are wide, you can save a lot of disk space and a lot of I/O. SAS will attempt to determine if your compression will actually save space, and will not compress the file if it believes that the size of the compressed file will be larger.

```
NOTE: Compression was disabled for data set WORK.BINARY because compression
      overhead would increase the size of the data set.
```

Be warned though that SAS doesn't always guess correctly!

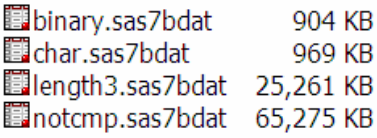
Consider the following data sets.

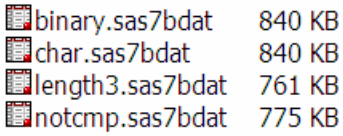
```
Data
    notCmp( keep=r: )
    char( keep=r: compress=char )
    binary( keep=r: compress=binary )
    length3( keep=s: )
;
 /* 10k bytes is 1024 bytes or 8192 bits */
    Length s1-s8192 3 ;
    Retain s1-s8192 r1-r8192 0 ;
    Do i=1 to 1000 ;  Drop i ;
        Output ;
    End ;
    Stop ;
Run ;
```

The log shows…

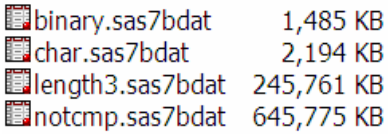
```
NOTE: The data set WORK.NOTCMP has 1000 observations and 8192 variables.
NOTE: The data set WORK.CHAR has 1000 observations and 8192 variables.
NOTE: Compressing data set WORK.CHAR decreased size by 98.52 percent.
      Compressed is 15 pages; un-compressed would require 1011 pages.
NOTE: The data set WORK.BINARY has 1000 observations and 8192 variables.
NOTE: Compressing data set WORK.BINARY decreased size by 98.62 percent.
      Compressed is 14 pages; un-compressed would require 1011 pages.
NOTE: The data set WORK.LENGTH3 has 1000 observations and 8192 variables.
```

Let's compare the actual files in Windows Explorer. RDC (**binary**) compression edged RLE (**char**) compression to win, but both performed magnificently over using three-byte numbers. (See figure at right.)





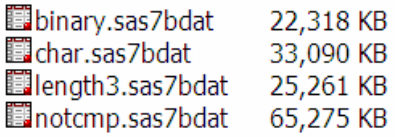
If we move the **Stop** statement before the **Output** statement, we get tables with no data. (See figure on left.) We see that the data set with binary compression uses 840 kilobytes just for the header. Subtract that and the 12 bytes per row of compression information from the 1000-row data set and we see that the 8,192,000 numbers in the data set are stored in 52 kilobytes!



If we increase the output to 10,000 rows, we see that the size of the RDC-compressed data set increases by about two-thirds. The size of the data set with RLE compression more than doubled when the amount of data increased tenfold.

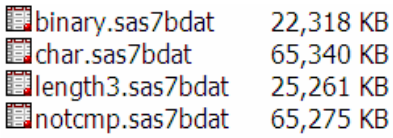
What happens when we store ones in the variables rather than zeros? Let's go back to 1000 rows of output.

```
    Retain s1-s8192 r1-r8192 1 ;
```



Binary compression barely beats-out three-byte numbers. We get identical file sizes if we store twos in the variables and the same is true for missing values.

What would happen if we stored 0.1 in the variables? Suddenly, char compression costs more than it gives back. However, binary compression still gives us smaller files than shortening the length of the variables to three bytes. And, of course, if we store 0.1 in a three-byte numeric variable, our data is compromised.




```
155  Data ;
156      Length x 3 ;
157      x = 0.1 ;
158  Run ;

NOTE: The data set WORK.DATA1 has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

159  Data _null_ ;
160      Set ;
161      If ( x eq 0.1 )
162          then putLog 'NOTE: Your data is okay.' ;
163          Else putLog 'ERROR: Your data has been compromised!' ;
164  Run ;

ERROR: Your data has been compromised!
```

So, if space is a problem, store your Boolean data as zeros and ones and use **compress=binary**.

Compression will increase the workload on your CPU, but modern CPUs are very fast and most of our processes are I/O bound, especially when stored on network drives. This is a particularly worthwhile trade-off if you are using generation data sets.

Documentation

Documentation for the **compress=** data set option is in the *SAS Language Reference: Dictionary*. You can also find information under *SAS Data Files* in *SAS Language Reference: Concepts*.

dbLabel=

Do you ever want your variable labels to head the columns of your Microsoft Excel worksheets when you export your SAS data sets to Excel? Then **dbLabel=** is your ticket.

```
Proc contents data=sasHelp.Shoes varNum ;
Run ;
```

We see that we have some variable labels.

#	Variable	Type	Len	Format	Informat	Label
1	Region	Char	25			
2	Product	Char	14			
3	Subsidiary	Char	12			
4	Stores	Num	8			Number of Stores
5	Sales	Num	8	DOLLAR12.	DOLLAR12.	Total Sales
6	Inventory	Num	8	DOLLAR12.	DOLLAR12.	Total Inventory
7	Returns	Num	8	DOLLAR12.	DOLLAR12.	Total Returns

However, when we write these to Excel, we get only the variable names.

```
LibName xlsLib ".\dbLabel.xls" ;
Data xlsLib.Shoes ;
    Set sasHelp.Shoes ;
Run ;
```

The SAS log tells us that the variable labels were not written to Excel…

```
NOTE: SAS variable labels, formats, and lengths are not written to DBMS
      tables.
```

…and Excel verifies that.

	A	B	C	D	E	F	G
1	Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
2	Africa	Boot	Addis Ababa	12	$29,761.00	$191,821.00	$769.00
3	Africa	Men's Casual	Addis Ababa	4	$67,242.00	$118,036.00	$2,284.00
4	Africa	Men's Dress	Addis Ababa	7	$76,793.00	$136,273.00	$2,433.00
5	Africa	Sandal	Addis Ababa	10	$62,819.00	$204,284.00	$1,861.00

Let's add the **dbLabel=yes** data set option.

```
Proc sql ;  Drop table xlsLib.Shoes ;  Quit ;
Data xlsLib.Shoes( dbLabel=yes ) ;
    Set sasHelp.Shoes ;
Run ;
LibName xlsLib clear ;
```

We get the same note in the SAS log. However, we have the variable labels for column headers in the Excel worksheet.

	A	B	C	D	E	F	G
1	Region	Product	Subsidiary	Number of Stores	Total Sales	Total Inventory	Total Returns
2	Africa	Boot	Addis Ababa	12	$29,761.00	$191,821.00	$769.00
3	Africa	Men's Casual	Addis Ababa	4	$67,242.00	$118,036.00	$2,284.00
4	Africa	Men's Dress	Addis Ababa	7	$76,793.00	$136,273.00	$2,433.00
5	Africa	Sandal	Addis Ababa	10	$62,819.00	$204,284.00	$1,861.00

This works with Microsoft Excel, but what about other databases? Microsoft Access, Microsoft SQL Server, and some others allow very long variable names and strange characters in those names; these do not transfer so easily to SAS. If you want to update one of these tables, you

can simply assign the database field name to the SAS variable label and then use the **dbLabel=yes** data set option to convert the SAS variable names to RDBMS column names. Of course, the name restrictions of the target RDBMS still apply.

Documentation

Documentation for the **dbLabel=** data set option is in the documentation for SAS/ACCESS®.

SAS Help

Click on *SAS/ACCESS 9.2 for PC Files: Reference*. Then click on *LIBNAME Statement and Pass-Through Facility on 32-Bit Microsoft Windows* (or on *Linux, UNIX, and 64-Bit Microsoft Windows*). Then click on *The LIBNAME Statement for PC Files on Microsoft Windows* (or its counterpart). There you will find *DBLABEL= Data Set Option*.

You can also find the option specified under *SAS/ACCESS 9.2 for Relational Databases Reference*. Look under *DBMS-Specific Reference* for *SAS/ACCESS for OLE DB*, *SAS/ACCESS for Oracle*, etc. There you should find *Data Set Options for [your specific database]*

For SAS 9.1.3, click on *SAS Products*, then *SAS/Access*, then *PC Files*, then *Accessing PC Files* and finally *The LIBNAME Statement for PC Files on Windows*. There you will find *DBLABEL=.*

You can also find the option specified under *Relational Databases* within *SAS/Access*. Click on *General References*, then *Data Set Options for Relational Databases* and there you will find *DBLABEL= Data Set Option*.

SAS® OnlineDoc

From the *Product Documentation* page for SAS 9.2, select *Access* under *Data Management*. Then click on *SAS/ACCESS® 9.2 for Relational Databases: Reference* or *SAS/ACCESS® 9.2 Interface to PC Files: Reference*. From there, go to the table of contents and expand *LIBNAME Statement and Pass-Through Facility on 32-Bit Microsoft Windows*. Then click on *The LIBNAME Statement for PC Files on Microsoft Windows*. Here you will find the list of data set options.

For SAS 9.1.3, select *SAS/ACCESS Software*, then *SAS/ACCESS for PC Files: Reference*, then *Accessing PC Files*, and then *The LIBNAME Statement for PC Files on Windows*. For *SAS/ACCESS for Relational Databases: Reference*, look under *DBMS Specific Reference*.

dbSasLabel=

We just saw how we could use **dbLabel=** to write variable labels to field names. SAS has a data set option, **dbSasLabel=**, whose default value of **yes** tells SAS to assign the database field name to the variable label in the SAS data set. This makes the combination of **dbSasLabel=yes** and **dbLabel=yes** very powerful when reading and updating tables from databases other than SAS. Column headers in your RDBMS table that do not conform to SAS convention will arrive in SAS with a SAS-compliant name and the RDBMS name in the variable label. The data can be modified and written back to the RDBMS using the original database's naming convention. Consider this **NobelPeacePrize** table in a Microsoft Access database.

```
LibName mdbLib ".\DatasetOptions.mdb" ;
Proc contents data=mdbLib.NobelPeacePrize varNum ;
Run ;
```

The database has column names with spaces. We see this in the variable labels. However, the variable names were converted to SAS-compliant names.

#	Variable	Type	Len	Format	Informat	Label
1	key	Num	8	11.	11.	Key
2	Award_Year	Num	8	6.	6.	Award Year
3	Prize_Portion	Num	8			Prize Portion
4	Laureate_s_Name	Char	96	$96.	$96.	Laureate's Name
5	Organization	Num	8	2.	2.	Organization
6	Country	Char	64	$64.	$64.	Country
7	Titles_and_Positions	Char	1024	$1024.	$1024.	Titles and Positions
8	Reason_for_the_Award	Char	1024	$1024.	$1024.	Reason for the Award

Suppose we received the 2007 laureates in an Excel file and we need to append these to the MS Access table.

```
LibName xlsLib ".\NobelPrize.2007.xls" ;
Proc contents data=xlsLib.'Peace$'n varnum ;
Run ;
```

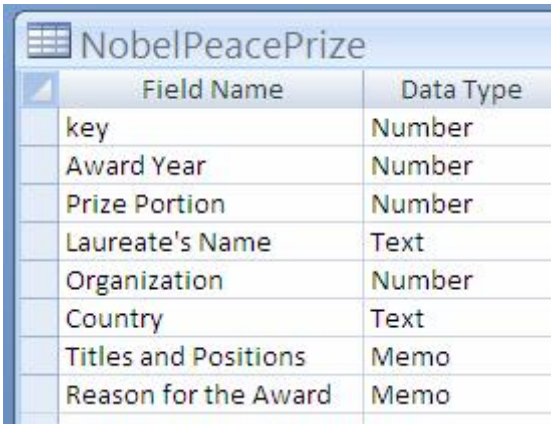
Our output shows the following variable information.

#	Variable	Type	Len	Format	Informat	Label
1	Award_Year	Num	8			Award Year
2	Prize_Portion	Num	8			Prize Portion
3	Reason_for_the_Award	Char	180	$180.	$180.	Reason for the Award
4	Laureate_s_Name	Char	48	$48.	$48.	Laureate's Name
5	Organization	Num	8			Organization
6	Country	Char	26	$26.	$26.	Country
7	Titles_and_Positions	Char	39	$39.	$39.	Titles and Positions

Let's add the 2007 Nobel Laureates from to our Excel file to the MS Access table.

```
Data mdbLib.NobelPeacePrize( dbLabel=yes ) ;
    Set
        mdbLib.NobelPeacePrize_2006
        xlsLib.'Peace$'n( in=fromNewWinners )
    ;
    Retain prevKey ;  Drop prevKey ;
    If fromNewWinners then key = prevKey + 1 ;
    prevKey = key ;
Run ;
```

If we look at our Access database, we see that we have the original names complete with spaces and apostrophes.


NobelPeacePrize

Field Name	Data Type
key	Number
Award Year	Number
Prize Portion	Number
Laureate's Name	Text
Organization	Number
Country	Text
Titles and Positions	Memo
Reason for the Award	Memo

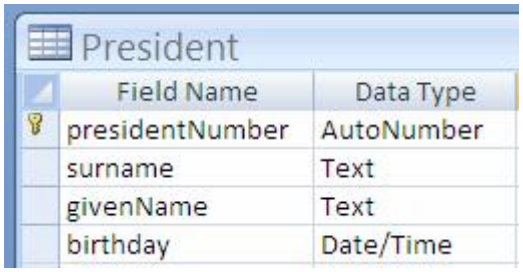
Sometimes, however, you might not want to preserve the database field name in the variable label. For example, suppose you are reading an Excel file with no column headers. The variable names will come to SAS as **F1**, **F2**, **F3**, etc. and the variable labels will be **F1**, **F2**, **F3**, etc. You would have to rename the variable names to something that makes sense, but you can simply avoid these trite variable labels with **dbSasLabel=no**.

Documentation

Documentation for the **dbSasLabel=** data set option is under *SAS/ACCESS*.

dbSasType=()

The **dbSasType=()** data set option specifies how we want our data to appear in SAS when we read it from a external RDBMS table. We can change its data types and character string lengths and convert date-time values to date values. We might need to do this so that our data types are compatible for a merge, or so our dates are compatible. For example, suppose we have a table in our MS Access database with the key (**presidentNumber**) as a number but our SAS table has that key as a 2-digit character string.


President

Field Name	Data Type
presidentNumber	AutoNumber
surname	Text
givenName	Text
birthday	Date/Time

We can change the data type of the key coming to us from the RDBMS to a two-digit character so that it will merge with the key in our SAS table.

```
Data Age ;
    Merge
        mdbLib.President( dbSasType=(
            presidentNumber=char2
            birthday=date
        ) )
        sasLib.Term(
            keep= presidentNumber inaugurationDate
            in=in_Term
        )
    ;
    By presidentNumber ;
    If in_Term ;
    If first.PresidentNumber ;
    age = floor( yrDif(birthday,inaugurationDate,'act') ) ;
Run ;
```

Documentation

Documentation for the **dbSasType=()** data set option is under *SAS/ACCESS*.

dbType=()

Suppose you want to write your SAS table to another RDBMS database (e.g., MS Access) and that table needs to join with another table on a key that's defined as byte. When you write your SAS data sets to an MS Access database, the numbers arrive as double. We need to be able to write this key value as byte.

```
Data mdbLib.toLoad( dbType=( key=byte ) ) ;
    Set toLoad ;
Run ;
```

We can also convert from character to numeric or back with the **dbType=()** option. Suppose **year** is a numeric variable in the SAS data set and we want it to be a character variable on the RDBMS database.

```
Data mdbLib.toLoad( dbType=(
    key=byte
    year='char(4)'
) ) ;
    Set toLoad ;
Run ;
```

Documentation

Documentation for the **dbType=()** data set option is under *SAS/ACCESS*.

sasDateFmt=()

The **sasDateFmt=()** data set option tells the SAS/ACCESS software what format to apply to a date-time variable. Instructions on how to convert the RDBMS date-time value to SAS are assumed to be known. Dates usually come to SAS as date-time values. Though we can convert them on the fly, only formats that have like-named informats can be used.

```
Proc print
    data=mdbLib.President( sasDateFmt=( birthday=yymmdd10. ) )
    noObs
;
Run ;
```

Documentation

Documentation for the **dbSasFmt=()** data set option is under *SAS/ACCESS*.

Fixing date problems

Have you ever looked at your RDBMS dates and found lots of values for January 1, 1960? Are you rather confident that they are wrong?

The problem might be that someone wrote an unformatted date value to a date-time field on your database. Maybe they updated the value with the **today()** function but did not change the format from a date-time format to a date format. If that's the case, you can look at the suspect values and find that the time of day is not 00:00:00 but instead some time of day such as 04:57:04. That's 17,824 seconds after midnight on the morning of January 1, 1960.

That number – 17,824 – also happens to be the number of days between January 1, 1960, and October 19, 2008. So, your data is there, it's just in an improper format. We can reclaim our dates by converting them from date-time values to date values.

First, let's create an update table that has our suspicious dates. We will bring the date values to SAS as date-time values. Then we will compare the number to a range of numbers representing SAS dates – since they might have started off as a representation of a date. If we find any, we will look at the time of day since all SAS date values written to our RDBMS should have the time of day set to midnight.

```
Data updateTable ;
    Set mdbLib.badPresidentBirthdays(
        keep= presidentNumber birthday
        sasDateFmt=( birthday=dateTime. )
    ) ;
    If (
        ( '01JAN1582'd le birthday le today() )
      & timePart(birthday)
    ) ;
    Format birthday date9. ;
Run ;
```

Suppose we looked at these suspicious dates and have determined that they are actually the problem as described. Then we can update out RDBMS table as follows:

We are being redundant by our use of **sasDateFmt=()** again, but we like to be safe.

```
Data mdbLib.badPresidentBirthdays ;
    Modify
       mdbLib.badPresidentBirthdays( sasDateFmt=( birthday=date9. ) )
       updateTable
    ;
    By presidentNumber ;
Run ;
```

You can update a RDBMS table using the **Modify** statement. However, be careful because it modifies in place and errors might not be recoverable.

Macro for Ordered Variable List

We said we were going beyond the **keep**= data set option. However, we want to provide a little macro that you might find useful with the **keep**=option. It's also useful other places, so you might want to put it in your autocall library. This macro will return the ordered list of all of the variables – space delimited – in the input data set. You can use it in a **keep**= or **drop**= data set option or statement.

```
%macro orderedVarNames( data= ) ;
   %local datasetId ;
   %let datasetId = %sysFunc( open( %scan(&data,1,%str(%()) , i ) ) ;
   %local varList ;
   %local i ;
   %do i=1 %to %sysFunc( attrn ( &datasetId , nVars ) ) ;
       %let varList = &varList %sysfunc( varName(&datasetId,&i)) ;
   %end ;
   %let datasetId = %sysFunc( close(&datasetId) ) ;
   &varList
%mEnd orderedVarNames ;
```

We will leave it to you to think of the uses. You might even want to modify this to provide a comma-delimited list of the variable names that might be handy in **Proc sql** or a quoted, comma-delimited list for a hash object's **defineData()** method

Conclusion

One of the goals for any programmer should be to coax the code to do as much of the work for you as possible. Data set options are one of the ways in which to make code both more powerful and more efficient. Consideration of these options often does not go beyond **drop=**, **keep=**, **rename=()**, and **where=()**, so our goal was to explore other options which are quite useful but often underutilized. **Label=** provides a simple method to include detailed documentation, something from which most programs could benefit. **GenMax=** and **genNum=** provide the ability to not only track and recover versions of our data sets, but to specifically access those versions in an easy way. This is a powerful combination in creating more efficient programs – especially when added to the space saving effects of **compress=**. **DbLabel=** and **dbSASLabel=** streamline the conversation between SAS and other databases, thus providing smoother, faster transitions for the programmer and anyone else who needs to handle the data. **DbSASType=** and **dbType=** also contribute to these transitions by allowing for customized matching of data types. **SasDateFmt=** provides consistency for all important date information, even when it was not originally handled by SAS.

In addition to delving further into the power of data set options, we included the final macro to show how even a “work horse” option like **keep=** has more to offer. It is noteworthy that the strength of many of these options really comes through when they are used in pairs, thus further demonstrating the depth of possibility inherent in manipulating the data set.

Reference

Borowiak, K.W. Using Data Set Options in PROC SQL. Paper 131-31. *Proceedings of the 31st Annual SAS Users Group International (SUGI) Conference.* 2006. http://www2.sas.com/proceedings/forum2007/135-2007.pdf.

"CHANGE Statement". The DATASETS Procedure. SAS Institute Inc. 2006. http://support.sas.com/onlinedoc/913/getDoc/en/proc.hlp/a000247645.htm.

Choate, P.A., & Martell, C.A. De-Mystifying the SAS® Libname Engine in Microsoft Excel: A Practical Guide. Paper 024-31. *Proceedings of the 31st Annual SAS Users Group International (SUGI) Conference.* 2006. http://www2.sas.com/proceedings/sugi31/024-31.pdf.

Compson, M. & Hennessey, J.C. HOW SUITE IT IS – Taking Full Advantage of SAS® Seamless Integration with Microsoft's Office Suite. Paper 004-2007. *Proceedings of the 1st Annual SAS Global Forum.* 2007. http://www2.sas.com/proceedings/forum2007/004-2007.pdf.

"Data Set Options for Microsoft SQL Server". SAS/ACCESS for Microsoft SQL Server. 2007. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/acreldb.hlp/a001405613.htm.

"Data Set Options for Oracle". SAS/ACCESS for Oracle. 2007. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/acreldb.hlp/a003113594.htm.

"Data Set Options for PC Files on Windows". The LIBNAME Statement for PC Files on Windows. 2003. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/acpcref.hlp/a002143650.htm.

"DBLABEL=". The LIBNAME Statement for PC Files on Windows. 2003. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/acpcref.hlp/a002261292.htm.

"DBTYPE=". The LIBNAME Statement for PC Files on Windows. 2003. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/acpcref.hlp/a002149863.htm.

"DELETE Statement". The DATASETS Procedure. 2006. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/proc.hlp/a000247666.htm.

"Example 4. DTS: Transferring Generations of SAS Data Sets". Examples of Data Transfer Services (DTS). 2003. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/connref.hlp/a000585273.htm.

"GENMAX= Data Set Option". SAS Data Set Options. 2007. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000243174.htm.

"GENNUM= Data Set Option". SAS Data Sets Options. 2007. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000243056.htm.

He, J., Jain, D., & Wang, C. Make Your SAS/Access® Query More Efficient. Paper 44-28. *Proceedings of the 28th Annual SAS Users Group International (SUGI) Conference.* 2003. http://www2.sas.com/proceedings/sugi28/044-28.pdf.

Karp, A.H. Indexing and Compressing SAS Data Sets: How, Why, and Why Not. Paper 5-25. *Proceedings of the 25th Annual SAS Users Group International (SUGI) Conference.* 2000. http://www2.sas.com/proceedings/sugi25/25/aa/25p005.pdf.

"LABEL= Data Set Option". SAS Data Set Options. 2007. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000131149.htm.

Lafler, K.P. Exploring SAS® Generation Data Sets. Paper 253-31. *Proceedings of the 31st Annual SAS Users Group International (SUGI) Conference.* 2006. http://www2.sas.com/proceedings/sugi31/253-31.pdf.

"SASDATEFMT= Data Set Option". Data Set Options for Relational Databases. 2003. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/acreldb.hlp/a001371624.htm.

"Syntax: DATASETS Procedure". The DATASETS Procedure. 2006. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/proc.hlp/a000393161.htm.

"Syntax for the PROC DOWNLOAD Statement". The Download Procedure. 2003. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/connref.hlp/prcd.htm.

"Syntax for the PROC UPLOAD Statement". The Upload Procedure. 2003. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/connref.hlp/a002558373.htm.

"Understanding an Audit Trail". SAS Data Files. 2005. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/lrcon.hlp/a001224397.htm.

"Understanding Generation Data Sets". SAS Data Files. 2005. SAS Institute Inc. http://support.sas.com/onlinedoc/913/getDoc/en/lrcon.hlp/a000934566.htm.

Wilson, S.A. Why SAS® is the Best Place to Put Your Clinical Data. Paper 112-29. *Proceedings of the 29th Annual SAS Users Group International (SUGI) Conference.* 2004. http://www2.sas.com/proceedings/sugi29/112-29.pdf.

Contact Information

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

Sarah Woodruff
Westat
1650 Research Boulevard
Rockville, MD 20850
Voice: (240) 314-7562
Email: SarahWoodruff@Westat.com

Ed Heaton
Westat
1685 Research Boulevard
Rockville, MD 20850
Voice: (301) 610-4818
Email: EdHeaton@Westat.com

The content of this paper is the work of the authors and does not necessarily represent the opinions, recommendations, or practices of Westat.