Literally Minded: the Use of SAS Literals and Why They Are Still Relevant!
Stanley Fogleman, Harvard Clinical Research Institute, Boston, MA

ABSTRACT
Many people are unaware that there are SAS literals that can be used to store dates, hexadecimals and even SAS variable names directly instead of going through time-consuming and error-prone conversion steps.

INTRODUCTION
SAS literals are a shorthand notation for special data types within SAS. In order to fully appreciate them, it is useful to see what kind of programming gymnastics would be necessary to accomplish the same thing were they to be suddenly removed from the language. SAS literals are important because they free the programmer from having to know the internal representation of SAS dates, times and datetimes. In addition, the range of formats that will be accepted as legitimate SAS dates and times is fairly wide, so again, there is less syntax to memorize (and possibly forget!)

OVERVIEW OF LITERALS
Literals fall into several broad classes: Name, Binary, Hexadecimal, Date (includes Date, Time and Datetime). Name literals are most useful when accessing databases (and Excel spreadsheets) that may have non-standard column names. (For example, they might contain a space or an ampersand.) Literals, like formats use a "look-up" scheme to convert an external value. For example, '01JAN2005'd is converted to an internal representation of 16437 just by virtue of your having added a D after the second single quote. Such a deal!

Consider the difference between these two data steps:

```sas
15   data step1;
16   testdate = '01JAN2005';
17   put 'step1:' testdate=
18   run;

step1:testdate=01JAN2005
NOTE: The data set WORK.STEP1 has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
   real time           0.01 seconds
   cpu time            0.01 seconds

19
20   data step2;
21   testdate = '01JAN2005'd;
22   put 'step2:' testdate=
23   run;

step2:testdate=16437
NOTE: The data set WORK.STEP2 has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
   real time           0.01 seconds
   cpu time            0.01 seconds
```

In the first step, we simply used a pair of single quotes to indicate that the string 01JAN2005 is a constant. Not very useful. In the second step we converted it to a SASDATE, which we can use to do date arithmetic, display in different formats, etc. Similarly, times and datetimes (dates paired with time) are also converted to a specific integer which has the same date arithmetic and formatting capabilities.
IMPORTING FROM EXCEL

In Version 8.2, when one used the import wizard in the Enhanced Editor to convert an Excel spreadsheet into a SAS dataset, column headings such as “Total Monthly Sales” would be converted into a variable called (surprise) TOTAL_MONTHLY_SALES. Using Enterprise Guide 3.0 and SAS 9.1.3 and the Import Spreadsheet tool, one finds that a variable called “Total Monthly Sales” has been created. Although one can admire SAS for its fidelity to the original variable name, its impossible to access this variable by conventional means.

Enter the SAS name literal.

```sas
data one;
set two;
if 'Total Monthly Sales'n > 0 then do;
```

Although it is still very unwieldy, it is now possible to perform any operation that one could on the previously named TOTAL_MONTHLY_SALES. Important Note: You can use a name literal only for variables, statement labels, and DBMS column and table names. You cannot use the name literal with a SAS data set name. If one were to read in a dataset containing spaces or special characters, you can use the NLITERAL function to convert the text string into a SAS name literal. Special consideration should be given when using data step names or first and last notation.

NOTE: code for illustration purposes only.

```sas
options validvarname=any;
data work.'test_data'n;
input string $50.;
newvar = nliteral(string);
datalines;
Mad Doggy
Gilsen & Fight Night
Fehderal Express
Try finding a Carpenter on a Friday afternoon
;Proc sort data=work.'test_data'n out=sorted;
By newvar;
Run;
Data laststep;
Set sorted;
Proc print;
Title 'Strings converted to N-literals';
run;

Results in the output:

Strings converted to N-literals

<table>
<thead>
<tr>
<th></th>
<th>17:52 Friday, July 1, 2005</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>n</td>
</tr>
<tr>
<td>t</td>
<td>e</td>
</tr>
<tr>
<td>r</td>
<td>w</td>
</tr>
<tr>
<td>o</td>
<td>v</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>n</td>
<td>r</td>
</tr>
<tr>
<td>1</td>
<td>Fehderal Express</td>
</tr>
<tr>
<td>2</td>
<td>Mad Doggy</td>
</tr>
<tr>
<td>3</td>
<td>Try finding a Carpenter on a Friday afternoon</td>
</tr>
<tr>
<td>4</td>
<td>Gilsen &amp; Fight Night</td>
</tr>
</tbody>
</table>

"Fehderal Express"N
"Mad Doggy"N
"Try finding a Carpenter on a Friday afternoon"N
"Gilsen & Fight Night"N

Coders' Corner
BIT TESTING (YOU WANT TO DO WHAT?!?)

Although I have never had occasion to use this feature, a bit test compares the binary form of a value against a bit mask. What that means is that a particular variable is converted into a signed integer and compares it against the mask (value in quotes followed by a small b) that you specify. The two values are compared left to right. It produces a true result (a value of one) if all the digits match, and a false result if any digits do not match.

In some cases, it would produce more readable code if a conventional SAS function was used. For example, instead of testing for a negative number by:

```sas
data one;
  if var1 = '1....... ........ ........ ........'b then put 'yep, this is a negative number';
  if sign(var1) = -1 then 'this is a negative number as well'; /* another way */
```

Note that bit masks cannot be used in assignment statements. The following code is illegal:

```sas
x='0101'b; /* error */
```

Also, one cannot use a bit test in a WHERE clause.

```
if x = '1.0'b;
  but not
    where ( x = '1.0'b );
```

If you need to convert character and numeric values to their binary values $BINARYw. And BINARYw. Formats and $BINARYw., BINARYw.d and BITSw.d informats can be useful for this purpose.

A very simple and elegant example of testing for even or odd values is provided by Paul Dorfmann in a post to SAS-L:

```sas
data _null_;  
do n=1357, 2468;  
  even = n eq '0'b; odd = n eq '1'b;  
  put even= odd=;  
end;  
run; 
```

```
even=0 odd=1
even=1 odd=0
```

DATE CONSTANTS

I think it is safe to assume that most SAS programmers know about the date constant, so I will spend precious little time on it. According to the always helpful SAS syntax manual, the general form for a SAS date inside single (or double) quotes is ‘ddmmm<yy>yy’D. Use if the century is strongly recommended to avoid a misinterpretation, and to avoid the dreaded YEARCUTOFF option. It is important to know that the value 1 represents Jan 1, 1960, and that the highest possible date value2 is 31DEC2000 by which time version 10 should be shipping. The oldest (farthest in the past?) date that SAS accepts in October 15, 1582 (thanks to the premiere of the Gregorian calendar). For many reasons, including ease of manipulation, printing, etc, I recommend getting any date you plan to use into SAS date format as quickly as possible.

TIME CONSTANTS

This is much more straightforward since a SAS time represents the number of seconds since midnight, so the value can be between 0 and 86400 internally. Of course, you have no need as a programmer to commit the 86400 to memory. The only thing you would typically need to know is that a SAS time is formed by the general expression inside single or double quotes is ‘hh:mm<ss.s>T.

DATETIME CONSTANTS
Holds date and time together – represents number of seconds in millions since midnight Jan 1, 1960 – general form is ‘dmmm<yy>yy:hh:mm<:ss,s>’dt. Especially useful for comparison with dates stored on an ORACLE database or and EXCEL spreadsheet, since this is their native default format.

HEXADECIMAL CONSTANTS

As with bit-testing functions, they are not often used, but can be very useful on odd occasions, such as looking for a line-feed, carriage control combination in an ascii file. The only limitation is that hex character constants have to be an even number of digits, so you have to append a 0 to the code value if there are an odd number of digits. If you are reading an external file, you can use the $HEX informat and format to create a value for comparison (again, use the LENGTH statement for odd lengths to avoid truncation).

CONCLUSIONS

SAS literals are an invaluable addition to the basic toolbox of any programmer. They allow a programmer to quickly and intuitive convert things like dates and times into an internal format to use for comparison and reporting purposes. Also, when you are importing a spreadsheet from an outside source, you frequently have no control over the column headings that are provided. In that instance, SAS name literals are a convenient way to refer to variables that are created by the import tool in Enterprise Guide.

REFERENCES:


ACKNOWLEDGMENTS

SAS is a Registered Trademark of the SAS Institute, Inc. of Cary, North Carolina.
Ian Whitlock for his lifetime contributions to all who are interested in learning about and using SAS.

CONTACT INFORMATION

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

Stanley Fogleman
Harvard Clinical Research Institute
930 Commonwealth Ave
Boston MA 02215
Email: stanley.fogleman@hcri.harvard.edu
Web: www.hcri.harvard.edu

**********************************************************************