

The %Tellme Macro

Russ Lavery Contractor, Ardmore, PA USA

ABSTRACT

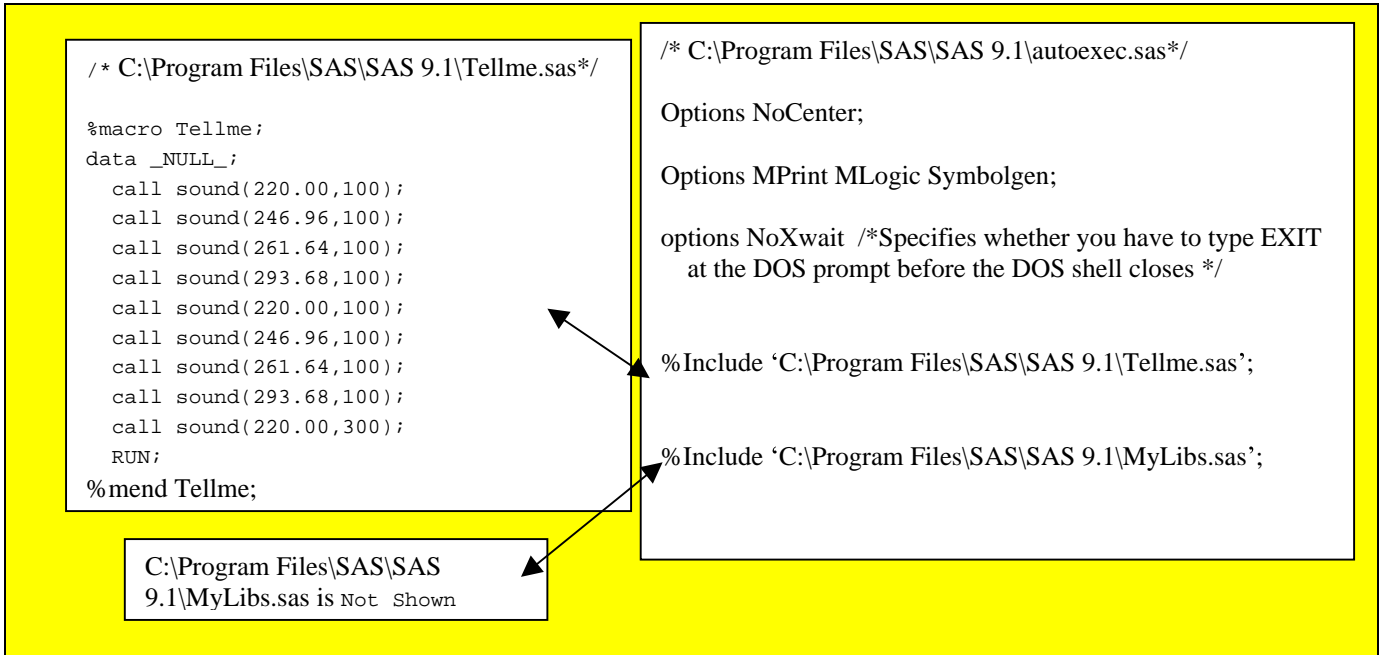
SAS programs, and even sections of SAS programs, often run for a long time. Rather than do nothing, programmers often try to do non-programming work until submitted code has finished –but knowing when the job is finished requires repeated (and annoying) checking of the job status. It is convenient to have SAS notify the programmer when an execution has finished. The macro in this paper will cause SAS to make a noise when the macro executes, thereby notifying the programmer to shift his/her attention back to the SAS program.

INSTALLING THE MACRO TELLME SO IT IS AVAILABLE FOR USE

In order to have this macro always available, I have placed it in a %include in my autoexec.sas.

SAS says that AUTOEXEC.SAS is a file containing SAS statements (statements that you would type into the Editor) that are executed automatically when SAS is invoked. Statements in the SAS autoexec file are executed immediately after SAS initializes and before any user input is accepted. These SAS statements can be used to invoke SAS programs automatically, set up certain variables for use during your SAS session, or set system options.

Unlike the configuration file, a SAS autoexec file is not required in order to run SAS and your system might not have an autoexec.sas to modify. If this is the case you must create one. It is easy to create an autoexec.sas using the SAS Editor. While using system options allows you to place the autoexec file pretty near anywhere you want, SAS will check the directory that contains SAS.exe to see if an autoexec.sas file is in that directory and run it if it is there. That makes (on my system) C:\Program Files\SAS\SAS 9.1\ a great place to put the autoexec.sas file and the file tellme.sas. You can use the Windows search to find SAS.exe on your system.



The macro could have gone directly in the autoexec.sas, but including it makes for a cleaner program follows the pattern of including a set of standard libraries.

USING THE MACRO %TELLME

Here are some examples of how to use the macro. The macro must be run locally, but can be used to tell the programmer when either local or remote jobs are finished.

RUNNING LOCALLY	RUNNING REMOTELY
<pre>Proc Print data = sashelp.class; run; data _null_; x=sleep(5); run; Proc Uinvairiate data=sashelp.class; run; %tellme</pre>	<pre>Rsubmit; Proc Print data = sashelp.class; run; data _null_; x=sleep(5); run; Proc Uinvairiate data=sashelp.class; run; endrssubmit; %tellme</pre>

CONCLUSIONS

This macro is very useful for programmers who have long run times. For those with musical talent, check out the code in the appendix. It is sure to annoy people in neighboring cubes.

REFERENCES:

Fielding, David; Making Music in SAS: Using Sound to Alert Users of Errors and Data Discrepancies

The Proceeding of the 29th Annual Sas User Group International; paper 048 in Coders corner

<http://www2.sas.com/proceedings/sugi29/048-29.pdf>

Whoever wrote the code in the appendix.

CONTACT INFORMATION [HEADER LEVEL 1]

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

Russ Lavery – independent contractor

russ.lavery@verizon.net

-----Appendix:-----

http://www.sas.com/offices/europe/uk/newsletter/feature/25feb_mar07/sound.html contains

Name that Tune using the SAS CALL SOUND Routine!

Here's some example code that illustrates how the datastep and the call sound routine can be used to generate a song!

```
/*Define Length of 4/4 Note*/
%let BaseLength=2000;

data Noten;
  length Note $5. Mode $1. Type $4.;
  retain BaseLength &BaseLength
         Staccato_Lag 0.2;
```

```

BaseFreq=440;
label Note='Note'
    Type='Type of Note, e.g. 1/4'
    Mode='Mode (Staccato/Legato)'
    Staccato_Lag='Short break before note to
                perform staccato'
    BaseFreq='Frequency of base-note A'
    Length='Length of Note'
    BaseLength='Length of Time ';
/*Define Array containing Scale*/
array Scale(0:11) $3 _temporary_
    ('A' 'AIS' 'H' 'C' 'CIS' 'D' 'DIS' 'E' 'F'
     'FIS' 'G' 'GIS');
/*Get file defined in macro-var file*/
input Note Type Mode;
/*Get the length of a Note in ms from input
   Character Column 'Type' in format x/y*/
if index(Type,'/') then
    length=Baselength*
        input(scan(Type,1,'/'),8.)/input(scan(Type,2,'/'),8.);
else length=Baselength*input(scan(Type,1,'/'),8.);
Note=upcase(Note);
select(Note);
    when('DES') Note='CIS';
    when('ES')  Note='DIS';
    when('GES') Note='FIS';
    when('AS')  Note='GIS';
    when('B')   Note='AIS';
    otherwise;
end;
/*If Note contains a plus or minus sign the tone should be x octave
higher or lower according to the integer x following
the sign. This is performed by multiplying the
base Frequency BaseFreq by 2**(sign*x) */
AnyPlusOrMinus=indexc(Note,'-','+');
if AnyPlusOrMinus then do;
    higherOrLower=input(substr(Note,AnyPlusOrMinus+1),1.);
    if substr(Note,AnyPlusOrMinus,1)='- ' then
        higherOrLower=-1*higherOrLower;
    BaseFreq=BaseFreq*(2**(higherOrLower));
    Note=scan(Note,1,'+-');
end;
/*If PAUSE then perform break of length LENGTH*/
if Note='PAUSE' then do;
    /*Delay for Length*/
    start=time();
    end=start+Length/1000;
    do while(time() < end);
end;
drop start end;
end;
/*Now create the Frequency (if not PAUSE)
and sound. Frequency increases by factor
2**(1/12) for each halftone*/
else do i=0 to 11;
    if Note= Scale[i] then do;
        Frequency=BaseFreq*(2**(i/12));
        if mode='S' then do;
            /*if Staccato then Delay for Staccato_lag
            before creating sound*/
            start=time();
            end=start+(Length*Staccato_Lag/1000);
            do while(time() < end);

```

```

        end;
        drop start end;
        /*Correct Length by Staccato_lag*/
        Length=Length-Length*Staccato_lag;
    end;
    call sound(frequency,Length);
    /*Finish do-loop*/
    i=12;
end;
end;
cards;
h      1/16   L
a      1/16   L
gis-1  1/16   L
a      1/16   L
c      1/8    L
PAUSE  1/8    L
d      1/16   L
c      1/16   L
h      1/16   L
c      1/16   L
e      1/8    L
PAUSE  1/8    L
f      1/16   L
e      1/16   L
dis    1/16   L
e      1/16   L
h+1    1/16   L
a+1    1/16   L
gis    1/16   L
a+1    1/16   L
h+1    1/16   L
a+1    1/16   L
gis    1/16   L
a+1    1/16   L
c+1    1/4    L
a+1    1/8    S
c+1    1/8    S
g      1/24   L
a+1    1/24   L
h+1    1/24   L
a+1    1/8    S
g      1/8    S
a+1    1/8    S
g      1/24   L
a+1    1/24   L
h+1    1/24   L
a+1    1/8    S
g      1/8    S
a+1    1/8    S
h+1    1/8    S
a+1    1/8    S
g      1/8    S
fis    1/8    S
e      1/8    S
PAUSE  1/8    L
h      1/16   L
a      1/16   L
gis-1  1/16   L
a      1/16   L
c      1/8    L
PAUSE  1/8    L
d      1/16   L

```

```
c      1/16  L
h      1/16  L
c      1/16  L
e      1/8   L
PAUSE 1/8   L
f      1/16  L
e      1/16  L
dis   1/16  L
e      1/16  L
h+1   1/16  L
a+1   1/16  L
gis   1/16  L
a+1   1/16  L
h+1   1/16  L
a+1   1/16  L
gis   1/16  L
a+1   1/16  L
c+1   1/4   L
a+1   1/8   S
h+1   1/8   S
C+1   1/8   S
h+1   1/8   S
a+1   1/8   S
gis   1/8   S
a+1   1/8   S
e      1/8   S
f      1/8   S
d      1/8   S
c      1/4   L
c      1/16  L
h      1/16  L
a      1/16  L
h      1/16  L
a      1/8   S
PAUSE 1/8   L
;
run;
```