ABSTRACT

The macro facility in the SAS System gives you extra functionality, more programming tools and a high level language for controlling Data Steps and Proc Steps. With the macro facility, you can pass values from step to step in normal SAS code, you can create bundles of SAS code to automate your work, and you can set up mechanisms for conditionally executing your SAS code. This paper introduces the macro facility and provides some basic SAS macro examples.

INTRODUCTION

Within a SAS program, all the SAS Data Steps and Proc Steps are compiled and executed separately. This structure allows great flexibility but imposes some restrictions on the programmer:

- How do we pass values form step to step?
- How do we use a value that might change from day to day, for example the date?
- How do we execute a Proc Step that depends on the value found in a previous Data Step?
- How do we execute a Data Step or Proc Step a number of times, dependant on the value of a variable?
- How can we construct code that depends on the value of a variable?

The macro facility provides the answer to all these problems. Using the macro facility can also make your programs more generic and flexible, making the code you write more usable and powerful.

THE SAS MACRO FACILITY

The SAS macro facility consists of 3 areas:

1. The ability to create “macro variables”, and use automatic “macro variables” (those that exist already)
2. The ability to create “macro code bundles”
3. The ability to use the “macro programming language”

The macro facility has 2 “triggers”:

1. The & sign followed by a non-blank character
2. The % sign followed by a non-blank character

When a section of SAS code is processed, the occurrence of a macro “trigger” causes the SAS supervisor to pass the section to the macro facility which in turn attempts to “resolve” it. Resolving a macro reference means finding the value it takes and giving that value back to the supervisor.

CREATING AND USING MACRO VARIABLES

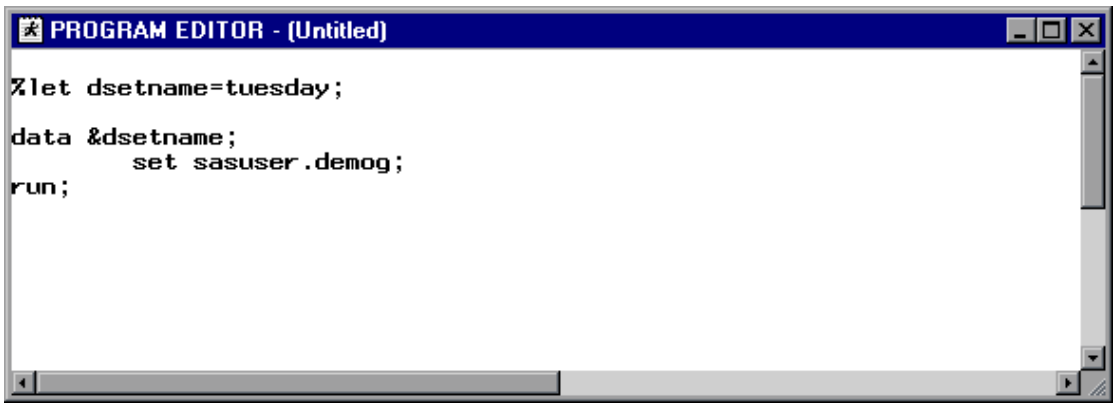
One way to create a macro variable is to use a %LET statement:

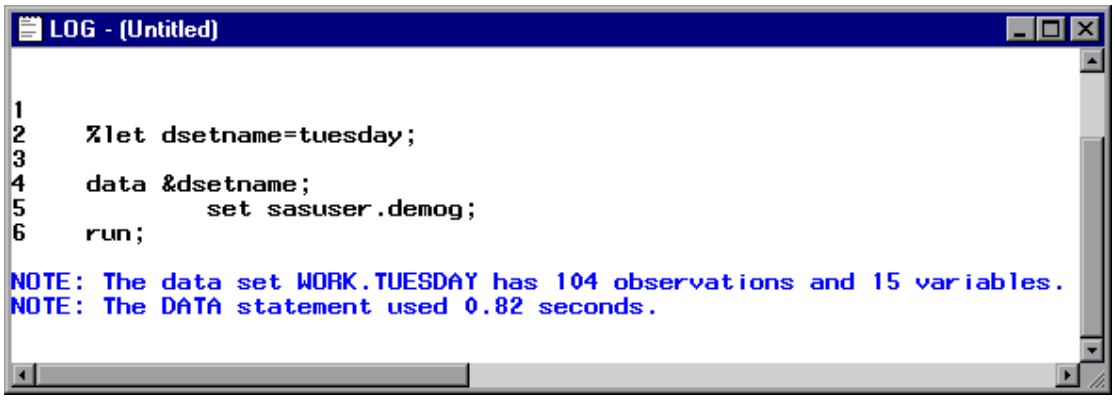
```
%let mvar=cvalue;
```

We have created a macro variable called mvar with a value of cvalue.

- There are no quotation marks around the value
- The macro variable is created as mvar but referred to as &mvar
- leading and trailing blanks are ignored.

Let’s create and use a macro variable in some SAS code:



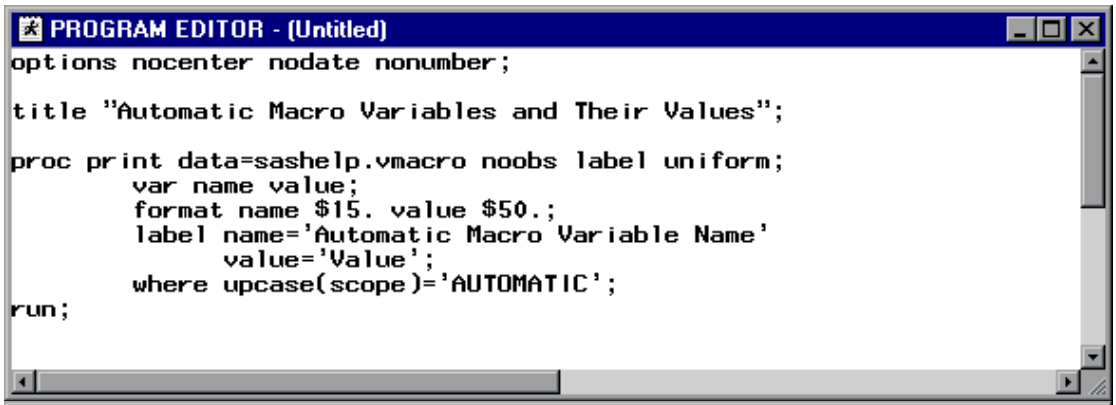


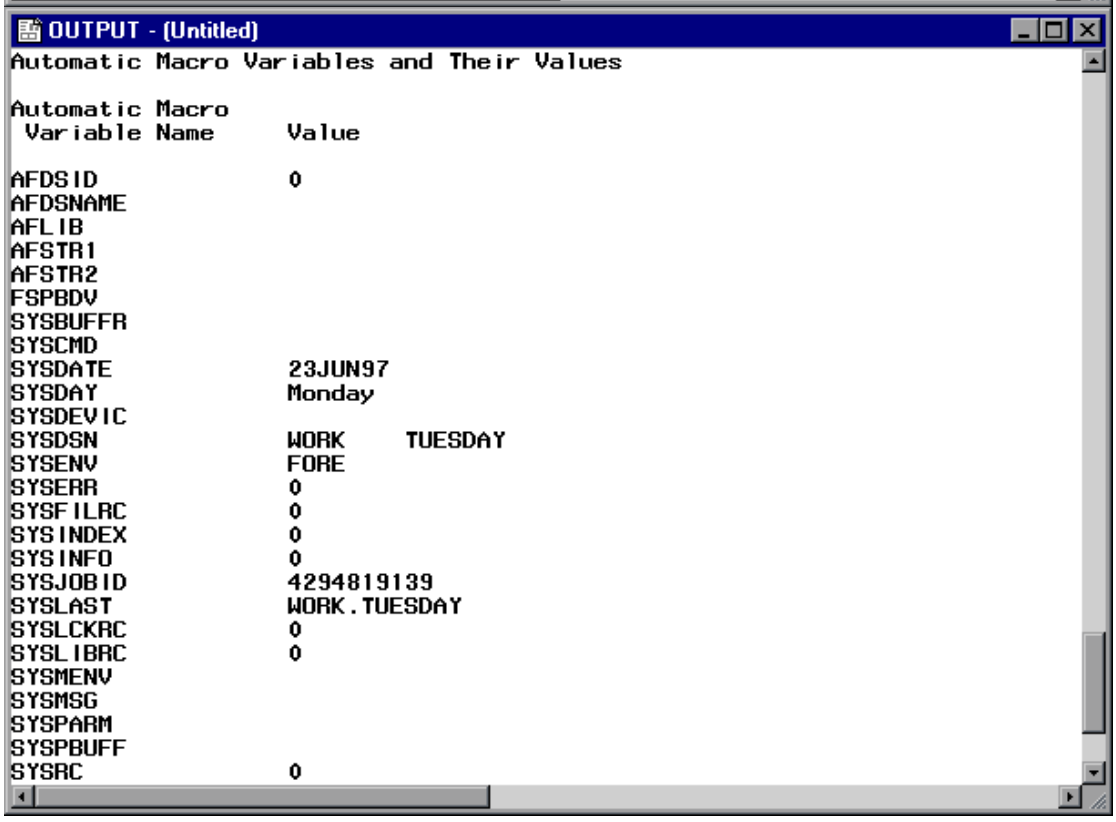
Notice the SAS code in the Log does not show the real name of the dataset.

Automatic Macro Variables

These are macro variables that are created by the SAS System at start-up.

If you are using a later release of the SAS System, for example, Release 6.11 or Release 6.12, you can use the following program to examine the automatic macro variables and their values:



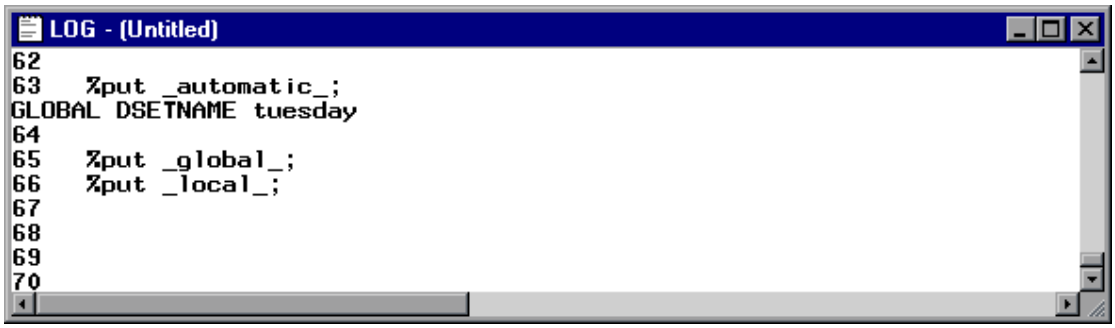


Alternatively, you can use the following %PUT statements to report on:

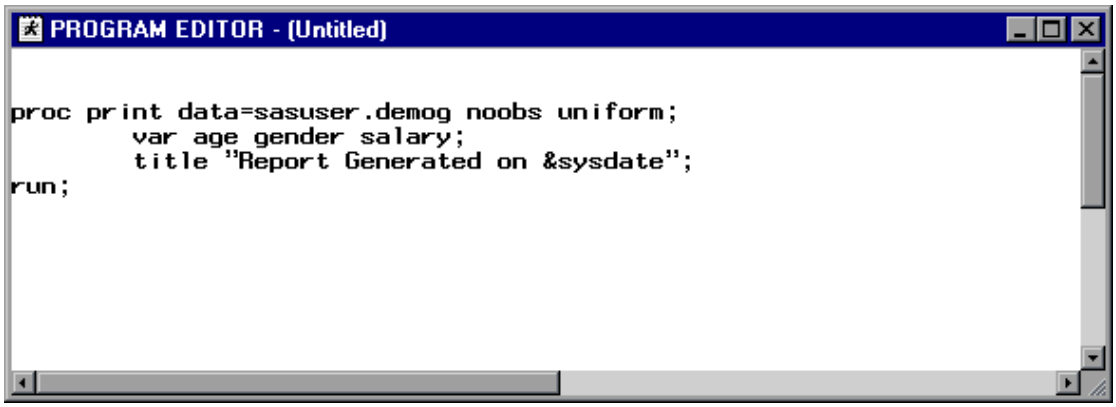
1. The automatic macro variables
2. Those created by the programmer outside a macro code bundle
3. Those created by the programmer inside a macro code bundle

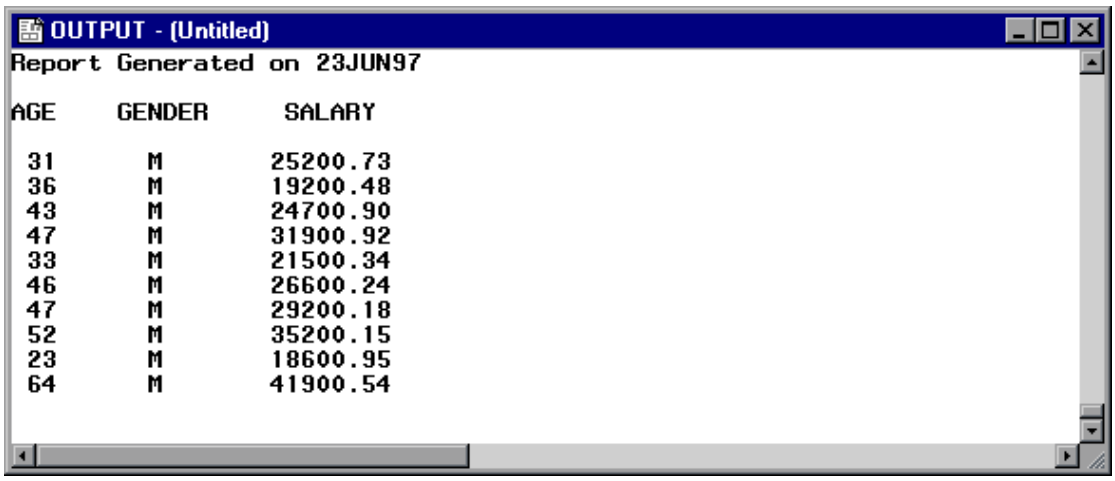
The values are placed in the Log, appearing before the relevant statement:

The %put _automatic_ statement creates similar output to that on the previous page and is not shown here.

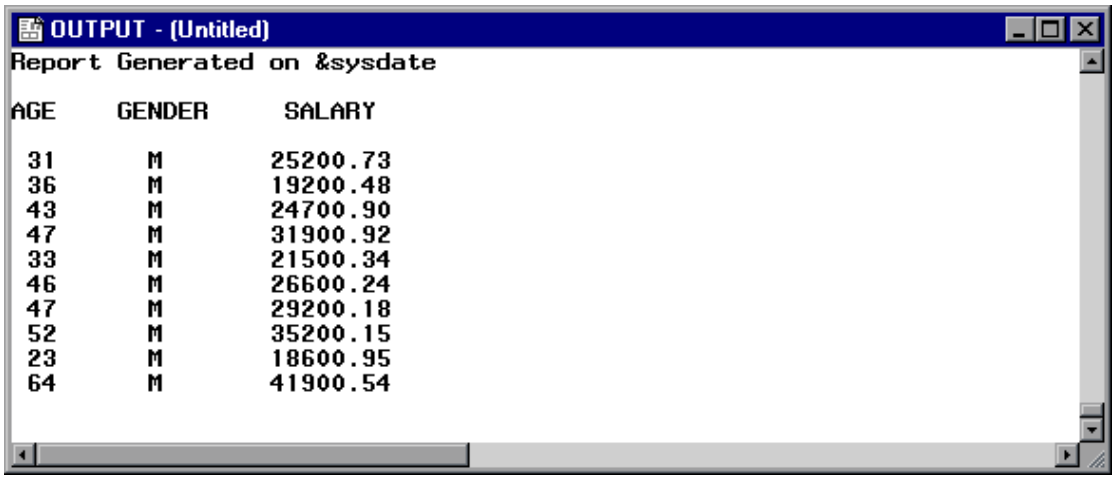


We could use the automatic macro variable &sysdate in a SAS program as follows:



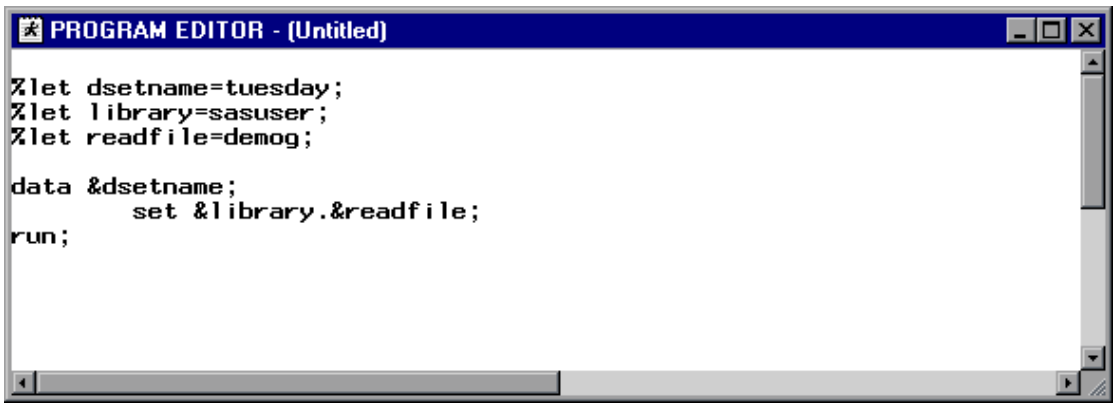


Notice the use of the double quotation marks around the title text.
The following report shows the effect of using single quotation marks:

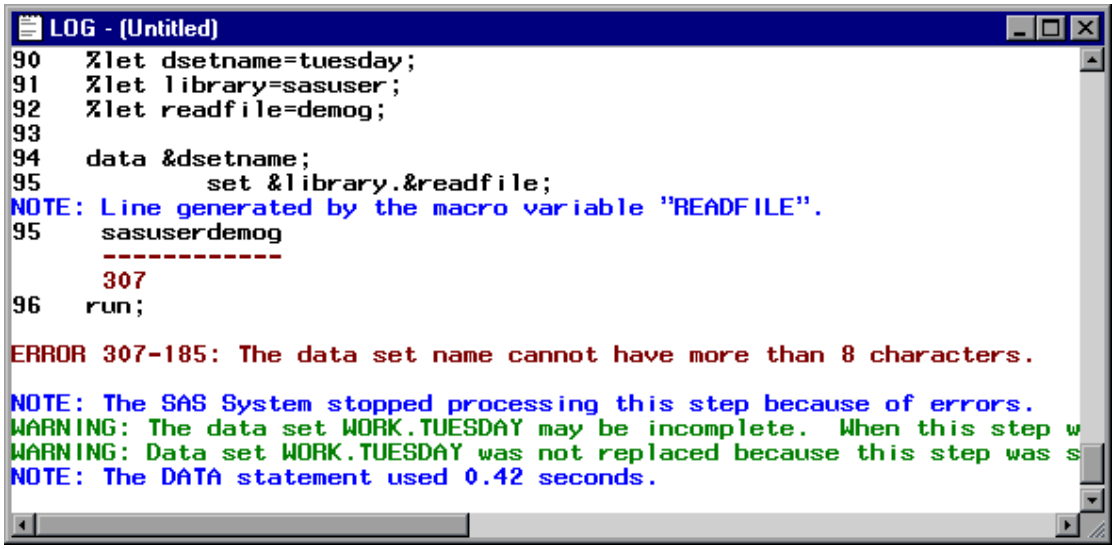


A Possible Problem

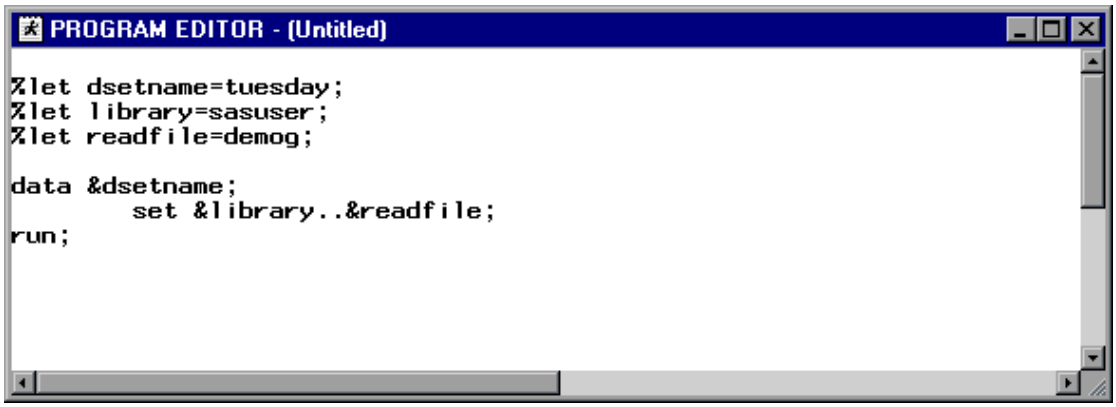
One possible problem in using macro variables is illustrated by the following program:



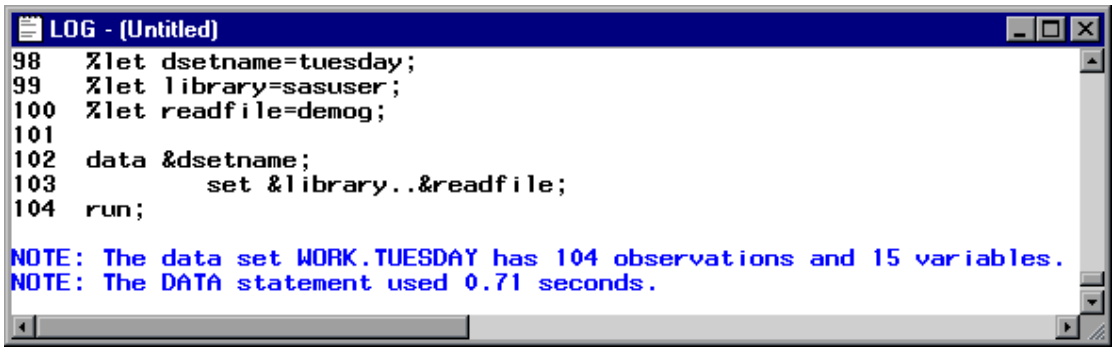
The SAS log from this program shows an error:



A full stop after a macro variable name can act as a delimiter to the name itself. We use 2 full stops in the program, one to act as the end of macro variable name marker, and one for the full stop between the library and dataset name:



The SAS Log now reports no errors:

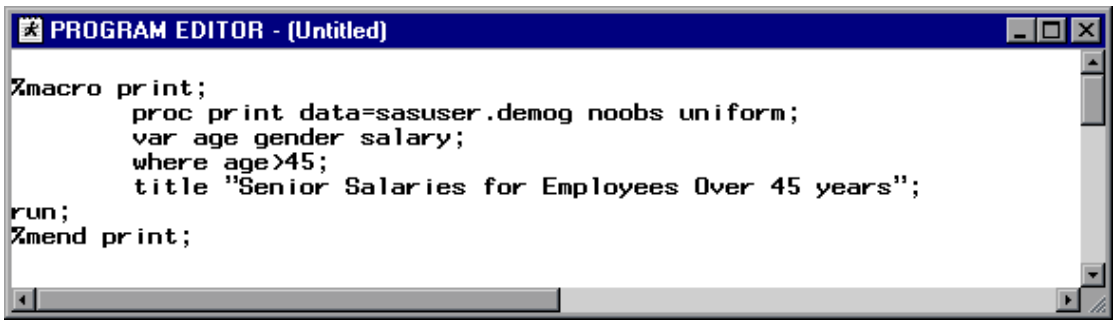


CREATING MACRO CODE BUNDLES

A Macro code bundle is commonly referred to as “a macro”, as distinct from a “macro variable” which we have just been discussing.

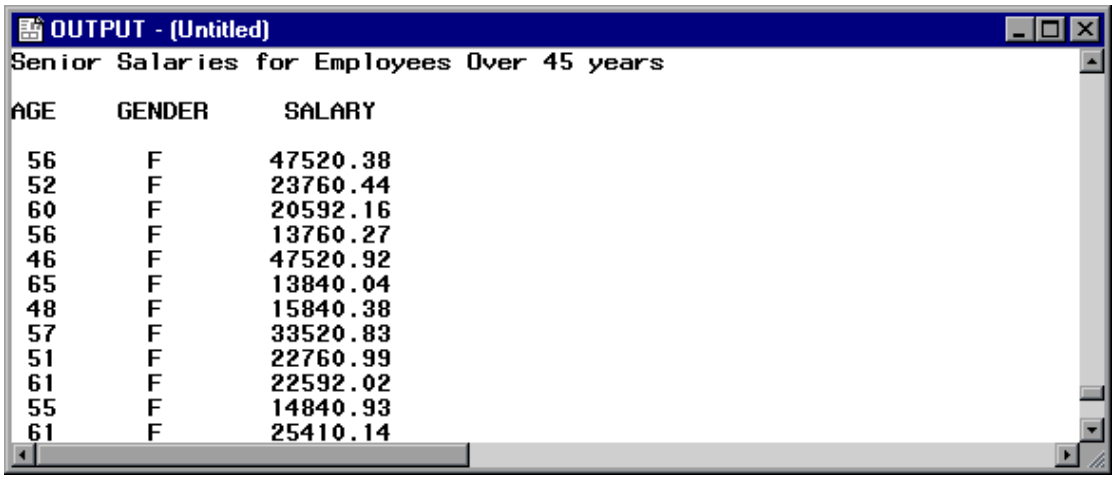
A macro code bundle allows you to define what the bundle will contain, and then run it.

Define the macro bundle using %macro and %mend statements:



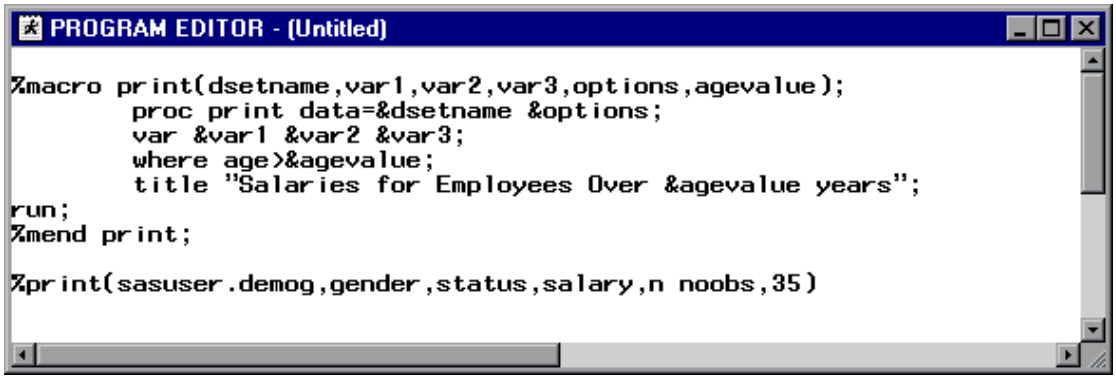
The macro is stored in compiled form in the work.sasmacr catalog and is ready to be run.

Run the macro by “calling” it :

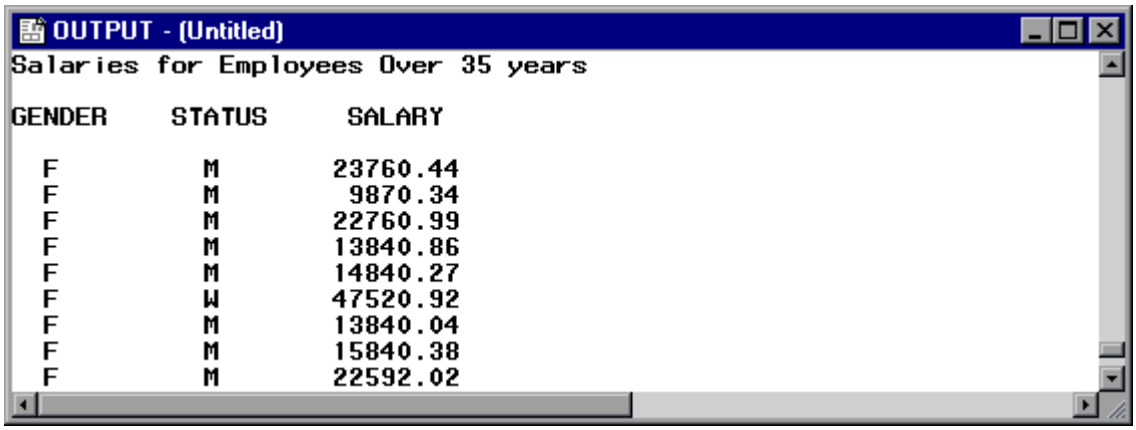


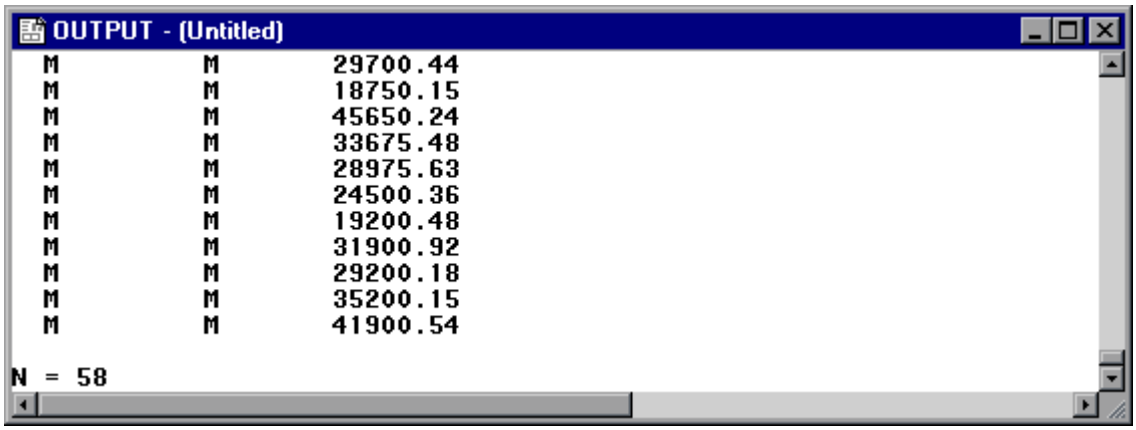
Notice the macro call requires no semi-colons, as the code that it generates contains all the semi-colons that are required.

Passing Parameters
We can also create a macro code bundle that expects to be passed several parameters.



Notice that we have used &agevalue in 2 places in the code bundle, and also that the &options macro variable can contain as many Proc Print options as we require.





USING THE MACRO PROGRAMMING LANGUAGE

The macro programming language consists of statements such as

```
%IF     %DO     %ELSE   %THEN   %END
```

among others.

There are also a number of character handling functions such as

```
%UPCASE         %SUBSTR         %SCAN
```

There are some Call Routines such as

```
CALL SYMPUT
```

For passing values from the data step to a macro variable

We can use these techniques within a macro code bundle to help control the program:

PROGRAM EDITOR - (Untitled)

```
%macro proc(proc,dsetname,var1,var2,var3,agevalue);
        %if %upcase(&proc)=PRINT or %upcase(&proc)=MEANS
        %then %do;
                proc &proc data=&dsetname;
                var &var1 &var2 &var3;
                where age>&agevalue;
                title "Salaries for Employees Over &agevalue years";
                run;
        %end;
        %else %put Please use Proc Print or Proc Means;
%mend proc;

%proc(print,sasuser.demog,gender,status,salary,25)
```

OUTPUT - (Untitled)

```
Salaries for Employees Over 25 years

OBS     GENDER     STATUS      SALARY

  1       F          M         8870.23
  3       F          M        28512.66
  4       F          M        14840.27
  5       F          M        47520.38
  6       F          M        23760.44
  7       F          M        20592.16
  8       F          M        13760.27
  9       F          W        47520.92
 10       F          M        13840.04
 12       F          M         9870.34
 15       F          M        15840.38
 16       F          M        33520.83
```

Calling Proc Means with the Macro PROC:

PROGRAM EDITOR - (Untitled)

```
%proc(means,sasuser.demog,age,height,weight,55)
```

OUTPUT - (Untitled)

```
Salaries for Employees Over 55 years

Variable     N            Mean         Std Dev         Minimum         Maximum
-----------------------------------------------------------------------------
AGE         11      59.8181818       3.3111382      56.0000000      65.0000000
HEIGHT      11      66.2272727      12.7502228      52.0000000      88.5000000
WEIGHT      11      67.6363636       6.1036502      60.0000000      81.0000000
-----------------------------------------------------------------------------
```

SUMMARY

The Macro Language Facility within the SAS System provides the programmer with more tools with which to control and improve SAS programs. This paper has shown how the 3 main aspects of the macro language can be used to create more generic and therefore more efficient SAS programs. For more information, consult the SAS Macro Language Usage and Reference or contact the author at the address on the next page.

CONTACT INFORMATION

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

Company:	Amadeus Software Limited
Address:	Mulberry House, 9 Church Green, Witney, Oxon OX28 4AZ
Work Phone:	+44 (0) 1993 848010
Email:	info@amadeus.co.uk
Web:	www.amadeus.co.uk