Paper 102-2008

Using a SAS® Table to Drive Any Macro

Mike Whitaker, CTB/McGraw-Hill, Monterey, CA

ABSTRACT

This paper presents a macro that repetitively calls a target macro using data read from a SAS® table as the parameters. The macro is completely general, and can be used at any point in a SAS® program, making it a useful utility for solving a wide variety of programming problems. Five applications given are: creating macro symbol lists, creating simple SAS® library reports, preventing unintentional inclusion of a new column to a SAS® table, systematic renaming of columns and using data from an Excel worksheet.

INTRODUCTION

It is possible to use a SAS® table as the source of parameters for repetitive execution of one or more macros. Two techniques often used are the Call Execute routine to place the target macro in queue after the data step and using an execution harness that uses a macro symbol array to execute the target macro. This array can be created with a Proc SQL statement or a Call Symput statement from a SAS® data step.

As show these techniques, we begin with the following macro.

```
%macro Printit(sastable);
    proc print data=&sastable;
    run;
%mend;
```

Along with a SAS® table named MyWorkTables containing a single column named SASTable with three observations.

SASTable
Dataset1
Dataset2
Dataset3

The following code demonstrates the Call Execute technique. When this data step runs, each string in “Execmacr” is inserted between the executing step, and any other data step or proc step following this step, effectively modifying the SAS® program with macro calls.

```
Data _null_;
  Set MyWorkTables;
  Execmacr = '%printit('||strip(sastable)||')';
  Call execute(execmacr);
  Run;
```

The second technique is a macro execution harness. This is a macro that will repeatedly call another macro based on a macro symbol string containing one or more parameters.

```
%macro doit(datasetlist);
   %local j thissastable;
   %let j = 1;
   %let thissastable = %scan(&datasetlist,&j);
```

```
   %do %while(%quote(&thissastable)~=);
       %printit(&thissastable);
       %let j = %eval(&j+1);
       %let thissastable = %scan(&datasetlist,&j);
       %end;
%mend;

Proc SQL noprint;
  Select sastable into :datasetlist separated by ' '
    From MyworkTables;
  Quit;

%doit(&datasetlist)
```

This paper presents a macro we call %TableDrivenMacro, which can replace the above techniques with a simple macro execution. The macro reads the target macro parameters directly from a SAS® table and then uses those values to execute the target macro for each row read from the SAS® table. Using this macro, the code above can be replaced with the follow macro execution.

```
%TableDrivenMacro(table=MyWorkTables,
                  Macro=Printit,
                  Parms=SASTable)
```

The next sections will present the execution requirements of the macro, followed by some of its advantages over the techniques presented here. Next, we give five applications using the macro. Finally, we give our conclusions. A copy of the macro is presented in Appendix A.

EXECUTING THE MACRO

%TableDrivenMacro's execution requirements are straightforward. The basic syntax is:

```
%TableDrivenMacro(Table=[SAS® table name],
                  Macro=[SAS® macro name],
                  Parms=[List of names of Macro Parameters/Table columns])
```

Where the parameters are:

1. **TABLE:** This is the SAS® table containing the parameter data. It can be constructed in any way a SAS® table is acceptable to a standard Set statement in a DATA STEP, so any number of valid Data Set options are permitted.
2. **MARCO:** This is the name of the target SAS® macro, without the % use when executing the macro.
3. **PARMS**: This is a list of macro parameters used by the target macro. They can be either keyword or positional parameters, and must be the same as the parameters found in the target macro's parameter list. The column names presented to the %TableDrivenMacro when it reads the SAS® table is must be identical to name of the parameters to the target Macro.

When fulfilling requirement 3, it is not important that the columns in the SAS® table to be named the same as the parameters in the SAS® macro. If they differ, we can rename the columns when we read the table. For example, if the column in MyWorkTables is named Col1, we use a rename option to align the column name and the macro name as shown below. To use the macro effectively, the column names as presented by the implied SET statement of the SAS® table must be coordinated with the parameter names used by the Target Macro.

```
%TableDrivenMacro(table=MyWorkTables(rename=(Col1=SASTable)),
```

```
                    Macro=Printit,
                    Parms=SASTable)
```

ADVANTAGES OF THE MACRO

%TableDrivenMacro has a number of advantages over the other two techniques presented.

1. **The Macro is General**. Since the macro itself uses no SAS® language statements, it can use it with any target macro at any appropriate place in a SAS® program. This is not the case for the Call Execute technique, where only macros that generate SAS® code at a Data or Proc boundary are appropriate.
2. **Calling the Macro is Simple**. Since the macro reads the SAS® table directly, there is no need to perform preprocessing steps like the Proc SQL statement seen in the second technique. In fact, the macro can be viewed as an improved execution harness over the harness shown. In addition, its ability to correctly process multiple parameters when calling the target macro is a clear advantage over the second technique, where all parameter are required to be placed in a single macro symbol.
3. **Data can come from Anywhere**. Although not a unique advantage, the technique promotes an interesting aspect of SAS®, where the data used to drive a target macro can be stored in any external data storage system (Oracle, MS Excel, MS Access, SQL Server, XML). All that is needed is an appropriate libname, possibly using a libname engine which may require a SAS/ACCESS® module. A suitable SQL or Data Step view could also be used. The basic requirement is that if a standard SET statement can read the data, then the object of that statement can be used as a parameter for this macro.
4. **Good Production Processing is Promoted**. For any production level SAS® Program, the macro promotes the use of tables to store data derived from a set of business rules. Given this fact, it improves the repeatability of a job, the documentation of how a job was executed or should be executed, and the longevity of the job over time since only the data needs to be updated, not code.

APPLICATIONS OF THE MACRO

CREATING MACRO SYMBOL LISTS

Assume we need a macro symbol that contains a list of the contents of a column in a table. This is the requirement fulfilled by the SAS® SQL code at the beginning of this paper. Instead of using that code, we can use %TableDrivenMacro when creating this list. We begin with the simplest possible macro.

```
%macro column(col1);
    &col1
%mend;
```

The macro simply returns the value of the argument. Using this macro, and any table, %TableDrivenMacro will create a space delimited list of the values of a given SAS® table's column. Using the example table from the SAS® SQL code, and the macro above, we can form an equivalent statement.

```
%Put DataSetList = %TableDrivenMacro(Table=MyWorkTables(
                                        rename=(SASTable=Col1)),
                                    Macro=Column,
                                    Parms=Col1);
```

This command puts the following string onto the SASLog.

DataSetList = Dataset1 Dataset2 Dataset3

PRODUCING SIMPLE SAS® LIBRARY REPORTS

During any SAS® session, SAS® keeps track of most session objects with SAS® Data Dictionary views. Views are "Virtual Tables" that do not contain data, but instead contain the code to acquire the data into a tabular form. Views can be treated just like any readonly table. The views we will be using are:

- **SASHELP.VSTABLE**: a view that lists every SAS® table available at any given time
- **SASHELP.VTABLE**: a view giving additional information on the tables found in SASHELP. VSTABLE
- **SASHELP.VCOLUMN**: a view that gives information on each column of each table found in SASHELP. VSTABLE. "

In this example, we will use the view SASHELP.VSTABLE, and the macro

```
%macro tableinfo(libname=,memname=);
     Title "The Contents and first 20 rows of the table &libname..&memname";
     Proc Contents data=&libname..&memname Varnum;
        Run;
     Proc print data=&libname..&memname(obs=20);
        Run;
%mend;
```

This execution of the %TableDrivenMacro will result in a small report on each table found in the SASUSER Library

```
%TableDrivenMacro(Table=sashelp.vstable(where=(libname='SASUSER')),
                  Macro=Tableinfo,
                  Parms=Libname Memname)
```

PREVENTING UNINTENDED COLUMN INCLUSIONS IN A SAS® TABLE

When updating a SAS® table using the SAS® language command UPDATE, it is often the case that we need to prevent the introduction of any additional columns through the transaction table given in this command. To prevent this, the ability to form lists of columns is very useful. Using the macro defined in the first example, the following statement uses %TableDrivenMacro to create a list of columns suitable for a Keep= dataset option.

```
Data sasuser.primarytable(
     keep=%TableDrivenMacro(Table= SASHelp.VColumn(
                                     rename=(Name= Col1)
                                     Where=(libname='SASUSER'
                                        And
                                            Memname='PRIMARYTABLE')),
                            Macro=Column,
                            Parms=Col1));
     Update  sasuser.primarytable TransactionTable;
        By Col1 Col2;
     Run;
```

We can make this a little easier to read if we create a small utility macro called 'CurrentCols'

```
%macro CurrentCols(Data);
    %local Libname Memname;
    %let libname = %scan(&data,1,%str(.));
    %let Memname = %scan(&data,2,%str(.));
    %if %quote(&memname)=
          %then %do;
                %let Memname = &libname;
```

```
                %let libname = %sysfunc(getoption(user));
                %if %quote(&libname)=
                     %then %do;
                           %let libname = WORK;
                           %end;
                %end;
       %if ~%symexist(compileCol)
           %then %do;
                %global compileCol;
                %let compilecol = 1;
                %macro column(col1);
                   &col1
                %mend;
                %end;
      %TableDrivenMacro(Table= SASHelp.VColumn(
                                    rename=(Name=Col1)
                                    Where=(libname="%upcase(&libname)"
                                       And
                                           Memname="%upcase(&memname)")),
                           Macro=Column,
                           Parms=Col1)
        %mend;
```

Now, the update code becomes

```
Data sasuser.primarytable(keep=%CurrentCols(sasuser.primaryTable));
     Update  sasuser.primarytable TransactionTable;
        By Col1 Col2;
     Run;
```

SYSTEMATIC RENAMING TABLE COLUMNS

We begin this example with two SAS® tables, both having columns called scr01 – scr40. One table holds reading scores, while the other table holds language scores. Our goal is to merge these two tables using the student id, which is also common to both tables, into a single table containing both reading and language scores. A straight forward way to do this is to merge these two tables by student id. We show the data and possible code below.

```
Data Reading;
  length Studentid scr01-scr40 $1;
  input studentid (scr01-scr40) ($1.);
  cards;
 1 12343234323454343234354343433234323433 4
 2 34354343231231023234302123023304343234 32
 3 34354343231231023234302123023304343234 32
;;;;

Data Language;
  length Studentid scr01-scr40 $1;
  input studentid (scr01-scr40) ($1.);
  cards;
 3 12343234323454343234354343433234323433 4
```

```
 2 34354343231231023234302123023304343234321
 1 34354343231231023234302123023304343234321
;;;;

Proc Sort data=Reading;
  By StudentId;
  Run;
Proc Sort data=Language;
  By StudentId;
  Run;
Data ReadingLanguage;
  Merge Reading(in=a)
        Language(in=b);
    By studentid;
  If a & b;
  Run;
```

Unfortunately, this program results in a dataset with only 41 columns, since the column names are the same in the both tables. We need to rename some columns to correct this. This can be done as a data set option on the each table in the merge statement.

```
Data ReadingLanguage;
     Merge Reading(in=a
            rename=(scr01=rdscr01 scr02=rdscr02 . . . scr40=rdscr40)
           Language(in=b
            rename=(scr01=lascr01 scr02=lascr02 . . . scr40=lascr40));
     By studentid;
     If a & b;
     Run;
```

We can use %TableDrivenMacro to help form this rename statement. We use the SAS® Data Dictionary view SASHELP.VCOLUMN to get a list of the columns we want to rename in the reading and language files. The code below shows the list of columns we will want to rename.

```
Proc Print data=SASHELP.VCOLUMN(
               keep=libname memname name
               where=(libname = 'WORK'
                  and memname = 'LANGUAGE'
                  and upcase(substr(name,1,3)) = 'SCR'));
```

We then create a macro to correctly form the rename syntax.

```
%macro RenameCol(name);
    %global Gprefix;
    &name = &prefix&name
%mend;
```

In this example, we are using a symbol that is not being passed into the macro as a parameter. We will be storing and retrieving this symbol from the global symbol table.

We also create a macro to call the %TableDrivenMacro correctly for this situation.

```
%macro RenameScr(data,prefix);
    %local Libname Memname;
    %global GPrefix;
    %let Gprefix = &prefix;
    %let libname = %scan(&data,1,%str(.));
    %let Memname = %scan(&data,2,%str(.));
    %if %quote(&memname)=
        %then %do;
              %let Memname = &libname;
              %let libname = %sysfunc(getoption(user));
              %if %quote(&libname)=
              %then %do;
                    %let libname = WORK;
                    %end;
              %end;
%TableDrivenMacro(table=SASHELP.VCOLUMN(
                    where=(libname = "%upcase(&libname)"
                       and memname = "%upcase(&memname)"
                       and upcase(substr(name,1,3)) = 'SCR')),
                macro=RenameCol,
                Parms=name)
%mend;
```

We can now use the RenameScr macro to help us write our rename code

```
Data ReadingLanguage;
     Merge Reading(in=a  rename=(%RenameScr(reading,rd)))
           Language(in=b rename=(%RenameScr(language,la)));
     By studentid;
     If a & b;
     Run;
```

USING DATA FROM EXCEL

Note: In order to do this techneque, the appropriate Libname Engine you will be using may need a SAS/ACCESS® License for the external data system. This would be SAS/ACCESS® for Oracle when reading Oracle tables, or SAS/ACESS® for PC Files when reading Excel or MS Access Tables.

%TableDrivenMacro does not care where the data is stored, just if we can read the data with a SET statement. For example, assume we have an Excel worksheet that has two named columns, Content and TestLevel. The worksheet is called “Tests” and is in an Excel File called “C:\Tests.xls”.

Content	Testlevel
RD	3
RD	4
RD	5
MA	3
MA	4

```
MA        8
```

Then the following code will print the Macro Symbols “Content” and “TestLevel” out onto the Saslog.

```
libname tests excel 'C:\sasglobalforum\tests.xls';
%macro test(content=,testlevel=);
  %put Content=&content Testlevel=&testlevel;
  %mend;
%TableDrivenMacro(table=TESTS.'Tests$'n,
                  macro=test,
                  Parms=content testlevel)
```

The requirement to quote the worksheet name TEST as ‘Tests$’n is a requirement of the SAS/ACCESS® for PC Files component to correctly read the excel table in a standard Set statement. This is peculiar to the Libname method of reading MS Excel worksheets as SAS® tables.

CONCLUSION

SAS® macros are one of the most useful tools the SAS® programmer has to standardize code, methods and processes. This paper has presented a macro that enables a programmer to use SAS® tables as inputs to perform repeated execution of any macro he wishes to use. The data can come from a number of data storage systems, since the only requirement to for using that data is that it is accessible through the Libname statement, using, if needed, an appropriate Libname engine. Finally, because the macro only uses macro statements, we have the freedom to use thisechnique to write code at any point in a SAS® pogram, resulting is a very general utility for the SAS® programmer.

A number of examples were given to show how this macro utility can solve some common programming problems. But the macro can also be used as a primary tool to improve production level data processing tasks, since it promotes the use of tables to store program requirements and business rules. I leave it to the reader to discover the macro’s ability to achieve this improvement at their own business or academic setting.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. If you would like a copy of this macro, and a copy of all of the examples presented in this paper, contact the author at:

Mike Whitaker
CTB/McGraw-Hill
20 Ryan Ranch Rd
Monterey, Ca 93940
(831) 393-7992
Mike_Whitaker@ctb.com

APPENDIX A

```
/*=============================================================================*
<header>
    <company>CTB/McGraw-Hill</company>
    <copyright>2008</copyright>
    <name>TableDrivenMacro</name>
    <author>Mike Whitaker</author>
    <description>
      The Macro reads a SAS Table and uses the values of
      a selected set of columns for that table to
      execute a macro with those values, one execution per
      row of the table.
    </description>
    <usage>
       Use this whenever you want to execute a macro a number of times, and the
       data to control the execution can be retrieved from an existing table
    </usage>
    <remarks> <ol class="decimal">
      <li>The Macro's Parameters and the Table's Columns must be the
          Same Names</li>
      <li>Any SASTable will work</li>
      <li>Any Dataset Options are permitted</li>
      <li>Any Macro is permitteded</li>
      <li>The Macro can call itself</li>
      <li>The Table and Macro Must exist beforehand</li>
    </remarks>
    <history>
        <action initials="MDW" date="Oct 12 2006"> Macro Created</action>
        <action initials="MDW" date="Jan 13 2008"> Made Tableid, libref and RC
Local Symbols</action>
    </history>
</header>
 *=============================================================================*/
/*=============================================================================*
<comment>
   <code type="Macro"> TableDrivenMacro</code>
   <param name="Table" default="">Any SAS Table with any DataSet options</param>
   <param name="Macro" default="">Any SAS Macro without the %</param>
   <param name="Parms" default="">The Column Names to use</param>
   <example>%TableDrivenMacro(<br>
         Table=myTable,<br>
         Macro=MyMacro,<br>
         Parms=Col1 Col2)<br></example>
</comment>
 *=============================================================================*/
%macro tabledrivenmacro(
```

```
           Table=,
           Macro=,
           Parms=
           );
%*****************************************************;
%* The library with the SAS dataset that has the     ;
%* Data for the macro variables used by the macro    ;
%* within the do loop below                          ;
%*****************************************************;
%local Tableid Rc libref;
%if %qscan(&Table,2,.) ~=
  %then %do;
        %let libref = %qscan(&Table,1,.);
        %if (%sysfunc(libref(&libref.)) ~= 0)
            %then %do;
                  %put %sysfunc(sysmsg());
                  %Return;
                  %end;
        %end;

%*******************************************;
%* open the SAS Table of macro variables     ;
%*******************************************;
%let tableid= %sysfunc(open(&Table,is));
%if (&tableid = 0) %then
   %do;
   %put %sysfunc(sysmsg());
   %Return;
   %end;

%*******************************************;
%* State that we want all of the values      ;
%* of on a row                               ;
%*******************************************;

%syscall set(tableid);

%*******************************************;
%* Read the first observation                ;
%*******************************************;

%let rc=%sysfunc(fetch(&tableid));
%if (&rc ~= 0) %then
   %do;
   %put %sysfunc(sysmsg());
   %let rc = %sysfunc(close(&tableid));
   %Return;
```

```
   %end;

%*******************************************;
%* Do until end of file found               ;
%*******************************************;

%do %while(&rc=0);

%*******************************************;
%* Use the Parameters List to strip off the   ;
%*  White Space                               ;
%*******************************************;
   %local ThisParm
          parmstring
          J;
   %let parmstring=;
   %let j = 1;
   %let Thisparm = %qscan(&Parms,&j);
   %do %while(%quote(&thisparm) ~=);
     %let &thisparm = %sysfunc(strip(&&&thisparm));
     %let parmstring = &parmstring%str(,)&thisparm=&&&thisparm;
     %let j = %eval(&j+1);
     %let Thisparm = %qscan(&Parms,&j);
     %end;

%*********************************************;
%* Nibble off the first comma and put in         ;
%*  Parentheses.                                  ;
%*********************************************;
   %let parmstring = (%substr(&parmstring,2));

%*******************************************;
%* Call the macro you want to execute using   ;
%* this parameter string                      ;
%*******************************************;

    %&Macro&parmstring

%*******************************************;
%* Now Read the next record from the Table  ;
%*  and do it again                         ;
%*******************************************;
  %let rc=%sysfunc(fetch(&tableid));
  %end;
%*******************************************;
%* close the table                          ;
%*******************************************;
```

```
%if (&rc ~= -1) %then
   %do;
   %put %sysfunc(sysmsg());
   %Return;
   %end;
%let rc = %sysfunc(close(&tableid));
%mend;
```