MOBILE MACROS – GET UP TO SPEED SOMEWHERE NEW FAST

Author: Patricia Hettinger, Data Analyst – Consultant
Oakbrook Terrace, IL

ABSTRACT:

Have you ever been faced with this scenario? It's your first day on the job and you know absolutely nothing about the data you're supposed to be working with. The documentation is scanty and people are too busy to give a detailed orientation – if they know enough to do so in the first place. This paper gives some macros the reader can use anywhere and explains some SAS® concepts behind them. It also details how you can use SAS® Enterprise Guide to create input screens for more flexibility.

INTRODUCTION:

As a consultant, the author sees a lot of different SAS installations. There's usually limited time for getting up to speed on each client's system and data idiosyncrasies. This paper covers

1. Use of dictionary tables to explore possible values for variables whether the exact name is known or not. The nuances of dictionary tables for Oracle and SAP systems will also be detailed.
2. Creating SAS code to generate other SAS code.
3. Returning a value from a macro for use in further processing
4. Checking for the existence of a dataset and processing accordingly. Very useful when using a primitive scheduler that does not have file dependency options (i.e. Cron)
5. Use of time and date variables
6. Difference between setting a macro variable within the macro as opposed to the data step or proc SQL
7. Use of %SYSFUNC and limitations
8. Automatic macro variables you may find useful
9. Use of parameters/prompts manager in SAS® Enterprise Guide to receive input and automate searches
10. Point processing – what is this, why would you use this and where?

MACROS AND CONCEPTS:

A. Time window – check for elapsed time before performing an action.
Do processing based on elapsed time. Also displays some useful automatic variables.

Concepts in this section:

There are a limited number of macro functions available. %SYSFUNC lets you use most of the data step functions by enclosing them with this macro function. The basic syntax is %SYSFUNC(*function*(*arg1,arg2,etc.*)). You can even nest %SYSFUNC calls with this syntax %SYSFUNC(*function*(%SYSFUNC(*function*(*arg1,arg2,etc*)).

INTCK function refinements. This is one of the data step functions that have no equivalent macro function. You however can call this by using %SYSFUNC.

Using proc EXPORT with the DBMS= and DELIMITER= option gives you great flexibility in how you can export a SAS dataset without having to code FILE and PUT statements.

Proc CONTENTS will tell you when a SAS dataset was last updated but using the OPEN function can be much quicker, especially if you want to store that time for future use. The ATTRN function will give you many attributes of the dataset, including the modification date as in this example.

You may pass parameters to a macro as in this example. When you do so, the macro variable is only valid within that macro unless you make it global.

Code in its entirety:

```
%macro time_window(start_time);
       %LET curtime=%SYSFUNC(time());
       %let curdate=%SYSFUNC(date());
       %LET filetime=%SYSFUNC(compress(%SYSFUNC(time(),time11.),":"));
       %let filedate=%SYSFUNC(date(),date9.);
       %put &curtime &curdate &filetime &filedate;
       %let minutes_pass=%SYSFUNC(INTCK(min,&start_time,&curtime));
       %if &minutes_pass gt 180 %then %do;
              data dms.newlead;
              set SASPERS.baseball;
              run;
              PROC EXPORT DATA= dms.newlead
                     OUTFILE= "/SAS/data/files/newleads_&filedate_&filetime"
                     DBMS=dlm  REPLACE;
                     delimiter = "|";
              RUN;
       %end;
%mend;
data _null_;
attrib pw format=time12.;
   dsid=open("SASPERS.baseball");
   pw=timepart(attrn(dsid,"modte"));
   call symput('pw',pw);
      put  pw;
run;
%time_window(&pw);
run;
%put &sysdate &systime &sysuserid;
```

Step by step:

1. **`%macro time_window(start_time);`** Pass a parameter to the macro called start_time. Start_time will be local to the macro and only exist during its execution. Any reference to it outside of the macro will result in an error.
2. **`%LET curtime=%SYSFUNC(time());`** %SYSFUNC lets you execute most of the functions available to you in a dataset. Here it lets you extract the current system time as opposed to &systime which gives you the time when you first started the SAS session. The same is true for the date() function which gives the current date not the date (&sysdate) you first started the session.
3. **`filetime=%SYSFUNC(compress(%SYSFUNC(time(),time11.),":"));`** Nested %SYSFUNC calls. Just as you can nest functions in a data step, you can nest them in setting macro variables. The difference is that each one needs %SYSFUNC calls for predictable results. Here this statement calls the current time function and removes the colons from the results.
4. **`%let minutes_pass=%SYSFUNC(INTCK(min,&start_time,&curtime));`** You may have used the INTCK function to get the number of days, weeks, months or years between one date and another. You may not have known that you can use the same function to do time calculations. Here we are calculating the number of minutes between one time and another. The logic is to create a new SAS dataset and output it to a pipe-delimited dataset if it's been more than 3 hours since the last one was created. Append the date and time as an identifier.
5. **`PROC EXPORT DATA= dms.newlead`**
   `OUTFILE= "/SAS/data/files/newleads_&filedate_&filetime"`
   `DBMS=dlm  REPLACE;`
   `delimiter = "|";`
   Use proc export to export your new file in pipe-delimited format.
   The macro filedate variable was set in mmddyyyy format and filetime in hhmmss to give a unique identifier to the output file.

6. **dsid=open("SASPERS.baseball");** Some macro variables such as the last time a SAS dataset was updated are just better set within a data step. This code set a variable to the dataset id of the SAS dataset you're trying to find the latest modification date for.
7. **pw=timepart(attrn(dsid,"modte"));** use the dataset id to find the latest modification date and take just the time portion.
8. **call symput('pw',pw);** while still in the data step, create macro variable pw or update its value.
9. **%time_window(&pw);** call the time_window macro with the last time the dataset was updated

B. Check_source. Check for existence of a SAS data set and return a value

It is possible to write a macro that behaves somewhat like a true function in that you can give it arguments and it will return a value. The limitation here is that there cannot be any other kind of SAS code like data steps or procedures in the macro or that code is what it will return.

Code in its entirety:

```
%MACRO CHECK_SOURCE(DSNAME);
       %LET EXIST_IND = %SYSFUNC(EXIST(&DSNAME));
              %put &exist_ind;
       %if &exist_ind eq 1 %then %LET found_ind = Y;
       %else %LET found_ind = N;
       &FOUND_IND
%MEND CHECK_SOURCE;
%MACRO TIMEIT(DSNAME);
%IF %CHECK_SOURCE(&DSNAME)= Y %THEN %INCLUDE PROGRAM1;
       %else %do;
       DATA _NULL_;
              FILE PRINT NOTITLES;
              PUT "**** &dsname not available";
              PUT "Please investigate";
       RUN;
%END;
%MEND TIMEIT;
%TIMEIT(SASPERS.AUCTIONS);
```

Step by step:

1. %**LET EXIST_IND = %SYSFUNC(EXIST(&DSNAME));** %SYSFUNC calls the EXIST function to determine whether a certain SAS dataset exists or not. If the value is 1 then the dataset exists otherwise it's 0.
2. **&FOUND_IND** – this is the value we want to return from the macro. Limitation is that the macro will return regular SAS code if present. To demonstrate, if we inserted even our data _NULL_ statement in the check_source macro, the function would return the statement with the dsname variable substituted as well as the FOUND_IND:

   ```
   DATA _NULL_; FILE PRINT NOTITLES; PUT "****
   SASPERS.AUCTIONS not available"; PUT "Please investigate";
   RUN; N
   ```

   Instead of just N which is what we checking for in the TIMEIT macro.
3. **%IF %*CHECK_SOURCE*(&DSNAME)= Y %THEN %INCLUDE PROGRAM1;** Include the program using the SAS dataset as input and execute.
4. **FILE PRINT NOTITLES;PUT "**** &dsname not available";** One way to report when there is nothing to report. File PRINT directs output to a report listing.

C. Getmemb macro and outlist table creation– Process every member in the library for variable values equal to those requested

There are two possible constructs for loop processing within a macro: %do %until(condition) and %do %while(condition). %Do %until checks for the value at the end of the expression and %do %while at the

beginning. If you were incrementing a counter for execution ten times, you would code %do %until(&ctr gt 10) and %do %while(&ctr le 10).

%Eval is a function that forces a macro variable to be treated as a numeric value. This is used to make sure the counter increments properly. The %LET CTR = %EVAL(&CTR+1) statement will increment the ctr variable by 1, making it 2, 3, 4, etc. each time it is executed in the loop. Leaving out the %eval will give you a literal value of 0+1+1+1 and so on as often as the statement is executed.

Using SAS's dictionary library has a few advantages over proc contents data=_all_; One is the proc contents procedure will stop dead if it hits a member name in the library that has embedded spaces or special characters, quite likely if libnaming external data like SAP or Excel spreadsheets. The dictionary has no such restrictions. Here we are retrieving member names by using proc SQL, getting the number of member names returned by examining the automatic SAS variable SQLOBS.

The last major concept in this section is POINT processing or a random read of a dataset based upon a pointer. This is likely the fastest way to extract an observation from a SAS dataset, faster even than an index. The pointer must be a variable in the data step. Here we set it according to the macro variable ctr.

Code in its entirety including comments. Library must be allocated first.

```
%macro getmemb(endcount);
*Do this loop until the counter is greater than the number of members in the
library;
       %let ctr = 1;
              %do %until (%eval(&ctr) gt %eval(&endcount));
*memname and libname from memname and libname variables in this dataset;
*Point processing lets us find each member in the list quickly.  Data _null_
suppresses any work file from being processed.  Call symput will give us the
library and member name for further processing;
                     data _null_;
                           oneob=&ctr;
                           set outlist
                           point=oneob;
                           output;
                           memname=trim(memname);
                           call symput('memname',memname);
                           call symput('libname',libname);
                           stop;
                     run;
%put &memname &libname;

*can do anything here, not just print or proc contents;  The advantage of this
is that even if you error out for some reason, you can just continue with the
next member;
              proc print data=&libname..&memname;
              var address_line1 address_line2 address_city address_state
              address_zip_code carrier_route_code mail_sort_Sequence;
              WHERE carrier_route_code IN
                     (
                           '60181C001'
                           ,'60181C002'
                           ,'60181C003'
                           ,'60181C004'
                           ,'60181C005'
                     )
              ;

              title "CARRIER ROUTE SEARCH FOR &LIBNAME..&MEMNAME, MEMBER #&CTR";

              RUN;
       *INCREMENT THE COUNTER BY 1;
       %LET CTR = %EVAL(&CTR+1);
       %END;
```

```
%MEnD GETMEMB;
*END OF MACRO – must be submitted first before any use;
*Create table outlist for a given library. Using the macro – get a list of
tables.  Here we are using the dictionary but proc contents would work also.
We're interested in DATA types only;
      proc sql;
            create table outlist as
            SELECT *
            FROM
            (
                  select *
                  from dictionary.MEMBERS
                  where memtype = 'DATA'
                  and LIBNAME = 'CUSTOMER')
            )
            ORDER BY LIBNAME, MEMNAME
            ;
            QUIT;
            RUN;
*The nice thing about using the dictionary with proc sql is that you will
automatically get the count of records returned;
%PUT &SQLOBS;
*Call the macro with the number of records returned;
%GETMEMB(&sqlobs);
RUN;
```

Step by step:

1. **%do %until (%eval(&ctr) gt %eval(&endcount));** %LET CTR = %EVAL(&CTR+1);%END; Loop processing within a macro. The endcount variable is incremented at the end of the loop. Syntax is %do %until (*condition*). %eval forces the macro variable to be treated as number, rather than character.

2. **oneob=&ctr;set outlist point=oneob;** Random read of a dataset based upon a pointer. Oneob is set to the value of the counter and the point statement selects the corresponding observation. Point comes after the set statement, no semicolon. This lets you pick just the record you need instead of cycling through the entire dataset.

3. **proc sql;** - many people's favorite SAS procedure, including the author. One advantage is that you don't need to sort the datasets first to merge or join them together. Another advantage is proc SQL uses a version of SQL, the query language used in most relational database systems like Oracle, DB2, Teradata, etc. This makes the logic easier to follow for someone not conversant with SAS.

4. **create table outlist as** create a SAS dataset as output from query

5. **select * from dictionary.MEMBERS where memtype = 'DATA' and LIBNAME = 'CUSTOMER').** There are several dictionary tables. The MEMBERS table contains a list of all datasets within a particular library. Here we are asking for all members in the CUSTOMER library.

6. **%PUT &SQLOBS;** This is an automatic macro variable that gives the count of observations every time you run Proc SQL. Unlike '_n_' in the data step, you don't have to code 'call symput' to create it.

About the SAS dictionary tables

If you have RDBMS databases allocated to your session using the libname option, querying the dictionary tables will result in a dynamic call to the RDBMS for metadata. This can take some time. For example, if you have SAP allocated to your session via the SAS R/3 interface with trace turned on, you will see calls very much like this:

```
r3prep().
Entering send_metadata_request_and_load_metadata()
ACCESS ENGINE: Table returned is AEOI (Revision Numbers)
```

The above message will repeat until all of the SAP table information has been returned. Once all of the SAP metadata has been collected, your R3 session will close like this:

```
Entering r3close.
Leaving r3close.
```

Processing Oracle directory entries will look much like this:

```
ACCESS ENGINE: Entering ydedopn
ACCESS ENGINE: Using a utility connection for directory open
ACCESS ENGINE: Entering dbiopen
ORACLE: oropen()
ORACLE: ordesca()
ORACLE: ordesc()
ORACLE: DESCRIBE on ORACLE_0
ORACLE: ordtfmt()
ACCESS ENGINE: Entering ydedrd
ACCESS ENGINE: Table returned is ALL_OBJECTS
ACCESS ENGINE:  Exiting ydedrd with rc=0x00000000
```

The above message will repeat until all of the Oracle tables have been listed. Once that happens, your trace would show these calls:

```
ACCESS ENGINE: Entering dbiclose
ORACLE: orclose()
```

If you don't need to do a search for RDBMS information, clearing the libname with a statement like this will make the dictionary run a lot faster: `Libname clear` *`rdbms_name`* `;`

D. Getvar macro to do a data dump of 50 observations for any variable meeting the search critiera. Very useful when you don't quite know the name of the variable but you do have a good idea of what should be in it.

This is a fairly complex macro even with SAS® Enterprise Guide being used to set the search parameters. That's certainly not the only way we could set them. We could set them in the editor before running the code, use SCL or even a .Net interface. The advantage of SAS® Enterprise Guide is staying in one environment and the ease of building basic input screens.

What we are doing behind the scene is to use SAS code to build other SAS code. Here we create three filters, search_lib (the SAS library or libraries to search), search_mem (further restriction of the members) and search_str (restrict variable names). We are also restricting the number of variables that can be dumped so that the system doesn't hang. Something many people have considered helpful is to put the criteria selected into the title whether you got anything back or not.

For testing purposes, we used these options when we ran this: MPRINT, MLOGIC and SYMBOLGEN. MLOGIC shows the flow of the macro statements being executed. MPRINT shows any regular SAS code being executed and SYMBOLGEN shows the value of any macro variables.

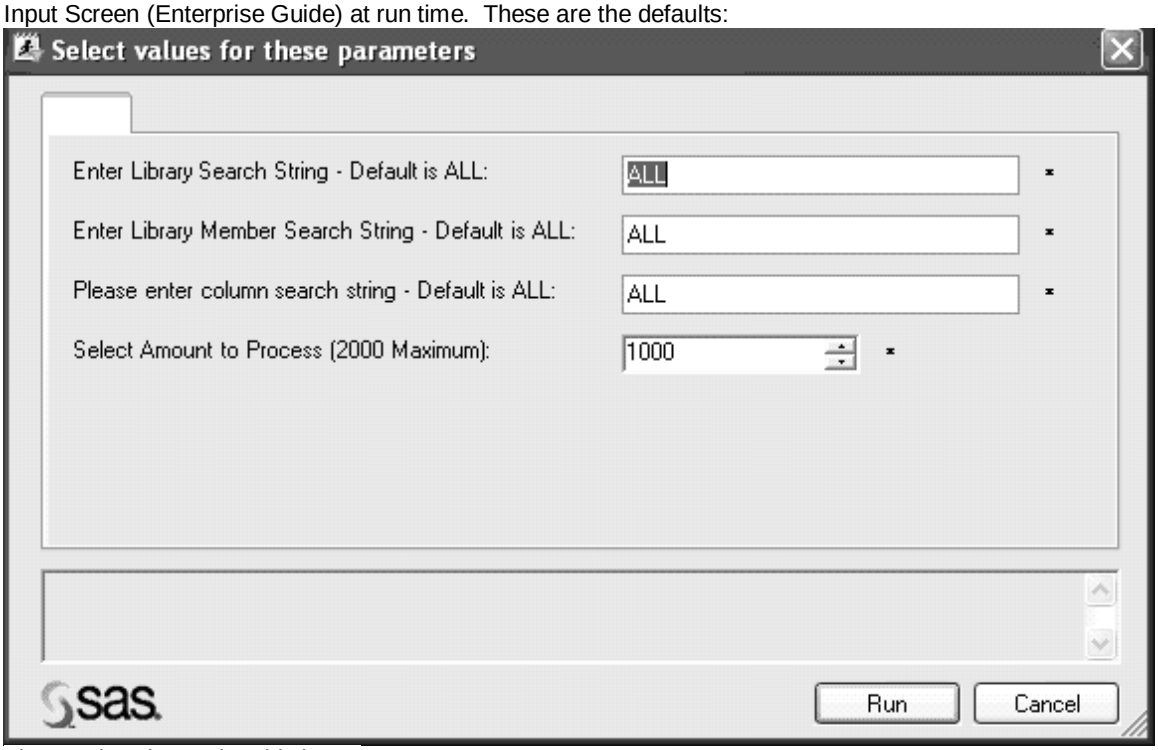
Code in its entirety including comments:

```
Options MPRINT MLOGIC SYMBOLGEN;
%macro GETVAR(endcount);
              TITLE ' ';TITLE2 ' ';TITLE3 ' ';
       %if %eval(&endcount) gt %eval(&maxreturn_count)
              %then %let endcount = &maxreturn_count;
       %IF &ENDCOUNT EQ 0 %THEN %DO;
              DATA _NULL_;
                     FILE PRINT NOTITLES;
                     PUT "**** NOTHING IN THE DIRECTORY MEETS THE CRITERIA";
                     PUT "CRITERIA WAS &WHERE_STATEMENT";
                     PUT "PLEASE TRY AGAIN";
              RUN;

       %END;
       %ELSE %DO;
              %let ctr = 1;
              %do %until (%eval(&ctr) gt %eval(&endcount));
*memname and libname from memname and libname variables in this dataset;
              data _null_;
                     oneob=&ctr;
                     set all_outlist
                     point=oneob;
                     output;
                     memname=trim(memname);
                     name="'"||trim(name)||"'n";
                     call symput('memname',memname);
                     call symput('libname',libname);
                     call symput('fld_name',name);
                     stop;
              run;
%put &memname &libname;

       proc print data=&libname..&memname (obs=50);
              var "&fld_name"n;
       title "FIELD SEARCH FOR &LIBNAME..&MEMNAME, MEMBER #&CTR";
              title2 "LIBRARY SEARCH CRITERIA <&SEARCH_LIB>, MEMBER SEARCH
       CRITERIA <&SEARCH_MEM>";
       TITLE3 "VARIABLE NAME SEARCH CRITERIA <&SEARCH_STR>";
       RUN;
%LET CTR = %EVAL(&CTR+1);
              %END;
       %END;
```

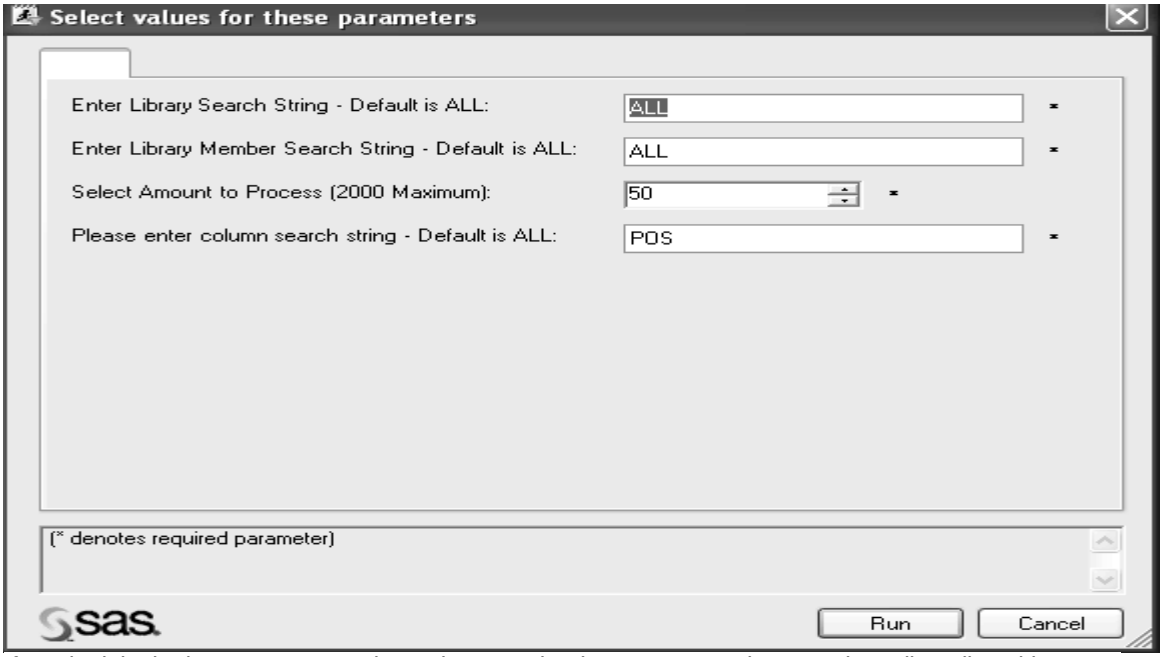
```
%MEND GETVAR;

*Create table outlist for a given library;
      data _NULL_;
            ATTRIB       LIBCRITERIA FORMAT=$50.
                         MEMCRITERIA FORMAT=$50.
                         STRCRITERIA FORMAT=$50.
                         where_statement format=$200.;
                         searchlib1=UPCASE(trim("&search_lib"));
                  if searchlib1 = 'ALL' THEN DO;
                         libcriteria = '';
                         LIBCRIT_COUNT=0;
                  END;
                  else DO;
                         libcriteria = "UPCASE(libname) like "
                         ||"'%"||searchlib1||"%'";
                         LIBCRIT_COUNT=1;
                  END;
                  searchmem1=UPCASE(trim("&search_mem"));
                  if searchmem1 = 'ALL' THEN DO;
                         memcriteria = '';
                         MEMCRIT_COUNT=0;
                  END;
                  else DO;
                  memcriteria = "UPCASE(memname) like " ||
                  "'%"||searchmem1||"%'";
                         MEMCRIT_COUNT=1;
                  END;
                  searchstr1=UPCASE(trim("&search_str"));
                  if searchstr1 = 'ALL' THEN DO;
                         strcriteria = '';
                         STRCRIT_COUNT=0;
                  END;
                  else DO;
                  strcriteria = "UPCASE(name) like " ||
                  "'%"||searchstr1||"%'";
                         STRCRIT_COUNT=1;
                  END;
                  CRIT_COUNT=LIBCRIT_COUNT+MEMCRIT_COUNT+STRCRIT_COUNT;
                  if CRIT_COUNT = 0  then where_statement = ' ';
                         else IF CRIT_COUNT = 1 THEN WHERE_STATEMENT =
                         'AND ' || TRIM(LIBCRITERIA) || TRIM(MEMCRITERIA) ||
                         TRIM(STRCRITERIA) ;
                  ELSE IF CRIT_COUNT = 3 THEN WHERE_STATEMENT =
                         'AND ' || trim(LIBCRITERIA) || ' AND ' ||
                         trim(MEMCRITERIA) || ' AND ' ||trim(STRCRITERIA);
                  ELSE IF CRIT_COUNT = 2 THEN DO;
                         IF LIBCRIT_COUNT = 1 AND MEMCRIT_COUNT = 1 THEN
                         WHERE_STATEMENT = 'AND ' || trim(LIBCRITERIA) |
                         | ' AND ' || trim(MEMCRITERIA);
                  IF LIBCRIT_COUNT = 1 AND STRCRIT_COUNT = 1 THEN
                         WHERE_STATEMENT = 'AND ' || trim(LIBCRITERIA) || '
                         AND ' || trim(STRCRITERIA);
                  IF MEMCRIT_COUNT = 1 AND STRCRIT_COUNT = 1 THEN
                         WHERE_STATEMENT = 'AND ' || trim(MEMCRITERIA) || '
                         AND ' || trim(STRCRITERIA);
                  END;
            call symput('WHERE_STATEMENT',WHERE_STATEMENT);
      run;
      proc sql;
            create table ALL_outlist as
            SELECT *
            FROM
```

```
            (
            select *

            from dictionary.COLUMNS

            where memtype = 'DATA'
            &WHERE_STATEMENT
            )

            ORDER BY LIBNAME, MEMNAME, name
            ;
      QUIT;
      RUN;
%GETVAR(&sqlobs);
```

Input Screen (Enterprise Guide) at run time. These are the defaults:



The results when using this input:



If you look in the log, you can see the code created. First we create what goes into all_outlist table. Note the WHERE statement equates to 'AND UPCASE(NAME) like '%POS%':

```
127        proc sql;
128        create table ALL_outlist as
129        SELECT *
130        FROM
131        (
132        select *
133
134        from dictionary.COLUMNS
135
136        where memtype = 'DATA'
137        &WHERE_STATEMENT
SYMBOLGEN:  Macro variable WHERE_STATEMENT resolves to AND   UPCASE(name) like
'%POS%'

138        )
139
140        ORDER BY LIBNAME, MEMNAME, name
141        ;
NOTE: Table WORK.ALL_OUTLIST created, with 7 rows and 18 columns.

142        QUIT;
```

We got 7 rows out of this so we will execute the code following 7 times. Here is the log for the first time:

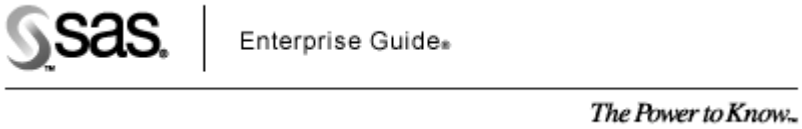
```
MLOGIC(GETMEMB):  Beginning execution.
SYMBOLGEN:  Macro variable SQLOBS resolves to 7
MLOGIC(GETMEMB):  Parameter ENDCOUNT has value 7
MPRINT(GETMEMB):   TITLE ' ';
MPRINT(GETMEMB):  TITLE2 ' ';
MPRINT(GETMEMB):  TITLE3 ' ';
SYMBOLGEN:  Macro variable ENDCOUNT resolves to 7
SYMBOLGEN:  Macro variable MAXRETURN_COUNT resolves to 1000
MLOGIC(GETMEMB):  %IF condition %eval(&endcount) gt
%eval(&maxreturn_count) is FALSE
SYMBOLGEN:  Macro variable ENDCOUNT resolves to 7
```

```
MLOGIC(GETMEMB):  %IF condition &ENDCOUNT EQ 0 is FALSE
MLOGIC(GETMEMB):  %LET (variable name is CTR)
MLOGIC(GETMEMB):  %DO %UNTIL(%eval(&ctr) gt %eval(&endcount)) loop
beginning.
MPRINT(GETMEMB):   *memname and libname from memname and libname
variables in this dataset;
MPRINT(GETMEMB):   data _null_;
SYMBOLGEN:  Macro variable CTR resolves to 1
MPRINT(GETMEMB):   oneob=1;
MPRINT(GETMEMB):   set all_outlist point=oneob;
MPRINT(GETMEMB):   output;
MPRINT(GETMEMB):   memname=trim(memname);
MPRINT(GETMEMB):   name=trim(name);
MPRINT(GETMEMB):   call symput('memname',memname);
MPRINT(GETMEMB):   call symput('libname',libname);
MPRINT(GETMEMB):   call symput('fld_name',name);
MPRINT(GETMEMB):   stop;
MPRINT(GETMEMB):   run;
```

And here the proc print we built:

```
MPRINT(GETMEMB):   proc print data=SASPERS.BASEBALL (obs=50);
SYMBOLGEN:  Macro variable FLD_NAME resolves to 'position'n
MPRINT(GETMEMB):   var 'position'n ;
SYMBOLGEN:  Macro variable LIBNAME resolves to LOCAL
SYMBOLGEN:  Macro variable MEMNAME resolves to BASEBALL
SYMBOLGEN:  Macro variable CTR resolves to 1
MPRINT(GETMEMB):   title "FIELD SEARCH FOR LOCAL   .BASEBALL
, MEMBER #1";
SYMBOLGEN:  Macro variable SEARCH_LIB resolves to ALL
SYMBOLGEN:  Some characters in the above value which were subject to macro
quoting have been
           unquoted for printing.
SYMBOLGEN:  Macro variable SEARCH_MEM resolves to ALL
MPRINT(GETMEMB):   title2 "LIBRARY SEARCH CRITERIA ALL, MEMBER SEARCH CRITERIA
ALL";
SYMBOLGEN:  Macro variable SEARCH_STRING resolves to POS
MPRINT(GETMEMB):   TITLE3 "VARIABLE NAME SEARCH CRITERIA POS";
MPRINT(GETMEMB):   RUN;
```

Report output:



FIELD SEARCH FOR LOCAL .BASEBALL , MEMBER #1
LIBRARY SEARCH CRITERIA ALL, MEMBER SEARCH CRITERIA ALL
VARIABLE NAME SEARCH CRITERIA POS

Obs	position
1	1O
2	C
3	UT
4	3S
5	CF
6	C
7	2B
8	RF

Step by step:

1. **`name="'"||trim(name)||"'n";`** Enclosing a possible variable name in single quotes and "n" lets you use that variable even if there are embedded spaces or special characters.
2. **`searchlib1=UPCASE(trim("&search_lib"));`** put whichever library we're looking for in upper case and trim trailing blacks.
3. **`if searchlib1 = 'ALL' THEN DO; libcriteria = '';LIBCRIT_COUNT=0;END;`** We will generate no library criterion if the default value 'ALL' was chosen. The criteria count will be 0. We will use this variable later when generating the WHERE statement.
4. **`else do; libcriteria = "UPCASE(libname) like " ||"'%"||searchlib1||"%'";`**
`LIBCRIT_COUNT=1;END;` If it isn't the default value 'ALL' then build the statement with the value entered. Concatenate UPCASE statement with search string. UPCASE will evaluate the search string regardless of case. **`" ||"'%"||searchlib1||"%'" "`** puts the search string into a wildcard statement so that any library containing the search string as part of its name will be searched.
5. **`memcriteria = "UPCASE(memname) like " || "'%"||searchmem1||"%'";`**
Similar statement to generate SQL criterion for library member if an actual value was entered. **`"'%"||searchmem1||"%'"`** puts the search string into a wildcard statement so that any member containing the search string as part of its name will be searched.
6. **`strcriteria = "UPCASE(name) like " || "'%"||searchstr1||"%'";`**
Similar statement to generate SQL criterion for variable name if an actual value was entered. **`"'%"||searchstr1||"%'"`** puts the search string into a wildcard statement so that any variable containing the search string as part of its name will be found.
7. **`CRIT_COUNT=LIBCRIT_COUNT+MEMCRIT_COUNT+STRCRIT_COUNT;`** Add all of the criteria count together.
8. **`if CRIT_COUNT = 0  then where_statement = ' ';`**
`else IF CRIT_COUNT = 1 THEN WHERE_STATEMENT =`

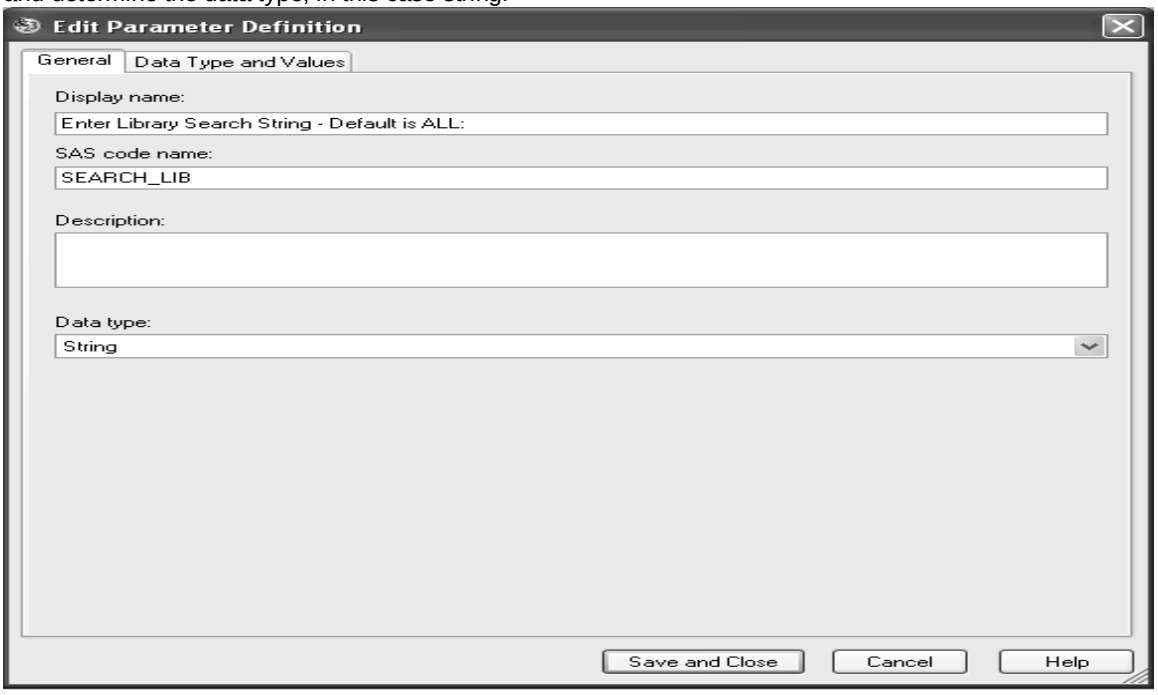
'AND ' || TRIM(LIBCRITERIA) || TRIM(MEMCRITERIA) || TRIM(STRCRITERIA) ; WHERE statement will generate correctly if just one of these are present. The TRIM function will remove any trailing blanks.

9. **where memtype = 'DATA' &WHERE_STATEMENT** WHERE statement will either be blank (no restriction) or have criteria for any combination of the library, member or variable name.
10. **&search_lib, &search_mem and &search_str.** These are set using Parameters/Prompts Manager in Enterprise Guide but may be set in the code with %let statements or even %input if using a version of SAS that allows this.
11. **where memtype = 'DATA' &WHERE_STATEMENT** WHERE statement will either be blank (no restriction) or have criteria for any combination of the library, member or variable name.
12. **&search_lib, &search_mem and &search_str.** These are set using Parameters/Prompts Manager in Enterprise Guide but may be set in the code with %let statements or even %input if using a version of SAS that allows this.

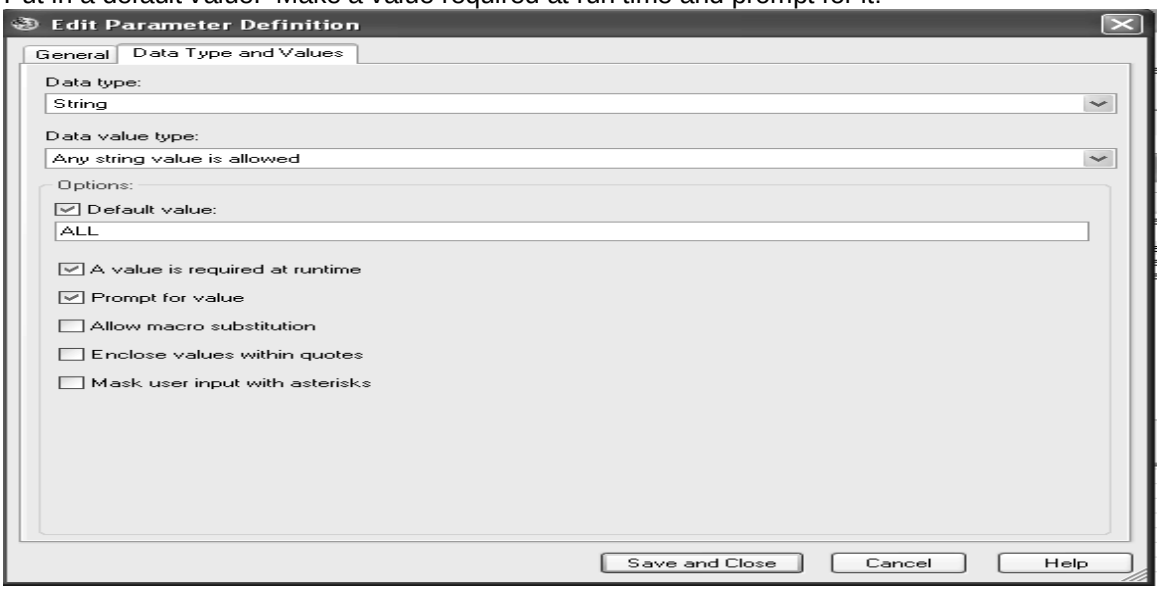
Parameters/Prompts Manager setup

1. Display name determines what the user prompt will look like.
2. SAS code name is the name of the macro variable that will be set. This must agree with the macro variable in your code
3. Data type is the format of the variable. Even though all macro variables are character type, choosing another type such as integer will make SAS attempt to treat it as such with varying degrees of success
4. Data value type – can allow any string value or a numeric range as shown above. Can also restrict values to those in a list.
5. Default Value – assigns a default value if selected. Here it is ALL
6. A value is required at runtime – check so that a new value may be entered every time
7. Prompt for value – bring up the input screen every time the code runs
8. To connect the parameters to the code in SAS® Enterprise Guide, right-click on the code icon and add them to the parameters.

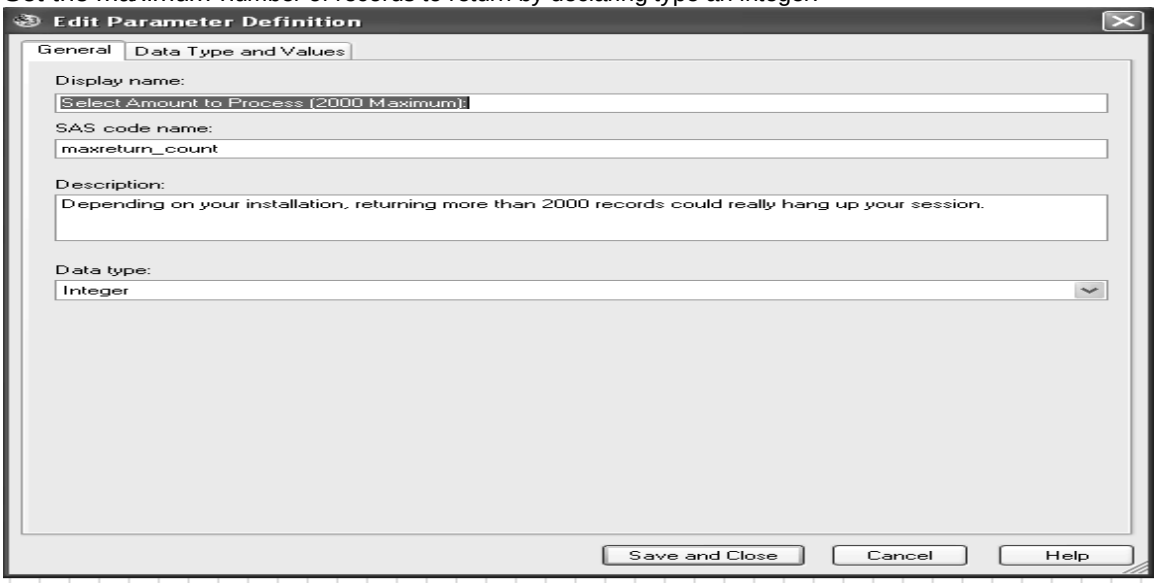
Parameter set up for one of the search strings – Name parameter, decide how the prompt will be displayed and determine the data type, in this case string.



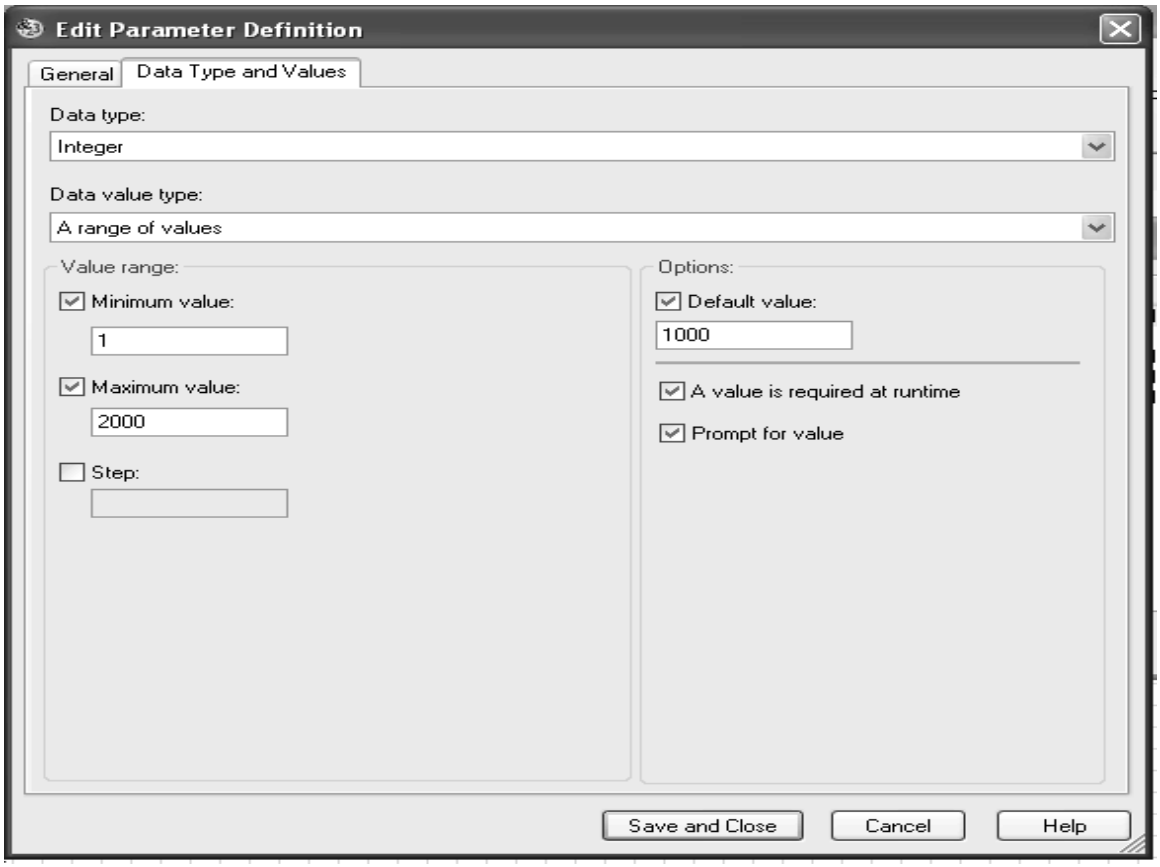
Put in a default value. Make a value required at run time and prompt for it:



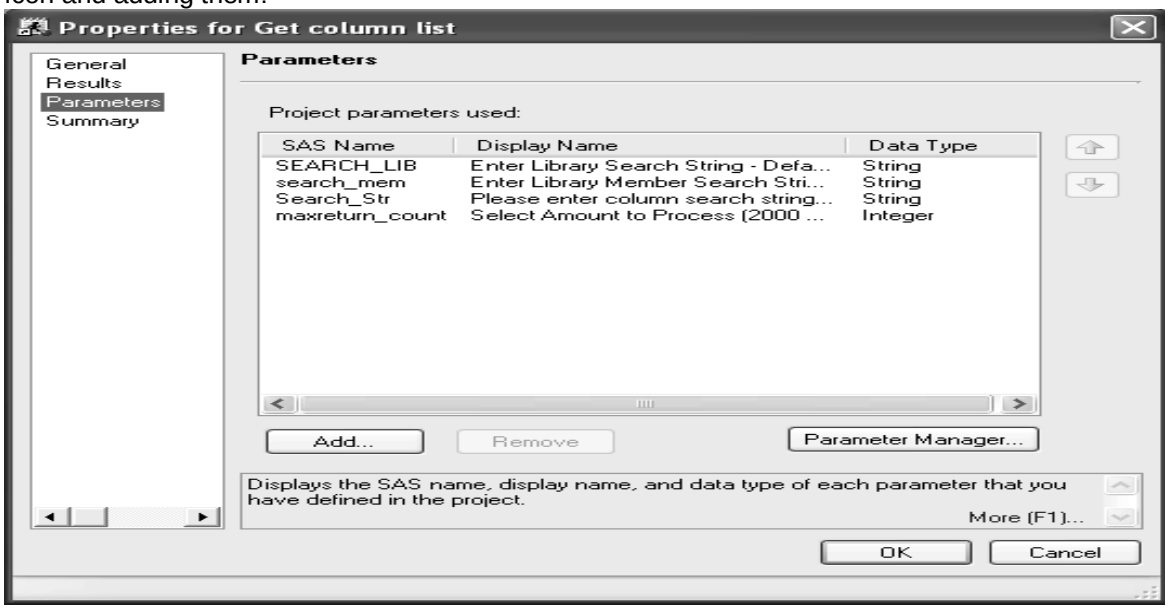
Set the maximum number of records to return by declaring type an integer:



Minimum value will be 1 and maximum 2:



Add these parameters for your Get column list program by right-clicking on the program icon and adding them:



CONCLUSION:

The reader should be able to use these macros 'as-is' for any installation running SAS 8 or later. Some of the code in this paper may change with future releases of SAS. Some may even become obsolete altogether. But the basic concepts should stand – use of the metadata to get some idea of what is actually in a variable as opposed to its name, variables with the same name but different domains, variables with different names but the same domain, jobs running with missing data or harder to catch, old data. These macros should help with all of those issues.

RECOMMENDED READING

Online SAS Manual – SAS Institute, particularly the macro section for more ideas.

ACKNOWLEDGEMENTS:

Thanks to Joe and Paul Butkovich for your encouragement and support.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. The author is often on the road but can be contacted at

Patricia Hettinger
Email: patricia_hettinger@att.net
Phone: 331-462-2142