Paper 078-29

Dynamic SAS® Programming Techniques, or How NOT to Create Job Security

Steven Beakley, Lucid Analytics Corporation
Suzanne McCoy, Lucid Analytics Corporation

INTRODUCTION

Many SAS programmers, particularly consultants, joke about creating job security by creating SAS programs and applications that require long, ongoing maintenance. In all seriousness, though, most of us would prefer not to have the "security" of mundane, ongoing maintenance tasks, preferring instead to move on to new development opportunities. However, sometimes this is not as easy as we would like; requirements change, data structures change, users change. This often leaves consultants or developers constantly making minor modifications to "production" applications.

While certain ongoing maintenance cannot be avoided, there are techniques that can be utilized to help limit it. This paper will describe a number of these techniques, including the use of SAS metadata and dictionary tables, simple and more advanced user input before and during program execution, plus dynamic library and filename referencing. The use of these and other techniques should make your programming time more effective for your clients or employers and more rewarding for you.

METADATA IS YOUR FRIEND

Metadata is, simply put, data about data. Each and every SAS session has available to it a large variety of information about the data sources that you are using, both permanent and temporary. The information is housed in the SASHELP library in several views, as listed in Table I. These are known as "dictionary tables".

Table I: SASHELP._ALL_ (subset for dictionary tables only)

MLVIEW	VCATALG	VCOLUMN	VEXTFL	VINDEX	VMACRO
VMEMBER	VOPTION	VSACCES	VSCATLG	VSLIB	VSTABLE
VSTABVW	VSTYLE	VSVIEW	VTABLE	VTITLE	VVIEW

The beauty of these views lies in the fact that they are automatically updated by the SAS System while your session executes, so they always reflect the most current status. Each of them could be useful to a developer in different situations, but there are several that will be specifically addressed here.

VSLIB and VMEMBER identify all libraries that are assigned in the current session. They also provide the location of these libraries (they differ in that VSLIB contains abbreviated information from VMEMBER). Similarly, VEXTFL identifies all files that have filename assignments and provides their locations. VTABLE and VSTABLE provide table-specific information, while VCOLUMN provides information about the columns in each and every table – names, labels, formats, types, and so on. Finally, VOPTION, VMACRO, and VINDEX provide information about option settings, macros defined, and indexes respectively. Take some time to review the contents of these views; even those familiar with them may be surprised at what a variety of valuable information is available.

Hopefully it will be very apparent that since dictionary tables are updated automatically, programs can be strategically written to utilize them and avoid a great deal of hard-coding and modification later. The following sections describe a couple of development scenarios using some of dictionary tables described.

EXAMPLE 1: A MOVING TARGET

This scenario is one that the authors, and undoubtedly many other developers, have experienced over and over. Overlapping deadlines between departments mean that data structures are not finalized before programmers must begin coding reports and applications that use the data. This means that variable names, formats, and even their locations can change while programming is ongoing. This of course, can be very frustrating. But, thanks to a little foresight and planning, use of the SAS dictionary tables can alleviate this frustration.

Before: Hard-coded Programming

```
Libname a 'c:\sugi\a';

data class;
  set a.class;
  where age <= 12;
run;
```

A simple program, but suddenly, on June 2, it doesn't run anymore. The dataset is no longer available, though the owner swears it is. As it turns out, the dataset has been moved, and is moved subsequent to that as well. Therefore, constant modification to the code is required.

Here is where using the dictionary tables can help out. Assume that the dataset could be in one of four possible SAS libraries assigned at startup. Further assume that the dataset is named CLASS or CLASSES.

After: Dynamic Programming

```
Libname a 'c:\sugi\a';
Libname b 'c:\sugi\b';
Libname c 'c:\sugi\c';
Libname d 'c:\sugi\d';

data _null_;
  set sashelp.vmember
       (keep=libname memname path);
  where upcase(memname) like 'CLASS%' and
  libname not in ('WORK','MAPS','SASHELP');
  call symput('path',left(trim(path)));
run;

libname mylib "&path";
```

The modified program utilized the VMEMBER dictionary table to identify the path of the library that contains the dataset being sought. Then a macro variable is created which contains that path. If a dataset of the same name appears in more than one directory, only the last one found in the search path will be used. Finally, a new libname statement is issued to the path.

Obviously, this example is very basic and serves to illustrate one of the ways to use the dictionary tables. Example 2 is much more complicated, and differs from the first in another significant factor. The SAS dictionary tables that have been referenced thus far are actually views to SQL dictionary tables with similar, but not quite identical, names. This example will illustrate using the dictionary tables through PROC SQL rather than datastep language. Either method can achieve the same result, but do be aware that naming differences exist. If using Proc SQL, libname 'DICTIONARY' is automatically assigned. The members associated with the dictionary libname are show below.

Table II: Proc SQL; select memname from dictionary.tables;

CATALOGS	COLUMNS	EXTFILES	INDEXES	MACROS
MEMBERS	OPTIONS	TABLES	TITLES	VIEWS

Yet another example of using the SQL Dictionary Tables is included as Appendix A.

EXAMPLE 2: DYNAMIC MACRO VARIABLE ASSIGNMENTS

This scenario starts from a SAS/WebAF application screen used for data entry. The values entered on the screen need to be dynamically passed from the data entry screen, though some Java processing, and then be loaded into the master dataset via macro variables. This can be implemented dynamically using dictionary tables so that the code would not have to change if/when variables were added to or removed from the dataset.

Step 1 – determine which variables exist in the work dataset. Create a dataset of the variable names and a macro variable that contains the list of variable names.

```
proc sql noprint noerrorstop;
  create table work.varnames as
  select upcase(name) as name
  from   dictionary.columns
  where  libname=upcase("&lib")
    and  memname=upcase("&dsname")
  order  by name;

  select name
         into :listofvars separated by ' '
  from   work.varnames;
quit;
```

Step 2 – the variables are going to be used multiple ways, so they are made global to the application.

```
%global &listofvars;
```

Step 3 – Create the macro variables that contain the names of the macro variables to be created and populated. Also create a macro variable that contains the number of macro variables created.

```
data _null_;
  set work.varnames end=eof;
  call symput
   (compress('varname'||put(_n_,best.))
   ,compress(name));
  if eof then
   call symput
  ('numofvars',compress(put(_n_,best.)));
run;
```

Step 4 – Populate the macro variables with the value captured from the work dataset underlying the data entry screen. Note that the macro variable has the same name as the data set variable.

```
data _null_;
  set &lib..&dsname (obs=1);
   %do i=1 %to &numofvars;
     call symput
       ("&&varname&i",&&varname&i);
   %end;
run;
```

YOUR INPUT IS DESIRED

Another source of numerous modifications to programs, and a significant security threat, is the hardcoding of usernames and passwords into program code. Oracle and other DBMS libnames, as well as SPDS libnames, require a username and password before connecting to the data source. An option on the libname statement, DBPROMPT, can be assigned with a value of YES to prompt the user to enter the username and password values. However, a limitation of this is when the session is not an interactive session; for example, a remote session on a Unix server established through SAS/CONNECT from a Windows workstation (a session initiated through SAS/IntrNet or WebAF would also behave similarly). Of course, a user input application can be developed using SAS/AF, SAS/IntrNet, or AppDev Studio, but what if only Base SAS is available? Fortunately, the SAS Macro Facility provides a solution with %WINDOW and %DISPLAY. The example presented in Appendix B will illustrate their use. Also notice that it includes yet another example of using dictionary tables.

COMPILE SOURCE CODE – PREPARATION FOR PRODUCTION DEPLOYMENT

In many 'production' environments, only compiled source code is deployed. The methodology shown here was used to compile SAS programs from a dedicated directory into a SAS catalog. This example is from a Windows NT platform.

```
*dir is the location of the programs;
%let dir='dir "pathto\macroSource';

*avoid quoting issues that will come into play on filename statements;
%let srcdir = pathto\macroSource\;

*define location and options for the compiled code;
libname macrosrc 'pathto\macroSource';
options mstored sasmstore=macrosrc;
/*--------------------------------------
Requirements for successful use of this methodology:
1.  Programs in the directory must be named "program_name.sas"
2.  The macro within the program must have the same name as the SAS program
3.  The macro definition line must have a option to store (e.g., %macro program_name
/store;
4.  Program names cannot contain blanks
--------------------------------------*/
```

```
filename programs pipe &dir; *pipe executes when the filename is used;

*create a dataset from the directory listing produced by the pipe;
data dirlist;
 infile programs missover pad;
 input name $100.;
run;

*keep only items in the directory listing that end with '.sas';
data programs(drop=name);
 set dirlist;
 if scan(name,-1,'.') = 'sas';
 program_name=scan(name,-1,' '); *no blanks allowed in the names;
run;

*turn the program names into macro variables;
data _null_;
  set programs end=eof;
  call symput(compress('prog'||put(_n_,best.)),program_name);
  if eof then call symput('nobs',compress(put(_n_,best.)));  *create counter variable
&nobs;
run;


%macro compile_macros;
  %do i=1 %to &nobs;
    filename thisfl&i "&srcdir&&prog&i";
    %include thisfl&i;
  %end;
%mend compile_macros;
%compile_macros; *execute the compile;
```

CONCLUSION

There never seems to be a lack of SAS challenges for developers on any given project. The way to grow and broaden experience is by freeing up time that one might otherwise spend performing ongoing maintenance or other repetitive tasks. The goal of this paper and presentation has been to expose developers of all levels to techniques and ideas for more dynamic, less “hands-on” programming, creating applications and programs that can be turned over to less experienced programmers or even non-programmers for future maintenance. Other, non-programmatic tools can aid in achieving this, like spreadsheets or text files that serve as “drivers” or “control files” for SAS applications. The primary motivator to a developer should be to provide better and better applications, thereby creating the RIGHT kind of job security!

AUTHOR INFORMATION

Steve Beakley is a SAS V8 Certified Professional with over 15 years SAS experience. His experience has included implementation, development and maintenance of SAS-related projects in a number of industries including airlines, medical software and services, energy, and retail. He has served as an officer for the Kansas City Area SAS Users Group, has twice been a SUGI presenter, and was an invited presenter at a SAS Executive Conference. Steve can be reached at sbeakley@lucid-analytics.com.

Suzanne McCoy is a SAS Certified Advanced Programmer with more than 17 years of SAS experience. She is a co-owner and principle of Lucid Analytics Corp., a SAS Silver Alliance Partner and the founder and contact person for the Wilmington, NC area SAS Users Group. Suzanne can be reached via email at smccoy@lucid-analytics.com.

Appendix A: Think like a Developer instead of a Programmer!

```
*Scenario: Spreadsheets are imported into SAS datasets;
*          every month.  The extra spaces need to be;
*          removed from the character variables;

libname here './';

 data here.data1;
  set here.data1;
  manager_1=left(compbl(manager_1));
  manager_2=left(compbl(manager_2));
 run;

 data here.data2;
  set here.data2;
  manager_1=left(compbl(manager_1));
  manager_2=left(compbl(manager_2));
 run;


-------------------------------------------------------
*The next month they add a new dataset and a new variable;
*A programmer goes in and manually maintains the code;
*Here is the revised program, if you're thinking like a Programmer;

libname here './';

 data here.data1;
  set here.data1;
  manager_1=left(compbl(manager_1));
  manager_2=left(compbl(manager_2));
  system   =left(compbl(system));
 run;

 data here.data2;
  set here.data2;
  manager_1=left(compbl(manager_1));
  manager_2=left(compbl(manager_2));
  system   =left(compbl(system));
 run;

 data here.data3;
  set here.data3;
  manager_1=left(compbl(manager_1));
  manager_2=left(compbl(manager_2));
  system   =left(compbl(system));
 run;



-------------------------------------------------------
*Someone finally gets the idea to make it a macro;
*and make it do all character variables;
*You're starting to think like a Developer now.;

libname here './';

%macro remove_extra_blanks(data=);

  proc sql noprint;
   *determine all character variable names in the dataset;
   create table charvars as
   select name
   from   dictionary.columns
```

```
   where  libname="HERE"
     and  memname=upcase("&data")
     and  type='char'
   ;
  quit;

  *turn the variable names into macro variables;
  data _null_;
   set charvars end=eof;
   call symput(compress('var'||put(_n_,best.)),name);
   if eof then call symput('nobs',put(_n_,best.));
  run;

  *drive the variable names via a macro do loop;
  data here.&data;
   set here.&data;
   %do i=1 %to &nobs;
     &&var&i=left(compbl(&&var&i));
   %end;
  run;
%mend remove_extra_blanks;

%remove_extra_blanks(data=data1);
%remove_extra_blanks(data=data2);
%remove_extra_blanks(data=data3);




-------------------------------------------------------
*End of maintenance??

*NO WAY!  Next month there is yet another dataset;
*A programmer still has to go and add a macro call;


%remove_extra_blanks(data=data4);



-------------------------------------------------------

*How do you totally eliminate the maintenance on this;
*Very simply.  Use the SAS metadata once again.;
*NOW YOU'RE THINKING LIKE A DEVELOPER!;

libname here './';

%macro remove_extra_blanks;

  *determine the table names;
  proc sql noprint;
   create table datasets as
   select name
   from   dictionary.tables
   where  libname="HERE"
   ;
  quit;

  *turn the table names into macro variables;
  data _null_;
   set datasets end=eof;
   call symput(compress('tab'||put(_n_,best.)),name);
   if eof then call symput('tabnobs',put(_n_,best.));
  run;
```

```
  *drive the table names via a macro do loop;
  %do i=1 %to &tabnobs;
   *determine the character variables in this dataset;
   proc sql noprint;
    create table charvars as
    select name
    from   dictionary.columns
    where  libname="HERE"
      and  memname=upcase("&&tab&i")
      and  type='char'
    ;
   quit;

   data _null_;
    set charvars end=eof;
    call symput(compress('var'||put(_n_,best.)),name);
    if eof then call symput('nobs',put(_n_,best.));
   run;

   data here.&&tab&i;
    set here.&&tab&i;
    %do j=1 %to &nobs;
      &&var&j=left(compbl(&&var&j));
    %end;
   run;
  %end;
%mend remove_extra_blanks;
%remove_extra_blanks;
```

Appendix B: %WINDOW/%DISPLAY Example

```
************************************************************
*                                                          *
*  LOGON.SAS                                               *
*                                                          *
*  This program prompts a user upon starting a SAS session *
*  for username and password for connections to DB2,        *
*  Oracle, and SPDS.  It then makes the remote connection  *
*  to Unix and, if necessary, an MVS host (DB2), passes    *
*  the usernames and passwords supplied, and sets up the   *
*  libnames.  Once return is controlled to the local       *
*  session, the libraries that have been defined remotely  *
*  are defined in the local session as well.               *
*                                                          *
*  Created 2/18/2003 by Steven Beakley.                    *
*                                                          *
************************************************************;

*  Turn off source so that no passwords are passed to the log.;

options nosource;

*  Initialize all the username/passwords to blank before starting.;

%let unxusr = ;
%let unxpwd = ;
%let hstusr = ;
%let hstpwd = ;
%let orausr = ;
%let orapwd = ;
%let spdusr = ;
%let spdpwd = ;

*  Display the window that prompts for the various usernames and
   passwords.  If the user provides any of the inputs they are
   passed as macro variables to the remainder of the program.;

%window welcome color=white
           #2  @26 'Welcome to SAS/Enterprise Miner!' color=green attr=rev_video
           #5  @10 "Enter your Unix username and password here: (REQUIRED)" color=red
           #7  @22 "UNIX:" color=red +2 unxusr 8 color=red required=yes attr=rev_video
autoskip=yes
                +2 unxpwd 24 display=no required=yes color=red attr=rev_video
           #11 @12 "If you need to establish a connection to the MVS host,"
           #12 @23 "enter your username and password here:"
           #14 @22 "HOST:" color=blue +2 hstusr 8 color=blue attr=rev_video autoskip=yes
                +2 hstpwd 24 display=no color=blue attr=rev_video
           #18 @15 "If you need to establish a connection to Oracle,"
           #19 @23 "enter your username and password here:"
           #21 @20 "ORACLE:" color=blue +2 orausr 8 color=blue attr=rev_video autoskip=yes
                +2 orapwd 24 display=no color=blue attr=rev_video
           #25 @21 "If you need to establish a connection to SPDS,"
           #26 @23 "enter your username and password here:"
           #28 @22 "SPDS:" color=blue +2 spdusr 8 color=blue attr=rev_video autoskip=yes
                +2 spdpwd 24 display=no color=blue attr=rev_video
           #31 @31 'Press ENTER to continue.' color=green attr=rev_video;
%display welcome;

*  Remote connection established to the Unix box.;

%let sasunix=remote.unixserver.com;
options comamid=TCP remote=sasunix;
filename rlink 'c:\program files\sas institute\sas\v8\connect\saslink\tcpsasunix.scr';
signon sasunix;
```

```
*  Create the needed macro variables on the remote session.;

%macro passuser;
%syslput unxusr=&unxusr;
%syslput unxpwd=&unxpwd;
%syslput hstusr=&hstusr;
%syslput hstpwd=&hstpwd;
%syslput orausr=&orausr;
%syslput orapwd=&orapwd;
%syslput spdusr=&spdusr;
%syslput spdpwd=&spdpwd;
%mend passuser;
%passuser;

rsubmit sasunix;

%*  Turn off source so that no passwords are passed
    to the log from the Unix session.;

options nosource;

%macro conx;

%*  An MVS host session is initiated if requested, the DB2 library is
    defined remotely, and then passed back to the Unix session.;

%if "&hstusr" ^= "" %then %do;
signon mvs.server noscript user=&hstusr PW="&hstpwd";
rsubmit mvs.server;

libname mvsdb2 db2 ssid=mvsssid schema=db2schema;

endrsubmit;

libname mvsdb2 slibref=mvsdb2 server=mvs.server;
%end;

%*  The SPDS library is defined if requested.;

%if "&spdusr" ^= "" %then %do;
libname dw sasspds "_prg" host="sasunix" serv="9999" user="&spdusr" passwd="&spdpwd"
unixdomain=yes netcomp=no;
%end;

%*  The Oracle library is defined if requested.;

%if "&orausr" ^= "" %then %do;
libname unxora oracle user="&orausr" pass="&orapwd" path="unxora" schema="orschema";
%end;
%mend conx;
%conx;

*  Clear macro variables with the passwords defined.;

%let unxpwd = ;
%let hstpwd = ;
%let orapwd = ;
%let spdpwd = ;

*  Turn on source for subsequent user requests.;

options source;

endrsubmit;
```

```
*  Establish local references to saswork, sasuser, and sashelp defined remotely (call them
rmtwork and rmtuser).;

libname rmtwork slibref=work server=sasux101;
libname rmtuser slibref=sasuser server=sasux101;
libname rmthelp slibref=sashelp server=sasux101;

*  Establish local references to each of the other libraries defined remotely.;

%macro loclibs;

data _null_;
  set rmthelp.vslib end=eof;
  where upcase(libname) ^in
('MAPS','SASHELP','SASUSER','WORK','RMTWORK','RMTUSER','RMTHELP','SASCATCA','SASADMIN','RPOSM
GR');
  num+1;
  call symput ('lib'||left(trim(num)),left(trim(libname)));
  if eof then call symput('num',num);
run;

%do i=1 %to &num;
libname &&lib&i slibref=&&lib&i server=sasunix;
%end;

*  Deassign rmthelp, we are done with it now.;

libname rmthelp clear;

%mend loclibs;
%loclibs;

*  Clear macro variables with the passwords defined.;

%let unxpwd = ;
%let hstpwd = ;
%let orapwd = ;
%let spdpwd = ;

*  Turn on source for subsequent user requests.;

options source;

*  Logon.sas complete;
```