

Data Management: Building a Dynamic Application

Arthur L. Carpenter, Data Explorations
Richard O. Smith, Data Explorations

ABSTRACT

You have written a series of interesting and often complex SAS® programs that perform a variety of data entry operations, data checks, exception reporting, statistical analyses, and summary reporting. Since the next study is somewhat similar to the last one, you are planning to build another set of programs based on (cannibalized from) the ones that you just used.

We have been there before, but STOP. Wouldn't you rather build and validate the programs just once?

The solution lies in building and using data dictionaries that act as control files. These control files are in turn used to create a series of SAS macro variables that are available as arrays to each of the various programs. All project, data set, and variable specific information is stored in the macro variables and hence never in the programs themselves. Once implemented all the programs in your application become data independent.

Let them change the data. Let them redefine the project. Your code is ready.

KEYWORDS

Clinical trial, macro variables, &&VAR&I, CALL SYMPUT, data dictionary

INTRODUCTION

At the end of the end of a successful project everyone should be happy with what you have done as a SAS® programmer. If everything went fairly well then, you are pleased, the boss is pleased, and the client is pleased. Of course your job is not really completed. The problem is that you now need to document what you did, create the data dictionaries, and prepare for that next study.

Often this means that you will need to reconstruct what you have done and then modify the existing programs for the upcoming project. Once modified these programs will of course then have to go through the validation process again as well.

Wouldn't you rather build the programs so that they will be ready to go for the next study regardless of the number of data sets, names of variables, and error check specifications. Wouldn't you rather just validate your programs once and not over and over again? Imagine the savings in change control management alone with only one version of each program.

Many studies, including most clinical trials, are very data intensive. Very often there is a large number of distinct SAS® data sets, each with a diverse suite of variables, and the data sets themselves may range from a few to many observations. Just keeping track of these data sets becomes a chore for the data base manager.

Management issues become even more intense when an application is developed that must operate against these data sets. As part of the data management, analysis, and reporting process, numerous SAS® programs are usually written to support the application.

Very often names of data sets, variables, and other data set and project specific information is embedded within these programs. This makes it difficult, time consuming, and expensive to modify the application for another similar project. A dynamic and automated application will overcome these limitations by avoiding any project, data set, or variable dependencies.

How then do we create an application that will work for any number of data sets, with any number of observations, and with any combination of variables? How can the application be written so that it requires little or no recoding when being ported from one project to the next?

Fortunately, although there are major differences between studies, many of the tasks are similar. Most studies require data entry and data validation. Many of the exception and adverse event captures and reports are similar. Since these major events are similar across projects, it ought to be possible to generalize our programs so that they need not be modified for each project.

Indeed it is possible to build the SAS programs so that they are general enough to work for each of your studies. The answer lies in the creation of data dictionaries that can be used as control files to build a series of macro variables, which are in turn used dynamically by the application. The key is to build a structure into your programs that is based on those things that are common to all projects. Some of these commonalities include: project identification, library and folder relationships, data sets with specifiable characteristics, variables within data sets with specifiable characteristics, and variable specific value constraints (Carpenter and Smith, 2002).

CONTROL FILES

The data dictionaries or control files become both the starting point and heart of the control process. This means that the addition of a new data set into the study or a change in a data set's variables may be as simple as changing one control file. Done properly, all of the programs that depend on these control files will require NO modification when the control files are changed. This also implies that implementing a new study is as simple as building a new set of data dictionaries, and, of course, this is something that you would be doing anyway.

The control files can be SAS data sets, Excel tables, or even flat files. Ultimately it is usually easiest to build and maintain these control files as SAS data sets through the use of a SAS/AF® or FRAME application. The variables that will be contained within these control files will depend of course on how you use them. The three control data sets shown below are the primary ones of the several that you may need (for demonstration purposes each has been simplified). They are:

DBDIR	Data set definitions
VRDIR	Variables within data sets
FLDDIR	Data field constraints

Each column or variable in these data sets will become a vector of macro variables with one macro variable for each data value in the data set.

DBDIR

This data set will contain one observation for each data set in the project. In addition to the data set name, variables often include CRF page number, key variables, data set label, other data set specific information such as might be required by the various analysis programs.

OBS	DSN	PAGE	KEYVAR
1	DEMOG	1	SUBJECT
2	MEDHIS	2	SUBJECT MEDHISNO SEQNO
3	PHYSEXAM	3	SUBJECT VISIT REPEATN SEQNO
4	VITALS	4	SUBJECT VISIT SEQNO REPEATN

VRDIR

This data set contains one observation for each variable in each data set. Variables that are in all data sets within the project have ALL as the data set name so that they do not need to be constantly repeated. Although not shown in this example, many other variable specific items such as formats and variable length can also be included in this control file.

OBS	DSN	VAR	PG	LABEL
1	ALL	SUBJECT	\$	Patient number
2	ALL	PTINIT	\$	Patient initials
3	DEMOG	DOB	\$	Date of birth
4	DEMOG	SEX	\$	Sex
5	MEDHIS	MEDHISNO	8	Medical History Number
6	MEDHIS	MHDT	\$	Date of medical history
7	PHYSEXAM	PHDT	\$	Physical exam. date
8	PHYSEXAM	WT	\$	Weight

The data sets DBDIR and VRDIR are used to build the data dictionary and to document the data sets and the variables that they contain.

FLDDIR

The data set FLDDIR identifies data constraints for each data entry field or variable. These constraints can be used to build data exception and error trapping reports.

OBS	DSN	VAR	CHKTYPE	CHKTEXT
1	DEMOG	CENTRE	notmiss	
2	DEMOG	RACE	list	('1', '2', '3')
3	MEDHIS	MHDT	format	date7.

Several different types of checks are possible. Shown here are:

- notmiss the variable may not contain missing values
- list the value must be in the list of values in CHKTEXT
- format the formatted value of the variable (using the format in CHKTEXT) must not be missing. User defined formats are permitted.

BUILDING MACRO VARIABLES

Each of the control files is used to create a series of macro variables. The observations are counted and the observation number becomes a part of the macro variable name. This results in names such as &LIVEDB1, &LIVEDB2, &LIVEDB3, ...

Although the macro language does not have an array statement *per se* this series of macro variables can be used as a vector of values. Effectively this vector becomes a macro array.

Building the list of data sets

In the following example the variable I ❶ counts the observation number. It is then converted to a left justified character variable (II) ❷, which is appended to the name of the macro variable ❸. The macro variables are created using the CALL SYMPUT routine. ❹

```
data _null_;
  set datamgt.dbdir end=eof;
  i+1; ❶
  ii=left(put(i,3.)); ❷
  call symput('livedb' || ❸ii,trim(dsn)); ❹
  call symput('keys' || ii,keyvar); ❹
  if eof then call symput('livecnt',ii); ❺
run;
```

Using this DATA step and the DBDIR data set from above, the second observation yields the macro variable &LIVEDB2 as MEDHIS and the associated key fields are stored in &KEYS2 as SUBJECT MEDHISNO SEQNO. Since there is one observation for each data set in the project, the total number of observations in the data set DBDIR will be the same as the number of data sets in the project and this number is stored in &LIVECNT ❺.

Building the variable list

The information for each individual variable is also read in a similar manner. The data set name is included ❶ as is the name of each variable ❷. Other information that can be transferred into macro variables includes variable labels ❸, formats, length, and variable type ❹.

```
data _null_;
  set datamgt.vrdir end=eof;
  i+1;
  ii=left(put(i,3.));
  call symput('vdsn' || ii,dsn); ❶
  call symput('var' || ii,var); ❷
  call symput('label' || ii,label); ❸
  call symput('vtyp' || ii,vartype); ❹
  if eof then call symput('varcnt',ii);
run;
```

Building the field check list

A similar data step is applied to the data set FLDDIR. Since these macro variables are used less frequently, they are only loaded into the symbol table when they are needed. The step that reads the data set is shown below in the section that also discusses how these macro variables are utilized.

USING &&VAR&I CONSTRUCTS AS MACRO ARRAYS

Within a macro, a macro %DO loop can be used to step through the list of macro variables that contain the names of data sets or variables within a data set. The index for a %DO loop is also a macro variable and it can be used as part of the name of the macro variable that is to be resolved.

As was shown above, the macro variable name is formed by concatenating a number onto the name portion. In the following SYMPUT the value contained in the character variable ii is concatenated onto 'LIVEDB'. The subsequent macro variable will contain the value stored in the data step variable DSN.

```
call symput('livedb' || ii, trim(dsn));
```

Resolving &&var&i

The following macro %DO loop will execute &LIVECNT times.

```
%do i = 1 %to &livedb;
  &&livedb&i
%end;
```

When &LIVECNT is 4 the loop creates the macro variable list of:

```
&LIVEDB1 &LIVEDB2 &LIVEDB3 &LIVEDB4
```

which further resolves to:

```
DEMOG MEDHIS PHYSEXAM VITALS
```

The process of the resolution of macro variables can be viewed in the LOG by using the SYMBOLGEN system option.

Stepping through a list of data sets

We now have the ability to step through a list of data sets. The following macro loops through each data set in the study. The PROC FSEDIT is executed for each data set ❶ using the appropriate customized SCREEN (of the same name) ❷.

```
%do jj = 1 %to &livedb;
  proc fsedit data=livedb.&&livedb&jj ❶
    screen=appls.descrn.&&livedb&jj...screen; ❷
  run;
%end;
```

Notice the use of the three decimal points (dots) ❷. More than one is required as the SAS interpreter will see them as delimiters when they immediately follow a macro variable.

Checking for duplicate observations

In the macro %DUPCHK, which follows, each data set is to be checked for duplicate observations by using the appropriate list of BY variables. Again we step through the data sets ❸. The BY variable list for the data set &&livedb&jj will be stored in &&keys&jj ❹. The macro %NW ❺ (Carpenter, 1998, p.193 the macro %COUNT is similar to %NW) counts the number of variables in the list which is then stored in &KEYCNT. The data can be sorted using the key variables ❻, and FIRST and LAST processing can also be used ❼ to determine if there are duplicate observations.

```
%macro chkdup;
%do jj = 1 %to &livedb; ❸
  %nw(&&keys&jj ❹, wordvar=key, wordcnt=keycnt) ❺
  * Sort the data sets for the
```

```

* key variables;
proc sort data=live.&&livedb&jj out=base;
  by &&keys&jj; ❸
  run;

* Check for duplicate key values;
%let dupp = 0;
data dupp; set base;
  by &&keys&jj;
  * determine if this is a dup obs;
  if not (first.&&key&keycnt and last.&&key&keycnt); ❹
  call symput('dupp','1');
  run;

%if &dupp %then %do;
  proc print data=dupp;
    id &&keys&jj;
    title1 "&&livedb&jj";
    title2 "DUPLICATE KEYFIELDS in LIVE Data set";
  run;
%end;
%end; * end the DSN do loop;
%mend chkeddup;

```

Coordinating two macro variable lists

Sometimes the loop through the list of data sets will also require a second loop to pass through the variable list appropriate for each data set. This requires a double %DO loop with coordination between the two. In the following example we build a series of zero observation data sets that will be used as prototypes for the analysis data sets. For each data set the list of variables in the KEEP= option, the LENGTH statement, and the LABEL statement is built dynamically. Notice in each these statements the outer loop (&JJ) increments once for each data set while the inner loop (&KK) cycles through all possible combinations of data sets and variables. A macro %IF statement selects the appropriate variables for a given data set.

```

%macro bldlive;
%do jj = 1 %to &livecnt;
  * One data step for each data set;
  data livedb.&&livedb&jj(keep= ❶
    %* Build the var list to keep for this DB;
    %do kk = 1 %to &varcnt; ❷
      %if &&livedb&jj=&&vdsn&kk
        or &&vdsn&kk=ALL %then &&var&kk;❸
    %end;
  );
  * Use length to define variable attributes;
  length ❹
    %do kk = 1 %to &varcnt;
      %if &&livedb&jj=&&vdsn&kk or
        &&vdsn&kk=ALL %then &&var&kk &&vtyp&kk;
    %end;
  ;
  * Define the variable labels;

```

```

label ⑤
  %do kk = 1 %to &varcnt;
    %if &&livedb&jj=&&vdsn&kk
      or &&vdsn&kk=ALL
      %then &&var&kk = "&&label&kk";
    %end;
  ;
stop;
run;
%end;

```

Within the DATA step, %DO loops are used to build a KEEP= data set option ❶, LENGTH statement ❷, and a LABEL statement ❸ each using a variable list appropriate to that data set. Since the variable loop encompasses all variables in all data sets the %IF ❹ is used to select the appropriate variables from the jjth data set.

Two %DO loops are used to coordinate the two lists of macro variables. The outer loop (with index of &JJ) steps through the list of data sets. The inner loop (with index &KK) steps through all the variables for all the data sets. Since we are only interested in the variables for the data set identified by &&LIVEDB&JJ, that value is compared to the value of the data set in the inner loop (&&VDSN&KK) ❺.

Since the inner loop must pass through all variables in all data sets for each data set of interest, there is a built in inefficiency. This inefficiency is avoided in the following example dealing with the field checks.

Building a list while within a loop

The values in the FLDDIR data set are not loaded into macro variables until they are needed. This means that if we are within a macro loop that spans data sets (&&LIVEDB&JJ), we can create the field check list appropriate only for that particular data set. Consequently the symbol table will not include macro variables that are not needed.

```

%do jj = 1 %to &livecnt;
  %* check if dsn present;
  %exists(livedb.&&livedb&jj,no=&no) ❶

  %if &exists = N %then
    %put live.&&livedb&jj is not yet
      present in live.;
  %else %if &exists = Y %then %do;
    %put * Field error check for &&livedb&jj
      *****;

    * Build macro vars that will be used to;
    * construct the tests;
    %let fldcnt = 0;
    data _null_;
      set datamgt.flddir
        (where=(dsn="&&livedb&jj")) end=eof;❷
      i+1;
      ii=left(put(i,3.));
  %end;
%end;

```

```

call symput('fvar' || ii, trim(var)); ❸
call symput('ftyp' || ii, trim(chktype));
call symput('ftxt' || ii, trim(chktext));
if eof then call symput('fldcnt', ii); ❹
run;

```

```

%if &fldcnt gt 0 %then %do; ❺

```

We can first filter for the data sets that already exist by using the %EXIST macro ❶ (Carpenter, 1998, p.149). The DATA _NULL_ step that creates the macro variables only reads those observations from FLDDIR that match the name of the data set of interest (&&LIVEDB&&JJ) ❷. The macro variables are then built ❸ for each observation that passes the WHERE criteria. It is entirely possible that there will be no observations for a particular data set, this results in &FLDCNT = 0 ❹. This is noted using a %IF ❺ prior to performing the checks.

Field checks are made using IF-THEN-ELSE processing and assignment statements that are built inside of a data step by using the macro variables created in the previous step. In the code below any detected field errors are stored in the data set TEMPERR ❶. Each observation receives a date stamp with the date the error was first detected ❷. A macro %DO loop ❸ steps through each of the field checks for this data set (&&LIVEDB&&JJ).

```

* Perform field and intra-field checks;
data temperr(keep= status &&keys&&jj dsn var
              count msg text value chkdate); ❶
  set datamt.&&livedb&&jj;
  by &&keys&&jj;

  * Date these field check problems ;
  * were first detected);
  retain chkdate %sysfunc(today()); ❷
  format chkdate date9.;

  * Count the number of times this ;
  * problem has been detected;
  * Status will be controlled by the ;
  * manager - initialize to NEW;
  length status $12;
  retain count 1 status 'NEW';

  * Specify various lengths to the ;
  * data set variables;
  * VALUE is only given a length of 15;
  * this can cause some truncation (in display);
  length dsn var $8 value $15 text msg $100;

  * Place the name of the data set into;
  * a data variable;
  retain dsn "&&livedb&&jj";

  %* Build the Field and Intra-Observation;
  %* Field error checks;

```

```

%do i = 1 %to &fldcnt; ❸
  %if %upcase(&&ftyp&i) = LIST %then %do; ❹
    if &&fvar&i not in&&ftxt&i then do; ❺
      var = "&&fvar&i"; ❻
      msg = 'Value is not on list';
      text = "&&ftxt&i";
      value = &&fvar&i;
      output temperr;
    end;
  %end;

```

There are several types of field checks, including LIST which specifies a list of acceptable values. When **&&FTYP&I** is LIST ❹, an IF-THEN-DO block is defined which checks to see if the value stored in the variable named in **&&FVAR&I** is in the list stored in **&&FTEXT&I**. When the value is not in the list ❺, a series of assignment statements are executed ❻. For the second observation in FLDDIR:

OBS	DSN	VAR	CHKTYPE	CHKTEXT
2	DEMOG	RACE	list	('1','2','3')

the DO block becomes:

```

if RACE not in('1','2','3') then do;
  var = "RACE";
  msg = 'Value is not on list';
  text = "('1','2','3)";
  value = RACE;
  output temperr;
end;

```

USING &&VAR&I MACRO ARRAYS IN SCL PROGRAMS

Within a SCL program it is not possible to use the **&&VAR&I** macro variable form. Any direct macro variable references made with the use of ampersands will be resolved at the compilation of the SCL program rather than at its execution. Instead macro variables are accessed by using the SYMGET and SYMGETN functions to create SCL variables.

Simple macro variables are retrieved directly without the use of subscripts. In the following example the project code and the task path have been stored in macro variables with the names of project and path respectfully. These are retrieved using the SYMGET function to create SCL variables which are then used to build a *libref*.

```

* Determine path to the two DE data locations;
path = symget('path');
project = symget('project');
opath = trim(path) || '\ ' || trim(project) ||
        trim(tst) || '\data\ ' || els;
* Establish the libref for the DE;
sysrc = libname('ode', opath);

```

When you want to step through a series of macro variables that have been created with a subscript, a

loop is again used. This time, of course, it will be a SCL loop ❶ and the SCL variable is used as the index to identify the specific macro variable. The SCL loop creates a numeric index variable, which is converted to character ❷. This index is then appended to the root name of the macro variable series ❸ and the concatenated value is retrieved using SYMGET or SYMGETN ❹.

```
* Step through the list of data sets;
cnt = symgetn('livecnt');
do i=1 to cnt; ❶
  ii = left(put(i,3.)); ❷
  * Get the data set name and open it.;
  dsn = 'datamgt.'
      ||left(symget('livedb' || ii❸)); ❹
  dsid = open(dsn);
```

SUMMARY

Control data sets are used to store any project, data set, or variable specific information that will be needed by the application programs. This information includes the names of data sets, the variables within those data sets, variable attributes, and field check information. These control data sets are then used to create a series of macro variables. Application programs that require project specific information, such as the names of data sets and variables, use these macro variable lists to dynamically build the SAS code needed for the project/data set/variable of interest.

Dynamic code building requires the use of SAS macros and a number of macro statements. The macro %DO loop is used extensively as is the &&VAR&I macro variable form. The double ampersand macro variable (&&VAR&I) acts like a macro variable array with VAR as the array name and the &I macro variable as the subscript. The macro variables themselves are generally created from the control data sets by the use of the CALL SYMPUT routine.

This is an advanced macro topic. Creating the SAS programs and macros that can take advantage of &&VAR&I macro variables is not initially easy. You may need to practice and work with the techniques discussed in this paper for awhile before dynamic programming techniques become second nature for you.

REFERENCES

Burlew, Michele M., *SAS® Macro Programming Made Easy*, Cary, NC: SAS Institute, Inc., 1998, 280 pp.

Carpenter, Arthur L., 1997, "Resolving and Using &&var&i Macro Variables", *Proceedings of the Twenty-Second Annual SAS User Group International Conference*, Cary, NC: SAS Institute Inc .

Carpenter, Arthur L., 1998, "Advanced Macro Topics: Utilities and Examples", *Proceedings of the Twenty-Third Annual SAS User Group International Conference*, Cary, NC: SAS Institute Inc.

Carpenter, Arthur L., 2004, *Carpenter's Complete Guide to the SAS® Macro Language, 2nd Edition*, Cary, NC: SAS Institute Inc.

Carpenter, Arthur L. and Richard O. Smith, 2002, "Library and File Management: Building a Dynamic

Application", *Proceedings of the Twenty-Seventh Annual SAS User Group International Conference*, Cary, NC: SAS Institute Inc.

SAS® Macro Language: Reference, Version 8, Cary, NC: SAS Institute Inc., 1999, 310 pp.

ABOUT THE AUTHORS

Richard Smith and Art Carpenter are SAS Certified Professionals™. Both are senior partners at Data Explorations, a SAS Alliance Member™, which provides data management, analyses, and SAS programming services nationwide.



Arthur L. Carpenter

Art Carpenter's publications list includes three books on SAS topics (*Annotate: Simply the Basics*, *Quick Results with SAS/GRAPH® Software*, and *Carpenter's Complete Guide to the SAS® Macro Language, 2nd Edition*), two chapters in *Reporting from the Field*, and over six dozen papers and posters presented at various user group conferences. Art has been using SAS since 1976 and has served in a variety of positions in user groups at the local, regional, and national level.

Richard O. Smith

Richard Smith has a masters in Biology/Ecology and has provided complete data management and analysis services for numerous environmental research projects as a senior biologist, SAS programmer, and project manager. He has also provided programming and management services for the health related industries. He has been using SAS extensively since 1981.

AUTHOR CONTACT

Data Explorations
PO Box 430
Vista, CA 92085

Arthur L. Carpenter
(760) 945-0613
art@caloxy.com

Richard O. Smith
RSmith@SciX.com

TRADEMARK INFORMATION

SAS, SAS/AF, SAS/GRAPH, SAS Certified Professional, and SAS Alliance Partner are registered trademarks of SAS Institute Inc. in the USA and other countries.

® indicates USA registration.