

USING METADATA AND PROJECT DATA FOR DATA-DRIVEN PROGRAMMING

Brian Varney, COMSYS IT Partners, Inc.
Kalamazoo, MI



Abstract

Creating dynamic, data driven programs is a very powerful tool. Often we can use the metadata or the project data itself to help write our programs. This paper will introduce some of the concepts related to writing SAS® programs which will generate pieces of SAS code in a data driven manner.

Parts of the SAS language used in this discussion include the following:

1. macro call routines (call symput and call execute)
2. PROC SQL
3. SAS Dictionary Tables
4. SAS Component Language (SCL) Functions
5. Stored Processes

Programming techniques used in this discussion include the following:

1. using sequential macro variables with macro do loops
2. generating SAS code, writing it to an external file, and %including it at a later time
3. generating and executing SAS code using call execute
4. checking if a data set exists
5. checking if a variable exists in a data set
6. checking if a data set is empty
7. using subqueries for multiple data set subsetting

Introduction

A major problem that can occur in project code is hard coding project specific facts into a program. A good example is a program used across multiple protocols in a clinical trial. For instance, we may know that all protocols except for 3 and 5 already have the variable age calculated in the demographics module. This may tempt us to have something like the following in our program.

```
%if &prot.=3 or &prot=5 %then  
%do;  
%* calculate the age variable.;  
%end;
```

This works fine but what happens if more protocols come up at a later time which also lack the age variable. The code will have to be changed when this happens. It may be beneficial to programmatically check to see if the variable age exists in the demographics module. Based on that data driven check, the age variable can be calculated if necessary.

Using the parts of the SAS language and programming techniques discussed in this paper can result in low maintenance, reusable programs. They can also sometimes save a lot of typing!

This paper is intended for those with previous exposure to macro and SQL. Please be advised that some of these SAS tools are not available prior to SAS version 6.12.

Parts of the Sas Language

First we will review the parts of the SAS language utilized in this paper.

1. Call Symput

Call symput is a data step statement used to create a macro variable from a value or data set variable. Following is an example...

```
data _null_;  
    call symput('macvar','macro variable value');  
run;
```

The first argument is the name of the macro variable. The second argument is the macro variable value. This can also be a data set variable.

Call Execute

Call execute is also a data step statement which is used to execute whatever is passed into it. An example follows...

```
data _null_;  
    call execute('proc options; run;');  
run;
```

There is only one argument in a call execute.

Caution: There is a version 7 bug which compresses out spaces when combining multiple call executes. For example, the following code runs in version 6.12 but not in version 7.

```
data _null_;  
    call execute('proc ');  
    call execute('options; run;');  
run;
```

Version 7 ends up trying to execute the code

```
procoptions; run; instead of proc options; run;
```

This was fixed in Version 8.

2. PROC SQL

PROC SQL will be used in this paper to access the dictionary tables as well as create macro variables similar to how we can use call symput.

An example follows which places the data set label from SASHELP.RETAIL into a macro variable called dslabel;

```
proc sql noprint;
  select memlabel into :dslabel
  from dictionary.tables
  where libname='SASHELP' and memname='RETAIL';
quit;
```

Following is an example creating sequential macro variables from multiple records. This loads variable names from SASHELP.RETAIL into macro variables var1, var2, ..., varn. Even though room is left for 9,999 sequential macro variables only the necessary ones are created.

```
proc sql noprint;
  select name into :var1 - :var9999
  from dictionary.columns
  where libname='SASHELP' and memname='RETAIL';
quit;
```

Following is an example creating one concatenated macro variable from multiple records. This loads variable names from SASHELP.RETAIL into macro variable varlist separated by whatever we specify. This provides us with a handy list of variables to include in a keep= or drop= data set option later in the program.

```
proc sql noprint;
  select name into :varlist separated by ' '
  from dictionary.columns
  where libname='SASHELP' and memname='RETAIL';
quit;
```

If you are interested in building up a macro variable of quoted values try the following separated by ‘”’

3. Dictionary Tables

SAS dictionary tables contain the data behind the scenes in SAS a.k.a. the metadata about your current SAS session and data it is pointing to. The dictionary tables are only directly accessible from PROC SQL. There is also a corresponding set of data step views available via the SASHELP library.

Below is the list of dictionary tables and their corresponding SASHELP views as of SAS 9.1.3 Service Pack 3.

Member Name	Corresponding SASHELP View
CATALOGS	VCATALOG
CHECK_CONSTRAINTS	VCHKCON
COLUMNS	VCOLUMN
CONSTRAINT_COLUMN_USAGE	VCNCOLU
CONSTRAINT_TABLE_USAGE	VCNTABU
DICTIONARIES	VDCTNRY
ENGINES	VENGINE
EXTFILES	VEXTFL
FORMATS	VFORMAT
GOPTIONS	VGOPT
INDEXES	VINDEX
LIBNAMES	VLIBNAM
MACROS	VMACRO
MEMBERS	VMEMBER
OPTIONS	VOPTION
REFERENTIAL_CONSTRAINTS	VREFCON
REMEMBER	VREMEMB
STYLES	VSTYLE
TABLES	VTABLE
TABLE_CONSTRAINTS	VTABCON
TITLES	VTITLE
VIEWS	VVIEW

Below is a listing of the dictionary tables and their contents as of SAS 9.1.3 Service Pack 3.

Member	Column	Column Type	Column Label
CATALOGS	ALIAS	char	Object Alias
CATALOGS	CREATED	num	Date Created
CATALOGS	LEVEL	num	Library Concatenation Level
CATALOGS	LIBNAME	char	Library Name
CATALOGS	MEMNAME	char	Member Name
CATALOGS	MEMTYPE	char	Member Type
CATALOGS	MODIFIED	num	Date Modified
CATALOGS	OBJDESC	char	Object Description
CATALOGS	OBJNAME	char	Object Name
CATALOGS	OBJTYPE	char	Object Type
CHECK_CONSTRAINTS	CHECK_CLAUSE	char	Check Clause
CHECK_CONSTRAINTS	CONSTRAINT_CATALOG	char	Constraint Catalog

Member	Column	Column Type	Column Label
CHECK_CONSTRAINTS	CONSTRAINT_NAME	char	Constraint Name
CHECK_CONSTRAINTS	CONSTRAINT_SCHEMA	char	Constraint Schema
COLUMNS	FORMAT	char	Column Format
COLUMNS	IDXUSAGE	char	Column Index Type
COLUMNS	INFORMAT	char	Column Informat
COLUMNS	LABEL	char	Column Label
COLUMNS	LENGTH	num	Column Length
COLUMNS	LIBNAME	char	Library Name
COLUMNS	MEMNAME	char	Member Name
COLUMNS	MEMTYPE	char	Member Type
COLUMNS	NAME	char	Column Name
COLUMNS	NOTNULL	char	Not NULL?
COLUMNS	NPOS	num	Column Position
COLUMNS	PRECISION	num	Precision
COLUMNS	SCALE	num	Scale
COLUMNS	SORTEDBY	num	Order in Key Sequence
COLUMNS	TRANSCODE	char	Transcoded?
COLUMNS	TYPE	char	Column Type
COLUMNS	VARNUM	num	Column Number in Table
COLUMNS	XTYPE	char	Extended Type
CONSTRAINT_COLUMN_USAGE	COLUMN_NAME	char	Column
CONSTRAINT_COLUMN_USAGE	CONSTRAINT_CATALOG	char	Constraint Catalog
CONSTRAINT_COLUMN_USAGE	CONSTRAINT_NAME	char	Constraint Name
CONSTRAINT_COLUMN_USAGE	CONSTRAINT_SCHEMA	char	Constraint Schema
CONSTRAINT_COLUMN_USAGE	TABLE_CATALOG	char	Libname
CONSTRAINT_COLUMN_USAGE	TABLE_NAME	char	Table
CONSTRAINT_COLUMN_USAGE	TABLE_SCHEMA	char	Table Schema
CONSTRAINT_TABLE_USAGE	CONSTRAINT_CATALOG	char	Constraint Catalog
CONSTRAINT_TABLE_USAGE	CONSTRAINT_NAME	char	Constraint Name
CONSTRAINT_TABLE_USAGE	CONSTRAINT_SCHEMA	char	Constraint Schema
CONSTRAINT_TABLE_USAGE	TABLE_CATALOG	char	Libname
CONSTRAINT_TABLE_USAGE	TABLE_NAME	char	Table
CONSTRAINT_TABLE_USAGE	TABLE_SCHEMA	char	Table Schema
DICTIONARIES	FORMAT	char	Column Format
DICTIONARIES	INFORMAT	char	Column Informat
DICTIONARIES	LABEL	char	Column Label
DICTIONARIES	LENGTH	num	Column Length
DICTIONARIES	MEMLABEL	char	Dataset Label
DICTIONARIES	MEMNAME	char	Member Name

Member	Column	Column Type	Column Label
DICTIONARIES	NAME	char	Column Name
DICTIONARIES	NPOS	num	Column Position
DICTIONARIES	TYPE	char	Column Type
DICTIONARIES	VARNUM	num	Column Number in Table
ENGINES	ALIAS	char	Alias
ENGINES	DESCRIPTION	char	Description
ENGINES	ENGINE	char	Engine Name
ENGINES	PREFERRED	char	Preferred?
ENGINES	PROPERTIES	char	Engine Dialog Properties
EXTFILES	FILEREF	char	Fileref
EXTFILES	XENGINE	char	Engine Name
EXTFILES	XPATH	char	Path Name
FORMATS	DEFD	num	Default Decimal Width
FORMATS	DEFW	num	Default Width
FORMATS	FMTNAME	char	Format Name
FORMATS	FMTTYPE	char	Format Type
FORMATS	LIBNAME	char	Library Name
FORMATS	MAXD	num	Maximum Decimal Width
FORMATS	MAXW	num	Maximum Width
FORMATS	MEMNAME	char	Member Name
FORMATS	MIND	num	Minimum Decimal Width
FORMATS	MINW	num	Minimum Width
FORMATS	OBJNAME	char	Object Name
FORMATS	PATH	char	Path Name
FORMATS	SOURCE	char	Format Source
GOPTIONS	GROUP	char	Option Group
GOPTIONS	LEVEL	char	Option Location
GOPTIONS	OPTDESC	char	Option Description
GOPTIONS	OPTNAME	char	Option Name
GOPTIONS	OPTTYPE	char	Option type
GOPTIONS	SETTING	char	Option Setting
INDEXES	IDXUSAGE	char	Column Index Type
INDEXES	INDXNAME	char	Index Name
INDEXES	INDXPOS	num	Position of Column in Concatenated Key
INDEXES	LIBNAME	char	Library Name
INDEXES	MEMNAME	char	Member Name
INDEXES	MEMTYPE	char	Member Type
INDEXES	NAME	char	Column Name
INDEXES	NOMISS	char	Nomiss Option

Member	Column	Column Type	Column Label
INDEXES	UNIQUE	char	Unique Option
LIBNAMES	ENGINE	char	Engine Name
LIBNAMES	FILEFORMAT	char	Default File Format
LIBNAMES	LEVEL	num	Library Concatenation Level
LIBNAMES	LIBNAME	char	Library Name
LIBNAMES	PATH	char	Path Name
LIBNAMES	READONLY	char	Read-only?
LIBNAMES	SEQUENTIAL	char	Sequential?
LIBNAMES	SYSDESC	char	System Information Description
LIBNAMES	SYSNAME	char	System Information Name
LIBNAMES	SYSVALUE	char	System Information Value
MACROS	NAME	char	Macro Variable Name
MACROS	OFFSET	num	Offset into Macro Variable
MACROS	SCOPE	char	Macro Scope
MACROS	VALUE	char	Macro Variable Value
MEMBERS	DBMS_MEMTYPE	char	DBMS Member Type
MEMBERS	ENGINE	char	Engine Name
MEMBERS	INDEX	char	Indexes
MEMBERS	LIBNAME	char	Library Name
MEMBERS	MEMNAME	char	Member Name
MEMBERS	MEMTYPE	char	Member Type
MEMBERS	PATH	char	Path Name
OPTIONS	GROUP	char	Option Group
OPTIONS	LEVEL	char	Option Location
OPTIONS	OPTDESC	char	Option Description
OPTIONS	OPTNAME	char	Option Name
OPTIONS	OPTTYPE	char	Option type
OPTIONS	SETTING	char	Option Setting
REFERENTIAL_CONSTRAINTS	CONSTRAINT_CATALOG	char	Constraint Catalog
REFERENTIAL_CONSTRAINTS	CONSTRAINT_NAME	char	Constraint Name
REFERENTIAL_CONSTRAINTS	CONSTRAINT_SCHEMA	char	Constraint Schema
REFERENTIAL_CONSTRAINTS	DELETE_RULE	char	Delete Rule
REFERENTIAL_CONSTRAINTS	MATCH_OPTION	char	Match Option
REFERENTIAL_CONSTRAINTS	UNIQUE_CONSTRAINT_CATALOG	char	Unique Constraint Catalog
REFERENTIAL_CONSTRAINTS	UNIQUE_CONSTRAINT_NAME	char	Unique Constraint Name
REFERENTIAL_CONSTRAINTS	UNIQUE_CONSTRAINT_SCHEMA	char	Unique Constraint Schema
REFERENTIAL_CONSTRAINTS	UPDATE_RULE	char	Update Rule
REMEMBER	LIBNAME	char	Library Name
REMEMBER	MEMNAME	char	Member Name

Member	Column	Column Type	Column Label
REMEMBER	OFFSET	num	Offset into Text Remembered
REMEMBER	PW	char	Password
REMEMBER	RTEXT	char	Text Remembered
STYLES	CRDATE	num	Date Created
STYLES	LIBNAME	char	Library Name
STYLES	MEMNAME	char	Member Name
STYLES	STYLE	char	Style Name
TABLES	ATTR	char	Dataset Attributes
TABLES	AUDIT	char	Audit Trail Active?
TABLES	AUDIT_ADMIN	char	Audit Admin Image?
TABLES	AUDIT_BEFORE	char	Audit Before Image?
TABLES	AUDIT_DATA	char	Audit Data Image?
TABLES	AUDIT_ERROR	char	Audit Error Image?
TABLES	BUFSIZE	num	Bufsize
TABLES	COMPRESS	char	Compression Routine
TABLES	CRDATE	num	Date Created
TABLES	DATAREP	char	Data Representation
TABLES	DATAREPNAME	char	Data Representation Name
TABLES	DBMS_MEMTYPE	char	DBMS Member Type
TABLES	DELOBS	char	Number of Deleted Observations
TABLES	ENCODING	char	Data Encoding
TABLES	ENCRYPT	char	Encryption
TABLES	FILESIZE	num	Size of File
TABLES	GEN	num	Generation number
TABLES	INDXTYPE	char	Type of Indexes
TABLES	LIBNAME	char	Library Name
TABLES	MAXGEN	num	Maximum number of generations
TABLES	MAXLABEL	num	Longest label
TABLES	MAXVAR	num	Longest variable name
TABLES	MEMLABEL	char	Dataset Label
TABLES	MEMNAME	char	Member Name
TABLES	MEMTYPE	char	Member Type
TABLES	MODATE	num	Date Modified
TABLES	NLOBS	num	Number of Logical Observations
TABLES	NOBS	num	Number of Physical Observations
TABLES	NPAGE	num	Number of Pages
TABLES	NVAR	num	Number of Variables

Member	Column	Column Type	Column Label
TABLES	OBSLEN	num	Observation Length
TABLES	PCOMPRESS	num	Percent Compression
TABLES	PROTECT	char	Type of Password Protection
TABLES	REQVECTOR	char	Requirements Vector
TABLES	REUSE	char	Reuse Space
TABLES	SORTCHAR	char	Charset Sorted By
TABLES	SORTNAME	char	Name of Collating Sequence
TABLES	SORTTYPE	char	Sorting Type
TABLES	TYPEMEM	char	Dataset Type
TABLE_CONSTRAINTS	CONSTRAINT_CATALOG	char	Constraint Catalog
TABLE_CONSTRAINTS	CONSTRAINT_NAME	char	Constraint Name
TABLE_CONSTRAINTS	CONSTRAINT_SCHEMA	char	Constraint Schema
TABLE_CONSTRAINTS	CONSTRAINT_TYPE	char	Constraint Type
TABLE_CONSTRAINTS	INITIALLY_DEFERRED	char	Initially Deferred?
TABLE_CONSTRAINTS	IS_DEFERRABLE	char	Is Deferred?
TABLE_CONSTRAINTS	TABLE_CATALOG	char	Libname
TABLE_CONSTRAINTS	TABLE_NAME	char	Table
TABLE_CONSTRAINTS	TABLE_SCHEMA	char	Table Schema
TITLES	NUMBER	num	Title Number
TITLES	TEXT	char	Title Text
TITLES	TYPE	char	Title Location
VIEWS	ENGINE	char	Engine Name
VIEWS	LIBNAME	char	Library Name
VIEWS	MEMNAME	char	Member Name
VIEWS	MEMTYPE	char	Member Type

3. SAS Component Language (SCL) Functions

There are many SAS Component Language functions which can be called from the data step or %SYSFUNC() which can help our programs make data driven decisions.

4. Stored Processes within the SAS Microsoft Office Plug-in

Stored processes are SAS programs stored in a central location and registered with the Management Console. These stored processes can be utilized by the Microsoft Office Plug-In as well as other BI tools within the new SAS 9 technology packages. A stored process code is not much different than a normal SAS macro nested between %STPBEGIN; and %STPEND;.

Now that we have discussed the major SAS tools of interest in this paper, let's talk about using these components to write code that writes code.

Programming Techniques

Following is one problem solved using three different programming techniques.

Many times we have access to files, which contain variable labels. Instead of typing all of these in or doing some awkward cutting and pasting. Consider the following data set called work.vardata containing the variables var and varlabel. A data set called demo contains the variables var1, var2, and var3 which we want to apply these variable labels to.

Var	Variable
Var1	Patient #
Var2	Date of Visit
Var3	Date of Birth

There are various ways that we can generate the label statement programmatically. Even though there are only three variables here, there are times that we are presented with hundreds or thousands of variables, which need labels generated and we do not want to type them in! What we want to end up with is the following label statement inside of a SAS data step or procedure.

```
Label var1='Patient #'  
      var2='Date of Visit'  
      var3='Date of Birth';
```

1. Using Sequential Macro Variables with Macro Do Loops

```
proc sql noprint;  
  select var into :VAR1-:VAR9999  
  from vardata;  
  
  select varlabel into :VL1-:VL9999  
  from vardata;  
  
  select count(*) into :NUMVL  
  from vardata; quit;  
  
%macro runit;  
  
proc datasets lib=work;  
  modify demo;  
  label  
    %do a=1 %to &NUMVL.;  
    &&VAR&a="&&VL&a."  
%mend runit;
```

```
        %end;  
    ;  
quit;  
  
%mend runit;
```

Note: macro do loops must be inside of a macro definition.

The double ampersand reference causes the macro reference to be scanned twice. For instance, when `a=1` `&&var&a` is resolved in the following way.

First scan: when two ampersands are encountered, they are resolved into one ampersand. The `&a` resolves into `1` leaving us with `&var1`.

Second scan: `&var1` resolves as it normally would.

2. Generating SAS Code, Writing it to an External File, and %including it at a Later Time

```
data _null_;  
    file 'tempprog.sas';  
    put @1 'proc datasets lib=work;';  
    put @4 'modify demo;';  
    put @4 'label ' ;  
    do until(last);  
        set vardata end=last;  
        put @7 var '=' varlabel ' ' ;  
    end;  
    put @4 ';;';  
    put @1 'quit;';  
    stop;  
run;
```

```
%include 'tempprog.sas';
```

3. Generating and Executing SAS Code Using Call Execute

```
data _null_;  
    call execute('proc datasets lib=work;');  
    call execute('modify demo;');  
    call execute('label ');  
    do until(last);  
        set vardata end=last;  
        call execute(var||'='||varlabel||'');  
    end;
```

```
call execute(';');  
call execute('quit;');  
stop;  
run;
```

Another handy thing to do programmatically is to check the status of a data set.

4. Checking to See if a Data Set Exists

```
proc sql noprint;  
  select left(put(count(*),8.)) into :exist  
  from dictionary.members  
  where libname='WORK' and  
         memname='JUNK';  
quit;
```

If the data set work.junk exists, the macro variable exist will be equal to 1; otherwise it will be equal to 0.

Another way to accomplish this with an SCL function is by the following...

```
data _null_;  
  exist=exist("work.a","DATA");  
  if exist then  
    put "LOCALNOTE: This SAS data set does exist so run the program.";  
  else  
    put "LOCALNOTE: This SAS data set does NOT exist so DO NOT run the program.";  
run;
```

or with a %SYSFUNC as follows.

```
%let aexist=%SYSFUNC(exist(work.a));
```

5. Checking to See if a Variable Exists in a Data Set

```
proc sql noprint;  
  select left(put(count(*),8.)) into :exist  
  from dictionary.columns  
  where libname='WORK' and  
         memname='JUNK' and  
         name='DOG';  
quit;
```

If the variable dog exists in the data set work.junk, the macro variable exist will be equal to 1; otherwise it will be equal to 0.

6. Checking to See if a Data Set is Empty

```
proc sql noprint;
  select left(put(nobs,8.)) into :nobs
  from dictionary.tables
  where libname='WORK' and
         memname='JUNK' ;
quit;
```

7. Using Subqueries for Cross Data Set Subsetting

```
proc sql;
  create table final as
  select *
  from prot008.demo
  where patient not in (select distinct patient
                       from prot008.vitals);
quit;
```

The data set final only has records from demo for patients who do not appear in the vitals data set.

Conclusion

As one can imagine, there are many opportunities to apply these techniques in our day to day programming. Whenever developing SAS code to be used over again or as part of a larger system of SAS programs, we should try to avoid hard coding information into our programs. This paper only scratches the surface in providing examples of data driven programming. Try to apply these ideas to your programming problems.

For even more ideas in data driven programming please see the paper “%SYSFUNC – The Brave New Macro World” by Chris Yindra. This paper begins on page 259 of the SUGI 23 Proceedings.

References

SAS Institute Inc, SAS Macro Language Reference

SAS Institute Inc, Getting Started with the SQL Procedure

SAS Institute Inc, SUGI 23 Proceedings(1998)

“%SYSFUNC – The Brave New Macro World”, Chris Yindra

SAS Version 9 Online Help

Acknowledgments

I would like to take this time to thank everyone I have worked with and learned from in my exciting SAS adventures.

Contact Information

Your comments and questions are valued and encouraged. The author may be contacted at:

Brian Varney
COMSYS IT Partners, Inc., National SAS Practice
5278 Lovers Lane
Portage, MI 49002
Work Phone: (800) 831-6314
Fax: (269) 344-1887
Email: BVarney@COMSYS.com

As a SAS Gold Partner with more than 20 years of SAS consulting experience, COMSYS provides assessments, clinical services, solution support and production analytics to make SAS work for your business.

About COMSYS

COMSYS IT Partners, Inc. (NASDAQ: CITP) is a leading information technology services company with 42 offices across the U.S. and offices in Canada and the U.K. Leveraging more than 30 years of experience, COMSYS has enhanced its core competency of IT staffing services by creating client-centric, cost-effective information system solutions. COMSYS' service offerings include contingent staff augmentation of IT professionals, permanent recruiting and placement, vendor management and project solutions, including network design and management, offshore development, customized software development and maintenance, software globalization/localization translation services and implementation and upgrade services for SAS, business intelligence and various ERP packages. COMSYS serves clients in a variety of industries, including financial services, insurance, telecommunications, energy, pharmaceutical and healthcare industries and government agencies. For more information, visit www.comsys.com.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.