Publishing SAS® Metadata Using Macros, PROC SQL and Dictionary Tables

John Fahey and Barry Campbell,

Reproductive Care Program of Nova Scotia, DHW, Halifax, Canada

ABSTRACT

Users of your SAS data may not have the skills or access to obtain the metadata about it they require. This paper provides a method to extract and publish a catalogue containing variable names and types, descriptions, availability, and proportion missing. The macro is most useful when dealing with wide data; the motivating example has more than 10,000 variables. It iterates through a provided list, and uses PROC SQL to pull the relevant details from built-in dictionary tables. A summary dataset is produced and used as the basis for delivery to the end user via Excel, web applications, or portable documents. One unique feature of this macro is that, for datasets with an observation date, it determines the start and end of availability for each variable.

INTRODUCTION

In databases that collect data over many years, such as health data registries, data elements may be introduced and dropped over time as the needs and circumstances evolve. An important aspect of the metadata for such databases is that of dating information to indicate when variables started and stopped being collected.

Users of research datasets frequently want to know the time periods for which given variables are available. For example, they may be doing a multi-year trend analysis and want to know the how many years of usable data are available. When did data collection start and stop for each variable?

Generating such metadata requires a repetitive algorithm to automatically inspect and provide summary information for what may be a large collection of variables. In this paper, we describe code that combines data structure lookups from SAS dictionary tables, aggregation functions of SQL and the code generation facility of the SAS macros to compile metadata listings for arbitrary datasets.

OVERVIEW

The macro accepts a dataset, a list of variable names from the dataset, and a single date variable. For each variable in the list, it finds the find the earliest and latest date for which a non-missing value exists, counts the number of non-missing values, and stores the dates and counts in a result table.

The date variable would typically be an observation or event date that places observations in chronological sequence. For instance, in our motivating example of health record data, we are interested in the date a patient was discharged from hospital.

HOW IT WORKS: SUBSETTING OUR VARIABLES

One cool thing about SAS (among many!) is the way you can abbreviate lists of variables names. For example, Var1 Var2 Var3 or Var1 – Var3 or Var1 -- Var3 or _numeric_ or _all_ are all forms the user of our macro can choose to specify VarList. Given VarList, the first step filters out unwanted variables by creating a temporary work dataset that contains only the variables the user is interested in. This, along with the binary compression option, helps optimize the program when operating on very large and wide datasets. Getting rid of unnecessary variables at this stage also makes querying the SAS dictionary tables straightforward, as we will see later. Here is the data step in which we filter out unwanted variables:

```
data Work._Subset_    ( compress  =   binary );
    set &DSN        ( keep  =   &DateVar &VarList );
run;
```

START WITH THE END IN MIND

We want our program to produce a list of variable names, the first and last date each variable is available (i.e. non-missing), a count of non-missing and the total number of observations. In PROC SQL, we can set up an empty work table to hold this information using an SQL CREATE statement, specifying the names and data types of the columns we need. The following statement creates the _MSSWork_ table which we will soon fill with a row for each variable named in VarList.

```
create table Work._MSSWork_
        ( VName     char(32),
          VType     char(4),
          VLabel    char(256),
          FirstD8   num  format = YYMMDD10.,
          LastD8    num  format = YYMMDD10.,
          NonMiss   num,
          NTotal    num,
          RowNum    num );
```

PULLING VARIABLE NAMES AND TYPES FROM THE DICTIONARY

SAS dictionary tables reside in the SASHelp library and PROC SQL provides the most convenient and optimized access to them. The VColumn, actually a view of the Column table, contains a row describing every variable in every dataset visible to SAS. After our initial data step to subset our requested VarList as Work._Subset_, SAS makes information about the dataset's structure available immediately in dictionary tables. We can easily obtain the names and types of our variable subset by running a SQL SELECT statement on VColumn.

```
select name, type, label
      into :VNames separated by ' ',
           :VTypes separated by ' ',
           :VLabels separated by '09'x
      from SASHelp.VColumn
      where libname = "WORK"
        and memname = "_SUBSET_"
        and name NE "&DateVar";
```

Note the use of **`into`** variables VNames, VTypes,and VLabels and the **`separated by`** clause. This is special SAS syntax that tells PROC SQL to return the results of the query into macro variables as delimited string lists, one for variable names, and another for variable types, and a third for labels. For the labels, blanks are a poor choice of delimiter because labels can legitimately contain them, so we use the tab (ASCII 09) character instead. After PROC SQL executes this statement, macro variable &VNames will, for example, contain "Var1 Var2 Var3 ... Var1000" and &VTypes will contain "char num char ... num". In the next step, our program will iterate over these lists and generate SQL specific to each variable.

Note also the explicit exclusion of the date variable (name NE "&DateVar"). We don't want this variable in our lists because we are using it to look up start and end dates for all other variables.

AGGREGATION WITH SQL

Before we move on, however, let's look at how we can use SQL to roll up data. SQL aggregate functions provide an ideal way to do this because they allow you to obtain summary information about a subset of observations. Unlike their DATA step counterparts, aggregate functions sum(), min(), max() and count() can operate on a set of data rows and return a single-row result. For example, in a dataset recording daily temperature observations, SQL can easily find the first and last available observations by aggregating over the non-missing values using the min and max functions:

```
select min(ObsDate), max(ObsDate) from DailyTemp

where not (missing(Temperature))
```

Similarly, it can count the non-missing observations using the count() aggregate function:

```
select count(*) from DailyTemp where not (missing(Temperature))
```

CODE GENERATION WITH SAS MACRO LOOPING

SQL aggregation does a nice job of summarizing a single variable but we need many such select statements to get aggregate information for all variables in our dataset. And since we want the program to work for any dataset and list of variables we give it, we will need to automate the generation of the SQL statements. We accomplish this by enclosing the SQL in a SAS macro %do loop that iterates over the list of variable names and builds custom SQL statements for each variable. The set of values resulting from the SELECT statements can then be INSERTed into _MSSWORK_, and UPDATEd in the next loop.

The %do statements make use (vicariously) of the built-in macro variable &SQLObs. This variable stores the number of rows (observations) processed by the last SQL statement executed, in this case, the select from dictionary table VColumn we ran to get variable names, types and labels.

As well as a looping construct, we use conditional logic to determine which SQL expression becomes the object-item populating the FirstD8 and LastD8 fields. The ELSE part simply chooses the min or max value of the date variable for the field, whereas the IF part makes use of INTNX (see below) to move the start date back to the beginning of the fiscal year, and the end date to the last day of the fiscal. Whether to choose IF or ELSE is determined by the fiscal year input flag &FY. Similar conditional code substitution is part of the WHERE clause for the SELECT, which decides whether all non missing, or all non-zero rows, are to be examined for start and finish date. This distinction is obviously only important for variables with type=num.

As an aside, the transformation of dates to fiscal year boundaries makes use of the intnx function which increments a date (or time or datetime) by a given interval:

```
intnx ( 'Year.4', min ( &DateVar ), 0 , 'Beginning' )
```

Here the interval is year, shifted to start/end at the fourth sub-period (i.e. month, thus April Fool's Day). The starting point is provided by the earliest valid date, from which we wish to move 0 years forward, and pick the beginning of the period – April 1st of the (fiscal) year corresponding to the earliest valid date. For the motivating example we know that collection of many variables began on fiscal year boundaries although, due to their sporadic occurrence, no actual values may appear until sometime after the first day of the year. Similarly, the end of fiscal can be placed in LastD8.

The result of the repeated iteration of the %do loop, after the SAS macro processor does its text substitution magic, is an expanded version of the PROC SQL step that contains many, maybe thousands, of SQL statements to populate the result table. To see the expanded code, we enable the MPRINT logging option that shows the text generated by the macro's execution. See Figure 1 for an example.

Fig 1.
Expanded Code

```
insert into Work._MSSWork_ select "Var1" , "char" , "" , min ( ObsDate ) , max ( ObsDate ) , . , . , 1 from Work._Subset_ where not ( missing ( Var1 ) ) ;
insert into Work._MSSWork_ select "Var2" , "num" , "" , min ( ObsDate ) , max ( ObsDate ) , . , . , 2 from Work._Subset_ where not ( missing ( Var2 ) ) ;
insert into Work._MSSWork_ select "Var3" , "char" , "" , min ( ObsDate ) , max ( ObsDate ) , . , . , 3 from Work._Subset_ where not ( missing ( Var3 ) ) ;
…
insert into Work._MSSWork_ select "Var1000" , "num" , "" , min ( ObsDate ) , max ( ObsDate ) , . , . , 1000 from Work._Subset_ where not ( missing ( Var1000 ) ) ;
```

END WITH THE START IN MIND

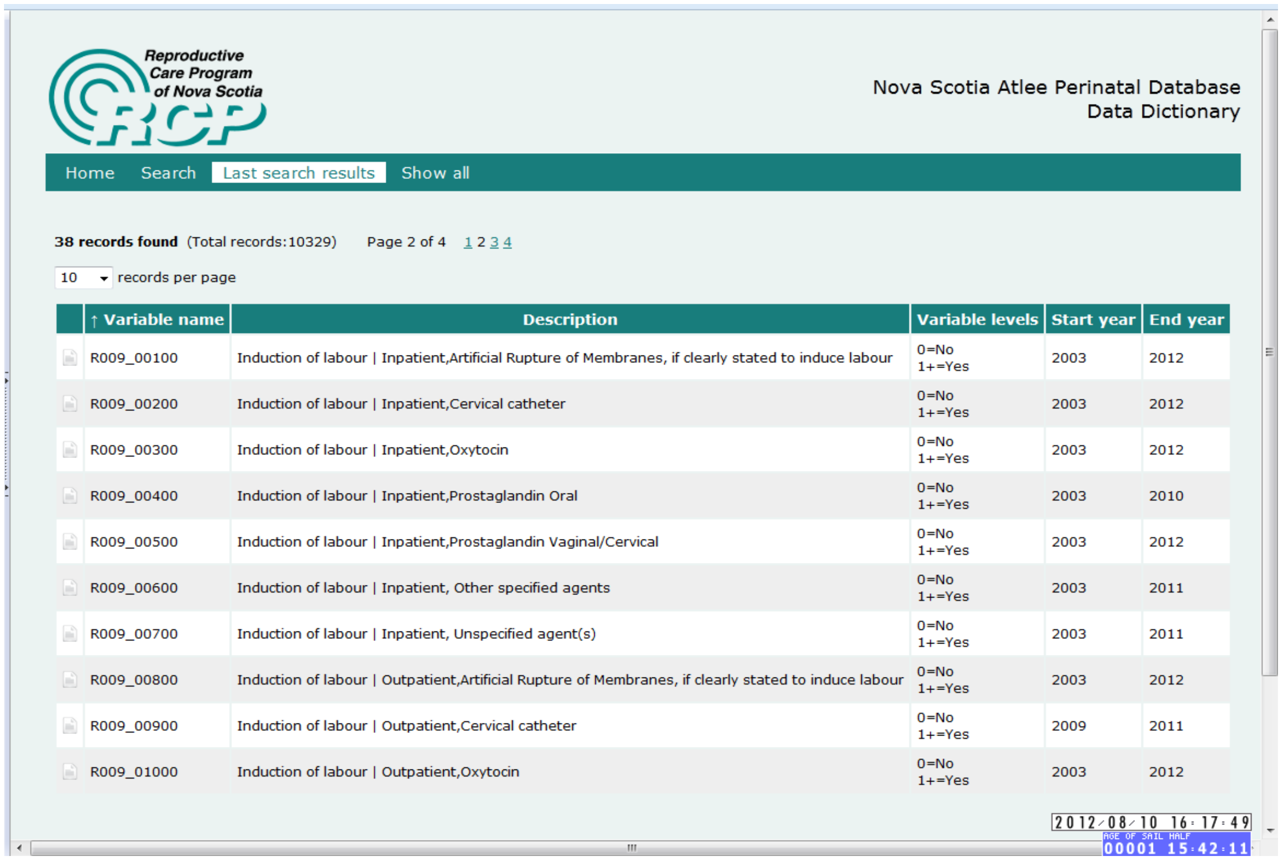
The final step in the macro is another SQL statement, this one to update the rows just INSERTed by the previous %do loop. During the INSERT, the variables representing the total number of rows for that input field, that are dated within the range from first to last date of valid (i.e. non-missing or non-zero) data occurrence, and the proportion of those that are valid, were left missing. Since the dates were determined, and then adjusted as necessary, they're now available for use. Again the variables are examined one at a time with the name and type picked out of the concatenated strings, and then the number of rows counted using a correlated sub-query. The same conditional logic as was needed in the where clause is applied to see how many within this restricted date range are valid and the result table row populated

HOW TO CALL THE MACRO IN YOUR PROGRAMS

```
%start_end ( Atlee.monster1, _num_, DLAdmsd8, 1, 0, Work.VarTest ) ;

data Work.ForTest;

set sashelp.class;

DummyD8 = int ( ranuni ( 0 ) * 10000 );

format dummyd8 yymmdd10.;

if _N_ >= 5 then Weight = .;

run;


%start_end ( ForTest, Name -- Weight, DummyD8, 1, 0, TestOut )
```

PUBLISHING THE CATALOGUE

One method for presenting the metadata result table in a more-accessible format is to convert it to a database and have third-party tools convert it to web pages. For our motivating example we loaded the result table into a MySQL database and used DaDaBik, an open-source tool that generates web-based database front-ends, to build a simple searchable catalogue. Display as web pages can, in its simplest form, be done within SAS with


Reproductive Care Program of Nova Scotia
RCP
Nova Scotia Atlee Perinatal Database
Data Dictionary
Home
Search
Last search results
Show all

38 records found (Total records:10329) Page 2 of 4 1 2 3 4

10 records per page

	↑ Variable name	Description	Variable levels	Start year	End year
	R009_00100	Induction of labour | Inpatient,Artificial Rupture of Membranes, if clearly stated to induce labour	0=No 1+=Yes	2003	2012
	R009_00200	Induction of labour | Inpatient,Cervical catheter	0=No 1+=Yes	2003	2012
	R009_00300	Induction of labour | Inpatient,Oxytocin	0=No 1+=Yes	2003	2012
	R009_00400	Induction of labour | Inpatient,Prostaglandin Oral	0=No 1+=Yes	2003	2010
	R009_00500	Induction of labour | Inpatient,Prostaglandin Vaginal/Cervical	0=No 1+=Yes	2003	2012
	R009_00600	Induction of labour | Inpatient, Other specified agents	0=No 1+=Yes	2003	2011
	R009_00700	Induction of labour | Inpatient, Unspecified agent(s)	0=No 1+=Yes	2003	2011
	R009_00800	Induction of labour | Outpatient,Artificial Rupture of Membranes, if clearly stated to induce labour	0=No 1+=Yes	2003	2012
	R009_00900	Induction of labour | Outpatient,Cervical catheter	0=No 1+=Yes	2009	2011
	R009_01000	Induction of labour | Outpatient,Oxytocin	0=No 1+=Yes	2003	2012

2012/08/10 16:17:49
AGE OF SAIL HALF 00001 15:42:11

ODS HTML output. We have also used PROC PRINT with PAGEBY to give a simple table of contents as PDF bookmarks.

SUGGESTIONS FOR IMPROVEMENT

Macro variables are restricted to a maximum of 2^{16}-2 characters so the technique of selecting variable names, types and labels into macro variables may fail when very wide datasets with verbose labels or very long names are provided as input. An alternative method would be to have these three metadata variables simply selected from VColumn into a dataset. Pedagogically, this doesn't allow exploration of the cool features of the interface between PROC SQL and the macro facility, although it does provide a simpler and more robust solution. Conversion to this technique is left as an exercise.

For health administrative data, rounding the start and end dates to fiscal year boundaries is a natural choice. Other time units could be supported, e.g. month or day, or by using a datetime instead of a date, shorter intervals as well.

The ZERO flag, indicating whether to interpret 0 as a missing value, currently affects the treatment of all (numeric) variables but it could be made per variable by supplying a set of flags of the same length as the number of variables, e.g. 0101110011010001. This could be performed by using a substring function within the %do loop.

CONCLUSIONS

The code generation facility of SAS macros, the availability of metadata in dictionary tables and the extensions in PROC SQL to interface with these, make a powerful combination for building metadata catalogues in an automated fashion. This can be important for datasets that have large quantities of it. The unique feature of this macro is that there's a time component to the data and this is extracted and displayed separately for each variable, along with completeness information. If you don't need this feature, another alternative is %FreqAll (see references, especially the bibliography from the first reference).

REFERENCES

http://www.nesug.org/proceedings/nesug07/cc/cc11.pdf

http://www.lexjansen.com/wuss/2007/CodersCorner/COD_Chow_MacroToPut.pdf

http://www.ats.ucla.edu/stat/sas/library/nesug99/bt066.pdf

http://support.sas.com/documentation/cdl/en/sqlproc/62086/HTML/default/viewer.htm#a001385596.htm

ACKNOWLEDGMENTS

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

John Fahey or Barry Campbell
Reproductive Care Program of Nova Scotia
5991 Spring Garden Road, Suite 700
Halifax, NS, Canada B3H 1Y6 Work Phone:
Fax: 902-470-6791
Email: john.fahey@iwk.nshealth.ca
Web: http://rcp.nshealth.ca

* *

APPENDIX I

```
Options symbolgen mlogic mprint;

%macro start_end ( DSN, VarList, DateVar, Zero, FY, RslTbl )
;
/* Macro start_end                                                */
/* Input:   DSN [ DataSet Name ]
            VarList [ VARiable LIST ] of variables from DSN where each
                may have missing rows, may be a mix of num and char
            DateVar [ DATE VARiable ] defining time period over which
                values may be potentially missing
            Zero flag to indicate whether missing values are of interest
                or values of 0; defined globally, not per-variable
            FY [ Fiscal Year ] flag to indicate whether date of first
                or last occurrence is to be used or these should be
                expanded to include the entire fiscal years in which
                those dates occur
            RslTbl [ ReSuLT taBLe ] name of dataset into which results
                will be compiled                                  */
/* for each variable in VarList, find the earliest and latest value of */
/* DateVar for which a non-missing value exists in dataset DSN         */
/* Unless Zero is true, in which case a numeric non-zero value is the  */
/* one sought; if FY is true then return the beginning of the          */
/* Fiscal Year in which there's first a non-missing value and the end  */
/* of the FY for which there's last a non-missing value                */
/* The results are returned in the specified dataset RslTbl which      */
/* includes the variable's name, label, date of first non-missing ( or */
/* non-zero ) value, date of last value ( or the beginning and end of  */
/* the fiscal years in which these dates occur ), number of non-missing*/
/* ( or non-zero values ), and total number of data rows in that time  */
/* period ( i.e. 100 - [NonMiss / NTotal] = % missing )                */
/* also produces working datasets _Subset_ and _MSSWork_               */
/* NOT deleted after execution                                         */

data Work._Subset_
    ( compress  =   binary )
;
    set &DSN
        ( keep  =   &DateVar &VarList )
```

```
;
run;

proc sql noprint ;
    create table Work._MSSWork_
        ( VName     char(32),
          VType     char(4),
          VLabel    char(256),
          FirstD8   num
                    format = YYMMDD10.,
          LastD8    num
                    format = YYMMDD10.,
          NonMiss   num,
          NTotal    num,
          RowNum    num );
    select  name
            , type
            , label
        into    :VNames
            separated by ' '
                , :VTypes
            separated by ' '
                , :VLabels
            separated by '09'x  /* tab-delimited */
        from    SASHelp.VColumn
        where   libname =   "WORK"
            and memname =   "_SUBSET_"
            and name    NE  "&DateVar" ;

    %let    NVars   =   &SQLObs
;
    %do i = 1 %to &NVars ;
        %let    ThisVar =   %scan ( &VNames, &i, ' ' );
        %let    ThisType=   %scan ( &VTypes, &i, ' ' );
        %let    ThisLbl =   %qscan ( %bquote ( &VLabels ), &i, '09'x );
        insert into Work._MSSWork_
            select  "&ThisVar"
                    , "&ThisType"
                    , "&ThisLbl"
                    , %if &FY %then intnx ( 'Year.4', min ( &DateVar ), 0 ,
'Beginning' );
                      %else min ( &DateVar );
                    , %if &FY %then intnx ( 'Year.4', max ( &DateVar ), 0 , 'End'
);
                      %else max ( &DateVar );
                    , .
                    , .
                    , &i
                from    Work._Subset_
                %if ( ( &Zero ) and ( &ThisType = num ) ) %then %do;
                    where   &ThisVar NE 0 ;
                %end ;  %* if;
                %else %do;
                    where   not ( missing ( &ThisVar ) ) ;
                %end;   %* else;
    %end;   %* do;


    %* DO it over again, filling in the (a) proportion missing and (b) total no.
of rows;
    %* required correct dates to be already calculated as (a) and (b) refer to
obs within the date range only;


    %do i = 1 %to &NVars ;
        %let    ThisVar =   %scan ( &VNames, &i );
        %let    ThisType=   %scan ( &VTypes, &i );
        update  Work._MSSWork_ as W
            set NTotal  =   ( select   count ( * )
                                from    Work._Subset_
                                where   &DateVar between W.FirstD8 and W.LastD8
),
                NonMiss =   ( select   sum ( %if ( ( &Zero ) and ( &ThisType =
num ) ) %then ( &ThisVar NE 0 );
                                             %else not ( missing ( &ThisVar ) );
                                           )
                                from    Work._Subset_
                                where   &DateVar between W.FirstD8 and W.LastD8 )
                where   RowNum = &i
;
    %end;   %* do;
quit;
```

```
%mend start_end ;

/*
%start_end ( atlee.monster1, _num_, dladmsd8, 1, 0, Work.VarTest ) ;
*/

data Work.ForTest;
set sashelp.class;
DummyD8 = int ( ranuni ( 1111 ) * 10000 );
format dummyd8 yymmdd10.;
if _N_ >= 5 then Weight = .;
run;

%start_end ( ForTest, Name -- Weight, DummyD8, 0, 0, TestOut )
;
```