# Paper TU03

# Simple Ways to Use PROC SQL and SAS DICTIONARY TABLES to Verify Data Structure of the Electronic Submission Data Sets

Christine Teng, Merck Research Labs, Merck & Co., Inc., Rahway, NJ Wenjie Wang, Merck Research Labs, Merck & Co., Inc., Rahway, NJ

#### **ABSTRACT**

In preparing an electronic submission to the FDA, we are required to follow specific requirements such as: the length of a data set name and a variable name cannot exceed 8 characters, a data set label and a variable label should be less than 40 characters in length, and no user-defined format should be associated with any variables, etc. This paper will show you how we can achieve simple quality assurance steps using PROC SQL and SAS dictionary tables without hard coding.

#### **INTRODUCTION**

Electronic submission to the Food and Drug Administration (FDA) has become a standard practice in the pharmaceutical industry for a New Drug Application (NDA), Supplemental NDA (sNDA), safety update, or response to FDA review questions. Upon receiving the electronic submission package, the statistical reviewer in the FDA can check the analyses and summaries by manipulating the very data sets and programs that are used to generate them. The submission of these documents in electronic format should improve the agency's efficiency in processing, archiving, and reviewing them.

In January 1999, the FDA released the *Guidance for Industry: Providing Regulatory Submission in Electronic Format – NDAs*<sup>[1]</sup>. In Section K of Part IV of this release, the FDA provided guidance on format of the data sets, organization of data, documentation of the data sets, general consideration for data sets, etc. In order to comply with the agency's regulations, each company has its own set of Standard Operating Procedures (SOP). This paper will show simple ways to assure compliance with SOP when preparing the Statistical Review Aid (SRA) portion of the electronic submission package. Specifically, we will address how to verify the submitted data sets whether they meet the following five requirements:

- 1. The names of data sets cannot exceed 8 characters, each data set must have label and the length of data set label cannot exceed 40 characters.
- 2. The names of variables cannot exceed 8 characters, each variable must have label and the length of the variable label cannot exceed 40 characters
- 3. No user-defined format should be associated with any variables in a data set.
- 4. The length of any character values cannot exceed 200 characters and the optimal length for these variables will be provided.
- 5. Variables with the same name across multiple data sets must have the same attributes, in order to ensure consistency.

#### DICTIONARY TABLES vs. SASHELP VIEWS

SAS DICTIONARY tables are metadata (data about data); they are automatically available when a SAS session starts and are updated automatically whenever there is a change in a data set. SASHELP views are created from the DICTIONARY tables, so they replicate the information stored in the dictionary tables. However, there are some differences between these two. The major difference is that DICTIONARY tables are not available outside of the SQL procedure, while you can reference SASHELP views in the DATA step, as well as in other procedures, including PROC SQL. Another difference is that you cannot use data set options with DICTIONARY tables while you can with SASHELP views. This paper mainly focuses on the usage of DICTIONARY tables but you can achieve the same results by using SASHELP views since SASHELP views are based on the DICTIONARY tables.

Below we list four views that use dictionary tables<sup>[2,3]</sup>:

SASHELP View Name	PROC SQL Statement to Create the View	Function	
SASHELP.VCOLUMN	create view sashelp.vcolumn as select * from dictionary.columns;	Includes one observation for every variable available in the session. Information includes name, type, length, library and member name of the data set in which the variable resides.	
SASHELP.VMEMBER	create view sashelp.vmember as select * from dictionary.members;	Lists all data sets, catalogs, views, and multidimensional databases available in the session.	
SASHELP.VTABLE	create view sashelp.vtable as select * from dictionary.tables;	Contains the library reference, data set name, and other information for every data set available in the session. Does not include data views.	
SASHELP.VCATALG	create view sashelp.vcatalg as select * from dictionary.catalogs;	Contains a row of information for each SAS catalog: the name, location, and type of catalog (MACRO, FORMAT, and so on).	

To see how each DICTIONARY table is defined, submit a DESCRIBE TABLE statement. After you know how a table is defined, you can use its column names in the WHERE clause to get more specific information. Please read SAS documentation for additional information about this topic.

Let's look at the structure of four dictionary tables we would be using extensively. Notice the describe view returns a SELECT statement to the dictionary table.

```
TITLE "Using dictionary tables to get data set definitions";
PROC SOL;
   describe table dictionary.members;
   describe table dictionary.columns;
   describe table dictionary.tables;
   describe table dictionary.catalogs;
   describe view sashelp.vcolumn;
QUIT;
The output is shown in the log file as follow:
7031
          describe table dictionary.members;
NOTE: SQL table DICTIONARY.MEMBERS was created like:
create table DICTIONARY.MEMBERS
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   engine char(8) label='Engine Name',
   index char(32) label='Indexes',
   path char(1024) label='Path Name'
  );
          describe table dictionary.columns;
7032
NOTE: SQL table DICTIONARY.COLUMNS was created like:
create table DICTIONARY.COLUMNS
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   name char(32) label='Column Name',
   type char(4) label='Column Type',
   length num label='Column Length',
   npos num label='Column Position',
   varnum num label='Column Number in Table',
   label char(256) label='Column Label',
   format char(16) label='Column Format',
   informat char(16) label='Column Informat',
   idxusage char(9) label='Column Index Type'
  );
7033
          describe table dictionary.tables;
NOTE: SQL table DICTIONARY. TABLES was created like:
create table DICTIONARY.TABLES
   libname char(8) label='Library Name',
```

```
memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   memlabel char(256) label='Dataset Label',
   typemem char(8) label='Dataset Type',
   crdate num format=DATETIME informat=DATETIME label='Date
          Created',
   modate num format=DATETIME informat=DATETIME label='Date
          Modified',
   nobs num label='Number of Observations',
   obslen num label='Observation Length',
   nvar num label='Number of Variables',
   protect char(3) label='Type of Password Protection',
   compress char(8) label='Compression Routine',
   encrypt char(8) label='Encryption',
   npage num label='Number of Pages',
   pcompress num label='Percent Compression',
   reuse char(3) label='Reuse Space',
   bufsize num label='Bufsize',
   delobs num label='Number of Deleted Observations',
   indxtype char(9) label='Type of Indexes',
   datarep char(32) label='Data Representation',
   requector char(24) format=$HEX informat=$HEX
             label='Requirements Vector'
  );
          describe table dictionary.catalogs;
NOTE: SQL table DICTIONARY.CATALOGS was created like:
create table DICTIONARY.CATALOGS
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   objname char(32) label='Object Name',
   objtype char(8) label='Object Type',
   obidesc char(256) label='Object Description',
   created num format=DATETIME informat=DATETIME label='Date
           Created',
   modified num format=DATETIME informat=DATETIME label='Date
            Modified',
   alias char(8) label='Object Alias'
  );
          describe view sashelp.vcolumn;
NOTE: SQL view SASHELP. VCOLUMN is defined as:
        select *
          from DICTIONARY.COLUMNS;
```

## **EXAMPLES OF HOW TO RUN SIMPLE QA SCRIPTS**

We will use dictionary tables in our examples. The following scripts are tailored to specific requirements when preparing for the electronic submission package. DATADIR is the library name used in the scripts. You can also include multiple library names by using IN clause. In our examples, we will use one library name.

**Requirement #1**: All submitted analysis data sets should have a label, and the length of any label cannot exceed 40 characters. Also, the length of data set names should not exceed 8 characters.

The output will identify those non-complying data sets that reside in the DATADIR directory.

**Requirement #2:** All variables in the submitted analysis data sets should have names and labels, and the lengths of these should not exceed 8 and 40 characters, respectively.

```
TITLE "Check variable name, label and their lengths";

PROC SQL;
   select memname, name, length(name) as Variable_Length, label,
        length(label)as label_length
   from dictionary.columns
   where libname="DATADIR" and memtype="DATA" and
        (length(name)>8 or length(label)=1 or
        length(label)>40);

QUIT;
```

The output will identify non-complying variables: those variables whose names or labels are too long, and those that do not have labels.

**Requirement #3:** No user defined formats should be permanently associated with any variables in a data set.

```
TITLE "Check if variables are associated with user-defined formats";
```

The SAS-Supplied formats are one of compiled part of the hidden SAS catalogs. This implies that in the DICTIONARY.CATALOGS table, you won't be able to find SAS-supplied formats, only user defined formats. However, the format name stored in the Catalogs table doesn't have the ending period, while in the Columns table, the period is preserved, so we need to use the function of translate and trim to make the match.

**Requirement #4:** The length of any character value should not exceed 200 characters and the optimal length for these variables will be provided.

```
TITLE "Check length of character variable values that were
defined >= 200";
PROC SOL;
   create table longvar as
      select memname, name, length
      from
            dictionary.columns
      where libname="DATADIR" and memtype="DATA" and type='char'
      and
             length >= 200;
/* The following will build SQL script for each character
   variable that has length >= 200 and the actual max length of
   all values of the variable.
   select "select '" | memname | " as ds name, '"
           | name | | "' as var_name, max(length("
           | trim(left(name)) | | ")) as max_len, '"
          || put(length, best.)
           "' as defined_len from datadir."
          | trim(left(memname)) |  "; "
   into
          :select1-:select999
          longvar;
   from
OUIT;
```

First identify the variables whose defined length is equal to or greater than 200 characters. However, this does not mean that the true value of these variables exceed 200 characters. So in the second SELECT statement of the above PROC SQL block, we build the SQL scripts and put them into macro variables select1 through select999 (or any number big enough to hold all the variables whose length equal to or grater than 200 characters). Later on, we can run those scripts to identify non-complying variables. The syntax of string concatenation seems overwhelming. In SAS 9, there are several new

string functions such as CAT, CATS, CATT and CATX - can be used to concatenate strings.

The equivalent syntax using CATX is depicted as follows:

You can use the macro variable &sqlobs set by PROC SQL to determine the number of rows executed by previous SQL procedure statement. Please keep in mind that this automatic macro variable is changed for each SQL run. You may want to re-assign it to another user-defined macro variable to avoid getting wrong value.

```
%put &sqlobs;
```

The following macro executes all SQL scripts built earlier to print the character variable whose defined length equaled or exceeded 200 characters. In the output, you will see the actual maximum length of the variable in that table, and its defined length. Some functions such as SCAN, will give the default length of 200 in an assignment statement - if you did not define the length of the assigned variable. This script will let you determine the variables whose length we can reduce without loss of information. When the storage size of the data sets is the concern, this would be a good way to identify improvement.

Output of the above script:

ds_name	var_name	max_len	defined_len
DISCRESC	reasond	117	255

The macro call gives output for the SELECT statement for each variable that has length of 200 or more. It will be easier if we put the output into a data set and display the result all at once. A complete script for this purpose can be found in Appendix.

**Requirement #5:** Variables with the same name across multiple data sets must have the same attributes, in order to ensure consistency.

The following code snippet will identify those variables with same name across multiple data sets. Their attributes will be listed for comparison purposes. This script is also useful for cross reference if you have data feed from many sources. Here, we use SAS ODS to create an RTF report.

```
*Select variable names that are in multiple data sets;
OPTIONS orientation=Landscape;
ODS RTF FILE="c:\temp\columns.rtf";
TITLE "List of Columns in Multiple Data Sets";
PROC SOL;
   select name length=10, memname length=28,
          label length=50, type length=8,
          length length=8, format length=8,
         informat length=8
   from dictionary.columns
  where libname="DATADIR" and memtype="DATA"
   group by name
   having count(*) > 1
   order by name, memname;
QUIT;
ODS RTF CLOSE;
```

If you are interested in the variables with differences only, use the following:

```
OPTIONS orientation=Landscape;
ODS RTF FILE="c:\temp\columnsDiff.rtf";
TITLE "List of Columns in Multiple Data Sets with Discrepancy";

PROC SQL;
   select name length=10, memname length=28, label length=50,
        type length=8, length length=8, format length=8,
        informat length=8
   from dictionary.columns
   where libname="DATADIR" and memtype="DATA"
```

#### **CONCLUSION**

Using PROC SQL with dictionary tables can reduce the coding time for many data processing tasks such as validation. Other dictionary tables not mentioned in this paper can also provide SAS system information to allow you to automate processes without hard coding. The codes provided are very simple and you can easily enhance them with additional conditions or create a macro to handle all these checking conditions.

#### REFFERENCES

- 1. FDA Guidance for Industry: Providing Regulatory Submission in Electronic Format NDAs <a href="http://www.fda.gov/cder/guidance/2353fnl.pdf">http://www.fda.gov/cder/guidance/2353fnl.pdf</a>
- 2. SAS Bits and Bites by Melinda Thielbar, "The SASHELP Library: It Really Does Help You Manage Data" <a href="http://support.sas.com/sassamples/bitsandbytes/sashelp2.html">http://support.sas.com/sassamples/bitsandbytes/sashelp2.html</a>
- 3. SAS SQL Procedure User's Guide

#### **ACKNOWLEGEMENTS**

The authors would like to thank our management team for their encouragement and review of this paper.

#### **CONTACT INFORMATION**

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

Christine Teng
Merck & Co., Inc.
RY34-A320
P.O. Box 2000
Rahway, NJ 07065
(732) 594-3427
christine teng@merck.com

Sas Certified Advanced Programmer

Wenjie Wang Merck & Co., Inc. RY34-A320 P.O. Box 2000 Rahway, NJ 07065 (732) 594-3441 wenjie\_wang@merck.com



#### **TRADEMARKS**

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.

## **Appendix**

```
%MACRO print long var;
   PROC SQL;
        select "select '" || memname || "' as ds_name,'"
            | name | | "' as var_name, max(length("
              trim(left(name)) || ")) as max_len, '"
            put(length, best.)
             "' as defined_len from datadir."
             trim(left(memname)) || "; "
        into
                :select1-:select999
               (select memname, name, length
        from
                from dictionary.columns
                where libname="DATADIR" and
                memtype="DATA" and
                type='char' and
                length >= 200);
   QUIT;
   %let cnt=&sqlobs;
   PROC SQL;
     create table T1
       (ds name char(20),
       var_name char(20),
       max len num,
       defined len char(12));
     %do i=1 %to &cnt;
         insert into T1
         &&select&i;
     %end;
   QUIT;
   ODS RTF FILE="c:\temp\longvar.rtf";
   TITLE "List of Columns with Defined Length >= 200";
   PROC SQL;
       select ds_name length=20, var_name length=20,
               max len length=8, defined len length=12
       from
               T1
       order by ds_name, var_name;
   ODS RTF CLOSE;
%MEND print_long_var;
%print_long_var
```