MWSUG 2017 - Paper TT03

Arbovirus, Varicella and More: Using SAS® for Reconciliation of Disease Counts

Misty Ann Johnson, Wisconsin Department of Health Services, Madison, Wisconsin

ABSTRACT

Communicable disease surveillance by the Centers for Disease Control and Prevention (CDC) relies upon reporting done by public health jurisdictions of the United States. Communicable disease reporting is facilitated by an electronic surveillance system that communicates directly with the CDC for specified diseases. Disease reports are generated when a disease is classified as “confirmed” or “probable.” These reports can then be updated with more information during case investigation, or may be determined to not be a case. Each addition of information results in a new report generated and sent to the CDC. The receiving system at the CDC is programmed to keep only the most recent report for each case.

The State of Wisconsin reconciles disease reporting between cases acknowledged by the program epidemiologist and disease reports sent to the CDC by the surveillance system on an annual basis. A SAS® 9.4 program utilizing simple DATA steps with BY-group processing is used to determine which reports were counted by the CDC. The TABULATE and REPORT procedure makes it easy to produce line-lists and simple case counts by disease. The Portable Document Format (PDF) and Excel® destination of the Output Delivery System (ODS) are used to save the output files in consumable format. These output reports are used by the epidemiologist to verify their records and counts are complete and in agreement with the CDC.

This paper is meant for all levels of SAS programmers and demonstrates basic coding techniques to perform simple data cleaning and validation, followed by removal of redundant reports and ending with the creation of five different pairs of output reports. Intermediate coding techniques include the use of macro variables to assign input and output file names and paths, Access to PC Files to import an Excel file and printing output to file using ODS destinations.

INTRODUCTION

The State of Wisconsin uses a commercial, off-the-shelf disease reporting and surveillance system product known to end users as the Wisconsin Electronic Disease Surveillance System (WEDSS). Data contained in WEDSS is stored in a SQL-Server database; SQL Server Management Studio (SSMS) is used to obtain mass amounts of data for analysis and data validation. A Transact-SQL (T-SQL) script pulls all disease reports sent to the CDC over the reporting year.

Disease reports follow the Health Level-7 (HL7) standard for messages, which is simply delimited text, organized by segments. Disease reports, or messages, are generated initially when the case is determined to be either “probable” or “confirmed” by local or state epidemiology staff. In response, the disease message is generated and sent to the CDC by WEDSS. As the case investigation proceeds, the case is updated with new information. New information may either confirm or disregard the case status. Any update made to the case is also sent to the CDC. The CDC only retains the most recent message for each case.

At the close of the year, the state epidemiologist must sign off on the counts by disease. The state epidemiologist relies on the collaboration between the program epidemiologists and the WEDSS informaticist to ensure that the counts are accurate. A SAS program is used to find the final message sent for each case, determine if it is countable, and produce line-lists and total counts by disease for the program epidemiologists to complete their reconciliation. Though the SAS code is very specific to the work of health analytics, this paper will point out some coding tips and techniques on using macro variables, BY-group processing, and logic using LAST.*variable* that could easily apply to other areas of data work.

MACRO VARIABLES MAKE THE PROGRAM REUSABLE

Reconciliation of disease counts is done on an annual basis. You can use macro variables to set the basic parameters of the program, such as report year and the date of the data to easily make the program reusable each year:

```
*%%%%% UPDATE THESE VARIABLES BEFORE RUNNING THE PROGRAM %%%%%%%%%%%%%%%%%
* DECLARE PROGRAM VARIBLES;
%LET YEAR = 2016;           /* REPORT YEAR */
%LET STARTDT    = '03JAN2016'D;   /* START OF 2016 REPORT YEAR */
%LET ENDDT      = '31DEC2016'D;   /* END OF 2016 REPORT YEAR */
%LET ASOF = '23MAY2017'D;   /* DATE OF DATA PULL */

%%%%% VARIABLES BELOW WILL UPDATE THEMSELVES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
* BUILD INPUT/OUTPUT ROOT;
%LET ROOT       = L:\Wedss\...\...\&YEAR.; /* SECURE LOCATION */
* BUILD INPUT DATA LOCATION;
%LET FROMSQL    = "&ROOT.\NEDSSpull&YEAR._&ASOFDTN..xlsx";

* FORMAT DATA DATE AS TEXT;
%LET ASOFDTN=%SYSFUNC(PUTN(&ASOF.,MMDDYY6.));
%PUT ASOFDTN=&ASOFDTN.;           /* CHECK IT */
```

Comments and consistent symbols are used to describe and denote which macro variables must be updated and which will update themselves. This convention of comments and symbols makes the SAS code easy to read, update, and makes it transportable. You should always write SAS code with the next user in mind.

SYSTEM OPTIONS TO USE WITH MACRO VARIABLES

Although macro variables can make your job easier, they can also complicate things if they are incorrect. You should consider using system options such as SYMBOLGEN or MPRINT when creating and using macro variables. Use of these options will write notes to the log that will verify the resolved value of each macro variable.

Shown below is the OPTIONS statement for invoking SYMBOLGEN, followed by a macro variable assignment and invocation of that macro variable:

```
OPTIONS SYMBOLGEN;
%LET YEAR = 2016;
%LET ROOT = L:\Wedss\PHIN and NEDSS\Annual Counts\&YEAR.;
```

The SYMBOLGEN option displays the resolved value of the macro variable YEAR to the program log upon compilation:

```
41  * DECLARE INPUT/OUTPUT ROOT;
42  %LET ROOT = L:\Wedss\PHIN and NEDSS\Annual Counts\&YEAR.;
SYMBOLGEN: Macro variable YEAR resolves to 2016
```

LOADING THE DATA INTO SAS

The IMPORT procedure is used to load the results of the T-SQL data pull into SAS from a Microsoft Excel® file:

```
PROC IMPORT OUT= ALL1 DATAFILE= &FROMSQL. DBMS=EXCELCS REPLACE;
 RANGE="Sheet1$";
 TEXTSIZE=32767;
RUN;
```

Note the use of the macro variable for the input data file location; this saves another user from needing to find where to update the file name in the program. The data source identifier, EXCELCS, denotes the use of the SAS/ACCESS Interface to PC Files connection. This method of access was chosen due to the combination a 32-bit Microsoft Office file input and the 64-bit CDC-Grantee version of SAS. The TEXTSIZE parameter is changed from the default length of 1024 to 32767 due to the long length of the HL7 messages contained in the file. Now that the data set is in the temporary work library, the next step is to process the data.

PROCESSING THE DATA

The data set contains all the messages that the surveillance system sent to the CDC during the course of the report year. Along with the messages, each observation contains an identification number unique to the patient, the date the message was sent, the case status, the general disease category and a specific disease.

USE OF LAST.*VARIABLE* TO FIND THE LAST MESSAGE

Recall that disease messages update themselves and the CDC disregards all but the last message sent for each case. You can use the automatic variable LAST.*variable* in a DATA step with a BY statement to find the last message sent for each BY group. The data set is sorted by the general disease category, the patient identification number (ID) and the date it was sent. The last message sent by patient ID for each general disease group is the output to the new data set as shown below:

```
* PROCESS THE DATA: KEEP ONLY THE LAST OBSERVATION BY INCIDENT ID;
PROC SORT DATA=ALL2;
 BY MESSAGE INCIDENTID DATECASESENTCDC;
RUN;

DATA ALL3;
 SET ALL2;
 BY MESSAGE INCIDENTID DATECASESENTCDC;
 IF LAST.INCIDENTID THEN OUTPUT;
RUN;
```

CATEGORIZING MESSAGES BY PROGRAM LOGIC

The resulting data set has the final message sent for each disease group for each patient. The logic of counting disease reports is replicated in a long DATA step that processes the data set, line-by-line, to see if observations meet criteria of being “counted” by disease. The logic of counting disease reporting is quite complex, and differs by disease group. Some disease reports are counted if the resolution status is indicated as “confirmed” or “probable”, while other disease reports are only counted if it’s marked as “confirmed”.

The bulk of the DATA step is composed of IF-THEN-ELSE statements that subdivide the data set into general disease groups, such as Arbovirus or Varicella. Within each general disease group, the resolution status is examined to further determine if the message should be counted and the HL7 itself is read to determine the appropriate reporting year of the case. The FIND and SUBSTR functions are used to extract elements from the HL7 message to further classify the message in the disease counts.

Use of Character Functions: FIND() and SUBSTR()

The FIND function is used to search for a string of characters within a text string; if it finds the text string, it will return its location as a numeric variable. If it cannot find the string, the function will return zero. In the code snippet below, FIND is used to determine if an element in the HL7 segment named “INV137” exists in the variable named “msgcontent”:

```
IF FIND(MSGCONTENT,"INV137") > 0 ...
```

The SUBSTR function extracts a sub-string within a string given the starting and optional length to extract. In the code snippet above, “INV137” is the element identifier that denotes the line that contains

the disease onset date. The actual disease onset date, underlined in Figure 1 below, is eight (8) characters long and is located 45 spaces past the location of “INV137”:

OBX|5|TS|INV137^Onset Date^2.16.840.1.114222.4.5.232||20170605|||||F

Figure 1. A sample line from an HL7 message.

If the FIND function returns a value that is greater than zero, we know that the message does indeed have a disease onset date to extract. The variable named “onset” can then be set to the sub-string within the variable “msgcontent,” starting at the location of the text string “INV137”, adding 45 spaces, and extracting exactly eight characters as shown below:

```
IF FIND(MSGCONTENT,"INV137") > 0 THEN
 ONSET=SUBSTR(MSGCONTENT,FIND(MSGCONTENT,"INV137")+45,8);
ELSE ONSET='N';
```

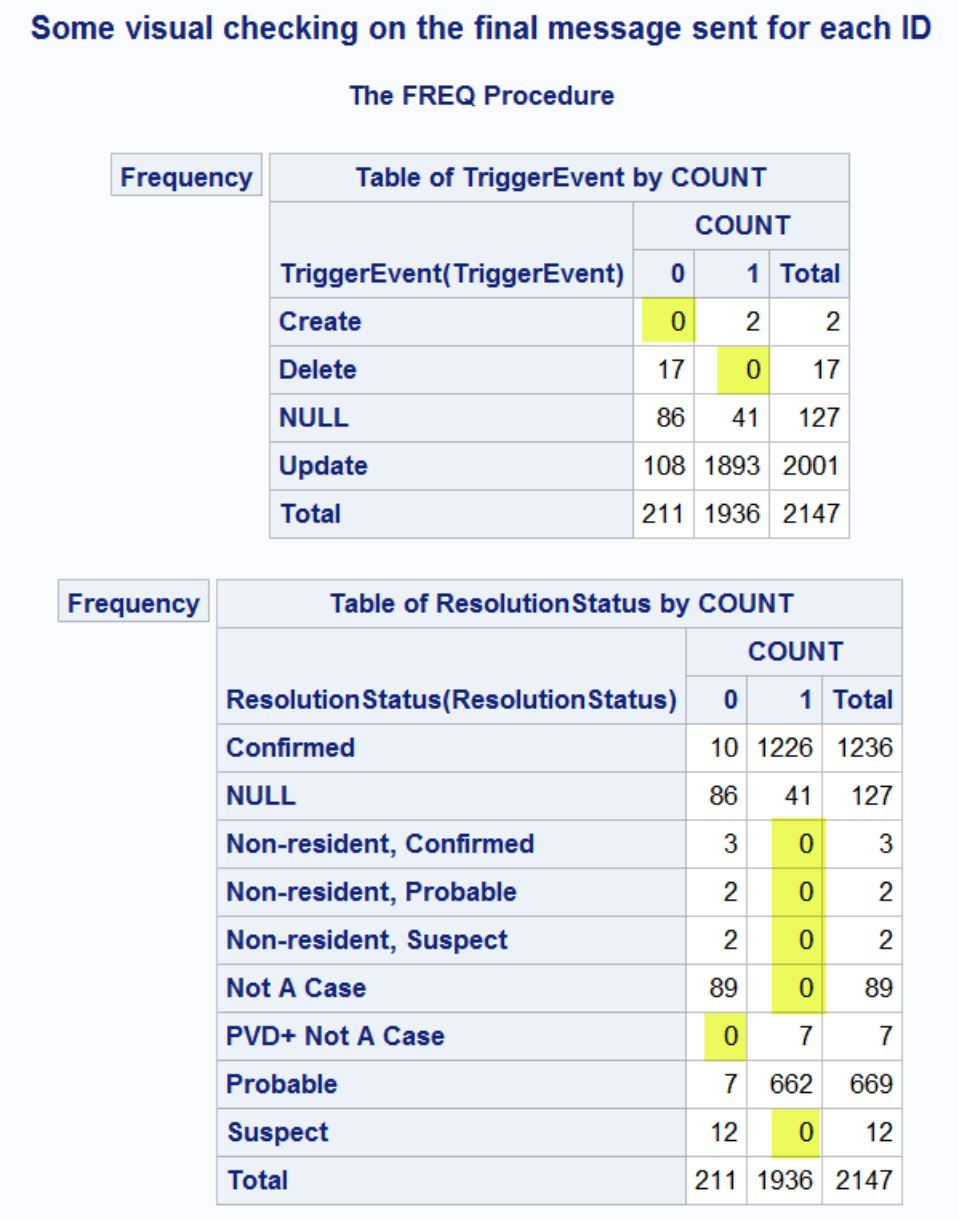
If the element denoting disease onset date is not found, i.e., the FIND function returns zero, then the variable “onset” is set to “N” in the code above.

SIMPLE DATA VALIDATION WITH THE FREQUENCY PROCEDURE

The FREQUENCY procedure is used to do some quick, visual checking of the data after all variables have been extracted. Notice how the comments document some of the conditions that should not be counted (other comments removed for brevity of this paper). Frequency tables are produced for “trigger events” and “resolution status” by “(disease) count” as shown below:

```
* CHECK CONDITIONS SHOULD NOT BE COUNTED:
      * TriggerEvent = Delete
      * ResolutionStatus other than Confirmed, Probable
  PROC FREQ DATA=ALL4;
   TITLE "Some visual checking on the final message sent for each ID";
   TABLES   TRIGGEREVENT * COUNT
                 RESOLUTIONSTATUS * COUNT / NOCUM NOPERCENT NOROW NOCOL;
  RUN;
  TITLE ;
```

The results of the PROC FREQ are shown in Figure 2; values highlighted in yellow confirm that known conditions were either not counted (count=0) or counted as expected (count=1). For example, no “Delete” messages should be counted; neither should any “Non-Resident” cases:



Some visual checking on the final message sent for each ID

The FREQ Procedure

Frequency	Table of TriggerEvent by COUNT		
	COUNT		
TriggerEvent(TriggerEvent)	0	1	Total
Create	0	2	2
Delete	17	0	17
NULL	86	41	127
Update	108	1893	2001
Total	211	1936	2147

Frequency	Table of ResolutionStatus by COUNT		
	COUNT		
ResolutionStatus(ResolutionStatus)	0	1	Total
Confirmed	10	1226	1236
NULL	86	41	127
Non-resident, Confirmed	3	0	3
Non-resident, Probable	2	0	2
Non-resident, Suspect	2	0	2
Not A Case	89	0	89
PVD+ Not A Case	0	7	7
Probable	7	662	669
Suspect	12	0	12
Total	211	1936	2147

Figure 2. Output from the Frequency procedure for visual data validation.

CREATION OF OUTPUT REPORTS

After the data have been validated, the data set is split into five main disease groups to create output reports. Required output reports include a simple count by disease and resolution status as well as an identified line-listing for the program epidemiologist to cross-reference to their documented cases.

The TABULATE procedure is used to calculate disease counts by disease sub-set and resolution status:

```
* CREATE FINAL COUNT;
ODS PDF FILE=&ARBOCOUNT.;
PROC TABULATE DATA=LINEARB1;
 CLASS DISEASE RESOLUTIONSTATUS;
 TABLE     DISEASE = 'Disease'
           ALL='Total'*[STYLE=[FONT_WEIGHT=BOLD]],
    /* add count at bottom, bold */
```

```
          RESOLUTIONSTATUS='Resolution Status'*(N='')
     ALL='Total'*[STYLE=[FONT_WEIGHT=BOLD]];
 TITLE1 "WI Arbo Cases &YEAR.";
 TITLE2 "Annual Count--as of: &ASOFDT.";
RUN;
ODS PDF CLOSE;
```

Notice the use of macro variables in the code above for the output filename, the year, and data as-of date in the title. This allows you to specify these variables at the beginning of the program and invoke them later. The CLASS statement allows the counts to be tabulated by disease and resolution status. In the TABLE statement, the ALL keyword creates an additional table dimension with totals.

The PROC TABULATE step begins and ends with an Output Delivery System (ODS) statement. Some authors describe ODS as “wrapper”: the ODS statement wraps around the procedure code, ending with the CLOSE statement. You can use the ODS PDF destination to quickly and easily create PDF output. The results are shown in Figure 3 below.

WI Arbo Cases 2016
Annual Count--as of: May 23, 2017

	Resolution Status		Total
	Confirmed	Probable	N
Disease			
ARBOVIRAL ILLNESS, CALIFORNIA, NON-NEUROINVASIVE	2	.	**2**
ARBOVIRAL ILLNESS, CHIKUNGUNYA	1	1	**2**
ARBOVIRAL ILLNESS, DENGUE	3	4	**7**
ARBOVIRAL ILLNESS, FLAVIVIRUS	1	.	**1**
ARBOVIRAL ILLNESS, JAMESTOWN CANYON, NEUROINVASIVE	3	.	**3**
ARBOVIRAL ILLNESS, JAMESTOWN CANYON, NON-NEUROINVASIVE	3	.	**3**
ARBOVIRAL ILLNESS, LA CROSSE ENCEPHALITIS, NEUROINVASIVE	2	.	**2**
ARBOVIRAL ILLNESS, LA CROSSE ENCEPHALITIS, NON-NEUROINVASIVE	1	1	**2**
ARBOVIRAL ILLNESS, POWASSAN, NEUROINVASIVE	3	.	**3**
ARBOVIRAL ILLNESS, POWASSAN, NON-NEUROINVASIVE	1	.	**1**
ARBOVIRAL ILLNESS, WEST NILE VIRUS, NEUROINVASIVE	8	1	**9**
ARBOVIRAL ILLNESS, WEST NILE VIRUS, NON-NEUROINVASIVE	4	.	**4**
ARBOVIRAL ILLNESS, ZIKA VIRUS DISEASE	60	.	**60**
ARBOVIRAL ILLNESS, ZIKA VIRUS INFECTION	3	.	**3**
Total	**95**	**7**	**102**

Figure 3. Results of PROC TABULATE; the data is also stored as a PDF.
Disclaimer: Counts as of May 23, 2017, may not reflect final 2016 counts submitted to CDC.

Similarly, the REPORT procedure is used to create line-lists for each disease. Since the line-lists are essentially just the output of the countable cases in the data set by the disease group, the code is relatively simple, using just the COLUMN statement to declare which variables to include. The code to produce a line-list for varicella is shown below:

```
* MAKE LINE LIST FOR VARICELLA;
ODS EXCEL FILE=&VARILINEX.;
 PROC REPORT DATA=LINEVAR1 NOWD;
```

```
    COLUMN DATECASESENTCDC LASTNAME FIRSTNAME INCIDENTID RESOLUTIONSTATUS;
    RUN;
ODS EXCEL CLOSE;
```

Macro variables are again used for the output filename. Due to the large size of the line-lists and the need of the program epidemiologist to compare to his or her files, the output is sent directly to Excel using the ODS EXCEL destination.

CONCLUSION

Data work behind communicable disease reporting requires knowledge specific to the discipline, including disease message content and operating logic of the surveillance system. Complex logic, unique to any discipline, can easily be documented and applied to data with SAS.

The power of IF-THEN-ELSE statements, combined with BY-group processing and automatic variables such as LAST.*variable*, makes logic easy to apply to data. Character functions, such as FIND() and SUBSTR(), are very useful to determine existence of a string and extract parts of a string. Simple procedures such as PROC FREQ, PROC TABULATE, and PROC REPORT can be used for data validation and report creation.

This paper also demonstrated some tips on making your programs reusable and easy to use and troubleshoot. The use of macro variables can make your code reusable from year to year; using comments and consistent symbols to denote which macro variables need updating, and which will update all by themselves, makes your program transportable from user to user. The system option, SYMBOLGEN, is useful in troubleshooting and understanding the resolution of macro variables. ODS destinations make saving to PC and other file formats quick and simple.

SAS is a powerful tool that can be used for a variety of data validation, reporting, and utility tasks, regardless of the discipline. The tips and techniques presented in this paper could be utilized by programmers in various applications.

REFERENCES

Cody, Ron. 2004. *SAS® Functions by Example*. Cary, NC: SAS Institute Inc.

Delaney, Kevin and Carpenter, Arthur. 2004. "SAS® Macro: Symbols of Frustration? %Let us help! A Guide to Debugging Macros." Proceedings of the 29th SAS Users Group International Meeting. Cary, NC USA. SAS Institute Inc. Available at http://www2.sas.com/proceedings/sugi29/128-29.pdf

Delwiche, Lora and Slaughter, Susan. 1998. *The Little SAS® Book*. 2nd ed. Cary, NC: SAS Institute Inc.

Health Level 7. 2017. "Health Level 7." Accessed July 3, 2017. https://en.wikipedia.org/wiki/Health_Level_7.

Hemedinger, Chris. The SAS Dummy. 2012. "The top gotchas when moving to64-bit SAS for Windows." Accessed February 13, 2017. Available at http://blogs.sas.com/content/sasdummy/2012/05/01/64-bit-gotchas/

SAS Institute Inc. 2011. *SAS® Certification Prep Guide: Advanced Programming for SAS®9, Third Edition*. Cary, NC: SAS Institute Inc.

SAS Support. 2012. "Problem Note 46472: Character strings can be truncated at 255 or 1024 characters when importing Excel files into SAS®." Accessed March 4, 2017. Available at http://support.sas.com/kb/46/472.html .

SAS Support. 2014. "Problem Note 52649: Using the IMPORT and EXPORT procedures with Microsoft Excel and Microsoft Access fails and returns a "Class not registered" error." Accessed July 7, 2017. Available at http://support.sas.com/kb/52/649.html .

RECOMMENDED READING

- *The Little SAS® Book*
- *SAS® Certification Prep Guide: Base Programming for SAS®9*
- *SAS® Certification Prep Guide: Advanced Programming for SAS®9*

CONTACT INFORMATION

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

Misty Ann Johnson
Wisconsin Department of Health Services
(608) 267-7812
MistyA.Johnson@wi.gov
https://www.linkedin.com/in/mistyjohnson4/