Putting the Meta into the Data: Managing Data Processing for a Large Scale CDC Surveillance Project with SAS®

Louise S. Hadden, Abt Associates Inc.

ABSTRACT

There are myriad epidemiological and surveillance studies ongoing due to the pervasive COVID-19 pandemic, often embodying the definition of “big data” with thousands of participants, variables, and lab samples. Data can be derived from many different streams in a given study, for example: REDCap software, electronic medical records (EMR), chart abstraction, laboratory records, etc. Different contractors can be managing distinct aspects of the same project, the data is changing minute to minute, and the deliveries are required at a fast and furious pace. Wrangling all the different data sources requires robust data management routines, and SAS® can help, with tools to obtain data via APIs and PROC HTTP, metadata resources, and programming techniques. This paper and presentation will outline best practices for managing multiple aspects of large scale CDC surveillance projects, using SAS.

INTRODUCTION

Any big study involves data acquisition, deliverable schedule, data and programming management concerns, ever-changing data structures, the dangers of programming “silos”, data and code quality concerns, tracking and communication, and most of all time concerns. This paper will look at two large scale, multi-site CDC surveillance studies involving COVID and flu testing, and other health metrics. Both projects are with the CDC, involve distinct aspects of COVID 19, and the CDC is heavily involved in analytic planning and specifications. The EMR project had monthly deliveries, while the REDCAP project has weekly to biweekly deliveries, and in rare cases more than one delivery a week. Results from the REDCAP project have gone directly to CDC and the White House, so the stakes are extremely high in terms of quality assurance and rapid turnaround. The EMR project was designed to assess the effect of COVID 19 on pregnant women and children and had three sites across the United States. The REDCap project (one of a number of different projects in house) was designed to monitor the ongoing effects of the COVID 19 pandemic on first responders in the US, and has numerous sites and REDCaps.

If you are not familiar with REDCap, it is a browser-based, metadata-driven EDC software and workflow methodology for designing clinical and translational research databases. Many institutions use REDCap to develop online surveys and databases. There are millions of projects using this resource. It definitely has a pretty clunky interface with regards to SAS, a little better with R, but we endeavor to make it work.

SCENARIOS

We will discuss how we use SAS to help wrangle these COVID related data bases. For scenario 1, we will dive into data dictionary read in and uses in labeling, formatting, etc. For scenario 2, we will look at comparisons of data dictionaries and formats.

As noted above, these projects have different study designs: Scenario 1 has a combination of REDCap survey data and EMR data, while Scenario 2 has a combination of REDCap survey data and serology test results. These two projects have common and differing data management and SAS programming challenges. Commonalities for both of these scenarios are: a vast amount of data from different data sources, multiple data areas, and multiple analytic agendas; quick turnaround deliveries (monthly on one project, weekly+ on another project); and very high visibility (results going into White House briefs) which necessitates actionable quality assurance.

A cornerstone for big projects like these is data management. My company mandates data management and code quality assurance plans for every project. Key data management tools that we use on a daily basis include: data receipt and delivery logs; shared process trackers; details of the data management and code quality plans, staffing plans, and last, but not least, data dictionaries. We will drill down on the use of data dictionaries to drive much of our processing for both projects, creating files and processes

from the metadata contained in data dictionaries and files to drive the provision of variable and value labels, order files, assign variable prefixes and split out subsets of files, as well as drive the creation of format assignment statements and reporting for file subsets.

SCENARIO 1

A data dictionary for a file based on Electronic Medical Records (EMR) contains variables which represent an unknown number of COVID-19 tests for an unknown number of infants – there is no way to know in advance how many iterations of the COVID test variable will exist in the actual data file from medical entities. In addition, variables in this file may exist for three distinct groups (pregnant women, postpartum women, and infants), with PR, PP and IN prefixes, respectively. We will demonstrate how to ingest data dictionaries to collect metadata for this project allowing identification of groups for processing, and drive label (and value label) description creation for iterated (and other) labels using SAS functions, PROC FORMAT, AND DATA step processing, as well as other utilities.

Data dictionaries that are constructed via a collaboration between the CDC, study sites, and project staff are provided to study sites, who return extracted electronic medical record (EMR) data to us for processing.

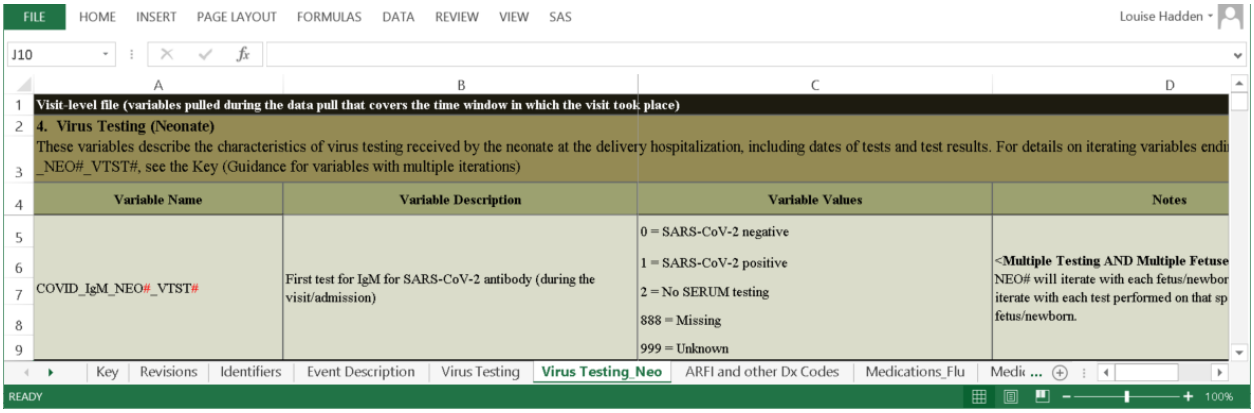
There are two separate data dictionaries used for this project, for person-level and visit-level variables. The data dictionaries in use for the study have multiple tables and thousands of variables. Individual variables have up to two # signs (iteration flags), appear in a single row in the data dictionaries, and may occur more than sixty times. Variables to the right of the screenshot shown below indicate whether the variable is present for pregnant women, postpartum women, and/or infants. There are six data files delivered from each site – 2 data dictionaries x 3 populations. We need variable and value labels for each of the six data files, so we need to build LABEL statements, a format library, and format assignments. We receive data on an ongoing basis (monthly), so the solution needs to be as efficient and data driven as possible. Iterated variables, which appear with pound signs in the data dictionaries, appear with numbers in the actual data and we need a programmatic answer to match the variables in the data dictionaries to the variables in the data files.

Solutions presented include ingesting data dictionaries to collect metadata allowing identification of groups for processing, and drive processing with information derived from data dictionaries.

Data dictionary read in

We read in the separate tabs in the workbook in the data dictionary and collect information to be used for variable names, labels, and value label, as well as other information used solely for data processing. Note the red # signs, the variable description, which will be turned into a label, and the variable values, which will be turned into value labels (a format) in Figure 1 shown below.

Challenges for read in include: multiple tabs in each Excel workbook: different starting rows in each tab, requiring reading in specific ranges; clean up of special characters (tabs, carriage returns), disaggregation of some fields, making sure errors encountered in read in are addressed; and the need to collect information on two levels (variable information vs value label information).



	A	B	C	D
1	Visit-level file (variables pulled during the data pull that covers the time window in which the visit took place)			
2	4. Virus Testing (Neonate)			
3	These variables describe the characteristics of virus testing received by the neonate at the delivery hospitalization, including dates of tests and test results. For details on iterating variables endi _NEO#_VTST#, see the Key (Guidance for variables with multiple iterations)			
4	Variable Name	Variable Description	Variable Values	Notes
5–9	COVID_IgM_NEO#_VTST#	First test for IgM for SARS-CoV-2 antibody (during the visit/admission)	0 = SARS-CoV-2 negative 1 = SARS-CoV-2 positive 2 = No SERUM testing 888 = Missing 999 = Unknown	<Multiple Testing AND Multiple Fetuse NEO# will iterate with each fetus/newbor iterate with each test performed on that sp fetus/newborn.

Figure 1. Sample Data Dictionary

Since multiple tabs are read in with the same structure on each tab, we take advantage of macro processing to read in our file metadata. We use ranges in our PROC IMPORT, storing valuable metadata about our incoming data. Note that it is possible to have two level range names, similar to library names, by preceding the range with the tab name and separated by a dot.

```
************************************************************;
*** Import Personal Data Dictionary one tab at a time    ***;
************************************************************;

%macro imptabs(tabn=1, tabnm=identifiers, intab=Identifiers, startrow=10, endcol=H);

       proc import dbms=xlsx out = temp datafile = " \file.xlsx" replace;
              RANGE="&intab.$A&startrow.:&endcol.999";
              getnames=YES;
run;

. . .
```

First, the LENGTH function is used to calculate the length of “variable”. The length of variable names is limited to thirty-two characters and the name an of iterated variable may exceed the limits. The data dictionary is a living document, and if any overlong variables that exist once prefixes and iterated counts are added to their base, the spelling is adjusted. Identification variables are exempt from prefixes. When a file is first processed, variables, with the exception of ID variables, have prefixes added, using the CATS function. Additionally, label strings are created by concatenating prefixes and existing variable descriptions using the CATX function.

```
data labels&tabn.;
         length label labelstr $ 300 variable_type $ 8;
         set &tabnm (keep=variable_: pw_preg pw_pp inf
         where=(variable_name ne '' and variable_description ne ''));
              label=catx(": ","&tabnm.",variable_description);
              labelstr=cats(variable_name,'="',label,'"');

              variable_length=length(variable_name);
              length_flag=(variable_length+7 GT 32);
              label variable_length="Length of Variable"
             length_flag="Current Variable Length + 7 exceeds 32";
```

In preparation for iteration, we use the INDEXC function to find the location (or existence) of # signs. If we wanted to look for a string (as we do later on) we can use the INDEX function. We use COUNTC to count how many times an iteration flag occurs in a variable name. Multiple iterations of a variable can occur if, for example, multiple neonates in a single pregnancy have multiple virus tests.

```
              /* find out the # of iterations within a variable name */
              iteration_flag=(indexc(variable_name,'#') gt 0);
              iteration_count=countc(variable_name,'#');

              label iteration_flag="Binary: Variable iterations"
              iteration_count="# of iteration points within variable name";
run;

%mend;

%imptabs(tabn=1, tabnm=Identifiers, intab=Identifiers, startrow=4, endcol=H);
```

Iteration

It is relatively simple to replace a single iterator, in this case, a #, in a variable name. It is more complicated to replace two or more iterators, especially if you do not know how many iterations there are. SAS functions process one variable transformation at a time – that is, they stop after completing a single operation on a string. After confirming the existence of a # sign in a variable and finding its position using the INDEX function, we then use the SUBSTR function to replace the # using a do loop, outputting additional label records for each iteration.

As noted above, we use the COUNTC function to discover how many #s exist in a variable name. You can use functions to discover the number of iterations needed as well in the actual data – including the REVERSE and ANYNUM functions – in the actual data. Additionally, the iteration numbers are added to the label strings using CAT functions. Multiple supplemental label records are created until no more # signs appear in the variable names.

We have thousands of variables, and multiple occurrences of iteration and the need to replace (via the SUBSTR function) items of different lengths. We quickly realized we would need to employ macro loops to manage the different requirements for a number of situations (number of iterations, the "base" of the variables needing to be iterated, one or two iteration symbols, and substring length). Sample code for a simple loop and more complex loop follow below.

Simple loop

```
%macro do_list1(maxiter=1,suffix=neo);

%do i=1 %to &maxiter;

data iter&suffix.1_&i (drop=loc);
    length variable $ 50 labelstr $ 300;
    set formats0 (where=(count(variable,"#")=1 and
index(variable,"IDENTIFIER#")>0));

    *get the first indexed # location;
    loc=index(variable,"#");

    substr(variable,loc,1)="&i";
    labelstr=catt(labelstr," #&i");

run;

%END;

%MEND DO_LIST1;
```

Complex loop

```
%macro do_list2(maxiter=20,suffix=vtst);

%if &maxiter le 9 %then %do i=1 %to &maxiter;

data iter&suffix.1_&i (drop=loc);
    length variable $ 50 labelstr $ 300;
    set formats0 (where=(count(variable,"#")=1 and
index(variable,"VTST#")>0));

    *get the first indexed # location;
    loc=index(variable,"#");

    substr(variable,loc,1)="&i";
```

```
    labelstr=catt(labelstr," #&i");

run;

proc print data=iter&suffix.1_&i (obs=5) noobs;
    var variable labelstr;
run;

%END;

%if &maxiter gt 9 %then %do;

%do i=1 %to 9;

data iter&suffix.1_&i (drop=loc);
    length variable $ 50 labelstr $ 300;
    set formats0 (where=(count(variable,"#")=1 and
index(variable,"VTST#")>0));

    *get the first indexed # location;
    loc=index(variable,"#");

    substr(variable,loc,1)="&i";
    labelstr=catt(labelstr," #&i");

run;

proc print data=iter&suffix.1_&i (obs=5) noobs;
    var variable labelstr;
run;

%END;

%do i=10 %to &maxiter;

data iter&suffix.1_&i (drop=loc);
    length variable $ 50 labelstr $ 300;
    set formats0 (where=(count(variable,"#")=1 and
index(variable,"VTST#")>0));

    *get the first indexed # location;
    loc=index(variable,"#");

    substr(variable,loc,2)="&i";
    labelstr=catt(labelstr," #&i");

run;

proc print data=iter&suffix.1_&i (obs=5) noobs;
    var variable labelstr;
run;

%END;

%END;

%MEND DO_LIST2;
```

Practical applications

Variable Labels

The iteration techniques discussed above are employed in several different scenarios: data quality checks, creating variable labels, creating format assignment statements, driving range checks, and producing missingness reports. Below follow snippets of code to create a data driven variable label statement. label strings are created by concatenating prefixes and existing variable descriptions using the CATX function, The CATX function is used to add information as a prefix to the label, such as the month of data collection or the tab the variable came from in the data dictionary. The labelstr variable is a sentence that applies a variable label to a variable. When put out to a flat file, it can be included to label variables in a data set.

Assign a filename for the LABEL statement:

```
filename label1 ".\&short._Labels.txt";
```

Create iterations of variables with # signs using macro loops described above:

```
%do_list1(maxiter=3,suffix=id);
%do_list2(maxiter=4,suffix=vtst); . . .
```

Add iterated records created by the do loops together:

```
data expand_labels;
    set iterid: itervtst: . . ._ ;
run;
```

Add iterated records to the records that did not require iteration:

```
data labels;
    length variable $ 32;
    set labels0 (where=(index(variable,"#")=0))
        expand_labels (where=(index(variable,"#")=0))
        ;
run;
```

Output the LABEL statement:

```
data tolabel;
    retain VARIABLE_CATEGORY VARIABLE LABELSTR
           VARIABLE_TYPE VARIABLE_LENGTH
           PW_PREG PW_PP INF ITERATION_COUNT INLABELS INPOS NUM ;
    file label1 lrecl=400;
    set matchtest ( keep= VARIABLE_CATEGORY VARIABLE LABELSTR
                          VARIABLE_TYPE
                          VARIABLE_LENGTH PW_PREG PW_PP INF
                          PRIORITY_VARIABLE
                          MISSING_NOT_OK ITERATION_COUNT
                          INLABELS INPOS NUM DD_ORDER);
    by NUM;
    STATEMENT=compbl(cats(variable,'="',labelstr,'"'));
    if inlabels=1 and inpos=1 then put statement;
run;
```

Include the LABELstatement:

```
filename label1 ".\&short._Labels.txt";
filename retain1 ".\&short._retain.txt";
```

```
run;

data &outfi. (label="Labeled &short");
    retain
        %include retain1;
    ;
    set &infi.;
    label
        %include label1;
    ;
run;
```

Figure 2 is a snippet of the text file included to produce variable labels.

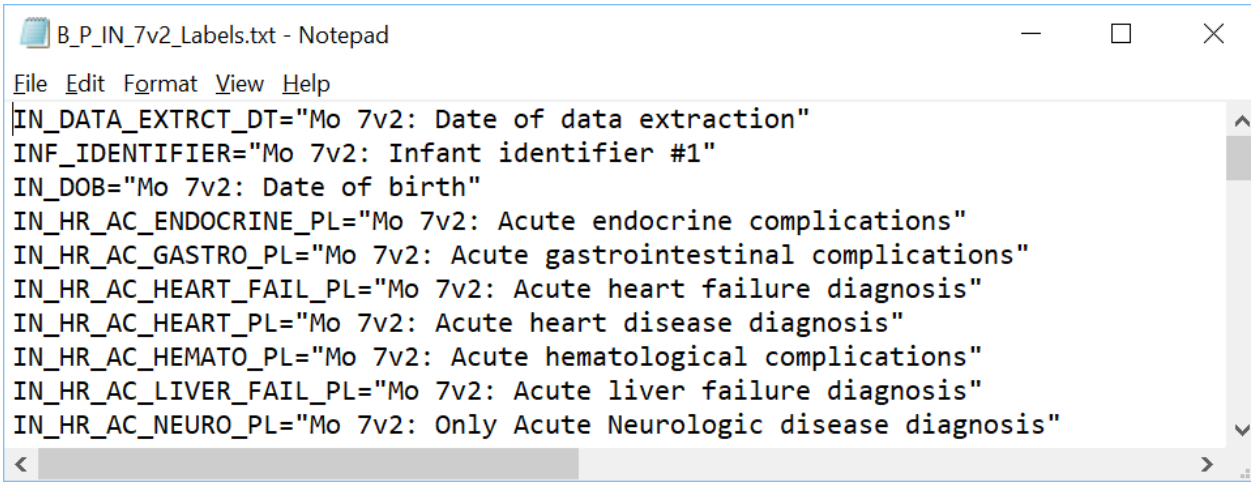
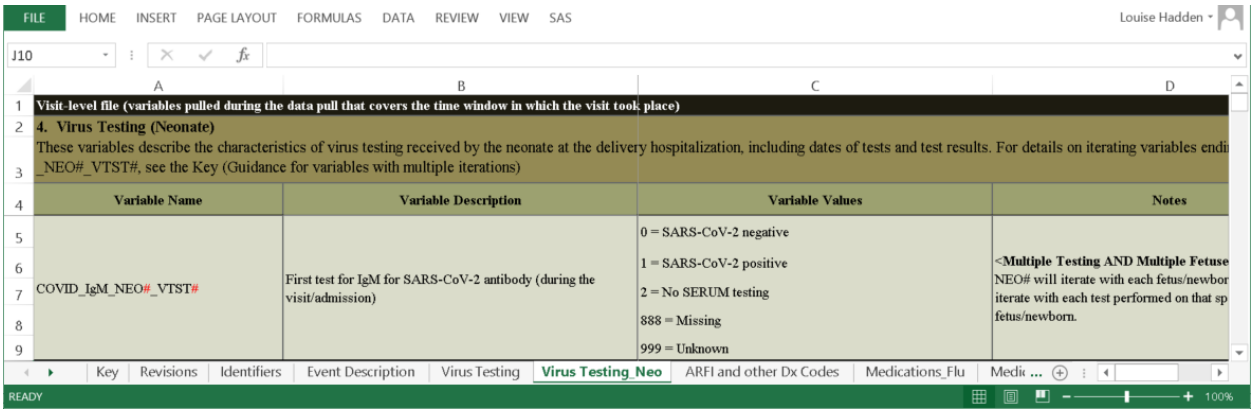


Figure 2. Sample Label Statement Text File

Value Labels (FORMATS)

Let us review the data dictionary screenshot again, focusing on the variable values columns and rows. Note that the variable name, variable description and notes fields are single rows, but the value labels are separate rows. Originally, the value labels were on single rows in the Excel data dictionaries but that made parsing the variable values field incredibly difficult so we redesigned the data dictionary to have a separate row for each variable value. When we read in the rows shown, we get five rows of data. Only the first line of data contains variable name, variable description, and notes, while each value label is on a separate row. To produce variable labels, we remove rows in blanks in the variable name field as we saw above, for value labels, we need to fill in those missing cells programmatically.



Visit-level file (variables pulled during the data pull that covers the time window in which the visit took place)			
4. Virus Testing (Neonate)			
These variables describe the characteristics of virus testing received by the neonate at the delivery hospitalization, including dates of tests and test results. For details on iterating variables endi _NEO#_VTST#, see the Key (Guidance for variables with multiple iterations)			
Variable Name	**Variable Description**	**Variable Values**	**Notes**
COVID_IgM_NEO#_VTST#	First test for IgM for SARS-CoV-2 antibody (during the visit/admission)	0 = SARS-CoV-2 negative 1 = SARS-CoV-2 positive 2 = No SERUM testing 888 = Missing 999 = Unknown	<**Multiple Testing AND Multiple Fetuse** NEO# will iterate with each fetus/newbor iterate with each test performed on that sp fetus/newborn.

```
*************************************************************;
*** Import Personal Data Dictionary one tab at a time    ***;
*************************************************************;

%macro imptabs(tabn=1, tabnm=identifiers, intab=Identifiers, startrow=10, endcol=H);

proc import dbms=xlsx out = temp datafile = " \file.xlsx" replace;
               RANGE="&intab.$A&startrow.:&endcol.999";
               getnames=YES;
run;

data labels&tabn.;
    length variable_name $ 32 variable_values_edited varlabel $ 300 start $ 8
           variable_type $ 8 ;
    set &tabnm (keep=variable_: pw_preg pw_pp inf
        where=(variable_values ne '' or variable_name ne ''));

/* replace special characters such as tabs with blank and remove extraneous blanks */
    variable_values_edited=translate(variable_values,' ','09'x);
    variable_values_edited=translate(variable_values_edited,' ','0A'x);
    variable_values_edited=translate(variable_values_edited,' ','0D'x);
    variable_values_edited=compbl(variable_values_edited);

    /* create start and label variables for a start on building formats */
    if variable_type not in('ID','DATE') then do;
        start=scan(variable_values_edited,1,"=");
        varlabel=scan(variable_values_edited,2,"=");
    end;
```

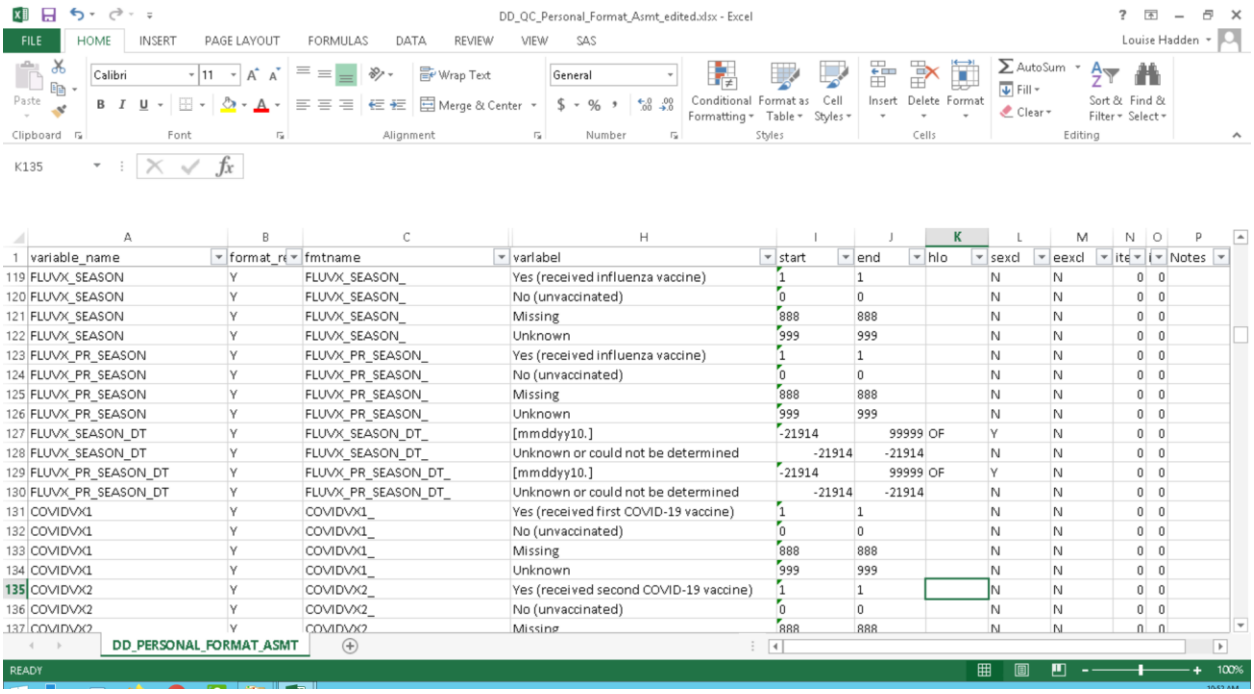
This is a similar read in as used for variable labels above, but there are key differences. For example, we are keeping records with either the variable name present OR variable value labels present. The reason for this is that some variables do not have formats assigned, or they have a blank line where a range should be specified, so if there is a variable name only, we keep the record. This will be addressed in a manual review step later, and if required, the data dictionary corrected.

In the code snippet below, we fill in the missing rows with the RETAIN statement, and create a format name, among other things. We then export the temporary data set to a spreadsheet for manual checks and adjustments.

```
    retain _variable_type;

    if not missing(variable_type) then _variable_type=variable_type;
    else variable_type=_variable_type;
    drop _variable_type;
    formatstr=variable_values_edited;

    if variable_name ne '' then fmtname=cats(variable_name,'_');
```



	A	B	C	H	I	J	K	L	M	N	O	P
1	variable_name	format_r	fmtname	varlabel	start	end	hlo	sexcl	eexcl	ite	i	Notes
119	FLUVX_SEASON	Y	FLUVX_SEASON_	Yes (received influenza vaccine)	1	1		N	N	0	0	
120	FLUVX_SEASON	Y	FLUVX_SEASON_	No (unvaccinated)	0	0		N	N	0	0	
121	FLUVX_SEASON	Y	FLUVX_SEASON_	Missing	888	888		N	N	0	0	
122	FLUVX_SEASON	Y	FLUVX_SEASON_	Unknown	999	999		N	N	0	0	
123	FLUVX_PR_SEASON	Y	FLUVX_PR_SEASON_	Yes (received influenza vaccine)	1	1		N	N	0	0	
124	FLUVX_PR_SEASON	Y	FLUVX_PR_SEASON_	No (unvaccinated)	0	0		N	N	0	0	
125	FLUVX_PR_SEASON	Y	FLUVX_PR_SEASON_	Missing	888	888		N	N	0	0	
126	FLUVX_PR_SEASON	Y	FLUVX_PR_SEASON_	Unknown	999	999		N	N	0	0	
127	FLUVX_SEASON_DT	Y	FLUVX_SEASON_DT_	[mmddyy10.]	-21914	99999	OF	Y	N	0	0	
128	FLUVX_SEASON_DT	Y	FLUVX_SEASON_DT_	Unknown or could not be determined	-21914	-21914		N	N	0	0	
129	FLUVX_PR_SEASON_DT	Y	FLUVX_PR_SEASON_DT_	[mmddyy10.]	-21914	99999	OF	Y	N	0	0	
130	FLUVX_PR_SEASON_DT	Y	FLUVX_PR_SEASON_DT_	Unknown or could not be determined	-21914	-21914		N	N	0	0	
131	COVIDVX1	Y	COVIDVX1_	Yes (received first COVID-19 vaccine)	1	1		N	N	0	0	
132	COVIDVX1	Y	COVIDVX1_	No (unvaccinated)	0	0		N	N	0	0	
133	COVIDVX1	Y	COVIDVX1_	Missing	888	888		N	N	0	0	
134	COVIDVX1	Y	COVIDVX1_	Unknown	999	999		N	N	0	0	
135	COVIDVX2	Y	COVIDVX2_	Yes (received second COVID-19 vaccine)	1	1		N	N	0	0	
136	COVIDVX2	Y	COVIDVX2_	No (unvaccinated)	0	0		N	N	0	0	
137	COVIDVX2	Y	COVIDVX2	Missing	888	888		N	N	0	0	

The screenshot above shows some (but not all) of the columns used to create complex formats from a data set. This spreadsheet output is reviewed carefully, any corrections made, and then it is imported into a SAS data set for use in creating formats and assignment statements.

Building a format library programmatically

We are keeping variable name in our SAS data set derived from the spreadsheet above, so that we can create FORMAT statements as well as a format catalog. For expediency, we name the format name with the variable name with a trailing underscore, stripping the iterator pound signs. To build a library, you need the following three fields:

FMTNAME - format name

LABEL - value label

START - start of a range or value

Additional fields that are used in our processing are:

Variable_name – used to build format assignment statements

Format_required – some variables do not require a format

END – end of a range

HLO – specialized formats – high, low, other

SEXCL (exclude the start of the range)

EEXCL (exclude the end of the range)

We have some complex formats for dates, ranges and nested formats which require END, HLO, SEXCL and EEXCL.

How do FORMATS work?

The best way to figure out how formats work is to analyze them. The client for this project wanted to assign special date values for missing values. We create a small data set with the special dates and explore to see how this complex format looks in various forms. The same technique can be used to look

at ranges. What we are trying to achieve is an input data set which looks like what SAS expects under all conditions.

```
data temp;
    d1='01jan1900'd; d2='01jan1960'd; d3=today(); d4='01jan1940'd;
run;

proc print data=temp;
run;

proc print data=temp;
format d1 d2 d3 d4 mmddyy10.;
run;

proc format fmtlib;
     value foo '01jan1900'd='Invalid'
               '01jan1940'd='Still in'
               '01jan1960'd='SAS zero'
               other=[mmddyy10.];
run;

proc print data=temp;
format d1 d2 d3 d4 foo.;
run;

proc format cntlout=foo2;
run;

proc print data=foo2;
run;
```

Below we see the number representation of the special dates, followed by their formatted version (SAS data format), followed by our user-defined format.

```
Obs      d1      d2      d3       d4

  1     -21914     0    22475    -7305

Obs      d1            d2            d3            d4

  1     01/01/1900    01/01/1960    07/14/2021    01/01/1940

Obs      d1          d2           d3           d4

  1     Invalid    SAS zero    07/14/2021    Still in
```

Below follows the result of PROC FMTLIB, showing how SAS represents the special date format in printed form. Note the other – this is a nested format indicating that any “other” dates should appear in MMDDYY10. format. You can use any SAS-supplied or user-created format as long as the program has access to where the format is stored.

```
-------------------------------------------------------------------------
|       FORMAT NAME: FOO      LENGTH:   10   NUMBER OF VALUES:    4       |
|   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH:  10  FUZZ: STD      |
|-----------------------------------------------------------------------|
|START           |END             |LABEL  (VER. V7|V8   14JUL2021:13:08:59)|
|----------------+----------------+---------------------------------------|
|          -21914|          -21914|Invalid                                |
|           -7305|           -7305|Still in                               |
|               0|               0|SAS zero                               |
|**OTHER**       |**OTHER**       |[MMDDYY10.]                            |
-------------------------------------------------------------------------
```

PROC FORMAT CNTLOUT produces a SAS data set from a format catalog file, which produces yet another vision of the same format. It is this version that we need to reproduce in order to use metadata to create format catalogs.

Using PROC FORMAT CNTLIN

Here we create the input to PROC FORMAT CNTLIN from our data dictionary import file, making adjustments to conform to SAS' requirements for CNTLIN data sets.

01.1_Person_CNTLIN.sas - Notepad

File Edit Format View Help

```
*****************************************************************************************
*** Create CNTLIN file
*****************************************************************************************

data personformats_&procmo (keep=variable_name fmtname start end
    label type hlo sexcl eexcl iterated);
    length fmtname $32 type $1 start $14 label $300;
    set person_cntlin (where=(format_req ne 'N') rename=(varlabel=label));
    if variable_type in('CHAR','ID') then type='c';
    else type='n';
    iterated=(index(variable_name,'#')>0);

label fmtname = 'Name of Format'
      start = 'Start of Range for Format'
      end = 'End of Range for Format'
      sexcl = 'Starting value excluded from Range'
      eexcl = 'Ending value excluded from Range'
          label = 'Label for Format'
          type = 'Type of Format'
          hlo = 'High-Low-Other flag'
          iterated = 'Iterated variable' ;

run;
```

Ln 151, Col 27

```
proc format library=library.personformats cntlin=personformats_&procmo fmtlib ;
run;
```

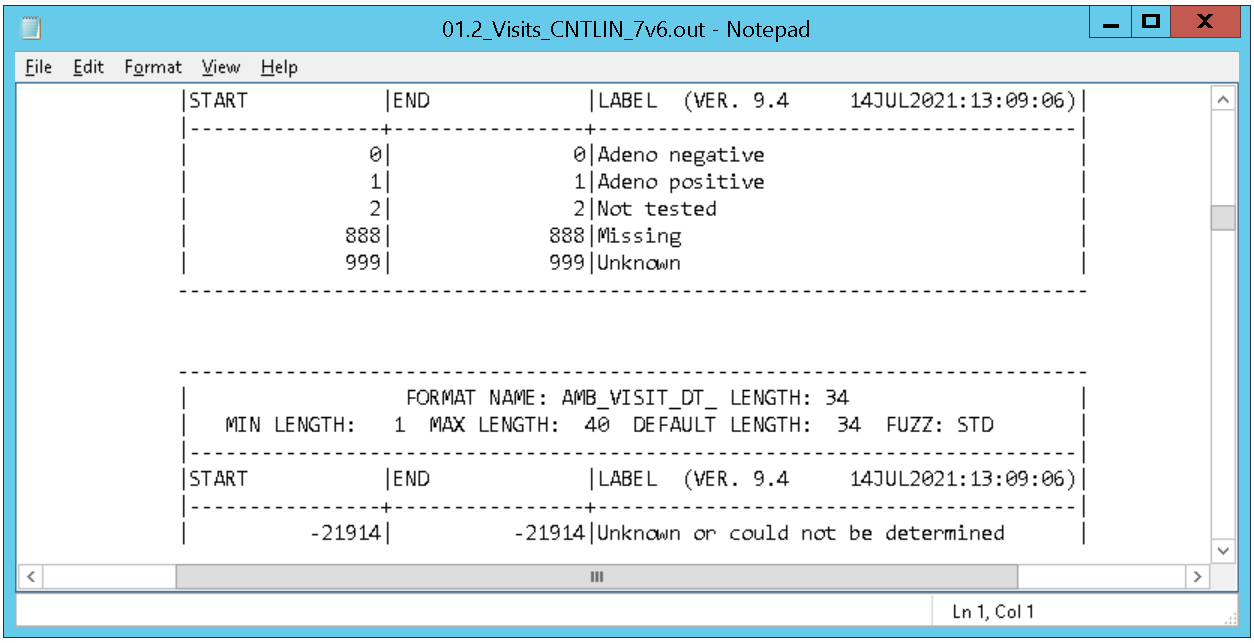
01.1_Person_CNTLIN.lst - Notepad

File Edit Format View Help

```
                                                                                                  D  L
          F                                                D                                      D  A  A
          M                                                E   L         P       N            D  I  T  N
          T         S                              L       F   E         R       O    S  E     E  G  A  G
          N         T                              A       A   N    F    E  M  F  E  T  E  E     C  3  T  U
      O   A         A              E               B    M  M   U   G    U    F  U  I  D  Y  X  X  H  S  S  Y  A
      b   M         R              N               E    I  A   L   T    Z    I  L  L  I  P  C  C  L  E  E  P  G
      s   E         T              D               L    N  X   T   H    Z    X  T  L  T  E  L  L  O  P  P  E  E

      1  FOO          -21914          -21914  Invalid    1  40  10  10  1E-12    0    0  N  N  N
      2  FOO           -7305           -7305  Still in   1  40  10  10  1E-12    0    0  N  N  N
      3  FOO               0               0  SAS zero   1  40  10  10  1E-12    0    0  N  N  N
      4  FOO  **OTHER**        **OTHER**       MMDDYY10.  1  40  10  10  1E-12    0    0  N  N  N  OF
```

Ln 44, Col 1



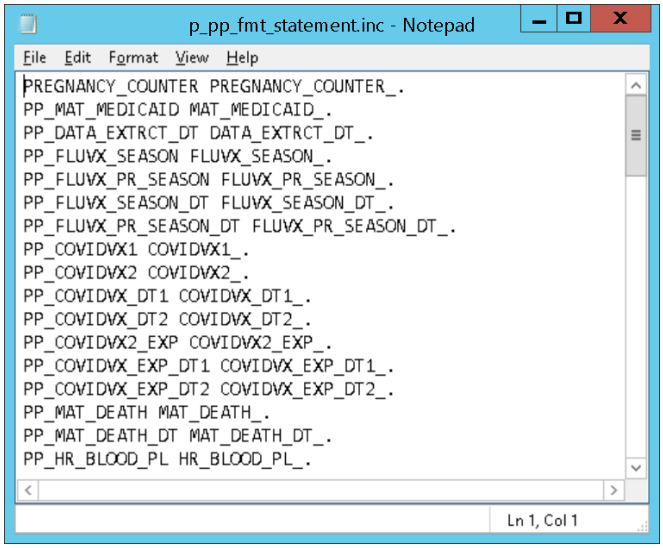
```
01.2_Visits_CNTLIN_7v6.out - Notepad
File  Edit  Format  View  Help
|START           |END             |LABEL  (VER. 9.4     14JUL2021:13:09:06)|
|----------------+----------------+-------------------------------------|
|               0|               0|Adeno negative                       |
|               1|               1|Adeno positive                       |
|               2|               2|Not tested                           |
|             888|             888|Missing                              |
|             999|             999|Unknown                              |
-----------------------------------------------------------------------


-----------------------------------------------------------------------
|                 FORMAT NAME: AMB_VISIT_DT_ LENGTH: 34                |
|   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH:  34  FUZZ: STD   |
|----------------------------------------------------------------------|
|START           |END             |LABEL  (VER. 9.4     14JUL2021:13:09:06)|
|----------------+----------------+-------------------------------------|
|          -21914|          -21914|Unknown or could not be determined   |
Ln 1, Col 1
```

Creating a format assignment statement programmatically

```
data fmtstm;
    length fmtstm $ 80 fmtdot $ 33 variable_name $ 32;
    file ".\&outfi._fmt_statement.inc" lrecl=80;
    set temp (where=(indata ne 0));
    fmtdot=cats(fmtname,'.');
    fmtstm=catx(' ',variable_name,fmtdot);
    put fmtstm;
run;
```

We use the same data set used to create the format catalogs / SAS data sets containing formats to generate format assignment statements. Note the use of SAS functions to add the dot following the format name, as this does not exist in the SAS data set. This format statement can be included in programs to analyze and characterize the data files. Below follows a screenshot of the format assignment file.



```
p_pp_fmt_statement.inc - Notepad
File  Edit  Format  View  Help
PREGNANCY_COUNTER PREGNANCY_COUNTER_.
PP_MAT_MEDICAID MAT_MEDICAID_.
PP_DATA_EXTRCT_DT DATA_EXTRCT_DT_.
PP_FLUVX_SEASON FLUVX_SEASON_.
PP_FLUVX_PR_SEASON FLUVX_PR_SEASON_.
PP_FLUVX_SEASON_DT FLUVX_SEASON_DT_.
PP_FLUVX_PR_SEASON_DT FLUVX_PR_SEASON_DT_.
PP_COVIDVX1 COVIDVX1_.
PP_COVIDVX2 COVIDVX2_.
PP_COVIDVX_DT1 COVIDVX_DT1_.
PP_COVIDVX_DT2 COVIDVX_DT2_.
PP_COVIDVX2_EXP COVIDVX2_EXP_.
PP_COVIDVX_EXP_DT1 COVIDVX_EXP_DT1_.
PP_COVIDVX_EXP_DT2 COVIDVX_EXP_DT2_.
PP_MAT_DEATH MAT_DEATH_.
PP_MAT_DEATH_DT MAT_DEATH_DT_.
PP_HR_BLOOD_PL HR_BLOOD_PL_.
Ln 1, Col 1
```

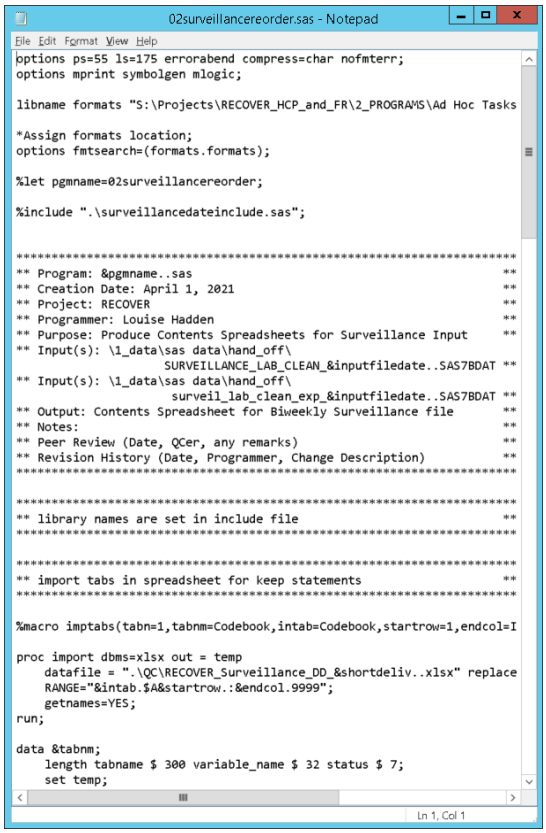
SCENARIO 2

Our second project is also a large scale, multi-site CDC study involving surveillance, COVID and flu testing, and other health metrics including lab data handled separately. Data dictionaries are used extensively, but unlike scenario 1, we do not start with set data dictionaries. The REDCap system is highly flexible and allows for the addition of new forms over time, which leads to different data dictionaries, variable lists, etc. Further, different sites may deploy REDCap forms at a slightly different pace, resulting in input streams with different variables, variable types, etc. REDCaps for each site generate data dictionaries, format programs, and csv files for each collection of forms each week, and these are reviewed for the following weeks' processing. Keeping up with the changes is a full time job for several people managing different processing streams. It is not in the scope of this particular paper to drill down on the exact processes used to manage the task of weekly file generation for the CDC, but we will discuss how we use data dictionaries to design a system to document files sent to the CDC, and compare data and documentation from REDCap (and more) with the final data and documentation sent to CDC. We will focus on a single data stream, surveillance data.

Solutions include using data management tools such as control files, and the use of SAS procedures such as PROC CONTENTS, ODS OUTPUT, and PROC COMPARE. We produce preliminary metadata on incoming weekly surveillance data, which has been pre-processed in a separate set of steps. A control data dictionary is produced from the prior week's delivery, and includes any changes or additions made during the prior week based on review of added forms, etc. The "curated" metadata is used to create a draft deliverable file (corrected position, etc.), and new metadata is produced. Metadata from the incoming file and the curated metadata is compared, and additionally, the format library created by the incoming REDCap data stream is compared to the curated format library. This allows the detection of anomalous data and inconsistent formatting.

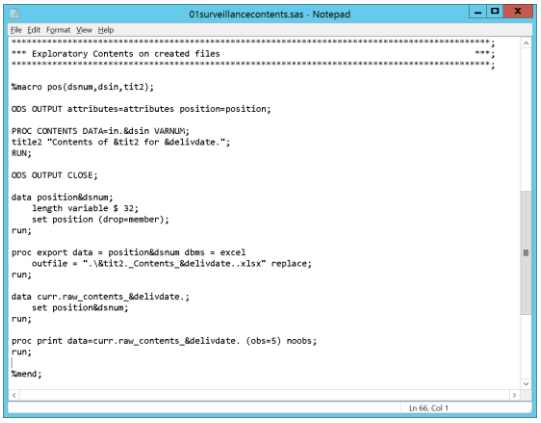
Using a control file

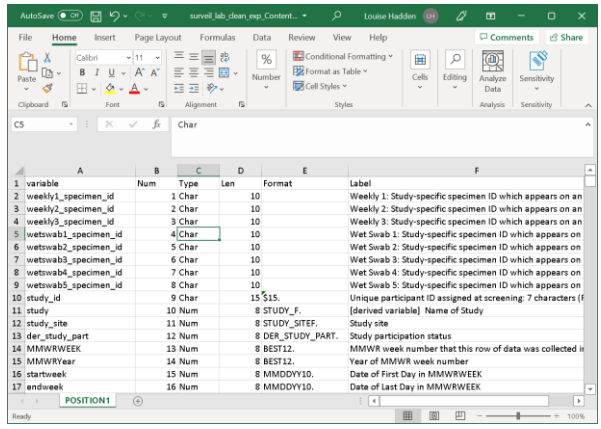
A helpful solution for repetitive processes with multiple programs or scripts is the use of a control file. In this case, a week's processing involves hundreds of processes and all the processes use "dated" data. We use a control file for all surveillance processing to set the dates for incoming files and deliveries, and to set up libraries and filenames for processing. This prevents accidental typos or the use of an incorrect file. The date include file is %included in each program in the surveillance processing stream.



Create metadata from the incoming surveillance file

This is a straightforward process. There are myriad ways to report on SAS metadata – Here is one way to produce the elements desired for our process. Note that the results are saved as a permanent SAS dataset.

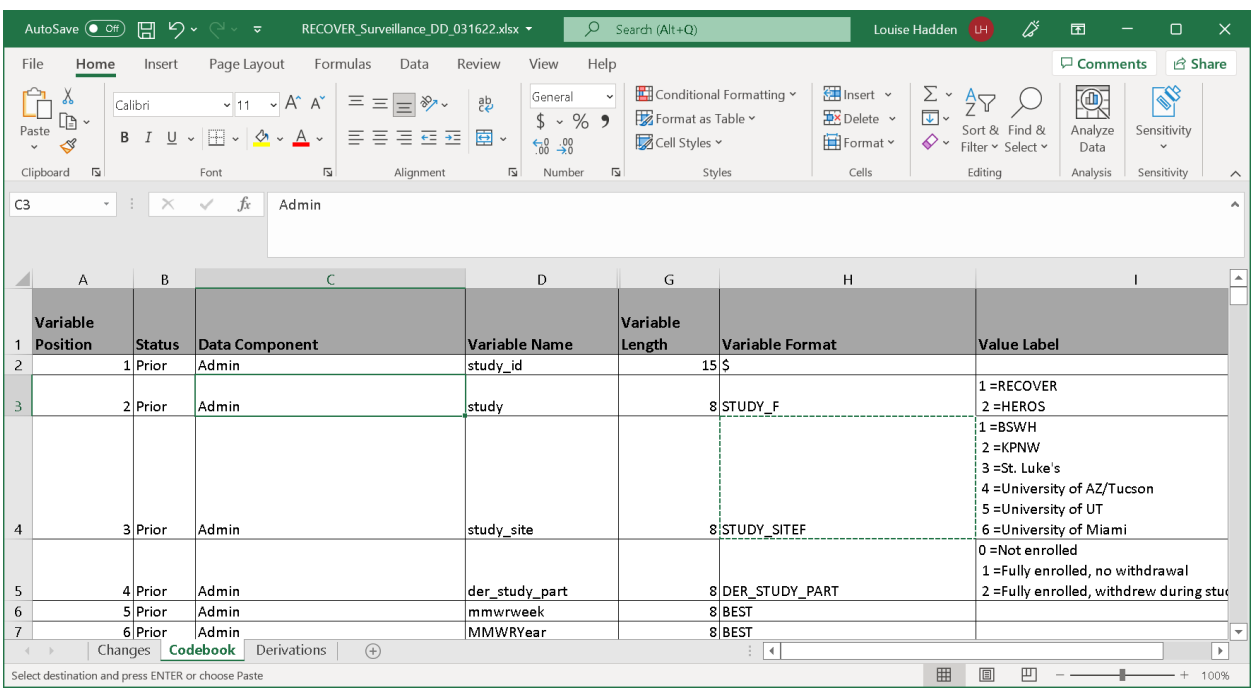




variable	Num	Type	Len	Format	Label
weekly1_specimen_id	1	Char	10		Weekly 1: Study-specific specimen ID which appears on an
weekly2_specimen_id	2	Char	10		Weekly 2: Study-specific specimen ID which appears on an
weekly3_specimen_id	3	Char	10		Weekly 3: Study-specific specimen ID which appears on an
wetswab1_specimen_id	4	Char	10		Wet Swab 1: Study-specific specimen ID which appears on
wetswab2_specimen_id	5	Char	10		Wet Swab 2: Study-specific specimen ID which appears on
wetswab3_specimen_id	6	Char	10		Wet Swab 3: Study-specific specimen ID which appears on
wetswab4_specimen_id	7	Char	10		Wet Swab 4: Study-specific specimen ID which appears on
wetswab5_specimen_id	8	Char	10		Wet Swab 5: Study-specific specimen ID which appears on
study_id	9	Char	15	$15.	Unique participant ID assigned at screening: 7 characters (f
study	10	Num	8	STUDY_F.	[derived variable] Name of Study
study_site	11	Num	8	STUDY_SITEF.	Study site
der_study_part	12	Num	8	DER_STUDY_PART.	Study participation status
MMWRWEEK	13	Num	8	BEST12.	MMWR week number that this row of data was collected i
MMWRYear	14	Num	8	BEST12.	Year of MMWR week number
startweek	15	Num	8	MMDDYY10.	Date of First Day in MMWRWEEK
endweek	16	Num	8	MMDDYY10.	Date of Last Day in MMWRWEEK

Report on metadata from the prior week's surveillance file

The results from the prior week's processing are output in the form of a "codebook" which includes both contents information from the prior week's file and additional items, which include Status, Data Component, and Value Label. Status and Data Component are collected from the prior week's DD, and Value Label is generated programmatically from the format sas data set for the week's processing.



	Variable Position	Status	Data Component	Variable Name	Variable Length	Variable Format	Value Label
2	1	Prior	Admin	study_id	15	$	
3	2	Prior	Admin	study	8	STUDY_F	1 =RECOVER 2 =HEROS
4	3	Prior	Admin	study_site	8	STUDY_SITEF	1 =BSWH 2 =KPNW 3 =St. Luke's 4 =University of AZ/Tucson 5 =University of UT 6 =University of Miami
5	4	Prior	Admin	der_study_part	8	DER_STUDY_PART	0 =Not enrolled 1 =Fully enrolled, no withdrawal 2 =Fully enrolled, withdrew during stu
6	5	Prior	Admin	mmwrweek	8	BEST	
7	6	Prior	Admin	MMWRYear	8	BEST	

Importing the prior data dictionary (DD)

The spreadsheet Codebook tab is read into a SAS data set and saved as a permanent file, via a process which takes metadata from the current data, the prior "curated" DD, and a module which takes the result of looking at the format catalog and linking value labels to variable names. The first code snippet below

imports the codebook, and the second generates include files (text) for use in creating the deliverable sas data set, including the status, data component, and label fields.

02surveillancereorder.sas - Notepad

```
*********************************************************************;
** import tabs in spreadsheet for keep statements                   **;
*********************************************************************;

%macro imptabs(tabn=1,tabnm=Codebook,intab=Codebook,startrow=1,endcol=I);

proc import dbms=xlsx out = temp
    datafile = ".\QC\RECOVER_Surveillance_DD_&shortdeliv..xlsx" replace;
    RANGE="&intab.$A&startrow.:&endcol.9999";
    getnames=YES;
run;

data &tabnm;
    length tabname $ 300 variable_name $ 32 status $ 7;
    set temp;
    tabname = "&intab";
    if status='Updat' then status='Updated';
    if variable_name='' then delete;
run;

proc print data=&tabnm (obs=5) noobs;
format _character_ $40.;
title2 "Test Print of &tabnm";
run;

proc contents data=&tabnm varnum;
title2 "Contents of &tabnm";
run;

proc sort data=&tabnm.;
    by variable_position;
```

Ln 48, Col 14

02surveillancereorder.sas - Notepad

```
*********************************************************************;
** now create include files for status, label and data component **;
** using edited DD from prior delivery                              **;
*********************************************************************;

data _null_;
    length status_statement $ 100;
    file '.\INCLUDE_status.txt' lrecl=100;
    set curr.Codebook_&shortdeliv.;
    word1="IF name='";
    word2="' THEN status='";
    word3="';";
    status_statement=cats(word1,variable_name,word2,status,word3);
    put status_statement;
run;

data _null_;
    length data_component_statement $ 300;
    file '.\INCLUDE_data_component.txt' lrecl=300;
    set curr.Codebook_&shortdeliv.;
    word1="IF name='";
    word2="' THEN data_component='";
    word3="';";
    data_component_statement=cats(word1,variable_name,word2,data_component
    put data_component_statement;
run;

data _null_;
    length label_statement $ 1000;
    file '.\INCLUDE_label.txt' lrecl=1000;
    set curr.Codebook_&shortdeliv.;
```

Ln 1, Col 1

Reordering the incoming data set

You may have noticed that the contents of the incoming file had a different variable order than the DD from the prior delivery. This piece of the program reorders the variables to match what is desired, and exports the contents of the deliverable file.

02surveillancereorder.sas - Notepad

```
*********************************************************************;
** Set up a list to reorder the combined file                       **;
*********************************************************************;

PROC SQL;
SELECT VARIABLE_NAME
INTO :COMBKEEP SEPARATED BY ' '
FROM curr.Codebook_&shortdeliv.;
quit;

%put &combkeep;

*Create dataset with shortened dataset name for codebook macro;
data cb out.surv_formatted_&delivdate.  ;
retain &combkeep ;
set in.surveil_lab_clean_exp_&inputfiledate;
    keep &combkeep.;

    %include '.\INCLUDE_label.txt';
run;

*Read out reordered dataset with formats stripped;
data out.RECOVER_Surveillance_&delivdate.;
retain &combkeep.;
set cb ;
informat _all_ ;
format _all_ ;
run;

*************************************************************************
```

Ln 153, Col 1

02surveillancereorder.sas - Notepad

```
proc export data = position dbms = excel
    outfile = ".\Formatted_Surveillance_Contents_&delivdate..xlsx" replace
run;

proc export data = position (where=(flag_label_length=1)) dbms = excel
    outfile = ".\TooLongLabelLengths&delivdate..xlsx" replace;
run;

ODS OUTPUT attributes=attributes position=position;

PROC CONTENTS DATA=out.RECOVER_Surveillance_&delivdate. VARNUM;
title2 "Contents of Unformatted Surveillance File for &delivdate.";
RUN;

ODS OUTPUT CLOSE;

proc print data=position (obs=5) noobs;
title2 "Test print contents of deliverable Surveillance file for &shortdel
run;

data position;
    length variable $ 32;
    set position (drop=member);
    length_label=length(label);
run;

proc export data = position dbms = excel
    outfile = ".\Unformatted_Surveillance_Contents_&delivdate..xlsx" repla
run;
```

Ln 201, Col 1

Preparing the metadata files for comparison by cleaning and standardizing

We are skipping two programs – one is to create a weekly format library and another is to create a new DD in the identical format as the prior DD. As you have seen, we have created permanent metadata files of steps along the way. First up, we try to give the metadata a fair chance by making it as equivalent as

possible. Anything someone has edited in Excel has a good chance of having garbage in it like tabs and line feeds so we strip them.

05CompareDocumentationWIP.sas - Notepad

```
** massage data sets to trim down variables to compare **;
*******************************************************;

data prior;
    length value_label $ 500;
    set curr.codebook_&shortdeliv. (keep=variable_name status variable_position
                                    data_component variable_description
                                    variable_type variable_length
                                    variable_format value_label);
informat _all_;
format _all_;

    value_label=TRANSLATE(value_label,' ','09'x); /* replace tabs with a single space */
    value_label=translate(value_label,' ','0A'x); /* replace CR with a single space */
    value_label=translate(value_label,' ','0D'x); /* replace LF with a single space */
    value_label=compbl(value_label);
    value_label=left(value_label);

run;

proc sort data=prior;
    by variable_position;
run;

data new;
    length data_component $ 63 value_label $ 500;
    set curr.metadata_full (keep=name status varnum data_component
                           label vtype length format value_label
                       rename=(name=variable_name varnum=variable_position
                           label=variable_description vtype=variable_type
                           length=variable_length format=variable_format));
```

Ln 45, Col 1

05CompareDocumentationWIP.sas - Notepad

```
proc sort data=master_formats (keep=fmtname) out=master_names nodupkey;
    by fmtname;
run;

proc sort data=surv_formats (keep=fmtname) out=surv_names nodupkey;
    by fmtname;
run;

data inboth;
    merge master_names (in=a) surv_names (in=b);
    by fmtname;
    if a and b;
run;

data sub_master (keep=fmtname start end hlo label type);
    length start end $ 32 fmtname $ 33;
    merge inboth (in=a) master_formats (in=b);
    by fmtname;
    if a and b;
    if start ne '';
    start=left(start);
    end=left(end);
run;

data sub_surv (keep=fmtname start end hlo label type);
    length start end $ 32 fmtname $ 33;
    merge inboth (in=a) surv_formats (in=b);
    by fmtname;
    if a and b;
    if start ne '';
    start=left(start);
    end=left(end);
```

Ln 97, Col 1

We move on to a comparison of the two metadata files, and then the two format catalogs in the form of SAS data sets.

Comparison of Metadata files

05CompareDocumentationWIP.sas - Notepad

```
*******************************************************;
** PROC COMPARE of prior metadata with new metadata     **;
*******************************************************;
ods trace on;

ods output comparedatasets=comparedatasets
comparevariables=comparevariables comparesummary=comparesummary;

proc compare base=prior
             compare=new ;
    id variable_position;
title1 'Comparison of major variables';
run;

ods output close;

proc print data=comparedatasets noobs;
title1 'Comparedatasets';
run;

proc print data=comparevariables noobs;
title1 'Comparevariables';
run;

proc print data=comparesummary noobs;
title1 'Comparesummary';
run;

ods trace off;
```

Ln 122, Col 1

05CompareDocumentationWIP.lst - Notepad

```
Variables Summary

Number of Variables in Common: 9.
Number of Variables with Differing Attributes: 3.
Number of ID Variables: 1.


Listing of Common Variables with Differing Attributes

Variable              Dataset     Type  Length  Label

Variable_Position     WORK.PRIOR  Num        8  Variable Position
                      WORK.NEW    Num        8  Variable Number
Variable_Description  WORK.PRIOR  Char     255  Variable Description
                      WORK.NEW    Char     255  Variable Label
Variable_Format       WORK.PRIOR  Char      31  Variable Format
                      WORK.NEW    Char      32  Variable Format


Observation Summary

Observation      Base  Compare  ID

First Obs           1        1  Variable_Position=1
Last  Obs        1042     1042  Variable_Position=1042

Number of Observations in Common: 1042.
Total Number of Observations Read from WORK.PRIOR: 1042.
Total Number of Observations Read from WORK.NEW: 1042.

Number of Observations with Some Compared Variables Unequal: 0.
```

Ln 1, Col 1

Comparison of format catalogs

Below follows a format comparison – as you can see, the master format catalog, which is updated weekly but retains error-laden value labels, has coded missing values, changes in value labels, etc. For those of you who are not familiar with PROC COMPARE output, any mismatches in presence (in this case value labels that are only in one format catalog or the other) are output with an indicator of whether they are in the BASE or COMPARE data set. Any mismatches in data values have three lines – the BASE record, the COMPARE record, and a DIFF record. The DIFF record has a “mask” which indicates where differences are found in the variable value. In our case, we see that some value labels have changed over time, and we need to address to provide the client with consistent data.

PROC COMPARE requires the files to be sorted when using the ID statement. We want to match on the format name and start value. Rather than rely on printed output, we use the OUT= option of PROC COMPARE to output the comparison and then export to a spreadsheet.

Code snippet of PROC COMPARE code

05CompareDocumentation.sas - Notepad

File Edit Format View Help

```
proc sort data=sub_master;
    by fmtname start;
run;

proc sort data=sub_surv;
    by fmtname start;
run;

proc compare base=sub_master
             compare=sub_surv out=result outnoequal outbase outcomp outdif
     noprint;
    id fmtname start;
run;

proc print data=result noobs;
title1 'Comparison of master and surveillance formats';
run;
```

Ln 1, Col 1

Spreadsheet of format comparison output

format_compare_result_03162022.xlsx

C5 DAYS_SYMPTOMS_

	A	B	C	D	E	F	G	H
1	_TYPE_	_OBS_	fmtname	start	end	LABEL	TYPE	HLO
2	BASE	4	DAYS_SYMPTOMS_	**OTHER**	**OTHER**	BEST32.	I	OF
3	BASE	5	DAYS_SYMPTOMS_	.A	.A	Missing	N	
4	BASE	6	DAYS_SYMPTOMS_	0	0	0	N	
5	BASE	14	DAYS_SYMPTOMS_	miss	miss	Missing	C	
6	BASE	15	DAYS_SYMPTOMS_	miss	miss	.A	I	
7	BASE	27	H1_A5_COMPLETE_	**OTHER**	**OTHER**	BEST32.	I	OF
8	BASE	28	H1_A5_COMPLETE_	0	0	No - surveillance contact is incomplete	N	
9	COMPARE	22	H1_A5_COMPLETE_	0	0	No - incomplete	N	
10	DIF	22	H1_A5_COMPLETE_	0	.....	.....XXXXXX.XXXXX.XXXXXXX.XX.XXXXXXXXXX.....	.	.....
11	BASE	29	H1_A5_COMPLETE_	1	1	Yes - surveillance contact is complete	N	
12	COMPARE	23	H1_A5_COMPLETE_	1	1	Yes - complete	N	
13	DIF	23	H1_A5_COMPLETE_	1	.....	.....XXXXXXXXXXXX.XXXXXXX.XX.XXXXXXXX.....	.	.....
14	BASE	30	H1_A5_COMPLETE_	miss	miss	.A	I	
15	BASE	31	H1_B4_COMPLETE_	**OTHER**	**OTHER**	BEST32.	I	OF
16	BASE	32	H1_B4_COMPLETE_	0	0	No - surveillance contact is incomplete	N	
17	COMPARE	24	H1_B4_COMPLETE_	0	0	No - incomplete	N	

RESULT

CONCLUSION

It is clear that using metadata to drive processing and quality assurance can be incredibly helpful and comprises a valuable addition to one's SAS toolbox, along with PROC FORMAT and PROC COMPARE. I hope you will have some fun iterations with functions with your metadata as well.

ACKNOWLEDGEMENTS

This type of complex programming and processing is a team sport. I could not have created and implemented these techniques on my own. A very heartfelt thank you to all my Abt colleagues working on COVID projects.

CONTACT INFORMATION

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

Louise S. Hadden
Abt Associates Inc.
Louise_hadden@abtassoc.com