Paper 240-2007

Using PROC FORMAT for data validation and clean-up

Christianna S. Williams, PhD
The University of North Carolina at Chapel Hill, Chapel Hill, NC

ABSTRACT

Sometimes one is faced with a large and potentially "dirty" data set that needs to be checked for invalid data. The data set may contain character fields that have been entered in varied and non-standard ways. However, the set of valid data values for some of the variables may itself be large and unwieldy, making a long series of statements of the type "IF *x* NOT IN (this, that, other. . .)" a very unattractive programming option. Alternatively, there may be a set of observations, identifiable by being among a large set of values of a key variable, that need to be deleted from your big data set. If the set of valid (or alternatively invalid) values can be enumerated and fed into a SAS® data set, PROC FORMAT with the CNTLIN option can be a real code saver. This paper will present a step-by-step guide to using PROC FORMAT in this way as an aide to data validation and "cleaning", using a real example from health research.

I will give a little background on the specific problem, describe the types of "clean-up" needed, outline the process used to go from the original dirty data set to the final, analyzable file, and show much of the SAS code that was used to accomplish these steps. I will also take a few sidesteps along the way to illustrate some other features of PROC FORMAT that demonstrate its usefulness in data clean-up.

BACKGROUND – The Problem – Messy Character Data

We have conducted a study of roughly 2,800 residents in 225 long-term care facilities (i.e. nursing homes and assisted living facilities). Among a multitude of other data we've collected on these residents, our research assistants abstracted medical records to determine what medications these residents were taking. I want to estimate the prevalence of different types of medications, determine which resident and facility characteristics are associated with the use of certain types of medications, and test whether use of specific categories of medications is associated with quality of life outcomes in the residents. The problem is – the data coming in from the field are very messy. The research assistants simply photocopied or manually copied lists of medications for each resident, which may have been illegible, misspelled, and/or abbreviated in non-standard ways. Sometimes brand names were recorded; other times generic names were used. Moreover, occasionally some items, such as procedures or devices, were recorded that were not actually medications, because they were listed as orders in the residents' medical charts. Ultimately we want to categorize the medications into fairly broad therapeutic classes of interest for our statistical analyses, but we have to get the data cleaned up first. Also, I've been in this line of work long enough to know that we'll be doing something similar again in a few years – if at all possible I'd like to be able to re-use some of this clean-up effort in the next study, and the next one after that…

Figure 1 shows a small subset of the original data to illustrate the starting point. In the original study, up to twenty medications were recorded for each of the 2,800 participating residents. Figure 2 illustrates the destination for the same subset of data – the medication names have been cleaned up in ways that I'll describe and categories have been assigned for each.

This illustrates a principle I always try to follow when I have some data manipulation to accomplish – whether it is a very simple task or a complex one involving combination of data from multiple sources and transpositions. I always try to have a very clear picture – either in my mind or better yet drawn out on paper – of what I want the final data set to look like, including its shape (e.g. normalized or not) and the elements to include. This helps enormously in guiding the process of getting to that destination. I will refer back to these figures throughout the paper.

ID	MED_1	MED_2	MED_3
001	NITROQUIK	PAXIL	ZYPREXA
002	OXYCONTIN	ZANTAC	EYEWASH
003	ZANITIDINE	ATIVAN	
004	ATIVAN	ASPIRIN	ZIPREXA
005	ENSURE	LASIX	ECOTRIN

Figure 1. Subset of the original medication data for five residents. The name for the corresponding SAS data set is ORIGINAL.

ID	DRUG_1	DRUG_2	DRUG_3	CODE_1	CODE_2	CODE_3
001	NITROGLYCERIN	PAROXETINE	OLANZAPINE	240	284	286
002	OXYCODONE	RANITIDINE		280	560	
003	RANITIDINE	LORAZEPAM		560	287	
004	LORAZEPAM	ASPIRIN	OLANZAPINE	287	280	286
005	FUROSEMIDE	ASPIRIN		400	280	

Figure 2. Desired data coding and layout for the subset of residents shown in Figure 1. The name for the corresponding SAS data set is FINAL.

TYPES OF CLEAN-UP NEEDED

So, before detailing the actual process of data clean-up, I'll list the specific types of changes I want to make. There are essentially three types of recoding that need to be done:

1) Replace brand names with generic names. For example, I want to replace "NITROQUIK" (Figure 1: MED_1 for Resident 001) with "NITROGLYCERIN" (see Figure 2). It is critical that this process allow for all spellings of the original drug names. For example, both "ZYPREXA" (Figure 1: MED_3 for Resident 001) and "ZIPREXA" (MED_3 for Resident 004) should be mapped to "OLANZAPINE" when we make the brand name to generic conversion. Further, as multiple brand names (correctly spelled or not) may correspond to the same generic name, we need to allow for this. For example, both "ZANTAC" (Figure 1: MED_2 for Resident 002) and "ZANITIDINE" (Figure 1: MED_1 for Resident 003) are brand names that should be converted to the generic name "RANITIDINE" (Figure 2).
2) Delete non-drugs. As noted above, sometimes items that were not drugs were abstracted because they were listed with the medications as standing orders in the medical chart. "EYEWASH" (Figure 1: MED_3 for Resident 002) and "ENSURE" (Figure 1: MED_1 for Resident 005) fall into this category. When these are deleted, I want to "slide" the other drugs over to fill in the gap (See Figure 2, Resident 005).
3) Code drugs into categories. As noted above, many of our analyses will treat medications in fairly broad categories based on therapeutic purpose (such as anti-depressants or pain medications), so I want to construct variables that map each drug to the correct category. Often, this is a many-to-one mapping. For example, both "OXYCODONE" (Figure 2: DRUG_1 for Resident 002) and "ASPIRIN" (Figure 2: DRUG_2 for both ID 004 and 005) fall into the category of analgesics (coded 280 in this study).

ANOTHER WAY TO VIEW THE PROBLEM…

Each of these types of recoding can be viewed as a table look-up, or mapping or translation – that is, particular values of an input variable (e.g. a particular brand name of a drug) need to be mapped to a new value (e.g. the generic name). Further, each of these types of look-up is a many-to-one situation (from multiple types of misspelling/brand names to a single correct generic name for each unique drug; from multiple non-drugs to a single "DELETE" category; and from multiple generic drugs to a single therapeutic category for analyses). As I'll describe, PROC FORMAT is very well-suited to this type of translation problem.

STEPS TOWARDS A SOLUTION

So, now that we've described the types of recoding and transformation that are needed, I'm going to outline the steps that I went through to get there. Then I'll go back over each of the steps with the SAS code that was used to accomplish each. Of course, it can't all be accomplished just with SAS – at least the first time. Someone who knows something about drugs – a content expert, if you will – has to help. However, my goal was that this person would have to correct each of the myriad types of misspelling only once, would have to tell me the generic name corresponding to a given brand name only once, and would have to tell me how each generic name should be categorized only once. Further, the next time around I won't need quite so much help – we'll have to deal only with new drugs and new misspellings! I broke the task down into the following steps:

1) Transpose the original data set (subset shown in Figure 1) to make an unduplicated list of the drugs (old names – prior to recoding).
2) Obtain corrected generic names for all the drugs. Items that are not drugs are given the name "DELETE".
3) Use the result of Step 2 to produce a SAS FORMAT that maps the brand names to the correct generic names. Non-drugs get mapped to "DELETE".
4) Produce an unduplicated list of the generic names of the actual drugs. Add the drug category codes.
5) Produce a SAS FORMAT that maps the corrected generic names to drug category codes.
6) Transpose the original data (duplicates and non-drugs included) to one observation per ID-drug combination.

7) Use FORMAT built in Step 3 to convert original names to generic names and the format from Step 5 to categorize generic names into drug categories. Delete non-drugs.
8) Transpose result so that it is again one observation per ID.

STEP 1 – Produce an Unduplicated List of the Old Drug Names

For this step the input data is the data shown in Figure 1. The data set is called ORIGINAL. Here is the SAS code* for the desired transposition:

```
DATA vertical (KEEP = oldname);
  SET original ;

ARRAY meds{3} med_1 - med_3 ;
DO i = 1 TO 3;
   oldname = meds{i} ;
   IF oldname NE ' ' THEN OUTPUT;
END;
RUN;

PROC SORT DATA=vertical NODUP;
BY oldname;
RUN;
```

(*NOTE: Here and in code examples throughout the paper, SAS keywords are in all caps to readily distinguish them from user-defined names, which are in lower case.)

The DATA step outputs an observation for each non-missing medication name in the data set ORIGINAL, calling the original drug names OLDNAME. In the actual study the array had 20 elements though not all residents had 20 non-missing values: this would work for an array of arbitrary size. Before the SORT, the VERTICAL data set has 14 observations (as ID 003 had only two non-missing values). As the only variable in the VERTICAL data set is OLDNAME, using PROC SORT with the NODUP option eliminates exact duplicates, such as we have for “ATIVAN”. Thus, before sorting VERTICAL has 14 observations; afterwards it has 13. See Figure 3.

OLDNAME
ASPIRIN
ATIVAN
ECOTRIN
ENSURE
EYEWASH
LASIX
NITROQUICK
OXYCONTIN
PAXIL
ZANITIDINE
ZANTAC
ZIPREXA
ZYPREXA

Figure 3. Data set VERTICAL, which contains an unduplicated list of 'old' names of drugs

STEP 2 – Assign Correct Generic Names to all Items

This step requires content knowledge, that is, someone who is familiar with the drugs and knows what generic name corresponds to each of these “old” names or brand names. For our research team, this expert was a pharmacy student. I provided her a list similar to – but much larger than – the one above (Figure 3) in a spreadsheet file, and she entered the correct generic names for each of these items. If something was not a drug, she coded it as “DELETE”. This is, of course, a labor-intensive process, but she has to code each brand name only once – or more

precisely each spelling variation of each brand name once. Figure 4 shows the result, which was read back into a SAS data set called OLD_NEW.

OLDNAME	GENERIC
ASPIRIN	ASPIRIN
ATIVAN	LORAZEPAM
ECOTRIN	ASPIRIN
ENSURE	DELETE
EYEWASH	DELETE
LASIX	FUROSEMIDE
NITROQUICK	NITROGLYCERIN
OXYCONTIN	OXYCODONE
PAXIL	PAROXETINE
ZANITIDINE	RANITIDINE
ZANTAC	RANITIDINE
ZIPREXA	OLANZAPINE
ZYPREXA	OLANZAPINE

Figure 4. Data set OLD_NEW, which contains an unduplicated list of old (brand) names along with corresponding generic names. Note many-to-one coding and that some items (non-drugs) are marked “DELETE”.

STEP 3 – Generate FORMAT to Map Old Names to Generic Names

Leaving aside for a moment the use of the CNTLIN= option in PROC FORMAT, which turns a specially constructed data set into a user-defined format, let's be clear on what we want this first format to do…map the "old names" to the "generic names". For this small subset, the PROC FORMAT code to create a character format (indicated by the $ sign before the format name) could look like this:

```
* Map OLDNAME to GENERIC (the hard way) ;
PROC FORMAT;
VALUE $old_new
      'ASPIRIN'     = 'ASPIRIN'
      'ATIVAN'      = 'LORAZEPAM'
      'ECOTRIN'     = 'ASPIRIN'
      'ENSURE'      = 'DELETE'
      'EYEWASH'     = 'DELETE'
      'LASIX'       = 'FUROSEMIDE'
      'NITROQUICK'  = 'NITROGLYCERINE'
      'OXYCONTIN'   = 'OXYCODONE'
      'PAXIL'       = 'PAROXETINE'
      'ZANITIDINE'  = 'RANITIDINE'
      'ZANTAC'      = 'RANITIDINE'
      'ZIPREXA'     = 'OLANZAPINE'
      'ZYPREXA'     = 'OLANZAPINE'
          ;
RUN;
```

This would allow us to convert the "old" names – including misspellings into the consistent '"generic" names. This was tedious typing for a handful of drugs – it would be downright mind-numbing (not to mention extremely error-prone) for several hundred or several thousand! We can use the data set we already have and the CNTLIN= option of PROC FORMAT to streamline the process considerably.

The data set OLD_NEW in Figure 4 is almost exactly what we need to use PROC FORMAT with the CNTLIN= option, which allows us to read in a SAS data set and convert it to a format, with one variable corresponding to the value ranges and the other to the labels. The code below provides specifics.

```
DATA cntl_oldnew ;
  SET old_new  (RENAME=(oldname=START generic=LABEL)) ;
RETAIN FMTNAME 'old_new'   TYPE 'C';
KEEP FMTNAME TYPE START LABEL;
RUN;
```

This DATA step puts the data set (CNTL_OLDNEW) in the exact format required by PROC FORMAT with the CNTLIN= option. When using PROC FORMAT in this way, the variable corresponding to the values must be called START for the beginning of the range and END for the end of the range. Here, the variable OLDNAME corresponds to the beginning of the value range (and hence is RENAMEd START), and as no END variable is specified, the range includes only the value given START. We want these values to map to the variable called GENERIC in Figure 4, but PROC FORMAT expects the label variable to be called LABEL, so GENERIC is RENAMEd to LABEL. We also need to provide a name for the FORMAT we'll be constructing, and this name (here OLD_NEW) must be contained in a variable called FMTNAME. Because the START variable is a character variable, this will be a character FORMAT and so the variable TYPE (again the name of the variable is the requirement of the FORMAT procedure) is given the value 'C'. (By default TYPE would be 'N' for numeric). So, the data set produced by this code would also work:

```
DATA cntl_oldnew ;
  SET old_new  (RENAME=(oldname=START generic=LABEL)) ;
RETAIN FMTNAME '$old_new' ;
KEEP FMTNAME TYPE START LABEL;
RUN;
```

Producing the data set CNTL_OLDNEW by either method, we then use PROC FORMAT with the CNTLIN= option to actually construct the $OLD_NEW format:

```
LIBNAME LIBRARY 'C:\NHstudy\resdata\drugfmts\';
PROC FORMAT CNTLIN=cntl_oldnew FMTLIB  LIBRARY=LIBRARY ;
RUN;
```

The PROC FORMAT code itself is very simple. I want this to be a permanent format so I have a LIBNAME LIBRARY statement to specify the desired location for the FORMAT library. The CNTLIN= option specifies the data set that will provide the format information (here CNTL_OLDNEW) and the LIBRARY=LIBRARY indicates where to write the format. Because no second level name is specified, a format catalog called FORMATS will be created (or added to if it already exists) in the specified directory. The FMTLIB option provides a listing of all the formats in the library. The listing produced is shown below. In the actual study, this format had several thousand values.

```
-----------------------------------------------------------------------
|   FORMAT NAME: $OLD_NEW LENGTH:   13   NUMBER OF VALUES:   13       |
|  MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH  13  FUZZ:    0   |
|---------------------------------------------------------------------|
|START         |END          |LABEL  (VER. V7|V8   27JAN2007:07:30:07)|
|--------------+-------------+-------------------------------------|
|ASPIRIN       |ASPIRIN      |ASPIRIN                              |
|ATIVAN        |ATIVAN       |LORAZEPAM                            |
|ECOTRIN       |ECOTRIN      |ASPIRIN                              |
|ENSURE        |ENSURE       |DELETE                               |
|EYEWASH       |EYEWASH      |DELETE                               |
|LASIX         |LASIX        |FUROSEMIDE                           |
|NITROQUICK    |NITROQUICK   |NITROGLYCERIN                        |
|OXYCONTIN     |OXYCONTIN    |OXYCODONE                            |
|PAXIL         |PAXIL        |PAROXETINE                           |
|ZANITIDINE    |ZANITIDINE   |RANITIDINE                           |
|ZANTAC        |ZANTAC       |RANITIDINE                           |
|ZIPREXA       |ZIPREXA      |OLANZAPINE                           |
|ZYPREXA       |ZYPREXA      |OLANZAPINE                           |
-----------------------------------------------------------------------
```

Note that if the WORK (i.e. temporary) library and a permanent library contain formats with the same name, the version in the WORK space will generally take precedence (although this can be changed using the global OPTION

FMTSEARCH, which specifies the order in which SAS will "search" through directories to find a specified FORMAT when the FORMAT is applied).

Note that two items, which we will not want to include in the final data set are formatted to “DELETE” and there are multiple names in the START column which will be formatted to the same name. We will come back to the use of this FORMAT in STEP 7, but now we need to get the generic drug information in shape to be assigned numeric codes corresponding to the categories.

A quick review of the key features here of using the CNTLIN= option in PROC FORMAT:

- The CNTLIN data set must have a variable called FMTNAME; the *value* assigned to the FMTNAME variable specifies the name of the format that will be created. It will have the same value for all observations on the CNTLIN data set.
- The CNTLIN data set must have a variable called START, which specifies the starting value for each range; these must be unique – that is, no two observations in the CNTLIN data set may have the same value for the START variable. Put another way, the ranges must be non-overlapping.
- If the value ranges are indeed ranges, then the CNTLIN data set must include a variable END, which specifies the ending value for each range. If there is no END variable, then each range includes the single value specified by the START variable (as in this example).
- If the format to be created is a character format, then the CNTLIN data set must include a variable TYPE with value 'C' (as shown here). Alternatively, the value of the FMTNAME variable can begin with a '$' (i.e. I could have specified `RETAIN FMTNAME '$OLD_NEW';` and not mentioned TYPE). If neither of these is done, FORMAT will assume you are creating a numeric format – which would clearly have caused an error here.
- This option can also be used to create an INFORMAT (which will be demonstrated later in the paper). The TYPE variable must have a value of 'I' for a numeric INFORMAT and 'J' for a character INFORMAT. PICTURE formats can also be created (TYPE = 'P').
- The CNTLIN data set must contain a variable LABEL. The values of LABEL specify what value each START-END range will be mapped to. It is perfectly ok (as in this example) for many START values (or START-END ranges to map to the same value of the LABEL variable. Here, multiple "old names" map to a single "generic" LABEL, and many map to "DELETE".
- There are many other FORMAT options that can be specified with the CNTLIN data set, and PROC FORMAT can also be used to create SAS data sets using the CNTLOUT= option. These are beyond the scope of this paper, but see References and Recommended Reading at the end of the paper.

ASIDE: USING THE 'HLO' VARIABLE IN A CNTLIN DATA SET

Also, before moving on to the next step in the process, I want to point out a possible alternative to my explicit mapping of non-drugs to the value 'DELETE'. Suppose my OLD_NEW data set didn't include the observations where OLDNAME is "EYEWASH" or "ENSURE", but I was including what I knew to be an exhaustive list of all the possible "old" drug names. Just as in the use of PROC FORMAT to create a user-defined format in which one uses the OTHER=keyword to map all values not explicitly defined in specified ranges to some desired value, one can use a special variable named HLO on the CNTLIN data set to specify what to do with any values that are not in any of the specified START-END ranges. Here is a modification of the code to create the CNTL_OLDNEW data set to illustrate. I'm calling this slightly modified version of the OLD_NEW data set (without the EYEWASH and ENSURE observations) OLD_NEW2. The format created is also slightly different, so I'm giving it a new name.

```
DATA cntl_oldnew2;
  SET old_new2  (RENAME=(oldname=START generic=LABEL)) END=last;
RETAIN FMTNAME 'old2new'   TYPE 'C';
KEEP FMTNAME TYPE START LABEL HLO;

OUTPUT ;

IF last THEN DO;
   HLO = 'O';
   LABEL = 'DELETE';
   OUTPUT ;
END;
RUN;
```

```
PROC FORMAT CNTLIN=cntl_oldnew2 FMTLIB  LIBRARY=LIBRARY ;
RUN;
```

What this additional code will do is map any values other than those explicitly mapped to a generic name (LABEL) to 'DELETE'. This would include missing values. This is accomplished by adding one additional record to the CNTLIN dataset. On this observation, the variable HLO gets the value of 'O' to indicate that the range is OTHER and label is assigned the value 'DELETE'. This is essentially equivalent to a line of code in a typical VALUE statement in PROC FORMAT that states "OTHER='DELETE'; " The variable HLO will be missing for all other observations in the CNTLIN data set. If you knew that you could define the entire set of valid values for the variable to be mapped, and that anything else could be assigned to 'DELETE' (or some other desired value indicating invalid data), then this could be the way to go. You might NOT want to do this if you wanted to be sure to capture any unexpected values (i.e. didn't automatically set them to DELETE)…we'll see how this works when we actually apply this format in Step 7 below.

This would not have been my preferred method in the application I am illustrating because the earlier FORMAT ($OLD_NEW) documents all of the items that are mapped to delete, which is useful protocol documentation to have. Nonetheless, there are many cases when the 'OTHER'= range is very useful. Here is the FMTLIB listing of this new format $OLD2NEW:

```
-----------------------------------------------------------------------
|    FORMAT NAME: $OLD2NEW LENGTH:   13   NUMBER OF VALUES:   12       |
|  MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH  13  FUZZ:      0   |
|---------------------------------------------------------------------|
|START          |END            |LABEL  (VER. V7|V8   23FEB2007:18:08:07)|
|---------------+---------------+-------------------------------------|
|ASPIRIN        |ASPIRIN        |ASPIRIN                              |
|ATIVAN         |ATIVAN         |LORAZEPAM                            |
|ECOTRIN        |ECOTRIN        |ASPIRIN                              |
|LASIX          |LASIX          |FUROSEMIDE                           |
|NITROQUICK     |NITROQUICK     |NITROGLYCERIN                        |
|OXYCONTIN      |OXYCONTIN      |OXYCODONE                            |
|PAXIL          |PAXIL          |PAROXETINE                           |
|ZANITIDINE     |ZANITIDINE     |RANITIDINE                           |
|ZANTAC         |ZANTAC         |RANITIDINE                           |
|ZIPREXA        |ZIPREXA        |OLANZAPINE                           |
|ZYPREXA        |ZYPREXA        |OLANZAPINE                           |
|**OTHER**      |**OTHER**      |DELETE                               |
-----------------------------------------------------------------------
```

STEP 4 – Eliminate duplicates, delete non-drugs, and specify category codes.

The elimination of the duplicate generic names and deletion of the non-drugs is a simple matter, which can be handled by a single PROC SORT shown below, in which only the observations where GENERIC is not coded "DELETE" are read; duplicates (on the key GENERIC) are eliminated by the NODUPKEY option. Even with a large data set the sort should not be too much of a memory hog since we are just sorting a single variable.

```
PROC SORT DATA=old_new (KEEP=generic
  WHERE=(generic NE 'DELETE'))  OUT=generic NODUPKEY;
BY generic;
RUN;
```

The input for the SORT is the data set shown in Figure 4. The output will be a list of unique generic names; the DATA set GENERIC in this case will have eight observations (there are eight unique generic names) and a single variable. The next part of this step again requires the content expertise of the pharmacy student who will assign each generic drug name to the appropriate therapeutic category. For example, the code '240' indicates cardiovascular medications, '280' are analgesics, and '400' are diuretics. This coding was again done in an external spreadsheet program and then read back into a SAS data set, containing just the unique generic names and the corresponding category CODEs. See Figure 5.

GENERIC	CODE
ASPIRIN	280
FUROSEMIDE	400
LORAZEPAM	287
NITROGLYCERIN	240
OLANZAPINE	286
OXYCODONE	280
PAROXETINE	284
RANITIDINE	560

Figure 5. Data set NEW_CODE, which contains an unduplicated list of generic names (produced by PROC SORT above) and the corresponding category codes (added by content expert)

STEP 5 – Generate FORMAT to Map Generic Names to Categories.

This step is very similar to Step 3 above. Here we will use the data set illustrated in Figure 5, configure it so that the variables are named appropriately for PROC FORMAT's CNTLIN= option, and run this procedure to generate a format "$NEW_CAT," which will be added to our format library. The purpose of this format is to group the generic drug names into the correct therapeutic code categories. See code below.

```
DATA cntl_newcode ;
SET new_code    (RENAME=(generic=START code=LABEL)) END=last;

RETAIN FMTNAME 'new_cat'  TYPE 'C';
KEEP FMTNAME TYPE START LABEL HLO;

OUTPUT ;

IF last THEN DO;
   HLO = 'O';
   LABEL = '999';
   OUTPUT ;
END;
RUN;

PROC FORMAT  CNTLIN=cntl_newcode FMTLIB    LIBRARY=LIBRARY ;
RUN;

** Add descriptions to formats ;
PROC CATALOG CATALOG=LIBRARY.FORMATS ;
MODIFY old_new.FORMATC  (DESCRIPTION="Brand names to generics");
MODIFY old2new.FORMATC  (DESCRIPTION="Brand names to generics w/OTHER");
MODIFY new_cat.FORMATC  (DESCRIPTION="Generics to drug classes");
CONTENTS;
QUIT;
```

I again use the HLO variable to specify that a category "OTHER = '999' is created, knowing that 999 is not a valid drug category code and it will be assigned if I later try to apply the FORMAT $NEW_CAT to any drug names that have not been assigned to a category. A portion of the listing produced by including the option FMTLIB is shown below. The previously written formats "$OLD_NEW" and "$OLD2NEW" would be printed as well.

I use the MODIFY statement in PROC CATALOG to add descriptions to each of the formats I've created. The CONTENTS statement provides a listing of all the entries in the format library (here just three) with their descriptions. This is also shown below.

```
 ----------------------------------------------------------
|FORMAT NAME: $NEW_CAT LENGTH: 3  NUMBER OF VALUES: 8      |
|MIN LENGTH:  1  MAX LENGTH: 40 DEFAULT LENGTH 3  FUZZ: 0  |
|----------------------------------------------------------|
|START          |END            |LABEL                     |
|---------------+---------------+--------------------------|
|ASPIRIN        |ASPIRIN        |280                       |
|FUROSEMIDE     |FUROSEMIDE     |400                       |
|LORAZEPAM      |LORAZEPAM      |287                       |
|NITROGLYCERIN  |NITROGLYCERIN  |240                       |
|OLANZAPINE     |OLANZAPINE     |286                       |
|OXYCODONE      |OXYCODONE      |280                       |
|PAROXETINE     |PAROXETINE     |284                       |
|RANITIDINE     |RANITIDINE     |560                       |
|**OTHER**      |**OTHER**      |999                       |
 ----------------------------------------------------------


Contents of Catalog LIBRARY.FORMATS
# Name    Type            Create Date       Modified Date Description
-------------------------------------------------------------------------------
1 NEW_CAT FORMATC  27JAN2007:07:30:07  27JAN2007:07:30:07 Map generics to
                                                          drug classes
2 OLD_NEW FORMATC  27JAN2007:07:30:07  27JAN2007:07:30:07 Map brand names
                                                          to generics
3 OLD2NEW FORMATC  27JAN2007:07:30:07  27JAN2007:07:30:07 Map brand names
                                                          to generics w/
                                                          OTHER
```

STEP 6 – Transpose the original data set

The purpose of this step is to transpose the original data set (Figure 1) to one observation per drug, maintaining the duplicates. This will make it simpler to remove the non-drugs and apply the formats we have constructed. PROC TRANSPOSE with a BY statement will do the trick. By doing the transposition by ID, the ID variable itself is not transposed, and the drug names (renamed OLDNAME from the default COL1) remain associated with the correct ID, which identifies the individual in the study (and can, of course, be used as a link to other data collected on each person in the study).

The TRANSPOSE code is shown below, and the resulting data set (named VERTICAL_ID) is shown in Figure 6. Note that this method of transposition (as opposed to the DATA STEP method shown in Step 1) above in which we only output an observation for non-missing values of OLDNAME) will also output a third observation for ID 003 with a missing value for OLDNAME). Either method will work – one just needs to know what the results will look like so that other code can be adapted as needed.

```
PROC TRANSPOSE DATA=original
       OUT=vertical_id (DROP=_NAME_  RENAME=(COL1=oldname)) ;
BY id;
VAR med_1 - med_3 ;
RUN;
```

ID	OLDNAME
001	NITROQUICK
001	PAXIL
001	ZYPREXA
002	OXYCONTIN
002	ZANTAC
002	EYEWASH
003	ZANITIDINE
003	ATIVAN
003	
004	ATIVAN
004	ASPIRIN
004	ZIPREXA
005	ENSURE
005	LASIX
005	ECOTRIN

Figure 6. Data set VERTICAL_ID, which corresponds to the original data set (Figure 1) after transposition BY ID. Duplicates, non-drugs and missing values are included.

STEP 7 – Apply FORMATS

Our data set is now in perfect shape to apply the two FORMATS we have constructed. Via the PUT function, the FORMAT $OLDNEW. will map the OLDNAMEs shown in Figure 6 to the generic names, and then after deleting those that map to the label "DELETE", the FORMAT $NEW_CAT. will map each generic name to the correct drug category code. Here is the simple DATA step that will do this.

```
DATA almost notdrugs;
  SET vertical_id ;

LENGTH generic $13 ;
IF oldname NE ' ' then generic = PUT(oldname,$old_new.);
ELSE DELETE ;
IF generic = 'DELETE' THEN DO;
   OUTPUT notdrugs;
   DELETE;
END;

code=INPUT(PUT(generic,$new_cat.),3.);
OUTPUT almost ;
RUN;
```

The data set specified in the SET statement is the one shown in Figure 6. The variable GENERIC will contain the result of translating the OLDNAMEs into the "new" or generic names. I do not want to include observations with missing values for OLDNAME, which will exist for all IDs that have fewer then the maximum number of drugs (in this case just one observation for ID 003 who had only two old drug names listed) so those observations are deleted with the "ELSE DELETE;" statement. The LENGTH statement assures that if there are longer generic names later in the data set that they will not be truncated (the requisite length can be obtained from the FMTLIB output – this could be automated.) By nesting the PUT function inside the INPUT function in the assignment statement, the variable CODE will be a numeric variable.

The resulting DATA set (ALMOST) is shown in Figure 7. Additionally, creating an output dataset (DELETED, not shown) which contains the deleted items (non-drugs in this case), allows a check that everything is working as expected. In this case this data set includes 2 observations, one for ID 002 (OLDNAME=EYEWASH) and one for ID 003 (OLDNAME=ENSURE). In both cases our $OLD_NEW format maps these to DELETE.

ID	OLDNAME	GENERIC	CODE
001	NITROQUICK	NITROGLYCERIN	240
001	PAXIL	PAROXETINE	284
001	ZYPREXA	OLANZAPINE	286
002	OXYCONTIN	OXYCODONE	280
002	ZANTAC	RANITIDINE	560
003	ZANITIDINE	RANITIDINE	560
003	ATIVAN	LORAZEPAM	287
003	ECOTRIN	ASPIRIN	280
004	ATIVAN	LORAZEPAM	287
004	ASPIRIN	ASPIRIN	280
004	ZIPREXA	OLANZAPINE	286
005	LASIX	FUROSEMIDE	400
005	PAXIL	PAROXETINE	284

Figure 7. DATA set ALMOST, produced by applying the FORMAT $OLD_NEW and $NEW_CAT. If a normalized file is desired, this would be the final product.

We would have gotten the exact same results if we had used the $OLD2NEW format instead. As the DATA statement suggests, we are almost to our desired final data set product. In fact, there might be reason to leave the data set in this normalized shape (one observation per drug as opposed to one observation per ID).

ASIDE: USING A NUMERIC FORMAT TO LABEL DRUG CODES

At this point, I might very well wish to produce some type of report on the drug categories…if nothing else a simple frequency count. For this purpose, the normalized data set is probably preferable. Of course, to most people the numeric categories have no meaning; so we'd like to label them, and this is clearly a task for PROC FORMAT, used in the way it is probably most commonly used – to assign value labels to numeric data to make reports more comprehensible. As the CODE variable is a numeric variable, this is a numeric format. I have a data set CODELABL that maps the codes to their labels. A portion of it (including only categories represented in this mini-data set used here) is shown in Figure 8.

code	catlabel
240	Cardiovascular drugs
280	Analgesics, antipyretics, NSAIDs, and narcotics
284	Antidepressants
286	Antipsychotic agents
287	Anxiolytics, sedatives and hypnotics
400	Diuretics and other agents affecting fluid balance
560	Gastrointestinal drugs

Figure 8. Data set CODELABL, which defines the drug category codes used in the examples.

I can again use PROC FORMAT with the CNTLIN= option, after slight tweaks to the CODELABL data set. A simple PROC FREQ step will provide a frequency listing for the drug categories in this little data set. See the code below:

```
DATA cntl_lablcat ;
SET codelabl (RENAME=(code=START catlabel=LABEL)) ;
RETAIN FMTNAME 'catlabl'  TYPE 'N';
KEEP FMTNAME TYPE START LABEL;
RUN;
```

```
PROC FORMAT CNTLIN=cntl_lablcat FMTLIB  LIBRARY=LIBRARY ;
RUN;

PROC FREQ DATA=almost ;
TABLES code /NOCUM ;
FORMAT code catlabl. ;
RUN;
```

Of course, one could permanently assign the CATLABL format to the CODE variable in a DATA step if desired rather than just applying it "on the fly" in the FREQ procedure. The FREQ output is shown below.

Drug Category

code	Frequency	Percent
Cardiovascular drugs (inc. antihypertensives/antilipemics)	1	8.33
Analgesics, antipyretics, NSAIDs, and narcotics	3	25.00
Antidepressants	1	8.33
Antipsychotic agents	2	16.67
Anxiolytics, sedatives and hypnotics	2	16.67
Diuretics & agents affecting electrolyte/caloric/water balance	1	8.33
Gastrointestinal drugs	2	16.67

STEP 8 – Transpose to return data set to one observation per ID format

While the normalized form might be useful for some types of analysis, because most of the other data in this study was at the person (ID) level, I wanted to transpose the data one more time to get it so there is one record per ID.

I used a DATA step to convert the data to the desired final format. Here is the code:

```
DATA final ;
 SET almost (DROP = oldname);
 BY id;

ARRAY drug{3}$20 drug_1-drug_3;
ARRAY cat{3} code_1-code_3 ;

IF FIRST.id THEN DO;
    index = 0;
    DO i = 1 TO 3;
          drug{i} = ' ';
          cat{i} = . ;
    END;
END;
RETAIN drug_1-drug_3 code_1-code_3;

index + 1;
drug{index} = generic;
cat{index} = code ;

IF LAST.id THEN OUTPUT;
DROP index generic code i;
RUN;
```

The DATA step processes the input (SET) data set ALMOST by ID. For the first input observation for each ID, there is an initialization step, which consists of setting a counter to 0 (variable INDEX), and setting each drug name (DRUG_1 – DRUG_3) and the corresponding category codes (CODE_1 – CODE_3) to missing values (so that values RETAINed during processing of previous ID will not be left behind. Then, for each observation within each BY group (i.e. for all observations for a given ID), the index is incremented by one and the generic names and codes are placed into the elements of the array. In a real situation, these arrays could, of course, have many more elements. Finally, once this process is complete for the last observation for an ID, a record is output, and the excess variables are dropped. The result is the desired final product, shown in Figure 2 above. Finally!

ASIDE: USING CNTLIN= TO CREATE AN INFORMAT

As I noted earlier the CNTLIN= option in PROC FORMAT can be used to create INFORMATs as well as FORMATs. This would be useful if the data to be cleaned up was in an external text file that could be read in using the INFILE statement. Assuming we already had the data set OLD_NEW, the code to construct an INFORMAT that would map the old names to the generic name is very similar to that used to create the FORMAT in Step 3. In fact, the only difference is that we specify TYPE as 'J' rather than 'C';

```
DATA cntl_oldnew_I ;
  SET old_new  (RENAME=(oldname=START generic=LABEL)) ;
RETAIN FMTNAME 'oldnew'   TYPE 'J';
KEEP FMTNAME TYPE START LABEL;
RUN;

PROC FORMAT CNTLIN=cntl_oldnew_I FMTLIB  LIBRARY=LIBRARY ;
RUN;
```

Only the header information for this INFORMAT distinguishes it from the FORMAT $OLD_NEW. Part of the FMTLIB output is shown below. Note the @ sign before the INFORMAT name, which SAS uses to distinguish INFORMATs from FORMATs in the format catalog, and the column header, which says INVALUE instead of LABEL:

```
-------------------------------------------------------------------------
 |   FORMAT NAME: @$OLDNEW LENGTH:   13   NUMBER OF VALUES:   13       |
 |  MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH  13  FUZZ:    0   |
 |-----------------------------------------------------------------------|
 |START         |END          |INVALUE (VER. V7|V8  27JAN2007:07:30:07)|
 |--------------+-------------+---------------------------------------|
 |ASPIRIN       |ASPIRIN      |ASPIRIN                                |
 |ATIVAN        |ATIVAN       |LORAZEPAM                              |
 |ECOTRIN       |ECOTRIN      |ASPIRIN                                |
 |ENSURE        |ENSURE       |DELETE                                 |
 |... continues as for the format $OLD_NEW ...                          |
 -------------------------------------------------------------------------
```

This INFORMAT could be used in different ways, but I'll just illustrate one here. Let's say that the data that was our ORIGINAL data set was in an external file. We can use the $OLDNEW INFORMAT to transform the drug names into the generic names when they are read in. This is a little artificial since that same data was used to create the format in this case, which required it to be in a SAS data set, but it's useful for illustration.

```
DATA original2 ;
FORMAT id Z3. ;
INPUT
  @1 id 3.  @5 med_1 $oldnew.  @20 med_2 $oldnew.  @35 med_3 $oldnew.;
DATALINES;
001 NITROQUICK     PAXIL          ZYPREXA
002 OXYCONTIN      ZANTAC         EYEWASH
003 ZANITIDINE     ATIVAN
004 ATIVAN         ASPIRIN        ZIPREXA
005 ENSURE         LASIX          ECOTRIN
;
RUN;
```

The resulting data set is shown in Figure 9. Note that this data set would still need some manipulation to eliminate the "DELETE" values – particularly if we didn't want missing values in positions to the "left" of valid drugs, but this would be readily accomplished with a little transposition of the data. Of course the $NEW_CAT format could be used to map these drug names to their respective categories.

ID	MED_1	MED_2	MED_3
001	NITROGLYCERIN	PAROXETINE	OLANZAPINE
002	OXYCODONE	RANITIDINE	DELETE
003	RANITIDINE	LORAZEPAM	
004	LORAZEPAM	ASPIRIN	OLANZAPINE
005	DELETE	FUROSEMIDE	ASPIRIN

Figure 9. Data set ORIGINAL2, based on raw data read using the $OLDNEW informat.

CONCLUSIONS

The purpose of this paper has been to illustrate the use of PROC FORMAT with the CNTLIN= option to help in the process of data validation and "cleaning". Along the way, various methods of transposing data are also demonstrated as well as some other ways of using PROC FORMAT. The example used here were based on a real research study in which medication data had been collected on a large number of nursing home residents, but because of many misspellings, abstraction of inappropriate items and the need to combine many drugs into broad categories for analysis, substantial cleaning was needed before the data could be reported or used in statistical analyses. The labor-intensive (i.e. expensive) part of this process is having a content expert specify the correct values for the drug names and the categories. However, by setting up the process outlined in this paper, I reduced the effort needed in the current study. Further, by storing the corrections and codes in a FORMAT library, the effort will be reduced in future studies as well. As applications go, this is a very simple one, and it has been applied to a very specific situation, but it is my hope that the process and many of the techniques used could generalize to other content areas and perhaps be incorporated into more elaborate data cleansing and coding applications.

REFERENCES

SAS OnlineDoc 9.1, Base SAS: Procedures Guide, The FORMAT Procedure

ACKNOWLEDGMENTS

Thanks to my terrific NESUG colleague Rob Russell for a careful reading of the paper and for the excellent suggestion to add descriptions to the formats using PROC CATALOG.

RECOMMENDED READING

For many great data-cleaning tips, including some using FORMATs and INFORMATs:

Cody, Ron 1999. *Cody's Data Cleaning Techniques Using SAS® Software*, Cary, NC: SAS Institute Inc.

For lots of great examples of using PROC FORMAT for many different purposes:

Bilenas, Jonas V. 2005. *The Power of PROC FORMAT*. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

I welcome comments, suggestions and questions at:

Christianna S. Williams, PhD
Cecil G. Sheps Center for Health Services Research
725 Martin Luther King, Jr. Boulevard
Campus Box # 7590
University of North Carolina at Chapel Hill
Chapel Hill, North Carolina 27599
Email: Christianna_Williams@unc.edu