Paper CC104

Let SAS® Do the Coding for You!

Robert Williams
WellPoint, Inc.

ABSTRACT

Many times, we need to create the same reports going to different groups based on the group's subset of queried data or we have to develop many repetitive SAS codes such as a series of IF THEN ELSE statements or a long list of different conditions in a WHERE statement. It is cumbersome and a chore to manually write and change these statements especially if the reporting requirements change frequently. This paper will suggest methods to streamline and eliminate the process of writing and copying/pasting your SAS code to be modified for each requirement change. Two techniques will be reviewed along with a listing of key words in a SAS dataset or an Excel® file: 1) Create code using the DATA _NULL_ and PUT statements to an external SAS code file to be executed with %INCLUDE statement. 2) Create code using the DATA _NULL_ and CALL SYMPUT to write SAS codes to a macro variable. You will be amazed how useful this process is for hundreds of routine reports especially on a weekly or monthly basis. *RoboCoding* is not just limited to reports; this technique can be expanded to include other procedures and data steps. Let the *RoboCoder* do the repetitive SAS coding work for you!

INTRODUCTION

Many times, we have to write a series of SAS code that repeats itself with minor adjustments such as changing the WHERE statements in a data step, a PROC REPORT procedure or even a block of statements within the DATA step. Typically, it can be handled by copying and pasting and then making the necessary adjustments to the code. If the copy/paste is not feasible, then it can be labor intensive to manually write the code especially when we need to create a series of IF/THEN/ELSE statements or a long list of different conditions in the WHERE statement. In this paper, I will show two techniques of auto-generating SAS codes using a listing of key words in a SAS dataset or an Excel file.

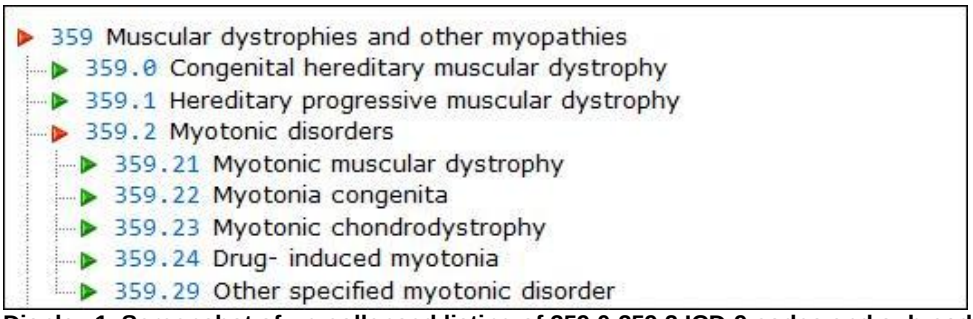
1) Technique #1: Generate SAS code using the DATA _NULL_ and CALL SYMPUT to write SAS codes to a macro variable to be invoked.
2) Technique #2: Generate SAS code using the DATA _NULL_ and PUT statements to write to an external SAS program file to be executed with %INCLUDE statement.

I call this SAS *Robo-coding*. The best way to show these two SAS *Robo-coding* techniques is by example in a hypothetical situation. In our typical world of the medical health field, suppose a project request was received asking for a subset of the claims based on a list of conditions. We are provided with this list of diagnoses and diagnosis categories corresponding to a mixed range of diagnosis codes like this below.

- Muscle problems that are of a disabling nature, limited to: 356.1, 359.0-359.2, 045.9, 728.11
- Hemophilia: 286.0-286.4
- Acquired or congenital heart disease: 745.0-747.9; 424.0-429.9

CODING DILEMMA: DIRECT CODING OR USE ROBO-CODING

We are faced with a dilemma on how to best write the SAS code to incorporate a list of mixed diagnoses codes and sub-codes. For those of you who are non-health care professionals, the medical diagnosis codes have many sub-codes and we need to account for each 1-digit or 2 digit sub-code. For example, to account for all medical diagnosis codes from 359.0 to 359.2, according to the website, www.icd9data.com (Display 1), the dictionary of medical coding shows some of 359 codes have just 1-digit sub-codes while others has 2-digit sub-codes. The sub-codes are the digits to the right of the period.



Display 1. Screenshot of un-collapsed listing of 359.0-359.2 ICD-9 codes and sub-codes

If we code directly into the WHERE statement to subset the sample claims data and then categorize the data into disease categories via a series of IF/THEN/ELSE or SELECT/WHEN statements, we would have coded a DATA step similar to this code below. As you may have noticed, the DATA step have lines of code repeating itself with some adjustment for each condition in the WHERE statement as well as the categories in the WHEN statements.

```
data WORK.CLAIMS_SUBSET_DX_cat;
   set WORK.SESUG_samp_claims;
   where (IDCD_ID like "0459%" or IDCD_ID like "3561%" or IDCD_ID like "3590%"
          or IDCD_ID like "3591%" or IDCD_ID like "3592%" or IDCD_ID like "72811%"
          or IDCD_ID like "2860%" or IDCD_ID like "2861%" or IDCD_ID like "2862%"
          or IDCD_ID like "2863%" or IDCD_ID like "2864%" or IDCD_ID like "424%"
          or IDCD_ID like "425%" or IDCD_ID like "426%" or IDCD_ID like "427%"
          or IDCD_ID like "428%" or IDCD_ID like "429%" or IDCD_ID like
          "745%" or IDCD_ID like "746%" or IDCD_ID like "747%");
   format DX_category
   $150.; select;
       when (substr(IDCD_ID,1,4) = '0459') DX_category = 'Muscle problems';
       when (substr(IDCD_ID,1,4) = '3561') DX_category = 'Muscle problems';
       when (substr(IDCD_ID,1,4) = '3590') DX_category = 'Muscle problems';
       when (substr(IDCD_ID,1,4) = '3591') DX_category = 'Muscle problems';
       when (substr(IDCD_ID,1,4) = '3592') DX_category = 'Muscle problems';
       when (substr(IDCD_ID,1,5) = '72811') DX_category = 'Muscle problems';
       when (substr(IDCD_ID,1,4) = '2860') DX_category = 'Hemophilia';
       when (substr(IDCD_ID,1,4) = '2861') DX_category = 'Hemophilia'; when
       (substr(IDCD_ID,1,4) = '2862') DX_category = 'Hemophilia'; when
       (substr(IDCD_ID,1,4) = '2863') DX_category = 'Hemophilia'; when
       (substr(IDCD_ID,1,4) = '2864') DX_category = 'Hemophilia'; when
       (substr(IDCD_ID,1,3) = '424') DX_category = 'Heart disease'; when
       (substr(IDCD_ID,1,3) = '425') DX_category = 'Heart disease'; when
       (substr(IDCD_ID,1,3) = '426') DX_category = 'Heart disease'; when
       (substr(IDCD_ID,1,3) = '427') DX_category = 'Heart disease'; when
       (substr(IDCD_ID,1,3) = '428') DX_category = 'Heart disease'; when
       (substr(IDCD_ID,1,3) = '429') DX_category = 'Heart disease'; when
       (substr(IDCD_ID,1,3) = '745') DX_category = 'Heart disease'; when
       (substr(IDCD_ID,1,3) = '746') DX_category = 'Heart disease'; when
       (substr(IDCD_ID,1,3) = '747') DX_category = 'Heart disease';
       OTHERWISE DX_category = "WHAT CATEGORY?!";
   END;
run;
```

This direct coding requires a lot of typing. If you were like me, I copied and pasted each line and modify each line of code. However, there is another way to do this by dynamically create the repetitive lines of SAS codes. Using this fictional project, here is an outline that demonstrates the two techniques on how to dynamically create SAS codes.

- Technique #1: *Macro variable*
  - Set up a SAS data set with the list of key words i.e. list of diagnosis codes in 1st column of Table 1.
  - Create a macro variable to incorporate the list of diagnoses for WHERE statement. This process uses the DATA _NULL_ and CALL SYMPUT to create the macro variable from the SAS data set.

- Technique #2: *External SAS program file*
    - Set up a SAS data set with the same list of key words in technique #1 above but add another column that includes the corresponding diagnosis categories.
    - Create an external SAS program file to incorporate the list of diagnoses codes and the associated categories for the SELECT/WHEN statement block. This process uses the DATA _NULL_ and PUT statements to write to an external SAS program file from the SAS data set
- Run the DATA step
    - The DATA step will subset the sample claims data and categorize the diagnoses by incorporating the generated SAS codes from the two techniques above. This is how we incorporate the generated codes.
        - Technique #1: Invoke the macro variable inside the WHERE statement with the ampersand "&" symbol.
        - Technique #2: Insert the external SAS code within the DATA step with %INCLUDE statement.

SETTING UP SAS DATA SET

The first step is to create a SAS data set with key data that you will need for the WHERE statement as well as assigning the disease categories. The key data can be created in Excel and then be imported into a SAS data set or it can be created by a separate DATA step (sample SAS code including the sample data is at the bottom of this paper). In this example, I created a column below with the key parts of the diagnosis codes and their corresponding disease category. The first column have the key words that we want to use in the WHERE statement such as "`WHERE IDCD_ID like '747%' or IDCD_ID like '0459%' or …`". For the 2nd technique, we add the second column what we will used together with the first column for the SELECT/WHEN statement block such as "`when (substr(IDCD_ID,1,3)='747') DX_category = 'Heart disease';`" and so on. Using Table 1, we will create SAS codes with a list of LIKE conditions for the WHERE statement (using technique #1) and a series of WHEN statements for the disease categories (using technique #2). For the remainder of this paper, the name of this SAS data set will be SESUG_DX_LISTING.

DX_list	DX_CATEGORY
0459	Muscle problems
3561	Muscle problems
3590	Muscle problems
3591	Muscle problems
3592	Muscle problems
72811	Muscle problems
2860	Hemophilia
2861	Hemophilia
2862	Hemophilia
2863	Hemophilia
2864	Hemophilia
424	Heart disease
425	Heart disease
426	Heart disease
427	Heart disease
428	Heart disease
429	Heart disease
745	Heart disease
746	Heart disease
747	Heart disease

Table 1. SESUG_DX_LISTING SAS data set

TECHNIQUE #1: WRITING SAS CODES TO MACRO VARIABLE

In this technique, we generate the SAS code by creating a SAS macro variable from the SESUG_DX_LISTING table (Table 1). This is a DATA _NULL_ process that does not actually write a new SAS data set hence the "_NULL_" phrase. The purpose of this DATA step is to build a string of characters that will be used in the WHERE statement. First, we created a variable called "`where_string`" which I make it a length of 5,000 characters. That way, there will be enough space to create the whole string. Below is the SAS code that will build the string of characters from the SESUG_DX_LISTING table (Table 1).

```
data _null_;
   set WORK.SESUG_DX_LISTING END=last;
   format where_string $5000.;
   retain where_string;
   /* This is the first line for the where statement */
   if _n_ = 1 then where_string = 'IDCD_ID like "'||trim(DX_LIST)||'%"';
   /* subsequent lines that adds the "OR" line for the where statement
   */ else where_string = trim(where_string)||' '||'OR IDCD_ID like
   "'||trim(DX_LIST)||'%"';
   /* At the end, this outputs entire where statement to the macro variable
   */ if last then call symput("IDCD_ID_where_statement",where_string);
run;
```

There are several key points that goes into building the WHERE statement string.

- Uses the RETAIN statement for the "`where_string`" variable. It helps retain the data string as the variable goes through each DATA step iteration. That way, the data string is retained as it goes to the next row of data to be modified. Otherwise, the data string is cleared out at the start of the next DATA step iteration.
- Uses the DATA step internal variable "_n_" especially for "`_n_ = 1`". This will allow us to start building the string with the first data row. As you can see in the IF statement, it will write "`IDCD_ID like '0459%'`" to the "`where_string`" variable. That takes care of the beginning of the string.
- After the first data step iteration, the internal variable "_n_" will be greater than one. Therefore, executing the ELSE statement portion of the code. It will take the data string from the previous DATA step iteration and adds "`OR IDCD_ID like '3561%'`" to the data string. This is where you see how important the RETAIN statement is because it "carries" the data string from the previous iteration to be concatenated together with the current iteration that includes the OR statement.
- The TRIM statement is used to trim off the trailing blank spaces before the data string is concatenated.
- The END statement used on the SET line which I called it LAST. When the last row is processed, the LAST in the IF statement will output the series of concatenated statements into the macro variable called "`IDCD_ID_where_statement`". The CALL SYMPUT statement is used to output the "`where_string`" into the macro variable called "`IDCD_ID_where_statement`".

Now, you can use the %PUT statement to see what the macro variable contains as shown in Output 1. From the SESUG_DX_LISTING (Table 1), the %PUT statement will show this in the log. This will save us so much work having to code all these LIKE conditions.

```
%put &IDCD_ID_where_statement;

IDCD_ID like "0459%" or IDCD_ID like "3561%" or IDCD_ID like "3590%" or IDCD_ID like
"3591%" or IDCD_ID like "3592%" or IDCD_ID
like "72811%" or IDCD_ID like "2860%" or IDCD_ID like "2861%" or IDCD_ID like
"2862%" or IDCD_ID like "2863%" or IDCD_ID like
"2864%" or IDCD_ID like "424%" or IDCD_ID like "425%" or IDCD_ID like "426%" or
IDCD_ID like "427%" or IDCD_ID like "428%" or
IDCD_ID like "429%" or IDCD_ID like "745%" or IDCD_ID like "746%" or IDCD_ID
like "747%"
```

Output 1. Log showing %PUT statement of the macro variable "IDCD_ID_where_statement"

TECHNIQUE #2: WRITING SAS CODES TO AN EXTERNAL TEXT FILE

In this technique, we create SAS code by writing to an external text file using the SAS data set SESUG_DX_LISTING table (Table 1). Again, this is a DATA _NULL_ data step that will not generate a new SAS data set. The purpose of this data step is to write a series of SELECT/WHEN statements to be used for categorizing the DX categories as organized in the SESUG_DX_LISTING table (Table 1). First, we create a variable called "`when_string`" which has a length of 250 characters. That way, there will be enough space to write a line of SAS code. Below is the SAS code that will write each line of SAS code to the external SAS program file from the SESUG_DX_LISTING table (Table 1).

```
filename tmpwhen "C:\SESUG\tempwhenstatements.sas"; /* Makes a SAS program file */
data _null_;
```

```
    set WORK.SESUG_DX_LISTING END=last;
    format when_string $250.;
    file tmpwhen; /* This is the file the PUT statements writes to */
    /* This sets the length for the SUBSTR function in the WHEN statement */
    str_length = put(length(DX_LIST),1.0);
    /* Write first line which is just the SELECT statement */
    if _n_ = 1 then put @1 'select;';
    /* Write each WHEN statement for each DX with the corresponding category
    */ when_string = "when (substr(IDCD_ID,1,"||trim(str_length)||") =
    '"||trim(DX_LIST)||"') DX_category = '"||trim(DX_CATEGORY)||"';";
    put @5 when_string;
    /* Finish writing the last two statements in the SELECT/WHEN statement block */
    if last then do;
           put @5 'OTHERWISE DX_category = "WHAT CATEGORY?!";';
           put @1 'END;';
    end;
run;
```

There are several key points that go into the process of writing SAS codes into the external text file.

- Prior to the DATA _NULL_ data step, we use the FILENAME statement. This statement will establish a reference to a text file. The example uses a reference name "`tmpwhen`" which actually references to a text file location such as: "`C:\SESUG\tempwhenstatements.sas`". This is the SAS program file that we will write the series of SELECT/WHEN statements to.
- In the DATA _NULL_ statement, use the FILE statement along with the file reference to "`tmpwhen`" that was established with the FILENAME statement. The FILE statement is where the DATA step will write to.
- Uses the DATA step's internal variable "_n_" especially for "`_n_ = 1`". This will allow us to write the first line to the external SAS program file. The IF statement, it will write a simple statement, "`select;`" as the first line of the external file. This is the beginning of dynamic coding the SELECT/WHEN statement block. After the first DATA step iteration, the internal variable "_n_" will be greater than one. Therefore, it will not write the SELECT statement again.
- Before we start writing the WHEN statement lines after the SELECT statement, we need to define a parameter size for the SUBSTR function. Needing a value for the parameter size, the "`str_length`" variable is created to determine the length of the DX code from the first column of the SESUG_DX_LISTING (Table 1). This number will be embedded the SUBSTR function for each WHEN statement. This will allow us to dynamically change the parameters for the SUBSTR function based on the size of the DX codes.
- The "`when_string`" variable is used to put together the WHEN statement from the SESUG_DX_LISTING table (Table 1) along with the "`str_length`" variable. The "`when_string`" will have something like this "`when (substr(IDCD_ID,1,3)='747') DX_category = 'Heart disease';`."
- After the WHEN statement string is created for the "`when_string`" variable, we write this statement to the external SAS program file using this PUT statement: "`put @5 when_string;`". This will write a line to the external file. I use "`@5`" in the PUT statement to make an indent at 5 spaces from the left. This makes the SAS code easier to read.
- The TRIM statement is used to trim off the trailing blank spaces before the data string is concatenated together.
- The END statement used on the SET line which I called it LAST. When the last row is processed, the LAST flag in the IF statement will write the last two statements for the SELECT/WHEN statement block like this

```
OTHERWISE DX_category = "WHAT CATEGORY?!";
END;
```

- After the DATA _NULL_ process finishes, the external SAS program file "`tempwhenstatements.sas`" is created with SELECT/WHEN statement block generated by the PUT statements.

Now, you can open up the external file, "`tempwhenstatements.sas`", to see SELECT/WHEN statement block we just created from the SESUG_DX_LISTING table (Table 1). The external file will show what was written below. You will see parameter values of 3, 4 and even 5 in the SUBSTR function portion of the WHEN statements. The parameter values are dynamically changed with the value of "`when_string`" variable depending on the size of the first column of the Table 1. This alone shows the power of dynamically creating the repetitive lines of code. This

technique will save us so much work having to code all these WHEN statements with different size parameters for the SUBSTR function.

The code inside the external file: "`tempwhenstatements.sas`"

```
select;
   when (substr(IDCD_ID,1,4) = '0459') DX_category = 'Muscle problems';
   when (substr(IDCD_ID,1,4) = '3561') DX_category = 'Muscle problems';
   when (substr(IDCD_ID,1,4) = '3590') DX_category = 'Muscle problems';
   when (substr(IDCD_ID,1,4) = '3591') DX_category = 'Muscle problems';
   when (substr(IDCD_ID,1,4) = '3592') DX_category = 'Muscle problems';
   when (substr(IDCD_ID,1,5) = '72811') DX_category = 'Muscle problems';
   when (substr(IDCD_ID,1,4) = '2860') DX_category = 'Hemophilia';
   when (substr(IDCD_ID,1,4) = '2861') DX_category = 'Hemophilia'; when
   (substr(IDCD_ID,1,4) = '2862') DX_category = 'Hemophilia'; when
   (substr(IDCD_ID,1,4) = '2863') DX_category = 'Hemophilia'; when
   (substr(IDCD_ID,1,4) = '2864') DX_category = 'Hemophilia'; when
   (substr(IDCD_ID,1,3) = '424') DX_category = 'Heart disease'; when
   (substr(IDCD_ID,1,3) = '425') DX_category = 'Heart disease'; when
   (substr(IDCD_ID,1,3) = '426') DX_category = 'Heart disease'; when
   (substr(IDCD_ID,1,3) = '427') DX_category = 'Heart disease'; when
   (substr(IDCD_ID,1,3) = '428') DX_category = 'Heart disease'; when
   (substr(IDCD_ID,1,3) = '429') DX_category = 'Heart disease'; when
   (substr(IDCD_ID,1,3) = '745') DX_category = 'Heart disease'; when
   (substr(IDCD_ID,1,3) = '746') DX_category = 'Heart disease'; when
   (substr(IDCD_ID,1,3) = '747') DX_category = 'Heart disease';
   OTHERWISE DX_category = "WHAT CATEGORY?!";
END;
```

PUT SAS CODES TOGETHER: RUNNING THE AUTO-GENERATED SAS CODE

Now, we are ready to incorporate the auto-generated SAS code from the two techniques in the DATA step to extract a subset of claims and categorize the disease. The SAS code from the first technique is a macro variable to be invoked inside the WHERE statement using "`&`" sign or ampersand symbol i.e. "`where (&IDCD_ID_where_statement);`". The SAS code from the second technique is an external SAS program file that will be inserted using the %INCLUDE statement i.e. "`%include tmpwhen;`". The "`tmpwhen`" statement is the same reference file name as the external SAS program file that was created. It will insert the external SAS code in its place.

```
/* Source2 option will allow the external SAS code to display in the log
*/ options source2;
data WORK.CLAIMS_SUBSET_DX_cat;
   set WORK.SESUG_samp_claims;
   /* The macro variable will put the WHERE conditions we created
   */ where (&IDCD_ID_where_statement);
   format DX_category $150.;
   /* Inserts the external SAS code that we created the WHEN statements */
   %include tmpwhen;
run;
```

After invoking the macro variable and inserting the external SAS program code, this DATA step will look exactly the same as if we were coding this directly in the "Coding Dilemma" section of this paper. The results of the DATA step will produce Table 2 which has the subsets of the sample claims data with the proper diagnosis selections in the WHERE statement and categorization of the diagnoses in the WHEN statements.

Claim_ID	DOS	IDCD_ID	DX_category
521959269	01MAR2013	2863	Hemophilia
620558593	01MAR2013	3591	Muscle problems
834446494	01MAR2013	42511	Heart disease
350650440	01MAR2013	2860	Hemophilia
999400510	04MAR2013	7467	Heart disease
295583464	04MAR2013	74510	Heart disease

Claim_ID	DOS	IDCD_ID	DX_category
294778600	04MAR2013	7452	Heart disease
725175228	06MAR2013	2860	Hemophilia
576261199	06MAR2013	4280	Heart disease
325277861	06MAR2013	35921	Muscle problems
702352846	07MAR2013	4254	Heart disease
575425830	07MAR2013	4281	Heart disease
740288358	07MAR2013	7464	Heart disease
245189238	08MAR2013	2860	Hemophilia
930080969	11MAR2013	42821	Heart disease
411880687	11MAR2013	3591	Muscle problems
931851740	14MAR2013	3591	Muscle problems
305813261	16MAR2013	3590	Muscle problems

Table 2. Final data set of the sample claims

CONCLUSION

By looking at the DATA step we just *Robo-coded*, the two techniques saved us a significant amount of manual labor of coding all those LIKE statements as well as the WHEN statements. This example may not seem like a lot of coding to do for such a short list in the SESUG_DX_LISTING table (Table 1). But these two techniques will be a big help if we have a large number of repetitive SAS statements to code. This is not limited to just making the WHERE statements or the SELECT/WHEN statements. You will be amazed how useful these techniques can be applied to dozens or hundreds of routine reports especially on a weekly or monthly basis. *RoboCoding* techniques can be expanded to include other SAS procedures and DATA steps. Let the *RoboCoder* do the repetitive SAS coding work for you! I am confident that you will add these two valuable techniques to your expanding SAS bag of coding tips.

ACKNOWLEDGEMENTS:

I would like to thank the Southeast SAS Users Group for accepting my abstract and paper as well as for conducting a successful conference. I would also like to thank Joe Urbi at WellPoint, Inc. for providing feedback.

REFERENCES

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

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

The Web's Free 2013 Medical Coding Reference. (http://www.icd9data.com/)

CONTACT INFORMATION

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

Robert Williams
WellPoint, Inc.
1300 Amerigroup Way
Virginia Beach, VA 23464
Robert.Williams@wellpoint.com

SAMPLE SAS CODE

This is the full sample SAS code for this paper including SESUG_DX_LISTING table (Table 1) and the sample claims data. Run the first two DATA steps to import the diagnosis listing and the sample claims data. The rest of the code is for the two *Robo-Coding* techniques.

```
/* this is the data set of key DX and categories for the SESUG_DX_LISTING table */
data WORK.SESUG_DX_LISTING;
    format DX_list $5.;
    format DX_CATEGORY $50.; infile
    datalines delimiter='|';
       input DX_list $ DX_CATEGORY $
       ; datalines;
0459|Muscle problems
3561|Muscle problems
3590|Muscle problems
3591|Muscle problems
3592|Muscle problems
72811|Muscle problems
2860|Hemophilia
2861|Hemophilia
2862|Hemophilia
2863|Hemophilia
2864|Hemophilia
424|Heart disease
425|Heart disease
426|Heart disease
427|Heart disease
428|Heart disease
429|Heart disease
745|Heart disease
746|Heart disease
747|Heart disease
run;
/* This is a sample claims data for this paper */
/* It has 40 rows of claims data. */
/* Note: Claims ID has been scrambled and dates of service has been randomized */
data WORK.SESUG_samp_claims;
    format Claim_ID 9.0 DOS DATE9. IDCD_ID $5.;
       informat DOS DATE9.;
       infile datalines
       delimiter='|'; input Claim_ID
       DOS IDCD_ID $; datalines;
299162360|01MAR2013|29622
991925024|01MAR2013|29680
577369858|01MAR2013|38200
430302367|01MAR2013|5589
876331892|01MAR2013|65583
521959269|01MAR2013|2863
620558593|01MAR2013|3591
834446494|01MAR2013|42511
350650440|01MAR2013|2860
953814363|01MAR2013|29570
783984003|04MAR2013|65703
999400510|04MAR2013|7467
295583464|04MAR2013|74510
294778600|04MAR2013|7452
571969019|05MAR2013|78099
866712984|05MAR2013|79501
725175228|06MAR2013|2860
576261199|06MAR2013|4280
325277861|06MAR2013|35921
702352846|07MAR2013|4254
575425830|07MAR2013|4281
740288358|07MAR2013|7464
245189238|08MAR2013|2860
930080969|11MAR2013|42821
411880687|11MAR2013|3591
```

```
666030620|11MAR2013|29530
607775910|11MAR2013|3094
357294485|12MAR2013|V053
239756817|12MAR2013|30981
698580465|12MAR2013|2967
869433614|13MAR2013|25000
397244252|13MAR2013|3029
811434889|13MAR2013|4019
864832781|13MAR2013|4660
931851740|14MAR2013|3591
529180268|15MAR2013|78903
403535027|15MAR2013|1105
305813261|16MAR2013|3590
874030959|18MAR2013|4644
928100359|18MAR2013|4779
run;
/* Technique #1:  Macro Variable method  */
/* Generates the list of conditions for the WHERE statement */
/* the list is outputted as a SAS macro variable            */
data _null_;
      set WORK.SESUG_DX_LISTING END=last;
      format where_string $5000.;
      retain where_string;
      /* This is the first line for the where statement */
      if _n_ = 1 then where_string = 'IDCD_ID like "'||trim(DX_LIST)||'%"';
      /* subsequent lines that adds the "OR" line for the where statement
      */ else where_string = trim(where_string)||' '||'OR IDCD_ID like
      "'||trim(DX_LIST)||'%"';
      /* At the end, this outputs entire where statement to the macro variable
      */ if last then call symput("IDCD_ID_where_statement",where_string);
run;
/* Taking a look how the where statement that was outputted into the macro variable
*/ %put &IDCD_ID_where_statement;
/* Technique #2:  External SAS Program file method  */
/* create a category SELECT/WHEN statement block using the same where criteria
*/ /* It is to create a separate SAS file from a data _null_ step */
filename tmpwhen "C:\SESUG\tempwhenstatements.sas"; /* Makes a SAS program file */
data _null_;
      set WORK.SESUG_DX_LISTING END=last;
      format when_string $250.;
      file tmpwhen; /* This is the file the PUT statements writes to */
      /* This sets the length for the SUBSTR function in the WHEN statement */
      str_length = put(length(DX_LIST),1.0);
      /* Write first line which is just the SELECT statement */
      if _n_ = 1 then put @1 'select;';
      /* Write each WHEN statement for each DX with the corresponding category
      */ when_string = "when (substr(IDCD_ID,1,"||trim(str_length)||") =
      '"||trim(DX_LIST)||"') DX_category = '"||trim(DX_CATEGORY)||"';";
      put @5 when_string;
      /* Finish writing the last two statements in the SELECT/WHEN statement block */
      if last then do;
            put @5 'OTHERWISE DX_category = "WHAT CATEGORY?!";';
            put @1 'END;';
      end;
run;
/* Actual DATA step incorporating technique #1 and #2 */
/* Source2 option will allow the external SAS code to display in the log
*/ options source2;
data WORK.CLAIMS_SUBSET_DX_cat;
      set WORK.SESUG_samp_claims;
      /* The macro variable will put the where conditions we created
      */ where (&IDCD_ID_where_statement);
      format DX_category $150.;
      /* Inserts the external SAS code that we created the WHEN statements */
      %include tmpwhen;
run;
```