Paper 249-2011

ODS Output Datasets that Work for You

Stuart Long, Westat, Durham, NC
Ed Heaton, Data and Analytic Solutions, Inc., Fairfax, VA
Dan Scharf, Westat, Durham, NC

ABSTRACT

Although ODS Tables, combined with the ODS TRACE and ODS Output statements, provide a consistent method for retrieving statistics generated by SAS® Procedures, the design of these tables is inconsistent.

Many of the ODS Output data sets created with ODS Tables revolve around statistics describing one specific variable or groups of variables. We commonly use only a subset of the data found in a SAS Listing when reporting or continuing an analysis. Each of these data sets could be streamlined to contain these values plus additional descriptive information concerning these variables of interest -- such as the variable name, value, label, and formatted value.

This paper proposes a consistent data structure for these data sets that better facilitates reporting and analysis. It demonstrates a method to translate the data in the ODS Tables into this format.

After showing how to design data sets with this common convention, this paper demonstrates how to (1) automate their creation and (2) combine the summarized data from multiple ODS Tables within a procedure or between different procedures.

INTRODUCTION

SAS introduced the Output Delivery System (ODS) in SAS Version 7.0 as the new standard for output of data from procedures. ODS provides a method to consistently store statistics and descriptive information in objects referred to as ODS Tables. All methods of reporting data from procedures use these tables to retrieve data generated by a given procedure. Thus all data reported in the SAS Listing is first stored in and then retrieved from ODS Tables. So if a data value appears in the SAS Procedure Listing, it must also exist in one of the ODS Tables generated by the same SAS Procedure.

The information in the ODS Tables can be routed to various destinations which can be selected by the user. The SAS Listing is the default destination for many of these ODS Tables. However, the destination that this paper will concentrate on is the SAS dataset. The ODS Output statement is the method used to assign the data in the ODS Tables to SAS datasets.

The majority of SAS procedures generate ODS Tables containing statistics pertaining to a variable or group of variables across all observations, or on subsets of observations with a CLASS or BY statement. Procedures may also compute association statistics between this first set of variables with a second variable, such as modeling a dependent response variable with independent exposure variables.

What Does SAS Give Us?

Due to the large number and variety of procedures in the various SAS software modules, it is not surprising that the structure of the ODS Tables (and hence the output datasets that they produce) is not always consistent. This is especially evident in the identification of the variables, variable values and statistics that are involved in ODS output. Figures 1, 2, and 3 below illustrate selected ODS output from the UNIVARIATE, FREQ and LOGISTIC procedures, respectively.

These examples use several different techniques for identifying variable names. The Moments table shown from PROC UNIVARIATE stores the name of the analytic variable in a column called VarName, while the ParameterEstimates table from PROC LOGISTIC uses Variable for the name of this column. The CrossTabFreqs table in PROC FREQ uses a separate column for each analytic variable, with the names of the analytic variables as the names of the columns while also embedding the variable names in a separate column named Table. Similarly, the OddsRatios table from PROC LOGISTIC embeds the variable name at the beginning of the Effect column.

Similar differences can be seen in the way these tables handle variable values. PROC LOGISTIC's ParameterEstimates table stores variable values (such as Past Smoker and Current Smoker) in the ClassVar0 column, while its OddsRatios table embeds pairs of these values within the Effect column. Finally, PROC FREQ's CrossTabFreqs table stores these values in the column that is specifically dedicated to the variable in question.

Finally, the tables are not all consistent in their identification of statistics produced by the procedures. Most tables dedicate a separate column to each statistic (Estimate, StdErr, etc.), using the name of the statistic as the variable name for the column. The Moments and BasicMeasures tables produced by PROC UNIVARIATE, however, employ an idiosyncratic arrangement that stores 2 statistics on each record, mirroring the display in the SAS Listing that presents the statistics in two columns.

Figure 1: PROC UNIVARIATE ODS Tables

TABLE: Moments

VarName	Label1	cValue1	nValue1	Label2	cValue2	nValue2
Age	N	11458	11458	Sum Weights	11458	11458
age	Mean	49.3183802	49.318380	Sum Observations	565090	565090
age	Std Deviation	13.341921	13.341921	Variance	178.006856	178.006856
age	Skewness	0.05781753	0.057818	Kurtosis	-0.7172626	-0.717263
age	Uncorrected SS	29908748	29908748	Corrected SS	2039424.55	2039425
age	Coeff Variation	27.0526342	27.052634	Std Error Mean	0.12464187	0.124642

/***/

TABLE: BasicMeasures

VarName	LocMeasure	LocValue	VarMeasure	VarValue
age	Mean	49.31838	Std Deviation	13.34192
age	Median	49.00000	Variance	178.00686
age	Mode	42.00000	Range	73.00000
age		_	Interquartile Range	21.00000

Figure 2: PROC FREQ CrossTabFreqs Table

Table	education	diabetes	_TYPE_	Frequency	Percent	Row Percent	Col Percent	smoke
Table education * diabetes	<=High School	.	11	60	.	.	.	.
Table education * diabetes	<=High School	No	11	4542	40.993	72.6371	57.0890	.
Table education * diabetes	<=High School	Yes	11	1711	15.442	27.3629	54.7695	.
Table education * diabetes	<=High School	.	10	6253	56.435	.	.	.
Table education * diabetes	>High School	.	11	318	.	.	.	.
Table education * diabetes	>High School	No	11	3414	30.812	70.7272	42.9110	.
Table education * diabetes	>High School	Yes	11	1413	12.753	29.2728	45.2305	.
Table education * diabetes	>High School	.	10	4827	43.565	.	.	.
Table education * diabetes	.	.	01	.	.	.	.	.
Table education * diabetes	.	No	01	7956	71.805	.	.	.
Table education * diabetes	.	Yes	01	3124	28.195	.	.	.
Table education * diabetes	.	.	00	11080	100.000	.	.	.
Table smoke * diabetes	.	.	11	352	.	.	.	Never Smoked
Table smoke * diabetes	.	No	11	4379	39.522	72.8377	55.0402	Never Smoked
Table smoke * diabetes	.	Yes	11	1633	14.738	27.1623	52.2727	Never Smoked
Table smoke * diabetes	.	.	10	6012	54.260	.	.	Never Smoked
Table smoke * diabetes	.	.	11	20	.	.	.	Past Smoker
Table smoke * diabetes	.	No	11	2615	23.601	71.8802	32.8683	Past Smoker
Table smoke * diabetes	.	Yes	11	1023	9.233	28.1198	32.7465	Past Smoker
Table smoke * diabetes	.	.	10	3638	32.834	.	.	Past Smoker
Table smoke * diabetes	.	.	11	6	.	.	.	CurrentSmoker
Table smoke * diabetes	.	No	11	962	8.682	67.2727	12.0915	CurrentSmoker
Table smoke * diabetes	.	Yes	11	468	4.224	32.7273	14.9808	CurrentSmoker
Table smoke * diabetes	.	.	10	1430	12.906	.	.	CurrentSmoker
Table smoke * diabetes	.	.	01	.	.	.	.	.
Table smoke * diabetes	.	No	01	7956	71.805	.	.	.
Table smoke * diabetes	.	Yes	01	3124	28.195	.	.	.
Table smoke * diabetes	.	.	00	11080	100.000	.	.	.

Figure 3: PROC LOGISTIC ODS Tables

TABLE: ParameterEstimates

Variable	ClassVal0	DF	Estimate	StdErr	WaldChiSq	Prob ChiSq
Intercept		1	-1.0358	0.0355	850.8952	<.0001
education	>High School	1	0.1041	0.0426	5.9544	0.0147
smoke	Past Smoker	1	0.0550	0.0470	1.3703	0.2418
smoke	Current Smoker	1	0.2745	0.0635	18.6858	<.0001

/************************ **/

TABLE: OddsRatios

Effect	Odds RatioEst	LowerCL	UpperCL
education >High School vs <=High School	1.110	1.021	1.206
smoke Past Smoker vs Never Smoked	1.057	0.964	1.159
smoke Current Smoker vs Never Smoked	1.316	1.162	1.490

/***/

TABLE: ResponseProfile

OrderedValue	Outcome	Var2N
1	Yes	3124
2	No	7956

What Would We Like for SAS to Give Us?

After wrestling with the inconsistencies illustrated above in our efforts to produce reports that combined data from multiple procedures, we decided to come up with a more consistent general structure for these tables. We could then develop tools to convert the existing ODS dataset output into our preferred format, as well as to much more easily combine output from multiple procedures. The rules for our new common structure are as follows:

1) Use a separate column for each statistic, using the name of the statistic as the column header
2) Store the name of the analytic variable in a column called VarName
3) If data are provided for multiple categorical levels of the analytic variable, use a separate record for each level, store the numeric unformatted value in a column named VarValue and the formatted value in a column named VarFmt.
4) If the ODS output contains statistics for multiple variables that are associated with each other, adjust the column names accordingly, e.g. VarName1, VarName2. If VarName1 is a categorical variable, maintain a separate record for each level of this variable, however, represent multiple levels for additional variables using separate columns, rather than separate records, with appropriate descriptive names (This can best be understood by looking at Figures 5 and 6 below).
5) Retrieve and store the variable label for each analytic variable, with column names of VarLabel or VarLabel1, VarLabel2, etc., if appropriate.
6) Use standard lengths for all character variables: 32 for those containing variable names, 256 for those containing variable labels or formatted variable values.
7) Add a key variable column to reflect the original specified order of the "primary" variable, VarName1, in the output. This allows us to return to this order after using variable names to merge the output from different procedures.

Figures 4, 5, and 6 below illustrate our new common structure to output from PROC UNIVARIATE, PROC FREQ and PROC LOGISTIC. (Variable labels were omitted from Figures 5 and 6 due to space constraints.) This structure has made it much easier for us to combine, summarize and report analytic results from within and between SAS procedures.

Figure 4: Restructured, Combined Output for Tables from PROC UNIVARIATE

Obs	Var Name	Var Label	N	Mean	StdDev	Kurtosis	Median	Univariate Key
1	age	Participant Age	11458	49.3184	13.3419	-0.71726	49.00	1

Figure 5: Restructured dataset for CrossTabFreqs Table from PROC FREQ

Obs	FKey	VName1	VName2	_TYPE_	VValue1	VLabel1	VFmt1	VValue2_1	VValue2_2	VValue2_3	VFmt2_1	VFmt2_2	VFmt2_3	CFrequency_1	CFrequency_2	CFrequency_3	CTotal
1	EDUCATION	DIABETES	01	.	.	.	.	0	1	.	No	Yes	.	7956	3124	.	.	.	11080
1	EDUCATION	DIABETES	11	0	<=High School	.	0	1	.	No	Yes	60	4542	1711	.	57	54	6253
1	EDUCATION	DIABETES	11	1	>High School	.	0	1	.	No	Yes	318	3414	1413	.	42	45	4827
2	SMOKE	DIABETES	01	.	.	.	.	0	1	.	No	Yes	.	7956	3124	.	.	.	11080
2	SMOKE	DIABETES	11	0	Never Smoked	.	0	1	.	No	Yes	352	4379	1633	.	55	52	6012
2	SMOKE	DIABETES	11	1	Past Smoker	.	0	1	.	No	Yes	20	2615	1023	.	32	32	3638
2	SMOKE	DIABETES	11	2	Current Smoker	.	0	1	.	No	Yes	6	962	468	.	12	14	1430

Figure 6: Composite LogisticStats Dataset

Obs	VarName1	Var Value1	Prob Chisq	Odds Ratio	LowerCL	UpperCL	Logistic Key	VArLabel1	VarFmt1
1	education	0	.	.	.	.	1	Education	<=High School
2	education	1	0.0147	1.110	1.021	1.206	1	Education	>High School
3	smoke	0	.	.	.	.	2	Smoke Status	Never Smoked
4	smoke	1	0.2418	1.057	0.964	1.159	2	Smoke Status	Past Smoker
5	smoke	2	<.0001	1.316	1.162	1.490	2	Smoke Status	Current Smoker

bs	VarName2	VarLabel2	Var Value2_1	Var Value2_2	Var Fmt2_1	Var Fmt2_2	VarN2_1	VarN2_2
1	diabetes	Dx with Diabetes	0	1	No	Yes	6937	2652
2			.	.			.	.
3			.	.			.	.
4			.	.			.	.
5			.	.			.	.

REDESIGNING OUTPUT DATASETS: A HARDCODED APPROACH

Restructuring PROC UNIVARIATE Output

Regardless of how we plan to manipulate the ODS Table data to get the statistics we want to present, the first step is always to output the information stored in ODS Tables into datasets using the ODS OUTPUT statement. However, we cannot specify the ODS OUTPUT statement until we know the names of the ODS Tables that contain the statistics we wish to retrieve.

The ODS TRACE statement provides an easy way to find the names of ODS Tables. To execute an ODS TRACE, embed the procedure code between ODS TRACE ON and ODS TRACE OFF, as shown below:

```
ODS TRACE ON / LISTING;
PROC UNIVARIATE DATA=mydata;
  VAR age; RUN;
ODS TRACE OFF;
```

After running this code, we can review the SAS Listing (partial output shown in Figure 7) to determine which ODS Tables contain the statistics we need. The LISTING option of the ODS TRACE statement prints the name of each ODS Table above the output generated from that table.

With ODS Table names in hand, we can specify the ODS OUTPUT statement. The following code creates datasets named Moments and BasicMeasures from the Moments and BasicMeasures Tables, respectively:

```
ODS OUTPUT Moments=Moments
           BasicMeasures=BasicMeasures;
PROC UNIVARIATE DATA=mydata;
  VAR age;  RUN;
ODS OUTPUT CLOSE;
```

Now that the information in the ODS Tables has been output to datasets, our goal is to create a composite summary dataset with one observation per summarized analytic variable, as well as a separate column containing each statistic.

Figure 7: SAS Listing

```
The UNIVARIATE Procedure
Variable:  age  (Participant Age)

Output Added:
-------------
Name:       Moments
Label:      Moments
Template:   base.univariate.Moments
Path:       Univariate.age.Moments
-------------
                          Moments
N                     11458    Sum Weights             11458
Mean              49.3183802    Sum Observations       565090
Std Deviation      13.341921    Variance              178.006856
Skewness          0.05781753    Kurtosis              -0.7172626
Uncorrected SS      29908748    Corrected SS          2039424.55
Coeff Variation   27.0526342    Std Error Mean        0.12464187

Output Added:
-------------
Name:       BasicMeasures
Label:      Basic Measures of Location and Variability
Template:   base.univariate.Measures
Path:       Univariate.age.BasicMeasures
-------------

               Basic Statistical Measures
    Location                     Variability
Mean     49.31838     Std Deviation            13.34192
Median   49.00000     Variance                178.00686
Mode     42.00000     Range                    73.00000
                      Interquartile Range      21.00000
```

To accomplish this, we first impose this structure on each individual ODS output dataset, and then merge them together by the analytic variable name. The following code converts dataset Moments (see Figure 1) to the desired structure:

```
DATA Moments (KEEP = VarName UnivariateKey n mean stddev kurtosis);
  LENGTH VarName $32;
  SET Moments;
    BY VarName;
  IF Label1="N"              THEN N        = nValue1;
  IF Label1="Mean"           THEN Mean     = nValue1;
  IF Label1="Std Deviation"  THEN StdDev   = nValue1;
  IF Label2="Kurtosis"       THEN Kurtosis = nValue2;
  IF LAST.VarName THEN DO;
     UnivariateKey+1;
     OUTPUT Moments;
  END;
  RETAIN n mean stddev kurtosis;  RUN;
```

Since we may want to present the statistics from PROC UNIVARIATE in the order that they were run, we have preserved that order in variable UnivariateKey. This variable only need be created on one of the two PROC UNIVARIATE output datasets since these datasets will be merged together by analytic variable name.

The following code converts BasicMeasures (see Figure 1) to the desired structure:

```
DATA BasicMeasures (KEEP = VarName Median);
  LENGTH VarName $32;
  SET BasicMeasures;
    BY VarName;
  IF LocMeasure="Median" THEN Median=LocValue;
  IF LAST.VarName        THEN OUTPUT BasicMeasures;
  RETAIN Median;
RUN;
```

We now have each separate ODS output dataset in the desired structure, and have also standardized the variable identifying the analytic variable name as VarName, with a standard length of 32 characters (the SAS maximum). The only remaining items to retrieve are the labels for the analytic variables. However, the labels cannot be found in any of the ODS Tables produced by PROC UNIVARIATE. This brings us to the realization that while all of the statistics generated by a procedure can be found in ODS Tables, not all of the descriptive information found in the SAS Listing can be found in ODS Tables. We will have to look external to PROC UNIVARIATE to capture the labels we need. The ODS Tables produced by PROC CONTENTS provide an efficient solution for locating these labels that is both easy to use and understand. The following code outputs labels for Age to dataset Contents:

```
ODS OUTPUT Variables=Variables (KEEP = Variable Label
                                RENAME = (Variable = VarName
                                          Label    = VarLabel));
PROC CONTENTS DATA=mydata (KEEP = age);
RUN;
ODS OUTPUT CLOSE;
```

The Variables ODS Table in PROC CONTENTS always sets the length of Variable to 32 characters, so we only need to rename it to VarName to complete the standardization of this variable. However, the variable Label from this ODS Table varies in length depending on the data. So in addition to renaming Label to VarLabel, we will need to reset its length to our standard of 256 characters.

The code below creates our final composite dataset, UnivariateStats (displayed in Figure 4), by merging together the Moments, BasicMeasures, and Variables datasets while resetting the length of VarLabel:

```
DATA UnivariateStats;
  LENGTH VarLabel $256;
  MERGE Variables BasicMeasures Moments;
    BY VarName;
RUN;
```

Restructuring PROC FREQ Output– OneWayFreqs Table

PROC FREQ has multiple ODS Tables to handle the storage of frequencies and percentages. Table OneWayFreqs handles those involving only one variable, while CrossTabFreqs handles crosstabulations of two or more variables. And in what is a rare feature associated with PROC FREQ, these ODS Tables retain all of the analytic variable attributes that we need to conform the data to our set of rules for a common structure. In the examples that follow for PROC FREQ (and also for PROC LOGISTIC), we will limit the analytic variables processed to categorical numeric variables, which for our purposes is defined as those numeric variables that have a one-to-one relationship between unformatted and formatted values.

The following code outputs table OneWayFreqs to dataset OneWayFreqs (displayed in Figure 8):

```
ODS OUTPUT OneWayFreqs=OneWayFreqs(drop=F_diabetes F_smoke);
PROC FREQ DATA=mydata;
  TABLES diabetes smoke / MISSPRINT;
RUN;
ODS OUTPUT CLOSE;
```

Since the numeric variables Diabetes and Smoke are passed through to OneWayFreqs with formats still attached, we can discard the text variables F_Diabetes and F_Smoke, which PROC FREQ adds to hold the text for the formatted values of Diabetes and Smoke. All other variables in the OneWayFreqs dataset are numeric except for the character variable Table, which contains the analytic variable name as part of the table request.

Figure 8: PROC FREQ OneWayFreqs ODS Table

Obs	Table	diabetes	Frequency	Percent	Cum Frequency	Cum Percent	smoke
1	Table diabetes	.	378	.	.	.	.
2	Table diabetes	No	7956	71.81	7956	71.81	.
3	Table diabetes	Yes	3124	28.19	11080	100.00	.
4	Table smoke	.	6364	55.54	6364	55.54	Never Smoked
5	Table smoke	.	3658	31.93	10022	87.47	Past Smoker
6	Table smoke	.	1436	12.53	11458	100.00	Current Smoker

Now we want to convert OneWayFreqs to our desired structure, which is: (1) one observation per analytic variable level; (2) separate variables for the analytic variable name, label, (numeric) unformatted value, and (character) formatted value; and (3) one variable for each statistic calculated, i.e., the frequencies and percentages. While there are many methods available for converting the data to our new structure, using DATA Step arrays (as shown in the code below) has the advantage of preserving any special missing values in the analytic variables.

```
DATA OneWayFreqs (KEEP = VarName VarFmt VarLabel VarValue
                         frequency percent cumfrequency cumpercent);
  LENGTH VarName   $32
         VarLabel
         VarFmt    $256
         VarValue  8;
  SET OneWayFreqs;
  ARRAY vars {*} diabetes smoke;
  VarName=SCAN(table,2,' ');
  DO I = 1 TO DIM(vars);
    IF UPCASE(VarName)=UPCASE(VNAME(vars(i))) THEN DO;
      VarFmt   =VVALUE(vars(i));
      VarLabel =VLABEL(vars(i));
      VarValue =vars(i);
      RETURN;
    END;
  END;
RUN;
```

The DATA Step above first extracts the value for VarName from the Table variable. It then searches for the value of VarName in the array VARS, which contains the same variable names that were in the TABLES statement of our PROC FREQ. Once a match is found, VarLabel, VarValue, and VarFmt are populated. Figure 9 displays the restructured OneWayFreqs dataset.

Figure 9: Restructured dataset for OneWayFreqs Table from PROC FREQ

Obs	VarName	VarLabel	VarFmt	Value	Frequency	Percent	Cum Frequency	Cum Percent
1	diabetes	Dx with Diabetes	.	.	378	.	.	.
2	diabetes	Dx with Diabetes	No	0	7956	71.81	7956	71.81
3	diabetes	Dx with Diabetes	Yes	1	3124	28.19	11080	100.00
4	smoke	Smoking Status	Never Smoked	0	6364	55.54	6364	55.54
5	smoke	Smoking Status	Past Smoker	1	3658	31.93	10022	87.47
6	smoke	Smoking Status	Current Smoker	2	1436	12.53	11458	100.00

An anomaly was found in the structure of the OneWayFreqs output dataset when analytic variables in the TABLES statement had a length less than 8 bytes. After the first observation where variables for the raw and formatted values were populated, values were retained on all subsequent observations, regardless of whether the analytic variable was part of the Table request for that observation. No such problem was observed with the CrossTabFreqs output dataset. This issue was still outstanding in SAS 9.2 TS Level 2M3 for the XP_PRO platform.

Restructuring PROC FREQ Output - CrossTabFreqs

One of the most common reporting needs is to present statistics for one variable in its relationship with another variable. Crosstabulations of two variables, with levels of one variable presented in rows and levels of the other variable presented in columns and frequencies and percentages in the body of the display, are a basic, standard way to present the relationship of one variable to another.

While both the CrossTabFreqs and List tables can output the results of *n*-way crosstabulations, we have chosen to focus on CrossTabFreqs for the examples in this paper. All crosstabulations requested in PROC FREQ can be output to CrossTabFreqs ODS Table. However, since this table is designed to handle *n*-way tables and not just the two-way tables we are interested in, the row and column structure described in our rules for design is not available in this ODS Table. Instead, it is structured so that the frequency for each cell of our desired display is on a different row of the ODS Table, and furthermore, like the OneWayFreqs ODS Table, each analytic variable occupies its own column in the output dataset. This can be an unwieldy dataset to use for generating reports or performing data merges with other ODS Tables.

To present the data in our desired display, we need to restructure (transpose) the values of the second variable so that we have separate frequency variables for each level of the second variable, while maintaining one observation for each level of the first variable. We will have separate generic variables for name and label of the two analytic variables in the

crosstabulation, eliminating the structure with each analytic variable in its own column. To accomplish this, we will simply elaborate further on the DATA Step array method used for the OneWayFreqs Table. The following code produces the raw CrossTabFreqs output dataset displayed in Figure 2, while Figure 5 shows our desired final dataset structure.

```
ODS OUTPUT CrossTabFreqs=CrossTabFreqs;
PROC FREQ DATA=mydata;
  TABLES (education smoke)*diabetes / MISSPRINT;  RUN;
ODS OUTPUT CLOSE;
```

Since we are now generating statistics for the relationship between 2 variables, we will identify our analytic variables by VarName1 (EDUCATION SMOKE) and VarName2 (D IABETES). Each of the variables containing descriptive information will contain this ordinal naming convention.

We will begin the alteration by creating the four pieces of descriptive information for the VarName1 and VarName2 variables, namely, analytic variable name & label, and formatted & unformatted values for the level of the analytic variable.

```
DATA CrossTabFreqs (KEEP = FreqKey VarName: VarFmt: VarLabel: VarValue: _TYPE_ frequency
                           colpercent);
  LENGTH VarName1   VarName2    $32
         VarFmt1    VarFmt2
         VarLabel1  VarLabel1  $256 ;
  ARRAY var1ary {*} education smoke;
  ARRAY var2ary {*} diabetes;
  SET CrossTabFreqs ;
    BY Table notsorted;
  IF FIRST.Table then FreqKey+1;
  VarName1=UPCASE(SCAN(TABLE,2," "));
  VarName2=UPCASE(SCAN(TABLE,4," "));
  DO i = 1 TO DIM(var1ary);
     IF VarName1  = UPCASE(VNAME(var1ary(i))) THEN DO;;
        VarFmt1   = VVALUE(var1ary(i));
        VarLabel1 = VLABEL(var1ary(i));
        VarValue1 = var1ary(i);
     END;
  END;
  DO i = 1 TO DIM(var2ary);
     IF VarName2  = UPCASE(VNAME(var2ary(i))) THEN DO;;
        VarFmt2   = VVALUE(var2ary(i));
        VarLabel2 = VLABEL(var2ary(i));
        VarValue2 = var2ary(i);
     END;
  END;    RUN;
```

Variables representing levels of VarName2 and association statistics between VarName1 and VarName2 will be propagated across the observations containing the levels of VarName1. The VarName2 and VarLabel2 variables will not be propagated, but rather appear once on each observation, which leaves us with four items to transpose:

VarValue2, VarFmt2, Frequency and ColPercent.

Before transposing, we will sort the CrossTabFreqs dataset by four items:

FreqKey: for retaining the original order of pairs of variables
TYPE: for identifying how the frequencies are associated with the two variables
VarValue1: to group all observations for EDUCATION and SMOKE
VarValue2: to order DIABETES levels for propagation across each level of EDUCATION and SMOKE

As pointed out, the _TYPE_ variable will identify the types of associations that the frequencies represent, as can be seen in Figure 3::

"00": Identifies the overall frequency total for the association of EDUCATION and SMOKE crossed with DIABETES.
"01": Identifies column total counts and percents for each level of DIABETES.
"10": Identifies row total counts and percents for each level of EDUCATION and SMOKE.
"11": Identifies frequencies and percents for each cell produced by recorded values of EDUCATION and SMOKE crossed with DIABETES.

The following code then sorts the data and divides it into two data sets – one to be transposed, and one to be merged with the transposed data. The dataset to be transposed, CrossTabFreqs11, will contain observations where _TYPE_ is "01" or "11".

The variable Var2Ord will be created to provide an ordinal suffix for each variable transposed by PROC TRANSPOSE. This dataset contains the observations with data that will be transposed. The second dataset, CrossTabFreqs00, contains observations where _TYPE_ equals “10”, row totals, and “00”(the total count across all cells). On CrossTabFreqs00, the value of _TYPE_ will be changed to “11” and “01” respectively to facilitate final merging of column and overall totals.

```
PROC SORT DATA=CrossTabFreqs;
  BY FreqKey _TYPE_ VarValue1 VarValue2;  RUN;

DATA CrossTabFreqs11
     CrossTabFreqs00 (KEEP   = VarName1 VarLabel1 VarValue1 VarFmt1 VarName2 VarLabel2
                               _TYPE_ FreqKey frequency
                      RENAME = (frequency=total));
  SET CrossTabFreqs;
    BY FreqKey _TYPE_ VarValue1 VarValue2;
  IF FIRST.VarValue1 THEN Var2Ord=0;
  IF FIRST.VarValue2 THEN Var2Ord+1;
  SELECT(_TYPE_);
    WHEN("01","11")
        OUTPUT CrosstabFreqs11;
    WHEN("00","10") DO;
        _TYPE_=CATS(SUBSTR(_TYPE_,1,1),"1");
        OUTPUT CrossTabFreqs00;
    END;
  END;    RUN;
```

Due to the constraints of PROC TRANSPOSE, each variable must be transformed independent of the others, after which a merge will be performed to associate the transformed variables with the proper level of VarName1. This code shows an example of how the first PROC TRANSPOSE will be written to handle propagating the VarValue2 variable for each level of VarName1. VarFmt2, Frequency and Colpercent will be transposed in identical fashion. The following code produces the dataset seen in Figure 10.

```
PROC TRANSPOSE DATA=CrossTabFreqs11 OUT=TranCTF1 (DROP=_NAME_ _LABEL_)
               PREFIX=VarValue2_;
  BY FreqKey VarName1 VarLabel1 VarName2 VarLabel2 _TYPE_ VarValue1 VarFmt1;
  ID var2ord;
  VAR VarValue2;    RUN;
```

Four datasets, TranCTF1 to TranCTF4, will be created by the four executions of PROC TRANSPOSE. The final, restructured dataset is then created by merging these four datasets together with CrossTabFreqs00 (Figure 11) to create the final dataset seen in Figure 5 on page four of this paper.

```
DATA FreqStats;
  MERGE TranCTF1 TranCTF2 TranCTF3 TranCTF4 CrossTabFreqs00;
    BY FreqKey VarName1 VarName2 _type_ VarValue1 ;   RUN;
```

Figure 10: Tranposed dataset: TranCTF1

									V	V	V
									a	a	a
		V		V		V			r	r	r
	V	a	V	a		a			V	V	V
F	a	r	a	r		r	V		a	a	a
R	r	L	r	L	_	V	a		l	l	l
e	N	a	N	a	T	a	r		u	u	u
q	a	b	a	b	Y	l	F		e	e	e
K	m	e	m	e	P	u	m		2	2	2
E	e	l	e	l	E	e	t		_	_	_
y	1	1	2	2	_	1	1		1	2	3
1	EDUCATION	Education	DIABETES	Dx with Diabetes	01	.		.	.	0	1
1	EDUCATION	Education	DIABETES	Dx with Diabetes	11	0	<=High School		.	0	1
1	EDUCATION	Education	DIABETES	Dx with Diabetes	11	1	>High School		.	0	1
2	SMOKE	Smoke Status	DIABETES	Dx with Diabetes	01	.		.	.	0	1
2	SMOKE	Smoke Status	DIABETES	Dx with Diabetes	11	0	Never Smoked		.	0	1
2	SMOKE	Smoke Status	DIABETES	Dx with Diabetes	11	1	Past Smoker		.	0	1
2	SMOKE	Smoke Status	DIABETES	Dx with Diabetes	11	2	Current Smoker		.	0	1

Figure 11: CrossTabFreqs00

VName1	VName2	VFmt1	VLabel1	VLabel2	_TYPE_	_Total_	FreqKey	VValue1
EDUCATION	DIABETES	.	Education	Dx with Diabetes	01	11080	1	.
EDUCATION	DIABETES	<=High School	Education	Dx with Diabetes	11	6253	1	0
EDUCATION	DIABETES	>High School	Education	Dx with Diabetes	11	4827	1	1
SMOKE	DIABETES	.	Smoke Status	Dx with Diabetes	01	11080	2	.
SMOKE	DIABETES	Never Smoked	Smoke Status	Dx with Diabetes	11	6012	2	0
SMOKE	DIABETES	Past Smoker	Smoke Status	Dx with Diabetes	11	3638	2	1
SMOKE	DIABETES	Current Smoker	Smoke Status	Dx with Diabetes	11	1430	2	2

We have focused on two-way tables in this example, but the same methods can be expanded for *n*-way tables by having the variables represent cross-tabulations of multiple variables (e.g., males under age 50, etc.), or by having each observation represent the same kind of crosstabulation, or by doing both if you have you have 4-way (or more) table.

Restructuring PROC LOGISTIC Output

Regression models in SAS usually do not provide all of the descriptive information in ODS Tables. In addition, some of the descriptive information that is available may be embedded within character variables found in the tables. In PROC LOGISTIC, if a numeric categorical analytic variable has a format attached, then the ODS Tables will show the text formatted value when displaying the levels of that categorical variable. If the variables are not formatted, then raw numeric values are displayed. Using raw numeric values offers advantages over format character strings. First, it is easier to identify numeric category values when they are embedded within a character string. Second, it is easier to re-associate formatted character data with numeric values rather than the other way around.

To create our composite summary dataset from PROC LOGISTIC ODS Tables, we also need to retrieve the descriptive information in addition to retrieving chi square probabilities and odds ratios with corresponding confidence intervals. We will output the ParameterEstimates table for chi square probabilities and the OddsRatios table for odds ratios and confidence intervals. We will also output a third dataset from the ResponseProfile table in order to include data describing the response variable in our final dataset. In order to keep this example simple, we will use only numeric categorical variables.

```
ODS OUTPUT ParameterEstimates=ParameterEstimates
           OddsRatios=OddsRatios
           ResponseProfile=ResponseProfile  ;
PROC LOGISTIC DATA = mydata NAMELEN=32;
  CLASS education smoke / ORDER=INTERNAL PARAM=REF REF=FIRST;
  MODEL diabetes(ORDER=INTERNAL DESCENDING ) = education smoke ;
RUN;
ODS OUTPUT CLOSE;
```

Figure 3 gives us a look at how the three ODS Tables produced by the code above are designed. Our goal is to create a summary dataset that contains the statistics of interest (highlighted in Figure 3) from these three datasets as well as our four pieces of descriptive information (analytic variable name & label as well as formatted & unformatted values for each analytic variable level). Figure 6 contains a printout of what our final dataset will look like. In Figure 6, we see that all of the statistics are associated with a category level of each exposure variable in one concise dataset. The name, values and counts for the response variable and its corresponding levels, are stored on the first observation as a header record for easy access.

Each of the datasets created from these ODS Tables will require subsequent DATA Step processing. To better facilitate the coding methods needed for each DATA Step, we will remove as much of the extraneous information and unneeded observations as is possible at the time of their creation. We can also rename the variables that contain descriptive information at this time. Figure 12 shows the output datasets created by the code below.

```
ODS OUTPUT ParameterEstimates= ParameterEstimates
                                (KEEP = variable classval0 probchisq
                                 RENAME = (Variable = VarName1)
                                 WHERE = (VarName1 NE "Intercept"))
           OddsRatios        = OddsRatios
                                (KEEP = effect oddsratioest lowercl uppercl
                                 RENAME = (OddsRatioEst=OddsRatio))
           ResponseProfile=ResponseProfile
                                (KEEP= outcome count
                                 RENAME = (count=VarN2)) ;

PROC LOGISTIC DATA = mydata NAMELEN=32;
        CLASS education smoke / ORDER=INTERNAL PARAM=REF REF=FIRST;
    MODEL diabetes(ORDER=INTERNAL DESCENDING ) = education smoke;
    FORMAT _numeric_;
RUN;
ODS OUTPUT CLOSE;
```

Figure 12: PROC LOGISTIC adjusted datasets from ODS Tables

TABLE: ParameterEstimates

Obs	VarName1	Class Val0	Prob ChiSq
1	education	1	0.0147
2	smoke	1	0.2418
3	smoke	2	<.0001

/***/

TABLE: OddsRatios

Obs	Effect	Odds Ratio	LowerCL	UpperCL
1	education 1 vs 0	1.110	1.021	1.206
2	smoke 1 vs 0	1.057	0.964	1.159
3	smoke 2 vs 0	1.316	1.162	1.490

/***/

TABLE: ResponseProfile

Obs	Outcome	VarN2
1	1	3124
2	0	7956

While the above code can create VarName1 with a RENAME statement on the ParameterEstimates Table and a subsequent assignment statement (shown in code below) can easily create VarValue1, these variables are not as easily created from the OddsRatio Table. In the OddsRatio Table, these variable must be extracted from the character variable, EFFECT.

We want our final PROC LOGISTIC composite dataset to have one observation for each level of VarName1, including the referent level, with descriptive information available on all observations, and relational statistics available on all except the observation for the referent level. To set this up, we need to create two datasets from OddsRatios – one with referent level data (containing only descriptive information) and one with non-referent level data (containing descriptive information and relational statistics). These will later be merged together with the ParameterEstimates dataset.

The following code creates the final ParameterEstimates dataset and the two OddsRatio datasets discussed, which are displayed in Figures 13, 14, and 15:

```
DATA ParameterEstimates ( KEEP = VarName1 VarValue1 probchisq );
  LENGTH VarName1 $32;
  SET ParameterEstimates ;
    BY VarName1 NOTSORTED;
  VarValue1 = INPUT(classval0,BEST8.);
RUN;

DATA OddsRatios  (KEEP = VarName1 VarValue1 LogisticKey OddsRatio LowerCL UpperCL)
     OddsRatios0 (KEEP = VarName1 VarValue1 LogisticKey);
  LENGTH VarName1 $32;
  SET OddsRatios ;
  VarName1  = SCAN(EFFECT,1);
  VarValue1 = INPUT(SCAN(EFFECT,2),BEST8.);
  IF VarName1^=LAG(VarName1) THEN DO;
     LogisticKey+1;
     OUTPUT OddsRatios;
     VarValue1 = INPUT(SCAN(EFFECT,4),BEST8.);
     OUTPUT OddsRatios0;
  END;
  ELSE OUTPUT OddsRatios;
RUN;
```

Figure 13: Adjusted Parameter Estimates Dataset

VarName1	Var Value1	prob chisq
education	1	0.0147
smoke	1	0.2418
smoke	2	<.0001

Figure 14: Adjusted OddsRatios Dataset

VarName1	Var Value1	Odds Ratio	LowerCL	UpperCL	Logistic Key
education	1	1.110	1.021	1.206	1
smoke	1	1.057	0.964	1.159	2
smoke	2	1.316	1.162	1.490	2

Figure 15: Adjusted Odds Ratios0 Dataset

VarName1	Var Value1	Logistic Key
education	0	1
smoke	0	2

Variable labels and formats will be retrieved from PROC CONTENTS:

```
ODS OUTPUT Variables=ExposureInfo (KEEP   =   Variable Label Format
                                   RENAME = (Variable = VarName1
                                             Label    = VarLabel1));
PROC CONTENTS DATA=mydata (KEEP = education smoke ); RUN;
ODS OUTPUT CLOSE;
```

We can now sort the ParameterEstimates, OddsRatios and OddsRatios0 datasets by VarName1 and VarValue1 followed by a DATA Step where these three datasets are merged together. Since our desired dataset also includes the data output from PROC CONTENTS, merged by VarName1, we can complete the addition of the descriptive information within the same DATA Step using Hash Table processing:

```
DATA LogisticStats (DROP=Format);
  LENGTH VarLabel1 VarFmt1 $256 ;
  IF 0 THEN SET ExposureInfo;
  DECLARE HASH c(DATASET:"ExposureInfo");
  c.DEFINEKEY("VarName1");
  c.DEFINEDATA(ALL:"yes");
  c.DEFINEDONE();
  DO UNTIL (done);
     MERGE ParameterEstimates OddsRatios OddsRatios0 END=done;
       BY VarName1 VarValue1;
     rc=c.FIND(); DROP rc;
     VarFmt1=PUTN(VarValue1,Format);
     OUTPUT LogisticStats;
  END;
  STOP; RUN;
```

By sorting the LogisticStats dataset by the LogisticKey variable, we will place the observations in the original order requested in the execution of the PROC LOGISTIC. Figure 16 displays the LogisticStats dataset after execution of the above DATA Step.

Figure 16: Composite LogisticStats Dataset

VarName1	Var Value1	Prob Chisq	Odds Ratio	LowerCL	UpperCL	Logistic Key	VArLabel1	VarFmt1
education	0	.	.	.	.	1	Education	<=High School
education	1	0.0147	1.110	1.021	1.206	1	Education	>High School
smoke	0	.	.	.	.	2	Smoke Status	Never Smoked
smoke	1	0.2418	1.057	0.964	1.159	2	Smoke Status	Past Smoker
smoke	2	<.0001	1.316	1.162	1.490	2	Smoke Status	Current Smoker

Our final task is to add the descriptive information for the response variable, VarName2. Logistic orders the response variable in descending order. For our purposes, we will sort the response variable in ascending order. We can retrieve the needed descriptive items for VarName2 from PROC CONTENTS and merge it with the ResponseProfile dataset. But first, we must restructure the ResponseProfile dataset.

```
DATA ResponseProfile ;
  LENGTH VarName2 $32  ;
  MERGE ResponseProfile ;
  VarName2="diabetes";
  Var2Ord=_N_;
  VarValue2=INPUT(outcome,8.0);
RUN;
```

We can now add the descriptive information from the ResponseInfo dataset (output from the Variables Table from PROC CONTENTS):

```
DATA ResponseProfile ;
  LENGTH VarLabel2 VarFmt2 $256 ;
  MERGE ResponseProfile ResponseInfo  ;
  VarFmt2=PUTN(VarValue2,Format);
RUN;
```

Two descriptive items (VarValue2 and VarFmt2) and one statistic (VarN2) will be propagated across one observation using PROC TRANSPOSE in the same manner as was seen in the CrossTabFreqs example. We will also create the order variable for use in PROC TRANSPOSE. We will display the code for transposing VarValue2.

```
PROC TRANSPOSE DATA=ResponseProfile OUT=TranRP1 (DROP=_NAME_ _LABEL_)
               PREFIX=VarValue2_;
  BY VarName2 VarLabel2;
  ID var2ord;
  VAR VarValue2;
RUN;
```

After creating a composite dataset of these transposed response items:

```
DATA TranRP;
  MERGE TranRP1 TranRP2 TranRP3 ;
    BY VarName2 VarLabel2;
RUN;
```

we will merge it onto the first observation of dataset containing the OddsRatios and ParameterEstimates summary information. Once TranRP is merged with LogisticStats, we have the dataset that can be seen in Figure 6 on Page 4 of this paper.

```
DATA LogisticStats;
  MERGE LogisticStats TranRP;
RUN;
```

AUTOMATE THE PROCESS

The code presented in this paper up to now, has centered on creating adjusted ODS output datasets for specific variables. We can design modules of SAS code which are capable of processing these analytic tasks on generic sets of variables. Within SAS, the SAS Macro Facility is a tool that can be used to automate general tasks that we wish to compartmentalize or repeat. We have created a macro module for each procedure that we wish to generate a summary set of statistics conforming to our set of rules of design. Each macro module executes independently, given the parameters passed during the time of invocation, and returns a single product: a concise restructured output dataset from one or more ODS Tables generated by the procedure executed in the macro module.

Automating PROC UNIVARIATE

The PROC UNIVARIATE example, that we have presented, produces a summary dataset which contains select statistics for all recorded observations of the input dataset across the processed variables. In our automated version, we will add a CLASS statement to the PROC UNIVARIATE and retrieve statistics from the Moments Table, only. This will require that we use the variables VarName1 and VarName2, where VarName1 contains the names of the processed variables and VarName2 contains the name of the CLASS variable.

We will pass, as parameters:

1) the name of the input dataset
2) the list of variables to process
3) the name of the CLASS variable
4) the name of the output dataset.

```
%procUnivariate(data          = mydata  ,
                varList       = age bmi ,
                class         = diabetes,
                UnivariateOut = UnivariateStats    )
```

Figure 17 contains the listing of the dataset of summary statistics from the Moments table that is returned by the ProcUnivariate macro. The ProcUnivariate macro can be found in Appendix 2.

Figure 17: Dataset Returned by %ProcUnivariate (CLASS Statement Added)

VarName1	VarLabel1	VarName2	VarLabel2	Univariate Key	Value1	Value2	Mean1	Mean2	StdDev1	StdDev2
age	Patient Age	diabetes	Dx With Diabetes	1	0	1	No	Yes	49.3367	48.8451	14.7130	12.6577
bmi	Body Mass Index	diabetes	Dx With Diabetes	2	0	1	No	Yes	27.5817	27.0641	4.6234	4.1684

Automating PROC FREQ

The ProcFreq macro module (found in Appendix 3) contains automated code based directly on the hard coded example presented in this paper. The macro module returns a dataset identical to the one in Figure 5. Parameters passed include:

1) the name of the input dataset
2) the list of row variables
3) the variable(s) to contain column statistics
4) the name of the dataset of summary statistics returned by the macro.

```
%ProcFreq(data     = mydata ,
          var1List = education smoke ,
          var2List = diabetes,
          out      = FreqStats    )
```

Figure 18 contains the listing for this dataset.

Figure 18: The FreqStats Dataset Returned by %ProcFreq

Freq Key	Var Name 1	Var Name 2	_Type_	Var Value 1	Var Fmt 1	Var Value 1_1	Var Value 2_1	Var Value 3_1	Var Value 1_2	Var Value 2_2	Var Value 3_2	Freq Query 1_1	Freq Query 2_1	Freq Query 3_1	Col Percent 1_1	Col Percent 2_1	Col Percent 3_1	Total
1	education	diabetes	01	.	.	.	0	1	.	No	Yes	.	7956	3124	.			11080
1	education	diabetes	11	0	<=High School	.	0	1	.	No	Yes	60	4542	1711	.	57	54	6253
1	education	diabetes	11	1	>High School	.	0	1	.	No	Yes	318	3414	1413	.	42	45	4827
2	smoke	diabetes	01	.	.	.	0	1	.	No	Yes	.	7956	3124	.			11080
2	smoke	diabetes	11	0	Never Smoked	.	0	1	.	No	Yes	352	4379	1633	.	55	52	6012
2	smoke	diabetes	11	1	Past Smoker	.	0	1	.	No	Yes	20	2615	1023	.	32	32	3638
2	smoke	diabetes	11	2	Current Smoker	.	0	1	.	No	Yes	6	962	468	.	12	14	1430

Automating PROC LOGISTIC

Appendix 4 contains the macro module for PROC LOGISTIC. The code in the macro deviates from that discussed in the paper in that it will allow for the programmer to pass continuous exposures into the model as well as categorical exposures using separate parameters in the macro call. The categorical variables will appear in the CLASS statement of the PROC LOGISTIC while both the categorical and continuous variables appear in the MODEL statement. The dataset of summary statistics will appear nearly identical to the one presented in this paper. The observable difference is that the VarValue1 and VarFmt1 are missing for the continuous variables. The following macro call to ProcLogistic will generate the results seen in Figure 19.

```
%MACRO ProcLogistic(data            = mydata,
                    response        = diabetes,
                    class_vars      = smoke education,
                    continuous_vars = age bmi,
                    out             = LogisticStats,
                    rc              = rc
                    );
```

Six parameters are passed in the call to %ProcLogistic:

1) The name of the input dataset
2) The name of the response variable
3) The names of Class Exposures
4) The names of Continuous Exposures
5) The name of the output dataset (a default value of LogisticStats)
6) A variable to keep track of the return code for the hash table

The hash table return code is passed as a parameter with the default name of rc. This allows the programmer to change the name of the return code variable if it is known that this variable already exists in the input dataset and is being processed by the PROC LOGISTIC.

Figure 19: The LogisticStats Dataset Returned by %ProcLogistic

Obs	Var Label1	Var Fmt1	Var Name1	Var Value1	Odds Ratio	Upper CL	Lower CL	prob chisq
1	Smoking Status	Never Smoked	smoke	0	.	.	.	.
2	Smoking Status	Past Smoker	smoke	1	0.95	0.87	1.05	0.346
3	Smoking Status	Current Smoker	smoke	2	0.77	0.68	0.87	<.000
4	Education	< High School	education	0	.	.	.	.
5	Education	High School	education	1	1.59	1.37	1.83	<.000
6	Education	College	education	2	1.34	1.16	1.55	<.000
7	Patient age		age	.	1.01	1.00	1.02	0.004
8	Body Mass Index		bmi	.	1.03	1.02	1.04	<.001

Obs	Var Name2	Var Label2	Var Value2_1	Var Value2_2	Var Fmt2_1	Var Fmt2_2	Var N2_1	Var N2_2
1	diabetes Dx	Diabetes	0	1	No	Yes	7760	2996
2			.	.			.	.
3			.	.			.	.
4			.	.			.	.
5			.	.			.	.
6			.	.			.	.
7			.	.			.	.
8			.	.			.	.

Designing Utility Macros

Data processing using PROC CONTENTS and PROC TRANSPOSE appears within the design phase of more than one of our procedures that are summarized. Since the structure for these two procedures is identical each time they are executed, we can design a utility macro for each, in a generic sense, that can be invoked when needed. We will also design two additional utility macros, named %SPLIT and %AnalysisSet, which will be discussed further in this section. Appendix 1 contains our version of these utility macros.

Designing a Supervisory Macro

Once you have designed macro modules that return concrete useful items such as the datasets that we have discussed, you can design other macros that use them as building blocks to achieve a larger task. This type of macro is often referred to as a supervisory macro. It governs the invocation of each macro module within its boundaries (%MACRO to %MEND), passing appropriate data through macro parameters, executing the modules in the proper order and building a final product based on the results received from the tasks it delegates to each module.

Since we have discussed and provided macros for three procedures, we will design a supervisory macro that will delegate tasks to all three of these modules and combine the resulting datasets into one composite dataset with statistics from multiple procedures which can be used to generate a summary report of select statistics. Let's begin by defining the %MACRO statement for this supervisory macro:

```
%MACRO supervisor(data     =  ,
                  Varlist1 = ,
                  Varlist2 = ,
                  Out      = );
```

We will be passing in an input dataset, two lists of variables and the name of an output datasets as macro parameters:

```
%supervisor(data     = mydata   ,
            Varlist1 = smoke education #age #bmi ,
            Varlist2 = diabetes,
            Out      = SummaryStats)
```

It is beneficial to include a validation macro module which can be invoked within the supervisory macro. This validation macro will assure that all of the items that are passed into the supervisory macro actually exist and conform to the needs of the procedures that will process them. If any of the items fail validation testing, processing of the supervisory macro should cease with notification written to the SAS Log. Due to space issues, we will not include a validation macro with this paper but we recommend you review other papers for ideas on how to create this type of macro module[1].

Our task will include running a PROC LOGISTIC with a categorical response variable, identified by the parameter Varlist2, with exposure variables identified by the parameter Varlist1. The exposures can be either categorical or continuous. Categorical and continuous variables will be handled differently by the supervisory macro. Categorical variables will be processed by the ProcFreq macro module and then passed to the ProcLogistic macro module by the class_vars parameter. Continuous variables will be processed by the ProcUnivariate macro module and then passed to the ProcLogistic macro module by the continuous_vars parameter. To distinguish between the two types of variables, we attach an identifying character as a prefix to the name of the continuous variables in the VarList1 parameter of the invocation for the supervisory macro. We will use the "#" symbol as this control character. If there is no prefix, then the exposure will be processed as a categorical variable. The supervisory macro will separate these two sets of variables using the %Split macro function (Appendix 1):

```
%LOCAL class_vars continuous_vars;
%LET varlist1        = %UPCASE(&varlist1);
%LET varlist2        = %UPCASE(&varlist2);

%LET class_vars      = %split(vars= &varlist1,control=);
%LET continuous_vars = %split(vars= &varlist1,control=#);
```

We will be processing data by three procedures, PROC UNIVARIATE, PROC FREQ and PROC LOGISTIC. PROC UNIVARIATE and PROC FREQ will return statistics on all observations within the input dataset that contain recorded data. PROC LOGISTIC, however, will remove all observations that contain any missing values across the list of exposures and the response variable. This means that PROC LOGISTIC is returning results based on the lowest common denominator of the three procedures. In order to properly compare the results from all three procedures, they must each input an identical dataset based on the lowest common denominator. This will require creating an analysis dataset which can be passed to each macro module to assure that the procedures process data that can be properly combined, compared and reported together. We will create this analysis set with the %AnalysisSet macro (Appendix 1).

```
Data;Stop;Run;  %LOCAL AnalysisSet; %LET AnalysisSet = &sysLast ;
%AnalysisSet(data      = &data,
             variables = &response &class_vars &continuous_vars,
             out       = &AnalysisSet)
```

Next we will create a summary dataset of counts and percents for our categorical variables using the ProcFreq macro module:

```
Data;Stop;Run;  %LOCAL FreqStats; %let FreqStats = &sysLast ;
%Proc Freq(data     = &data ,
           Rowvars  = &class_vars,
           Colvars  = &var2list,
           out      = &FreqStats    )
```

Means and standard deviations will be obtained using the ProcUnivariate macro module with the continuous variables passed in the VarList1 parameter:

```
Data;Stop;Run;  %LOCAL UnivariateStats; %LET Univariate_Stats = &sysLast ;
%procUnivariate(data          = &data,
                varList       = &class_vars,
                class         = &varlist2,
                UnivariateOut = &UnivariateStats    )
```

Our final macro to invoke will be the ProcLogistic module:

```
Data;Stop;Run;  %LOCAL LogisticStats; %LET LogisticStats = &sysLast ;
%ProcLogistic(data             = &data,
              response         = &varlist2,
              class_vars       = &class_vars,
              continuous_vars  = &continuous_vars,
              out              = &LogisticStats,
              rc               = rc
              );
```

Sort the &FreqStats and &LogisticStats datasets prior to the final merge of these three datasets.

```
PROC SORT DATA=&FreqStats;
  BY VarName1 VarValue1;
RUN;

PROC SORT DATA=&LogisticStats;
  BY VarName1 VarValue1;
```

The &FreqStats and &LogisticStats datasets will be combined in our final dataset using a simple merge statement. Since the statistics from the &UnivariateStats dataset will be combined using a different set of merge criteria, being just the VarName1 variable, we will use a has table for processing this data in the final DATA Step.

```
DATA &out ;
  If 0 then set &UnivariateStats  ;
      Declare hash c( dataset:"&UnivariateStats" )   ;
      c.defineKey("VarName1") ;
      c.defineData(all:"yes") ;
      c.defineDone() ;
  Do until (done) ;
     MERGE &FreqStats (WHERE = (_TYPE_="11"))
           &LogisticStats
           end=done   ;
       BY VarName1 VarValue1;
     IF c.find() NE 0 THEN  call missing( of mean_: stdDev_: ) ;
     OUTPUT;
  End ;
  Stop ;
RUN;
```

This macro is self-cleaning in the sense that all macro variables used are local and all temporary datasets created by this macro are deleted before exiting the macro.

```
PROC DATASETS;
  DELETE %SUBSTR(&AnalysisSet,6)
         %SUBSTR(&FreqStats,6)
         %SUBSTR(&UnivariateStats,6)
         %SUBSTR(&LogisticStats,6);
QUIT;

%MEND supervisor;
```

The only item remaining after the %Supervisor macro compiles and executes will be a dataset with composite summary statistics from PROC UNIVARIATE, PROC FREQ and PROC LOGISTIC, named SummaryStats.

Figure 20 contains a report generated from our dataset produced by the supervisory macro.

Figure 20: Example report:

	Diagnosed with Diabetes									
	Yes (N=2996)				No N=(7760)					95% Confidence Interval
	N	%	Mean	StdDev	N	%	Mean	StdDev	p-value	Odds Ratio	Upper	Lower
Smoking Status												
Never Smoked	1556	51.9			4265	54.9			.	.	.	.
Past Smoker	991	33.1			2558	32.9			0.1058	0.923	0.837	1.017
Current Smoker	449	14.9			937	12.0			<.0001	0.776	0.683	0.881
Education												
< High School	358	11.9			645	8.3			.	.	.	.
High School	1326	44.2			858	49.7			<.0001	1.658	1.432	1.921
College	1312	43.7			257	41.9			<.0001	1.435	1.233	1.670
Patient Age			48.8	14.7			49.3	12.7	0.0036	1.005	1.002	1.009
Body Mass Index			27.1	4.2			27.6	4.2	<.0001	1.027	1.016	1.039

CONCLUSION

SAS ODS provides an efficient method for saving SAS Procedures output to SAS datasets. Although the design of these datasets is inconsistent, as we have seen, uniform rules can be adopted and applied to restructure them to more easily combine them to meet analysis and reporting needs. The macro modules we have created here will output what may be a near minimum of data that one might want to summarize from the ODS tables produced by the three procedures we have covered. However, the reader can utilize the principles presented here to retrieve additional statistics from these tables or from other ODS tables produced by the same procedures or any SAS procedure. Furthermore, the reader can design summary datasets to handle situations we have not programmed our macros to handle, such as character variables or formatted continuous numeric variables. The coding methods needed to handle these types, or even a combination of these types, are very similar. In summary, if the structure of the ODS output datasets that SAS provides does not meet your reporting and analysis needs, our rules for redesigning these datasets can serve as a blueprint for creating datasets that do (meet your needs).

DISCLAIMER: The contents of this paper are the work of the authors and do not necessarily represent the opinions, recommendations, or practices of Westat and Data and Analytic Solutions, Inc.

ACKNOWLEDGEMENTS

The authors would like to thank Mike Rhoads for his assistance and review of the methods presented in this paper.

REFERENCES

1. Long, S., Abolafia, J., Park, L. (2006). Using SAS® ODS to extract and merge statistics from multiple SAS procedures into a single summary report, a detailed methodology. *Proceedings of the 31st Annual SAS Users Group International Conference.*

CONTACT INFORMATION

Stuart Long (long3@niehs.nih.gov)
Westat
1009 Slater Road, Suite 110
Durham, NC 27703

APPENDIX 1: UTILITY MACROS

```
*************************************************************************;
* Macro ProcContents
*
* Creates a contents dataset of key variable attributes (name,
*  variable type, label, and format name) for selected variables
*  using PROC CONTENTS.
*
* Parameters:
* Data        = input data set name (data set on which PROC CONTENTS is run)
* Vars        = list of selected variables whose attributes are kept
* I           = text, typically a number, to insert in output variable names
*               Var&i.Name and Var&i.Label (optional).  If not specified
*               output variable names remain as VarName and VarLabel.
* ContentsOut = output data set name
*************************************************************************;
%MACRO ProcContents(data        = &SYSLAST,
                    vars        = ,
                    i           = ,
                    ContentsOut =  );
    ODS LISTING CLOSE ;
    ODS OUTPUT Variables=&ContentsOut(KEEP= Variable Type Format Label
                                      RENAME=(Variable = VarName&i
                                              Label    = VarLabel&i) );
    PROC CONTENTS DATA=&data(KEEP = &vars );RUN;
    ODS OUTPUT CLOSE ;
    ODS LISTING ;
%MEND ProcContents;

*************************************************************************;
* Macro ProcTranspose
*
* Transposes selected variables by running one PROC TRANSPOSE per
*  specified variable using common BY and ID statements, and then
*  merging the transposed datasets together to create the output
*  dataset.  Transposed variable names created from the specified
*  variables are a concatenation of the original variable name and the
*  value of the ID variable.
*
* Within each unique combination of BY-variable levels in the transpose:
*  - the input dataset is assumed to contain multiple records, uniquely
*     identified by the ID variable
*  - the output dataset will contain one record, with multiple variables
*     for each of the transposed variables
*
* Parameters:
* Data         = input data set name (data before transposing)
*                Default is the last data set created.
* Vars         = list of variables to transpose (used one at a time in
*                the VAR statement in PROC TRANSPOSE)
* ByVars       = list of BY-variables for the transpose (entire list
*                is used in the BY statement of each PROC TRANSPOSE)
* ID           = ID variable used in each PROC TRANSPOSE
* TransposeOut = output data set name
*************************************************************************;
%MACRO ProcTranspose(data    = &SYSLAST,
                     vars    = ,
                     BYVars  = ,
```

```
                    ID      = ,
                    TranOut = );

  %LOCAL i;
  %LET i=1;

  %DO %UNTIL(NOT %LENGTH(%SCAN(&vars,&i)));

      DATA;STOP;RUN; %LOCAL TranOut&i; %LET TranOut&i = &SYSLAST ;
      PROC TRANSPOSE DATA=&data out=&&TranOut&i (DROP=_NAME_ _LABEL_)
                     prefix=%SCAN(&VARs,&I)_;
        BY &BYVars;
        ID &id;
        VAR %SCAN(&VARs,&I);
      run;
      %LET i = %EVAL(&i+1);

  %END;

  DATA &TranOut;
    MERGE %DO j = 1 %TO %EVAL(&i-1); &&TranOut&j %END; ;
      BY &BYVars ;
  RUN;

  PROC DATASETS;
    DELETE %DO j = 1 %TO %EVAL(&i-1); %SUBSTR(&&TranOut&j,6) %END; ;
  QUIT;
%MEND ProcTranspose;


*********************************************************************;
* Macro Function Split
*
* Takes a space-delimited list of variable names and subsets it, keeping
*  only the variables that contain a specified single-character prefix
*  that is a character other than a letter or an underscore (_).  The
*  prefix is removed from all variable names in the subsetted list.
* If no prefix is specified, the subsetted list contains all variable
*  names from original list that DO NOT have such a prefix, i.e., those
*  that start with a letter or an underscore.
* The subsetted list is output in place of the macro call, so the Split
*  macro call can be assigned as follows to a macro variable:
*  %LET contvars = %Split(vars= male #age diabetes, conrol=#);
*
* Parameters:
* Vars    = input list of variables, each of which may or may not have a
*           single-character prefix that is not a letter or underscore
* Control = single-character prefix to search for in first character of
*           variable names in the list.  Default=<blank> (no prefix).
*********************************************************************;
%MACRO Split(vars    = ,
             Control = );
 %LOCAL  count varlist chars;
 %IF &vars NE %THEN %DO;
   %IF &control NE %THEN %DO;
     %LET count=1 ;
     %DO %UNTIL (%QSCAN(&vars,&count)= ) ;
       %LET var = %QSCAN(&vars,&count);
       %IF %SUBSTR(&var,1,1)=&control %THEN
           %SYSFUNC(COMPRESS(%QSCAN(&vars,&count),&control))  ;
```

```
        %LET count = %EVAL(&count+1);
      %END ;
    %END;
    %ELSE %DO;
      %LET count=1 ;
      %DO %UNTIL (%QSCAN(&vars,&count)= ) ;
       %LET var = %QSCAN(&vars,&count);
       %IF %INDEX(ABCDEFGHIJKLMNOPQRSTUVWXYZ_,%UPCASE(%SUBSTR(&var,1,1))) NE 0 %THEN
       %QSCAN(&vars,&count) ;
       %LET count = %EVAL(&count+1);
      %END ;
    %END;
 %END;
%MEND split;


*************************************************************************;
* Macro AnalysisSet
*
* Checks a list of variables.  If any observation has a missing value
*  for any one of the variables, then that observation is removed from
*  the returned analysis set
*
* Parameters:
* Data       = input data set name.
* Variables  = input list of variables to check
*              variable names in the list.  Default=<blank> (no prefix).
* Out        = output data set name for the analysis dataset
*************************************************************************;
  %MACRO AnalysisSet(data      = ,
                     variables = ,
                     out       = ) ;
     DATA &out ;
       SET &data (KEEP = &variables);
       IF NMISS(OF _numeric_) = 0 THEN OUTPUT;       RUN;
  %MEND AnalysisSet;
```

APPENDIX 2: ProcUnivariate MACRO

```
*********************************************************************;
* Macro ProcUnivariate
*
* Creates a dataset of summary statistics (N, mean, standard deviation)
*  for selected variables from a dataset using PROC UNIVARIATE.
*  Variable information is added from PROC CONTENTS using hash tables.
*
* External macros used: ProcContents, ProcTranspose
*
* Parameters:
* Data          = input data set name (unsummarized raw data)
* VarList       = list of variables for which summary statistics are
*                 calculated
* Class         = variable name for CLASS statement in PROC UNIVARIATE
*                 (required).  Only one CLASS variable may be specified
*                 with this macro.
* RC            = work variable, which can be renamed if default name
*                 conflicts (default=RC)
* UnivariateOut = output data set name
*********************************************************************;
%MACRO ProcUnivariate(
    data=
  , varList=
```

```
  , class=
  , UnivariateOut=
  , rc=rc
) ;
    %put NOTE: _local_ ;
    ODS LISTING CLOSE;
 /*-------------------------------------------------------------------------*/
  %LOCAL Var1Info Var2Info;
  DATA;STOP;RUN; %LET Var1Info = &SYSLAST ;
  DATA;STOP;RUN; %LET Var2Info = &SYSLAST ;


  %ProcContents(data        = &data,
                 vars        = &varlist,
                 i           = 1,
                 ContentsOut = &Var1Info);
  %ProcContents(data        = &data,
                 vars        = &class,
                 i           = 2,
                 ContentsOut = &Var2Info);
/*-------------------------------------------------------------------------*/
    %LOCAL moments ;
    ODS OUTPUT Moments=_data_(DROP= CValue1 CValue2
                              RENAME=(VarName=VarName1) );
    PROC UNIVARIATE DATA=&data ;
        CLASS &class;
        VAR &varlist;
        FORMAT &class;
    RUN;
    %let Moments = &sysLast ;
    ODS output close ;
 /*-------------------------------------------------------------------------*/
    Data &UnivariateOut (DROP = &class);
          LENGTH VarFmt2 $256
               FORMAT  $32
               VarLabel1 VarLabel2 $256;
     /* Let's load the class variable info from PROC CONTENTS. */
        Set &Var2Info ;
     /* Let's put the dataset with analysis variable info from PROC
        CONTENTS into a hash table. */
        If 0 then set &Var1Info ;
        Declare hash v( dataset:"&Var1Info" ) ;
        v.defineKey("VarName1") ;
        v.defineData("VarLabel1") ;
        v.defineDone() ;
        Do until (done) ;
            Set &moments end=done ;
              By VarName1 NOTSORTED &CLASS NOTSORTED;
            IF FIRST.VarName1  THEN Var2Ord=0;
            IF FIRST.&CLASS THEN Var2Ord+1;
            VarValue2=INPUT(&class,BEST8.);
         /* The SELECT block is a little more wordy, but will run more
            efficiently.  I think it's also easier to read because it has an
            explicit end; the IF statements suddenly switch from looking at
            LABEL1 to LABEL2 without as clear a delineation. */
            Select (Label1) ;
                When ("N")                    N = nValue1 ;
                When ("Mean")               Mean = nValue1 ;
                When ("Std Deviation")   StdDev = nValue1 ;
                Otherwise ;
            End ;
            Drop Label1 NValue1 Label2 NValue2 ;
                    VarFmt2=PUTN(VarValue2,Format);
            If last.&class then do ;
                &rc = v.find() ;
                Drop &rc Format;
```

```
                Output &UnivariateOut;
             End ;

          End ;
          Stop ;
      Run ;
      ODS LISTING;
           PROC PRINT DATA=&UnivariateOut;
         RUN;
         ODS LISTING CLOSE;
 /*-------------------------------------------------------------------------*/
      %ProcTranspose(data    = &UnivariateOut ,
                     vars    = VarValue2 VarFmt2 mean stddev ,
                     BYVars  = VarName1 VarLabel1 VarName2 VarLabel2,
                     ID      = var2ord,
                     TranOut = &UnivariateOut);
 /*-------------------------------------------------------------------------*/
      PROC DATASETS;
        DELETE %SUBSTR(&Var1Info,6)
               %SUBSTR(&Var2Info,6)
               %SUBSTR(&Moments,6);
      QUIT;
      ODS listing ;
 /*-------------------------------------------------------------------------*/
%mEnd procUnivariate;
/*=========================================================================*/
```

APPENDIX 3: ProcFreq MACRO

```
*************************************************************************;
* Macro ProcFreq
*
* Creates a dataset of counts and column percentages from 2-way cross-
*  tabulations produced using PROC FREQ.  All possible cross-tabulations
*  are produced by crossing each row variable from a list of row
*  variables with each column variable from a list of column variables.
*  Missing values are included in denominators when calculating column
*  percentages.
*
* External macros used: ProcTranspose
*
* Parameters:
* Data     = input data set name (unsummarized raw data)
* Var1List = list of row variables
* Var2List = list of column variables
* Out      = output data set name
*************************************************************************;
%macro ProcFreq(
    data=
  , rowvars=
  , colvars=
  , out=
) ;
    %put _local_ ;
    ODS listing close ;
 /*-------------------------------------------------------------------------*/
    %local crossTabFreqs ;
    ODS output CrossTabFreqs=_data_ ;
    PROC FREQ DATA=&data;
        TABLES (&rowvars)*(&colvars) / MISSPRINT;
```

```
   RUN;
   %let crossTabFreqs = &sysLast ;
   %put NOTE: %nrStr(&crossTabFreqs)=&crossTabFreqs ;
   ODS output close ;
/*-------------------------------------------------------------------*/
   %local reworkedCrossTabFreqs ;
   DATA _data_( KEEP=FreqKey VarName: VarValue: VarLabel: VarFmt: _type_
                     frequency colpercent ) ;
       LENGTH VarName1   VarName2    $32
              VarFmt1    VarFmt2     VarLabel1  VarLabel2  $256 ;
       ARRAY var1ary{*} &rowvars;
       ARRAY var2ary{*} &colvars;
       SET &crossTabFreqs ;
       BY Table notSorted ;
       If first.Table then FreqKey+1;
       VarName1=SCAN( TABLE , 2 ) ;
       VarName2=SCAN( TABLE , 3 ) ;
       DO i=1 TO DIM(var1ary);
          IF upcase(VarName1) = UPCASE(VNAME(var1ary{i})) THEN DO;
             VarFmt1   = VVALUE(var1ary{i});
             VarLabel1 = VLABEL(var1ary{i});
             VarValue1 = var1ary{i};
          END;
       END;
       DO i = 1 TO DIM(var2ary);
          IF upcase(VarName2) = UPCASE(VNAME(var2ary{i})) THEN DO;
             VarFmt2   = VVALUE(var2ary{i});
             VarLabel2 = VLABEL(var2ary{i});
             VarValue2 = var2ary{i};
          END;
       END;
   RUN;
   %let reworkedCrossTabFreqs = &sysLast ;
   %put NOTE: %nrStr(&reworkedCrossTabFreqs)=&reworkedCrossTabFreqs ;
/*-------------------------------------------------------------------*/
   PROC SORT data=&reworkedCrossTabFreqs ;
       BY FreqKey _TYPE_ VarValue1 VarValue2;
   RUN;
/*-------------------------------------------------------------------*/
   %local total ;  Data;Stop;Run;  %let total = &sysLast ;
   %put NOTE: %nrStr(&total)=&total ;
   DATA
       &reworkedCrossTabFreqs
       &total(
           KEEP= VarName1 VarValue1 VarLabel1 VarFmt1 VarName2 VarLabel2 _TYPE_
                 FreqKey frequency
           RENAME=( frequency=total ) );
       SET &reworkedCrossTabFreqs ;
         BY FreqKey _TYPE_ VarValue1 VarValue2;
       IF FIRST.VarValue1 THEN Var2Ord=0;
       IF FIRST.VarValue2 THEN Var2Ord+1;
       SELECT(_TYPE_);
           WHEN("01","11") OUTPUT &reworkedCrossTabFreqs ;
           WHEN("00","10") DO;
               SUBSTR(_TYPE_,2) = "1" ;
               OUTPUT &total;
           END; *DO;
       END; *SELECT;
   RUN;
/*-------------------------------------------------------------------*/
```

```
    %local tranCTF ;  Data;Stop;Run;  %let tranCTF = &sysLast ;
    %put NOTE: %nrStr(&tranCTF)=&tranCTF ;
    %procTranspose(data    = &reworkedCrossTabFreqs ,
                   Vars    = VarValue2 VarFmt2 Frequency ColPercent ,
                   Byvars  = FreqKey VarName1 VarLabel1 VarName2 VarLabel2 _type_
                             VarValue1 VarFmt1 ,
                   Id      = Var2ord ,
                   TranOut = &tranCTF
    )
 /*-------------------------------------------------------------------*/
/*    PROC SORT DATA=&tranCTF ;*/
/*        BY FreqKey _TYPE_ VarValue1;*/
/*    RUN;*/
/*    PROC SORT DATA=&total ;*/
/*        BY FreqKey _TYPE_ VarValue1;*/
/*    RUN;*/
    Data &out ;
        Merge &tranCTF &total ;
        BY FreqKey _TYPE_ VarValue1 ;
    RUN;
 /*-------------------------------------------------------------------*/
 /* A little cleanup is in order. */
    Proc sql ;
        Drop table
            &crossTabFreqs
          , &reworkedCrossTabFreqs
          , &total
          , &tranCTF
        ;
    Quit ;
  /*-----------------------------------------------------------------*/
  ODS LISTING;
%MEND ProcFreq;
/*===================================================================*/    Options
```

APPENDIX 4: ProcLogistic MACRO

```
*********************************************************************;
* Macro ProcLogistic
*
* Get model statistics from PROC LOGISTIC.  See comments below.
*
* External macros used: ProcTranspose, Split
*
* Parameters:
* Data            = input data set name
* Response        = name of the 0/1 response variable for PROC LOGISTIC model
* Class_vars      = list of Categorical exposure variables for PROC LOGISTIC model
* Continuous_vars = list of Continuous exposure variables for PROC LOGISTIC model
* Out             = output data set name
* RC              = work variable, which can be renamed if default name
*                   conflicts (default=RC)
*********************************************************************;


/************* PROC LOGISTIC ****************/
```

```
*********************************************************************;
* Macro ProcLogistic
*
* Get model statistics from PROC LOGISTIC.  See comments below.
*
* External macros used: ProcTranspose, Split
*
* Parameters:
* Data            = input data set name
* Response        = name of the 0/1 response variable for PROC LOGISTIC model
* Class_vars      = list of Categorical exposure variables for PROC LOGISTIC model
* Continuous_vars = list of Continuous exposure variables for PROC LOGISTIC model
* Out             = output data set name
* RC              = work variable, which can be renamed if default name
*                   conflicts (default=RC)
*********************************************************************;
%MACRO ProcLogistic(data           = ,
                    Response       = ,
                    class_vars     = ,
                    continuous_vars = ,
                    out            = Logistic_Stats,
                    rc             = rc
                    );

  %LET class_vars       = %UPCASE(&class_vars);
  %LET continuous_vars  = %UPCASE(&continuous_vars);

  %LOCAL class_info response_info  pe rp or or_ref TranRP;
        DATA;STOP;RUN;  %LET class_info    = &SYSLAST ;
        DATA;STOP;RUN;  %LET response_info = &SYSLAST ;
        DATA;STOP;RUN;  %LET pe            = &SYSLAST ;
        DATA;STOP;RUN;  %LET rp            = &SYSLAST ;
        DATA;STOP;RUN;  %LET or            = &SYSLAST ;
        DATA;STOP;RUN;  %LET or_ref        = &SYSLAST ;
        DATA;STOP;RUN;  %LET TranRP        = &SYSLAST ;


  %ProcContents(data  = &data,
                 vars  = &class_vars &continuous_vars,
                 i           = 1 ,
                 Contentsout = &class_info     );

  ODS LISTING CLOSE;
    ODS OUTPUT ParameterEstimates = &pe (KEEP = variable classval0 probchisq
                                                estimate StdErr
                                        RENAME = (Variable = VarName1)
                                        WHERE  = (VarName1 ^= "Intercept") )
               ResponseProfile    = &RP  (RENAME = (Count=VarN2))
               OddsRatios         = &or (KEEP = EFFECT OddsRatioEst LowerCL UpperCL
                                        RENAME = (OddsRatioEst = OddsRatio) );
  PROC LOGISTIC DATA = &data   ;
       CLASS &class_vars / ORDER=INTERNAL PARAM=REF REF=FIRST;
    MODEL &response (ORDER=INTERNAL DESCENDING) = &class_vars &continuous_vars;
    FORMAT &response &class_vars &continuous_vars;  RUN;
  ODS OUTPUT CLOSE;
  ODS LISTING;

  DATA &pe ( KEEP = VarName1 VarValue1 probchisq LogisticKey);
    LENGTH VarName1 $32;
    SET &pe ;
      BY VarName1 NOTSORTED;
    IF VarName1^=LAG(VarName1) THEN LogisticKey+1;
    VarValue1 = input(classval0,BEST8.);
  RUN;
```

```
DATA &or     (KEEP = VarName1 VarValue1 LogisticKey OddsRatio LowerCL UpperCL)
     &or_ref (KEEP = VarName1 VarValue1 LogisticKey);
  LENGTH VarName1 $32 VarValue1 8;
  SET &or ;
     VarName1  = SCAN(EFFECT,1);
  VarValue1 = INPUT(SCAN(EFFECT,2),BEST8.);
  IF VarName1^=LAG(VarName1) THEN DO;
        LogisticKey+1;
        OUTPUT &or;
        Reference=SCAN(EFFECT,4);
        IF Reference ^= "" THEN DO;
        VarValue1 = INPUT(Reference,BEST8.);
           OUTPUT &or_ref;
        END;
  END;
     ELSE OUTPUT &or;
RUN;

PROC SORT DATA=&pe;     BY VarName1 VarValue1;RUN;
PROC SORT DATA=&or;     BY VarName1 VarValue1;RUN;
PROC SORT DATA=&or_ref; BY VarName1 VarValue1;RUN;

DATA &out (DROP = format type);
      LENGTH VarLabel1 VarFmt1 $256 ;
      If 0 then set &class_info ;
      Declare hash c( dataset:"&class_info" ) ;
      c.defineKey("VarName1") ;
      c.defineData(all:"yes") ;
      c.defineDone() ;
      Do until (done) ;
          MERGE
              &pe
              &or
              &or_ref
              end=done
          ;
          BY VarName1 VarValue1;
          &rc = c.find() ;  Drop &rc ;
          VarFmt1 = putN( VarValue1 , Format ) ;
          OUTPUT;
      End ;
      Stop ;
RUN;

PROC SORT DATA=&out;
  BY LogisticKey;
RUN;

PROC SORT DATA=&rp;
  BY outcome;
RUN;

DATA &rp (DROP=outcome);
     LENGTH VarName2 $32 VarValue2 8.0;
  SET  &rp ;
     var2ord=_N_;
     VarValue2=INPUT(outcome,8.0);
     VarName2="&response";
RUN;

%ProcContents(data  = &data,
              vars  = &response,
              i     = 2 ,
              Contentsout   = &response_info     );

DATA &rp (DROP = format );
```

```
    LENGTH VarLabel2 VarFmt2 $256;
    MERGE &rp &Response_Info;
         BY VarName2;
       VarFmt2=PUTN(VarValue2,Format);
  RUN;

  %ProcTranspose(data    = &rp ,
                 vars    = VarValue2 VarFmt2 VarN2 ,
                 BYVars  = varname2 varlabel2 ,
                 ID      = var2ord,
                 TranOut = &TranRP);

  DATA &Out;
    MERGE &Out &TranRP;
  RUN;

  PROC DATASETS;
    DELETE %SUBSTR(&class_info,6)
           %SUBSTR(&response_info,6)
           %SUBSTR(&pe,6)
           %SUBSTR(&rp,6)
           %SUBSTR(&or,6)
           %SUBSTR(&or_ref,6)
           %SUBSTR(&TranRP,6);
  QUIT;

%MEND ProcLogistic;
/*========================================================================*/
```