‘V’ for … Variable Information Functions to the Rescue

Richann Watson, Experis, Batavia, OH
Karl Miller, inVentiv Health, Lincoln, NE

ABSTRACT

There are times when we need to use the attributes of a variable within a data set. Normally, this can be done with a simple CONTENTS procedure. The information can be viewed prior to programming and then hardcoded within the program or it can be saved to a data set that can be joined back to the main data set. If the attributes are hardcoded then what happens if the data set changes structure, then the program would need to be updated accordingly. If the information from PROC CONTENTS is saved and then joined with the main data set, then this would need to be done for all data sets that need to be processed. This is where knowing your ‘V’ functions can come in handy. The ‘V’ functions can be used to return the label, format, length, name, type and/or value of a variable or a string within the data step. These functions can come quite in handy when you need to create summary statistics and if you need to perform an algorithm on a variable with a specific naming convention.

INTRODUCTION

SAS® has an assortment of functions that can help make programming more efficient. Some of the functions are the variable information functions. These functions allow the programmer to use the information about the variable within the data step without having to hard code the information or without having to do a secondary step to incorporate data from PROC CONTENTS. This will allow your program to be data and/or variable dependent.

‘V’ FUNCTIONS

For purpose of displaying examples in the table of ‘V’functions (table 2), the following variables, format, and arrays have been assigned the following attributes:

VARIABLE	TYPE	LENGTH	FORMAT	LABEL	VALUE
ID	Num	8	Z3.		001
Visit	Text	10			Baseline
TESTCD	Text	40	$TEST.	Test Short Code	WEIGHT
RESULT	Num	8	8.2	Standardized Result	109
ORGRSLT	Num	8	BEST12.	Original Result	240

Table 1. Example Variables and Attributes

Format $TEST. has a value that is 10 characters, so this would be considered the width of $TEST. format.

Defined arrays:

```
array allcrc(*) _character_;     /* all character variables */
array allnum(*) _numeric_;       /* all numeric variables */
```

The following table is a list of various ‘V’functions and examples of their results based on the variable attributes in Table 1.

‘V’ Functions **	Description	Example
VARRAY(*name*)	Whether the name is the name of an array	**VARRAY**(allcrc) = 1 **VARRAY**(allnum) = 1 **VARRAY**(allvars) = 0
VFORMAT(*name*)	Variable’s format attribute along with the width of the longest value in the format	**VFORMAT**(ID) = Z3. **VFORMAT**(VISIT) = $10. **VFORMAT**(TESTCD) = $TEST10. **VFORMAT**(RESULT) = 8.2 **VFORMAT**(ORGRSLT) = BEST12.
VFORMATD(*name*)	Decimal argument in format	**VFORMATD**(ID) = 0 **VFORMATD**(VISIT) = 0 **VFORMATD**(TESTCD) = 0 **VFORMATD**(RESULT) = 2

'V' Functions **	Description	Example
		VFORMATD(ORGRSLT) = 0
VFORMATN(*name*)	Format name associated with variable if applicable	**VFORMATN**(ID) = 0 **VFORMATN**(VISIT) = 0 **VFORMATN**(TESTCD) = $TEST **VFORMATN**(RESULT) = F **VFORMATN**(ORGRSLT) = BEST
VFORMATW(*name*)	Width argument in the format	**VFORMATW**(ID) = 3 **VFORMATW**(VISIT) = 10 **VFORMATW**(TESTCD) = 10 **VFORMATW**(RESULT) = 8 **VFORMATW**(ORGRSLT) = 12
VINARRAY(*name*)	Whether the variable is an element in an array	**VINARRAY**(ID) = 1 **VINARRAY**(VISIT) = 1 **VINARRAY**(TIME) = 0
VLABEL (*name*)	Variable label or if not label then it returns the variable name	**VLABEL**(ID) = ID **VLABEL**(VISIT) = Visit **VLABEL**(TESTCD) = Test Short Code **VLABEL**(RESULT) = Standardized Result **VLABEL**(ORGRSLT) = Original Result
VLENGTH (*name*)	Variable length	**VLENGTH**(ID) = 8 **VLENGTH**(VISIT) = 10 **VLENGTH**(TESTCD) = 40 **VLENGTH**(RESULT) = 8 **VLENGTH**(ORGRSLT) = 8
VNAME (*name*)	Variable name	**VNAME**(ID) = ID **VNAME**(VISIT) = Visit **VNAME**(TESTCD) = TESTCD **VNAME**(RESULT) = RESULT **VNAME**(ORGRSLT) = ORGRSLT
VTYPE (*name*)	Variable type	**VTYPE**(ID) = N **VTYPE**(VISIT) = C **VTYPE**(TESTCD) = C **VTYPE**(RESULT) = N **VTYPE**(ORGRSLT) = N
VVALUE (*name*)	Variable formatted value	**VVALUE**(ID) = 001 **VVALUE**(VISIT) = Baseline **VVALUE**(TESTCD) = WEIGHT **VVALUE**(RESULT) = 109 **VVALUE**(ORGRSLT) = 240

Table 2. 'V' Functions and Examples

** There is a corresponding VyyyX function that uses a 'string' instead of 'name' as the argument. These functions operate in a similar manner with the difference being that the VyyyX functions allows for an expression in the argument. For example, **VVALUEX**(vname(allcrc(i))). This function is using the expression **VNAME**(allcrc(i) as the input argument.

APPLICATION SCENARIOS

Although there are many benefits of implementation of the use of 'V' functions, here are three hypothetical scenarios:

1. Missing numeric data needs to be set to different missing codes depending on data type.
2. Summary statistics for variables when there is a varying decimal precision for each value is one of the desired outputs. For example, a report of summary statistics for baseline characteristics such as weight, height, BMI and disease duration. The baseline characteristics have different decimal precision and the correct decimal precision for each summary statistics needs to be incorporated.
3. The variables in all the data sets need to have the correct variable attributes. The problem variables in affected data sets needed to be renamed so that a new variable with original variable name can be created with the correct attributes.

SCENARIO 1 – SET MISSING NUMERIC VALUES TO A MISSING CODE BASED ON FORMAT OF THE DATA

After reviewing the data, it is decided that having a lot of missing fields in the data set is not the desired output. It is preferred that missing codes be applied to the missing numeric data based on the type of data. The missing data should be represented by the following:

NUMERIC DATATYPE	MISSING CODE
Float	.D
DATETIME	.T
DATE	.M
TIME	.H
All other numeric results	.Z

Table 3. Example of Missing Codes for Numeric Results

Again these missing codes could be manually applied if you knew the name of all the data sets and variables affected but this can be cumbersome and easy to apply the wrong missing code especially in cases when the data may appear as an integer but has an actual decimal format applied (i.e. float). Using VFORMATD and VFORMAT to determine the format of each numeric variable allows the missing codes to be applied appropriately regardless of the name of the variables.

```
data subj;
  set INDSN.SUBJ;
  array nv(*) _NUMERIC_;

  do i = 1 to dim(nv);
    if nv(i) = . then do;
      if vformatd(nv(i)) > 0 then nv(i) = .D;
      else if index(vformat(nv(i)), 'DATETIME') then nv(i) = .T;
      else if index(vformat(nv(i)), 'DATE') then nv(i) = .M;
      else if index(vformat(nv(i)), 'TIME') then nv(i) = .H;
      else nv(i) = .Z;
    end;
  end;

run;
```

Using the following values as an illustration:

i = 1
nv(1) = TRTN

a. **vformatd(nv(1)) → vformatd(TRTN) → 0**
b. **vformat(nv(1)) → vformat(TRTN) → 'BEST12.'**

STUDYID	USUBJID	TRT	TRTN	FEXPDATE	LEXPDATE	TRTDUR	TRTDURU	INFCNDT	RANDOMDT	DEATHDT	DEATHTM	DTHDDTM	DISDUR
ABC	ABC-101-01000		.	.	.	85		28-Jun-13	22-Aug-13	.	.	.	3.03
ABC	ABC-101-01002	DRUG3	3	15-Jul-13	7-Oct-13	85	DAY	2-Jul-13	15-Jul-13	.	.	.	.
ABC	ABC-101-01003	DRUG6	6	16-Jul-13	8-Oct-13	85	DAY	2-Jul-13	16-Jul-13	.	.	.	5.82
ABC	ABC-101-01004	DRUG1	1	19-Jul-13	9-Oct-13	83	DAY	5-Jul-13	19-Jul-13	.	.	.	6.11
ABC	ABC-101-01032	DRUG3	3	23-Aug-13	15-Nov-13	85	DAY	9-Aug-13	23-Aug-13	.	.	.	9.79
ABC	ABC-101-01037	DRUG4	4	24-Sep-13	19-Nov-13	57	DAY	10-Sep-13	24-Sep-13	.	.	.	2.88
ABC	ABC-101-01038	DRUG1	1	28-Oct-13	23-Dec-13	57	DAY	10-Sep-13	28-Oct-13	.	.	.	4.44
ABC	ABC-101-01040	DRUG3	3	27-Sep-13	20-Dec-13	85	DAY	13-Sep-13	27-Sep-13	.	.	.	3.91
ABC	ABC-102-01050	DRUG3	3	17-Oct-13	12-Dec-13	57	DAY	19-Sep-13	17-Oct-13	.	.	.	.
ABC	ABC-102-01051	DRUG6	6	11-Dec-13	11-Dec-13	1	DAY	8-Oct-13	11-Dec-13	5-Feb-14	4:30	05FEB2014:04:30:00	5.95
ABC	ABC-103-01100	DRUG1	1	30-Sep-13	25-Nov-13	57	DAY	23-Aug-13	30-Sep-13	.	.	.	7.5

Display 1. Sample Data Before Numeric Missing Codes Applied

STUDYID	USUBJID	TRT	TRTN	FEXPDATE	LEXPDATE	TRTDUR	TRTDURU	INFCNDT	RANDOMDT	DEATHDT	DEATHTM	DTHDDTM	DISDUR
ABC	ABC-101-01000		Z	M	M	85		28-Jun-13	22-Aug-13	M	H	T	3.03
ABC	ABC-101-01002	DRUG3	3	15-Jul-13	7-Oct-13	85	DAY	2-Jul-13	15-Jul-13	M	H	T	D
ABC	ABC-101-01003	DRUG6	6	16-Jul-13	8-Oct-13	85	DAY	2-Jul-13	16-Jul-13	M	H	T	5.82
ABC	ABC-101-01004	DRUG1	1	19-Jul-13	9-Oct-13	83	DAY	5-Jul-13	19-Jul-13	M	H	T	6.11
ABC	ABC-101-01032	DRUG3	3	23-Aug-13	15-Nov-13	85	DAY	9-Aug-13	23-Aug-13	M	H	T	9.79
ABC	ABC-101-01037	DRUG4	4	24-Sep-13	19-Nov-13	57	DAY	10-Sep-13	24-Sep-13	M	H	T	2.88
ABC	ABC-101-01038	DRUG1	1	28-Oct-13	23-Dec-13	57	DAY	10-Sep-13	28-Oct-13	M	H	T	4.44
ABC	ABC-101-01040	DRUG3	3	27-Sep-13	20-Dec-13	85	DAY	13-Sep-13	27-Sep-13	M	H	T	3.91
ABC	ABC-102-01050	DRUG3	3	17-Oct-13	12-Dec-13	57	DAY	19-Sep-13	17-Oct-13	M	H	T	D
ABC	ABC-102-01051	DRUG6	6	11-Dec-13	11-Dec-13	1	DAY	8-Oct-13	11-Dec-13	5-Feb-14	4:30	05FEB2014:04:30:00	5.95
ABC	ABC-103-01100	DRUG1	1	30-Sep-13	25-Nov-13	57	DAY	23-Aug-13	30-Sep-13	M	H	T	7.5

Display 2. Sample Data After Numeric Missing Codes Applied

SCENARIO 2 - CREATING THE SUMMARY STATISTICS WHEN THERE ARE VARYING DECIMAL PRECISIONS

PREPARING SUMMARY STATISTICS FOR FUTURE PROCESSING WITH 'V' FUCNTIONS

When creating the summary statistics we normally use a SUMMARY or MEANS procedure to produce the outputs. If the VAR statement lists all the variables to be summarized then the programmer usually specifies the new variable names for each summary statistic. However that can be a bit cumbersome especially if you have numerous variables that need to be summarized. Another approach would be to create a macro and send each variable to be summarized as a macro parameter.

```
/* OPTION 1 */
/* obtain summary stats for each variable */
proc means data=INDSN.ADSL noprint nway;
  format _numeric_ best12.;
  var HEIGHT WEIGHT BMI DISDUR;
  output out=allstats n=HEIGHT_N WEIGHT_N BMI_N
                      mean=HEIGHT_MEAN WEIGHT_MEAN BMI_MEAN DISDUR_MEAN
                      std=HEIGHT_STDDEV WEIGHT_STDDEV BMI_STDDEV DISDUR_STDDEV
                      median=HEIGHT_MEDIAN WEIGHT_MEDIAN BMI_MEDIAN DISDUR_MEDIAN
                      min=HEIGHT_MIN WEIGHT_MIN BMI_MIN DISDUR_MIN
                      max=HEIGHT_MAX WEIGHT_MAX BMI_MAX DISDUR_MAX;
run;
```

This would give us data that looks like the following display, all the summary statistics for each parameter and visit on the same record. With only three variables to be summarized in the analysis, we end up with 18 variables. So the number of variables per record can become pretty cumbersome rather quickly and having to manually enter all those names into the output statement can be tedious work. If you plan on implementing a macro later to process that is based on the name of these variables then it can be easy to mistype a variable (e.g. MEDIAN could be spelled MEDAIN on one variable).

HEIGHT_N	WEIGHT_N	BMI_N	DISDUR_N	...	HEIGHT_Max	WEIGHT_Max	BMI_Max	DISDUR_Max
548	548	548	548		196.2	170	58.7	12

Display 3. PROC MEANS Option 1

```
/* OPTION 2 */
/* obtain summary stats for each variable */
%macro stats(var);
  proc means data=INDSN.ADSL noprint nway;
    format _numeric_ best12.;
    var &var.;
    output out=stats&var. n=&var._N
                          mean=&var._MEAN
                          std=&var._STDDEV
                          median=&var._MEDIAN
                          min=&var._MIN
```

```
                          max=&var._MAX;
  run;
%mend stats;
%stats(HEIGHT);   %stats(WEIGHT);  %stats(BMI);    %stats(DISDUR);
```

This would give us data that is similar to the above display, but instead of having one data set with all the variables we would have separate data sets with only the necessary variables. For example, STATSHEIGHT would only contain the HEIGHT summary statistic variables. There would also be a separate data set for STATSWEIGHT, STATSBMI and STATSDISDUR. So for processing, to get the summary statistics in the correct format, each data set would either need to be processed separately or need to be combined to look like the data from Option 1.

HEIGHT_N	HEIGHT_Mean	HEIGHT_StdDev	HEIGHT_Median	HEIGHT_Min	HEIGHT_Max
548	168.6895985	10.11266792	169	145	196.2

Display 4. PROC MEANS Option 2

Option 1 or Option 2 is fine if there are few variables that need summarizing. However if there are numerous variables then a new approach may be more ideal. This new approach uses the current functionality of PROC MEANS to automatically name the variables. The approach along with the use of the 'V' functions can allow for a more dynamic program; no more need to manually name all the summary statistic variables for each variable. No more need to create a macro to loop through each of the variables and have separate data sets. The ***autoname*** option on the ***output*** statement will automatically name the summary statistic variables based on the variable name and the summary statistic.

```
/* obtain summary stats for each variable */
proc means data=INDSN.ADSL noprint nway;
  format _numeric_ best12.;
  var HEIGHT WEIGHT BMI DISDUR;
  output out=allstats n=
                      mean=
                      std=
                      median=
                      min=
                      max=   / autoname;
run;
```

This would produce the same results as option 1 would but without having to manually enter all the variable names and without any potential chance of a typo.

FORMATTING THE SUMMARY STATISTICS FOR REPORTING

Now that all the pre-work for preparing the data has been completed, the summary statistics from PROC MEANS data set can be used for the final processing with the 'V' functions. Each of the summary statistics needs to be formatted based on the decimal precision for the specific characteristic and the data needs to be displayed in a specific fashion such as:

Characteristic	Statistics	Baseline
Baseline Height (cm)	N	xx
	Mean	xx.xx
	Median	xx.xx
	Standard Deviation	xxx.xxx
	Min, Max	xx.x, xx.x
Baseline Weight (kg)	N	xx
	Mean	xx.xx
	Median	xx.xx
	Standard Deviation	xxx.xxx
	Min, Max	xx.x, xx.x
Baseline BMI (kg/m2)	N	xx
	Mean	xx.xx
	Median	xx.xx
	Standard Deviation	xxx.xxx
	Min, Max	xx.x, xx.x
Disease Duration (years)	N	xx
	Mean	xx.x
	Median	xx.x
	Standard Deviation	xxx.xx
	Min, Max	xx, xx

Display 5. Sample of Desired Output

The normal way to create this table would require a series of repeated data step statements formatting and outputting each variable individually as indicated in the sample code below. This can be monotonous and easy to overlook adding/updating a variable for a particular section.

```
   /* format the data for output the normal tedious way by manually */
   /* typing all the information for each variable                  */
   data allstats2 (keep=PARAM: section_lbl sort label rsltc);
     set allstats (drop=_:);
     length section_lbl $20. label $50. rsltc $30.;
     section_lbl = "HEIGHT";
     label = 'n';
     if HEIGHT_N ne . then rsltc = put(&var._N, 5.);
     else rsltc = '';
     sort = 1;
     output;
     label = 'Mean';
     if HEIGHT_MEAN ne . then rsltc = put(&var._MEAN, 7.2);
     else rsltc = '';
     sort = 2;
     output;
       …
     /* repeat for each variable in the allstats data set */
run;
```

Using the ‘V’ functions can help to eliminate this mind-numbing work as well as facilitate the updates if more numeric baseline characteristics are added, or if the decimal precision or variable label changes. Below is an illustration of how the ‘V’ functions can create the desired SAS data set and still be generic enough to add more variables or make updates to the data or variable labels. This paper will walk through this scenario step by step. Note that the primary

purpose of the paper is to illustrate the use of 'V' functions therefore some of the SAS code is omitted. The complete code can be found in Appendix 1.

Step 1 Bring in the Summary Stats and Formatting Information

The first step of the process is to bring the summary statistics that are generated from PROC MEANS using the 'autoname' option. In addition, the formats for each summary statistics for each variable needs to be merged with the summary statistics. For this particular scenario there is only one record for **allstats** and one record for **maxdec** and since we want to maintain the one record, the two data sets are merged but the by statement is left off so that a Cartesian join can occur. In addition, to retrieve the summary statistics and formats, arrays need to be initialized. The initialization of the arrays is an important step for future use in the data step.

```
data allstats2 (keep=section_lbl sort label rsltc);
  merge allstats (drop=_:) maxdec;
  length section_lbl label $50. rsltc $30.;
  array lbl (5) $30. _TEMPORARY_ ('n' 'Mean' 'Standard Deviation'
                                  'Median' 'Min, Max');
  array val (6) $7. ('_N' '_Mean' '_StdDev' '_Median' '_Min' '_Max');
  array fmt (6) $7. ('5.' 'onefmt' 'twofmt' 'onefmt' 'samefmt' 'samefmt');
  array sct (*) &varlist.;
```

Step 2 Determine the 'Section' Labels

The desired output requires that the variable labels be used to describe each section. To dynamically create the section labels without having to hard code for each section, the following 'V' functions are used: vlabelx, vname, vvalue.

```
section_lbl = vlabelx(strip(cat(vname(sct(section)), vvalue(val(1)))));
```

Note that in some cases the use of VyyyX is necessary. The X version of the 'V' function works the same way as the regular 'V' function with the difference being the input is a string.

Using the following values as an illustration:

&varlist = HEIGHT WEIGHT BMI DISDUR
section = 1

a. **vvalue(val(1)) → '_N'**
   Since SAS creates temporary variables for arrays we wanted the value of the temporary variable returned, so vvalue was used.
b. **vname(sct(section)) → vname(sct(1)) → vname(HEIGHT) → 'HEIGHT'**
   The first element in the sct array is HEIGHT. If we would have indicated sct(section), then the return value would have been the actual value of HEIGHT and not the variable name. Therefore, we use vname to return the actual name of the variable.
c. **vlabelx(strip(cat(vname(sct(section)), vvalue(val(1))))) → vlabelx(strip(cat('HEIGHT', '_N'))) → vlabelx(strip('HEIGHT_N')) → vlabelx('HEIGHT_N') → 'Baseline Height (cm)'**
   Using the values created in steps a. and b., a new string is created. The new string represents a variable in the allstats data set. Because the input is a string, the X version of the function is used so that it knows to evaluate the input as a string and return the desired output.

Step 3 Create Variables to Store Results and Formats

Using the arrays defined in Step 1, temporary variables can be created to store the results of the current statistics for the current baseline characteristics.

Using the following values as an illustration:

&varlist = HEIGHT WEIGHT BMI DISDUR
section = 1
sort = 5

```
varname = strip(cat(vname(sct(section)), vvalue(val(sort))));
varval = input(vvaluex(varname), best.);
```

a. **vvalue(val(sort)) → vvalue(val(5)) → '_Min'**

b. **vname(sct(section)) → vname(sct(1)) → vname(HEIGHT) → 'HEIGHT'**
c. **vvaluex(varname) → vvaluex('HEIGHT_Min') → 145**

```
varfmt = strip(cat(vname(sct(section)), vvalue(fmt(sort))));
```

a. **vvalue(fmt(sort)) → vvalue(fmt5) → 'samefmt'**
b. **vname(sct(section)) → vname(sct(1)) → vname(HEIGHT) → 'HEIGHT'**

The combination of step a. and b. yield varfmt = 'HEIGHTsamefmt' which the value of varfmt represents an actual variable name that is found in maxdec. This variable contains the actual format that will be used to produce the result in the desired output.

HEIGHTmaxdec	HEIGHTonefmt	HEIGHTtwofmt	HEIGHTsamefmt
1	8.2	8.3	8.1

Display 6. Portion of maxdec data set

Step 4 Create a Record for Each Summary Statistic for Each Characteristic

Now that the result and format are stored in temporary variables, the output can be generated. For illustration purposes, we will only focus on the 'else if sort = 5', since this portion encompasses similar logic as the other if-then-else statements.

Using the following values as an illustration:

&varlist = HEIGHT WEIGHT BMI DISDUR
section = 1
sort = 5

```
...
    else if sort = 5 then do;
      varname2 = compress(cat(vname(sct(section)), vvalue(val(sort+1))));
      varval2 = input(vvaluex(varname2), best.);
      varfmt2 = strip(cat(vname(sct(section)), vvalue(fmt(sort+1))));
      if varval ne . or varval2 ne . then
         rsltc = catx(", ", putn(varval, vvaluex(varfmt)),
                            putn(varval2, vvaluex(varfmt2)));
    end;
    else rsltc = '';
    output;
```

a. **vvalue(val(sort+1)) → vvalue(val(6)) → '_Max'**
b. **vname(sct(section)) → vname(sct(1)) → vname(HEIGHT) → 'HEIGHT'**
c. **vvaluex(varname) → vvaluex('HEIGHT_Max') → 196.2**
d. **vvalue(fmt(sort)) → vvalue(fmt6) → 'samefmt'**
e. **vname(sct(section)) → vname(sct(1)) → vname(HEIGHT) → 'HEIGHT'**
f. **vvaluex(varfmt) → vvaluex('HEIGHTsamefmt') → '8.1'**
g. **vvaluex(varfmt2) → vvaluex('HEIGHTsamefmt') → '8.1'**
h. **rsltc = catx(", ", putn(varval, vvaluex(varfmt)), putn(varval2, vvaluex(varfmt2))) → catx(", ", putn(145, 'HEIGHTsamefmt', putn(196.2, 'HEIGHTsamefmt')) → catx(", ", "145.0", "196.2") → '145.0, 196.2'**

Using a combination of the 'V' functions to retrieve the necessary labels, values and formats, allows us to get the data in the desired output without having to do a lot of hardcoding.

section_lbl	label	rsltc	sort
Baseline Height (cm)	n	548	1
Baseline Height (cm)	Mean	168.69	2
Baseline Height (cm)	Standard Deviation	10.113	3
Baseline Height (cm)	Median	169.00	4
Baseline Height (cm)	Min, Max	145.0, 196.2	5
Baseline Weight (kg)	n	548	1
Baseline Weight (kg)	Mean	81.40	2
Baseline Weight (kg)	Standard Deviation	18.511	3
Baseline Weight (kg)	Median	79.55	4
Baseline Weight (kg)	Min, Max	40.5, 170.0	5
Baseline BMI (kg/m2)	n	548	1
Baseline BMI (kg/m2)	Mean	28.47	2
Baseline BMI (kg/m2)	Standard Deviation	5.406	3
Baseline BMI (kg/m2)	Median	27.80	4
Baseline BMI (kg/m2)	Min, Max	15.2, 58.7	5
Disease Duration (years)	n	548	1
Disease Duration (years)	Mean	6.1	2
Disease Duration (years)	Standard Deviation	3.02	3
Disease Duration (years)	Median	6	4
Disease Duration (years)	Min, Max	0, 12	5

Display 7. Final Output Data set

Note that if Min and Max are to be on separate lines the code would be modified. For this scenario Min and Max were on the same line to illustrate that this process can still be used to combine the individual summary statistics if it is desired.

SCENARIO 3 – RENAMING VARIABLES BY APPENDING A COMMON PREFIX OR A SUFFIX TO VARIABLES IN A DATA SET

For a specific study, the majority of the data sets used the incorrect variable attributes. The correct attributes are stored as a macro that is read in at the final step before producing the final output. However, when the attributes get updated it produces a WARNING message similar to the one below.

```
WARNING: Multiple lengths were specified for the variable XXXXXX by input data
set(s). This may
         cause truncation of data.
```

We could manually rename all the variables that are causing the warnings and then create a new variable with the original variable name with the correct attributes. Of course this is assuming you know which data sets and variables are causing the issue and if you know all data sets and variables affected. Manually changing this information is tedious and prone to human error. An alternative is to utilize the 'V' functions.

Step 1 Retrieve Names of Data Sets

To rename the variables for each data set in the specified library, the data set names need to be retrieved and stored in a macro variable so that the process of renaming can loop through each data set individually.

```
/* determine what data sets are in the library */
proc sql noprint;
  select distinct memname into :listdsn separated by " "
  from dictionary.columns
  where libname = 'INDSN';
quit;
```

Step 2 Retrieve the Variable Names within Each Data Set

The first part of the looping process is to create macro variables that concatenate all the character variables and all the numeric variables with each variable, prefixed or suffixed, with the desired value to make the new variable. These macro variables will be used to define arrays in the next step so that each variable can be renamed. Note for

illustration of this example only code pertaining to character variables is shown. The complete code can be found in Appendix 2.

```
/* create a macro variable with a '_' added as a prefix to each variable       */
/* note that the character variables and numeric variables need to be separate */
proc sql noprint;
  /* create a macro variable that contains all the character variables */
  select cats("_", name) into :_&dsn.cv separated by " "
  from dictionary.columns
  where libname = 'INDSN' and memname = "&dsn." and xtype = 'char';
quit;
```

Step 3 Rename Original Variables to New Variables

Within a data step, all the formats are removed from the current data so that formats from the correct variable attributes can be set in the next step and arrays are created based on the macro variables defined above. New variables can be created by processing each element in both the character variable array and the numeric variable array. Before proceeding, it is ideal to save the original data sets in a different location so that a comparison can be made to ensure that the values did not change.

```
/* initialize the arrays that contain the character and numeric variables */
array _cvrs (*) $200. &&_&dsn.cv;

/* variables that will capture the original var names concatenated together  */
/* this will be used to create a macro var that will be used in the next step*/
length cnames $1000.;
do i = 1 to dim(_cvrs);
  _cvrs(i) = vvaluex(substr(vname(_cvrs(i)), 2));
  cnames = catx(" ", cnames, substr(vname(_cvrs(i)), 2));
end;
```

Using the following values as an illustration:

&dsn = ADSL
i = 2
_cvrs(1) = _country
country = USA
_cvrs(2) = _actarm
actarm = DRUG3

a. **vname(_cvrs(2)) → vname(_actarm) → '_actarm'**
b. **vvaluex(substr(vname(_cvrs(2)), 2)) → vvaluex(substr('_actarm', 2) → vvaluex('actarm') → 'DRUG3'**
c. **cnames = catx(" ", cnames, substr(vname(_cvrs(2)), 2)) → cnames = catx(" ", cnames, 'actarm')) → cnames = 'country actarm'**
   Note that since i=2, the do loop is on the second iteration which means that cnames would already have the value of the first variable, 'country'. So the second iteration is just concatenating the variable to the existing value. At the end of the data step, cnames will be made into a macro variable to use for array initialization in the next step.

Step 4 Renaming the New Variables Back to Original Variable Names with Correct Attributes

Now that all the variables have been renamed, the original variables can be recreated with the correct attributes. This can be done by initializing arrays and then just setting the original variables equal to the new variables.

```
/* redefine all the variables back to original name with the correct attributes */
data OUTDSN.&dsn.;
  %&dsn.
  set &dsn.2;
  array _cvrs (*) $200. &&_&dsn.cv;
  array cnames(*) &cnames;
  do i = 1 to dim(_cvrs);
    cnames(i) = _cvrs(i);
  end;
  drop _: i;
run;
```

The original data can be compared to the new data using a simple COMPARE procedure. As illustrated below the data before the renaming of the variables and after the renaming of variables, show that the values are exactly equal and that the attributes have been updated.

The COMPARE Procedure
Comparison of INDSN.ADSL with OUTDSN.ADSL
(Method=EXACT)
Data Set Summary

Dataset	Created	Modified	Nvar	Nobs
INDSN.ADSL	28FEB14:14:21:07	28FEB14:14:27:56	36	553
OUTDSN.ADSL	28FEB14:14:28:58	28FEB14:14:28:58	36	553

Variables Summary

Number of Variables in Common: 36.
Number of Variables with Differing Attributes: 11.

Listing of Common Variables with Differing Attributes

Varible	Dataset	Type	Length	Format	Label
actarm	INDSN.ADSL	Char	200		Description of Actual Arm
	OUTDSN.ADSL	Char	100		Description of Actual Arm
arm	INDSN.ADSL	Char	200		Description of Planned Arm
	OUTDSN.ADSL	Char	100		Description of Planned Arm
STUDYID	INDSN.ADSL	Char	12		Study ID
	OUTDSN.ADSL	Char	12		Study Identifier
USUBJID	INDSN.ADSL	Char	15		Unique Subject ID
	OUTDSN.ADSL	Char	22		Unique Subject Identifier
TRTSDT	INDSN.ADSL	Num	8	DDMMYY10.	Date of First Exposure to Treatment
	OUTDSN.ADSL	Num	8	DATE9.	Date of First Exposure to Treatment
TRTEDT	INDSN.ADSL	Num	8	DDMMYY10.	Date of Last Exposure to Treatment
	OUTDSN.ADSL	Num	8	DATE9.	Date of Last Exposure to Treatment
TR01SDT	INDSN.ADSL	Num	8	DDMMYY10.	Date of First Exposure in Period 01
	OUTDSN.ADSL	Num	8	DATE9.	Date of First Exposure in Period 01
TR01EDT	INDSN.ADSL	Num	8	DDMMYY10.	Date of Last Exposure in Period 01
	OUTDSN.ADSL	Num	8	DATE9.	Date of Last Exposure in Period 01
INCNDT	INDSN.ADSL	Num	8	DDMMYY10.	Date of Informed Consent
	OUTDSN.ADSL	Num	8	DATE9.	Date of Informed Consent
RANDDT	INDSN.ADSL	Num	8	DDMMYY10.	Date of Randomization
	OUTDSN.ADSL	Num	8	DATE9.	Date of Randomization
DTHDT	INDSN.ADSL	Num	8	DDMMYY10.	Death Date
	OUTDSN.ADSL	Num	8	DATE9.	Death Date

Observation Summary

Observation	Base	Compare
First Obs	1	1
Last Obs	553	553

Number of Observations in Common: 553.
Total Number of Observations Read from INDSN.ADSL: 553.
Total Number of Observations Read from OUTDSN.ADSL: 553.

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 553.

NOTE: No unequal values were found. All values compared are exactly equal.

Display 8. Comparison of Data Before Variables are Renamed with Data After Variables are Renamed

CONCLUSION

Utilizing 'V' functions can make a tedious job easier while ensuring your code to be more proficient and efficient. In all opportunities where a programmer can eliminate manual updates in coding (aka. Hardcoding) for variables, formats, etc. it is recommended to do so. Not only for ease of programming but for reducing, if not potentially eliminating, errors while adding the benefit of consistency for any future use of the code.

REFERENCES

Aster, R. 2004. *Profession SAS Programmer's Pocket Reference*.5th ed. Breakfast Communications Corporation. 181p.

Cody, R. 2004. *SAS® Functions by Example*. SAS Institute Inc. 359-377p.

CONTACT INFORMATION

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

Richann Watson
Experis
richann.watson@experis.com

Karl Miller
inVentiv Health
karl.miller@inventivhealth.com

APPENDIX 1 COMPLETE CODE FOR SCENARIO 2

```
/* specify the list of variables that are to be used for summary statistics */
/* it is assumed that these variables exist in one data set                 */
%let varlist = HEIGHT WEIGHT BMI DISDUR;

/* obtain summary stats for each variable */
proc means data=INDSN.ADSL noprint nway;
  format _numeric_ best12.;
  var &varlist.;
  output out=allstats n=
                      mean=
                      std=
                      median=
                      min=
                      max=    / autoname;
run;

/* determine number of decimals for each parameter */
data temp;
  set INDSN.ADSL;

  %macro numdec;
    %let x = 1;
    %let var = %scan(&varlist., &x);

    %do %while (&var. ne );
      temp = strip(put(&var., best.));
      if &var. ne . and int(input(temp, best.)) = &var. then &var.dec = 0;
      else if &var ne .  and indexc(temp, '.') then
         &var.dec  = length(compress(scan(temp, 2, '.')));

      %let x = %eval(&x. + 1);
      %let var = %scan(&varlist., &x);
    %end;

  %mend numdec;
  %numdec
run;

/* determine maximum number of decimals for each parameter */
/* then create formats for 0, 1 and 2 decimals past maximum*/
proc sql;
  create table maxdec (drop=dummy) as
  select
        %macro maxdec(var=);
          %let x = 1;
          %let var = %scan(&varlist., &x);
          %do %while (&var. ne );
            max(&var.dec) as &var.maxdec,
            cat('8.', put(calculated &var.maxdec + 1, 1.)) as &var.onefmt length = 5,
            cat('8.', put(calculated &var.maxdec + 2, 1.)) as &var.twofmt length = 5,
            cat('8.', put(calculated &var.maxdec, 1.)) as &var.samefmt length = 5,

           %let x = %eval(&x. + 1);
           %let var = %scan(&varlist., &x);
         %end;
       %mend maxdec;
       %maxdec

        1 as dummy  /* this is only a place holder and can be dropped on the final */
                    /* data output it is needed because list of selected variables */
                    /* generated in the macro will be expecting another variable   */
```

```
                    /* to be added                                                    */
  from temp;
quit;

/* format the data for display purpose */
/* there is no by statement since each data set has */
/* only one record and we want to keep one record   */
data allstats2 (keep=section_lbl sort label rsltc);
  merge allstats (drop=_:) maxdec;
  length section_lbl label $50. rsltc $30.;

  array lbl (5) $30. _TEMPORARY_ ('n' 'Mean' 'Standard Deviation'
                                  'Median' 'Min, Max');
  array val (6) $7 ('_N' '_Mean' '_StdDev' '_Median' '_Min' '_Max');

  /* Note that 'N' is always a whole number so the first format will be a   */
  /* set value while the other formats will be determined based on the data */
  array fmt (6) $7. ('5.' 'onefmt' 'twofmt' 'onefmt' 'samefmt' 'samefmt');
  array sct (*) &varlist.;

  do section = 1 to dim(sct);
    /* determine the section label (i.e. Baseline Height (cm)) based on the */
    /* label of the 'string' that represents the variable in the data set   */
    section_lbl = vlabelx(strip(cat(vname(sct(section)), vvalue(val(1)))));

    do sort = 1 to dim(lbl);
      label = lbl(sort);

      /* retrieve the stored value */
      varname = strip(cat(vname(sct(section)), vvalue(val(sort))));
      varval = input(vvaluex(varname), best.);

      /* determine formats */
      varfmt = strip(cat(vname(sct(section)), vvalue(fmt(sort))));

      /* generate the result variable in the desired format */
      if sort = 1 then rsltc = putn(varval, fmt(sort));
      else if 1 < sort < 5 and varval ne . then rsltc = putn(varval, vvaluex(varfmt));
      else if sort = 5 then do;
        varname2 = compress(cat(vname(sct(section)), vvalue(val(sort+1))));
        varval2 = input(vvaluex(varname2), best.);
        varfmt2 = strip(cat(vname(sct(section)), vvalue(fmt(sort+1))));
        if varval ne . or varval2 ne . then
           rsltc = catx(", ", putn(varval, vvaluex(varfmt)),
                              putn(varval2, vvaluex(varfmt2)));
      end;
      else rsltc = '';
      output;

    end;
  end;
run;
```

APPENDIX 2 COMPLETE CODE FOR SCENARIO 3

```
libname indsn "C:\Users\us68157\Desktop\MWSUG\2014\Drafts\Scenario 3\Incorrect";
libname outdsn "C:\Users\us68157\Desktop\MWSUG\2014\Drafts\Scenario 3\Correct";
libname specs "C:\Users\us68157\Desktop\MWSUG\2014\Drafts\Scenario 3\StudySpecs.xlsx"
mixed=yes header=yes;

data attribs (keep= DATASET attribute);
  set specs."Correct Attribs$"n;
  where DATASET ne '';
```

```
  length length $5. attribute $200. dformat $30.;

  if DATATYPE = 'text' then length = cat("$", SASLENGTH, ".");
  else length = cat(SASLENGTH, ".");

  if DISPLAY_FORMAT not in ('') then dformat = cat("format = ", DISPLAY_FORMAT);
  attribute = compbl(cat("attrib ", VARIABLE, " label = '", strip(LABEL), "'", "
                    length = ", length, " ", dformat, ";"));

run;
libname specs CLEAR;

/* build macros that will contain the variable attributes for each data set */
data _null_;
  set attribs;
  by DATASET;

  file "C:\Users\us68157\Desktop\MWSUG\2014\Drafts\Scenario 3\attrib.sas";
  if first.DATASET then put @1 '%macro ' DATASET ';';
  put @3 attribute;
  if last.DATASET then put @1 '%mend ' DATASET ';';
run;
%inc "C:\Users\us68157\Desktop\MWSUG\2014\Drafts\Scenario 3\attrib.sas";

/* determine what data sets are in the library */
proc sql noprint;
  select distinct memname into :listdsn separated by " "
  from dictionary.columns
  where libname = 'INDSN';
quit;

%macro rename;
  %let x = 1;
  %let dsn = %scan(&listdsn., &x.);

  %do %while (&dsn. ne );
    /* create a macro variable with a '_' added as a prefix to each variable       */
    proc sql noprint;
      /* create a macro variable that contains all the character variables */
      select cats("_", name) into :_&dsn.cv separated by " "
      from dictionary.columns
      where libname = 'INDSN' and memname = "&dsn." and xtype = 'char';

      /* create a macro variable that contains all the numeric variables */
      select cats("_", name) into :_&dsn.nv separated by " "
      from dictionary.columns
      where libname = 'INDSN' and memname = "&dsn." and xtype = 'num';
    quit;

    /* reassign all the var to the new variable name that has the '_' prefixed */
    /* so that the original var names can be reused with correct attributes    */
    data &dsn.2 (keep = _:) ;
      set INDSN.&dsn. end=last;

      /* remove all formats want to use formats associated with correct attributes*/
      format _all_;

      /* initialize the arrays that contain the character and numeric variables */
      array _cvrs (*) $200. &&_&dsn.cv;
      array _nvrs (*) 8.    &&_&dsn.nv;
      /* variables that will capture the original var names concatenated together  */
      /* this will be used to create a macro var that will be used in the next step*/
      length cnames nnames $1000.;
```

```
      do i = 1 to dim(_cvrs);
        _cvrs(i) = vvaluex(substr(vname(_cvrs(i)), 2));
        cnames = catx(" ", cnames, substr(vname(_cvrs(i)), 2));
      end;

      do i = 1 to dim(_nvrs);
        _nvrs(i) = vvaluex(substr(vname(_nvrs(i)), 2));
        nnames = catx(" ", nnames, substr(vname(_nvrs(i)), 2));
      end;

      /* create macro variables that contain all the character vars & numeric vars */
      if last then do;
        call symputx("cnames", cnames);
        call symputx("nnames", nnames);
      end;
    run;

    /* redefine all the variables back to original name with the correct attributes */
    data OUTDSN.&dsn. (drop = _: i);
      %&dsn.
      set &dsn.2;

      array _cvrs (*) $200. &&_&dsn.cv;
      array cnames(*) &cnames;

      array _nvrs (*) 8.    &&_&dsn.nv;
      array nnames(*) &nnames;

      do i = 1 to dim(_cvrs);
        cnames(i) = _cvrs(i);
      end;

      do i = 1 to dim(_nvrs);
        nnames(i) = _nvrs(i);
      end;
    run;

    %let x = %eval(&x. + 1);
    %let dsn = %scan(&listdsn., &x.);
  %end;
%mend rename;
%rename

proc compare base=INDSN.ADSL compare=OUTDSN.ADSL listall;
run;
```