

# A “bit” about Proc Summary: Technique for Working with Summary Output

Gregory T. Weber, Troy A. Ruth, Steve Light  
DataCeutics, Inc., Pottstown, PA

## ABSTRACT

This paper will present a technique used to create and subsequently select a subset of the summary records from the output data set created by Proc Summary. The technique, called bit-testing, is especially useful for identifying the specific summary records when multiple By and/or Class variables are used. The bit-testing method allowed us to easily identify and select the needed summary records. Unique to our application was the method used to dynamically generate the bit “masks” which are unique to the requested grouping and report options. The generalized nature of the method allows use of one set of standard code for use with any summary incorporating any number of By and/or Class variables. The paper will introduce the bit-testing method and share the technique used to generate the bit strings.

## INTRODUCTION

This paper will present a technique used to create and subsequently select a subset of the summary records from the output data set created by proc summary. The selection technique, called bit testing, is especially useful for identifying the summary records when there are By and/or Class variables. In our application, a macro to report counts and percentages, the grouping variables would be the page, row and column variables. The application also needed to allow for percentage calculations based upon different denominators and display the appropriate subject counts in the column headings. Options to include total columns and/or rows would also be supported by the macro. To create the report, the data would need to be summarized at many levels. For this reason, the decision was made to use proc summary to create an output data set containing all counts that could possibly be required and use bit testing to select the needed summary records based upon the requested report options.

Since the class variables and denominator options were to be passed ias macro parameters and therefore were not known ahead of time, a method was developed to dynamically generate the bit sequences needed to select the required summary records. After discussing the summarized data and a general discussion of using bit testing in proc summary, the discussion will concentrate on the creation of these bit sequences.

## SUMMARIZING THE DATA

This section provides an introduction to the data, how it was summarized and an example of a report that the macro would be required to produce.

## REPORT REQUIRED

For this presentation, we will tabulate the change of distribution of Low, Normal and High range flag values in three treatment groups, between a baseline and end point observation. Table 1 shows a mockup of the table required. This is a basic layout of the report types our macro needed to support.

Table 1 – Report Template

Laboratory Test Baseline - Study End	Placebo (N=xx)				100 mg (N=xx)				200 mg (N=xx)				Total (N=xx)			
	L	N	H	T	L	N	H	T	L	N	H	T	L	N	H	T
Parameter 1																
Low																
Normal																
High																
<i>Total</i>																

Note that there are a number of summarization types required for this table:

- N=xx in column headers represents the number of subjects in each treatment group
- Detail cells cross-tabulate the lab categories
- Total cells collapse L, N, H categories at baseline and/or end point
- Total Column cells (right-most treatment classification) collapse treatment groups *and* flag categories

## DATA

The input data contains the baseline flag value *baseflag* and endpoint flag value *postflag*, with one record per lab parameter *lbtest* per patient *usubjid*. A *Countflag*, count=1, is generated for each record in the data. Additional *Countflag* records are inserted with count=0, for combinations of parameters not already present in the data.

**Table 2 – Input data**

usubjid	lbtest	trtcd	baseflag	postflag	count
04-3069-01-0117	Hematocrit	Placebo	N	N	1
04-3069-01-0118	Hematocrit	200 mg	L	N	1
04-3069-01-0119	Hematocrit	100 mg	H	N	1
04-3069-01-0120	Hematocrit	200 mg	N	N	1
04-3069-01-0120	Hematocrit	200 mg	N	L	1
04-3069-01-0121	Hematocrit	200 mg	N	N	1
04-3069-01-0122	Hematocrit	200 mg	N	H	1
04-3069-01-0123	Hematocrit	Placebo	N	N	1
04-3069-01-0124	Hematocrit	100 mg	L	L	1

## PROC SUMMARY

As noted above, several types of summarization were required for our report. The input data does not contain total categories for treatment group or flag category, but proc summary will compute everything we need in one pass. Our task was simply to extract all the values required. The challenge for us was to find a robust method that allowed us to implement one set of code that would handle varying reporting conditions.

The following proc summary step was used to summarize lab data by computing the sum of *count*. In this example, the class variables include the lab test (&row), treatment code (&column) and the baseline and endpoint flags.

```
proc summary data=inputlabs chartype;
  class &pageby &row &column baseflag postflag;
  var count; /* Created in a previous data step */
  output out=sumdata sum=count;
run;
```

## OUTPUT DATA SET : SUMDATA

Table 3 presents the data generated by the above proc summary step (not all observations and variables are shown). Proc summary generates the variable *\_TYPE\_* which we used to select the records needed for presentation. We found it easiest to select the summary records by bit-testing the binary string version of *\_TYPE\_* values. The CHARTYPE option is used to request that proc summary represent the *\_TYPE\_* variables as a sequence of 1's and 0's. It is the binary (base 2) equivalent of the default *\_TYPE\_* variable always found in proc summary output. For example the default *\_TYPE\_* of 8 is equal to the binary string "1000". The default *\_TYPE\_* column is shown in the Table 3 for illustrative purposes. To keep the example manageable, we have included only the minimum amount of class variables and only the output for "Hematocrit" is displayed.

**Table 3 - Proc summary output records**

Obs	LBTEST	TRTCD	BASEFLAG	POSTFLAG	COUNT	_TYPE_	_TYPE_ (default)
65	Hematocrit				32	1000	8

66	Hematocrit			L	1	1001	9
67	Hematocrit			N	28	1001	9
68	Hematocrit			H	3	1001	9
69	Hematocrit		L		2	1010	10
70	Hematocrit		N		26	1010	10
71	Hematocrit		H		4	1010	10
72	Hematocrit		L	L	1	1011	11
73	Hematocrit		L	N	1	1011	11
74	Hematocrit		L	H	0	1011	11
75	Hematocrit		N	L	0	1011	11
76	Hematocrit		N	N	24	1011	11
77	Hematocrit		N	H	2	1011	11
78	Hematocrit		H	L	0	1011	11
79	Hematocrit		H	N	3	1011	11
80	Hematocrit		H	H	1	1011	11
81	Hematocrit	Placebo			8	1100	12
82	Hematocrit	100mg			7	1100	12
83	Hematocrit	200mg			17	1100	12
84	Hematocrit	Placebo		L	0	1101	13
85	Hematocrit	Placebo		N	7	1101	13
86	Hematocrit	Placebo		H	1	1101	13
87	Hematocrit	Placebo	L		0	1110	14
88	Hematocrit	Placebo	N		7	1110	14
89	Hematocrit	Placebo	H		1	1110	14
...							
102	Hematocrit	Placebo	L	L	0	1111	15
103	Hematocrit	Placebo	L	N	0	1111	15
104	Hematocrit	Placebo	L	H	0	1111	15
105	Hematocrit	Placebo	N	L	0	1111	15
106	Hematocrit	Placebo	N	N	7	1111	15
107	Hematocrit	Placebo	N	H	0	1111	15
108	Hematocrit	Placebo	H	L	0	1111	15
109	Hematocrit	Placebo	H	N	0	1111	15
110	Hematocrit	Placebo	H	H	1	1111	15

**FINAL REPORT**

The final report, sometimes called a lab shift table, is shown in Table 4. The colors match the proc summary output above to indicate what records were used for each section of the report. For this report both the “inner” totals (denoted by a “T”) and the “Total” column are optional. The patient counts (N=*n*) in the column headers are also optional. Also available are percentages, which could be calculated using the overall total or the individual row totals. All of the counts required to support these options are available in the proc summary output. A method to efficiently retrieve the needed records was required and we decided to use bit testing.

**Table 4 – Example Report**

Laboratory Test Baseline - Study End	Placebo (N=8)				100 mg (N=7)				200 mg (N=17)				Total (N=32)			
	L	N	H	T	L	N	H	T	L	N	H	T	L	N	H	T
L	0	0	0	0	1	0	0	1	0	1	0	1	1	1	0	2
N	0	7	0	7	0	4	0	4	0	13	2	15	0	24	2	26
H	0	0	1	1	0	2	0	2	0	1	0	1	0	3	1	4
T	0	7	1	8	1	6	0	7	0	15	2	17	1	28	3	32

## BIT TESTING

This section gives a quick introduction to bit testing and then describes how it was used in our application.

### BIT-TESTING FACILITY

The data step below selects the observations 81-83 (from Table 3 above) which represent the counts by lbtest and trtcd. This operation could also have been done on the proc summary out= option.

```
data select;
  set sumdata;
  if _type_ = '1100';
run;
```

If we had not, for some reason, used the CHARTYPE option we would have needed to use the bit-testing literal to test the default \_type\_ variable. The bit-testing literal works similar to the more widely used date literal (e.g '29NOV2005'd). These literals allow testing of special data types. In this case, adding a "b" after the second single quote converts the \_type\_ variable to binary when it is compared against the specified bit sequence.

```
data select;
  set sumdata;
  if _type_ = '1100'b;
run;
```

It does not matter if you use the CHARTYPE option or use the bit testing literal to generate the binary bit sequence. What is important is understanding how to use these bit sequences.

So how do you know what bit sequence to use? All you need to know is the order that the class variables were requested from proc summary (In this case LBTEST, TRTCD, BASEFLAG and POSTFLAG) and then "switch on" the bits for the summary observations that you want. Table 5 shows how the bit sequences relate to the summary records in Table 3 and the tabular results in Table 4.

**Table 5 – Relating the bit sequence to the summary records**

LBTEST	TRTCD	BASEFLAG	POSTFLAG	Description
1	0	0	0	LBTEST Overall total (highest level)
1	0	0	1	LBTEST * POSTFLAG totals
1	0	1	0	LBTEST * BASEFLAG totals
1	0	1	1	LBTEST * BASEFLAG * POSTFLAG totals
1	1	0	0	LBTEST * TRTCD totals
1	1	0	1	LBTEST * TRTCD * POSTFLAG totals
1	1	1	0	LBTEST * TRTCD * BASEFLAG totals
1	1	1	1	Detail (lowest level) totals

### DYNAMICALLY CREATING THE BIT SEQUENCES

In the above example there were no pageby variables and only one row variable so it was an easy matter to construct the bit sequence since we know that it will only ever have four "bits". When using the macro, the number of grouping variables will not be known until the input parameters are processed, so we needed a method to count and then construct the bit sequence.

Some rules to take into account are:

1. The user is required to supply one, and only one column, baseflag and postflag variable.
2. There must be at least one row variable.
3. Pageby variables are optional.

Since only pageby and row parameters could have more than one variable, this meant the final three bits would always be present and represent the column, baseflag and postflag variables respectively. This part of the bit sequence would be "fixed", however, the portion of the bit mask representing the pageby and row variables would be variable.

If there were two pageby variables and two row variables we need to construct a longer bit sequence (seven bits in this case).

Variable portion				Fixed portion		
PAGEBY1	PAGEBY2	ROW1	ROW2	COLUMN	BASEFLAG	POSTFLAG
0	0	0	0	0	0	0

We determined that we needed the following bit sequences, some of which would be stored as macro variables for use throughout the macro.

**pbits** – represents the number of pageby vars.

**rbits** – represents the number of row vars.

**vbits** – this will contain the entire variable part of the bit mask (&pbits concatenated with &rbits)

**nbits** – represents the summary records (needed for  $N=n$  in the column headers)

The following code uses the repeat function to construct the bit sequences. The repeat function repeats the specified string  $n + 1$  times so it is necessary to subtract 1. Using this method allowed us to dynamically create the bit sequences depending on the number of pageby and row variables.

Note: In this code, the macro variables &pvars and &rowvars contain the number of variables specified on the pageby and row parameters respectively.

```
data _null_;
  vbits = repeat('1',%eval(&pvars + %eval(&rowvars - 1)));
  call symput('vbits',trim(vbits));
  %if &pageby ^= %then %do;  /* If no pageby vars dont do this;
    pbits = repeat('1',%eval(&pvars - 1));
  %end;
  %else %do;
    pbits = "";
  %end;

  rbits = repeat('0',%eval(&rowvars - 1));
  nbits = compress(pbits || rbits);
  call symput('nbits',trim(nbits));
run;
```

## USING THE BIT SEQUENCES

Once the bit sequences were created, it was only a matter of using them to select the required summary records based upon the requested report options. Using the bit sequences, the data is output to individual data sets to be used in subsequent calculations. For example, the totden data set is used for calculating percentages based upon the total patient count, while the rowden data set is used if percentages are based upon the row totals. The nlabel data set will be used to calculate the ( $N=n$ ) in the column headers.

```
data totden rowden nlabel;
  set x_in;
  if _type_ = "&vbits.100" or _type_ = "&vbits.000" then output totden;
  if _type_ = "&vbits.110" or _type_ = "&vbits.100"
    or _type_ = "&vbits.000" or _type_ = "&vbits.010" then output rowden;
  if _type_ = "&nbits.100" or _type_ = "&nbits.000" then output nlabel;
```

```
run;
```

Using this same method, all other necessary data sets (e.g. the detail records) were created.

## ALTERNATIVE SOLUTIONS

As always in SAS<sup>®</sup>, there are alternative solutions such as using the `_TYPE_` variable directly or using macro code to dynamically generate multiple calls to `proc summary`. The `Proc Summary TYPES` and `WAYS` statements can be used to more tightly control creation of the needed class variable combinations. We present one of many possible solutions. We favor this method because it allows us to handle a wide range of `proc summary` configurations and to easily extract the required results with relatively simple code.

## CONCLUSION

In this paper we have given an overview of the `Proc Summary` bit string `_TYPE_` variable and a technique used to dynamically generate the bit sequences needed to select the required records to generate a somewhat complex report. The bit sequences needed to be created dynamically as the user may request multiple `pageby` and `row` variables. There are alternatives to using the bit string variable such as using the numeric `_TYPE_` variable directly or using macro code to generate multiple calls to `proc summary`. Our approach allowed us to keep the code quite simple (only one `proc summary`) and using the bit string `_TYPE_` made the task of identifying the required summary records quite intuitive.

## REFERENCES

SAS OnlineDoc<sup>®</sup>, Version 8 February 2000, SAS Institute Inc., Cary, NC

## CONTACT INFORMATION

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

Gregory T. Weber - [weberg@dataceutics.com](mailto:weberg@dataceutics.com)

Troy A. Ruth - [ruth@dataceutics.com](mailto:ruth@dataceutics.com)

Steve Light - [lights@dataceutics.com](mailto:lights@dataceutics.com)

DataCeutics, Inc. (<http://www.dataceutics.com>) is a SAS Alliance Partner and a CDISC member.



SAS and all other SAS Institute, Inc. product or service names are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.