

# SAS NOTES

Kent Lewandowski

Latest update on 7/13/2006

## Table of Contents

AND / OR Operators in macros.....	2
Arrays .....	2
AUDIT .....	2
Cards and Datalines (using CARDS to create datasets) .....	3
Changing Column Properties in DATA step .....	3
Combining Datasets (without MERGE).....	4
Combining Datasets (using MERGE) .....	4
Comments in Macros .....	4
Comparing Strings and Numbers (using the PUT function); .....	4
Converting Hex to Decimal and Back to Hex .....	5
Converting Strings and Numbers.....	5
Copying Datasets .....	5
DATALINES function (submit data or read raw data files).....	6
Dealing with Dates (PROC SQL, Intl Formats, Counting days, DATETIME and DATEPART functions) .....	6
Debugging Tips .....	8
Dropping Datasets .....	9
Dropping Formats and Labels .....	9
FILENAME and LIBNAME .....	9
Filter Data using DATA step.....	9
Finding duplicates (NODUPKEY , using FIRST and LAST).....	10
FIRST. and LAST. Variables .....	11
Functions inside Macros .....	11
GENNUM and OBS .....	12
GOTO and a util macro “Numobs” .....	12
IF and ELSE .....	13
Indexes.....	14
INTNX Function (measuring ranges between dates) .....	14
KEEP in DATA step.....	15
MOD() function .....	15
Modifying and Updating Datasets .....	15
Number Formats .....	16
ODS HTML with PROC GCHART .....	16
OUTPUT command (and EOF).....	16
Proc APPEND .....	17
PROC COMPARE.....	17
PROC CONTENTS .....	17
PROC EXPORT .....	17
PROC FORMAT .....	18

PROC FREQ.....	22
PROC MEANS.....	22
PROC REPORT.....	23
PROC SQL to create a table.....	23
PROC SQL SELECT INTO.....	24
PROC SQL SELECT WITH DESCENDING ORDER.....	24
PROC TABULATE.....	24
PROC TRANSPOSE.....	25
Proc UNIVARIATE.....	26
Quotes vs. Doublequotes.....	27
RETAIN and If-then-else.....	27
SCAN and LENGTH functions.....	27
SQL String creating method.....	28
String Parsing Functions (where to find them).....	28
SUBSTR function (ex. Streamed data).....	28
Time / Date Functions (INTCK and INTNX).....	29
TRANSLATE function.....	29
Unique or Duplicate Values Check.....	29

## AND / OR Operators in macros

```

%macro test;
    %let var1 = "hello";
    %let var2 = "world";
    %if (&var1 eq "hello") %then %put &var1;
    %if (&var2 eq "world") %then %put &var2;
    %if (&var1 eq "hello") && (&var2 eq "world") %then %put &var1 ...
&var2 !;
    %if (&var1 eq "x") | (&var2 eq "x") %then %put &var1 is x or &var2
is x!;
%mend test;
%test;

```

## Arrays

Use the syntax `array ColArray{*} &collist;`

## AUDIT

```

* turn on auditing on dataset LAB_NUM (see other "Format" examples);
* this will add 2 columns WHO and WHY to the dataset;

```

```

proc datasets lib = work nolist;
    audit lab_num;
    initiate;
    user_var who $20 label = 'Who made the change'
              why $20 label = 'Why the change was made';

```

```
run;
```

```

438 * turn on auditing;
439 proc datasets lib = work nolist;
440     audit lab_num;
441     initiate;

```

WARNING: The audited data file WORK.LAB\_NUM.DATA is not password protected. Apply an ALTER

```

        password to prevent accidental deletion or replacement of it and any associated
audit
    files.
442     user_var who $20 label = 'Who made the change'
443         why $20 label = 'Why the change was made';
444 run;

```

NOTE: The data set WORK.LAB\_NUM.AUDIT has 0 observations and 11 variables.

```

* PROC SQL INSERT: need to include last 2 fields b/c of AUDIT;
proc sql;
    insert into lab_num
    values ('Systemlab_0000', '2.45', '0', 'Kent', 'New Systemlab');
run;

* turn off auditing;
proc datasets library = work nolist;
    audit lab_num;
    terminate;
quit;

```

### ***Cards and Datalines (using CARDS to create datasets)***

CARDS is a very useful way to create test data. However, the DELIMITER option does not appear to be available with CARDS.

DATALINES is more flexible and allows you to use the DELIMITER option to separate your input fields with, say, a comma.

```

* in the following example, I fill each field with data using CARDS;
data test;
    input minor_fac $3. major_fac $3. charge_fac $3.;
    cards;
ABHOAK
LVRREG
LVMWCR
;
run;

```

```

* in the following example, I can use the DELIMITER argument with
DATALINES to separate my fields. Note, because I did not specify
length of fields, each will use default 8 bytes;
data person;
    infile datalines delimiter=',';
    input minor_fac $ major_fac $ charge_fac $;
    datalines;
ABH, OAK,
LVR, REG,
LVM, WCR,
run;

```

### ***Changing Column Properties in DATA step***

Using LENGTH as your first argument, you can convert the length of columns to your specification (in the example below, \$32 and \$128).

```

data fmctype;
  length id shortname $32. en_US es_ES de_DE it_IT hu_HU $128.;
  retain id shortname en_US es_ES de_DE it_IT hu_HU;
  set referenc.fmctype;
  keep id shortname en_US es_ES de_DE it_IT hu_HU;
run;

```

### **Combining Datasets (without MERGE)**

\* ref Programming Teil 2, 486.

```

DATA merged;
  SET data1 data2;
  BY id;
RUN;

```

### **Combining Datasets (using MERGE)**

```

DATA merged;
  MERGE data1 data2;
  BY id;
RUN;

```

- merge will combine the columns, so if data1 has columns a and b, and data2 has column c, “merged” will have cols a, b and c!

### **Comments in Macros**

- before I wrote `*%do cy=&startyear %to &thisyear;`
- the correct form to comment this is `*do cy=&startyear %to &thisyear;`
- I got this error:  
ERROR: There were 1 unclosed %DO statements.

### **Comparing Strings and Numbers (using the PUT function);**

It’s necessary to recognize that doing a string compare in a macro is different than in a data step. Most times I get messed up because I forget I should leave out the “ “ quotes surrounding a string in a macro.

```

* this section will be replaced with variables input via Intranet;
%let parmBGDT = '01Jan05'd;
%let parmEDDT = '31Mar05'd;
%let parmFAC = 'SRO';

%macro makeWhereStr;
  %if (&ISSUE_FAC eq ALL) %then %do;  <-I forgot to leave out quotes.
    data macros;
      format wherestr $128.;
      wherestr = "xsvc_from ge &BEG_DT and xsvc_from le &END_DT
and paid_v_contc = 'N' and ext_pay_type eq 'R'";
      call symput('wherestr', trim(wherestr));
    run;
  %end;
%else %do;
  data macros;
    format wherestr $128.;

```

```

        wherestr = "issue_fac eq '&ISSUE_FAC' and xsvc_from ge
&BEG_DT and xsvc_from le &END_DT and paid_v_contc = 'N' and
ext_pay_type eq 'R' " ;
        call symput('wherestr', trim(wherestr));
    run;
%end;
%mend makeWhereStr;
%makeWhereStr;
%put whereStr: ->&whereStr<-;

```

## Converting Hex to Decimal and Back to Hex

I wanted to create some dummy "objectid" keys (hex32) for testing. No easy way to do this. I used the following:

```

data work.patieneventresult_dummyinsert (drop = z zib4 zibhex32);
    set work.patieneventresult_repl1 (obs = 10);
    z = round(abs(RANNOR(today())*100000)); *create a random number
between 1 and 100000;
    zib4 = put(z,ib4.); * must convert to binary integer first;
    zibhex32 = put(zib4,$hex32.); * now convert to hex32;
    datel = datel + 60;
    objectid = zibhex32;
    *change every other record;
    IF MOD(_N_, 2) = 0 then
        patient = compress(patient||"HIST");
    else
        patient = compress(patient||"COMP");
run;

```

## Converting Strings and Numbers

- to convert a numeric to a string, use the following syntax:

```

data test;
    teststr = put(errtype,1.);
    format teststr $ERRTEXT.;
    put teststr=;
run;

```

- to convert a string to numeric, use INUPT(StrVar,1.) (or similar);  
- ref Little SAS book, p.205

- to check for numeric

```

DATA test;
    set labresults;
    if input (parameterlowerlimi, ?? 8.) = . then
        parameterlowerlimi1 = .;
    else
        parameterlowerlimi1 = parameterlowerlimi + 0;
run;

```

## Copying Datasets

ex.1:

```
DATA MYLIB.SAVESET;  
    SET WORK.SAVESET;  
RUN;
```

- w/ use of RENAME:

```
DATA work.copy(rename=(key=mykey));  
    SET work.temp;  
RUN;
```

- copying entire libraries;

```
PROC COPY IN = work OUT = newlib INDEX =YES; /* INDEX option keeps  
existing indexes */  
RUN;
```

- copy a single dataset from one lib to another (ex. to work)

```
proc copy in=ia out=work;  
    select employee_data;  
run;
```

### ***DATALINES function (submit data or read raw data files)***

- You can use a delimiter so you don't have to properly space your entries;

```
data work.delim;  
    infile datalines delimiter=',';  
    length key $16. num 8.;  
    input key $ num;  
    datalines;  
key1, 12.4  
key1, 1020  
key2, 13  
key3, 15  
run;
```

\* without a delimiter you need to space it;

```
data work.nodelim;  
    infile datalines;  
    input key $1-16 num;  
    datalines;  
key1      12.4  
key1      1020  
key2      13  
key3      15  
run;
```

### ***Dealing with Dates (PROC SQL, Intl Formats, Counting days, DATETIME and DATEPART functions)***

\* inserting dates using PROC SQL - you need to use the "date format identifier" ('DDMONYYYY'd);

```
proc sql;  
    insert into work.temp  
    values ('B100270118219786761', 'SystemLab_1714', '18OCT2001'd,  
'TRUE');
```

```
quit;
```

\* **Julian Dates** – converting dates to Julian and back.

```
* build a date table, e.g. rec 1 / 01Jan04 / 31Jan04
                        rec 2 / 01Feb04 / 28Feb04;
```

```
data days;
  format today startdate enddate date9.;
  start04 = juldate('01Jan04'd);
  do day = start04 to (start04+365);
    today = datejul(day);
    * x=intnx('month','05jan95'd,0);
    startdate = intnx('month',today,0);
    enddate = intnx('month',today,0,'end');
    output;
  end;
run;
```

\* *Intl formats*

- you have to replace "eur" in eurdfde9 with "esp"!

```
data test;
  today=today();
  name = put (today, DOWNNAME2.);
  cat_name = put (today, CATDFDWN2.);
  esp_name = put (today, ESPDFDWN2.);
  deu_name = put (today, DEUDFDWN2.);
  put today= name= cat_name= esp_name= deu_name=;
run;
today=16317 name=Fr cat_name=Di esp_name=vi deu_name=Fr
```

- if you don't want to use the \$COLNAME. date formats, you can also use OPTIONS.

```
options DFLANG='Spanish';
data _null_ ;
  adate=put (today(), EURDFDWN.);
  put adate;
run;
```

```
1  options DFLANG='Spanish';
2  data _null_ ;
3    adate=put (today(), EURDFDWN.);
4    put adate;
5  run;
```

**viernes**

\* *counting dates, using DATEPART:*

- in the following example, "patienteventencoun" has a DATETIME column, DATE1, which we want to store as a "regular" date in the column DATEPT. Also, we only want records where the date is less than 90 days old.

```
data work.temp;
  set db2sas.patienteventencoun (obs=5);
  DatePt=datepart(datel) - 90;
  put Datel= DatePt=;
run;
DATE1=05APR2004:02:43:00.000000 DatePt=16076
```

```
DATE1=05APR2004:02:43:00.000000 DatePt=16076
DATE1=05APR2004:02:44:00.000000 DatePt=16076
DATE1=05APR2004:02:43:00.000000 DatePt=16076
DATE1=05APR2004:02:43:00.000000 DatePt=16076
```

*\* using datetime variables in macros:*

```
data _null_;
  call symput('current_time',put(datetime(),timeampm.));
run;
%put current time: &current_time;
current time: 5:16:44 PM
```

*\* various Date functions:*

```
data _NULL_;
  hoy = date();
  dia = weekday(hoy);
  hoyday = put(hoy, yymmdd10.);
  put hoy= dia= hoyday=;
  put hoy weekday9.;
  put hoy weekday3.;
run;
```

```
hoy=16317 dia=6 hoyday=2004-09-03
Friday
Fri
```

*\* counting days using the **datediff** function:*

the following example counts the difference in number of days (normal calendar) between the beginning of the current year and the date stored in the column MOSTRECENT:

```
daysdiff = datdif( intnx('year',today(),0,'begin'), mostrecent,
'act/act');
```

## **Debugging Tips**

when you see such a message:

NOTE: Line generated by the invoked macro "RENAME".

```
4 call symput ('selCol','dateval')
```

```
----
180
```

```
***** macro rename: selCol is &selCol
```

```
22: LINE and COLUMN cannot be determined.
```

then I think SAS is looking for the DATA STEP!!!!!!

\* I tried turning on/off OPTION SPOOL and this problem was fixed!!!

\* also there was some problem using the reserved word RENAME for the macro, apparently.

\* to check strings using logDebug I found that the ->&string<- produces less err. msgs.

The Phantom "MISSING MEND STATEMENTS":

\* this probably means you didn't "complete" or "exit" the last macro you executed.  
Solution is to submit the MEND statement for the last macro a couple more times and check the log...;

```
2652 %mrename();
```

```
WARNING: Missing %MEND statement.
```

\* sometimes you need to restart SAS and everything works!

### ***Dropping Datasets***

```
proc sql noprint;  
    drop table work.temp;  
quit;
```

- or (faster)

```
proc datasets lib = work nolist;  
    delete temp;  
run;
```

- or (faster still, deletes all work data)

```
proc datasets lib=work nolist nowarn kill;  
quit;
```

### ***Dropping Formats and Labels***

To drop formats and informats, you can use PROC DATASETS with the column followed by no arguments.

```
proc datasets lib=work nolist;  
    modify fmctype;  
    format id shortname en_US es_ES de_DE it_IT hu_HU;  
    informat id shortname en_US es_ES de_DE it_IT hu_HU;  
quit;
```

Also you can use the DATA step

```
data fmctype;  
    set fmctype;  
    format id;  
    label id=;  
run;
```

### ***FILENAME and LIBNAME***

- assigns a file to an alias;

```
FILENAME SOURCE 'd:\user\kl\SAS-Kurs-prgr\SourceCode\empinfo.dat';
```

- creates and assigns a new library;

```
LIBNAME IA 'd:\user\kl\ia';
```

### ***Filter Data using DATA step***

- using IF:

```

data work.temp;
    set md.singlepatlabval;
    if upcase(actuallabresult) in ('TRUE', 'FALSE');
run;

```

- same statement using WHERE:

```

data work.temp;
    set md.singlepatlabval;
    where upcase(actuallabresult) in ('TRUE', 'FALSE');
run;

```

- same statement using PROC SQL:

```

proc sql;
    create table work.temp
    as select PATID, SYSTEMLAB, ACTUALLABRESULT
    from md.singlepatlabval
    where upcase(actuallabresult) in ('TRUE', 'FALSE')
    order by 1,2,3;
quit;

```

### ***Finding duplicates (NODUPKEY, using FIRST and LAST)***

- this is simpler than creating your own key values
- if there are duplicates, nodupkey will delete the last sequential record of the duplicate record set

```

proc sort data = work.temp1 out=work.temp1_srted nodupkey;
    by ownerid;
run;

```

- Other way is to use the FIRST or LAST key variables:

```

data work.temp;
    infile datalines delimiter=',';
    length key $16. num 8.;
    input key $ num;
    datalines;

```

```
key1, 12.4
```

```
key1, 1020
```

```
key2, 13
```

```
key3, 15
```

```
run;
```

```

DATA nodup;
    SET temp;
    BY key;
    IF FIRST.key;
    OUTPUT;

```

```
RUN;
```

```

data _null_;
    set nodup;
    put key= num=;

```

```
run;
```

```
key=key1 num=12.4
```

```
key=key2 num=13
```

```
key=key3 num=15
```

NOTE: There were 3 observations read from the data set WORK.NODUP.

## ***FIRST. and LAST. Variables***

From the SAS online help:

*“How the DATA Step Identifies BY Groups*

In the DATA step, SAS identifies the beginning and end of each BY group by creating two temporary variables for each BY variable: **FIRST.variable** and **LAST.variable**. These temporary variables are available for DATA step programming but are not added to the output data set. “

Here is an example data step that uses both FIRST and LAST to tell what sequence number the current record is for the BY variable.

```
* adjust the start and end date;
data work.tpw;
  set work.treatmentsperweek end=eof;
  by patid;
  retain conseccount fixendcount 0;
  drop conseccount fixendcount;

  * if this is the first record for the patient, fix the startdate
  if first.patid then startdate = '01Jan1960'd; * default;
  else conseccount+1; * counter for "consecutive" records;

  * make the last end date for the patient fall at end of year;
  if last.patid then do;
    enddate = intnx('year',today(), 0 , 'end' );
    fixendcount+1;
  end;
  if eof then put "tpw, " _N_= conseccount= fixendcount=;
run;
```

## ***Functions inside Macros***

some functions are only available for use in DATA step!

- the following two lines won't work because you can't use the int function with %let

```
%macro test(step);
  %let teststep=%int(&step) - 1;
  %put teststep is &teststep;
%mend test;
%test(4);
WARNING: Apparent invocation of macro INT not resolved.
```

- here is how you can use INT properly in the macro to assign a variable named “teststep”:

```
%macro test(step);
  data temp; * can also do data _null_;
    step=&step; * assign the macro parameter &step to the column
               "step";
    call symput('teststep',step); * assign the new variable
                                   "teststep";
```

```

run;
%put teststep is &teststep;
%mend test;
%test(4);
teststep is          4

```

## ***GENNUM and OBS***

- reads only the “-1” (the backup) dataset, first 100 records

```

data work.temp;
    set reports.holidayin(gennum=-1 obs=100);
run;

```

- to output a gennum dataset (keeps only 1 more copy of the DS)

```

data reports.labdata (gennum=2);
    set work.temp;
run;

```

I don't like GENNUM because you lose track of GENMAX (the maximum number of datasets that SAS will create) and end up wasting lots of disk space. Better to create your “own” backups (using suffix numbers like results1, results2, or separate backup library).

## ***GOTO and a util macro “Numobs”***

*\* this macro is used by all programs to check for no results;*

```

%macro numobs(lib,dsn) / STORE ;
    %global numobs ;
    %let numobs=0 ;
    %*put checking lib &lib , dataset &dsn;
    proc sql noprint ;
        select nob - delobs into :numobs
            from dictionary.tables
            where libname=%upcase("&lib") and memname=%upcase("&dsn") ;
    quit ;
%mend numobs ;

```

```

%macro gototest;
    * create a test dataset;
    proc sql;
        create table test as
        select *
        from db2sas.patient
        where currentcompany eq 'ESXXX';
    quit;
    * if there are no results, then skip to the end of the macro;
    %numobs(work,test);
    %if (&numobs eq 0) %then %do;
        %put no records found, skipping to end;
        %goto program_end;
    %end;
    %else %do;
        * else do something;
        data test2;
            set test;
            if _N_=1 then output;

```

```

        run;
    %end;
    * the end block is here;
    %program_end:
        %put end of pgm;
%mend gototest;
%gototest;

```

```

no records found, skipping to end
end of pgm

```

## ***IF and ELSE***

```

* create a test dataset with columns PAID_VND and COST;

```

```

data work.temp;
    infile datalines delimiter=',';
    input paid_vnd $ cost 8.1;
    datalines;

```

```

0090350, 12.4
0027372, 1020
0001176, 13
0090450, 15
0059214, 10
0099999, 1000
0099999, 2000
0099999, 3000

```

```

run;

```

```

* split the test dataset into 2 destinations: PLAN and REFERRALS. SAS
will evaluate only up to the first condition that is TRUE;

```

```

DATA plan
    referrals;
SET temp;
format subgroup $25.;
    if paid_vnd = '0090350' then do;
        subgroup = 'PLAN - DAMERON' ;
        output plan;
    end;
    else if paid_vnd = '0027372' then do;
        subgroup = 'PLAN - MT DIABLO' ;
        output plan;
    end;
    else if paid_vnd = '0001176' then do;
        subgroup = 'PLAN - ALTA BATES';
        output plan;
    end;
    else if paid_vnd = '0090450' then do;
        subgroup = 'PLAN - EMANUEL' ;
        output plan;
    end;
    else if paid_vnd = '0059214' then do;
        subgroup = 'PLAN - STANISLAUS';
        output plan;
    end;
    else output referrals;
RUN;

```

## ***IN= Data Set Option***

(from SAS Help)

The IN= data set option creates a special boolean variable that indicates whether the data set contributed data to the current observation. The variable has a value of 1 when true, and a value of 0 when false. You can use IN= on the SET, MERGE, and UPDATE statements in a DATA step.

The following example shows a merge of the OLD and NEW data sets where the IN= option is used to create a variable named X that indicates whether the NEW data set contributed data to the observation:

```
data master missing;
  merge old new(in=x);
  by id;
  if x=0 then output missing;
  else output master;
run;
```

## ***Indexes***

*\* creates multiple indexes for improving SQL joins based on the indexes columns;*

```
PROC DATASETS LIBRARY = reports NOLIST;
  MODIFY TreatmentResults;
  INDEX CREATE currentcompany patid date eventtpl
PatientEventEncounID;
QUIT;
```

- I don't like to use "unique" indexes in SAS because it creates overhead and should be enforced in the source system anyways

*\* creates combined unique index on the columns "currentcompany" and "patid";*

```
PROC DATASETS LIBRARY = reports NOLIST;
  MODIFY TreatmentResults;
  INDEX CREATE treatmentresults_uk = (currentcompany patid) /
unique;
QUIT;
```

*\* remove all indexes on this table;*

```
PROC DATASETS LIBRARY = reports NOLIST;
  MODIFY TreatmentResults;
  INDEX delete _all_;
QUIT;
```

## ***INTNX Function (measuring ranges between dates)***

```
data work.temp;
  format today date9.;
  format firstday date9.;
  format firstday2 date9.;
  format lastday date9.;
```

```

today = today();
firstday = intnx('month',today, 0,'begin');
firstday2 = intnx('month',today(), 0,'begin');
lastday = intnx('month',today, 0,'end');
put today= firstday= firstday2= lastday=;
run;
today=19MAR2003 firstday=01MAR2003 firstday2=01MAR2003 lastday=31MAR2003

```

### **KEEP in DATA step**

- you can do it 2 ways
- 1: KEEP as an option in the "SET" statement;

```

data temp2;
    set temp (keep=key num);
run;

```

- 2: KEEP as its own statement;

```

data temp2;
    set temp;
    keep num step;
run;

```

- the 2<sup>nd</sup> way is easier to read

### **MOD() function**

- here I removed each alternating observation to create a new DS;
- could have also used a DO ... BY 2 loop;

```

DATA work.somerecs;
    SET work.temp;
    IF MOD(_N_, 2) = 0; * where 2/2=0 or 4/2=0;
    OUTPUT;
RUN;

```

### **Modifying and Updating Datasets**

- MODIFY changes the values in the dataset without making a copy in memory. This is only useful if you have very large datasets! (> 1 mil records)

Obs	key	num
1	key1	12.4
2	key1	1020.0
3	key2	13.0
4	key3	15.0

```

data temp;
    modify temp;
    num=num*1.5;
run;

```

```

proc print data=temp; run;

```

Obs	key	num
1	key1	18.6
2	key1	1530.0
3	key2	19.5

**Number Formats**

```

data numtest;
    format numX 8.3;
    format num2 best.;
    format num83 8.3;
    format num81 8.1;
    format whole best.;
    format onedec best.;
    num1 = 1.22334433465;
    numX = num1;
    num2 = num1;
    num83 = num1;
    num81 = num1;
    whole = 8;
    onedec = 8.1;
    put numX= num1= num2= num83= num81= whole= onedec=;
run;
numX=1.223 num1=1.2233443347 num2=1.2233443347 num83=1.223 num81=1.2 whole=8 onedec=8.1

```

**ODS HTML with PROC GCHART**

- everything you put in between ODS HTML and ODS HTML CLOSE will be outputted to HTML.

```

ODS HTML BODY = 'c:\temp\klsave\report.html';
GOPTIONS HSIZE = 6 VSIZE = 5 DEVICE = gif;

PROC GCHART DATA = ia.AllDataTrans(OBS=10);
    TITLE 'Revenue Figures';
    FORMAT FlightDate mmddyy10.;
    WHERE FlightDate BETWEEN '01Dec1999'd and '07Dec1999'd;
    LABEL RevenueType = 'Revenue Types';
    HBAR FlightDate /          sumvar=revenue
                          subgroup=RevenueType
                          nostats discrete;

RUN;
PROC PRINT DATA = ia.AllDataTrans(OBS=10);
RUN;
ODS HTML CLOSE;

```

**OUTPUT command (and EOF)**

- read the \_tf file in for processing, store all results in \_tf\_out, keys only in \_idx;
- EOF is to recognize the last observation for key processing;

```

data out1 out2;
    set &infile END = EOF;
    if case1 then output out1;
    else output out2;
    if EOF then do;
        put "end of infile! " N=;
    run;

```

```
run;
```

## Proc APPEND

```
proc append base = work.trd data = work.trd_cursor;  
run;
```

## PROC COMPARE

```
data temp2;  
  modify temp;  
  num=num*1.5;  
run;  
  
proc compare base=temp  
  compare=temp2;  
run;
```

- gives you output like this:

Number of Variables in Common: 2.

### Observation Summary

Observation	Base	Compare
First Obs	1	.
Last Obs	4	.

Number of Observations in Common: 0.

Number of Observations in WORK.TEMP but not in WORK.TEMP2: 4.

Total Number of Observations Read from WORK.TEMP: 4.

Total Number of Observations Read from WORK.TEMP2: 0.

Number of Observations with Some Compared Variables Unequal: 0.

Number of Observations with All Compared Variables Equal: 0.

## PROC CONTENTS

- use VARNUM to get a list of vars in the order they appear in the DS;

```
proc contents data = lab_num varnum; run;
```

-----Variables Ordered by Position-----

#	Variable	Type	Len	Format	Informat	Label
1	SYSTEMLAB	Char	32	\$32.	\$32.	SYSTEMLAB
2	error	Char	1	\$LABERRR.		
3	val	Char	128			

## PROC DATASETS

Proc DATASETS is probably the most useful procedure. Here are some of the things you can do:

- get information on a library hosted on a remote server:

```
libname user 'OMSKAL.SASWORK' access=readonly;

proc datasets lib=user details;
quit;
```

- delete datasets on that library (note that I removed `access=readonly` and thereby I have write access)

```
libname user 'OMSKAL.SASWORK';

proc datasets lib=user details;
    delete paid03 paid04 paid05 paid05 ref03 ref04 ref05 ref06;
quit;
```

- create and delete indexes

```
PROC DATASETS LIBRARY = reports NOLIST;
    MODIFY TreatmentResults;
    INDEX CREATE treatmentresults;
QUIT;
```

```
* remove all indexes on this table;
PROC DATASETS LIBRARY = reports NOLIST;
    MODIFY TreatmentResults;
    INDEX delete _all_;
QUIT;
```

## **PROC EXPORT**

\* the following proc export creates a text file delimited by semicolon;

```
proc export data = work.tn_31_sum
    outfile = 'D:\temp\tn_31_sum.txt'
    dbms = dlm replace;
    DELIMITER = ';' ;
run;
```

- simpler forms

```
proc export data = temp
    outfile = 'd:\temp\temp.csv';
run;
```

```
proc export data = temp DBMS = CSV REPLACE
    outfile = 'd:\temp\temp.csv';
run;
```

## **PROC FORMAT**

*using Control Files, Saving Formats in a Library*

- first create a control dataset, in this case “testfmt\_cntlin”. **Note:** if the format is character, the format name needs to start with a [\$] symbol! In the following example,

we are setting up the character format “\$Testfmt”.

```
data testfmt_cntlin;
  infile datalines delimiter=',';
  length start $2. label $32. fmtname $8.;
  input start $ label $ fmtname $; * the columns you need in any
format control file;
  * retain fmtname '$Testfmt'; * 8 char limit including $;
  datalines;
0,no error,$Testfmt
1,missing value,$Testfmt
2,value contains more then one comma,$Testfmt
3,value contains comma,$Testfmt
4,value contains more then one dot,$Testfmt
5,value contains %,$Testfmt
6,value contains >,$Testfmt
7,value contains E,$Testfmt
8,value contains boolean,$Testfmt
run;
```

- next, assign a format library that maps to a folder on your local drive (PC SAS) or mainframe dataset.

```
* assign FORMAT library;
libname f_help 'd:\reportDB\Formats\FMT_EN';

* "write" the format to the format library;
proc format lib=f_help cntlin=work.testfmt_cntlin;
run;
NOTE: Format $TESTFMT has been written to FORMATS.FORMATS.
```

- now you can write to your format library

```
* deassign FORMAT library;
libname f_help ;

* test using the format we just created;
proc datasets lib=work nolist;
  modify lab_num;
  format error $testfmt.;
quit;
```

#### Deassigning / removing a FORMAT

```
* deassign the format on the column ERROR;
proc datasets lib=work nolist;
  modify lab_num;
  format error;
quit;
```

- You can also use the DATA step with an empty assignment like in the above `format error;`

```
proc print data=lab_num; run;
Obs    SYSTEMLAB                error    val
1      5389-2                    8       true
2      13454-4                   8       true
```

3	5389-2	8	false
4	5389-2	8	false
5	5389-2	8	false
6	5389-2	8	false
7	5389-2	8	false
8	SystemLab_1701	1	
9	SystemLab_3557	7	2.147483647E7
10	SystemLab_3545	1	
11	SystemLab_3454	1	
12	SystemLab_3513	1	
13	SystemLab_3450	1	
14	SystemLab_3450	1	

1

### *Using PROC FORMAT to create your own formats*

- To check if a format is loaded in the library "formats", enter:

```
proc format library = formats fmtlib;
    select $PATIENT;
run;
```

- otherwise use SAS explorer and look in the "formats" catalog (right click on the format - "properties" to find out when it was created)

- A simple example of creating your own format:

```
proc print data=lab_num; run;
Obs    SYSTEMLAB                error    val
1      5389-2                    8        true
2      13454-4                    8        true
3      5389-2                      8        false
4      5389-2                      8        false
5      5389-2                      8        false
6      5389-2                      8        false
7      5389-2                      8        false
8      SystemLab_1701              1
9      SystemLab_3557              7        2.147483647E7
10     SystemLab_3545              1
11     SystemLab_3454              1
12     SystemLab_3513              1
13     SystemLab_3450              1
14     SystemLab_3450              1
15     SystemLab_3450              1
```

\* "set" the format before you apply it to the dataset;

```
proc format lib = formats;
    value $laberrr '0'=' '
                  '1'='missing value'
                  '2'='value contains more then one comma'
                  '3'='value contains comma'
                  '4'='value contains more then one dot'
                  '5'='value contains %'
                  '6'='value contains >'
                  '7'='value contains E'
```

```

                                '8'='value contains boolean';
run;

* now apply the new format $laberr to the dataset;
proc datasets lib=work nolist;
    modify lab_num;
    format error $laberrr.;
quit;

proc print data=lab_num; run;

```

Obs	SYSTEMLAB	error	val
1	5389-2	value contains boolean	true
2	13454-4	value contains boolean	true
3	5389-2	value contains boolean	false
4	5389-2	value contains boolean	false
5	5389-2	value contains boolean	false
6	5389-2	value contains boolean	false
7	5389-2	value contains boolean	false
8	SystemLab_1701	missing value	
9	SystemLab_3557	value contains E	2.147483647E7
10	SystemLab_3545	missing value	
11	SystemLab_3454	missing value	
12	SystemLab_3513	missing value	
13	SystemLab_3450	missing value	
14	SystemLab_3450	missing value	
15	SystemLab_3450	missing value	

### *Saving a format into an output file*

- use this if for instance if you are using a terminal emulator to run SAS/ connect in MVS and you need the format)

```

proc format library=FMTLIB2 CNTLOUT=csn_fmt;
    select $CSN;
    title 'FMTLIB Output for the $CSN. Format';
run;

```

### *PROC FORMAT to view existing format definitions*

- for instance, if you need to know all the formats that exists in the libraries you are referencing

```

LIBNAME MDRFMT 'PMG.PUB1.SAS.FORMATS' DISP=SHR;
LIBNAME FMTLIB2 'OSV.CMS.FORMATS1' DISP=SHR;

options compress=yes macrogen symbolgen ps=50 fmtsearch=(MDRFMT,
FMTLIB2);

ods listing exclude all; * in order to turn off long output listings;

proc format library=MDRFMT FMTLIB cntlout=WORK.MRDFRMTS;
run;
proc format library=FMTLIB2 FMTLIB cntlout=WORK.OSVCMS;
run;

```

```
PROC DOWNLOAD;  
RUN;
```

### PROC FREQ

- Another of my favorite PROCs. You want to make a simple calc of freq count, percent of total freq, etc.

```
proc freq data=reports.invoices noprint;  
  tables paid_vnd * paid_to_cat / out=freqdata;  
  where xsvc_from ge '01Nov04'd;  
run;
```

### PROC MEANS

- Excellent if you want to make a simple calc of summary;

```
proc means data=summary n sum fw=8;  
  output out=meansdata sum=;  
  var tracked zero_paid fee_for_service total_paid;  
  title 'Summary of Capped Payments in SPN';  
run;
```

The MEANS Procedure

Variable	N	Sum
Tracked	4543	28423192
Zero_paid	4543	0
Fee_for_service	4543	1.1987E8
total_paid	4543	1.483E8

ex. 2: include mean max min range stdev

```
proc means data=aoms_rollup n sum mean max min range fw=8;  
  output out=meansdata;  
  var number_invoices total_billed total_paid;  
  title 'Summary of Presentation and Taste Scores';  
run;
```

Summary of Presentation and Taste Scores						
1						
The MEANS Procedure						
Variable	N	Mean	Maximum	Minimum	Range	Std Dev
PresentScore	20	76.1500	93.0000	56.0000	37.0000	9.3768
TasteScore	20	81.3500	94.0000	72.0000	22.0000	6.6116

## **PROC REPORT**

Uses of this proc are many and varied. I use it primarily for creating a summarized report where I want to group variables. One problem is that the default output from this report (non-HTML) is spaced pretty wide if you have some longer columns. Use the DEFINE / FORMAT= option to work around this. Also PROC REPORT does not summarize string variables like PROC SQL SELECT (DISTINCT STRINGVAR) can do. So my experience is that you first need to create some kind of summarized dataset prior to running PROC REPORT on it.

```
* create the summarized input dataset for PROC REPORT;
proc sql;
  create table reportin as
  select specialty, service, vendor,
         count(*) as invoices,
         sum(billed) as billed,
         sum(paid) as paid
  from invoices2
  group by specialty, service, vendor;
run;

* output by service and specialty;
proc report data =reportin nowd headline ;

  column specialty service vendor invoices billed paid;

  * use DEFINE FORMAT= to make the column shorter in list output;
  define specialty / GROUP format=$26.;
  define service / GROUP format=$23.;
  define vendor / GROUP format=$28.;
  define billed / format=dollar10. 'Billed';
  define paid / format=dollar10. 'Paid';

  * subtotal after group line ;
  break after specialty / summarize skip;

  * create a total line at the end of the report;
  * note that OL option (overline) affects only the LIST output;
  rbreak after / ol summarize ;
run;
```

## **PROC SQL to create a table**

- can be used to create tables, but DATA step is faster.

```
proc sql;
CREATE TABLE WORK.ARCHIVESTATUS (
  tablename VARCHAR(64),
  uniquekey VARCHAR(64),
  datearchived NUM,
  recsarchived NUM,
  status VARCHAR(128)
);
quit;
```

## **PROC SQL SELECT INTO**

```
proc sql noprint;
    SELECT Language, Format, Streamed into :CurrentLanguage,
:CurrentFormat, :streamed
    FROM work.Languages
    WHERE n=&i      ;
quit;
```

## **PROC SQL SELECT WITH DESCENDING ORDER**

```
proc sql;
    create table ncauths_invoices as
    select * from invoices
    where ext_pay_ref in
    (select ext_pay_ref from ncauths)
    order by paid desc ;
quit;
```

## **PROC TABULATE**

Here is our sample dataset containing referrals, vendor category, and total paid:

```
PROC SQL;
    CREATE TABLE AOMS_SUM AS
    SELECT EXT_PAY_REF,
           PAID_YEAR,
           VND_CATEGORY_CODE,
           SUM(PAID) AS PAID
    FROM AOMS_PAID
    GROUP BY EXT_PAY_REF,
           PAID_YEAR,
           VND_CATEGORY_CODE;
QUIT;
```

The following 2 examples explain how PROC TABULATE can create a report based on this data:

**Example 1:** create a table with one row and one column variable (simple)

```
* from SAS Help:
Text in quotation marks in all dimensions specifies headings for the
corresponding variable or statistic. Although Sum is the default
statistic, it is specified here so that you can specify a blank for its
heading.;
* 1. Just show the sum by paid year;
proc tabulate data=aoms_sum;
    class vnd_category_code paid_year;
    var paid;
```

\* in the TABLE line, SUM does not need to be specified, because it is the default statistic. By setting its label to ' ', however, we can cause it not to be shown;

```
table vnd_category_code = 'Category', paid_year = 'Year (Paid)' *
paid=' ' * sum=' ';
run;
```

Result looks like this:

	Year (Paid)		
	2004	2005	2006
Category			
AMB	21618.87	9626.65	11555.70
BLB	314258.70	307259.63	27305.76
DME	1918728.42	1715564.89	247261.84
EVT	90405347.25	100538430.31	10243674.20

...

**Example 2:** create a table with one row and two column variables

\* 2. Show two statistics in the columns: volume and paid. \* the PARENTHESIS () encapsulates different statistics in the same "level". The [\*] symbol in the () is used to combine FORMATS with the statistics;

```
proc tabulate data=aoms_sum;
class vnd_category_code paid_year;
var paid;
table vnd_category_code = 'Category',
paid_year = 'Year (Paid)' * (paid=' '*format=dollar13.
n='Volume'*format=comma11.);
run;
```

	Year (Paid)				
	2004		2005		2006
	Sum	Volume	Sum	Volume	Sum
Category					
AMB	\$21,619	14	\$9,627	3	\$11,556
BLB	\$314,259	80	\$307,260	93	\$27,306
DME	\$1,918,728	1,948	\$1,715,565	1,924	\$247,262
EVT	\$90,405,347	10,122	\$100,538,430	18,216	\$10,243,674

## PROC TRANSPOSE

• Here is a simple example:

```
proc print data = temp; run;
Obs key num
1 key1 12.4
2 key2 13.0
```

```
3   key3      15.0
4   key1     1020.0
```

- you need to sort the dataset first (otherwise you will get errors)

```
proc sort data=temp; by key; run;
```

- now transpose table to display each “key” as a column;

```
proc transpose data=temp
  out = work.temp_trans let; * the let option allows duplicate
  values in the ID value;
  id key; * what goes on top;
  var num; * what goes in the col;
run;
```

- output: WARNING: The ID value "key1" occurs twice in the input data set.

```
proc print data = temp_trans; run;
```

```
Obs  _NAME_  key1  key2  key3
1    num    1020  13    15
```

- a slightly more complex example:

```
proc print data = temp; run;
```

```
Obs  key  num  name
1    key1  12.4  Kent
2    key2  13.0  Jerry
3    key3  15.0  James
4    key1  1020.0  Bob
```

```
proc sort data=temp; by key; run;
```

\* this step transposes the dataset temp, keeping the "key" column KEY on the left, putting

NAME on the top, and using NUM as the variable to transpose by;

```
proc transpose data=temp
  out = work.temp_trans; *let is not required because dups in KEY
  will be handled;
  id name; * what goes on top;
  var num; * what goes in the col;
  by key; * your key column(s) - dups will be removed automatically;
run;
```

```
proc print data = temp_trans; run;
```

```
Obs  key  _NAME_  Kent  Bob  Jerry  James
1    key1  num    12.4  1020  .      .
2    key2  num    .      .      13     .
3    key3  num    .      .      .      15
```

## Proc UNIVARIATE

Galina showed me you can do a lot of different statistical functions using PROC UNIVARIATE. Note that the “by” together with “var” lines designate which columns to show. The rest of the math functions are applied to the columns in the order designated by “var”.

```
proc univariate data=set2 noprint;
  var billed paid btotal_days ded_copay 1;
```

```

output out=test sum= billed paid bttotal_days ded_copay l n=nbilled
mean= abilled median=mbilled q1=q1 q3=q3 p5=p5 p95=p95 mode=mode;
by ext_pay_ref lob ;
run;

```

- from the [SAS documentation](#) :

The UNIVARIATE procedure provides the following:

- descriptive statistics based on moments (including skewness and kurtosis), quantiles or percentiles (such as the median), frequency tables, and extreme values
- histograms and comparative histograms. Optionally, these can be fitted with probability density curves for various distributions and with kernel density estimates.
- quantile-quantile plots (Q-Q plots) and probability plots. These plots facilitate the comparison of a data distribution with various theoretical distributions.
- goodness-of-fit tests for a variety of distributions including the normal
- the ability to inset summary statistics on plots produced on a graphics device
- the ability to analyze data sets with a frequency variable
- the ability to create output data sets containing summary statistics, histogram intervals, and parameters of fitted curves

You can use the PROC UNIVARIATE statement, together with the VAR statement, to compute summary statistics.

### **Quotes vs. Doublequotes**

```
libname f_help "&DWHROOT.\Formats\FMT_ES"; --> will get resolved to
d\reportdb\formats ...
```

```
libname f_help '&DWHROOT.\Formats\FMT_ES'; --> will NOT get resolved.
```

### **RETAIN and If-then-else**

```

data out1 out2;
  set &infile END = EOF;
  retain countupdated 1.; * use RETAIN for our counter;
  if casel then do;
    flag = "updated";
    coutupdated+1;
    output out1;
  end;
  else output out2;
  if EOF then do;
    put "end of infile! " N= countupdated=;
  end;
run;
run;

```

### **SCAN and LENGTH functions**

```

data work.temp;
  nrgood = "1.443";

```

```

nrbad = "1.442.424";
scangood = scan(nrgood,3,"."); * scans for the 3rd substring
separated by the "token" [.];
scanbad = scan(nrbad,3,".");
lengood = length(scangood); * measure length of string "scangood";
lenbad = length(scanbad);
put _ALL_;run;
nrgood=1.443 nrbad=1.442.424 scangood= scanbad=424 lengood=1 lenbad=3 _ERROR_=0
_N_=1

```

### **SQL String creating method**

\* this method reads an input field and creates a sqlstring for use in a WHERE IN statement.

```

data temp;
  set reports.bsvc_from_top15_tins end=eof;
  retain sqlstr;
  format sqlstr $256.;
  if _N_ le 1 then sqlstr=compress("'"||paid_tin);
  else sqlstr=compress(sqlstr||','||paid_tin);
  if eof then do;
    sqlstr=compress(sqlstr||"'");
    put sqlstr=;
  end;
run;
sqlstr='680396600','770000209','770243143','770465765','770476190','941
201197','941461843','941698084','941719654','941724925','942281314','94
2494000','942832405','942965646','953372911'

```

### **String Parsing Functions (where to find them)**

see “Base SAS Software” -> “SAS Language Reference Dictionary” -> “Dictionary of Language Elements” -> “Functions and Call Routines” in the V8.2 online help (<http://v8doc.sas.com/sashtml/>)

### **SUBSTR function (ex. Streamed data)**

```

* since HU_HU in DESCRIPTIVINFRMTN is streamed process it using the
substring algorithm;
%if (targetlanguage eq "hu_HU") %then %do;
  data patientdiagnosis_ctlin;
    set patientdiagnosis_ctlin end=eof;
    format label_cleaned $128.;
    drop i;
    if label ne "" then do i = 2 to 256 by 2;
      * we only want the even character positions;
      * we have to handle " " in the string - so "trim" and
"compress" wont work;
      if i eq 2 then temp = substr(label,i,1);
      else temp = substr(label_cleaned,1,((i/2)-
1))||substr(label,i,1);
      label_cleaned = temp;
    end;
  run;
%end;

```

```

        end;
        * if eof then put _N=_;
run;
%end;

```

### ***Time / Date Functions (INTCK and INTNX)***

- intck: count the NUMBER OF INTERVALS BETWEEN 2 DATES:  
\* for example, use it to calculate the number of wednesdays ('week.4') between firstofmonth and thisdate;

```
nweddays = intck( 'week.4', firstofmonth, thisdate );
```

- intnx: count days in range:

```
endDateWork = intnx( 'month',today(),0, 'end');* last day of this month;
```

### ***TRANSLATE function***

- TRANSLATE is useful for editing inside strings:

```

%let teststring = "Kent! Martin";
data _null_;
    call symput('columnlistsep', translate(&teststring , "X" , "!"));
run;
%put columnlistsep = &columnlistsep;
30 %put columnlistsep = &columnlistsep;
columnlistsep = KentX Martin

```

### ***Unique or Duplicate Values Check***

\* the following step creates a "dups" dataset with records having duplicate ids in the dataset "sourcedata";

```

proc sql;
    create table dups as
    select * from sourcedata
    group by id
    having count(*) > 1;
quit;

```

\*\*\* END OF DOCUMENT \*\*\*

**\* Special Thanks to:  
Calcucare AG, Freiburg Germany**