The Most Useful New Parts of SAS® 9

Phil Mason, Wood Street Consultants, England

ABSTRACT

SAS® 9 has many new features, as well as all the old ones. This paper will try to outline some of the best new features that I have found over the last 3 years that I have been using SAS 9.

INTRODUCTION

The paper is based on a series of tips & techniques that I publish to a list of subscribers to my newsletter (see *www.woodstreet.org.uk*). This being a hands on workshop means that I will outline the feature we are looking at and then we will follow through some example code to see how the feature works and what can be varied. Remember that the tips & techniques covered here are only a small subset of everything that is available in SAS 9. You should make sure that you read the What's New documentation for the version of SAS 9 that you have. And don't forget to always read the changes and enhancements for each new service pack you get too.

DATA STEP, MACRO & SQL FEATURES

WRITING MESSAGES TO THE LOG, WHILE WRITING TEXT ELSEWHERE

In SAS 9 there is a new statement called PUTLOG, which explicitly writes to the SAS LOG. This means that you can direct regular PUT statements to write to another destination, and write to the log using PUTLOG without the need to redirect output to the LOG with a FILE LOG statement.
PUTLOG is similar to the ERROR statement, except PUTLOG does not set _error_ to 1. The ERROR statement is also available in SAS 6 & 8.

SAS LOG

```
55   data test ;
56     put 'This goes to LOG by default' ;
57     file print ;
58     put 'This goes to OUTPUT window, since I selected print' ;
59     putlog 'but this still goes to the LOG' ;
60     put 'This goes to OUTPUT' ;
61     putlog 'NOTE: and I can write proper messages using colours' ;
62     putlog 'WARNING: ...' ;
63     putlog 'ERROR: ...' ;
64   run ;

This goes to LOG by default
but this still goes to the LOG
NOTE: and I can write proper messages using colours
WARNING: ...
ERROR: ...
NOTE: 2 lines were written to file PRINT.
NOTE: The data set WORK.TEST has 1 observations and 0 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds
```

IN OPERATOR NOW ACCEPTS INTEGER RANGES

In SAS 9 the IN operator has been enhanced to accept integer ranges. This works well with IF statements, but doesn't seem to work with WHERE statements (yet).

SAS LOG

```
73   data sample ;
74     set sashelp.class ;
75       if age in (11, 13:15, 18:25) ;
76   run ;
```

```
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set WORK.SAMPLE has 13 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.02 seconds
```

CONCATENATING STRINGS THE EASY WAY

In SAS 9 there is a new function called CATX which makes concatenating strings easy. It will concatenate any number of character strings, removing leading & trailing blanks and inserting a separator.

SAS LOG

```
1    data test ;
2      a='  Phil         ' ;
3      b='   Mason ' ;
4      c=trim(left(a))!!' '!!left(b) ;
5      d=catx(' ',a,b) ;
6      put c= d= ;
7    run ;

c=Phil Mason d=Phil Mason
NOTE: The data set WORK.TEST has 1 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.55 seconds
      cpu time            0.06 seconds
```

COUNTING WORDS

Rather than using a more convoluted technique for counting the number of occurrences of a word in a character string, in SAS 9 we can now use the COUNT function. It will simply count the number of sub-strings that occur in a string, optionally ignoring the case (as in my example).

SAS LOG

```
15   data _null_ ;
16     sentence='This is ONE way of using one in One sentence' ;
17     num=count(sentence,'one','i') ;
18     put num= ;
19   run ;

num=3
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

WAYS TO USE THE PUT STATEMENT

The PUT statement is a very flexible tool in the data step programmers toolkit. Here are some different ways of using it – hopefully there may be one or two you have not seen before. These are not new in SAS 9, but I threw this in because I think its very useful.

Statement	*Explanation*
Put x y z ;	Write values of 3 variables out separated by a space
Put ‘hello’ ’09’x ;	Write text followed by hexadecimal 09 – which is a tab character (in ASCII)
Put 132*’_’ ;	Write 132 underscores
Put #3 @44 cost ;	Write value of cost out beginning at line 3 column 44
Put var 1-5 ;	Write the value of var out into the columns from column 1 to column 5
Put cost dollar12.2 ;	Write the value of cost out using the dollar12.2 format
Put (a b) (1. ‘,’ $3.) ;	Write the value of a out using a 1. format, then a comma, then the value of b using a $3. format
Put _infile_ ;	Write out the current input buffer, as read by the last input statement
Put _all_ ;	Write out the values of all variables, including _error_ and _n_
Put _ods_ ;	Write out the default or previously defined variables to the ODS destination

Put a b c @ ;	Write values of variables a, b & c out, separated by spaces & keep line "open" so that next put statement will continue on. If we reach end of data step iteration then line is "closed"
Put d e @@ ;	Write values of variables d & e out, separated by spaces & keep line "open", even if we reach end of data step iteration.
Put @10 name ;	Write value of name at column 10
Put @pos name ;	Write value of name at column specified in variable pos
Put @(3*pos) name ;	Write value of name at column calculated by value of pos multiplied by 3
Put a +3 b ;	Write value of a followed by 3 spaces and then value of b
Put a +gap b ;	Write value of a followed by a number of spaces specified in variable gap, and then value of b
Put a +(2*gap) b ;	Write value of a followed by a number of spaces calculated by value of gap multiplied by 2, and then value of b
Put #2 text ;	Write value of text at line 2
Put #line text ;	Write value of text at line specified in variable line
Put #(line*3) text ;	Write value of text at line calculated by value of line multiplied by 3
Put line1 / line2 ;	Write value of line1, then go to a new line and write value of line2
Put @1 title overprint @1 '_____' ;	Write value of title and then overprint underscores on that value. This only works on some print destinations and usually looks wrong on the screen.
Put _blankpage_ ;	Ensure that a totally blank page is produced. This means that if we had written even 1 character on a page, then that page will be written as well as another totally blank page.
Put _page_ ;	This finishes the current page, causing the next thing we write out to be on a new page.
Put name= phone= ;	Write the text "name=" followed by the value of name, and then "phone=" followed by the value of phone.
Put my_big_array(*) ;	Write each element of my_big_array in the form variable=value

USING PERL REGULAR EXPRESSIONS FOR SEARCHING TEXT

In SAS 9 there are new functions available for using PERL regular expressions to search text for sub-strings. There are two parts to using these, which I demonstrate in the code below.

1. Parse your PERL regular expression using the PRXPARSE function. This returns a pattern-id which can be used in other PERL functions.
2. Search using the parsed expression by using the PRXSUBSTR function. This returns the position & length of the text found, or 0 if none was found.

Note: For a useful quick reference on PERL regular expressions see http://www.erudil.com/preqr.pdf

SAS PROGRAM

```
data _null_;
  if _N_=1 then
    do;
      retain patternID;
      pattern = "/ave|avenue|dr|drive|rd|road/i";
      patternID = prxparse(pattern);
    end;
  input street $80.;
  call prxsubstr(patternID, street, position, length);
  if position ^= 0 then
    do;
      match = substr(street, position, length);
      put match : $QUOTE. "found in " street : $QUOTE.;
    end;
  datalines;
153 First Street
6789 64th Ave
4 Moritz Road
```

```
7493 Wilkes Place
;
run ;
```

LINES WRITTEN TO LOG

```
"Ave" found in "6789 64th Ave"
"Road" found in "4 Moritz Road"
```

FLEXIBLE NEW DATE FORMAT

In SAS version 9 there is a new date informat which interprets dates being read based on the value of a system option. The system option is called DATESTYLE and it is used to identify sequence of month, day and year when the ANYDATE informat data is ambiguous. When dates being read are not ambiguous, then the option is ignored and date is read correctly.

SAS LOG

```
33   options datestyle=mdy;
34   data _null_;
35     date=input('01/02/03',anydtdte8.); * ambiguous date ;
36     put date=date9.;
37   run;

date=02JAN2003
NOTE: DATA statement used (Total process time):
      real time           0.51 seconds
      cpu time            0.00 seconds


38   options datestyle=ydm;
39   data _null_;
40     date=input('01/02/03',anydtdte8.); * ambiguous date ;
41     put date=date9.;
42   run;

date=02MAR2001
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


43   options datestyle=myd;
44   data _null_;
45     date=input('01/31/2003',anydtdte10.); * unambiguous date, so option ignored
;

46     put date=date9.;
47   run;

date=31JAN2003
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

NEW IF FUNCTIONS

It's always worth looking at "What's New" for the latest SAS release. It often reveals very useful new additions, which often turn out to be available in prior releases of SAS also, though they are undocumented there.
Two of the more useful new functions are the IFN & IFC functions. These are useful for more efficient coding and can be used anywhere a function can be used, even in where clauses or macro language (using %sysfunc). In fact using the IF functions from macro language means that you can use IF logic in open code, rather than being forced to use a macro program.

IFN(condition, true-numeric-value, false-numeric-value, missing-numeric-value)

IFN returns a numeric value. It returns the true, false or missing value depending on whether the condition is true, false or missing.

IFC(condition, true-character-value, false-character-value, missing-character-value)

IFC returns a character value. It returns the true, false or missing value depending on whether the condition is true, false or missing.

EXAMPLE SAS LOG

```
21   * without IFN function ;
22   data test1 ;
23     set sashelp.class ;
24    * set entry price based on age ;
25     if age>=13 then
26       price=12.50 ;
27     else
28       price=8 ;
29   run ;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set WORK.TEST1 has 19 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds


30   * with IFN function ;
31   data test2 ;
32     set sashelp.class ;
33    * set entry price based on age ;
34     price=ifn(age>=13,12.50,8) ;
35   run ;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set WORK.TEST2 has 19 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds


36   %put %sysfunc(ifc(&sysscp=WIN,You are using Windows!,You are not using
Windows)) ;
You are using Windows!
```

SORTING ARRAY ELEMENTS

In SAS 9 there is a new routine which can be used to sort the values of a list of variables passed to it. SORTN should be used for sorting numerics, and SORTC for character values. If the variables belong to an array then these sort routines effectively sort the values of the array. The sorts are always done into ascending sequence, but by specifying the variables in reverse order you can effectively sort in descending sequence.
Note: character variables must be same length.

SAS LOG

```
88     call sortn(of v50-v1);
89     put 'Down: ' v(1)= v(2)= v(3)= v(48)= v(49)= v(50)= ;
90    * sort values between 3 character variables ;
91    * note: character variables must be same length to avoid errors ;
92     x='3 dogs ' ;
93     y='1 cat  ' ;
94     z='2 frogs' ;
95     call sortc(x,y,z) ;
96     put x= y= z= ;
97   run ;
```

```
NOTE: The SORTN function or routine is experimental in release 9.0.
Up: v1=1 v2=2 v3=3 v48=48 v49=49 v50=50
Down: v1=50 v2=49 v3=48 v48=3 v49=2 v50=1
NOTE: The SORTC function or routine is experimental in release 9.0.
x=1 cat y=2 frogs z=3 dogs
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds
```

SOME BASICS OF CREATING XML IN SAS 9

SAS 9 handles XML extremely well via the XML libname engine. To create an XML file I could use some code like this. Remember to free the libref so you can access the XML file once you have created it.

SAS CODE TO CREATE XML

```
libname test xml 'c:\test.xml' ;
data test.sample ;
  name='Phil Mason' ;
  age=40 ;
  sex='M' ;
  phone='01491 824905' ;
  country='England' ;
run ;
libname test ;
```

XML CREATED IN C:\TEST.XML

```
<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
   <SAMPLE>
      <name> Phil Mason </name>
      <age> 40 </age>
      <sex> M </sex>
      <phone> 01491 824905 </phone>
      <country> England </country>
   </SAMPLE>
</TABLE>
```

WHAT HAPPENS IF YOU WRITE MULTIPLE DATASETS TO AN XML FILE?

The following code will only write the last file (test.c) to the XML.

```
libname test xml 'c:\test.xml' ;
data test.a ; x=1 ; run ;
data test.b ; x=1 ; run ;
data test.c ; x=1 ; run ;
libname test ;
```

This code will also only write the last file (test.c) to the XML, even though the log seems to indicate that they have all been written.

```
libname test xml 'c:\test.xml' ;
data test.a test.b test.c ; x=1 ; run ;
libname test ;
```

I could use the following code though, which *would* create an XML file containing all three tables.

```
libname test xml 'c:\test.xml' ;
data a b c ; x=1 ; run ;
proc copy in=work out=test ;
  select a b c ;
run ;
libname test ;
```

SAS CODE TO USE XML

To make use of the XML file produced above I could simply point my libname statement at it, and then refer to the table name defined in the XML.

```
libname test2 xml 'c:\test.xml' ;
proc print data=test2.sample ;
run ;
libname test2 ;
```

IGNORING CASE FOR COMPARISONS

When comparing text in SAS you will note that uppercase and lowercase are not equivalent. So “abc” does not equal “ABC”. Sometimes you may want to compare text and ignore case. There are several ways to do this.

METHOD 1

If using a where clause you can use the **upcase()** function against any variables, and make sure that any text literals are in uppercase.

e.g.

```
proc print data=mydata ;
  where upcase(name)='PHIL' ;
run ;
```

METHOD 2

In a datastep convert all text to uppercase, so you will then compare uppercase with uppercase. You also need to make sure that any text in custom formats is also in uppercase. The following code demonstrates a simple but effective technique for creating an array containing all character variables and then processing them in some way. You can also use the **VNAME()** function to handle individual variables as exceptions.

e.g.

```
data test ;
  set sashelp.prdsale ;
 * make an array containing all character variables currently defined ;
  array c(*) _character_ ;
 * loop through array ;
  do _i=1 to dim(c) ;
    vname=vname(c(_i)) ;
   * uppercase each character variable, except COUNTRY ;
    if vname^="COUNTRY" then
      c(_i)=upcase(c(_i)) ;
  end ;
run ;
```

NEW CURRENCY FORMATS FOR UK AND OTHERS

Some of the SAS 9.1.3 features include new National Language formats, including a long awaited currency format for British Pounds. There are a range of others, reflecting a range of currencies from around the world:

- NLMNLAUD
- NLMNLCAD
- NLMNLCHF
- NLMNLCNY
- NLMNLDKK
- NLMNLEUR
- NLMNLGBP
- NLMNLHKD
- NLMNLILS
- NLMNLJPY
- NLMNLKRW
- NLMNLMYR
- NLMNLNOK
- NLMNLNZD
- NLMNLPLN
- NLMNLRUR
- NLMNLSEK
- NLMNLSGD
- NLMNLTWD
- NLMNLUSD
- NLMNLZAR

SAMPLE SAS PROGRAM

```
DATA _NULL_;
x = 1234.56;
PUT 'UK - '          @20 x NLMNLGBP. @40 x NLMNIGBP.
```

```
  / 'Japan - '       @20 x NLMNLJPY. @40 x NLMNIJPY.
  / 'Australia - '   @20 x NLMNLAUD. @40 x NLMNIAUD.
  / 'Euro - '        @20 x NLMNLEUR. @40 x NLMNIEUR.
  / 'New Zealand - ' @20 x NLMNLNZD. @40 x NLMNINZD. ;
RUN;
```

SAMPLE SAS LOG

```
77   DATA _NULL_;
78   x = 1234.56;
79   PUT 'UK - '          @20 x NLMNLGBP. @40 x NLMNIGBP.
80     / 'Japan - '       @20 x NLMNLJPY. @40 x NLMNIJPY.
81     / 'Australia - '   @20 x NLMNLAUD. @40 x NLMNIAUD.
82     / 'Euro - '        @20 x NLMNLEUR. @40 x NLMNIEUR.
83     / 'New Zealand - ' @20 x NLMNLNZD. @40 x NLMNINZD. ;
84   RUN;

UK -                  £1,234.56         GBP1,234.56
Japan -                JPY1,235            JPY1,235
Australia -          AU$1,234.56         AUD1,234.56
Euro -                 €1,234.56         EUR1,234.56
New Zealand -        NZ$1,234.56         NZD1,234.56
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

PUTTING NUMBERS IN MACRO VARIABLES A BETTER WAY

In SAS 9 there is a new function called SYMPUTX which creates a macro variable from a numeric variable without writing a note to the log, and trims leading & trailing blanks. It will use a field up to 32 characters wide and you can optionally tell it which symbol table to put the macro variable into.
The example below show how SAS does an automatic type conversion and uses BEST12.2 to convert a numeric to character in line 44. The following line shows how we can explicitly do the conversion and trim the result. The next line shows how to use SYMPUTX to simplify the process.

SAS LOG

```
42   data test ;
43     my_val=12345 ;
44     call symput('value0',my_val) ; * auto conversion done ;
45     call symput('value1',trim(left(put(my_val,8.)))) ; * v8 ;
46     call symputx('value2',my_val) ; * SAS 9 ;
47   run ;

NOTE: Numeric values have been converted to character values at the places given
by:
      (Line):(Column).
      44:24
NOTE: The data set WORK.TEST has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


48   %put =->&value0<-==->&value1<-==->&value2<-=;
=->       12345<-==->12345<-==->12345<-=
```

CREATING A RANGE OF MACRO VARIABLES FROM SQL WITH LEADING ZEROES

In versions of SAS prior to SAS 9 you could create ranges of macro variables, but could not use leading zeroes in the macro variable names. So in 8.2 you might end up with macro variables like var8, var9, var10. But in SAS 9 you could create variables like var08, var09, var10. This can make further use of those macro variables easier to code.

SAS PROGRAM

```
proc sql noprint ;
  select name into :name01-:name19 from sashelp.class ;
```

USE REGULAR EXPRESSIONS IN SQL

Chris Brooks (Office of National Statistics - UK) pointed out to me that you can use regular expressions within SQL. This can be quite powerful in selecting data that matches certain conditions. The following example shows a simple regular expression which selects only quarterly periods from a table contain years, quarters & months.

PROGRAM

```
data test ;
  length period $ 7 ;
  input period ;
cards ;
2005
2005Q1
2005JAN
;;
run ;
proc sql;
  create table qtrs as
    select *
      from test
        where prxmatch("/\d\d\d\d[qQ][1-4]/",period) ;
quit;
proc print data=qtrs ;
run ;
```

LOG

```
3189  data test ;
3190    length period $ 7 ;
3191    input period ;
3192  cards ;

NOTE: The data set WORK.TEST has 3 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           1.10 seconds
      cpu time            0.03 seconds


3196  ;;
3197  run ;
3198  proc sql;
3199    create table qtrs as
3200      select *
3201        from test
3202          where prxmatch("/\d\d\d\d[qQ][1-4]/",period) ;
NOTE: Table WORK.QTRS created, with 1 rows and 1 columns.

3203  quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.95 seconds
      cpu time            0.04 seconds


3204  proc print data=qtrs ;
3205  run ;

NOTE: There were 1 observations read from the data set WORK.QTRS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.70 seconds
```

```
        cpu time            0.03 seconds
```

OUTPUT

```
                                                                    The SAS
System                                                09:51 Thursday, June 15, 2006   1

                                                                             Obs
period

                                                                               1
2005Q1
```

BONUS QUICK TIPS

COUNT

Count up the number of substrings in a string.

```
data _null_ ;
  long='a b c d a e d a e t g d a c s' ;
  num_a=countc(long,'a') ;
  put num_a= ;
run ;
```

COUNTC

Count up the number of characters in a string.

```
data _null_ ;
  long='dog cat rat bat dog camel dingo snake bigdog' ;
  num_dog=count(long,'dog') ;
  put num_dog= ;
run ;
```

CHOOSEC & CHOOSEN

Choose the nth element from a list of items, either numeric or character.

```
data _null_ ;
  third=choosec(3,'dog','cat','rat','bat') ;
  put third= ;
  last=choosec(-1,'dog','cat','rat','bat') ;
  put last= ;
run ;
```

SOME MORE WORTH LOOKING AT

- FIND & FINDC
- IFC & IFN
- STRIP
- COALESCE & COALESEC
- FILENAME fileref CLIPBOARD ;
- PROC PROTO & PROC FCMP
- PROC SYLK
- PROC EXPORT SHEET=
- %SYMEXIST
- LIBNAME libref META
- _NEW_
- OPTION MPRINTNEST

ODS & GRAPH FEATURES

PUTTING MULTIPLE GRAPHS & TABLES ON AN HTML PAGE

The guys at SAS have produced a wonderful new tagset (thanks Eric!) called the HTML Panel tagset. This will be available in SAS 9.2, but can be downloaded and used now from this page http://support.sas.com/rnd/base/topics/odsmarkup/htmlpanel.html, which also describes how to use it.

The tagset lets you do pretty much anything that you can do with the normal ODS HTML destination, except you can break your page up into rows and columns. So you could have 2 graphs at the top, followed by a table, then perhaps 3 columns with graphs in each, etc. Just use your imagination.

The following code gives you a bit of an idea what can be done with this tagset. It starts with 4 columns of bar charts, over 3 rows. Then it has a report. And finally 5 columns of pie charts over 2 rows.

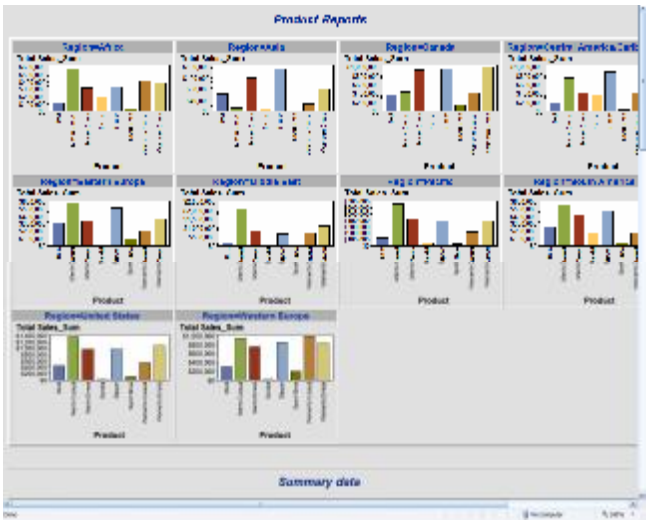
Note: before you can use this tagset you need to download it and run the proc template code in SAS to define it.

SAMPLE PROGRAM

```
%let panelcolumns = 4;
%let panelborder = 4;
ods tagsets.htmlpanel file="C:\bypanel2.html" gpath='c:\' options(doc='help');
goptions device=java xpixels=320 ypixels=240;
title1 'Product Reports' ;
footnote1 ;
proc summary data=sashelp.shoes nway ;
  class region product ;
  var stores sales inventory returns ;
  output out=sum sum= mean= /autolabel autoname ;
run ;
proc gchart data=sum ;
  by region ;
  vbar product / sumvar=sales_sum pattid=midpoint discrete ;
run;
quit;
proc summary data=sashelp.shoes nway ;
  class region subsidiary ;
  var stores sales inventory returns ;
  output out=sum sum= mean= /autolabel autoname ;
run ;
%let panelcolumns = 5;
%let panelborder = 1;
ods tagsets.htmlpanel ;
title 'Summary data' ;
proc print data=sum ;
run ;
title 'Subsidiary Reports' ;
%let panelcolumns = 5;
%let panelborder = 1;
ods tagsets.htmlpanel ;
goptions dev=java xpixels=160 ypixels=120;
proc gchart data=sum ;
  by region ;
  pie subsidiary / sumvar=sales_sum discrete ;
run;
quit;
ods _all_ close;
```

OUTPUT

Screen Shot 1



Screen Shot 2

AREA CHARTS WHICH CAN COMPARE MAGNITUDES OF VARIABLES IN CATEGORIES

The GAREABAR procedure in SAS/Graph lets you compare the magnitudes of 2 variables for each category of data using exact and relative magnitudes of values. It only works with ODS using the *activex* or *actimgx* driver though.
The following example plots *sales* on the x-axis, relative percent of number of *salespersons* on the y-axis, with a bar for each *site*. Additionally each bar is sub-grouped by *quarter*.

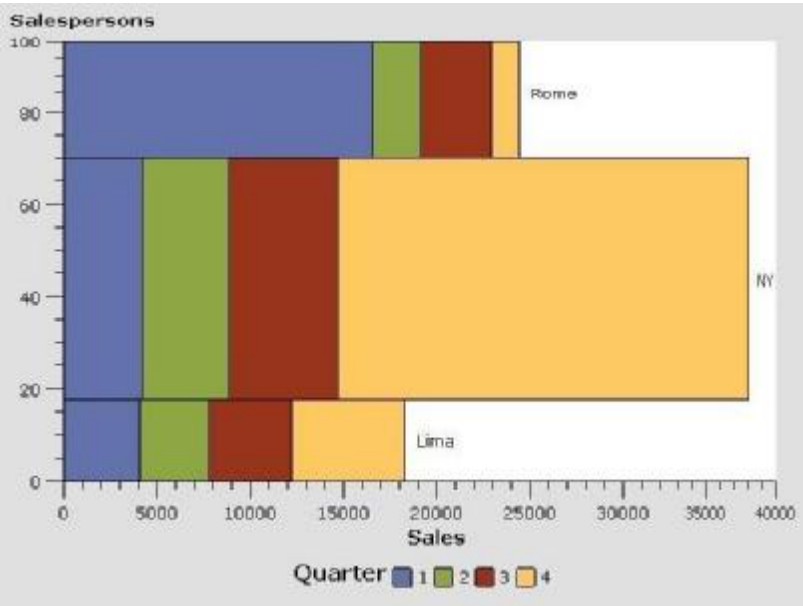
SAS LOG

```
goptions reset=all device=activex ;

ods html file='c:\test.html' ;

data totals;
   input Site $ Quarter $ Sales Salespersons;
cards;
Lima    1 4043.97    4
NY      1 4225.26   12
Rome    1 16543.97   6
Lima    2 3723.44    5
NY      2 4595.07   18
Rome    2 2558.29   10
Lima    3 4437.96    8
NY      3 5847.91   24
Rome    3 3789.85   14
Lima    4 6065.57   10
NY      4 23388.51  26
Rome    4 1509.08   16
;
proc gareabar data=totals;
   hbar site*salespersons /sumvar=sales
                           subgroup=quarter
                           rstat=SUM
                           wstat=PCT;
run ; quit ;

ods html close;
```



PLOT DETAILS OF SLICES IN A PIE GRAPH

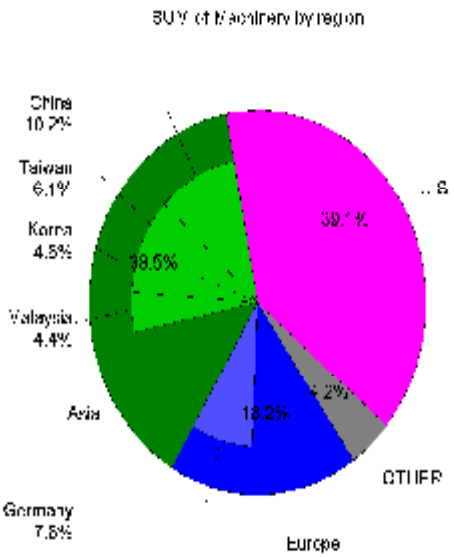
In SAS 9 there is a new parameter in PROC GCHART that can be used when making PIE charts. It allows you to produce an inner pie overlay, showing major components that make up outer pie slices. This can be useful to get even more information into your chart. See the example below.

SAS PROGRAM

```
ods html file='c:\test.html' gpath='c:\' ;
```

```
goptions reset=all device=png
         xmax=20 ymax=12          /* make PNG bigger */
         ftext='Arial' htext=4pct /* use some nice looking text */;

data countries;
  input country $ 1-14 region $16-26 Machinery;
  cards;
Taiwan          Asia       6.1
Korea           Asia       4.6
Malaysia        Asia       4.4
Malaysia2       Asia       3.9
Malaysia4       Asia       3.9
Malaysia5       Asia       1.5
U.S.            U.S.       39.1
Belgium         Europe     2.6
Germany         Europe     7.8
United Kingdom  Europe     3.9
France          Europe     3.9
Santa           Antarctica 1.1
Bob             Antarctica 1.0
Cydonia         Mars       1.1
Tims House      Mars       1.0
China           Asia       10.2
Malaysia3       Asia       3.9
;
run;
proc gchart;
  pie region / angle=320
               slice=outside
               percent=inside
               value=none
               sumvar=Machinery
               detail_percent=best
               detail=country
               descending ;
run; quit;
ods html close ;
```



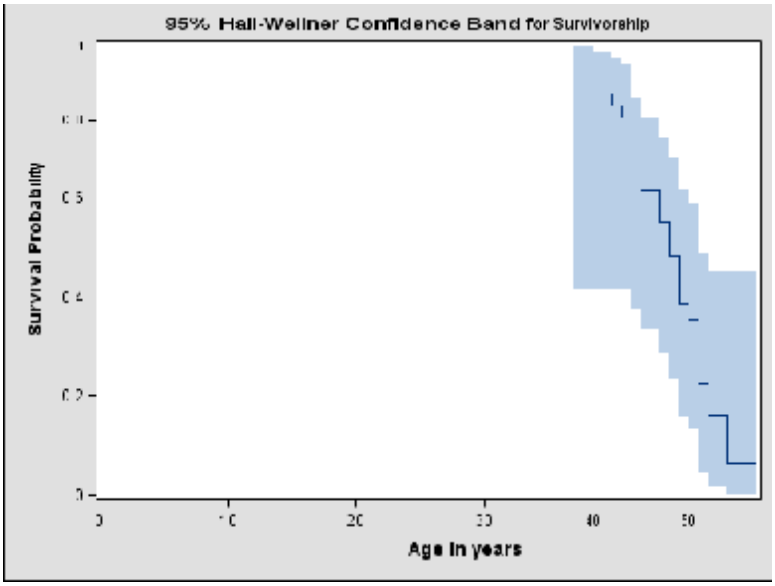
GENERATING GRAPHS AUTOMATICALLY FOR SOME PROCEDURES

There is a new feature available for some procedures in SAS 9 which will produce a graphic automatically for certain statistical procedures that are used. One such procedure is PROC LIFETEST. To get the graph you simply turn ODS GRAPHICS ON and then choose the graph required using the plots=(variable) option on proc lifetest.

SAMPLE CODE

```
ods listing close ;
ods html file='lifetest.html' style=mystyle;
ods graphics on;
proc lifetest data=sasuser.fitness;
  time age;
  survival confband=all plots=(hwb);
run;
ods graphics off;
ods html close;
```

GRAPH PRODUCED



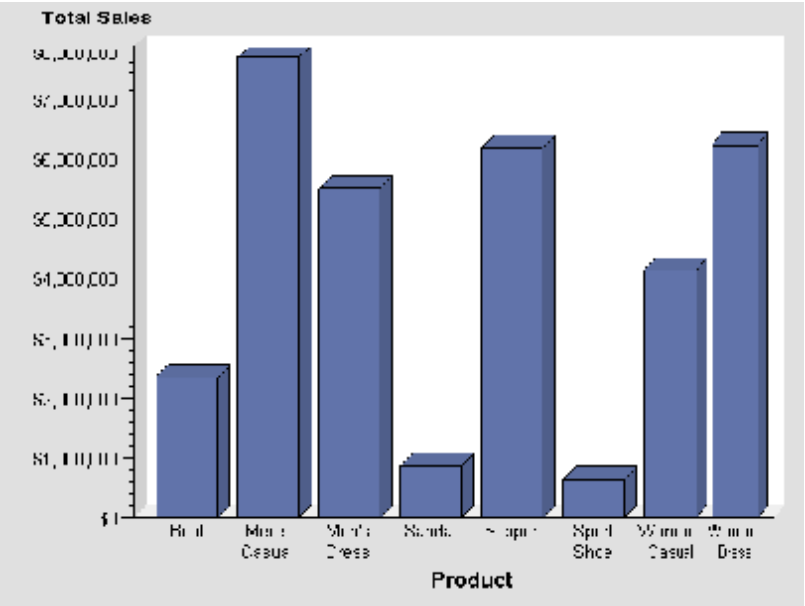
USING JAVA & ACTIVEX TO GENERATE STATIC GRAPHS

In SAS 9 there are 2 new graphic devices that can be used to create graphs called *actximg* & *javaimg*. These produce very good looking graphics but are not interactive graphics, as their siblings are (*ActiveX* & *Java*). The files produced are actually in PNG format and by default will be 640x480 pixels (VGA). Try them out and see how easy it is to produce excellent looking output. The graph produced below was only 5.6kb in size.

SAMPLE CODE

```
goptions device=actximg ;
/*goptions device=javaimg ;*/
ods html body='c:\test.html'
         gpath='c:\'
         (url=none) ;
proc gchart data=sashelp.shoes ;
  vbar3d product / sumvar=sales ;
run ;
ods html close ;
```

GRAPH PRODUCED USING ACTXIMG



MAKE COMBINED BAR CHART AND PLOT THE EASY WAY

In SAS 9 there is a new SAS/Graph procedure which can make a combined vertical bar chart with a plot line on it. It is great for comparing two exact and relative magnitudes of values. It only works with ODS using the *activex* or *actimgx* driver though.

SAS LOG

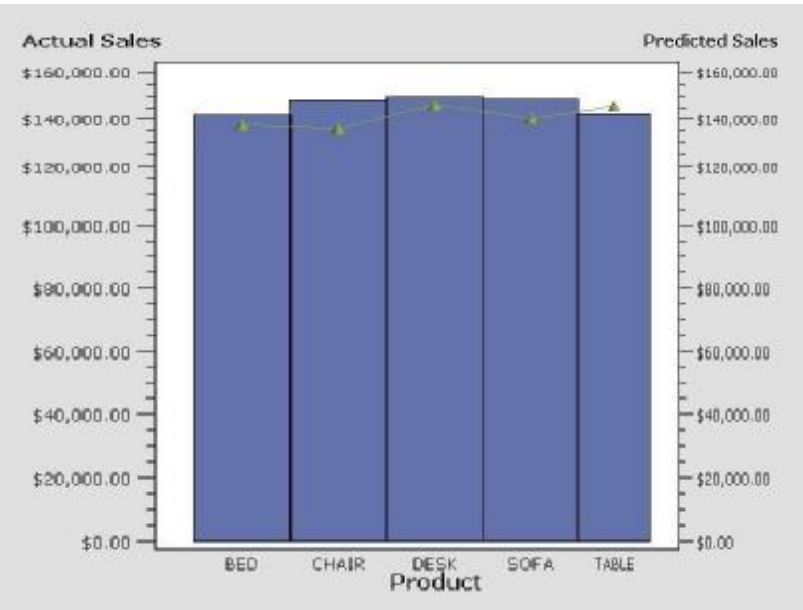
```
goptions reset=all device=activex ;

ods html file='c:\test.html' ;

proc gbarline data=sashelp.prdsale;
  bar product / sumvar=actual ;
  plot predict ;
run; quit;

ods html close;
```

SCREEN SHOT



PRODUCING MULTI-COLUMN REPORTS

From SAS 9 onwards there is a new ODS option called **columns=** which allows you to choose the number of columns to produce in your output. This works for some destinations including PDF, RTF & PS.
The following example shows how this can be used to send output to various destinations & select various numbers of columns by using macro variables.

SAS PROGRAM

```
%let dest=rtf; * pdf, ps or rtf ;
%let cols=2 ;
ods &dest columns=&cols file="c:\test.&dest" ;
goptions rotate=landscape ;
proc print data=sashelp.shoes ;
  var region stores sales ;
run ;
ods &dest close ;
```

1ST PAGE OF OUTPUT

Obs	Region	Stores	Sales
1	Africa	12	$29,761
2	Africa	4	$67,242
3	Africa	7	$76,793
4	Africa	10	$62,819
5	Africa	14	$68,641
6	Africa	4	$1,690
7	Africa	2	$51,541
8	Africa	12	$108,942
9	Africa	21	$21,297
10	Africa	4	$63,206
11	Africa	13	$123,743
12	Africa	25	$29,198
13	Africa	17	$64,891
14	Africa	9	$2,617
15	Africa	12	$90,648
16	Africa	20	$4,846
17	Africa	25	$360,209
18	Africa	5	$4,051
19	Africa	9	$10,532
20	Africa	9	$13,732
21	Africa	3	$2,259
22	Africa	14	$328,474
23	Africa	3	$14,095
24	Africa	14	$8,365
25	Africa	13	$17,337
26	Africa	12	$39,452
27	Africa	8	$5,172
28	Africa	4	$42,682
29	Africa	24	$19,282
30	Africa	1	$9,244
31	Africa	3	$18,053
32	Africa	18	$26,427
33	Africa	11	$43,452

Obs	Region	Stores	Sales
34	Africa	7	$2,521
35	Africa	1	$19,582
36	Africa	6	$48,031
37	Africa	16	$13,921
38	Africa	5	$57,691
39	Africa	10	$16,662
40	Africa	11	$52,807
41	Africa	10	$4,888
42	Africa	1	$17,919
43	Africa	3	$32,928
44	Africa	8	$6,081
45	Africa	3	$62,893
46	Africa	2	$29,582
47	Africa	9	$11,145
48	Africa	5	$19,146
49	Africa	2	$801
50	Africa	1	$8,467
51	Africa	25	$16,282
52	Africa	1	$8,587
53	Africa	19	$16,289
54	Africa	12	$34,955
55	Africa	10	$2,202
56	Africa	3	$28,515
57	Asia	1	$1,996
58	Asia	1	$3,033
59	Asia	1	$3,230
60	Asia	1	$3,019
61	Asia	1	$5,389
62	Asia	17	$60,712
63	Asia	1	$11,754
64	Asia	7	$116,333
65	Asia	3	$4,978
66	Asia	21	$149,013

Obs	Region	Stores	Sales
67	Asia	1	$937
68	Asia	2	$20,448

Obs	Region	Stores	Sales
69	Asia	7	$78,234
70	Asia	1	$1,155

PAGE X OF Y IN RTF & PDF

In SAS 9 if you want to add page numbers to your RTF output in the form of "page x of y" then you can use the in-line formatting by specifying an escape character and {pageof} – which will generate RTF code to display *x of y* on each page. {pageof} only works for the RTF destination.
If you want to do this for the PDF or Printer destination.then you need to use a slightly different technique. You can use the in-line style directive *{thispage}* which will give the current page number, and *{lastpage}* will give the last page number.

SAMPLE CODE FOR RTF

```
ods escapechar = '\';
title 'This document will have page x of y '
      j=r 'Page \{pageof}' ;
ods rtf file='c:\test.rtf' ;
proc print data=sashelp.prdsale;
run;
ods rtf close;
```

SAMPLE CODE FOR PDF

```
ods escapechar = '\';
title 'This document will have page x of y '
      j=r 'Page \{thispage} of \{lastpage}' ;
ods pdf file='c:\test.pdf' ;
proc print data=sashelp.prdsale;
run;
ods pdf close;
```

DISCOVERING ODS EVENTS, TO HELP MAKE TAGSETS

ODS is very flexible and provides PROC TEMPLATE which can be used to create your own custom tagsets. Then using the MARKUP destination you can use your tagset to create the required output. This is a hugely powerful and flexible facility, but often baffles those trying to use it. One common question is "what events should I define in my tagset?". Well the developer of the ODS tagset mechanism (Eric Gebhart at SAS) has provided 2 tagsets that can be used to display the events that occur when producing some ODS output.

EVENT_MAP

creates XML output that shows which events are being triggered and which variables are used by an event to send output from a SAS process to an output file. When you run a SAS process with EVENT_MAP, ODS writes XML markup to an output file that shows all event names and variable names as tags. The output helps you to create your own tagsets. e.g.

```
ods markup file='test.xml' tagset=event_map ;
proc print data=sashelp.class ; run ;
ods _all_ close ;
dm "wbrowse 'test.xml'" ;
```

SHORT_MAP

creates a subset of the XML output that is created by EVENT_MAP. e.g.

```
ods markup file='test.xml' tagset=short_map ;
proc print data=sashelp.class ; run ;
ods _all_ close ;
dm "wbrowse 'test.xml'" ;
```

MORE TIPS WORTH LOOKING AT

- PROC DOCUMENT
- PROC TEMPLATE enhancements
- LIBNAME libref SASEDOC
- ODS DECIMAL_ALIGN
- ODS CHTML
-

OTHER FEATURES

CREATE A PIVOT TABLE FROM SAS

A colleague and I were looking at the best way to automatically create a pivot table in EXCEL automatically from SAS. We considered solutions such as ODS with MSO XML directives, straight XML, DDE, and so on – but these were all very complex. He finally came up with the following simple method.
We use a SAS program to create a spreadsheet and then call a Visual Basic Script. The Visual Basic Script does the following:

- open the spreadsheet
- add a new sheet for pivot table
- create a pivot table using wizard
- set the fields to be used in the table

The SAS program could be extended to make a macro which creates the VBS file. This could then make it parameter driven to work for all data.

SAS PROGRAM

```
* create EXCEL spreadsheet ;
proc export data=sashelp.class
  outfile="c:\sas\class.xls"
  dbms=excel;
quit;
* call VB script to make the pivot table ;
data _null_;
  x 'c:\sas\pivot.vbs';
run;
```

VB SCRIPT PROGRAM

```
Set XL = CreateObject("Excel.Application")
XL.Visible=True
XL.Workbooks.Open "c:\sas\class.xls"
Xllastcell= xl.cells.specialcells(11).address
XL.Sheets.Add.name = "PivotTable"
xldata="class"
XL.Sheets(xldata).select
XL.ActiveSheet.PivotTableWizard SourceType=xlDatabase,XL.Range("A1" & ":" &
xllastcell),"Pivottable!R1C1",xldata

XL.ActiveSheet.PivotTables(xldata).PivotFields("Name").Orientation = 1
XL.ActiveSheet.PivotTables(xldata).PivotFields("Age").Orientation = 1
XL.ActiveSheet.PivotTables(xldata).PivotFields("Sex").Orientation = 1
XL.ActiveSheet.PivotTables(xldata).PivotFields("Height").Orientation = 4
XL.ActiveWorkbook.ShowPivotTableFieldList = False
```

BROWSING METATDATA FROM SAS

From SAS 9 you can browse through metadata via the Explorer Pane (if you have metadata via a metadata sever defined). To enable this follow these steps:

1. Start SAS 9
2. Click on the Explorer pane to activate it or use the menu VIEW/CONTENTS ONLY.
3. Now select TOOLS/OPTIONS/EXPLORER
4. Under the GENERAL tab select the METADATA SERVERS check box, and then click on OK.
5. There will now be a new METADATA SERVERS icon in the Explorer pane. Click on this to show its contents. There will be nothing there yet.
6. Right click in the explorer pane and select NEW. This allows you to define a connection to a metadata server – which may be on your machine, or elsewhere. Define this and any others you may want.
7. Now you can continue to drill down through the metadata to examine it.

EXPORTING DATA TO MICROSOFT ACCESS VIA XML

If you have SAS/Access for PC File formats, then you can use that to send you data between SAS & Access. However you can use an alternative method from Base SAS to export data to Access. You can use the XML engine on the libname statement, along with xmltype=msaccess, which creates XML specifically for Access. By also specifying "xmlmeta=schemadata" you will get the variables attributes included in the XML and imported by access too.

Finally you import the XML file from within Access by using "File", "Get external data".

Note: index creation is not supported.

SAMPLE PROGRAM

```
libname access xml 'd:\test.xml'
         xmltype=msaccess xmlmeta=schemadata;
data test(index=(year month)) ;
  set sashelp.retail ;
run ;
proc copy in=work out=access index=yes ;
  select test ;
run ;
libname access ;
```

LOG

```
1    libname access xml 'd:\test.xml'
2             xmltype=msaccess xmlmeta=schemadata;
NOTE: Libref ACCESS was successfully assigned as follows:
      Engine:        XML
      Physical Name: d:\test.xml
3    data test(index=(year month)) ;
4      set sashelp.retail ;
5    run ;

NOTE: There were 58 observations read from the data set SASHELP.RETAIL.
NOTE: The data set WORK.TEST has 58 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.68 seconds
      cpu time            0.06 seconds


6    proc copy in=work out=access index=yes ;
7      select test ;
8    run ;

NOTE: Copying WORK.TEST to ACCESS.TEST (memtype=DATA).
NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
WARNING: Indexes for ACCESS.TEST.DATA cannot be created.
WARNING: Engine XML does not support index create operations.
NOTE: There were 58 observations read from the data set WORK.TEST.
NOTE: The data set ACCESS.TEST has 58 observations and 5 variables.
NOTE: PROCEDURE COPY used (Total process time):
      real time           1.54 seconds
      cpu time            0.23 seconds


9    libname access ;
NOTE: Libref ACCESS has been deassigned.
```

EXPORT TO EXCEL VIA LIBNAME

In SAS 9 you can now read & write EXCEL spreadsheets from data steps and procedures. This greatly simplifies exporting data to EXCEL. The following log shows how I created an EXCEL file called test.xls, with a sheet called CLASS. The sheet lists variable names in the first row, followed by values on subsequent rows.

Note the error message demonstrating that there are some limitations with the EXCEL engine, preventing me from

overwriting a sheet once I have created it. Following that you can see that I can create more sheets within the file.

SAS LOG

```
41   libname out excel 'c:\test.xls' ;
NOTE: Libref OUT was successfully assigned as follows:
      Engine:        EXCEL
      Physical Name: c:\test.xls
42   data out.class ; set sashelp.class ; run ;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set OUT.class has 19 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.02 seconds


43   * try to replace dataset ;
44   data out.class ; set sashelp.class ; run ;

ERROR: The MS Excel table class has been opened for OUTPUT. This table already
       exists, or there is a name conflict with an existing object. This table
       will not be replaced. This engine does not support the REPLACE option.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


45   * make a new dataset ;
46   data out.shoes ; set sashelp.shoes ; run ;

NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
NOTE: There were 395 observations read from the data set SASHELP.SHOES.
NOTE: The data set OUT.shoes has 395 observations and 7 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.02 seconds


47   * free it so we can read the spreadsheet from EXCEL ;
48   libname out ;
NOTE: Libref OUT has been deassigned.
```

USING THE ZIP ENGINE TO READ ZIP FILES

There is a currently undocumented filename engine available in SAS 9 that can be used to read from compressed ZIP files directly. You simply specify the engine “SASZIPAM” on a filename statement, and when referring to it you must specify the file within it that you wish to read. In the example below “tomcat.zip” contains a number of files. I want to read “tomat.log” and therefore specify “in(tomcat.log)”, where “in” is the libref and “tomcat.log” is the file I will read from the zip file.

SAMPLE SAS PROGRAM

```
filename in saszipam 'c:\tomcat.zip';
data _null_;
  infile in(tomcat.log);
  input ;
  put _infile_;
  if _n_>10 then
    stop ;
run;
```

SAMPLE SAS LOG

```
4    filename in saszipam 'c:\tomcat.zip';
5    data _null_;
6      infile in(tomcat.log);
7      input ;
8      put _infile_;
9      if _n_>10 then
10       stop ;
11   run;

NOTE: The infile library IN is:
      Stream=c:\tomcat.zip

NOTE: The infile IN(tomcat.log) is:
      File Name=tomcat.log,
      Compressed Size=1894,Uncompressed Size=15793,
      Compression Level=-1,Clear Text=Yes

Using CATALINA_BASE:   C:\Tomcat4.1
Using CATALINA_HOME:   C:\Tomcat4.1
Using CATALINA_TMPDIR: C:\Tomcat4.1\temp
Using JAVA_HOME:       C:\j2sdk1.4.2_04
Using Security Manager
Starting service Tomcat-Standalone
Apache Tomcat/4.1.18
INFO: System properties were read from a file.
This discovery service deployment looks up remote services.

----------------------------------------------------------------------
NOTE: A total of 11 records were read from the infile library IN.
NOTE: 11 records were read from the infile IN(tomcat.log).
NOTE: DATA statement used (Total process time):
      real time           0.78 seconds
      cpu time            0.06 seconds
```

EXPORTING TO EXCEL

If you have "SAS/Access for PC File Formats" licensed then it is now very easy to import and export data between SAS and Microsoft EXCEL. You can reference EXCEL spreadsheets directly with a libname statement – no engine is required. You can then refer to a spreadsheet using the libref and a worksheet by using a "dataset name". For instance in the example below, nice.test refers to the spreadsheet 'c:\nice.xls' and within it the worksheet called 'test'.

SAS LOG

```
11   libname nice 'c:\nice.xls' ;
NOTE: Libref NICE was successfully assigned as follows:
      Engine:        EXCEL
      Physical Name: c:\nice.xls
12   data nice.test ;
13     set sashelp.class ;
14   run ;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set NICE.test has 19 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.02 seconds


15   libname nice ;
```

BONUS QUICK TIPS

SAS WEB INFRASTRUCTURE KIT

If you have SAS Integration Technologies, then make sure you know what this is since it is **the** most useful thing to come from SAS in the last 10 years (in my opinion). Why? This is what makes it possible to write a SAS program and make it instantly accessible to be run via a web browser. Supports full security protocols. If you have **Enterprise Guide** you can even use a wizard to write your program, convert to a stored process and generate a web page to let users run it.

SOME MORE WORTH LOOKING AT

- Pipeline parallelism in SAS/Connect
- PROC TIMESERIES in SAS/ETS

CONCLUSION

SAS 9 has a huge amount of new features in it. You should go and work through the What's New documents in the SAS Documentation.

CONTACT INFORMATION

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

Phil Mason
Wood Street Consultants Ltd.
16 Wood Street
Wallingford, Oxfordshire, OX10 0AY, ENGLAND
E-mail: phil@woodstreet.org.uk
Web: www.woodstreet.org.uk