PhUSE 2006

Paper TS09

Analyse That : Getting Information Quickly About Your SAS Data

Steve Prust, Covance, Leeds, UK

ABSTRACT

This paper looks at the difficulties in getting to grips with data. A detailed look is made of how a single utility can be constructed to deal with the all the type of questions a data user is likely to ask when first encountering new data. The paper features many code extracts (the full code used to develop the example in this paper is available in a separate file).

INTRODUCTION

How many times have you been asked to take some data and "do something with it" ? I suspect that for many people this type of request is very common and accompanied by " and I need it, like, yesterday".

SAS is a great tool for analysing data. However, when presented with new and unfamiliar data the methods that SAS gives you for finding out about your data are less than ergonomic. "What variables are there ?", "how many rows ?", "when was it created", "is it indexed and/or compressed", "are the variables character or numeric", "what formats ?", "what does the format look like ?", "what values are there ?", "is anything missing" etc. etc. Faced with large datasets the work involved in running PROC CONTENTS, PROC FORMAT, FSVIEW, maybe PROC FREQ, MEANS, or TABULATE can be at least time consuming and at worst error-prone.

In order to address this perceived ergonomic issue a SAS/AF utility – SQIT - was designed and written to bring together SAS's wealth of tools into one place. This paper details how the utility was made and includes all the significant code. The intention is that readers will be able to use this as a basis for constructing similar tools.

Note : code samples have been simplified for the purposes of this paper by removing items such as error processing.. It is recommended when producing utility software to consider how you want errors etc. to be trapped and reported. Also some of the processing regarding showing / hiding or relevant or otherwise screen components from the user has not been included.

OBJECTIVES

When we set out we wanted one place where we could view the following information:

- all the libraries and their datasets

For any dataset selected, we want to view

- dataset attributes and statistics
- a list of the variables and their attributes

For any variable selected, we want to view

- all unique values of character variables
- statistics for that variable
- the format information (if any)

Additionally we would want to sub-set the data using a where condition when viewing statistics and/or unique values.

Lastly, we wanted to be able to copy information so it can be pasted elsewhere.

CREATING THE UTILITY

The utility was created using SAS/AF the code has been tested in both versions 8.2 and 9.

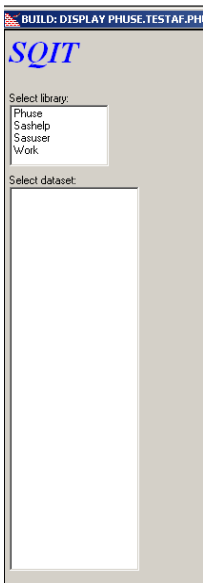
VIEWING THE LIBRARIES AND DATASETS

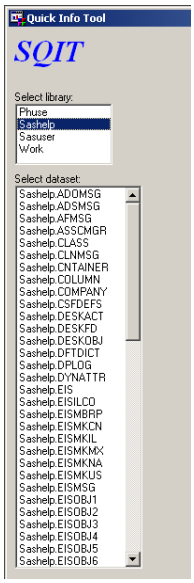
The means of doing this are two models : the library List Model and the Data Set List Model. These can be found on the SAS/AF component panel. They are – as their icons suggest - non-visual components and must be associated with visual components. In my example below I have associated both with listbox objects.

The creation sequence followed was :

1. create a Library List Model (named in the example “modLib”)
2. create a listbox (lbLibs) and amend the model property to “modLib”. The frame window now shows the current libraries in your SAS session
3. create a Data Set List Model component (named in the example “modDsn”).
4. For the Library property of modDsn set the Linkedto value to lbLibs.selecteditem
5. create a second listbox (lbDsns and amend the model property to “modDsn”.

Using the AF test button will change the build screen from something like the example below left to that on the right :





GETTING INFORMATION ABOUT THE SELECTED DATASET

Once a dataset is selected in the listbox it is very easy to find information using standard SAS functions. After the information is found it is placed in an SCL list prior to being loaded into a text pad area.

Code to do this could be :

```
seldsid = open(lbDsns.selectedItem,'IN');  /* open SAS dataset */
nvars   = attrn(seldsid,'NVARS');
nobs    = attrn(seldsid,'NOBS');
crdate = put(attrn(seldsid,'CRDTE'),datetime16.);
modate = put(attrn(seldsid,'MODTE'),datetime16.);
engine = attrc(seldsid,'ENGINE');
sortby = attrc(seldsid,'SORTEDBY');
label  = attrc(seldsid,'LABEL');
rc = close(seldsid);

rc = dellist(inflist);   /* delete any existing SCL list */
inflist = makelist();    /* create a blank SCL list */
rc = insertc(inflist,'Engine    : ' || engine,1);
rc = insertc(inflist,'Created   : ' || crdate,2);
rc = insertc(inflist,'Modified  : ' || modate,3);
rc = insertc(inflist,'Obs       : ' || put(nobs,best8.),4);
```

```
rc = insertc(inflist,'Vars      : ' || put(nvars,best8.),5);
rc = insertc(inflist,'Sorted by : ' || sortby,6);
rc = insertc(inflist,'Label     : ' || label,7);

tpDsnInfo.text = copylist(inflist);
tpDsnInfo.BorderTitle = 'Info for ' || seldsn;
```

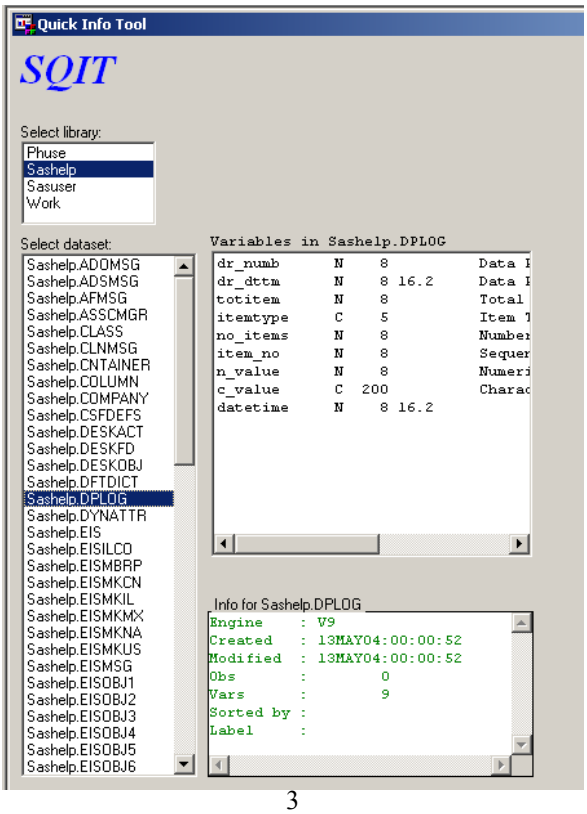
GETTING INFORMATION ABOUT VARIABLES IN THE DATASET

It would be possible to use both the Variable List Model and the Variable Values model at this point but the are perhaps a little restrictive. The following code allows more information to be displayed in a fairly readable form :

```
seldsid = open(lbDsns.selectedItem,'IN');  /* open SAS dataset */
rc=dellist(varlist);
varlist=makelist();
nvars=attrn(seldsid,'NVARS');
do j = 1 to nvars;
  selvarn = varname(seldsid,j);
  selvart = vartype(seldsid,j);
  selvarl = varlen(seldsid,j);  /* can get other details using VARxxx functions */

  /* try and align columns but can't practically cater for very long names */
  if length(selvarn) > 12 then
    varstr = trim(selvarn) || ' ' || selvart || ' ' ||
             put(selvarl,4.) ||  ' ' || put(selvarf,$8.) || ' ' || selvarb;
  else do;
    repnum = max( 12 - length(selvarn) , 1);
    varstr = trim(selvarn) || repeat(' ',repnum) ||  selvart || ' ' ||
             put(selvarl,4.) ||  ' ' || put(selvarf,$8.) || ' ' || selvarb;
  end;
  rc = insertc(varlist, varstr, j);
end;
lbVarDets.title = 'Variables in ' || seldsn;
lbVarDets.items = copylist(varlist);
```

adding this code and appropriate components into the previous example gives :

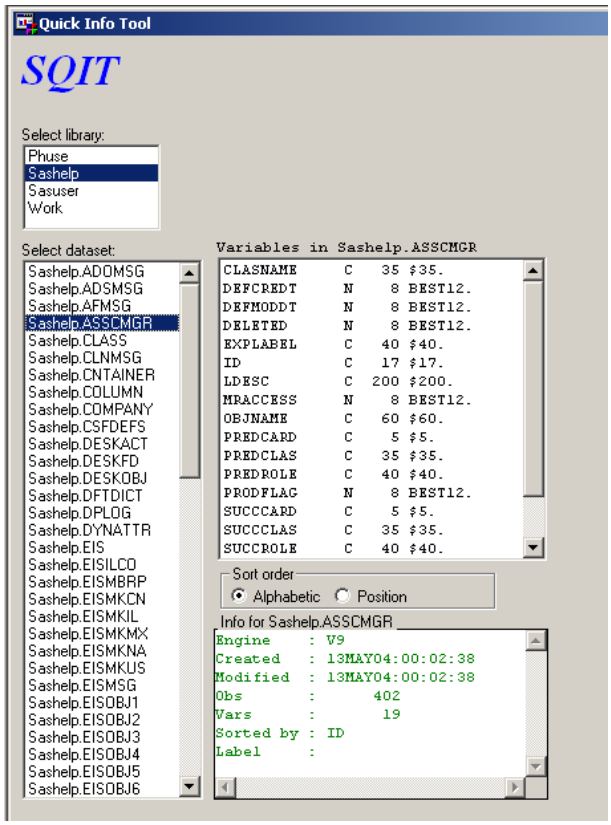


(note the new visual components have been formatted using monospace fonts for easier reading in columns and green is used to indicate information that cannot be “clicked” for further information)

This is of course similar to what is available via PROC CONTENTS (except that the information is sorted by variable order rather than alphabetically). It can also be tweaked to show any information that is particularly relevant to your site (see ATTRN ATTRC and VARxxx functions). You could even add the ability to sort the variable list by either variable position or name. For example, adding the following code and the relevant radio button could give :

```
if rbVarSort.selectedItem = 'Alphabetic' then
        rc = sortlist(varlist);
```

With the following results :



GETTING INFORMATION ABOUT A SINGLE VARIABLE

We can now add functionality to enable the user to discover information about any variable. This will be triggered by a variable line being clicked in the listbox (named in this example, lbVarDets).

Because the data returned by SelectedItem will include the variable type, format, and label we need to extract the relevant information first, then open the dataset and find the variable position (needed for future function calls).

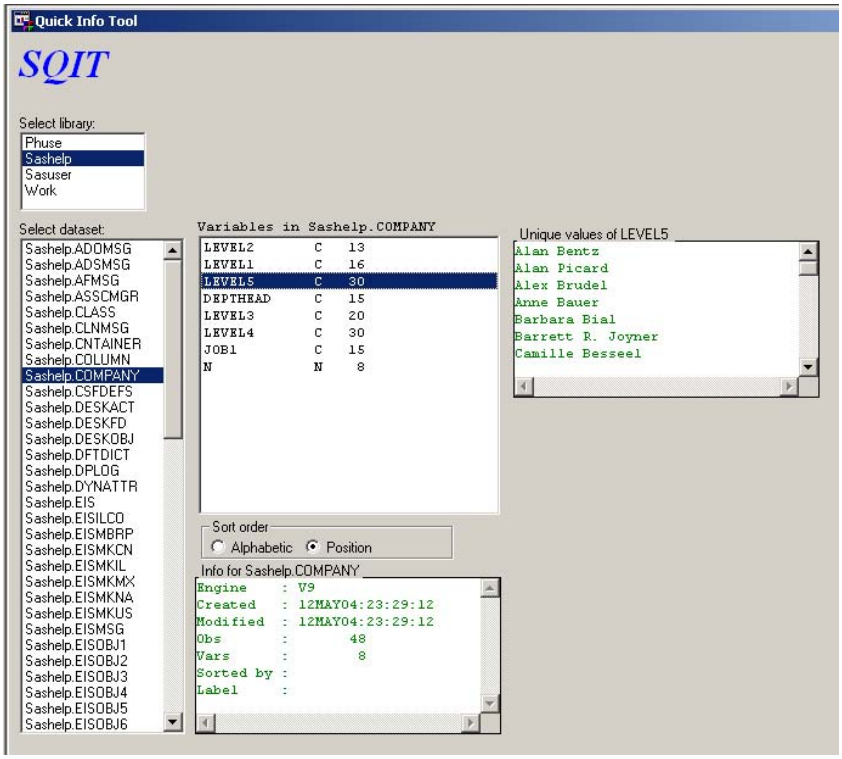
```
selivar = substr( lbVarDets.selectedItem,1, index( lbVarDets.selectedItem,' ') );
seldsid = open(seldsn,'IN');
selinum = varnum(seldsid,selivar);
```

Function LVARLEVEL

Lvarlevel is a great function returning all the unique values for a variable. Invoke it, sort the output and place the results in a textpad area :

```
rc = dellist(ValList);
ValList = makelist();
nunique=0;
rc = lvarlevel(seldsid,selivar,nunique,ValList);
rc = sortlist(ValList,'IG');
tpUniVal.BorderTitle = 'Unique values of ' || selivar;
tpUniVal.text  = copylist(ValList);
```

The resulting screen now looks like this :



VARIABLE STATISTICS

First we find the variable type before doing different things for character and numeric variables :

```
if vartype(seldsid,selinum) = 'C' then do;
```

For character variables the sensible statistics are number of missing values, number of non-missing values, number of unique values. These are obtained by the function VARSTAT:

```
rc = dellist(StatList);
StatList = makelist();
rcn = varstat(seldsid,selivar,'N NMISS NUNIQUE',ivn,ivnm,ivnu);
rc = insertc(statlist,'N       : ' || put(ivn,best8.),1);
rc = insertc(statlist,'NMiss   : ' || put(ivnm,best8.),2);
rc = insertc(statlist,'NUnique: ' || put(ivnu,best8.),3);
```

for numeric values we can extend the statistics obtained :

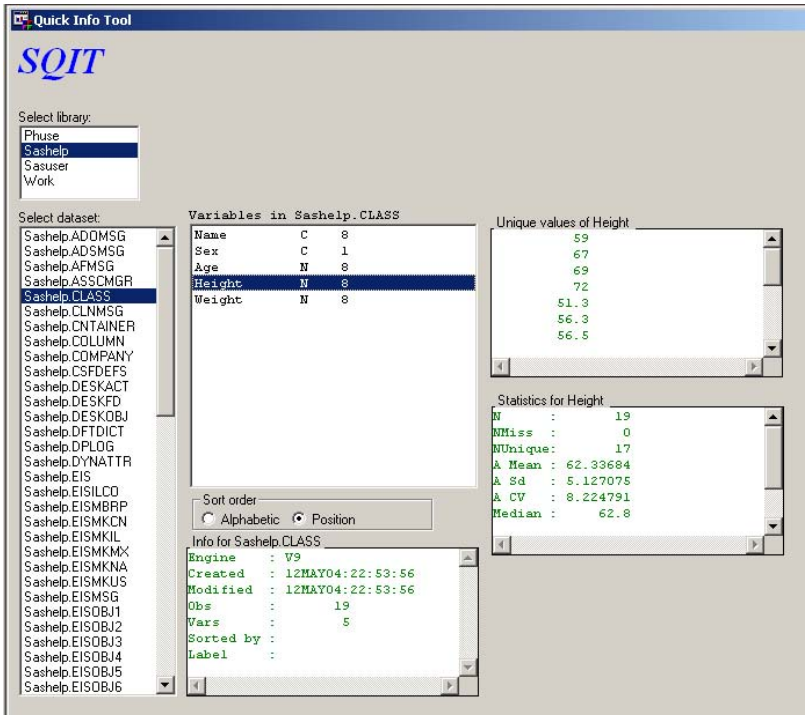
```
rcn = varstat(seldsid,selivar,
          'N MIN MAX MEAN MEDIAN STD STDERR CV NMISS NUNIQUE',
          ivn,ivmin,ivmax,ivmean,ivmed,ivstd,ivserr,ivcv,ivnm,ivnu);
rc = insertc(statlist,'N       : ' || put(ivn,best8.),1);
rc = insertc(statlist,'NMiss   : ' || put(ivnm,best8.),-1);
```

```
rc = insertc(statlist,'NUnique: ' || put(ivnu,best8.),-1);
rc = insertc(statlist,'A Mean : ' || put(ivmean,best8.),-1);
rc = insertc(statlist,'A Sd   : ' || put(ivstd,best8.),-1);
rc = insertc(statlist,'A CV   : ' || put(ivcv,best8.),-1);
rc = insertc(statlist,'Median : ' || put(ivmed,best8.),-1);
rc = insertc(statlist,'Min    : ' || put(ivmin,best8.),-1);
rc = insertc(statlist,'Max    : ' || put(ivmax,best8.),-1);
rc = insertc(statlist,'Stderr : ' || put(ivserr,best8.),-1);
```

before copying the information into a textpad control :

```
tpVarStats.BorderTitle = 'Statistics for ' || selivar;
tpVarStats.text = copylist(StatList);
```

This gives a screen as follows :



The function LVARSTATS doesn't include all the statistics that might be needed. If we need something such as a geometric mean then we need to work a little harder :

The method used is :

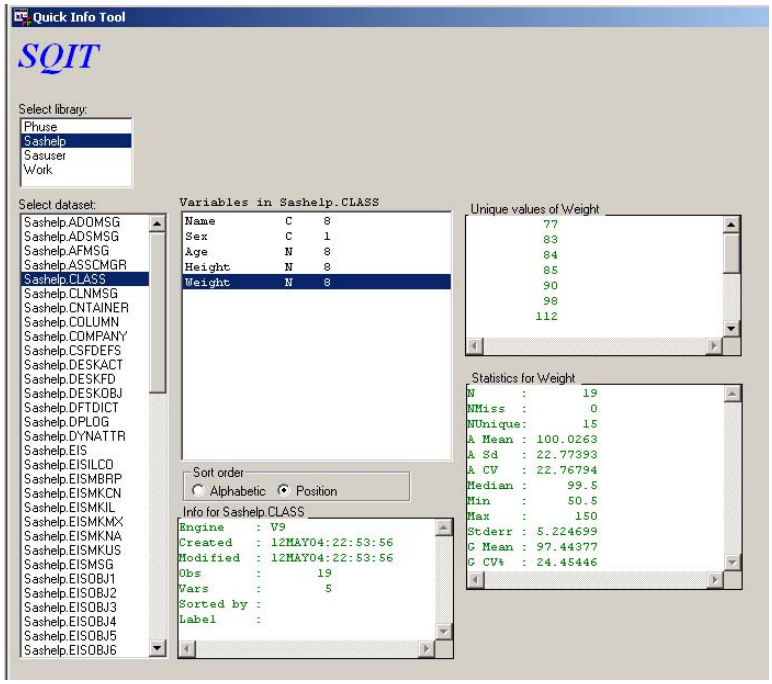
1. copy the selected dataset
2. edit the new dataset, adding a new variable which is set the log value of the selected variable
3. run lvarstats on the new dataset to get a mean and standard deviation of the new variable
4. calculate the geometric mean etc.
5. tidy up

The code is as follows :

```
tempdsn = 'work._sqitgeo';
rc = copy(seldsn,tempdsn);
tid = open(tempdsn,'US');
tcol = varnum(tid,selivar);
frc = fetch(tid);
do while(frc=0);
  gvar = getvarn(tid,tcol);
  lgvar = log(gvar);
  call putvarn(tid,tcol,lgvar);
```

```
      rc = update(tid);
      frc = fetch(tid);
    end;
    rcn = varstat(tid,selivar,'MEAN STD',gmean,gstd);
    if rcn = 0 then do;
      egmean = exp(gmean);
      gcv = (sqrt(exp(gstd*gstd)-1))*100;
      rc = insertc(statlist,'G Mean : ' ||put(egmean,best8.),-1);
      rc = insertc(statlist,'G CV%  : ' ||put(gcv,best8.),-1);
    end;
    rc = close(tid);
    rc = delete(tempdsn);
```

The resulting screen looks like :



Having got the ability to get statistics on the complete dataset it seems useful to extend this facility to a subset of the data. The is can be done by adding a facility for a where condition to be applied before running the unique values / statistics. The simplest method for obtaining a where condition is to provide a text box where the condition is typed in and a button to apply / remove the condition. Of course this will make it easy for the user to make mistakes – error checking will be necessary.

The code changes are fairly spread out but not lengthy:

1. a section to control the application / removal of the where condition :

```
pbApplyWhere:
  if wherec = 'OFF' then do;
    pbApplyWhere.label      = 'Remove WHERE condition';
    teWhereCond.BorderTitle = 'Where condition (applied)';
    wherec                  = 'ON';
  end;
  else do;
    pbApplyWhere.label      = 'Apply WHERE condition';
    teWhereCond.BorderTitle = 'Where condition';
    wherec                  = 'OFF';
  end;
  link getVardets;  /* re-apply variable inf. routine (in a subroutine now) */
return;
```

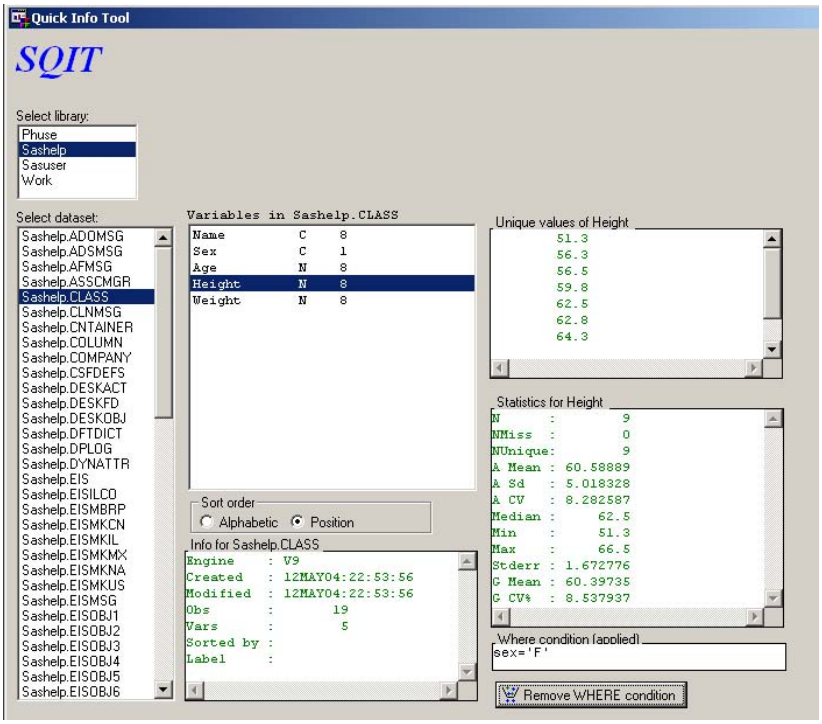
2. When datasets are being opened the addition of logic e.g. :

```
if wherec = 'ON' then do;
        txtwhere =  ' (where=(' || teWhereCond.text || '))';
  seldsid = open(seldsn || txtwhere ,'IN');
end;
else
  seldsid = open(seldsn,'IN');
```

3. a check to make sure the dataset opened correctly :

```
if seldsid = 0 then
  rc=insertc(StatList,'Unable to open dsn, rc=' || put(seldsid,best8.),1);
else do;
```

This then gives the following :



FORMAT INFORMATION

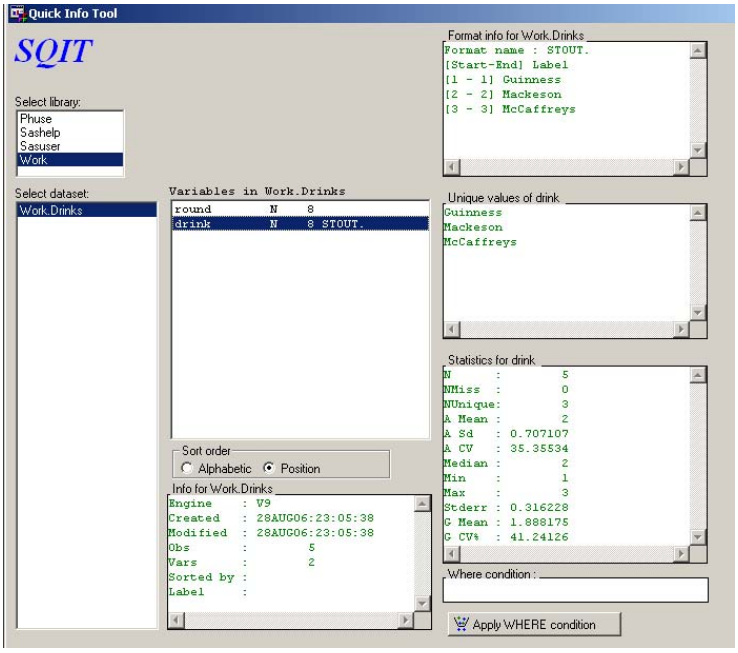
Finding out format information for a variable can be highly valuable. Against this it is a slightly awkward process that involves determining whether a format is built-in on not, what the system option FMTSEARCH is set to (this can be further complicated by how the LIBRARY libref can be used), and retrieving the information. The code could look something like :

```
selvarf = varfmt(seldsid,selinum);
rc = insertc(fmtlist,'Format name : ' || selvarf ,1);
if not missing(selvarf) then do;
  tempdsn2 = '_sqitfmt';
  fmtname = substr(selvarf,1,length(selvarf)-1);
  last = substr(fmtname,length(fmtname),1);
  do while (index('.01234567890',last));
    fmtname = substr(fmtname,1,length(fmtname)-1);
    last = substr(fmtname,length(fmtname),1);
  end;
  if not missing(fmtname)
  and fmtname not in ('$', 'BEST', 'DATETIME', 'DATE', 'TIME', 'COMMA') then do;
```

```
      fmtlibs = optgetc('FMTSEARCH');   /* may need extra logic here */
      SUBMIT CONTINUE;
        libname __tempf &fmtlibs;
        proc format library=__tempf cntlout=&tempdsn2;
          select &fmtname;
        run;
        libname __tempf;
      ENDSUBMIT;
      tid = open(tempdsn2,'I');
      if tid then do;
        stacol = varnum(tid,'start');
        endcol = varnum(tid,'end');
        labcol = varnum(tid,'label');
        rc = insertc(fmtlist,'[Start-End] Label',-1);
        frc = fetch(tid);
        do while(frc=0);
          fstart = getvarc(tid,stacol);
          fend = getvarc(tid,endcol);
          flabel = getvarc(tid,labcol);
          rc = insertc(fmtlist,
              '['|| compress(fstart) ||' - '|| compress(fend) ||'] '|| flabel,-1);
          frc = fetch(tid);
        end;
        rc = close(tid);
        rc = delete(tempdsn2);
      end;
      else
        rc = insertc(fmtlist,'Unable to retrieve format details', 2);
    end;
  end;

  tpFormat.text = copylist(fmtlist);
  tpFormat.BorderTitle = 'Format info for ' || seldsn;
```

And the results look like this :



PASTE BUFFER

The information available in SQIT could be useful in other windows / documents. A copy facility can be added using a Paste Buffer and one or more buttons. The paste buffer is not found on the standard component palette but can be found in SAS Explorer window in the SASHELP.CLASSES and dragged onto the frame.

The code for the button is very simple :

```
pbCopyUnival:
  pasteBuffer1.text = copylist(tpUniVal.text);
return;
```

FURTHER FACILITIES

SQIT is at this point a useful tool, able to do all of the objectives set out. It could be taken even further with the addition of :

- Better interface e.g. hiding / revealing information, cleaning boxes when higher level items change etc.
- Code generation : the paste facility could be extended to create code such as SELECT statements from the unique values list.
- A method of linking directly to viewing a dataset
- Automatic update of the library and dataset lists when there are additions / deletions (we currently use a button to refresh information as needed).
- Print Facilities

CONCLUSION

SQIT has been used for several years in-house and has proved to be highly useful and low maintenance.

Is SQIT an alternative to the SAS Explorer window ? Not quite – SAS Explorer supports all types of data (except ODS item item stores, graphics templates) SQIT is best used as a tool just for looking at data.

Please note that all the code used to create this example is available in a separate file on the same site as this paper

CONTACT INFORMATION

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

Steve Prust
Covance
Springfield House,
Hyde Street,
Leeds, UK
LS2 9NG
Email:stephen.prust@covance.com