All Things to All People: A User-Defined Report in SAS/AF®
Aileen D. Bennett, U.S. Census Bureau

ABSTRACT
You have built a SAS/AF application to maintain metadata on the survey data you process. The metadata are in a set of SAS data sets which can be joined with each other on various keys, much like a relational database. The users can add, delete, and browse the data. They also want to produce reports. There are some predefined reports, but they also want a user-defined report which can have any data from any of the data sets. They want the capability to choose data sets, specify a WHERE clause for each one, choose variables and a sort order, and store the report definition. You have to program the report to join the data sets properly and inform the user if he selected an unworkable combination of data sets. How? Use SCL lists and work from the data set relationship diagram.

INTRODUCTION
The Census Bureau’s American Community Survey Data Processing and Tabulation teams have tried to make the survey edit and table generation processes as data-driven as possible. To that end, they designed a data dictionary consisting of 14 SAS data sets (“entities”) which contain information about all the survey variables, including their formats, values, and history, all the tables, including the definition of each cell, and all the edits. Each data set can be joined with at least one other data set on such keys as variable name and table ID number, and the data sets as a group form a kind of relational database. (See the diagram on page 6.) The entities are CELRANGE, DICT, EDITVARS, FMTDEFS, HEDITALS, PEDITALS, ROOTDEFS, TABLES, TBLCELLS, TBLVARS, UNIVARS, UNIVERSE, and VARHIST.

To make maintenance of these data easier, a Version 8 SAS/AF application was built with browse, edit, and report capabilities. In addition to pre-defined reports, a user-defined report was requested, where the user could select any variables from any legal combination of entities, specify WHERE clauses and sort order, and store and retrieve the specifications once the report was run.

Programming such a report is not straightforward because the entities must be joined in different ways according to their keys and you must account for all the different combinations of entities which may be selected.

SCREEN DESIGN
The SAS/AF FRAME entry for the report is a tab layout object with three tabs: one for selecting entities and entering WHERE clauses, one for selecting variables for the header and body of the report, and one for running, printing, and saving the report and its definition which displays the user’s selections. The design is based on the example given in Stanley, Chapter 5.

INIT SECTION
The INIT section declares variables and arrays and creates the SCL lists varlist and validlist. Validlist is filled with all legal entity combinations listed in the file rel.dat. An SCL variable containing the libref is created for use with the data sets DICT, EDITVARS, TBLCELLS, TBLVARS, and UNIVERSE, because they may be joined with other entities more than once, and the two-level name will be used only for the first join. Finally, the first tab on the tab layout object is made active.

INIT:
length keepvars vars $ 500 t1-t10 $100;
declare char text sortv1-sortv4 headvars

TAB 1 – SELECT AND SUBSET ENTITIES
The select and subset tab is displayed first. The user can click on the entity name or check box and specify a WHERE clause to limit the selection. The SCL code for this tab clears previous selections and resets SCL variables.
TAB 2 – SELECT VARIABLES

Once entities are selected and WHERE clauses entered, the user clicks on the “Select Variables” tab. The code associated with this tab checks the list of selected entities against the list of valid combinations in the SCL list called validlist. The combination of UNIVERSE and ROOTDEFS would not be legal, for example, unless DICT was also selected, because there is no common variable to join them with (see diagram). If the combination is illegal, a message box is displayed to inform the user, and the “Select and Subset Entities” tab is redisplayed. If the combination is legal, the variables from the selected entities are put into the SCL list varlist, which is sorted and has duplicate variable names removed. The variable list is displayed in two list boxes, along with a list of selected entities. The user clicks on the variables he wants as “header” variables (BY variables) and other variables to appear in the body of the report. He must also specify at least one and up to four variables by which the data will be sorted. The SCL code creates a title for each entity selected, containing the entity name and the WHERE clause, if any, and variable lists.

```scl
define subroutine select_variables
    ;clear the titles
    t2 = ''; t3 = ''; t4 = ''; t5 = ''; t6 = ''; t7 = ''; t8 = ''; t9 = ''; t10 = '';
    ;clear the sort selections
    ;clear the variable lists
    headvars = ''; keepvars = ''; vars = ''; titlecount = 2;
    ;reset the title count
    titlecount + 1;
    ;string entity names together for search against list of valid combinations
    do i = 1 to listlen(entitylist);
        cur_entity = getitemc(entitylist,i);
        select (length(cur_entity));
        when (4) pad = '     ';
        when (5) pad = '    ';
        when (6) pad = '   ';
        otherwise pad = ' ';
        end;
        entities = entities||cur_entity||pad;
done;
    ;check if selected combination is legal
    rc = clearlist(varlist);
    msglist = makelist();
    if listlen(entitylist) > 1 &
        not(searchc(validlist,entities)) then do;
        rc = clearlist(varlist);
        msglist = makelist();
        msglist = insertc(msglist,'You have selected an invalid combination of entities.');
        msglist = insertc(msglist,'Please make another selection.',-1);
        text = messagebox(msglist);
        call notify('tab','_setActiveTab',1);
    end;
    rc = sortlist(varlist,'nodup');
    Entities_Tab2.text = entitylist;
    Headerbox.items = varlist;
    Varbox.items = varlist;
end;
```

The GETVARS section, which is linked for each entity chosen, opens the entity data set and adds the names of its variables to the SCL list varlist:

```scl
define subroutine getvars
    ;get variable names from data set and add to list
    do j = 1 to attrn(dsid,'nvars');
        rc = insertc(varlist,varname(dsid,j),-1);
    end;
    rc = close(dsid);
    return;
end;
```

TAB 3 – RUN REPORT

After selecting entities, variables, and sort order, the user then clicks on the third tab to run the report. The SCL code checks to see that all required choices have been made; if not, the user is directed to the appropriate selection tab with a message. The header and other variables are combined into another SCL list called keeplist. The third tab is displayed, with list boxes containing the user’s selections and buttons for running and saving the report and its output.

```scl
if selected_tab=3 then do; *Run Report;
    ;get where clause
    if celrange_where ^= '' then do;
        subsetlist = insertc(subsetlist,celrange_where,-1);
        titles[titlecount] = 'title'||left(put(titlecount,2.))||'
        CELRANGE, ''||celrange_where||''';
    end;
    else titles[titlecount] = 'title'||left(put(titlecount,2.))||'
        "CELRANGE"; titlecount + 1;
end;
```
if listlen(entitylist) < 1 then do;
*No entities selected;
    _msg = 'Select Entities';
call notify('tab', '_setActiveTab', 1);
return;
end;

headlist = makelist();
if listlen(Headerbox.Selecteditems) > 0 then do; *make list of header variables;
    do i = 1 to listlen(Headerbox.Selecteditems);
        headlist = insertc(headlist, getitemc(Headerbox.Selecteditems, i), -1);
    end;

keeplist = makelist();
if listlen(Varbox.Selecteditems) > 0 then do; *make list of other variables;
    do i = 1 to listlen(Varbox.Selecteditems);
        keeplist = insertc(keeplist, getitemc(Varbox.Selecteditems, i), -1);
    end;
else do; *No variables selected;
    _msg = 'Select Variables';
call notify('tab', '_setActiveTab', 2);
return;
end;

do i = 1 to listlen(keeplist);
    vars = vars || getitemc(keeplist, i);
end;
keeplist = copylist(headlist, 'n', keeplist);
/*sort list to eliminate duplicates*/
rc = sortlist(keeplist, 'nodup');
do i = 1 to listlen(keeplist);
    keepvars = keepvars || getitemc(keeplist, i);
end;

ŘRUNNING THE REPORT
When the run report button is clicked, the report code is built and submitted. If only one entity was chosen, a simple Proc Sort
and Proc Print are run, followed by a check on whether the subset has resulted in no observations being printed. A message
is displayed if there are no observations.

RUNRPT:
/*build where clauses*/
if celrange_where = '' then celrange_where = '(where=(' || celrange_where || '))';
   (repeat for the other entities)
if varhist_where = '' then varhist_where = '(where=(' || varhist_where || '))';
/*make data set names two-level*/
dcflag = 'dd.'; evflag = 'dd.';
tcflag = 'dd.'; tbflag = 'dd.';
tvflag = 'dd.'; unflag = 'dd.';
if listlen(keeplist) = 1 then do; *only one
    entity selected;
    wherestr = celrange_where || dict_where || edits_where || editvars_where || fmtdefs_where || rootdefs_where || tblcells_where || tblvars_where || univars_where || universe_where || varhist_where;
    ent1 = getitemc(keeplist, 1);
    submit continue;
    proc sort data=dd.&ent1 &wherestr
        out=outdata;
        by &sortv1 &sortv2 &sortv3 &sortv4;
        run;
    proc print;
        by &headvars notsorted;
        var &vars;
        title 'ACS Data Dictionary';
        &t2;
        run;
sendsubmit;
    dsid = open('outdata', 'i');
    if attrn(dsid, 'nobs') = 0 then do;
    return;
end;

Entitys_Tab3.text = entitylist;
Header.text = headlist;
Selvars.text = keeplist;
Sort.text = sortlist;
Subsets.text = subsetlist;
end; *Active Tab = 3 (Run Report);
return;
msglist = makelist();
msglist = insertc(msglist,'There were no observations in the subset.');
text = messagebox(msglist);
end;
else do;
link footer;
call execmdci('output');
end;
rc = close(dsid);
end; *one entity;

When more than one entity is selected, they must be joined together. The report specification requested inner joins. The join order depends on which entities were selected. The entities at the extremities of the diagram are dealt with first. EDITS is joined with EDITVARS, creating an EDITVARS data set in the WORK directory. Any further joins involving EDITVARS will use this temporary data set, so the SCL variable evflag is changed to missing. Similarly, CELRANGE and TBLCELLS are joined into a new temporary TBLCELLS data set. The SCL variable lastds contains the name of the last data set created and is used in the final submit block.

else do; *join entities;
    if index(entities,'EDITS    EDITVARS') then do;
        submit;
        proc sql;
        create table editvars as
        select * from dd.editvars
        &editvars_where, dd.edits &edits_where
        where editvars.edit = edits.edit;
        endsubmit;
        evflag = '';
        lastds = 'editvars';
        end;
    if index(entities,'CELRANGE') &
       index(entities,'TBLCELLS') then do;
        submit;
        proc sql;
        create table tblcells as
        select * from dd.celrange
        &celrange_where, dd.tblcells &tblcells_where
        where celrange.tblid = tblcells.tblid
        and celrange.lineno = tblcells.lineno;
        endsubmit;
        tcflag = '';
        lastds = 'tblcells';
        end;

    The TABLES data set is dealt with next. It is joined first with TBLCELLS, then with TBLVARS, and finally with UNIVERSE. UNIVARS and UNIVERSE are joined together next.

    if index(entities,'TABLES   TBLCELLS') then do;
        submit;
        proc sql;
        create table tables as
        select * from &tbflag tblcells
        &tblcells_where, dd.tables &tables_where
        where tables.tblid = tblcells.tblid;
        endsubmit;
        tbflag = '';
        lastds = 'tables';
        end;
    if index(entities,'TABLES') &
       index(entities,'TBLVARS') then do;
        submit;
        proc sql;
        create table tables as
        select * from &tbflag tables
        &tables_where, dd.tblvars &tblvars_where
        where tables.tblid = tblvars.tblid;
        endsubmit;
        tbflag = '';
        lastds = 'tables';
        end;

    Finally, there is DICT. DICT is the central entity, the one which can be joined with the most other entities. It is joined first with UNIVERSE (by UNID), then ROOTDEFS (by ROOT), and then with the entities it can be joined with by VAR, in no particular order: VARHIST, FMTDEFS, EDITVARS, TBLVARS (if TBLVARS hasn’t already been joined with TABLES above), HEDITALS, and PEDITALS.

    if index(entities,'DICT') &
       index(entities,'UNIVERSE') then do;
        submit;
        proc sql;
        create table universe as
        select * from &unflag universe
        &universe_where, dd.univars &univars_where
        where universe.unid = univars.unid;
        endsubmit;
        unflag = '';
        lastds = 'universe';
        end;
    if index(entities,'DICT') &
       index(entities,'ROOTDEFS') then do;
        submit;
        proc sql;
        create table dict as
        select * from dd.rootdefs
        &rootdefs_where, &dcflag dict &dict_where
        where rootdefs.root = dd.root;
        endsubmit;
        dcflag = '';
        lastds = 'dict';
        end;
    if index(entities,'DICT') &
       index(entities,'VARHIST') then do;
        submit;
        proc sql;
        create table dict as
        select * from dd.varhist
        &varhist_where, &dcflag dict &dict_where
        where varhist.var = dd.var;
        endsubmit;
        dcflag = '';
        lastds = 'dict';
        end;
endsubmit;
dcflag = '';
lastds = 'dict';
end;
if index(entities,'DICT') &
index(entities,'FMTDEFS')then do;
submit;
proc sql;
create table dict as
select * from dd.fmtdefs
&fmtdefs_where, &dcflag dict &dict_where
where fmtdefs.var = dd.var;
endsubmit;
dcflag = '';
lastds = 'dict';
end;
if index(entities,'DICT') &
index(entities,'EDITVARS')then do;
submit;
proc sql;
create table dict as
select * from &evflag editvars
&editvars_where, &dcflag dict &dict_where
where editvars.var = dd.var;
endsubmit;
dcflag = '';
lastds = 'dict';
end;
if index(entities,'DICT') &
index(entities,'TBLVARS') &
not(index(entities,'TABLES')) then do;
submit;
proc sql;
create table dict as
select * from &tvflag tblvars
&tblvars_where, &dcflag dict &dict_where
where tblvars.var = dd.var;
endsubmit;
dcflag = '';
lastds = 'dict';
end;
if index(entities,'DICT') &
index(entities,'HEDITALS')then do;
submit;
proc sql;
create table dict as
select * from dd.heditals
&heditals_where, &dcflag dict &dict_where
where heditals.var = dd.var;
endsubmit;
dcflag = '';
end;

Now that all chosen entities have been combined into one data set, submit code to sort and print the last data set created. If the combined data set has no observations, display a message box, otherwise call the FOOTER section to print "END OF REPORT" and display the output window.

submit;
proc sort data=&lastds(keep=&keepvars);
by &sortv1 &sortv2 &sortv3 &sortv4;
run;
proc print data=&lastds noobs;
by &headvars notsorted;
var &vars;
title 'ACS Data Dictionary';
&t2; &t3; &t4; &t5; &t6; &t7; &t8; &t9; &t10;
run;
endsubmit;
sysrc = filename('outfile','rptparams.sas');
rc = preview('file','outfile');
submit continue;
endsubmit;

dsid = open(lastds,'i');
if attrn(dsid,'nobs') = 0 then do;
msglist = makelist();
msglist = insertc(msglist,'There were no
observations in the joined data set.');
text = messagebox(msglist);
end;
else do;
link footer;
call execcmdi('output');
end;
r = close(dsid);
end; *multiple entities;
return;

FOOTER:
submit continue;
data _null_; file print;
put 'END OF REPORT';
return;
run;
endsubmit;
return;

SAVING AND PRINTING
The other buttons on the third tab allow the user to print the output, save the output and/or the report parameters in a file, run a report whose parameters have been saved previously, or clear the output window. The code shown below is executed when the user enters a filename for saving output or parameters or retrieving parameters. The file name is entered in a text entry field displayed when those buttons are clicked.

FILENAME:
if Filename.Text ^= ' ' then do;
select(fflag);
when ('o') do; *save parameters;
filecmd = 'cp rptparams.sas ' ||Filename.Text;
rc = system(filecmd);
end;
when ('r') do; *run a saved report;
sysrc = filename
('outfile',"'"||Filename.Text||"'"');
r = preview('include','outfile');
submit continue;
endsubmit;
link footer;
call execcmdi('output');
end;
when ('s') do; *save output;
filecmd = 'output;file "' ||Filename.Text||"'";
r = system(filecmd);
end;
CONCLUSION

With careful planning, it is possible to program for any possible combination of data sets in a report. Start with the data sets which are joined with only one other data set and work toward the one which can be joined with the most others.

Due to space limitations, not all of the SCL code for the report is shown in this paper. The complete code is available at http://users.starpower.net/abennet.

SAS/AF is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

REFERENCES


CONTACT INFORMATION

Aileen D. Bennett
U.S. Census Bureau
DSD, Room 1657, MS 8400
Washington, DC 20233
301-457-8052
Fax: 301-457-8077
aileen.d.bennett@census.gov