MANIPULATION OF HIERARCHY TABLES IN SAS/EIS® APPLICATIONS

Jeff Lessenberry, Jeff Lessenberry Consulting Group

Abstract

This paper is intended to give hope to all those developers who have been told: 'It can't be done!' SAS/EIS was designed to allow non-programmers to create standard GUI screens, but what happens when the applications become nonstandard. This paper describes how one such nonstandard problem was resolved. It shows how to create and change user defined hierarchy tables through interactive screens, dynamically update the metabase hierarchy, and allow applications to use the new hierarchy. These methods will allow the users more flexibility when using hierarchy tables in data analysis.

Introduction

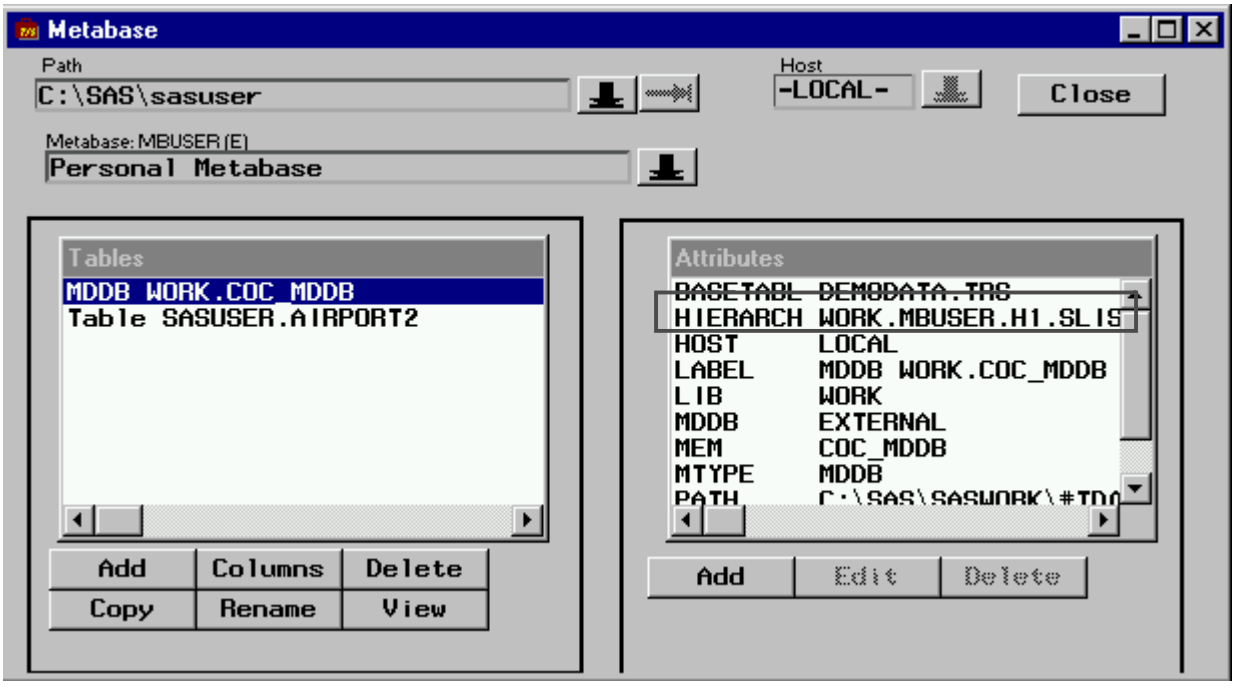
Trying to predict how a user will explore data is like trying to predict the next Elvis sighting, but this is the dilemma of a GUI application designer when the possibilities are endless. SAS/EIS applications use hierarchy tables (drill-down lists) to control data subsetting and exploration. One of the following routines is normally used to create hierarchy tables: register a data table in the metabase, interactive SAS/MDDB creation, or the SAS/MDDB procedure. The SAS/EIS application will use the metabase entry to define the sequence a 3-D graph or multi-dimensional report will use in the drill-down process. SAS allows the creation of multiple drill-down lists for each metabase entry. The EIS MDDB data model class includes the method, SWAP_HIERARCHY, to allow swapping from one predefined hierarchy to another. The complexity of this process grows exponentially with the number of variables in the data set. The question becomes do you create every possible drill-down sequence or do you decide which variables are to be kept and which are to be thrown away? The question remains, how do you give the users the flexibility they need without requiring a separate hard drive for the drill-down lists?

Data Table or MDDB

The choice between the SAS data table and MDDB (Multi-dimensional DataBase) comes down to cost versus speed. The MDDB server module may satisfy the need for more speed without purchasing new computer equipment. Many people, including programmers, ask 'How will MDDB make my application work faster?' When SAS data tables are used in SAS/EIS applications, a temporary MDDB is created in the *initialization phase*. That's right, the MDDB will be created on the workstation. The creation time will depend on processor speed, RAM availability, and disk speed. By passing the responsibility of creating the MDDB to a remote server, the run time between different workstations is negated. Creating the MDDB on a server has the additional benefit of requiring only one MDDB license. The cost of purchasing 100 new workstations is far greater than installing the MDDB server module on a UNIX client server system. The ability to make changes in your database structure and synchronize the changes with the metabase is another benefit to creating your own MDDB, which will be discussed later in this paper.

Metabase Registration

The registration of a data table or MDDB follows the normal procedure. SAS data tables must have the hierarchy attribute added, with all variables included in the hierarchy, to function properly. Look for the physical location of the hierarchy table in the Attributes Window.



The metabase attributes window shows the current location of the metabase hierarchy SLIST.

Metabase SAS data file showing the hierarchy file location.

VIEWTABLE: Personal Metabase

	OBJTYPE	DSETNAME	VARNAME	ATTRIBUT	VALUE
1	DATA	SASUSER.AIRPORT2	-NONE-	HIERARCH	SASUSER.MBUSER.H.SLIST
2	DATA	WORK.COC_MDDB	-NONE-	HIERARCH	WORK.MBUSER.H1.SLIST

The location of the hierarchy is usually the library in which the metabase resides. Since most metabases do not usually reside in the work library, this location needs to be changed. One method of achieving this would be to copy the metabase to the SAS work library, make the changes, and then copy the metabase back to the permanent location. If the metabase hierarchy is not updated in the SAS work library, the metabase data table must be updated. A SAS metabase consists of three major components: a SAS data table, a SAS catalog, and a SAS data table index. If the metabase MBUSER is located in SASUSER, then the data table will be SASUSER.MBUSER.

Open the file SASUSER.MBUSER in table form. Change the mode from browse to edit. Change the metabase hierarchy location to WORK.#######.H##.SLIST.

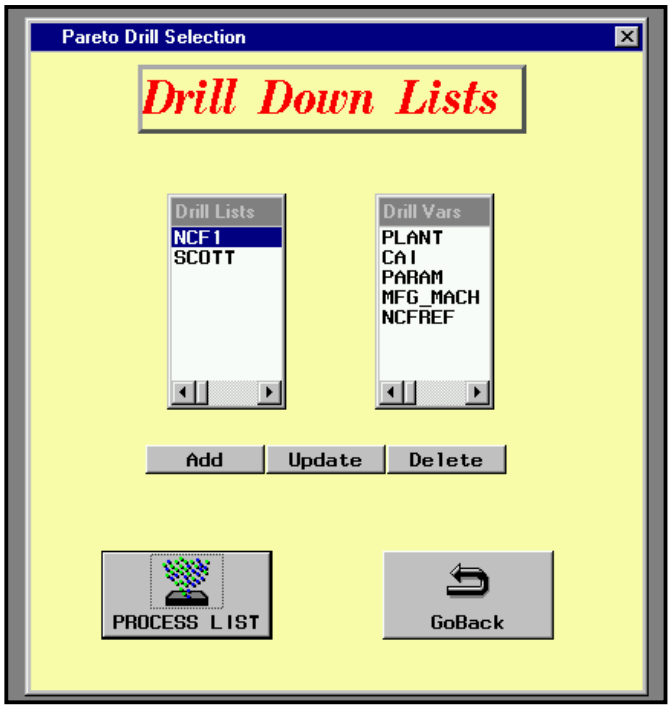
After changing the location to WORK, save the file. Take note of the SLIST member name because it will be used in a later step. Run a PROC COPY to copy the metabase files to the WORK library. This step must executed when the application is being run. Place the following code before the 'AF C=' or 'RUNEIS APPL=' statements in the SAS code file that starts the application.

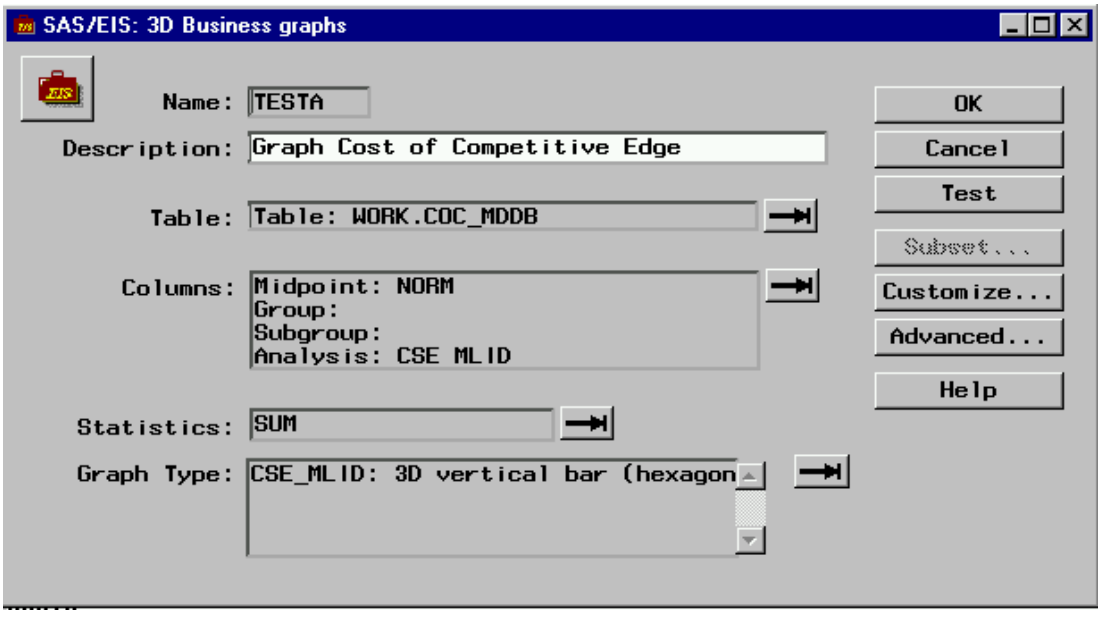
```
Proc copy in=SASUSER out=WORK;
     select MBUSER/memtype=catalog;
     select MBUSER/memtype=data;
run;
```

After copying the metabase to the WORK library, you may create the EIS applications as normal. Make sure the SAS/EIS application uses the SAS data table or MDDB that is registered in the WORK library metabase.

Creating the Drill List

The manipulation procedure of drill lists must be user-friendly but flexible enough to be accepted by all users. If the user is allowed to create drill-down lists and save them for future use, the manipulation process will go from being time consuming to time saving. The frame below shows one way of creating the drill lists.





Above: The drill-list screen allows creation, modify, and delete user-defined drill down lists.

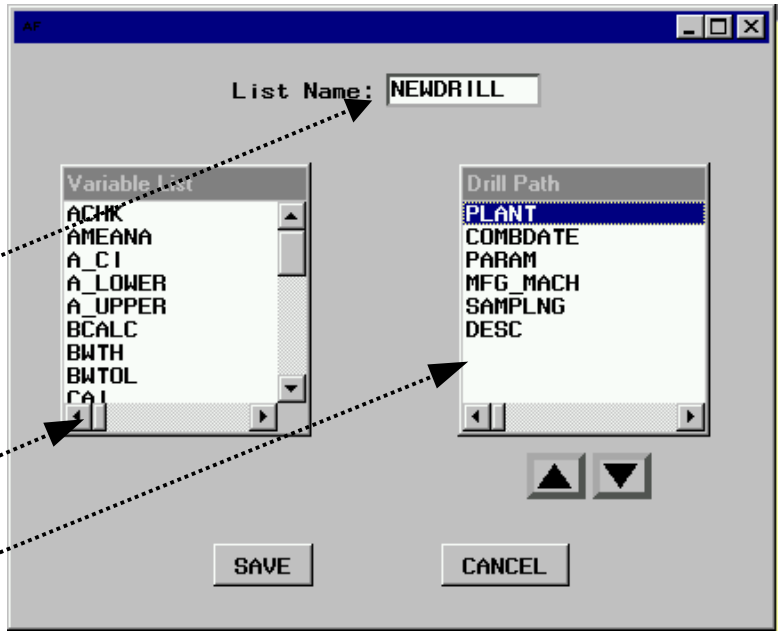
Left: EIS application definition screen. **Notice the table is in the WORK library.**

```
SLIST format=( LABEL=( PLANT='PLANT'
                       COMBDATE='COMBDATE'
                       PARAM='PARAM'
                       MFG_MACH='MFG_MACH'
                       SAMPLNG='SAMPLNG'
                       DESC='DESC'
                      )[1983]
               VARNAME=( 'PLANT'
                         'COMBDATE'
                         'PARAM'
                         'MFG_MACH'
                         'SAMPLNG'
                         'DESC'
                        )[1975]
              )[1967]
```

This SCL list will be copied into the SLIST entry in the metabase hierarchy catalog. It is already formatted for that purpose.

This SCL list will be used to create a class statement for PROC MDDB at run time.

Above: Shows the SCL list that is created for user drill list entries. Right: Shows the drill list creation and update screen.



The drill lists are stored as SLIST entries in a SAS catalog in the user's SASUSER library. The SLIST entries encompass two SCL lists: the LABEL list and the VARNAME list.

When creating a new drill list, the new list name input field will be empty.

When the Update button is selected, changing the list name can create a copy of an existing list.

The first list box contains the available variable to be used in the drill list.

The second list box contains the variables already selected for the drill list.

Variables are moved by double clicking the variable.

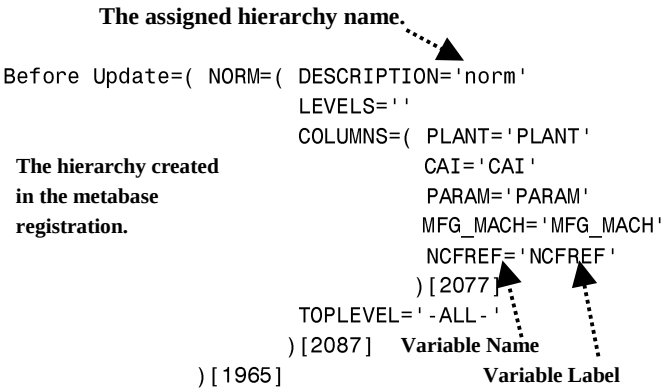
The sequence of the drill variables can be changed. Select the variable to be moved and use the up and down control buttons. The program creates a variable list by submitting the following code:

```
proc contents data=&indset noprint
     out=varlist short;
run;
data varlist;
     set varlist;
     if label = '' then label = name;
run;
proc sort data=varlist(where=(label ne ''))
          keep=name label);
     by label;
run;
data varlist;
     set varlist;
     select = '';
run;
data drillist;
     set varlist(obs=0);
run;
```

If the process is to update an existing drill list, the program will open the SLIST entry for that drill list and fill an SCL list with the contents. Variables existing in both files are flagged. A variable can not be selected twice, so it will be hidden from the variable selection list once it has been moved to the drill list. This is accomplished by only showing the variables that are not flagged. When the user presses the **SAVE** button the program will format the SCL lists, save it to the name specified, and exit the program. The start frame will reread the catalog entries to determine if a new drill list has been added. If so, the list boxes will be repopulated with the new entries.

Changing the Hierarchy SLIST

After a drill list has been selected and retrieved, the hierarchy SLIST can be changed. The SLIST is located in the SAS metabase catalog in the WORK library. It will usually be named H#.SLIST. This is the initial SLIST format:



The following code is taken from the process section in the main frame.

The first section fills the SCL list COMBOLST with the list selected in the drill list selection box. COMBOLST is then split into two SCL lists: LABLIST and NAMLIST. The first variable of the drill list is read and placed into a macro variable for future use.

```
listname = "sasuser.ardrilln."!!
           getitemc(N_DRILLS,nrow)!!".slist";
rc=fillist('catalog',listname,combolst);
lablist=getniteml(combolst,'LABEL');
  /* SLIST format  */
namlist=getniteml(combolst,'VARNAME');
  /* variable names only   */
vname=getitemc(namlist,1);
topvar = trim(vname);
  /* TOP Variable in list  */
call symput('topvarna',topvar);
```

The next section reads the SCL list NAMLIST and creates the variable NLINE, which will be substituted in the submit block to create the class statement in PROC MDDB later in the program.

```
 nline = '';
do it = 1 to listlen(namlist);
   vname=getitemc(namlist,it);
   if it = 1 then nline = trim(vname);
   else nline = nline!!" "!!trim(vname);
end;
```

The final section fills the SCL list PERMLIST with the SLIST entry from the metabase hierarchy catalog. The first list id number is put into LEVEL1. The list id number for the named list COLUMNS is COLLIST, which is cleared and the SCL list LABLIST is copied into its place. PERMLIST is saved back into the metabase hierarchy catalog.

```
if exist('work.archmeta.h.slist') = 1 then
      rc=fillist('catalog','work.archmeta.h.slist'
                 ,permlist);
level1=getiteml(permlist,1);
          /* Get 1st list id number    */
     collist=getniteml(level1,'COLUMNS');
          /* Get column list id number */
     rc=clearlist(collist);
          /* Clear column list         */
     collist=copylist(lablist,'Y',collist);
          /* Copy new list into column lst*/
rc=savelist('catalog','work.archmeta.h.slist',
            permlist);
```

After the SLIST has been replaced, the hierarchy list will have this structure:

```
After Update=( NORM=( DESCRIPTION='norm'
                      LEVELS=''
                      COLUMNS=( PLANT='PLANT'
                           COMBDATE='COMBDATE'
                           PARAM='PARAM'
                           MFG_MACH='MFG_MACH'
                           SAMPLNG='SAMPLNG'
                           DESC='DESC'
                           )[2077]
                      TOPLEVEL='-ALL-'
                     )[2087]
               )[1965]
```

After the changes have been saved, start the EIS programs with the following commands:

```
comm='RUNEIS APPL=work.archapps.grph1a.graphs';
call execcmdi(comm);
```

The SAS/EIS application will use the new hierarchy. If a new MDDB is created that has different variables than the MDDB metabase registry, the MDDB_SYNC method needs to be added to a method that runs before the screen is initialized. This method will synchronize the MDDB and the metabase registry.

```
call send(metabase,'_sync_mddb_',rc,msg,
          'work.mbuser','work.coc_mddb','N');
```

If a SAS data table is being used, the following program may be run before the EIS application to synchronize the metabase with the current data file.

```
METABASE = 'CAPEIS.CAPMETA';
   /*  Original metabase   */
SYNCFILE = 'WORK.CAPSET';
   /*  Data file to synch  */
TMPEIS   = 'WORK.CAPMETA';
   /*  Temporary metabase  */
/* allvars2 is all variables separated by commas */
call display('autosync.scl',metabase,
             syncfile,allvars2,tmpeis);
```

```
/*************************************/
/***          AUTOSYNC.SCL            ***/
/*************************************/

INIT:
   entry metabase $ 20 syncfile $ 20 nline2 $ 200
         tmpeis $ 20;
   length name $ 10 msg $ 55;
   Submit Continue Status;
          data &tmpeis;
               set &metabase ;
          run;
          proc sort data=&metabase
                    (where=(dsetname='&syncfile'))
                    out=tstvars(keep=varname )
                    noduplicates nodupkey;
               by varname;run;
          run;
          data tstvars;
               set tstvars;
               name=varname;
          run;
          proc contents data=&syncfile noprint
               out=sync;run;
          data newvol;merge sync tstvars;by name;
               if varname ne '' then delete;
          run;
          data newvars(keep=objtype dsetname varname
                         attribut value);
               length objtype $ 8 dsetname $ 17
               varname $ 8 attribut $ 8 value $ 8;
               set newvol;
               objtype = 'DEFLTVAR';
               dsetname = 'WORK.CAPSET';
               varname = name;
               attribut = 'VARNAME';
               value = name;
               value = trim(left(value));
               output;
               attribut = 'VARTYPE';
               if type = 1 then value = 'N';
               else value = 'C';
               value = trim(left(value));
               output;
               attribut = 'VARLEN';
               value = length;
               value = trim(left(value));
               output;
               attribut = 'VARLABEL';
               if label = '' then value = name;
               else value = label;
               value = trim(left(value));
               output;
               attribut = 'VARFMT';
               value = format;
               value = trim(left(value));
               if value ne '' then
                  value= trim(left(value))!!
                  trim(left(formatl))!!".";
               output;
               attribut = 'VARINFMT';
               value = informat;
               value = trim(left(value));
               if value ne '' then
                  value= trim(left(value))!!
                  trim(left(informl))!!".";
               output;
               objtype = 'VARIABLE';
               attribut = 'CATEGORY';
               value = '';
               value = trim(left(value));
               output;
          run;
          data &tmpeis;
                  set &tmpeis newvars;
          run;
          data &tmpeis
                (index=(ATTRIBUT VARNAME key2=(DSETNAME
                VARNAME ATTRIBUT)/UNIQUE));
              set &tmpeis(where=(dsetname='&syncfile'
                   and VARNAME IN ( &nline2 ,'-NONE-')));
          run;
      EndSubmit;
    return;
```

Conclusion

Hierarchy tables are powerful tools and by using the procedures discussed, many SAS/EIS applications become more flexible. This flexibility makes the applications easier and more efficient for the user, while requiring less maintenance and support. The workstation has becoming a graphic dumb terminal with many applications running in a client/server environment. The solution to making applications run faster should not be purchasing new equipment every six months, but implementing better development decisions. We should focus on upgrading our techniques of efficient coding and load balancing to fully utilize the capabilities of the existing hardware.

Considerations when customizing a metabase structure:

1. **If the metabase is edited and the hierarchy is opened interactively, the location of the hierarchy will revert to the metabase's edited location.**
2. **If a variable is added to the database, it will not be accessible to the metabase or hierarchy until the metabase registration is updated.**
3. **Any future SAS/EIS application changes will require the data table or MDDB exist already and the metabase must be copied into the work directory.**

Acknowledgements

Jeff Lessenberry
Jeff Lessenberry Consulting Group
Phone: (800) 654-5908
Email: JLCG@concentric.net