Paper 44

Using the SAS/MDDB® and User Written Methods to Enhance SAS/EIS® Objects

Ben Cochran, The Bedford Group, Raleigh, NC

Abstract

SAS/EIS software provides an extensive, interactive applications development environment and toolkit that can be very useful for rapidly building end-user applications.

Applications developers sometimes find that the list of pre-existing objects provides much of the functionality they need in their applications, but only up to a point. Eventually, they find that they must enhance their applications by making use of other tools provided as a part of the SAS® System. This paper addresses two strategies for extending SAS/EIS applications: one for enhancing drilldown speed, the other for enhancing information delivery.

Introduction

To accomplish its objectives, this paper is divided into three parts. The first section, examines the data that will be used, the registration of the data, and the actual building of a SAS/EIS object. The second part looks at building an MDDB, and then using it to enhance the object. Finally, the last section will address writing a method, then enhancing the object with it.

The Data: Understanding the data, or at least having a good idea about the contents of the data and what you are going to do with it, is important. The data will be used to build a drill down bar chart. This process will be illustrated with hospital data which is described below.

	Variable	Length	Format
_	SERVICE_	$20	
_	PROGRAM_	$20	
_	DRG	$3	
_	DESCRIO	$30	
_	CASES	8	BEST22.
_	HOSPITAL	$12	
_	ALOS	8	
_	AMOUNT	8	
_	REGION	$7	

The data is in the form of a SAS data set (MYLIB.HOSP9) which contains nine variables. There are three numeric variables; one (CASES) will be used as an analysis variable to determine the length of the bars. The remaining six are character variables, any one of which could be used as a 'midpoint' variable. However, by building a drilldown bar chart, we will use several of these variables in the drilldown hierarchy. In a very general sense, the order of the variables in the hierarchy is from general to specific. In other words, the variable with the least number of unique values should be at the beginning of the hierarchy, and the variable with the most unique values should be at the end. For example, the variable REGION has the fewest unique values so a graphic chart will be created initially with a bar for each REGION. When a bar is selected, a new report will be displayed with a bar representing each HOSPITAL in that REGION. So, the first task is to tell the SAS System how we want to use the data to generate these kinds of reports.

Registering the Data: Issue the EIS command from any window to make the EIS Main Menu appear. Next, select the Metabase icon to open the *Metabase* window. The metabase contains information about the data that will be used in the application. The SASUSER library, which is the default location, will be used to house the metabase. If this is the first time this window has been used, the Metabase will be empty.

To register a data set in this Metabase, select the .Add pushbutton below the *Tables* box. This opens the *Select Table* window. Click on the *Path* arrow and select the MYLIB library. This causes all of the SAS files in this library to appear in the *Available* listbox. Double-click on the HOSP9 data set and it moves to the *Selected* listbox. Then click on the OK pushbutton. When the Metabase window reappears, it will contain MYLIB.HOSP9. Click on this data set,

and the default attributes appear in the *Attributes* listbox. The attribute needed (HIERARCH) does not appear, so click on the Add pushbutton (below the *Attributes* listbox) and the *Table Hierarchy* window appears. This is the place that enables the developer to define the hierarchy.

First, supply a name, HIER1 for example, in the *Description:* field, This will be used as the name of the hierarchy. Next, choose the variables from the *Available* listbox in the order of the drilldown hierarchy. In this case the order is: REGION, HOSPITAL, SERVICE_ (service line area), PROGRAM_ (general program description), and DESCRI0 (specific procedure description). The variables will appear in the *Selected* listbox. Select the Add push-button, and then OK to complete the hierarchy registration.

The next task is to tell the SAS System which variable will determine the length of the bars. From the *Metabase* window, with MYLIB.HOSPITAL highlighted, select the Columns button. From the *Columns* window, select CASES, then the Attribute Add button. Next, choose Analysis as the new attribute, then OK. Accept the Default Analysis Type as SUM, then select OK. This completes the registration process, so select Close twice to return to the Main Menu.

Building the Object with a Data Set : From the Main Menu, select the *Build EIS* icon. If this is the first time this icon has been selected, the default Application Database, SASUSER.SASAPPL, appears. This will be used to house the EIS application being built. If SASUSER.SASAPPL does not appear automatically, it can be chosen by selecting the *Path* arrow. Select the Add button to begin the building process. From the Add window, select Multidimensional Reports from the *Object Databases* listbox, then 3D Business graphs from the *Objects* listbox. (These should already be selected by default). Next, select Build to open the *3D Business graphs* window.

Supply a name (TRY1) and an appropriate description., then select the *Table* arrow. The *Select Table* window opens containing only SAS files that would be appropriate for this object (those that have been registered with the HIERARCH attribute). Double-click on MYLIB.HOSP9, then OK. The *3D Business graphs* window reappears with MYLIB.HOSP9 as the selected table. Next, select the *Columns* arrow. The *Columns* window appears and is the location for the developer to tell the SAS System how the registered variables are to be used in building this object. First, select HIER1 (the name of the registered hierarchy) as the Midpoint. Next, from the *Type* listbox, select Analysis. This causes the variables registered with the analysis attribute to appear in the *Available* listbox. CASES should appear, so double-click on it, then click on the OK pushbutton to return to the *3D Business graphs* window.

The Customize button contains ways to enhance the report such as adding titles, footnotes, formats, labels, and other options. For the sake of this report, these characteristics will not be examined.

To see the reports select the Test button. A vertical bar chart appears with a bar for each region. Double-click on the bar for the WEST region. A new graph appears with a bar for each hospital in the region.

So far, so good. However, there are at least two ways in which this report can be improved. If the data used to generate this report is a large one, the response time in both initializing and in the drilldown may be less than adequate. Also, notice that the report does not indicate what subset is in effect. In other words, when the report showing a bar chart for each hospital appears, nowhere on the face of the report can the region be seen. The rest of this paper will address these two issues. First, an MDDB will be built to improve response times.

Building an MDDB

What is an MDDB? Basically the MDDB is a type of SAS file (it has a filetype of MDDB). It is a read-only file that provides the SAS System with a way to store large amounts of pre-summarized data (similar to, but different from PROC SUMMARY output).

It contains at least one summary structure known as an NWAY table. Additionally, it usually contains one sub-table created from the NWAY table. While there are no limits to the number of

sub-tables, too many unnecessary ones can result in storage space problems. It can be thought of as a 'cube' with multiple dimensions that provide fast and flexible access to the data. The SAS/MDDB has incremental update capability and can be created either in batch programs or on-line applications.

Before Building an MDDB, you should have a good understanding of your data and how it will be used. One of the biggest issues is finding a good balance between response time and storage space. You need to know which sub-tables to build so that you do not use disk space needlessly, and at the same time, have the best performance. In order to achieve the best response time, a good rule to remember is to have a sub-table that contains exactly the same categorical variables as the drill-down report.

Having a good understanding of user expectations will also enable you to address the time vs. space issue. Primarily, what reports do the end-users want and what drill-down hierarchy is needed.

Building an MDDB can be accomplished in more than one way. This paper will focus on the PROC MDDB approach. The general syntax is:

```
proc mddb data = dsname
      out = libref.mddb ;
      class categorical variable list ;
      hierarchy class variable list  /
         name= xxx display = yes | no ;
```

On the **proc** statement, the data = option names the SAS data set to be summarized. The out = option names the MDDB that will be created. Ex.

```
proc mddb data = sas.yr98 out = new.mddb ;
```

The **class** statement names the variables to be used as classification variables in the same way that a class statement is used in the summary procedure. Ex.

```
class country city company ;
```

The default order of the variables' values on the class statement is ascending. This can be changed by using any of the following order options: descending, ascformatted, desformatted, dsorder. For example, a data set has countries in order by continent and that order must be retained in a report. To accomplish this, the above class statement needs to be modified as follows:

```
class country / dsorder ;
class city company ;
```

The **hierarchy** statement creates sub-tables in the MDDB. The sub-table can be named with the name = option (this is highly recommended). By default, these named sub-tables are not recognized by SAS/EIS during the Metabase registration process. To change this default action, specify *yes* as the value for the display = option. This will eliminate the need to create hierarchies while going through the SAS/EIS registration process. Ex.

```
hierarchy country city company /
name='geo'  display = yes ;
```

Note: If the hierarchy statement is NOT used, then only the NWAY table is produced.

The **var** statement identifies the analysis variables and the statistics to be stored in the MDDB. Ex.

```
var sales96 sales97 / sum n ;
```

More than one var statement may be used, however a given variable may appear in only one var statement. While the sum is the default statistic, other statistics that can be used on the var statement are: n nmiss min max uss sumwgt and uwsum.

The SAS dataset that will be used in this section to illustrate the building of an MDDB is **MYLIB.HOSP9,** the same dataset used in building the object in the introductory section. The complete proc mddb step for building an MDDB is:

```
proc mddb in = mylib.hosp9
      out= sugi24.mddb9 ;
      class region hospital service_ program_
            descri0 ;
      var cases amount ;
      hierarchy region hospital service_
            program_ descri0  /
            name = 'drilldown'  display = yes ;
run ;
```

The above code builds the MDDB, but before it can be used in a SAS/EIS application, the MDDB needs to be registered.

Registering the MDDB: To register the MDDB so that it can be used in drill-down applications, the user needs to go through essentially the same process as registering a data set. Refer to the **Registering the Data** section in the **Introduction.** Go to the same Metabase used to register the data set. From the ***Metabase*** window**,** select the **Add** pushbutton under the *Tables* listbox. From the ***Select Table*** window, choose the path for the SUGI24 library, then scroll to find the MDDB9 mddb and select it. Choose OK to return to the ***Metabase*** window. The MYLIB.MDDB9 file should be in the *Tables* listbox. Notice the default attributes that appear in the *Attributes* listbox. Notice that the BASETABL is MYLIB.MDDB9, the MDDB is designated as an EXTERNAL MDDB, and that the HIERARCH attribute is already assigned. Double click on the LABEL attribute. Change the label to MDDB for SUGI24 DEMO. Next, select OK to close the ***Enter Label*** window. Now, double click on the HIERARCH attribute to open the ***Table Hierarchies*** window. Verify that the drilldown hierarchy exists as defined in the PROC MDDB step. Select **OK**, then **Close** to return to the SAS/EIS Main Menu. The MDDB is now registered to be used in a SAS/EIS object.

Building the Object with an MDDB: The next step is to build the 3D Bar Chart object that will use the MDDB. Refer to the **Building an Object** section in the **Introduction.** From the Main Menu in SAS/EIS, select the Build EIS icon to make the ***Build*** window appear. Use the same application database as before (SASUSER.SASAPPL), to build this object.

To begin, choose the Add pushbutton at the bottom of the ***Build EIS*** window. Select Multidimensional Reports from the Object Databases listbox. Then select 3D Business Graphs from the *Objects* listbox. Next, select the *Build* pushbutton to open the build environment.

Supply a name (TRY2) and description of your choice. Click on the arrow beside Table: and select the MDDB for SUG124 Demo (SUGI24.MDDB9) from the *Available* listbox. Select OK. Click on the arrow beside Columns:. For the Midpoint, select *drilldown* (the hierarchy built from the proc mddb step). Next, select Analysis and choose CASES, then click on OK to close the ***Column Selection*** window. The Statistics area should be filled in with Sum.
The report is ready to be viewed, so click on the Test pushbutton. The initialization and drill-down should be much faster than with the dataset.

In the fall of 1998, tests were performed by the Bedford Group in Raleigh, NC to compare the response times of datasets of varying sizes and the MDDBs built from them. The testing was run on the Multidimensional Report objects, instead of the 3D Business Graph object discussed in this paper. The following methodology was used in the comparison :

- A SAS dataset was created with hospital data named HOSP1.
- HOSP3 was created from HOSP1 with triple the size (of HOSP1).
- HOSP9 was created from HOSP3 with triple the size (of HOSP3).
- MDDB1 was created from HOSP1 with PROC MDDB.
- MDDB3 was created from HOSP3 with PROC MDDB.
- MDDB9 was created from HOSP9 with PROC MDDB.
- Six SAS/EIS objects were built, 3 with the SAS datasets (HOSP1, HOSP3, and HOSP9), and 3 with the MDDBs (MDDB1, MDDB3, and MDDB9).
-

Once the six objects were built, they were timed in three areas:

- initializing the report
- drilling down to the second level of the hierarchy,
- drilling down to the third level of the hierarchy.

There were seven trials that were timed with the minimum and maximum time being discarded. The chart below consists of the average score of the 5 remaining times. The figures in the chart are time is in seconds.

	Initialize	Drilldown to Second Level	Drilldown to Third Level
HOSP1	7.25	3.10	2.40
MDDB1	2.07	0.92	0.88
HOSP3	8.65	7.60	6.15
MDDB3	1.80	1.10	0.80
HOSP9	22.95	21.83	23.29
MDDB9	2.61	1.16	0.79

There is one remaining enhancement to be done with the 3D Business Graph (bar chart) object. The drill path is not captured in the title area. A method will be written to do this.

Writing a Method

A method is an SCL program than can effect the way an EIS object executes. Many methods are written to override the default behavior of an object.

The SCL program that captures the drill path and places it in the title area must contain a labeled section that:

- retrieves the current MODELID
- retrieves the current subset list
- builds the desired title text
- writes out the subtitle.

To begin, there must be statements to instantiate the specified methods that control navigation. These statements should include the location of the SCL program and the label that will be used to override the methods. Since this instantiation occurs just once for the running of the application, the _POSTINIT_ method is an ideal method to override. The sample code shown below is located in the **mylib** library, the **eiscusts** catalog, and the **dpttlgrf_sugi24.scl** entry. Notice the label (postinit).

```
modelid=modelid;
_self_=_self_;
rc=rc;

postinit: method;
   call send(_self_,'_set_instance_method_',
             '_navigate_up_',
             'mylib.eiscusts,dpttlgrf_sugi24.scl',
             'ddtitle','after');
   call send(_self_,'_set_instance_method_',
             '_navigate_down_',
             'mylib.eiscusts,dpttlgrf_sugi24.scl',
             'ddtitle','after');
   call send(_self_,'_set_instance_method_',
             '_navigate_left_',
             'mylib.eiscusts,dpttlgrf_sugi24.scl',
             'ddtitle','after');
   call send(_self_,'_set_instance_method_',
             '_navigate_right_',
             'mylib.eiscusts,dpttlgrf_sugi24.scl',
             'ddtitle','after');
endmethod;
```

Within the same scl program, the following method must also appear. Notice the label (ddtitle).

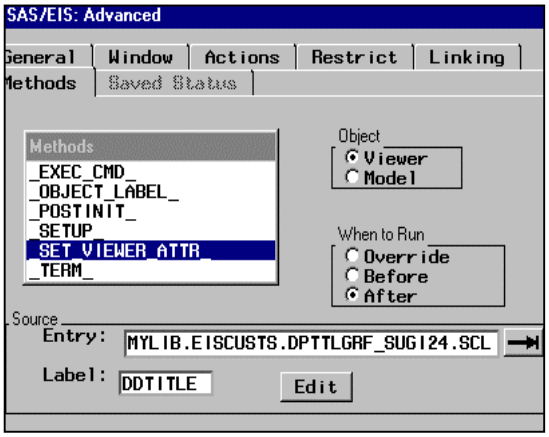
```
ddtitle:  method optional = cmdline $ rc 8;
   sublist=makelist( );
   call send (modelid, '_get_subsets_', sublist);
   where = ' ';
   do i = 1 to listlen(sublist);
      catvarlist=getiteml(sublist,i);
      namelist=nameitem(sublist,i);
      cat_str = ' ' ;
      if listlen(catvarlist) > 0 then do;
        do j = 1 to listlen (catvarlist);
          value = left(getitemc(catvarlist, j ));
          if j = 1 then cat_str = trim(value);
          else cat_str = cat_str || ',' || trim(value);
        end;
      end;
      if listlen(catvarlist) then do;
        where =
          where || ' '|| namelist || '=' || cat_str;
      end;
   end;

   call send (_self_, '_set_title_', 2 , where ) ;
      if length ( where) > 65 then
        call send (_self_,'_set_title_, 3 ,
          substr(where,65));
      if length (where) < 65 then
         call send (_self_,'_set_title_, 3 , ' ' ) ;
 rc = dellist ( sublist,  ' y ' ) ;
endmethod;
```

Once the method is written, it needs to be associated with the SAS/EIS 3D Business Graph object. First, go to the *Build EIS* window, and edit TRY2 (the object build earlier in this paper).

From the *3D Business graphs* window, select the Advanced… button, then the *Methods* tab. Next, choose the **_SET_VIEWER_ATTR_** method, then **Viewer** from the Object radio box, and **After** for When to Run. Click on the Source entry arrow, and supply the location of the SCL program containing the methods, in this case,

mylib.eiscusts.dpttlgrf_sugi24.scl. The last piece of information to include is **DDTITLE** which is the value for the Label: field.

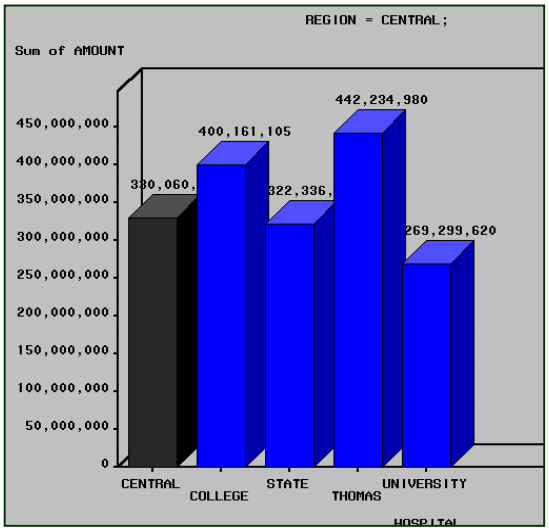
The completed window should look like the following figure.



Next, select the **_EXEC_CMD_** method. Click on **Viewer**, then **After**. Make the Source Entry: and Label the same as above.

Finally, select the **_POSTINIT_** method**.** Click on **Viewer,** then **After**. Make the Source Entry: **mylib.eiscusts.dpttlgrf_sugi24.scl** , and the Label: **POSTINIT**.

This completes the process of associating the method with the EIS object. Select OK , then Test to see the results. Select the bar representing the Central Region. The report should look like the one below:



Conclusion

The SAS System has many tools for exploiting data that may be in data warehouses. These tools are flexible and easily extendable. The SAS System has also greatly improved its ability to access data quickly with the offerings of multidimensional database. It is ideal for drill-down reports on datasets commonly found in data warehouses. Therefore, it greatly enhances the capability of SAS/EIS to be deployed as part of the front-end to data marts as well as data warehouses.

The author can be reached at the following location:

Ben Cochran
The Bedford Group
3216 Bedford Avenue
Raleigh, NC 27607
(919) 831-1191
bedford@interpath.com