Expanding Your EIS Capabilities

Don Stanley
EDS-Databank Systems Ltd
Wellington

The SAS/EIS product has been marketed as a tool for developing applications rapidly and effectively. This the product does remarkably well. Like any product designed to be quick and easy to develop with, their comes a time with SAS/EIS where you need to go beyond the basic system and expand your development capability.

This paper discusses how to extend SAS/EIS as follows:

- using customised PMENUS
- help entries
- FRAME entries to add to the basic functionality of the product
- how to make SAS/EIS applications perform a little or a lot faster.

The discussion is based around the business chart application type but the principles are similar for many of the application types. Since business charts are a popular first choice when creating an EIS, some discussion is also presented about when the business chart type is appropriate.

In the following discussion I assume that you do know how to build an EIS. I also assume knowledge of FRAME technology. All references to SAS releases are in the context of the OS2, V2.1 operating system running release 6.08 at TS407 support level.

CUSTOMISED PMENUS

An immediate change that many developers of EIS applications want to make is to the default menu buttons that appear at the bottom of each screen. The default menus are shown in Figure 1.1 for business charts.

I am going to retain just the VIEW, PRINT, GObACK and HELP buttons for this example.

The first step is to create the menu using PROC PMENU. Enter the following program in the program editor, save it to an appropriate catalog or external file, and then submit the program.

Figure 1.1
Note that the SASUSER.SASAPPL can be any catalog, I use that as it is the default catalog for EIS applications and I do most of my EIS development in there.

The GOBACK command does not have to be END. In the context of an EIS application going back from a screen usually means return to the last screen so a CANCEL would suffice just as well. There is no magic GOBACK command, the button just means END, CANCEL, QEND or QCANCEL in SAS Institute software. The GOBACK button is documented in SAS Institute documentation, but usually as a byline in the context of some other explanation (eg pg 92/93 SAS/AF Software: FRAME Entry Usage and Reference Version 6 First Edition).

The VIEW button is not documented anywhere that I am aware of. However, SAS/EIS supplies an excellent walkthrough of the product under the TUTORIAL button in the SAS/EIS main menu. The walkthrough uses a set of menus which have the source code stored in SASHELP.EIS.T_PMENU.SOURCE. In there are some useful button definitions including one for the view button.

One point here is that the menu commands in the SELECTION must be in upper-case. Some will work in lower-case, some won't. Both the PRINT and VIEW buttons do not function under release 6.08 at support level 4.07 if the menu defines them in lower-case.

Now that the menu is defined it is necessary to associate the menu with the EIS. To do this exit your EIS in the usual manner. Click on the methods button and then on the COMMAND MENU button. Select your menu from the list that gets popped up (you need to choose SASUSER, then SASAPPL, then EISOI from the windows that pop up to use the catalog I saved the above menu in).

When you invoke the earlier EIS with the customised menus, you see the screen in Figure 1.2.
HELP SCREENS

The first point to be aware of with help screens is that the HELP command in SAS opens its own AWS window. This overlays the application screen that you are requesting help on. If you have a custom help screen triggered by a HELP command, the custom screen will appear on a white background. Personally, I would prefer the help screen to act like a pop up window on top of the application window, but there appears nothing we can do about this. Calling a HELP screen from the CALL DISPLAY or CALL CBT SCL commands does not open the help AWS, and therefore displays help screens as windows on top of the application screen.

The HELP button executed within a SAS/EIS application cannot have its default help screen overwritten by any obvious method. However it is fair to say that the help provided is very useful, and tailored towards the end user rather than the developer. SAS Institute have done an excellent job of providing help that will allow an end-user to use the EIS you have set up productively within placing a high burden on the developer in a support role.

There may come a time however when you would like to provide more help to your application users, or even completely replacing the default help. I said there is no obvious way to overwrite help, but there are still some ways to achieve this:

- using the NOTES button that is provided with EIS as a help screen
- using a custom command assigned to a key that triggers our own help screens and overrides the usual help button
- same as above except we give the option of accepting either the SAS/EIS help or our own help

The difficulty with all these comes when a user wants to be able to get help on a specific value on a graph on the EIS screen. For example, in Figure 1.2 the user may have wanted to get some help on the bar labeled 10 Plus. A more likely example comes in the drill down from Figure 1.2. This drills down to a breakdown of the selected bar by department. A help screen may be required here to display information relevant to a department.

All these objectives can be met by coding. All bar the notes entry use a customised method which runs instead of (or in addition to) the usual EIS method.

USING THE NOTES BUTTON THAT IS PROVIDED WITH EIS AS A HELP SCREEN

The use of a notes entry as a help screen is not unusual, in fact the notes entry is itself a special sort of help anyway in that it offers a user some textual information not immediately available on-screen. If you do not require notes to be used in the manner that SAS Institute intended it there is no reason why you shouldn't use it as a help screen. Be aware though that you are restricted to using a SOURCE catalog entry and cannot call up a CBT.

CUSTOMISED SELECTION METHODS

Lets investigate each of the other three situations.

For all of these I are simply using the functionality of EIS outlined in the manual P253, User Written Methods And Customisation in SAS/EIS. This manual takes a bit of work to actually understand just how to use user-written methods, and isn't helped by a typographical error in a rather vital place. For the record, the list of return codes on page 19 should include setting RC=2 on exit from a custom selection method, and it is this return code that causes the EIS to not carry out default SAS processing after the custom method has finished.
Their may be another difficulty as well, depending on your platform and the SAS products you have installed. If you licence SAS/ASSIST ignore the rest of this paragraph, otherwise be aware that the %EIS macro that must be called by a custom method may fail on some platforms. The macro attempts to execute a macro called %ASASTYPE which is shipped with SAS/ASSIST, not SAS/EIS. This is a nuisance more than a show stopper, you can edit the EIS macro (stored in SAS/EIS/SASMACRO) and remove the reference to %ASASTYPE to get around it.

Calling a custom selection method is simple in EIS. You code the method in an SCL entry. You must include the following lines at the start, as the part of SAS/EIS that actually calls your custom entry passes it some parameters and will fail if not found:

```
%eis
%eisentry userlist 8 eisrc 3 ;
```

The USERLIST field is an SCL list identifier containing information about the entry in a named list. Included in this information is the text corresponding to the command that was entered.

EISRC is a variable that your custom method initialises which tells the calling SAS/EIS routines what to do next. If you set EISRC to zero the SAS/EIS method will continue after executing your method. Setting it to two causes the SAS/EIS entry to not continue, ie your method effectively replaces the SAS/EIS method.

**USING A CUSTOM COMMAND THAT TRIGGERS OUR OWN HELP SCREENS AND OVERRIDES THE USUAL HELP BUTTON**

The following code resides in an SCL entry. I tend to store my SCL custom entries outside of any EIS I am creating because it allows me to easily use them in multiple applications. Assume this code is stored in SASUSER.METHODS.TRAPHELP.SCL.

```
%eis
%eisentry userlist eisrc 3 ;

init:
    eisrc = 0 ; /* ensure always exit clean */
    /* check if the user requested help above the SAS help */
    if getnitemc(userlist,'COMMAND') eq 'MYHELP' then do ;
        call cbt('sasuser.sasappl.myhelp.cbt') ;
        eisrc = 2 ;
        return ;
    end ;
    return ;
```

So coding your own method can be that simple! But what does it all mean?

Firstly note the compulsory parameter setup lines at the start.

The whole method lives in an INIT section. That just reflects the fact that an SCL entry has to have at least one section, it doesn't have to be INIT.

I set EISRC to zero at the start. This is just a coding convention I have adopted over the years, to set default return codes early so I don't forget them later.

You may not have seen the GETNITEMC command before. If you get into SAS/EIS or FRAME development you are going to see it a lot. FRAME technology (SAS/EIS is written by SAS Institute using FRAME technology) has a tremendous amount of use of SCL lists, in fact I recently wrote a program to
loop through the numbers from 1 to 400 and print out the corresponding list with that number as a listid.
Now I know why FRAME needs at least 12 megabytes of memory! GETNITEMC is documented along with all other list functions and the list concept in general in P216, Changes and Enhancements to SAS/AF, SAS/FSP and SCL in Release 6.07. The function extracts a value from a named list.

The above use of GETNITEMC just searches the list for a named item called "COMMAND" and extracts its value, this being the last command issued. You don't to worry about how the command got into the list, SAS software does that for you when you click a button on the pmenu. I look for the last command having been "MYHELP" and carry out some custom processing if it were. Of course the help screen doesn't need to be a CBT.

How did MYHELP get on the pmenu? I simply modified the earlier custom menu to add a HELP button with a command "MYHELP" as follows:

```sas
proc pmenu c=sasuser.sasapp1; 
  menu myeismnu; 
  item 'Print' selection=print; 
  item 'Goback' selection=goback; 
  item 'Help' selection=help; 
  selection print 'PRINT'; 
  selection goback 'END'; 
  selection help 'MYHELP'; 
quit;
```

Figure 1.3 illustrates what happens when the "MYHELP" command is executed. Note that since the command runs a CBT that I coded, it means that as developer I have complete control over where the help screen sits and what its size is.

That was a fairly simple illustration of how to replace a help screen, and it also demonstrated how you can code methods to supplement or replace SAS Institutes own EIS processing.

Incidentally, when you issue a command that is a SAS command (eg HELP), the command gets executed before the SCL code behind the EIS screen. It appears that CONTROL ALLCMDS is not switched on, so when the command completes the SCL is not executed. This you cannot ever see the HELP command appearing in the custom method, as the method doesn't execute when a SAS command such as HELP is entered. However, as in standard SCL, when a non-SAS command is entered (such as MYHELP), the code runs and expects to find some
code that actually makes up the command.

**USING A CUSTOM COMMAND THAT ALLOWS EITHER OUR OWN HELP OR USUAL SAS HELP**

How can we allow for the default SAS help screen, and a custom help, allowing the user to choose between them?

There are two methods. One is easy and has a drawback which may cause problems sometimes, the other is more complex and less problematical.

The easy way is to define two buttons in the menu, one is the "MYHELP" used above, the other is "HELP". Clicking on the MYHELP button generates the custom help, the HELP button generates SAS Institutes EIS help. This is very simple and need not be explained further except to say that a problem arises when the application requires more menu buttons in which case the use of two HELP buttons may be a space waster on-screen.

The more difficult way is to modify the above method a little. The idea is to change it to use a list which pops up when the user requests help, and allows the user to choose either SAS/EIS help or customised help. The code changes to:

```sas
%eisentry userlist eisrc 3;

init:
eisrc = 0; /* ensure always exit clean */

/* check if the user requested help above the SAS help */
if getnitemc(userlist,'COMMAND') eq 'MYHELP' then do;
  listid = makelist();
  rc=insertc(listid,'Display Application Help Information') ;
  rc=insertc(listid,'Display SAS/EIS Help Information') ;
  rc = popmenu(listid) ;
  if rc eq 2 then call cbt('sasuser.sasappl.myhelp.cbt') ;
  else if rc eq 1 then do;
    submit command continue ;
    help
    endsubmit ;
  end;
eisrc = 2 ;
  rc = dellist(listid) ;
end;
return;
```

This code is somewhat more complex. I create a list containing the two help options, and when a "MYHELP" is detected pop up the list. The user selects an option and the appropriate help screen pops up.

**GETTING HELP ON DRILL DOWN VALUES**

This occurs when your user needs to access help that is relevant to a drill down value, rather than to the SAS/EIS screen as an entity. Using a help button is of little use in this case, as there is no way to detect which drill value help is required for.

Their is only one apparent way to do this, and it interferes with the usual flow of EIS processing. That is to intercept a drilldown request, and pop up a menu where the user chooses either drilldown or specific help.
The help screens need to be carefully set up, as it is probable that separate help will exist for each value on the current EIS screen.

I don't recommend attempting to create this sort of context sensitive help unless your user absolutely demands it. It gets very complex as you need to keep track of which drill value is requested, and help may not be required all the drill hierarchy fields. The following study illustrates the situation.

Firstly, assume that the first drill down hierarchy requires no help (it is the number of weeks graph we saw above) but the second does require a help (it is a department breakdown). Subsequent drilldowns require no help.

You need to be a little inventive about determining which hierarchy you are on. The easiest way is to keep a list of the text that can be displayed on the drill hierarchy that you want to supply help for. I want to supply help when a user clicks on a department number or bar, so I create a list of valid department numbers.

Now in my custom code TRAPHELP.SCL I check the COMMAND named item in USERLIST, looking for the command 'DRILL'. When I see that command, SAS/EIS will have set the field TEXT in the same list to the text at the left of the chart. I search the list of valid departments, to see if the text at the left of the graph is in that list. If it is, I pop up a menu requesting the user to select HELP or continue the drill down.

The help screens are a CBT. Each frame of the cbt represents an item that you want help about. You will need to consider a naming convention. Since my department numbers are all 3 chars long and numeric, I can place a character in front to obtain a valid name. Longer text items will cause some difficulty.

The TRAPHELP.SCL entry changes to:

```sas
%eisentry userlist eisrc 3 ;

init:
  eisrc = 0 ; /* ensure always exit clean */

/* check if the user requested help above the SAS help */
if getnitemc(userlist,'COMMAND') eq 'MYHELP' then do ;
  listid = makelist0 ;
  rc = insertc(listid,'Display Application Help Information') ;
  rc = insertc(listid,'Display SAS/EIS Help Information') ;
  rc = popmenu(listid) ;
  if rc eq 2 then call cbt('sasuser.sasappl.myhelp.cbt') ;
  else if rc eq 1 then do ;
    submit command continue;
    help
    end submit ;
  end ;
  eisrc = 2 ;
  rc = dellist(listid) ;
  return ;
end ;

if getnitemc(userlist,'COMMAND') eq 'DRILL' then do ;
  listid = makelist0 ;
  rc = insertc(listid,'741') ;
  rc = insertc(listid,'742') ;
  rc = insertc(listid,'743') ;
  rc = insertc(listid,'744') ;
  rc = insertc(listid,'746') ;
  rc = insertc(listid,'747') ;
  text = getnitemc(userlist,'TEXT') ;
```
if searchc(listid,text) then do;
  listid1 = makelistO;
  rc = insertc(listid1,'Drill To Next Hierarchy Level');
  rc = insertc(listid1,'View Help For ' || text);
  rc = popmenu(listid1);
  if rc = 1 /* view help */ then do;
    call cbt('sasuser.sasappl.dphelp.cbt',1,'D' || text);
    elsifrc = 2;
  end;
  rc = dellist(listid1);
end;
rc = dellist(listid);
end;
return;

When the user selects a drill down in any hierarchy level, the custom method takes over and searches the list for the text. If the text is found, the pop up menu is displayed, otherwise the drill down continues to the next hierarchy level. The pop up is shown in Figure 1.4, where the user has drilled on the 741 item.

That concludes the discussion on help screens. From this you will have seen that the evident restriction to just SAS Institutes help screen is not really a restriction.

Code presented in this section created popup lists and department lists as they were needed. This was for illustrative purposes, the lists should be created once at application startup and deleted at application end in a real application.

**ADDING FUNCTIONALITY TO SAS/EIS**

This section follows on from the discussion under HELP about invoking your own methods, into invoking methods that add extra drill down ability to the EIS.

The situation that is addressed here is when the user had drilled down to the last level of the hierarchy, the variable cannot break down further, but extra information is available and required.

The example in this section is from EDS-Databank's JARS system. This system records time spent on projects. It is a management tool that provides a lot of information on many aspects of EDS-Databank's day
to day running. The EIS used here provides charts that show the number of weeks that a project has been open with no time being put to it.

The drill hierarchy starts at number of weeks, drills down to view each number of weeks by the department leading projects, then drills to the project sponsor, then drills to the project leader and finally to a chart of all projects that fall in the 4 level hierarchy just described. There is nowhere to drill when the list of projects appears, essentially there is no further breakdown of project into lower level entities.

However each project has various attributes associated with it. For example, each project has an entry in a project register detailing such things as estimated cost, project leader, project title, estimated hours, estimated implementation date and other fields. Projects may have a project diary defined, where project leaders and others can enter free flow text about the project. Projects may have had time recorded against them.

These three features -- project register entry, diary text and time, are of interest to our JARS EIS user. From the 5th drill level -- the project (see Figure 1.5) -- I want to be able to allow the user to view these entities.

Again, the custom method is used. I'll continue to use the TRAPHELP.SCL code, but this time I will omit the earlier help code. This custom selection list code should immediately follow the code that controled the help system.

As in the help section, every drill to a new hierarchy variable will cause the custom code to run, and I am only interested in it from the last drilldown variable. Again I have to find a way to identify that variable, and this time as the last variable will be the project id, and every project has its id recorded in the project register, a search of that register will server to indicate if I have reached the last level.

If a search of the project register reveals the current TEXT setting from USERLIST is contained in it, I will call a custom frame entry to present some more options which are specific only to the project clicked on on that last drilldown.

The revised code is:

```%
%eisentry userlist eisrc 3 ;

init:
eisrc = 0 ; /* ensure always exit clean */

<<<< place the MYHELP help code in here >>>
```

660
if getnitemc(userlist,'COMMAND') ne 'DRILL' then return;

/* check if a record is in the project register and the user is trying to drill. If so, carry out custom processing and set the EIS return code to 2 to prevent SAS/EIS default methods running */

text = getnitemc(userlist,'TEXT')

dsid00 = open('library.projlist(keep=project)');
rc = where(dsid00,'project eq ., II text II m');
rc = fetchobs(dsid00,1);
call close(dsid00);

if rc ne -1 then do;
eisrc = 2;
call display(,reports.frame',text);
end;
return;

That's a fairly simple piece of code. Its difficult to make this sort of code generic and independent of the underlying data, as the final drill variable will require different code in the search of the file. Hence each time you wish to do this, the code will change a little.

Figure 1.5 shows the usual SAS/EIS screen that the user has drilled down to. The project id cannot be drilled further. As far as the user is concerned, the screen is a standard EIS screen. When the user clicks on a drill item, they see the screen in Figure 1.6. That screen is a frame entry, called in this case REPORTS.FRAME.

Note that the screen is at the fifth drill level. This can be seen in the data in the header which shows a breakdown for '10 Plus, 743, W, ME' where each of the items between commas is a drill level.

The final step here is to show what the REPORTS.FRAME entry looks like. The entry is just a set of icons that trigger code to show more data about a project. The initial screen that the user sees is in Figure 1.6.

One point to notice about Figure 1.6 is that the header (which is a graphic text class) contains the project id. In other words the header is not static but rather will change for each project that gets selected. This is one of the features of FRAME technology, there is very little in FRAME that is set in concrete when you create a frame entry, many objects...
on a frame can be manipulated at run time to produce a custom screen for different situations.

The code behind the frame entry in Figure 1.6 will change dependent on what your user desires to have happen when they select that final drilldown. The code shown here reflects the three requirements of a look at the project register, a browse of the project diary and a report of time spent on the project. That code for this case is:

```plaintext
entry text $ 6 ;
init:
call notify('header','_set_text_','Project ' || text || ' - Click Desired Action');
return ;
dobrowse:
call fsedit('library.projlist(where=(project="" || text || ""))',
'sasuser.screens.projlist.screen','BROWSE') ;
return ;
dodiary:
call wregion(5,1,20,80) ;
call preview('COPY','SASUSER.METHODS.EXAMPLE.SOURCE') ;
call preview('BROWSE','Project: ' || text || ' - Press PF3 To End') ;
call preview('CLEAR') ;
return ;
dosum:
submit continue sql;
select dept,sum(tothrs) label='Total Hours',
   min(wedate) label='Started' format=date7.,
   max(wedate) label='Ended ' format=date7.
from global.activity(where=(project = "$text")
   group by dept
   order by dept ;
endsubmit ;
return ;
```

**MAKING YOUR SAS/EIS BUSINESS CHART APPLICATION GO FASTER**

There are a few hints that can be employed to speed up an EIS business chart. These include:

- don't force your EIS to summarise data -- create small datasets with the data already summarised.
- don't add too many drilldown levels. 4 or 5 is near a maximum. The reason is that the more drill down levels you have the more likely it is that the underlying dataset is increasing in size causing more I/O at EIS runtime.
- don't have drilldown levels with more than 8-10 values for charts. There are two reasons, one is that as the number of values increases you force SAS/GRAPH to do a lot more work, the second is that the physical space in SAS/EIS that SAS/GRAPH has to fit the chart into doesn't lend itself to lots of values. Using lots of chart values can create a messy difficult to read graph -- precisely the opposite intention of the SAS/EIS product and the EIS concept.
be careful about what you code in custom selection methods. Remember that the more lists you create and the more code you have, the more work the EIS has to do between drills. If you can avoid custom methods altogether you have the optimal EIS -- only running the code that SAS Institute deem necessary.

if data is stored normally on a mainframe but the EIS is on a PC, and you use CONNECT to fetch the data for the EIS, do the data transfer at some time when system traffic is light and move the whole summarised dataset for the EIS onto the EIS platform -- don't use CONNECT at the time the EIS wants the data if possible.

when developing your EIS you may notice system performance getting sluggish during very long sessions, especially under OS/2 and Windows 3.1. The problem seems to be the number of lists created by frame which appear to increase as the session gets longer. This is easy to fix, just exit EIS and all the lists should be removed as they are a SAS/AF entity and SAS/EIS is the SAS/AF application running

you may not always need the drilldown capability in SAS/EIS. Sometimes only one chart or graph is needed to convey the necessary information. Consider creating that graph outside of EIS and store in a GRSEG catalog entry. That way you can display a graph immediately that may take several minutes to create.

SUMMARY

I have attempted to explore some extras that can add functionality to an EIS application. I believe that SAS/EIS is one of the quickest development tools for creating graphic applications on the market today, and my aim here has been to show that that rapid application feature does not prevent developers adding some bells and whistles as necessary.

The customisation features that I have discussed add to the SAS/EIS development philosophy. They are simple tools, and do not cause any major overheads to EIS. They add functionality in that you can now increase EIS userability by customising HELP screens to be more context sensitive, and have an insight into how to make EIS do things it doesn't do by default.

EIS as a concept demands virtually instant display of data on screen. Many sites target the EIS concept to managers, and the concept, along with a few reputations, can fail badly if those managers have to wait more than a few seconds for a supposedly immediate report or graph. I have given a few ideas about how you can speed up the running of your EIS systems.

Finally, a personal opinion about SAS/EIS. Always consider it in the context of an applications generator as well as an EIS. Just because the product is called SAS/EIS doesn't mean that some other application requiring graphics won't benefit in some way. Remember that the product is not just graphics, there are many other application types available. SAS/EIS generates applications, and should rightfully be regarded as a tool for doing that rather than a pure EIS tool.