Extending SAS/EIS® Software
Ann Weinberger, SAS Institute Inc., Cary, NC

ABSTRACT

SAS/EIS software provides objects that are ready to use without modifications. These objects provide most of the information delivery requirements many users need. However, SAS/EIS also enables you to extend, enhance and customize SAS/EIS objects to meet your specific information delivery needs. This paper describes the different ways that SAS/EIS objects may be customized, focusing on techniques available with Release 6.11 of SAS/EIS.

INTRODUCTION

SAS/EIS software provides a development environment and an execution environment for creating and using SAS/EIS applications. SAS/EIS applications are developed using SAS/EIS objects. A SAS/EIS object is a package of data and routines that performs a specific function. SAS/EIS objects enable you to create applications, using the CREATE method, and execute applications, using the RUN method. The CREATE method gathers information about the EIS application and stores it on an application list. The RUN method reads the stored application list and displays the requested data or performs a specific function, (such as executing a command).

By working within the SAS/EIS software development environment, you can use SAS/AF® software, FRAME entries, Screen Control Language(SCL) and SAS/AF CLASS libraries to modify the RUN and CREATE methods of existing SAS/EIS objects or to create new objects. You can use one or more of the following techniques:

- modify the RUN method of SAS/EIS objects
- replace methods in existing SAS/EIS objects
- use event handling to communicate between SAS/EIS objects on a Graphics Menu Builder
- develop new SAS/EIS objects
- convert existing applications into SAS/EIS objects

RUN METHODS

SAS/EIS software provides a RUN method for each object. The RUN method reads the application list and executes the application, displaying the requested data or performing a specific function. The new objects introduced in Release 6.11 of SAS/EIS use a common RUN method: SASHELP.EIS.RUNCEN.FRAME. The common RUN method provides the following functionality:

- Finds the application record on the application database and loads the application list
- Instantiates the viewer passing it the location of the application list
- Sets the pull-down menu and the status of individual items on the pull-down
- Controls the display of the window including the title, background color and pull-down menu
- Processes common SAS/EIS commands such as Notes, Bookmark, Save, Save as and End

The common RUN method assumes that the application list may contain the following items:

- VIEWER          Class name of the viewer, required
- WINTYPE         Type of window:
- STD or DIALOG
- MENU           Pull-down menu name
- NOTES          Notes file name
- HELP           Help file name
- BACKGROUND  Window background color

A viewer is an object or class that displays data. Viewers may be a subclass of a single widget or a composite of many widgets. The viewer interprets user actions such as single and double clicks or right mouse button popups. In some cases, the viewer is attached to a data model which provides the data for the viewer to display. For example, the Multidimensional Report object uses the Table Editor as its viewer and uses the Summary object as its data model. Other SAS/EIS objects, such as Graphics with Hotspots, provide only a viewer.

Viewers used by the common RUN method must have a minimum number of methods and instance variables. Required methods are:

- _INIT_ Initialize the viewer from the application list
- _EXEC_CMD_ Process commands, if applicable
- _UPDATE_APPL_ Save updates to the viewer, if applicable
- _OBJECT_LABEL_ Interpret user actions

Required instance variables are:

- CATLOC Catalog entry for application list
- PMENU_STATUS List of menu ids and their status. Negative for grayed; positive, ungrayed.

Using the common RUN method with a viewer provides certain advantages:

- Common SAS/EIS commands are handled automatically
- Code that appears in every RUN method is written and tested once
- RUN method is consistent between objects
- SAS/EIS objects developed with a viewer can be placed on the Graphics Menu Builder (EIS DESKTOP) without rewriting any code

EXTENSIONS TO THE RUN METHOD

Defining a Custom Pull-down Menu

You can make minor enhancements to a SAS/EIS object by adding or changing the functionality of a method. One of the simplest and most useful enhancements you can make is to customize the pull-down menu of a SAS/EIS object. For example, you want to add the command, MAGNIFY, to the Graphics with Hotspots object. This command will toggle between displaying the graph with 100% magnification and 200% magnification. The steps to do this are:

1. Create a method called MAGNIFY.
2. In the method, add code to toggle between magnifications.
3. Add the command to the pull-down menu.

You can test your enhancements by running the Graphics Menu Builder (EIS DESKTOP) and seeing the changes in the menu.
Define the new menu using PROC PMENU adding the Magnify item under the View pull-down.

%let sasystem = 64;
proc pmenu = suglappi;
  menu hotspotg;
  item 'File' menu=file; item 'View' menu=View; item 'Help' menu=help;
  menu file;
  item 'Print...' selection=printid id=2004;
  separator;
  item 'Bookmarks' selection=bookmark;
  item 'End' selection = end ;
  selection print 'printit';
  selection bookmark 'bookmark';
  selection end 'END';
  menu view;
  item 'Show Hotspots' selection=showhot id=2001;
  item 'Hide Hotspots' selection=hidehot id=2002;
  item 'Magnify' selection=magnify ;
  selection magnify 'magnify';
  selection showhot 'showhot';
  selection hidehot 'hidehot';
  menu help;
  item 'Notes' selection=notes id=2003;
  separator;
  selection notes 'notes';
  run;
quit;

Create an override to the _EXEC_CMO_ method to define the action that will occur when the MAGNIFY command is received. When the RUN method finds a command it does not know how to handle, it calls the _EXEC_CMD_ method, passing in the command. In the override, you process the command if it is MAGNIFY; otherwise, do a call super.

execomd: method cmd $ rc 6;
  if upcase(cmd) = 'MAGNIFY' then do;
    call sendLselL.'_GET _MAGNIFICATION.',magnify);
    if magnify = 100 then do;
      contorg = 'n';
      magnify = 200;
    end;
    else do;
      magnify = 100;
      contorg = 'y';
    end;
    call sendLselL.'_SET _CONTORL',contorgj;
    call sendLselL.'_SET _MAGNIFICATION_',magnify);
  end;
  else
    call superL •• IL'_EXEC_CMD_',amd,ra);
endmethod;

Define a Graphics with Hotspots application, specifying a graph. Turn on the scroll bars.

Change the menu to use the pull-down menu created in the first step. The Advanced window (Figure 1), accessible from the CREATE method, allows you to customize the appearance of the RUN method including the background color, menu, help and Notes files.

Specify the override for the _EXEC_CMD_ method in the Methods window. The Methods window (Figure 2) can be reached from the Advanced window. In this window, you can specify an override for a number of methods, including the _EXEC_CMD_ method. Select the method to override and specify the catalog entry where the override exists. You may specify when the override will execute with relation to the original method: before, after or override. For this example, select override.

Test the application. Select View -> Magnify from the pull-down menu to try out the new command.

Overriding a Viewer Method

Another simple modification is to override a method to change the viewer's behavior. For example, the Graphics with Hotspots object has a method, _FILL_, which is used to fill the viewer based on items found on the application list. The _FILL_ method looks for the name of the graph to display in the item, GRSEG, on the application list. You can build an application with the Graphics with Hotspots object and specify no GRSEG at build time. The name of the GRSEG can be filled in by an override of the _FILL_ method. The steps to accomplish this are:

Define the override to the _FILL_ method. In the override, fill in the GRSEG item on the application list with the graph that you want to display. Then do a call super.

tilt: method applist errcode 8;
  grseg = "sashelp.eisgraph.world2.grseg";
  rc = setiteml(applist,grseg,'GRSEG');
Applications Development

```plaintext
• Build a Graphics with Hotspots application specifying no GRSEG. In the Methods window, specify the override of the _FILL_ method. Specify Override for When to Execute.

• Test the application, verifying that the correct graph is displayed.

Adding a Method to a Viewer

Another extension to the RUN methods and viewers is to add a method to an existing viewer. For example, instead of having the Magnify command from the first example toggle the magnification, you would like to display a window that allows the end user to select the magnification to use. You could code this as a simple SAS/AF Frame entry that is called by the _EXEC_CMD_ method. However, to illustrate adding a method, we will code it as a method of the viewer. The steps to add and use a new method are:

• Define the new method, _SELECT_MAGNIFY_. Create a frame (in this example, SUGI.APPL.SETMSG.FRAME shown in Figure 3) that has a slider that allows the user to select the desired magnification.

Figure 3 _SELECT_MAGNIFY_ Method

• Create the SCL for the new frame. In the term of the frame, apply the magnification to the viewer.

```
init:
  call send(_self_,'get magnification',magnify);
  call notify('slider','set value',magnify);
  return;

term:
  if _status_ = 'C' then
    return;
  call notify('slider','get value',magnify);
  if magnify = 100 then
    content = ''; 
  else 
    content = 'n';
  call send(_self_,'set content',content);
  call send(_self_,'set magnification',magnify);
  return;

slider:
  call notify('slider','get value',magnify);
  return;

magnify:
  call notify('slider','set value',magnify);
  return;
```

• Create an override to the _POSTINIT_ method. In this override, use the _SET_INSTANCE_METHOD_ to add the new method to the viewer.

```
postinit:
  call super(_self_,'postinit_');
  call send(_self_,'set instance method_','select magnify_','sugi.appl.setmsg.frame');
endmethod;
```

• Create an override to the _EXEC_CMD_ method. When the command is MAGNIFY, execute the _SELECT_MAGNIFY_ method; otherwise, do a call super.

```
execcmd1: method cmd $ encode 8;
  if upcase(cmd) = 'MAGNIFY' then
    call send(_self_,'select magnify_');
  else
    call super(_self_,'exec cmd',cmd,encode);
endmethod;
```

• Build a Graphics with Hotspots application, specifying a graph to be displayed. Turn on the scroll bars.

• In the Advanced window, specify the pull-down menu used in the first example (SUGI.APPLHOTSPOTG.PMENU).

• In the Methods window, specify the overrides for both the _POSTINIT_ and _EXEC_CMD_ methods. Select Override for When to Execute for both methods.

• Test the application. Select Magnify from the View pull-down and select a new magnification.

Defining a New Viewer for an Existing Object

It is also possible to create new viewers for existing SAS/EIS objects. A new viewer may be any combination of objects, including existing viewers, so long as the minimum required methods and instance variables are provided for the common RUN method to use.

For example, suppose you want to have a Graphics with Hotspots application that not only displays the graph selected but also displays a company logo and a slider to allow the user to select a magnification for the graph. The steps to define and use a new viewer are:

• Create a composite widget that contains the logo, slider and original graphics viewer. To do this, you will need to create a new resource that contains the appropriate widgets. Copy the resource SASHELP.FSP.BUILD.RESOURCE and add the class, SASHELP.EIS.HOTGRAPH.CLASS.

• While defining the new viewer, add the instance variables: CATLOC and PMENU_STATUS in the Instance Variable Editor. CATLOC is a character variable with a length of 35. PMENU_STATUS is the id for a list.

• While defining the new viewer, override the _INIT_ and _OBJECT_LABEL_ methods in the Methods Editor. Also, under Actions, select Delegates to invoke the Delegates...
window. Set the Graphics with Hotspots object as a delegate of the composite. This will allow the original viewer to receive any methods that the composite does not have. (NOTE: You will not be able to name the Graphics with Hotspots Viewer within the composite. You will need to use the RM DESCRIBE command to determine the object name.)

- Create the override to the _INIT_ method. Use the CATLOC instance variable to fill in the application list. Then call the _FILL_ method of the original Graphics with Hotspots viewer. Get the magnification and set up the slider.

```plaintext
length catloc $ 35 current $ 8 contsort $ 1;
init: method;

call super(_self_,'_INIT_');
applist = makealist();
rc = fillist('eis',catloc,appllist);
call send(obj1,'_fill_','appllist',rc);

pmenu_status = getnitem(obj1,'PMENU_STATUS',1,1,0);
call send(obj1,'_get_magnification_','magfactor');
call send(slider,'_set_value_','magfactor');
call send(magnify,'_set_value_','magfactor');
endmethod;
```

- Create the override to the _OBJECT_LABEL_ method. If the slider was selected, get the new magnification and set it on the graphics viewer; otherwise, do a call super.

```plaintext
objlabel: method;

call send(_self_,'_get_current_name_','current');
current = upcase(current);
select(current);
when('SLIDER') do;
call send(slider,'_get_value_','magfactor');
call send(magnify,'_set_value_','magfactor');
end;
when('MAGNIFY') do;
call send(magnify,'_get_value_','magfactor');
call send(slider,'_set_value_','magfactor');
end;
otherwise;
end;
if magfactor = 100 then
  contsort = 'y';
else
  contsort = 'w';
call send(obj1,'_set_contsort_','contsort');
call send(obj1,'_set_magnification_','magfactor');
call super(_self_,'_objct_label_');
endmethod;
```

- Build a Graphics with Hotspots application, specifying a graph to display and assign some hotspots. Turn on the scroll bars. In the Advanced window, specify the new viewer.

- Test the application. Verify that the hotspots still work. Select a new magnification and verify that it was applied.

**EVENT HANDLING**

In SAS/EIS software, event-handling is used to communicate between some objects on the Graphics Menu Builder. When a SAS/EIS object executes a method that changes its status or state, the object issues an event. For example, when a user selects a hotspot on a Graphics with Hotspots object, the object issues the SELECT event. The event describes what changed and includes a list of parameter values that changed.

In addition to issuing events, an object can receive any event that it issues. When an event is received, a method of the receiving object is executed. Objects placed in a Graphics Menu Builder do not automatically receive events. You must link the objects and specify which events an object will receive. In addition, you must specify the method of the receiver that will execute when the event is received. This ability to issue and receive events allows objects to communicate with each other.

Some SAS/EIS objects can handle events with no additional code developed by the SAS/EIS applications developer. For example, if a 3D Business graph and a Multidimensional report are built using the same data set or table and the same hierarchy, they can communicate with the DRILLDOWN and NAVIGATE events. If linked in a Graphics Menu Builder, as one drills down or navigates through the hierarchy, the other application will receive the events and drill down or navigate.

However, other SAS/EIS objects require that additional code be developed to handle the event that is received. For example, you have defined a Graphics with Hotspots application that displays a map of the United States. Various states are hotspotted. However, no target application is assigned to the hotspots. Instead, you will be placing the application on a Graphics Menu Builder and linking it to a Catalog Entry with Hotspots application. As hotspots on the map are selected, the report displayed in the Catalog Entry viewer will change accordingly. In order to accomplish this, the 2 applications will be linked with events. The Graphics with Hotspots object will issue the SELECT event when the user clicks on a hotspot and the Catalog Entry with Hotspots object will execute the _CHANGE_ method when the event is received. The steps to accomplish this are:

- Create the override to the _CHANGE_ method that will execute when the Catalog Entry Viewer with Hotspots receives the SELECT event. Get the name of the selected hotspot and decide which report to display.

```plaintext
length name $ 8 output $ 35;
change: method infoist $;
  name=getnitem(infoist,'name',1,1,'');
  output='wgui.app'.trim(left(name))'f'.output;
  if exist(output) then
    call send(_self_,'_select_entry_','output');
  endmethod;
```

- Define the Graphics with Hotspots application. Specify the map to display and assign hotspots to various states. Do not assign target applications.

- Define the Catalog Entry with Hotspots application. Do not specify a catalog entry to display. In the Methods window, override the _CHANGE_ method, specifying the location of the override and selecting Override For When to Execute.

- Build a Graphics Menu Builder. Drag out regions for both applications and fill them with an EIS application, selecting the appropriate applications from the table of EIS applications.
Applications Development

- The Object attributes of the Graphics with Hotspots application displays the Events window (Figure 4). In this window, select the SELECT event for the Catalog Entry with Hotspots application to receive. Verify that the method to execute is the _CHANGE_ method.

![Figure 4 Events Window](image)

- Test the Graphics Menu Builder application. Select a state and verify that the correct report is displayed in the Catalog Entry viewer.

**DEFINING NEW OBJECTS WITH VIEWERS**

In previous sections, we have looked at methods for extending the functionality of existing SAS/EIS objects and viewers. Depending on your application requirements, each of these methods is an efficient solution to the problem. In some situations, however, you may want to create a new SAS/EIS object that uses the common RUN method with a viewer. In this section, we will look at creating a viewer for the Briefing Book example that was presented in SAS Technical Report P-253, User-written Methods and Objects in SAS/EIS Software, Release 5.08.

SAS/EIS objects that use the common RUN method do not require any special processing in the CREATE method except for the additional items that are required on the application list. These items: VIEWER, WINTYPE, BACKGROUND, MENU, NOTES and HELP, are used to set up the run-time frame. Of these, only the VIEWER is required. The others, if missing, will be set to a default value. If the CREATE method has the RUN method hard-coded it should be set to SASHELP.EIS.RUNGEN.FRAME. On the object database, the RUN method should be set to SASHELP.EIS.RUNGEN.FRAME.

The majority of the changes need to be made in the RUN method. Most of the functionality contained in the RUN method will be handled by methods of the new viewer. In general, you can map RUN method functionality to viewer methods as follows:

- Loading the application list and setting up the initial display: _INIT_ method; optionally a _FILL_ method
- Command processing from the MAIN label: _EXEC_CMD_ method
- Processing performed in an object's label within the SCL: _OBJECT_LABEL_ method
- Clean-up when ending: _TERM_ method

Examples in the book, SAS Technical Report P-253, User-written Methods and Objects in SAS/EIS Software, show how an EIS object can be converted to a SAS/AF subclass. This allows the EIS object to be placed in the Graphics Menu Builder (EIS DESKTOP) with other objects. However, this is not required for EIS objects that have viewers. The viewer may be placed on an EIS DESKTOP simply by filling a region with an EIS application and selecting the appropriate application. No further development is needed.

**Briefing Book Overview**

The basic briefing book object enables users to define applications that display a series of GRSEG catalog entries created with SAS/GRA?H software. The object includes navigational arrows that enable users to move successively through images one at a time, forward or backward. In SAS Technical Report P-253, User-written Methods and Objects in SAS/EIS Software, a CREATE method is developed that stores the following items on the application list:

- METHOD Numeric Indicates where graphs are stored
  - 1=data set
  - 2=list
- DSNNAME Character Data set containing graphs, if applicable
- VARIABLE Character Data set variable containing graph name, if applicable
- LIST Numeric List containing graph names, if applicable
- MENU Character Pull-down menu

The RUN method is a frame that displays a graph in a SAS/GRA?H_output object along with two navigational arrows. The bottom arrow allows you to move forward through the specified graphs. The top arrow allows you to re-display a previously displayed graph. The only commands allowed from the menu are Goback, Bookmark and Help.

**CREATE Method Changes**

For this example, the CREATE method for the briefing book example requires very little modification. Insert the following code when saving the application list to add the VIEWER, WINTYPE and BACKGROUND items to the application list:

```plaintext
rc = setnitemc(applist,'SUGLAPPLBRIEECLASS', 'VIEWER');
rc = setnitemc(applist,'SASHELP,FSP,MOREB,PMENU', 'WINTYPE');
rc = setnitemc(applist,'SECONDARY BACKGROUND', 'BACKGROUND');
```

Define the RUN method and CREATE method in the INIT label:

```plaintext
editpgm = '6LDBRF.FRAME';
rungm = 'SASHELP,EIS.RUNGEN.FRAME';
```

In addition, remove the Methods push-button and subsequent call to the Methods program window.

**Object Database Changes**

In the Object Manager, update the object database containing the Briefing Book object to point to the common RUN method, SASHELP.EIS.RUNGEN.FRAME.
Defining the New Viewer for the Common RUN Method

For the briefing book, define a viewer that is a composite of widgets. The composite contains a SAS/GRAPH Output object and two control arrows. The steps are:

- Build the composite, SUGI.APPL.BRIEF.CLASS. The parent class is SASHELP.FSP.COMPOSIT.CLASS.
- Edit the attributes of the class and create the SAS/GRAPH Output object, GRAPH, and the two control arrows, UP and DOWN. Add Region Attachments, if desired.
- In the Methods editor, override the _INIT_ and _OBJECT_LABEL_ methods.
- In the Method editor, add the NEXT_GRAPH method. This method will execute when the user scrolls up and down within the graphs to display.
- The viewer will not have an _EXEC_CMD_ method because the only commands that are allowed, Goback, Bookmark and Help, are handled by the common RUN method, SASHELP.EIS.RUNGEN.FRAME.
- In the Instance Variable editor, add the instance variables CATLOC, METHOD, DSNAME, VARIABLE, LIST and INDEX.

_Init_ Method

The_Init_method loads the application list, reads items from the list, stores application items in instance variables and then sets up the first graph, control arrows and INDEX instance variable. This functionality was performed within the INIT label of the original RUN method. Some of the functionality that was originally performed within the INIT label is not done in the_INIT_method of the viewer. The common RUN method, SASHELP.EIS.RUNGEN.FRAME, handles these functions. This includes determining the catalog location of the application list and setting up the bookmark.

```plaintext
length catloc $ 35 current $ 8 dsname $ 17 variable $ 8;

init: method;
    call super(_self_,'_init_');
    applist = makelist();
    rc = filllist('eis',catloc,applist);
    method = getitem(applist,'METHOD');
    dsname = getitem(applist,'DSNAME');
    variable = getitem(applist,'VARIABLE');
    list = copylist(getitem(applist,'LIST'))['y',list];
    index = 1;
    rc = delalist(applist,'y');
    call send(_self_,'_next_graph');
endmethod;

_Object_Label_ Method

The override of the _OBJECT_LABEL_ method handles the functionality provided by the UP and DOWN labels of the original RUN method. In the original RUN method, the UP label was executed when the up arrow was selected and the DOWN label when the down arrow was selected. In the new RUN method, when the down arrow is selected, the value of INDEX is decremented and the NEXT_GRAPH method is called. When the up arrow is selected, the value of INDEX is incremented and the NEXT_GRAPH method is called.

objlabel: method;
    call send(_self_,'_get_current_name',current);
    current = upcase(current);
    select(current);
        when('UP') index = index - 1;
        when('DOWN') index = index + 1;
        otherwise;
    end;
    call send(_self_,'_next_graph');
    call super(_self_,'_object_label');
endmethod;

_Next_Graph Method

Using the value of INDEX and METHOD, this method reads the next graph to display either from the data set or from the list. The method also decides whether the UP and/or DOWN arrows should be grayed, based on the value of INDEX. This functionality was provided by the NEXTDS and NEXTLIST labels of the original RUN method.

nextgraph: method;
    if method = 1 then do;
        graphname = getitem(getitem(list,index),1);
        if index = listlen(list) then
            call send(down,'_gray_');
            else
                call send(down,'_ungray_');
        end;
        else do;
            dsid = open(dsid);
            rc = fetchobs(dsid,index);
            if rc = 0 then
                graphname = getvarc(dsid,varnum(dsid,variable));
                nobs = attm(dsid,'NOBS');
                if index = nobs then
                    call send(down,'_gray_');
                else
                    call send(down,'_ungray_');
            rc = close(dsid);
        end;
        call send(graph,'_set_graph_',graphname);
        if index = 1 then
            call send(up,'_gray_');
        else
            call send(up,'_ungray_');
    endmethod;

Testing the EIS Object

Now that the CREATE method and viewer have been defined, the EIS object can be tested. In SAS/EIS, build a Briefing Book application. Select the Enter values... Method and enter some graphs to display. Test the application, verifying that the control arrows scroll through the graphs. You may also want to create an EIS DESKTOP application that uses the Briefing Book application to test that the viewer works within the EIS DESKTOP.

CONCLUSION
Applications Development

SAS/EIS software provides a variety of objects that can be used to develop complex EIS applications. For your application, these objects may not require any modifications. If, however, your specific application needs require customizations, SAS/EIS provides a development framework that enables you to extend and enhance existing objects. You may also use the development environment to provide your own custom EIS objects.

REFERENCES


SAS, SAS/EIS, and SAS/AF are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

The following files have been stored on SAS Institute's Internet gateway:

appl.sct01 - Catalog with all examples. Assign the libref SUGI to use the examples.
viewer.txt - Summary of viewer requirements.

You can download these files if you have access to the Internet. To download these files, connect to ftp.sas.com. Once you are connected, enter the following responses as prompted:

Name (ftp.sas.com:userid): anonymous
Password: your e-mail address

All SUGI 21 files are stored in the following directory:

/pub/sugi21

There is one subdirectory for each paper that has ancillary files. Download the following file in /pub/sugi21 for a complete index of all files in /pub/sugi21:

README.index

The file README.index has a description of each directory. The description will contain the title of the paper and the directory name where the files are stored.

You should use a binary file transfer when transferring SAS data sets, SAS catalogs, and other binary files.

Ann Weinberger
SAS Campus Drive
Cary, NC 27513
919-677-8000 ext. 6607
sasaaw@unx.sas.com