Subclassing the FRAME Class to Incorporate Documentation Templates and Standard Widgets.

Mark Bodt, SUNKEN TREASURE SOFTWARE SYSTEMS LIMITED, New Zealand

Written for SUGI 22 Coders' Corner.

Abstract

When a new FRAME entry is created using the build window, SAS® uses the FRAME class that is provided with SAS/AF software.

Standardising the look and feel of an SAS/AF application is one factor that determines how intuitive an application is. Frames often have common components such as OK, Cancel, Help buttons etc. The layout and positioning of these is important to ensure uniformity across different screens of an application. Features of SAS/AF release 6.11 (aka Orlando) promote Object Oriented Application Design and allow developers to develop their own classes to ensure that visual standards are easily incorporated into an application.

The SAS/AF FRAME class is one object that can be subclassed to aid developing applications that have a uniform look & feel. This paper will discuss the subclassing of the FRAME class, working through an example where on creating a new frame entry, the type of FRAME (eg dialog box) and the population of widgets in the FRAME will be demonstrated. Initializing the new FRAME's SCL entry with a standard template will also be discussed.

By subclassing the FRAME class in this manner, development time can be greatly reduced as it is not necessary to define the standard widgets and frame attribute settings each time a new frame is created. Uniformity of screens is also encouraged by initialising the screen layout.

The examples in the paper will detail all the steps and code required to subclass the FRAME class.

Scope

This paper covers topics discussed during the 20 minute Coders' Corner presentation. The basics of subclassing the FRAME class are outlined. However it is not intended to thoroughly discuss the topic. I am hoping to write an article for a future Observations® issue which cover this topic in depth.

Introduction

When a new FRAME entry is created in the Build window, SAS uses the FRAME class. By default, a blank FRAME is displayed and the SCL entry is empty. This default behaviour can be overridden by subclassing the FRAME class. This paper discusses the basics of creating a subclass of the FRAME Class. The subclass will allow the developer to specify whether the new FRAME is to initialise as the default blank FRAME, or as a dialog box with an OK widget. The subclass will also initialise the SCL window with a standard SCL template, containing a documentation header etc. As the time set for the paper is only 20 minutes, the examples will be kept very simple.

Creating a subclass of the FRAME class.

For the purposes of the example, a libname of 00P and a catalog called 00P.OP are created. In the build window, a new class is created by using the pull-down menu options File-New-Entry. The following dialog is displayed:

Enter SUGI for the Entry name & Class for the Entry type. Then click on OK. The following dialog is displayed:

Methods

FRAME developers will be familiar with the use of methods. In Object Oriented Programming, methods are used to initiate actions on an object. For example to hide an OK button on a FRAME, the following code could be used:

Call Notify('OK', '_hide_');

In this case the method that is used is the _hide_ method. It is not the intention discuss OOP concepts in this paper, but it helps to understand how the objectives will be achieved in this example.

Each Object comes with standard methods. The FRAME class also has methods. To achieve our objectives, the default behaviour of the FRAME class when a new FRAME is created needs to be changed. The FRAME class has several methods that are used at build time. These methods have a prefix of _B_. The methods are:

- **_BINIT_** Which runs when the build window for the FRAME is initialised.
- **_BPOST_** Which runs after the build window has been created, i.e after the _BINIT_ method has run.
- **_BTERM_** Which runs when the Build window for the FRAME is closed.
- **_BUPDATE_** Which runs after the GATTR (General Attributes) window for the FRAME is closed.
Overriding the _BPOSTINIT_ Method.

In order to modify the behaviour of the FRAME class on creating a new FRAME entry during a build, the _BINIT_ or _BPOSTINIT_ method needs to be modified. In the example, the _BPOSTINIT_ method is modified. In the dialog displayed above, click on Methods. The Methods dialog is shown as follows:

![Methods dialog](image)

All the methods that are applicable to the FRAME class are shown in the Methods List Box. Click on the _BPOSTINIT_ method. In the Source entry, enter a simple message dialog as shown below:

The dialog contains an Information icon, stored as a catalog image entry, a text widget and an OK pushbutton. For the example, we want to create this type of dialog when creating a new FRAME entry. The easiest way to do this is to create and populate the frame with widgets, exactly as what you require. This includes positioning, colours, banner settings etc.

Tip: An easy way to set the dialog position, and screen size attributes is by using the setwsz command. Follow these steps:
1. Resize the frame to the size you require by clicking & dragging the lower right hand corner of the frame.
2. Drag the frame to the position that you require on the screen by clicking & dragging on the window's title bar.
3. When the frame is the size required and in the correct position on the screen, enter the command setwsz. This will set the appropriate General Attributes (GATTR) for the frame.

Note down the window size attributes from the General Attributes dialog. In this example they were:
- Row Start: 7
- Column Start: 14
- Number of Rows: 10
- Number of Columns: 66

To keep this example simple, only one widget will be created in the _BPOSTINIT_ method, which will be the OK button. When the OK button is created in the _BPOSTINIT_ method, the coordinates of its position on the frame need to be specified. This can be determined by counting the number of rows, columns etc, but an easy way is to position the OK button where it is required in the frame as in above dialog, and write some simple code that returns the coordinates. In the SCL window for the above dialog, the following SCL is entered:

```scl
Init:
/*this is some temporary code to demonstrate how to determine the coordinates for a widget. In this example we will get the coordinates for the OK button. */
*create a temporary list;
region_list=makelist();
*get the region attributes and place them in the temporary list;
call notify('ok', 'get region', region_list);
*put the list to the message window so we can examine the contents;
call putlist(region_list, 'This is the region list for the OK button');
*delete the temporary list;
if listlen(region_list) > 0 then
  re=deallist(region_list);
Return;
```

Testing the code lists the OK buttons region list in the message window as follows:

This is the region list for the OK button
- ULX=28
- ULY=5
- LRX=44
- LRY=8
- UNITS='POINTS'
- PARENT=''
- TRANSPARENT='Y'

... etc ...

Of interest are the coordinates of the OK button which are listed as:
- ULX (Upper Left X)
- ULY (Upper Right Y)
- LRY (Lower Right Y)
Note down these values.
Those are all the settings that are required from the dialog. The
dialog is no longer needed as it is not used by the subclass that is
being created. This SCL & Frame entry can be closed.

Creating the Selection Dialog.
When a new FRAME entry is created, the user is to be prompted to
specify what type of frame is required and what widgets are required
on the new frame. This is done by using a selection dialog. This
dialog will now be built. A simple dialog is created as follows:

The widgets are as follows:
- A radio box called Type which returns N for Normal & D for
  Dialog
- A Check Box called Okbutton which returns Y for selected or N
  for deselected.
- A Pushbutton called OK.
The SCL for the screen is shown below:

```scil
entry type okbutton $;
Init:
  *hide check box as required;
  link type;
Return:
Main:
  Return:
Term:
  Return:
type:
  if type='D' then call notify('okbutton','_unhide_');
  else call notify('okbutton','_hide_');
return;

Compile the frame and close it.

Writing a new _BPOSTINIT_ method.
Open the SCL window for the overridden _BPOSTINIT_ method.
(Refer to ‘The _BPOSTINIT_ method’ section on the previous
page). The SCL code that follows is to be entered in this SCL
window. Note that for the purpose of explaining the code, the code
will be discussed in sections.

*define lengths;
  length frame classnamex $15 catnamex $1;
  _method_ $200 frametype ok $1;
/* avoid compilation warnings */
  _self_ = _self_; /*special variable;
  _frame_ = _frame_; /*special variable;
  _method_ = _method_; /*special variable;
  fo = fo;
  classnamex = classnamex;

In this first part of the SCL entry, the lengths are set for several
variables. To avoid compilation warnings stating that a variable has
not been initialised, some variables are initialised to themselves. The
variables _self_, _frame_, and _method_ are special variables that
are automatically set to the appropriate settings when the SCL
program is executing.

_BPOST_ : method;
  *call parent class method;
  _method_ = _method_; /*special variable;

/*if the frame is new then we need to get the user to
   select the required attributes by calling a dialog;
   we can detect if the frame is new by checking if it
exists in the catalog. */
  *get the name of the new frame;
  call send(_self_, '_GET_NAME_', frame);

The label that was specified for the overridden _BPOSTINIT_
method was BPOST. In actual fact, this code adds to the original
method. For this reason the original method is called with the
call super(_self_, '_bpostinit_') line.

As the overridden _BPOSTINIT_ method runs every time a frame is
opened, it is necessary to determine whether the _BPOSTINIT_
method is running for a new frame or for an existing frame that is to
be edited. The creation of the new frame is only to be done for a new
frame. Whether the frame is a new frame of an existing one is
determined by checking whether the frame entry exists in the
catalog. The _get_name_ method returns the 4 level catalog name of
the frame.

*check if frame does not exist then this is a new frame; if
  not exist(frame) then do; /*New Frame;*'

*prompt user as to what type of frame they require;
  call display('seltype.frame',frametype-ok);

*initialise frame as selected;
  if upcase(frametype)='D' then do;
    *set frame type & size for a dialog window;
    call send(_self_, '_set_window_size_','7,14,10,66);
    call send(_self_, '_set_window_type_','DIALOG');
    *switch off command line;
    call send(_self_, '_set_window_banner_','NONE');
    *set the title of the window;
    call send(_self_, '_set_title_','SUGI Frame Example');
    *set the background colour of the window;
    call send(_self_, '_set_color_','background','grey');
    /*This is just a sample of the frame attributes
     that can be set. Many other attributes can be
     set here. */

In this section of code, if the frame entry does not exist in the
catalog, then the frame entry is new.
The user is prompted to select the type of frame and whether an OK
button is required by calling the selection dialog built previously
(seltype frame ). This returns two variables being:
  * frametype D=Dialog N=Normal
  * OK=Y=Yes an OK button is required, N=No not required.
If the user selected a normal frame, then no frame attributes are
altered. If the user selected Dialog (D) then several frame attributes
need to be set. These are set using methods, which programmatically
change the settings in the General Attributes (GATTR) dialog of the
frame. Note the settings in the _set_window_size_ of 7,14,10,66.
These are the window size settings that were noted in the section
‘Building a Sample Dialog Window’. Only the basic attributes have
been set in this example. There are numerous others that can be set
if required.

if ok='Y' then do; /*create OK button selected;
  *create a list that contains the button's attributes;
  attr_list=makelist();
  *create a list that contains the button's region
    attributes; 
  region_list=makelist();
  *the region list is a named sublist of the
    attributes list;
  rc=setitem(atr_list,region_list,'_region_');

*button coordinates:
  ulx=28;
  uly=5;
  lrx=44;
  lry=8;
Creating the SCL template.
Most developers have their own standards for the layout of their SCL code, which may include a header describing the purpose of the code, who wrote the code etc. The second half of the _BPOSTINIT_ code, which may include a header describing the purpose of the method looks after the initializing of the new SCL screen from a template.

The above section of code creates the two lists, inserts the _region_ list into the attribute list and creates named list items for the coordinates. Note that the pushbutton coordinates are the values that were noted in the section 'Building a Sample Dialog Window'.

```
/*assemble new SCL entry name;
  scl=substr(frame,1, length(frame)-5) || 'SCL';
/*if the SCL entry does not exist then create it
  based on the SCL template. */
if not exist(scl) then do;
  rc=copy(scl_template, SC1, 'CATALOG');
  if rc ne 0 then do; *copy failed – error handling;
    sysmsg=sysmsg();
    put 'ERROR: New SCL entry ' scl
     ' could not be created.';
    put ' The system message was: ' sysmsg;
    end; *copy failed – error handling;
else do;
  *rename SCL entry description;
  scl=scan(scl,3, '.') ; /*get one level name*/
  *rename scl, SC1, 'CATALOG', 'SCL entry for ' ||
    scan(frame,3, '.') || '.FRAME' ;
end;
```

The name of the SCL entry associated with the frame is built by using the four level frame name, stripping of the FRAME part of the name and replacing it with SCL. The new SCL entry name is checked to ensure that it does not exist, to avoid overwriting an existing entry. If the entry does not exist then the template is copied to the name of the new SCL entry. Lastly, the description for the SCL entry is set to 'SCL entry for...FRAME'.

Once the method code has been entered, compile it and close the SCL & the Methods window. In the Class window, click on Instance Variables. The Instance variables dialog is displayed as shown below:

```
endmethod;
```

The list box lists the standard instance variables for the FRAME class. In this example, an additional instance variable is required: SCL_TEMPLATE. Click on Actions-Add mode on and enter the values for Name (scl_template) and Value (scltempl.scl) as shown below. Ensure that Type is set to character.
Once these settings have been completed, OK out of the instance variables dialog and also out of the Class entry window.

This concludes the creating of the SUGI FRAME subclass. The last two things that need to be done are to put together an SCL template and to define the class in the Resource entry.

The SCL Template.
The SCL template is simply an SCL entry that contains any code that is to be used as a template in the new SCL window when creating a new FRAME entry. The entry must have the same name as specified in the instance variable \texttt{SCL\_TEMPLATE}. i.e \texttt{SCL\_TEMPL\_SCL}

The contents of the SCL entry \texttt{SCL\_TEMPL\_SCL} for the example is:

```
* *******,***,, * *,*,,******.* *****,**,,**** * ****,******,* ;
* Sunken Treasure Software Systems Limited. *;
* --------------------------------------.–– *;
* 73 Pine St, Mt Eden, Auckland, New Zealand. *;
* Ph: (025) 725 386 Fax: +64 9 620 9079 *;
* Email: markbodtl@stss.co.nz *;
* SAS Institute (NZ) Quality Partner *;
* *;
* Function: *;
* *;
* *;
* Written by: M.R.Bodt Date: *;
* *;
* Modifications: *;
* *;
*define lengths;
/* avoid compilation warnings */
Init:
Return;
Main:
Return;
Term:
Return;
```

Registering the New Subclass in the Resource Entry.
For the example, a copy of the default resource entry will be made and the copy modified, so as not to affect the original resource entry. The resource entry \texttt{sashe/p.fsp.build.resource} is copied to the catalog where the SUGI FRAME class has been developed: \texttt{oop.oop}

In the build window, double click on the build.resource entry in the \texttt{oop.oop} catalog. The resource window is displayed as shown below:

To add the new SUGI FRAME class to the resource entry, click on \textit{Actions-Add} Select \texttt{OOP.OOP.SUGI\_CLASS} and click on \textit{OK}. The class is then added to the bottom of the list in the list box. Scroll to this entry as shown below:

Select the SUGI FRAME class by single clicking the list box entry, then click on \textit{Actions-Set active}. By setting the SUGI FRAME class as active specifies that this class is to be used when opening a build window for a FRAME entry. Close the resource window by clicking \textit{OK}.

The new SUGI FRAME class is now ready to use. The following screen prints demonstrate the results.

In the build window, a new frame is requested using the pull down menu File-New

The new entry dialog is shown and test is entered by the user. On clicking \textit{OK} the dialog is closed and the \textit{Select a Frame Type} dialog is displayed:
Conclusions

The frame can be edited so normal widgets can be added, deleted, and the OK button cleared. The frame is then deselected and has

References

available from local SAS institutes, Office of the SAS Institute's Books. By using the frame class, the code

Further Reading

Institution

this paper can be expanded to incorporate more sophisticated frame

Contact details

specifying in SAS software Consultancy for the

Copyright Notice

includes USA registration

trademark of SAS Institute Inc., in the USA and other countries. A trademark is a registered

AXS and observations are registered trademarks of SAS Institute

Building Source Titles 3.0

INTERNATIONAL MARKETING SERVICES CO. LTD

PH 02 77 38 66, FAX +64 9 362 3097

73 Pine Street, Eddi Auckland New Zealand

also Package Region