# Introduction to FRAME Entries in SAS/AF® Software

John C. Boling, SAS Institute Inc., Cary, NC.

## Introduction

SAS/AF software, announced in 1985, is an application facility for building interactive enterprise information systems. As the user interacts with the application, the application can customize and execute SAS source code interactively, noninteractively, or even on a remote machine.

Prior to the introduction of the FRAME entry, entries within a SAS/AF application could not easily intermix text and graphics.

The FRAME entry, introduced in Release 6.08, supports the intermixing of text and graphics and the use of industry accepted graphical user interface widgets, objects, and controls.

## Graphical User Interfaces

Graphical User Interfaces are multi-window, graphic-oriented user applications. Typically, they contain window elements such as bit-mapped and structured graphics, icons, pull-down and pop-up menus, command buttons, scroll bars, and sliders. Most also use a mouse device and pointer control. With GUI-based applications, users typically navigate through the application by pointing and clicking on their selections using a mouse.

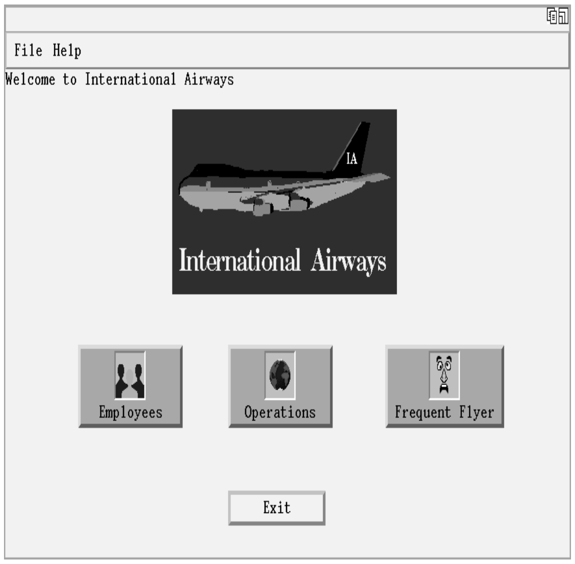

**Figure 1**

GUI applications are often simpler and quicker to use than command based interfaces. For example, GUI menu choices can be icons that users select instead of typing numbers or letters on a command line or pressing function keys. GUI applications are also more intuitive. Rather than requiring users to remember a series of commands and options, the interface prompts for the next action or choice. GUI applications enable users to maneuver quickly and easily through a series of tasks using graphical images.

## Object Oriented Programming Terms

The FRAME entry is predicated on object oriented programming structures. It is helpful to understand these basic terms.

The design of an object oriented system begins not with the task to be performed but rather the aspects of the real world that need to be modeled in order to perform that task.

Objects offer a very natural way of breaking down the problem to be solved. Each object has a range of behavior to be modeled and each has to maintain some information about its status. Why look for some other way to package procedures and data when the problem has already organized them for you.

An object is a software packet containing a collection of related data elements and a set of procedures called methods for operating on those elements. The data within an object can be accessed only by the object's methods. This arrangement, called encapsulation, protects data from corruption by other objects and hides low-level implementation details from other objects and the rest of the system.

Everything an object knows about itself is expressed in its instance variables. Everything it can do is expressed in its methods.

Once defined, new types of objects can be used as building blocks. This ability to create new, high-level structures on demand and use them to build applications is called data abstraction. This is a key theme because it allows us to think in terms of the problem, the task, the physical process, rather than the data types of the language and the procedural flow.

### Classes

Object-oriented programming supports the repeated use of common objects through the use of classes. A class is a general prototype which describes the characteristics of similar objects. The objects belonging to a particular class are said to be instances of that class. Conceptually, it may be helpful to think of a class as a rubber stamp template being used to stamp out instances of itself.

Classes allow objects to be defined in a very efficient manner. The methods and variables for a class are defined only once, in the class definition, without repeating them in every instance. The instances contain only the actual variable values.

Although it is possible to define classes independently of each other, classes are usually defined as special cases, or subclasses of each other. Through a process called inheritance, all of the subclasses for a given class can make use of the methods and variables for that class,

Inheritance increases the efficiency of the class mechanism even further; behavior that's characteristic of larger groups of objects is programmed only once, in the definition of the higher-level class, and the subclass merely adds to or modifies that behavior as required for their special cases.

Subclasses may be nested to any degree, and inheritance will accumulate down through all levels. The resulting treelike structure is known as a class hierarchy.

### Messages

Objects communicate with one another through messages. A message is simply the name of a receiving object together with the name of one of its methods. A message is a request to carry out the indicated method. Any number of objects can include the same method, and each can implement it according to its own unique needs. That allows any given message to be sent to lots of different objects without worrying about how the message will be handled or even knowing what kind of object will receive it. The ability to hide implementation details behind a common message interface is known as polymorphism. Polymorphism makes the object approach very flexible because it allows new kinds of objects to be added to a completed system without writing existing procedures.

## Components of the FRAME

The FRAME entry utilizes a new interactive editing environment and a set of predefined classes.

### Region Manager

When you edit a FRAME, you use the Region Manager (an interactive editing environment), for defining rectangular areas called regions within the window. The regions can be copied, resized, repositioned, emptied, and removed.

Each region is filled with an object, an instance of a widget class. Each object within the FRAME is assigned and referred to by a unique name.

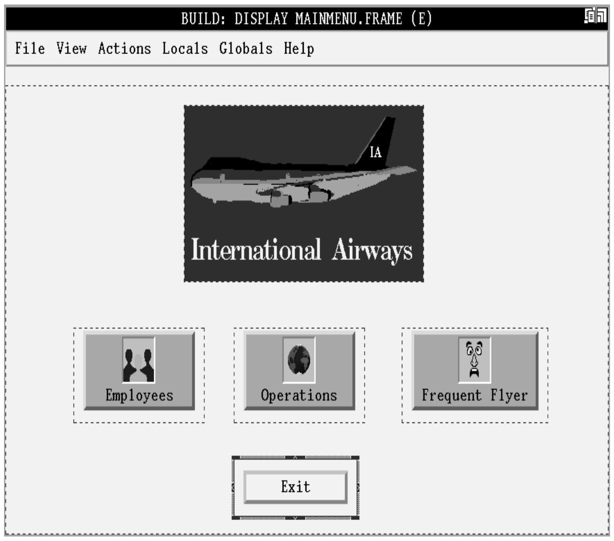
BUILD: DISPLAY MAINMENU.FRAME (E)
File View Actions Locals Globals Help
IA
International Airways
Employees
Operations
Frequent Flyer
Exit

**Figure 2**

## Widget Classes

The FRAME entry provides a group of predefined classes from which you can create objects. The term widget is often used to describe objects that are a component of a graphical user interface that displays information and/or accepts user entries. When you fill a region with a  widget, you are creating an object, an instance of that widget class.

Definitions of the production widget classes in Release 6.12 are:

| Widget | Typical Use |
|---|---|
| Block | provides a rectangular , text based object for menu selections |
| Catalog Entry Viewer | displays the text records of a catalog entry |
| Check Box | provides a marker and associated text that acts like a switch |
| Command Push Button | is a labeled button that presents an action |
| Container Box | provides a way to visually group objects |
| Control | provides a small push button whose label is a picture arrow |
| Critical Success Factor | creates a graphical representation of the position of some value in a range of data |

| Widget | Typical Use |
|---|---|
| Data Table | provides a tabular display of multiple rows of a SAS data set |
| Extended Input Field | extends the functionality of the Input Field including multiple lines, colors, and fonts |
| Extended Table | creates a row containing objects that are logically grouped together and repeated, the row is repeated as a table |
| Extended Text Entry | supports user entry of single or multiple lines of text |
| External File Viewer | displays the contents of an external file |
| Graphic Text | displays text using a variety of fonts, colors, and sizes |
| Graphics | creates and displays a variety of charts and plots |
| Hotspot | defines an area overlapping another object that can perform specific actions when selected |
| Icon | provides a pictorial representation of an object or task |
| Image | displays bitmap images |
| Image Icon | is a graphical equivalent to the Icon class with additional capability |
| Input Field | creates one line text fields that can accept user input and display text |
| Input Field Label | creates a one line label for an Input Field |
| List Box | provides scrollable lists of text entries from which users can make selections |
| Organizational Chart | creates hierarchical charts from data stored in SAS data sets |
| Process Flow Diagram | creates a diagram composed of symbols with connecting arrows and descriptive text |
| Push Button | provides a rectangle with text representing an action |
| Radio Box | provides a group of items from which users can make a single selection |
| SAS/GRAPH Output | displays graphics output from a GRSEG entry |
| Scrollbar | are graphical objects that allow users to move back and forth |
| Slider | allows users to select a numeric value within a range |
| Text Entry | accepts user input and displays text information or program output |
| Text Label | a single line of protected text that is used for labels, titles |
| Toolbar | allows FRAME developers to have tool, action, and style toolbars |
| Video Player | plays a video clip in a rectangular region |
| Work Area | provides an area where you can place functions |

## Screen Control Language

FRAME entries are controlled by Screen Control Language (SCL) programs which are stored separately from the FRAME in a SCL entry. By default, an SCL entry has the same name as the FRAME. For example, if the FRAME is named MAIN.FRAME, the corresponding SCL entry is named MAIN.SCL.

Since SCL entries are stored separately from FRAME entries, you can create more than one FRAME interface that uses the same SCL program without having to duplicate the SCL program for each FRAME entry. The compiled SCL program piggy backs the FRAME it is compiled from.

Some FRAME entries do not even require a SCL program because widget attributes and methods allow FRAME entries to perform many tasks without any additional SCL programming statements.

## SCL Programming Constructs

SCL is a programming language designed to facilitate the development of interactive applications. The language combines the syntax of the DATA step with additional statements and functions that enable developers to control the flow of their applications based on user interaction.

### Structure

SCL entries for FRAME entries contain labeled blocks that execute when users perform an action on an object, like selecting it or changing its value. Each label block begins with a label and ends with a RETURN statement. SCL statements are executed in sequence within the labeled block. The label for each block of code is the same as the object with which it is associated.

In addition a SCL entry can also have an INIT, MAIN, and TERM label.

| Label | Execution |
|---|---|
| INIT | executes before the FRAME entry is displayed to the user |
| MAIN | executes if any object is activated after the corresponding object labeled sections have executed |
| TERM | executes when the FRAME is closed by an END or CANCEL command |

## Methods

As part of its object-oriented nature, each widget class includes a set of methods. These methods define the operations that can be executed by any object you create from that class. Although each widget class includes a set of predefined methods, you can write your own methods.

When you want an object to perform an action, you send it the appropriate message. The message contains the name of the receiving object together with the name of one of its methods, To send messages, use the CALL SEND or CALL NOTIFY routines in the SCL program.

## Illustrations

The remainder of this paper examines two FRAME entries.

## Example 1

In this example MAINMENU.FRAME contains five objects. A SAS/GRAPH output object displays the airplane, three icon objects (named EMP, OPER, and FLYER respectively) display bitmap selections, and a push button object displays an exit.

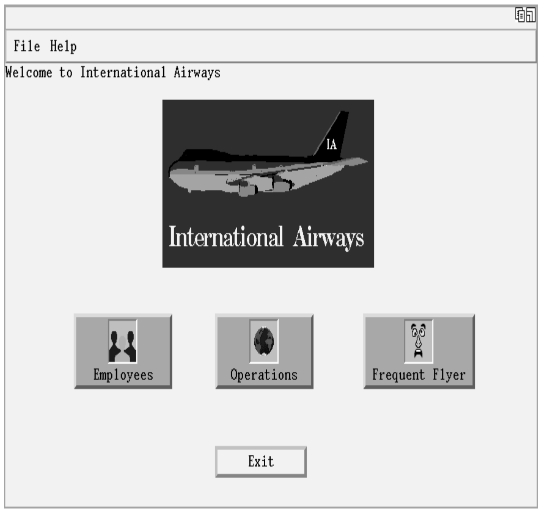
Welcome to International Airways

**Figure 3**

The MAINMENU.SCL program monitors the interaction between the user and MAINMENU.FRAME.

```
INIT:
  _msg='Welcome to International Airways';
return;

EMP:
  call display('empmenu.frame)');
return;

OPER:
  call display('opermenu.frame');
return;

FLYER:
   call display('flymenu.frame');
return;
```

**Figure 4**

The INIT label executes before the FRAME displays and initializes the message string.

The EMP label executes if the user clicks on the EMP icon. The CALL DISPLAY function displays the EMPMENU.FRAME.

The OPER label executes if the user clicks on the OPER icon. The CALL DISPLAY function displays the OPERMENU.FRAME.

The FLYER label executes if the user clicks on the FLYER icon. The CALL DISPLAY function displays the FLYMENU.FRAME.

When the user terminates any of the frames branched to, control returns to MAINMENU.FRAME.

When the user clicks on the EXIT push button, the SAS command ‘BYE’ (programmed as an attribute for the object) executes.

## Example 2

In this last example CHART.FRAME contains three objects. The list box object (named VARIABLE) prompts for a variable name, the radio box (named TYPE) object prompts for a chart type, and a graphics output (named CHART) displays the chart.

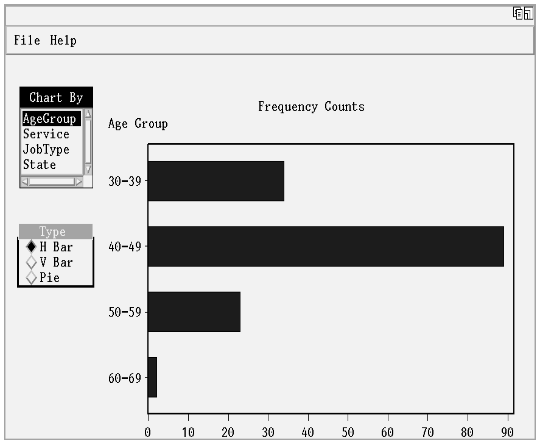

**Figure 5**

The CHART.SCL program monitors the user’s interaction with CHART.FRAME.

```
00001  LENGTH var $8;
00002
00003  INIT:
00004  call notify (‘chart’,’_set_sort_type_’,
00005    ‘ascending’);
00006  return;
00007
00008  VARIABLE:
00009   call notify (variable’,’_get_last_sel_’,row,issel,
00010     var);
00011   call notify ’chart’,’_set_indep_var_‘,var);
00012 return;
00013
00014 TYPE:
00015   call notify(‘type’,’_get_value_’,graftype);
00016   call notify(‘chart’,’_set_type_’,graftype);
00017 return;
```

**Figure 6**

Line 1 declares the variable VAR to be character length eight. Otherwise,  character variables are declared length 200 by default in a SCL program.

The INIT label executes before the FRAME displays.

The CALL NOTIFY function on Line 4-5 sends a message to the CHART object to arrange the bars in ascending order.

The VARIABLE label executes if a selection is made in the listbox object named VARIABLE. The CALL NOTIFY function on Line 9-10 sends a message to the listbox to learn the last value selected. The CALL NOTIFY function on Line 11 sends a message to the CHART object specifying the name of the independent variable.

The TYPE label executes if a selection is made in the radio box object named TYPE. The CALL NOTIFY function on Line 15 sends a message to learn the type of graph chosen. The CALL NOTIFY function on Line 16 notifies the CHART object of the type of chart to display.

## Summary

The FRAME entry in SAS/AF software is predicated on object oriented programming structures. Using a set of predefined widget classes, developers can easily compose graphical user interfaces to the SAS System. A Screen Control Language program controls the user’s interaction with the FRAME.