Getting Started with SAS/AF® Frame Subclasses

Thomas Miron
Miron InfoTec, Inc., Madison, WI
(608) 255-3531

Abstract

One of the most useful features of SAS/AF is the ability to create subclasses of the standard object classes provided by SAS Institute. Subclassing is part of the SAS/AF object-oriented programming (OOP) implementation. This tutorial shows you why and how to create a subclass.

The tutorial begins with a brief overview of some OOP terminology in the context of SAS/AF. Next, a step-by-step subclassing example is presented. The example shows how to create an Extended Input Field subclass that changes its background color when the cursor is in the field.

Overview

The notion of subclassing is an integral part of object-oriented programming (OOP). The following discussion provides an introduction to some OOP terms and concepts (in italics), but keep in mind that you do not need to adhere to object oriented methodology (OOM) in order to take advantage of *subclassing*, including the example in this tutorial.

In the world of OOP, a *class* is typically a software component that performs a fixed set of related functions. Normally, classes do nothing until *instantiated*, i.e., initialized to run in a particular circumstance. An initialized class is an active *object*. In SAS/AF there are graphical (display) objects called *widgets*, and *non-display objects*. An application is usually made up of several objects that work to together to create the user interface (frames and widgets) and provide “under the covers” processing (non-display objects).

When you create a subclass you create a new class (software component) that is based on an existing *parent class*. A set of parent and child classes is often referred to as a *class hierarchy*. Sometimes the parent class provides little or no actual processing but defines only the interface, i.e., the names of data items that its subclasses use and *methods* (operations) that its subclasses can perform. This type of parent class is called an *abstract class*. Sometimes the parent class provides almost all the processing power for its child subclasses. The child classes merely add specialized features or extend functions *inherited* from the parent. In our example we will be working with this second type of parent-child class relationship.

The example tutorial shows how to add a feature to the Extended Input Field widget class as provide by SAS Institute. The new class is a subclass of Extended Input Field that changes background color when the cursor is in the field. This provides a visual cue to the user as to which widget on the frame is currently active. The subclass is responsible only for this feature. All other functions are inherited from the parent class.

Documentation for SAS/AF classes and SCL

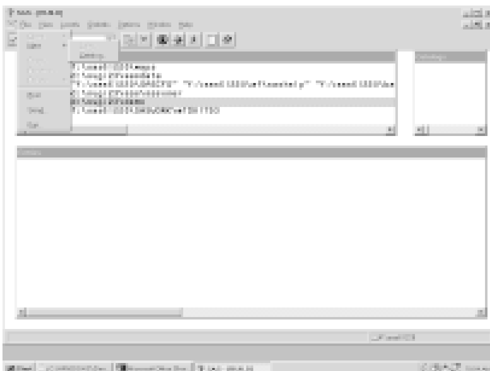
Before starting your own SAS/AF project you should have the following documentation available(SAS Version 6). All these books are available from SAS Institute publications (1-800 767-3228).

- *SAS Screen Control Language, Reference, Version 6, Second Edition*, this is the SCL language reference. Any new methods you write for your subclasses will be written in SCL.
- *SAS/AF Software FRAME Application Development Concepts, Version 6, First Edition*, this book covers the steps to create a subclass and discusses various OOP concepts in the context of SAS/AF and SCL.
- *SAS/AF Software: FRAME Class Dictionary, Version 6, First Edition*, this (large) book is the documentation for the classes provided by SAS Institute.

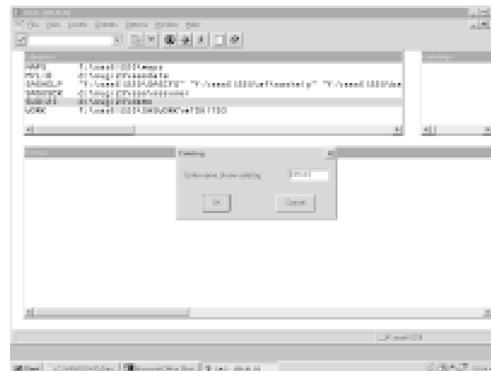
Agenda

This tutorial covers the following tasks:

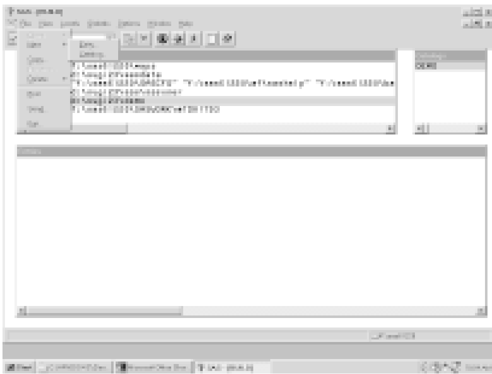
- **Creating a new CLASS entry**
- **Defining method overrides**
- **Adding an instance variable to a class**
- **Coding method overrides for a subclass**
- **Setting up a RESOURCE entry to make a new class available**
- **Using the new class on a frame**
- **Changing behavior without recompiling**
- **Managing your subclasses**
- **Recap and questions**



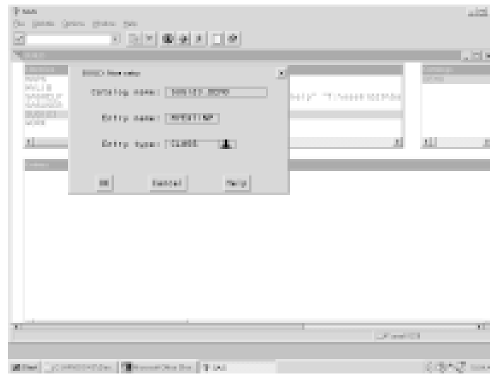
Start BUILD with the "BUILD" command.
Select an existing library, here it's SUGI23.
Select File/New.



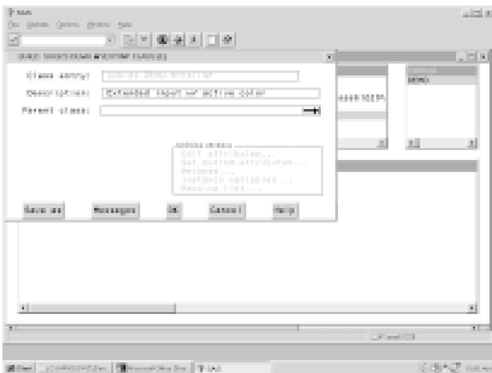
Enter new catalog name: DEMO.
OK.



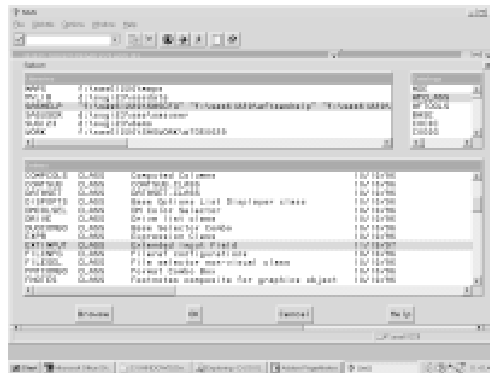
Select your new catalog.
Select File/New/Entry.



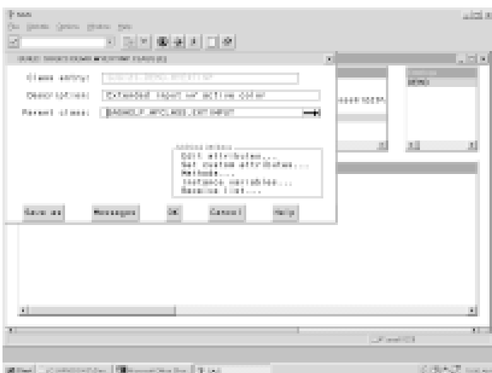
Entry name: MYEXTINP.
Entry type: CLASS
OK.



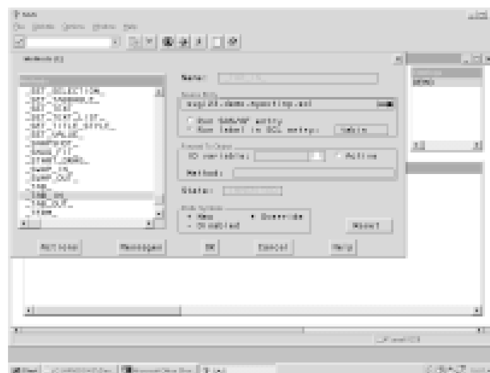
Enter a class description (any text).
Select the Parent Class selector control arrow.



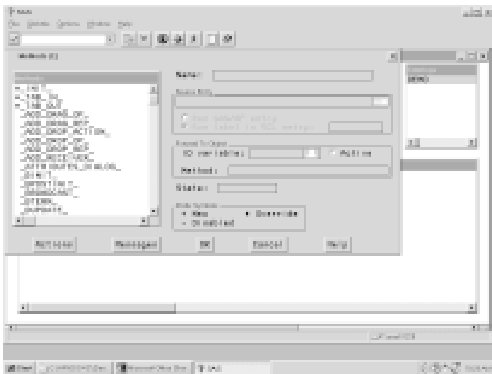
Select SASHELP.AFCLASS.EXTINPUT.CLASS as the parent class.
OK.



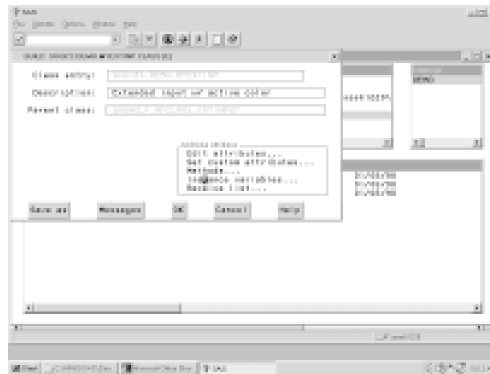
Select Methods... from the Additional Attributes list.



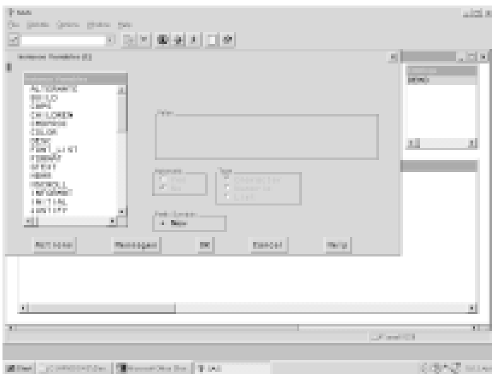
Select the _TAB_IN_ method.
In Source Entry type: SUGI23.DEMO.MYEXTINP.SCL
In run label type: TABIN



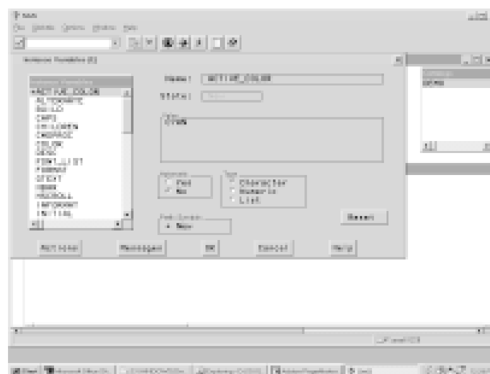
Repeat method overrides for _TAB_OUT_ (label TABOUT) and _INIT_ (label INIT).
OK.



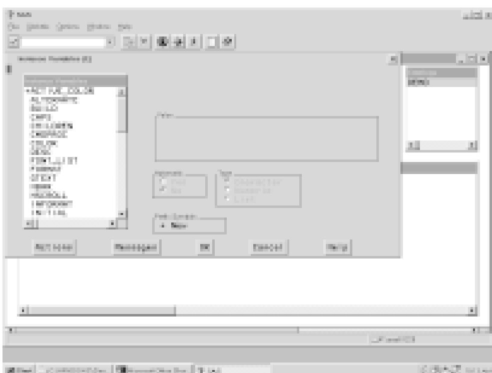
Select Instance Variables... from the Additional Attributes list.



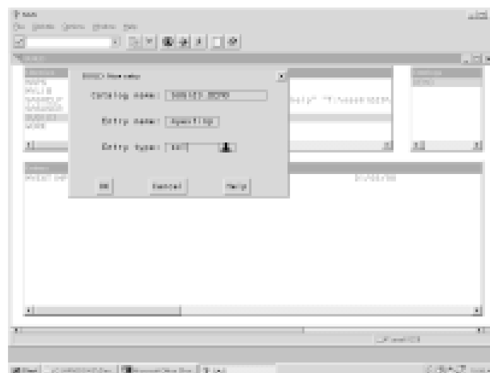
Select Actions button then Add.



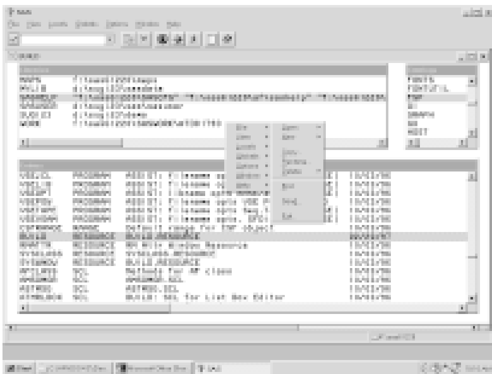
Create a new variable named ACTIVE_COLOR.
Value is CYAN.
Automatic is No (default).
Type is Character (default).
OK.



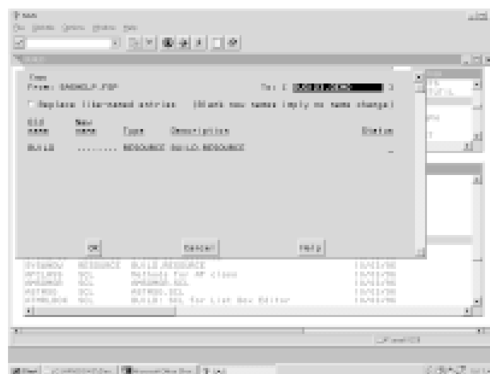
The instance variable list shows your new variable marked with +.



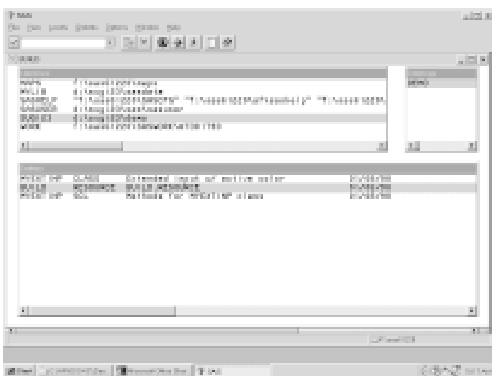
Create a new SCL entry named MYEXTINP in the SUGI23.DEMO catalog.
This entry will hold your method override code. See code listing following these screens.



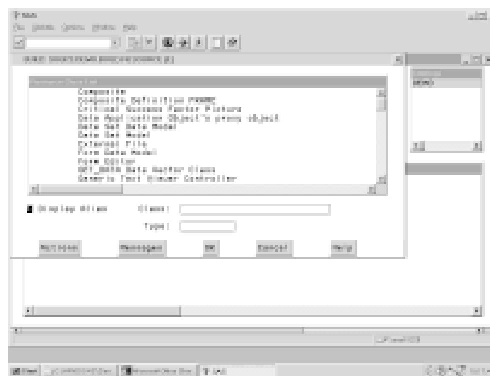
In the BUILD entry selector select entry SASHELP.FSP.BUILD.RESOURCE.
Select File/Copy.



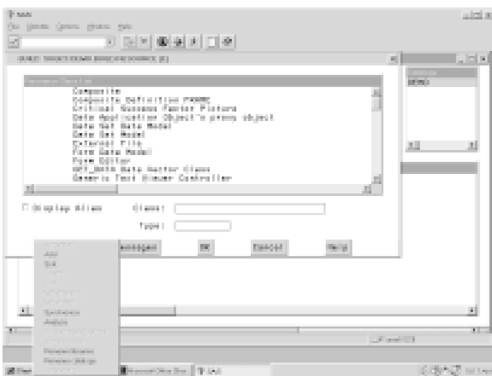
Type SUGI23.DEMO as copy target.
OK.



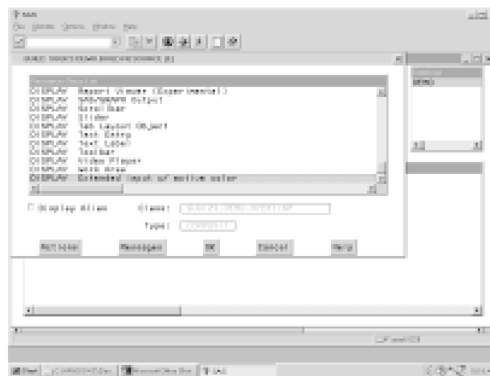
In the BUILD entry selector open (double-click) the SUGI23.DEMO.BUILD.RESOURCE entry.



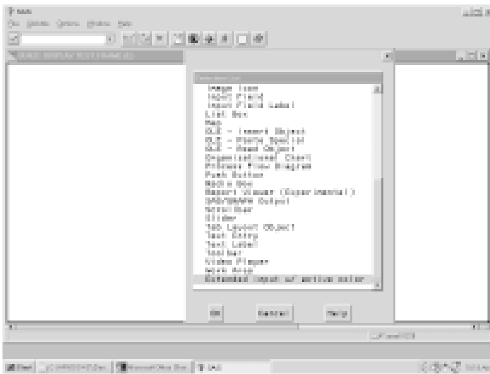
Select Actions button.



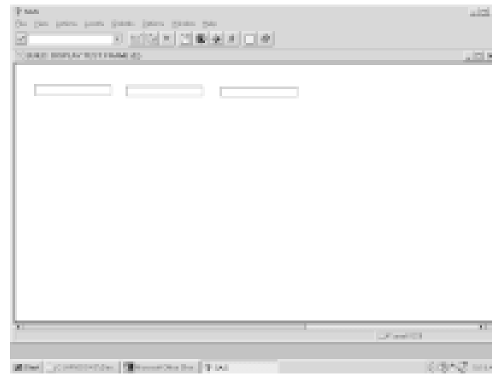
Select Add.
Select SUGI23.DEMO.MYEXTINP.CLASS from the entry selector.
OK.



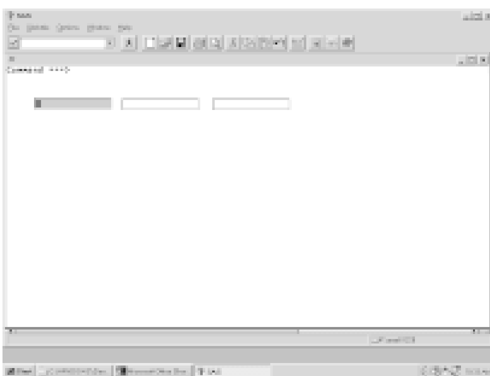
Your resource list now contains the new MYEXTINP class listed by its description text.



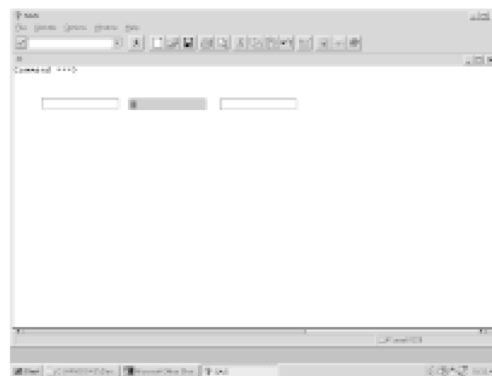
Create a frame entry in the SUGI23.DEMO catalog.
Use the Make command (or menu selection) to display the Make list.
Select the new MYEXTINP class (listed by its description).



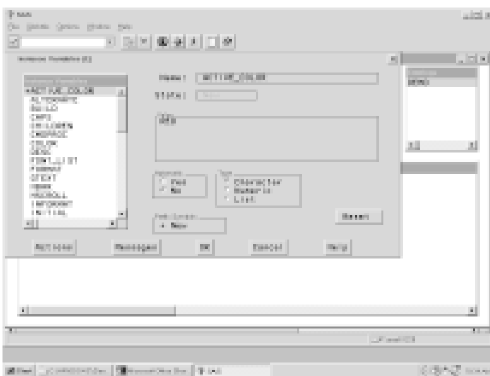
Create a few MYEXTINP widgets on the frame.



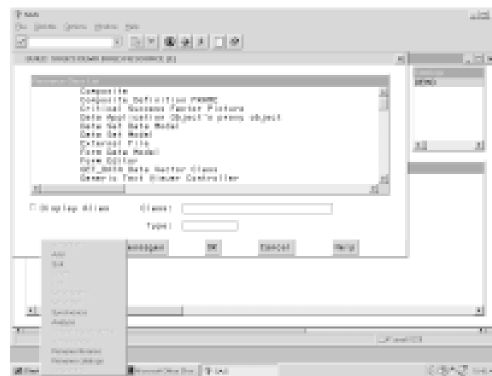
Use the TESTAF command (or menu selection) to test the new frame.



As you select or tab through fields the background color changes to CYAN for the active field.



To change the active color open the MYEXTINP CLASS entry, select the Instance Variables attribute and change ACTIVE_COLOR to RED or another valid SAS color.



When you change the default value of an instance variable for a widget, you must resynchonize the RESOURCE entry for frames using the widget class.
Open the RESOURCE entry, select Actions/ Synchronize.

Class SCL - Entry SUGI23.DEMO.MYEXTINP.SCL

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                          */
/* METHODS MYEXTINP CLASS                                   */
/*                                                          */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

length
   _method_ $40   active_color $24
;

_method_ = _method_;
_self_ = _self_;


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                          */
/* INIT                                                     */
/*                                                          */
/* CALL NAME:  _INIT_                                       */
/*                                                          */
/* STATUS:     OVERRIDE                                     */
/*                                                          */
/* RESET INITIAL BACKGROUND COLOR DISPLAY                   */
/*                                                          */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

INIT: method

;

   /* CALL SUPER */

   call super( _self_, _method_ );


   /* FORCE INITIAL BACKGROUND COLOR TO DEFAULT */

   call send( _self_, '_set_background_color_', 'background' );

 endmethod;
```

If you use the automatic variable _METHOD_, it must be defined as character.

Automatic variables are used in non-executing assignment statements to avoid "not initialized" messages from compiler.

Call default (parent's) _INIT_ method. The automatic variable _METHOD_ is set to the name of the currently executing method, in this case _INIT_.

The _INIT_ method runs before the widget is first displayed. To ensure that the widget starts with the default background color, set background explicitly.

(continued next page)

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                          */
/* TABIN                                                    */
/*                                                          */
/* CALL NAME:  _TAB_IN_                                     */
/*                                                          */
/* STATUS:     OVERRIDE                                     */
/*                                                          */
/* HANDLE TAB EVENT                                         */
/*                                                          */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

TABIN: method

;

   active_color = getnitemc( _self_, 'active_color' );

   call send( _self_, '_set_background_color_', active_color );

 endmethod;
```

_TAB_IN_ runs when the widget becomes the active widget.

In SAS version 6, instance variables are stored as items in the list defining the active object. When an object's method is running this list id is stored in the automatic variable _SELF_.

Retrieve the active color and apply as the widget's (_SELF_) background color.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                          */
/* TABOUT                                                   */
/*                                                          */
/* CALL NAME:  _TAB_OUT_                                    */
/*                                                          */
/* STATUS:     OVERRIDE                                     */
/*                                                          */
/* HANDLE TAB EVENT                                         */
/*                                                          */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

TABOUT: method

;

   call send( _self_, '_set_background_color_', 'background' );

 endmethod;
```

_TAB_OUT_ runs when the widget looses the active focus.

Reset the background color to the default background.

Managing your subclasses

As soon as you create more than one customized subclass, class management becomes an issue. The first question is "Where do I store CLASS and associated entries?" The two most common strategies are to store each group of entries (CLASS, SCL, SOURCE, RESOURCE) in a separate catalog or to store may classes in a single catalog (as in the SASHELP.FSP catalog).

If you store each group of entries in a separate catalog it is easier to manage individual customized classes and you will have fewer entry name conflicts. If you store many, possibly related, classes in a single catalog it is easier to keep related classes together if you routinely move catalogs among machines, as on a PC network. Storing related classes together also emphasizes their relationship when it comes time to make changes.

There is no single answer to the storage arrangement question. In actual use you may find it advantageous to use both storage strategies. Whatever you do, think about it first! Changing the LIBREF's and catalog names associated with class components can be a tricky process.

Recap

Subclassing is a feature of development systems that support object oriented programming. With the SAS/AF development system you do not need to adhere to an object oriented methodology to take advantage of the power of subclassing. Subclassing allows you to extend or alter the behavior of classes (widgets) supplied by SAS Institute and is useful for informal applications that do not use a rigorous object oriented design.