Getting Started with SAS/AF® 9.4 and Frames
# Contents

## PART 1  The SAS/AF Development Environment  1

**Chapter 1 • Introduction to SAS/AF Software**  3
- Overview  3
- Purpose of This Document  3
- Getting More Information  4
- Software Requirements  4

**Chapter 2 • The Building Blocks of Frame Applications**  5
- Components  5
- The SAS/AF Development Environment  6
- A Simple Methodology for Frame Development  9
- Using Models  10

**Chapter 3 • Adding SCL Programs to Frames**  13
- SAS Component Language (SCL)  13
- The Fundamentals of Frame SCL  15
- Dot Notation and SCL  17
- Controlling the Execution of SCL Programs  18
- Calling Other Frames  19
- Saving Frame SCL Programs  19
- Compiling Applications  19
- Testing Applications  20

## PART 2  Creating an Application  21

**Chapter 4 • Build a Frame Application**  23
- Overview of the Frame Application  23
- Build the Display_data Frame  24
- Build the Create_report Frame  34
- Build the Start_menu Frame  41

## PART 3  Appendixes  45

**Appendix 1 • Defining Attachments**  47
- Understanding Attachments  47
- Define Attachments That Resize the Table Viewer  52
- Test the Table Viewer Attachments  54
- Define Attachments That Move the Close Window Button  55

**Appendix 2 • Deploying Applications**  57

**Appendix 3 • Defining a Subclass**  61
- Subclassing  61
Create a Close Window Button Subclass ........................................... 62
Overriding Attributes ..................................................................... 62
Add the Close Window Button Class to the Components Window ....... 64
Test the New Close Window Button ............................................... 64

Glossary ............................................................................................. 65
Part 1

The SAS/AF Development Environment

Chapter 1

Introduction to SAS/AF Software ........................................... 3

Chapter 2

The Building Blocks of Frame Applications .......................... 5

Chapter 3

Adding SCL Programs to Frames ........................................... 13
Chapter 1

Introduction to SAS/AF Software

Overview

SAS/AF software is a set of tools for developing applications. Central to the SAS/AF development environment is the frame. You can think of a frame as an application window that contains the interface (the fields and buttons) of your application. With SAS/AF frame application development, you can build much of your application visually, using drag-and-drop components. And because SAS/AF applications are stored in SAS catalogs, they are portable to all SAS software platforms.

Purpose of This Document

This document is an introduction to the SAS/AF development environment. It guides you through the basic skills that you need to build a simple frame application. It also gives you a foundation with which you can transition to the larger reference manuals that fully cover SAS/AF software.

Although this document is intended for new users of SAS/AF, you should be familiar with basic SAS concepts such as libraries, catalogs, and catalog entries. You do not need object-oriented programming experience to benefit from this document, but familiarity with object-oriented concepts will certainly help.

Although specific to SAS/AF in SAS®9, the overall processes that are presented in this document also apply to versions of SAS/AF software starting with SAS 8.1.
Getting More Information

Help

Help is always available when you are using the SAS/AF development environment. To access help, select Help ⇒ SAS Help and Documentation, navigate to SAS Products and then navigate to SAS/AF.

You can also get help on most windows and dialog boxes inside SAS/AF by pressing the F1 key when the window or dialog box is the active window, or by selecting Help ⇒ Using This Window.

To access Help on a component in the Components window, right-click on the component, and then select Help on Class.

Documentation Available on the Web

SAS documentation, available in HTML or PDF, is available on the Web at support.sas.com/documentation/onlinedoc/af/.

The following books offer more information about developing applications using SAS/AF software:

- Guide to SAS/AF Applications Development
- SAS Component Language 9.4: Reference
- SAS/AF 9.4 Procedure Guide

Software Requirements

To build the frame applications in this document, you must have SAS/AF software installed, and you must have a monitor that is capable of displaying graphics. To run the frame applications in this document, you must have Base SAS software.

Mainframe Support

SAS/AF does not support frame application development on a mainframe. However, you can build a frame application on another platform and then port that application to a mainframe platform (see “Native Controls” on page 6).
Chapter 2
The Building Blocks of Frame Applications

Components

Components are pieces of software that you can use to build applications. SAS/AF provides several components that enable you to build graphical user interfaces and then link those interfaces to data. There are two basic types of components: controls and models.

Controls and Models

Controls constitute the graphical user interface, and include interface elements that you have seen in Web forms like Check Boxes, List Boxes, and Entry Fields. There are also controls that are specific to SAS/AF such as the Table Viewer (which displays SAS table data).

Models are another type of component. In contrast to the controls that are displayed to the user in the interface, models work behind the scenes to distribute data to controls. For example, to get a List Box to display a list of SAS libraries, you would attach a Library List model to the List Box.

Controls are sometimes called visual components, and models are sometimes called non-visual components. Controls and models are also generically called objects, especially in the context of object-oriented programming.
Native Controls

The controls that are supplied by SAS always appear as *native controls* on a platform, even if you ported your application to that platform. This means that if you wrote an application on Solaris, and then ported it to Windows 7, the application would look exactly like other Windows 7 applications.

However, if you run a graphical user interface application on a character-based display (usually on mainframes), the controls (for example, the entry fields and list boxes) are represented as characters, which means the controls will look different from the examples in this document.

The SAS/AF Development Environment

Development Environment Windows

The SAS/AF development environment (also called the build environment) has four main windows:

- the frame (contained in a Build window)
- the Components window
- the programming source code (in a Source window)
- the Properties window
The Frame

A frame is where you build the graphical user interface to your application. Frames are displayed in Build windows. When you save a frame, it is stored as a Frame entry in a SAS catalog. One application can use several frames, and you can have several frames open at the same time.

Figure 2.2  A Simple Frame at Build Time
The Components Window

The Components window lists commonly used components (controls and models) that you can drag onto a frame. By default the Components window appears when you open or create a frame. Alternatively, you can select a type of control and then double-click on the frame where you want to place it.

Figure 2.3 The Components Window (Abbreviated)

To access Help on a component in the Components window, right-click on the component, and then select Help on Class.

The Properties Window

The Properties window displays all of the properties of all the components on a frame (including the frame itself). From the Properties window you can view and edit properties.

Properties are the defining characteristics of a component. Properties are the attributes, methods, events, and event handlers that are defined on a component. Although events and event handlers are important for more complex applications, the example later in this document focuses mainly on attributes and uses only one method.

Attributes define information about a component, such as its name, height, and width. Methods define what a component can do, such as selecting or deselecting all the items in a List Box.

You use the Properties window to manipulate properties at build time, and you use programming code to manipulate properties at run time.

With a frame open, you can open the Properties window by selecting View ➔ Properties Window. You can also open the Properties window by right-clicking a component and selecting Properties.

You can open only one Properties window, and that window is shared between all open frames. The Properties window can only be opened when a frame is open.
The Source Window

Source windows contain a text editor for creating and editing programming code. You can have several Source windows open at the same time. The programming language that is used in SAS/AF is examined in Chapter 3, “Adding SCL Programs to Frames,” on page 13.

A Simple Methodology for Frame Development

The four main SAS/AF windows structure your work flow. Building a frame application typically consists of the following steps:

1. Create a frame and add components to it.
2. Modify the properties of the components, if necessary.
3. Add programming code, if necessary.
4. Save, compile if necessary, and test the frame.
5. Repeat steps 2 through 4 until your application works as desired.

---

**Using Models**

As previously explained, models access and distribute data to controls which then display that data. A simple example of how models and controls work together is populating a List Box with color choices. This is something you might do to let a user select the colors for a pie chart. You could add by hand to the List Box each of the standard SAS colors: Black, Blue, Brown, Cyan, Gray, Green, Magenta, Orange, Pink, Red, White, Yellow. Or you could associate a Color List model with the List Box, and have the List Box populated for you.

*Figure 2.6  A Color List Model Supplying a List Box with Available Colors*

Just as people can only communicate through a common language, certain models only work with certain controls. The following table lists the controls and models that SAS/AF software provides, and which controls and models can be used together. Within each row of the table, any control on the left can be used with any model on the right.
Table 2.1  Control and Model Compatibility

<table>
<thead>
<tr>
<th>Controls</th>
<th>Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combo Box</td>
<td>Catalog Entry List</td>
</tr>
<tr>
<td>Dual Selector</td>
<td>Catalog List</td>
</tr>
<tr>
<td>List Box</td>
<td>Color List</td>
</tr>
<tr>
<td>Radio Box</td>
<td>Data Set List</td>
</tr>
<tr>
<td>Spin Box</td>
<td>External File List</td>
</tr>
<tr>
<td></td>
<td>Library List</td>
</tr>
<tr>
<td></td>
<td>LIST Entry List</td>
</tr>
<tr>
<td></td>
<td>SAS File List</td>
</tr>
<tr>
<td></td>
<td>SLIST Entry List</td>
</tr>
<tr>
<td></td>
<td>Variable List</td>
</tr>
<tr>
<td></td>
<td>Variable Values List</td>
</tr>
<tr>
<td>Form Viewer</td>
<td>SAS Data Set</td>
</tr>
<tr>
<td>Table Viewer</td>
<td>SCL List</td>
</tr>
</tbody>
</table>

For models and controls that are designed to work with each other, all you need to do is connect them and they automatically communicate with each other. To connect models and controls at build time, simply drag a model onto a control when you are designing the frame, and SAS/AF sets the connection for you. You can also connect a model and a control manually, by setting the control's model attribute in the Properties window.

The example later in this document uses the following models: Catalog Entry List, Library List, and the Variable Values List.
SAS Component Language (SCL) is the programming language that controls SAS/AF applications (including frames and the controls on frames).

SCL programs are stored in SCL entries, separately from frames. Because of this separation, SCL entries can be accessed by more than one frame, which means that an SCL program or function can be written once and used many times.

SCL is an object-oriented programming language that was designed to facilitate the development of interactive SAS applications. SCL enables you to perform the following tasks:

- calculate and validate field values that are based on user input
- change the attributes of components on a frame at run time
execute methods of components on a frame
+ link to other SAS catalog entries, including other SCL entries and frames
+ submit SAS programs
+ read from and write to SAS tables, SAS catalog entries, and external files

This document does not cover all of these topics. For complete reference information about SCL, refer to the SAS Component Language: Reference (the complete text of which is in the SAS/AF help).

**Frame SCL**

Frame SCL is an SCL entry that is associated with a particular frame (and is the only type of SCL that is used in the example later in this document). Frame SCL is typically used to control a frame and the components on that frame.

For example, in the following diagram, the SCL code that is marked 'A' is the initialization code for the frame, and executes before the frame is displayed. This code specifies that the model that is associated with the List Box look in the SASHELP library for tables. The code that is marked 'B' is executed when a selection is made from the List Box. When a table is selected, the table is displayed in the Table Viewer.

![Figure 3.1 Frame SCL Controls a Frame](image)

You can view and edit the frame SCL for any frame that you have open and active by selecting View ➔ Frame SCL, or by selecting Frame SCL from the frame's pop-up menu.

Although you can open and edit frame SCL just as you can an SCL program that is not associated with a frame, you should only compile frame SCL from its associated frame (see “Compiling Applications” on page 19 for more information).
**When Frame SCL Is Not Required**

A frame does not require an SCL program. Some components that you can add to a frame are designed to perform tasks without additional SCL code. For example, you can add a Push Button control to a frame and set its commandOnClick attribute to `end;` (with the semicolon). The result is that when a user clicks the button, the END command executes, closing the frame that the button is on. Instead of compiling a frame that has no SCL code (which will produce an error), just save it.

**When Frame SCL Is Required**

Frame SCL is required if you need to do any of the following:
- link to other SCL entries or frames.
- submit SAS programs.
- modify a component's properties at run time. For example, changing the appearance of a control after a user enters input.
- execute a method on a component.
- validate the selections that a user makes.

**The Fundamentals of Frame SCL**

**The Parts of Frame SCL**

Typical frame SCL code consists of the following:
- labeled sections
- SCL variable declarations
- routines and functions

**SCL Labeled Sections**

An *SCL labeled section* is a set of programming statements that execute as a unit. A section in SCL begins with a label and ends with a RETURN statement.

Sections that are labeled with the name of a control on the associated frame execute when the user interacts with that control. For example, if you have a Push Button named `exitButton` on your frame, and you want to use it to confirm that the user wants to exit the application, you might have a labeled section similar to the following in your frame SCL:

```scl
exitButton:
  dcl list message={'Are you sure you want to exit?'};
  response=messagebox(message, '!', 'YN', 'Confirm Exit', 'N', '');
  if response='YES' then call exec(cmd('end;'));
  message=dellist(message);
return;
```
The code in the exitButton section executes when that exitButton is clicked, resulting in a dialog box to confirm the exit.

**Figure 3.2** The exitButton Confirmation Dialog Box

Section labels do not have to match the casing of the name of the control to which they are associated.

Frame SCL uses reserved sections for program initialization and termination (there is also a main processing section, but that is not covered in this document). The INIT section executes once before the frame is displayed to the user, and is typically used to initialize variables and open SAS tables. The TERM section executes once before the frame is closed, and is typically used to close tables and delete variables that are no longer needed. You should always delete lists when they are no longer needed.

Here are example INIT and TERM sections:

**INIT:**
```
dcl num variable1 rc; /* Declares two numeric variables. */
dcl list myList={};  /* Declares an empty list. */
return;
```

**TERM:**
```
rc=dellist(myList);  /* Deletes the list myList. */
return;
```

### SCL Variables

Each variable that is used in SCL is of a specific data type. The following SCL data types are used in the example application later in this document:

- **Character** declared with the keyword CHAR
- **Numeric** declared with the keyword NUM
- **List** declared with the keyword LIST

All variables should be declared using a DECLARE statement. DECLARE statements do not have to exist in labeled sections. You can declare several variables with one DECLARE statement. You can also use the abbreviation DCL.

CHAR, NUM, and LIST are reserved keywords that indicate the data type of the variables.

CHAR\((n)\) is a notation that enables you to define the length of a character variable, where \(n\) is the length in characters, a value up to 32767. By default, character variables are 200 characters long. Consider the following code:
DECLARE NUM n1 n2, /* Two numeric variables. */
CHAR c1, /* A character variable with length 200. */
CHAR(10) c2, /* A character variable with length 10. */
LIST myList={}; /* An empty SCL list. */

SCL Routines and Functions

SCL provides a rich set of routines and functions that, for example, enable you to inspect and manipulate SAS catalogs, SAS tables, and the controls on a frame. For detailed information about these functions, refer to SAS Component Language 9.4: Reference.

SCL also supports nearly all of the functions of the Base SAS language. For details about the Base SAS functions, see the SAS 9.4 Functions and CALL Routines: Reference.

Dot Notation and SCL

To improve code readability and to reduce the amount of coding that is necessary, SCL supports dot notation, a syntax for accessing component properties (attributes and methods). Using dot notation also enables the compiler to check your syntax at compile time.

In dot notation, the object (the List Box or Table Viewer) is separated from the property (the attribute or method) by a period, which is called a dot. The syntax follows this format:

object.property;

Dot notation is used to set or query attributes, for example:

/* Setting the text color. */
   textEntry1.textColor='green';

/* Querying the text color. */
/* The textEntry1.textColor attribute is */
/* returned to the variable 'color'. */
   dcl char(10) color;
   color = textEntry1.textColor;

Dot notation is also used to call methods. For example, the following code deselects all the items in listbox1:

listbox1._deselectAll();

Sometimes you must provide a method with values. These values are called arguments. For example, the following code selects a row of a Table Viewer using the _selectRow method when a Push Button is clicked. The method requires an argument that specifies the row to select, in this case row 5.

init:
   /* Set the table to view. */
   sasdataset1.table='sashelp.class';
   return;
pushbutton1:
   /* Select row 5. */
   dcl list row={5};
   tableviewer1._selectRow(row);
if row then row=dellist(row);
return;

The frame with the selected row:

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alim</td>
<td>M</td>
<td>14</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>Alice</td>
<td>F</td>
<td>13</td>
<td>56.5</td>
<td>86</td>
</tr>
<tr>
<td>Babu</td>
<td>F</td>
<td>13</td>
<td>55.3</td>
<td>90</td>
</tr>
<tr>
<td>Carol</td>
<td>F</td>
<td>16</td>
<td>52.8</td>
<td></td>
</tr>
<tr>
<td>Harry</td>
<td>M</td>
<td>14</td>
<td>53.5</td>
<td></td>
</tr>
<tr>
<td>James</td>
<td>M</td>
<td>12</td>
<td>57.3</td>
<td>85</td>
</tr>
<tr>
<td>Jane</td>
<td>F</td>
<td>12</td>
<td>59.8</td>
<td>84</td>
</tr>
<tr>
<td>John</td>
<td>M</td>
<td>18</td>
<td>50.3</td>
<td>84</td>
</tr>
</tbody>
</table>
```

Sometimes the arguments that you supply to methods are changed in the process of running the method. The following code determines whether item #3 in listbox1 is selected. If the specified row was selected, the variable `selected` is set to 1. If the specified row was not selected, the variable `selected` is set to 0.

dcl num selected,
   num row = 3;
listbox1._isSelected(row, selected);

---

**Controlling the Execution of SCL Programs**

A great deal of control is possible using simple conditional statements. To control application execution in an SCL program, use an **IF/THEN** conditional statement and a **DO** group. For example, the following code uses an **IF/THEN** conditional statement with a **DO** group to clear values that are entered on a frame when the `clearValuesButton` is clicked (but only if the `frameProtected` variable is set to No):

```
clearValuesButton:
   if frameProtected = 'No'
      then
         do;
            textEntryName.text = ".
            textEntrySalary.text = .
            textEntry.text = ".
      end;
   return;
   /* ...SCL statements... */
return;
```

For additional information about controlling application flow (the **DO WHILE**, **DO UNTIL**, **GO TO**, and the **SELECT** and **WHEN** statements), see the *SAS Component Language: Reference*. 
Calling Other Frames

You can use SCL to access one frame from another frame. In fact, your applications can consist of as many frames as you like. For example, the following SCL statement in the frame SCL for Frame1, runs the frame named Frame2 from the current catalog:

```sas
call display('Frame2.frame');
```

The SCL code for Frame1 transfers control to Frame2 and then waits for Frame2 to close. While Frame2 is open, the controls on Frame1 are not accessible. When Frame2 is closed, control returns to the Frame1 SCL, and continues execution, starting with the first statement following the CALL DISPLAY.

For example, assume that a frame contains a Push Button control named Rates. When a user clicks the Rates button, a frame named loanRates is opened that displays rate tables. The frame SCL for the Rates button would look something like this:

```sas
RATES:
    call display('loanRates.frame');
    return;
```

Saving Frame SCL Programs

Saving frames and frame SCL is normally a straightforward process: you simply select File ⇒ Save. However, because of the dependencies between a frame and its frame SCL, you must ensure that the frame properly references its SCL entry.

Normally, except for the SCL entry type extension “.scl”, frame SCL has the same name as the frame with which it is associated. For example, the frame myFrame.frame would by default have frame SCL named myFrame.scl.

If you change the name of a frame that already has associated SCL, you must also remember to change the name of the SCL entry to match the name of the FRAME entry. You must then recompile the frame.

Compiling Applications

If you add SCL code to a frame, you must compile the frame before you can run it. Compiling is the process of translating your SCL code into a language that can be executed.

To compile a frame in the build environment, make the frame active, and then select Build ⇒ Compile.

If the frame and SCL code compile successfully, you should see a “Code generated” message in the Log window (and no warnings or errors). For example:

```text
NOTE: Compiling MYFRAME.FRAME (SASUSER.EXAMPLE.MYFRAME.SCL).
NOTE: Code generated for MYFRAME.FRAME. Code size=4095.
```

To view the Log window, select View ⇒ Log.
But as you know, typing code can occasionally lead to a mistake. For example, assume that while typing in some code, you add an extra 'x' to the SCL section name columnsListbox. When you attempt to compile the frame, the Log window shows that the compiler has issued a warning about a potential error:

```
NOTE: Compiling MYFRAME.FRAME (SASUSER.EXAMPLE.MYFRAME.SCL).
WARNING: [Line 43] Label columnsListboxx is Defined but not referenced
NOTE: Code generated for MYFRAME.FRAME. Code size=4095.
```

Although the code compiles with only a warning (notice the words “Code generated” in the last line of the message), the frame will not function properly. Because of the mistake, the code for the columnsListbox will never be executed because the SCL label (with two x's) does not match the control name (with one x).

In general, if a frame compiles with a warning, you might be able to run it, but it could have run-time errors that make it unusable. You should review and understand the cause of all warnings. When appropriate, the cause of warnings should be fixed.

As discussed above, compiling a frame compiles the frame SCL for that frame. You can also compile frame SCL that was opened from its frame and maintain the frame/SCL association.

If you compile frame SCL independently of its frame, the compiled code is not associated with the frame, and the frame will not call the proper code when a user interacts with a control. Only frames with frame SCL code must be compiled.

---

**Testing Applications**

SAS/AF software provides a testing mode that is available from within the build environment. To test your frame, make it the active window, and then select **Build ⇦ Test**.

To test your application outside the build environment (that is, without having the frame open), open a SAS Explorer window, right-click on the frame, and select **Run**. If the frame has not been previously compiled, you will receive an error. You can compile the frame from the SAS Explorer window by right-clicking on the frame, and then selecting **Compile**.

The only limitation of the **Build ⇦ Test** menu command is that it does not process SUBMIT statements in SCL code. If you attempt to test a frame that has a SUBMIT block using **Build ⇦ Test**, the SUBMIT block will fail at run time. Frame applications that are tested outside the build environment (from the SAS Explorer window) perform with complete functionality.
Part 2

Creating an Application

Chapter 4

Build a Frame Application
Overview of the Frame Application

In this example you build a data viewing application that consists of three frames, each of which has frame SCL. Two of the frames display data, while the third is the navigation system that ties the frames together.
You build the first two frames individually, and then you compile and test them. When you complete the third and final frame you will compile it, and then test the entire application.

---

**Build the Display_data Frame**

**Purpose of the Display_data Frame**

The Display_data frame enables users to select and display a SAS table as columnar data. The frame also enables users to subset the displayed data by column, and by using a WHERE expression.

*Figure 4.1 Finished Display_data Frame*

![Sales Data](image)

Create the Display_data frame by entering the following command at the SAS command line:

```sas
build sasuser.example.Display_data.frame
```

The SAS command line is usually in the upper-left corner of the main SAS window.
The empty frame appears and the Components window is displayed.

**Build the User Interface for the Display_data Frame**

To create the graphical user interface for the frame, drag the following controls from the Components window onto the frame and position them as you see in Figure 4.3 on page 26:

- one Text Label Control (this is for the title at the top of the frame)
- three List Box Controls
- one Table Viewer Control
- three Push Button Controls

Alternatively, you can select each type of control and then double-click on the frame where you want to place it.

On some UNIX platforms, you might need to press two buttons on your mouse to activate a drag action (consult your host documentation).

If you run out of room while dropping controls on the frame, make the frame bigger by resizing it the way you would resize any other window. You can make it smaller later, after positioning the controls.

After dragging all the controls to the frame, you should now have a frame that resembles the following:
If you accidentally drop one control inside another so that the larger control completely surrounds the smaller, you might not be able to select the smaller control. To access the smaller control, select the larger control and move it out of the way so that you can select and move the smaller control to its proper position.

**Move and Resize the Controls**

After you place the controls on the frame, position them like the controls in Figure 4.1 on page 24 by clicking on a control and then placing the mouse pointer on a portion of the light gray border around the control. When the pointer changes into a hand, you can drag the control to a new position.

At this point, the only control that you need to resize is the Table Viewer. To resize a control, select the control and then place the mouse pointer on the dark handles around the control. When the pointer changes into an arrow, you can resize the control.

**Figure 4.4 Mouse Pointers for Moving and Resizing Controls**

You can also resize controls with pixel-level accuracy using attributes. Resizing controls using attributes is as simple as entering height and width values. Using attributes to size
controls is examined later in this document (see “Set the Attributes That Control the Interface” on page 28).

The remaining controls will be resized using attributes later in this example.

**Align the Controls**

Although you might have aligned the controls by hand already, there are layout tools available that can help you do the job precisely.

To align the left edges of the three List Box controls, select all three controls by either holding the SHIFT key and clicking each List Box, or by using the mouse to draw a box around all three (controls are selected when their borders turn thick and gray). Now select **Layout ⇄ Align ⇄ Lefts**.

You can drag controls that are selected as a group.

Align the remainder of the components so that your frame resembles Figure 4.1 on page 24.

**Set the Attribute Values for the Display_data Controls**

**How to Set Attributes**

To change the appearance and the behavior of the controls at build time, you must set their attributes by using the Properties window. The procedure for setting attributes at build time is the same for all attributes.

**Set the First Attribute**

The first attribute to set is the text of the banner at the top of the frame (which currently says **Label**). To set the Textlabel1 **label** attribute, follow these steps:

1. Open the Properties window by right-clicking in the frame and selecting **Properties**. Notice in the Properties window that all the components are listed on the left.
2. Select **Textlabel1** on the left side of the Properties window. The attributes for the Textlabel1 control are listed on the right.
3. Scroll up the list of attributes until you see the **label** attribute.
4. Click inside the Value column on the label row.
5. Type **Sales Data** and press ENTER.

Notice in the frame that the Textlabel1 control now displays **Sales Data**.
To set the font of the label, follow these steps:

1. Scroll to the font attribute.
2. Click inside the Value column where it says (list).
3. Click on the ellipsis button.
   The Font dialog box appears.
4. Change the font to Arial, Regular, 16.
5. Click OK.

In the frame, resize Textlabel1 to make it display the text correctly.

**Set the Control Names**

Because remembering the generic name for each control might be difficult (Listbox2 does not have much meaning), you should rename the controls. Recall that the names of controls are used as section labels in frame SCL code (which will be added later).

In the Properties window, set the name attribute of each control as follows:

- Listbox1 to LibrariesListbox
- Listbox2 to TablesListbox
- Listbox3 to ColumnsListbox
- Pushbutton1 to SubsetButton
- Pushbutton2 to ClearButton
- Pushbutton3 to CloseButton

**Set the Attributes That Control the Interface**

To set the text that a user sees in the interface, set the following:

- LibrariesListbox title attribute to Select library:
- TablesListbox title attribute to Select table:
- ColumnsListbox title attribute to Select columns:
• ClearButton label attribute to **Clear Subset**

On the SubsetButton, set the following to add text and an icon:

• buttonStyle attribute to **Icon with Text Under**

• icon attribute to 715

• height attribute to 40

• label attribute to **Subset with WHERE**

Although you might want to set the width of the SubsetButton at this point, wait until the next section.

To customize the CloseButton, set the following:

• commandOnClick attribute to `end;`
  
  Note the semicolon at the end of the command.

• height attribute to 30

• label attribute to **Close Window**

• width attribute to 80

So that users can select more than one item from the ColumnsListbox, set its selectionMode attribute to **Multiple Selections**.

---

**Set Attribute Values for Multiple Controls**

Attributes that are common between two or more controls can be set simultaneously, which can save time. To set the width of the three List Boxes and the SubsetButton, select all three list boxes and the SubsetButton. To select multiple controls on a frame, either hold down the SHIFT key while you click each control, or drag the mouse pointer across each control.

When all four of the components are selected, the Properties window displays only the attributes that the components have in common. Set the width of all four components to **107**.

---

**Figure 4.6**  *Displaying Shared Attributes*

Notice on the frame that all four components have changed widths.

You might need to move or align the controls so that they do not overlap.
**Attach Models to the Display_data Frame Controls**

Notice that the List Boxes all display a generic list of four items. This is because they, and the Table Viewer on the right side of the frame, have no access to data. To provide all the controls with data, you need to associate them with models. Dropping a model onto a control sets the *model* attribute on the control, which is all you need to do to associate a control and a model.

To associate the correct model with the proper control, drag the indicated model onto the control on the frame:

- a Library List Model onto the List Box labeled Select Library (LibrariesListbox)
- a Data Set List Model onto the List Box labeled Select Table (TablesListbox)
- a Variable List Model onto the List Box labeled Select Columns (ColumnsListbox)
- a SAS Data Set Model onto the Table Viewer (Tableviewer1)

You should now see a list of libraries in the LibrariesListbox. By default the Library List model references the currently defined SAS libraries, even at build time. This is why the first List Box displays the current libraries. The other controls don't display any data because, although they are associated with models, those models have not been told where to look for data. You add SCL code to define those data sources in the next section.

*Figure 4.7*  Display_data Frame after Attaching Models and Setting Attributes

---

**Add SCL Code to the Display_data Frame**

To add the SCL code, making the frame fully functional, open the frame SCL for the Display_data frame (right-click anywhere in the frame and select **Frame SCL**). Insert the following code:

```sas
/* This is the frame SCL for the Display_data frame. */
```
The user selects a library from the LibrariesListbox. The TablesListbox is then populated. The user selects a table from the TablesListbox. The ColumnsListbox and Tableviewer1 are then populated.

dcl num rc;  /* Numerical variable used as a return code. */
dcl char(30) displayTable;  /* A character variable. */
dcl list emptyList={};  /* Creates an empty list. */

/* Executes before the frame is displayed to the user. */
Init:
   /* Disable the SubsetButton and the ClearSubset buttons */
   /* by setting the 'enabled' attribute on each. */
   /* There is nothing yet to subset or clear. */
   subsetButton.enabled='no';
   clearButton.enabled='no';

   /*Set how the table is displayed in TablesListBox. */
   datasetlist1.levelCount=1;
   return;

/*Executes when a selection is made from the LibrariesListbox. */
LibrariesListbox:
   if LibrariesListbox.selectedItem ne ' ' then do;
      /* Set the Data Set List model to point to the library selected. */
      /* Because the Data Set List model is associated with the */
      /* TablesListbox, the TablesListbox is populated. */
      datasetList1.library=librariesListbox.selectedItem;
      variableList1.dataSet=' ';
      sasdataset1.table=' ';

      /* Enable the SubsetButton now that there is data to subset. */
      subsetButton.enabled='yes';
   end;
   return;

/*Executes when a selection is made from the TablesListbox. */
TablesListbox:
   if TablesListbox.selectedItem ne ' ' then do;
      /* Concatenate the selected library and the selected table */
      /* and give the result to the sasdataset model, the model */
      /* supplying the Table Viewer with data. */
      displayTable=librariesListbox.selectedItem || '.' ||
      TablesListbox.selectedItem;
      sasdataset1.table=displayTable;
      variableList1.dataSet=displayTable;
      SubsetButton.enabled='yes';
   end;
   return;

/*Executes when a selection is made from the ColumnsListbox. */
ColumnsListbox:
   if listlen(columnsListbox.selectedItems) gt 0 then
/* Copy the list of selected columns to the sasdataset1 model, */
/* the model supplying the Table Viewer with data. */
sasdataset1.columnOrder=copylist(ColumnsListbox.selectedItems);
return;

/* Executes when the 'Subset with WHERE' button is pressed. */
SubsetButton:
  if sasdataset1.table ne ' ' then
    /* Call the WHERE subset window. */
    rc = sasdataset1._setWhere(0, 'y');
    /* If a WHERE expression is in effect, enable the 'Clear Subset' button. */
    if rc=0 then ClearButton.enabled = 'yes';
  return;

/* Executes when the 'Clear Subset' button is pressed. */
ClearButton:
  if sasdataset1.table ne ' ' then
    sasdataset1._setWhere(emptyList); /* Clear the WHERE expression. */
    ClearButton.enabled = 'no'; /* Disable the 'Clear Subset' button. */
  return;

/* Delete the list when quitting. */
rc=dellist(emptyList);
return;

Save the code by selecting File ➪ Save, and then close the frame SCL window.

Because the frame entry was already named Display_data.frame, the SCL entry that is
associated with the frame is automatically named Display_data.scl. If you change the
name of the frame later, the frame assumes that the frame SCL name was also changed,
so you must either change the name of the SCL entry to match the frame, or edit the
frame’s SCLEntry attribute to reference the original SCL entry.

**Compile the Display_data Frame**

With the graphical user interface and the SCL finished, you can now compile the
Display_data frame. To compile the frame, make sure it is the active window, and then
select Build ➪ Compile.

If the frame and frame SCL compiled successfully, you should see in the Log window a
message similar to the following:

NOTE: Compiling DISPLAY_DATA.FRAME (SASUSER.EXAMPLE.DISPLAY_DATA.SCL).
NOTE: Code generated for DISPLAY_DATA.FRAME. Code size=4095.

To view the Log window, select View ➪ Log.

You should correct all errors and warnings before testing the frame (see “Compiling
Applications” on page 19).

**Testing**

**Test the Display_data Frame**

To test the Display_data frame, make sure it is the active window, and then select Build ➪ Test.
After the running frame appears, test it by selecting a library, and then a table. For example, select the Sashelp library and the CLASS table. The table data should appear in the Table Viewer.

![Figure 4.8 The Completed Display_data Frame](image)

Test the WHERE Subsetting
Test the subset capabilities by clicking the **Subset with WHERE** button. The WHERE Expression builder appears.

![Figure 4.9 The WHERE Expression Builder Window](image)

Assuming you're viewing the SASHELP.CLASS table, follow these steps to build a simple WHERE expression to display only the females in the SASHELP.CLASS table:
1. Click **Sex** in the Available Columns list.
2. From the operators list, select **EQ**.
3. Click `<LOOKUP distinct values>` in the Available Columns list.
4. Select F.
5. Click **OK**.

The CLASS data is displayed with only the rows that contain **SEX='F'**.

**Figure 4.10** Results of WHERE **SEX='F'** Subset

Clear the WHERE subsetting by clicking the **Clear Subset** button.

Close the frame by clicking the **Close Window** button.

**Removing the Frame Command Line**

You might have noticed the command prompt at the top of the frame (**Command ====>**). By default all frames have a command line where users can type SAS commands. To remove the command line at the top of a frame, set the `bannerType` attribute on the frame to None. The frame is listed in the Properties window as `_FRAME_`.

After setting the `bannerType` attribute, recompile, and then test the frame again. Close all `Display_data` frames when you are finished testing.

---

**Build the Create_report Frame**

**Adding to the Application**

The second frame to build is the Create_report frame. This frame enables users to define the criteria for and then generate a report based on data in a SAS table.
Build the Graphical User Interface for the Create_report Frame

Create the Create_report frame by entering the following command at the SAS command line:

```sas
build sasuser.example.Create_report.frame
```

To create the graphical user interface for the frame, drag the following controls onto the frame and position them as you see in Figure 4.11 on page 35:

- one Text Label Control
- one Combo Box Control
- two Radio Box Controls
- two Push Button Controls
- one External File Viewer Control

After dragging all the controls to the frame, your frame should resemble this:
Set Attribute Values for the Create_report Controls

Rename the controls so that they have meaningful names in the Properties window. These control names will be used as section labels in the SCL code that is added later.

In the Properties window, set the name attribute of each control as follows:

- Combox1 to **YearCombobox**
- Radiobox1 to **QuarterRadiobox**
- Radiobox2 to **CountryRadiobox**
- Pushbutton1 to **CreateRptButton**
- Pushbutton2 to **CloseButton**
- Externalfileviewer1 to **ReportViewer**

To set the text of the banner at the top of the frame, set the following:

- Textlabel1 font attribute to **Arial, Regular, 16**
- Textlabel1 label attribute to **Sales Reports**

Resize the Textlabel1 control to make the larger text display correctly.

To set the text for the controls that define the report criteria, set the following:

- YearCombobox borderStyle attribute to **Simple**
- YearCombobox borderTitle attribute to **Year**
- QuarterRadiobox borderTitle attribute to **Quarter**
- CountryRadiobox borderTitle attribute to **Country**

To add text and an icon to the CreateRptButton, set the following:

- buttonStyle attribute to **Icon with Text to Right**
- icon attribute to **296**
• label attribute to **Create Report**

Resize the CreateRptButton to make the icon and label display correctly.

To customize the CloseButton, set the following:

• commandOnClick attribute to `end;

  Note the semicolon at the end of the command.

• height attribute to 30

• label attribute to **Close Window**

• width attribute to 80

To prevent the command line from appearing on the frame, set the frame bannerType attribute to **None**.

Lastly, set the items attribute on the YearCombobox so that the years 1993 and 1994 are available to the user. Follow these steps:

1. Scroll to the YearCombobox items attribute.
2. Click in the Value column.
3. Click the ellipsis button in the Value column.

  The List Editor appears.
5. Exit the List Editor by clicking **OK**.

**Figure 4.13  The List Editor**

Lastly, enlarge the External File Viewer so that it is sized similarly to **Figure 4.11 on page 35**.
**Attach Models to the Create_report Frame Controls**

To associate the controls on the frame with the proper models, follow these steps:

- Drop a Variable Values List Model onto the QuarterRadiobox. This Variable Values List Model is automatically named Variablevalueslist1.
  
  In the Properties window, set the following attributes on Variablevalueslist1:
  
  - dataset attribute to `sashelp.prdsale`
  - variable attribute to `Quarter`

- Drop a Variable Values List Model onto the CountryRadiobox. This Variable Values List Model is automatically named Variablevalueslist2.
  
  In the Properties window, set the following attributes on Variablevalueslist2:
  
  - dataset attribute to `sashelp.prdsale`
  - variable attribute to `Country`

The two Radio Box controls should now have values in them.

Resize the controls on the frame if necessary. Sometimes the initial layout of the controls is not conducive to the data they present (for example, Radio Boxes that are too short for the data they contain). Expand the Radio Boxes vertically to make them resemble Figure 4.11 on page 35.

**Add SCL Code to the Create_report Frame**

Add the following code to the Create_report frame SCL. After adding the code, save it and then close the frame SCL window.

```scl
/* This is the frame SCL for the Create_report frame. */
/*                                                    */
/* The user selects a year, a quarter, and a country, */
/* and then clicks the CreateRptButton. Data that     */
/* matches the user selections is pulled from         */
/* SASHELP.PRDSALE, written to a file, and then       */
/* displayed in the External File Viewer.             */

dcl num rc; /* Numerical variable used as a return code. */
dcl list messageList={}; /* Creates an empty list. */

dcl char(?) countryName,
    char(1) quarterValue,
    char(4) yearValue,
    char(2) command; /* Declare character variables. */

INIT:
/* Define a warning message. */
rc=insertc(messageList, 'To create the report, please ' ||
   'select values for year, quarter, and country.');

/* Assign a fileref to an external file. */
rc=filename('out', ' ', 'temp');
```
/* Turn off 'End of file' message. */
ReportViewer._showEndOfFile('no');
return;

/* Executes when you click on the CreateRptButton */
CreateRptButton:
  /* If a fileref is already assigned to ReportViewer, clear the fileref. */
  if ReportViewer.fileref ne ' ' then ReportViewer.fileref=' ';

  /* Initialize variables with user selections. */
countryName = CountryRadiobox.selectedItem;
yearValue = YearCombobox.selectedItem;
quarterValue = left(QuarterRadiobox.selectedItem);

  /* If all criteria have been selected, create a report. */
  if countryName ne ' ' and
      yearValue ne ' ' and
      quarterValue ne ' ' then
    do;
      submit continue;

        /* Close the default ODS destination. */
        ods html close;

        /* Enable the ODS listing destination. */
        ods listing device=listing;

        /* Redirect SAS output to the temp file. */
        proc printto print=out new;

        /* Suppress printing the PROC title. */
        ods NOPROCTITLE;

        /* Set options to control procedure output. */
        options nodate nonumber nocenter;

        /* Create a summary report of the PRDSALE table using */
        /* selected values for Country, Year, and Quarter. */
        proc means data = sashelp.prdsale nonobs sum;
          where country = '&countryName' and
            year = &yearValue and
            quarter = &quarterValue;
          class product;
          var predict actual;
          title1 'Sales Figures for &countryName: ';
          title2 'Quarter &quarterValue in &yearValue';
          run;

        /* Redirect SAS output back to the default location. */
        proc printto;
        run;

        /* Reset options that control procedure output. */
        options date number center;
        endsSubmit;
        ReportViewer.fileref = 'out';
end;
/* If a criteria was not selected, display a warning dialog. */
else command = messagebox(messageList, '!', 'O',
    'Application warning message');
return;

TERM:
/* Delete the SCL list. */
messageList=dellist(messageList);

/* Clear the fileref assigned to ReportViewer. */
ReportViewer.fileref=' ';

/* Clear the fileref OUT. */
rc=filename('out', ' ');
return;

Note: In SAS 9.3 the default output, known as the destination, in the SAS Windowing environment was switched to HTML. Because the External File Viewer control used in this example cannot display HTML, the code above includes statements to disable the HTML destination and enable the listing destination.

Compile the Create_report Frame

To compile the Create_report frame, make sure it is the active window, and then select Build ➜ Compile.

If the frame and SCL code compiled successfully, you should see messages similar to these in the Log window:

NOTE: Compiling CREATE_REPORT.FRAME (SASUSER.EXAMPLE.CREATE_REPORT.SCL).
NOTE: Code generated for CREATE_REPORT.FRAME. Code size=4095.

To view the Log window, select View ➜ Log.

You should correct all warnings and errors.

Test the Create_report Frame

Although you tested the previous frame, Display_data, with the Build ➜ Test menu command, the Create_report frame is different because its frame SCL code contains a SUBMIT block.

The command that is run to execute a frame when you select Build ➜ Test is not capable of running SUBMIT blocks. If you try to test the Create_report frame using the Build ➜ Test menu command, the program will generate a run-time error when you click the Create Report button. Instead, you must test this frame from outside the SAS/AF build environment.

To run the Create_report frame from a SAS Explorer window, follow these steps:

1. Close the Create_report frame.
2. Click the Explorer tab (in the lower-left corner of the main SAS window).
3. Navigate to the Create_report frame (inside SASUSER.Example).
4. Right-click the Create_report frame, and then select Run.
When the frame is running, test it by selecting a Year, a Quarter, and a Country, and then clicking the Create Report button.

**Figure 4.14** Completed Create_report Frame

If you look in the Log window while the report is being created, you can see the SUBMIT block commands running.

Close all the Create_report frames after you have finished testing.

---

**Build the Start_menu Frame**

**Creating a Navigation System**

The final frame to build is the navigation system for the other two frames. When complete, the Start_menu frame enables a user to call either the Display_report frame or the Create_report frame with the click of a button.

**Figure 4.15** Finished Start_menu Frame
**Build the Graphical User Interface for the Start_menu Frame**

Create the Start_menu frame by entering the following command at the SAS command line:

```
build sasuser.example.Start_menu.frame
```

Resize the frame so that it is about half the default size.

Drag the following controls to the frame:
- one Text Label Control
- three Push Button Controls

You should now have a frame that looks something like this:

![Preliminary Start_menu Frame](image)

**Figure 4.16 Preliminary Start_menu Frame**

**Set the Attribute Values for the Start_menu Controls**

For the Textlabel1 object, set the following:
- label attribute to *Sales Viewer*
- font attribute to *Arial, Regular, 14*

Resize the Textlabel1 control so that the text displays properly.

For Pushbutton1, set the following:
- buttonStyle attribute to *Icon with Text Under*
- height attribute to *60*
- icon attribute to *212*
- iconStyle attribute to *Large Icons*
- label attribute to *Display Sales Data*
- name attribute to *DisplayDataButton*
- width attribute to *120*
For Pushbutton2, set the following:

- buttonStyle attribute to **Icon with Text Under**
- height attribute to **60**
- icon attribute to **335**
- iconStyle attribute to **Large Icons**
- label attribute to **Create Sales Report**
- name attribute to **CreateRptButton**
- width attribute to **120**

For Pushbutton3, set the following:

- label attribute to **Exit Application**
- name attribute to **ExitButton**
- width attribute to **100**

Lastly, set the bannerType attribute of the frame to **None**.

Arrange the controls so that they resemble **Figure 4.15 on page 41**.

**Add SCL Code to the Start_menu Frame**

Add the following code to the Start_menu frame SCL. When complete, save the SCL and then close the Frame SCL window.

DisplayDataButton:
```scl
call display('Display_data.frame');
return;
```

CreateRptButton:
```scl
call display('Create_report.frame');
return;
```

ExitButton:
```scl
/* This code displays a confirmation dialog box */
/* when a user clicks the ExitButton. */

/* Create a list that contains the text of the message. */
dol list message={'Are you sure you want to exit?', 'Be honest.'};

/*Use the SCL MESSAGEBOX function to display a YES/NO dialog box. */
response=messagebox(message, '!', 'YN', 'Confirm Exit');
if response='YES' then call exec commanding('end;');

/*Delete the list 'message'. */
message=dellist(message);
return;
```

**Test the Start_menu Frame**

Test the Start_menu frame by selecting **Build ⇒ Test**. Ensure that the **Display Sales Data** and **Create Sales Report** buttons call the appropriate frame, and that the **Exit Application** button performs as expected.
Test the Entire Application

To test the entire application, each frame must be compiled individually. And because the Create_report frame uses a SUBMIT statement, you must test the application from outside the build environment (recall the limitation of the testing environment in “Testing Applications” on page 20).

To test the entire application, follow these steps:

1. Close the Start_menu frame (and any other open frames).
2. Run the Start_menu frame from a SAS Explorer window.
3. Test the functionality of each control on each frame.

If you encounter a frame or control that does not function properly, exit the application, compile the offending frame, and then test, diagnose, and fix the problem separately from the larger application.
Part 3

Appendixes

Appendix 1
  Defining Attachments ...................................................... 47

Appendix 2
  Deploying Applications .................................................... 57

Appendix 3
  Defining a Subclass ....................................................... 61
Appendix 1
Defining Attachments

Understanding Attachments

Introduction
Attachments control the spatial relationships between the graphical user interface elements on a window when the window is resized. For example, with the appropriate attachments defined, you can make a control expand when the user enlarges the window, enabling the user to see more data. With a few simple attachments you can greatly increase the utility of an application.

This appendix is a brief introduction to attachments. It guides you through the process of defining attachments so that when a user enlarges the Display_data frame, the Table Viewer on that frame will also grow larger so that more data is visible.
Figure A1.1  Default Width of Display_data Frame

When you define attachments, the controls on a frame are represented by wire outlines.

Control Representation in Attach Mode

When you define attachments, the controls on a frame are represented by wire outlines. Compare the regular, build time TestFrame to the frame while defining attachments.
Attachment Points

You define attachments between a control and the edge of the frame, or between the edge or center of two controls.

An attachment to the edge of a control causes that edge to move when the other end of the attachment (the anchor) is moved. An attachment to the center of a control causes the entire control to move when the anchor for that attachment is moved. Attachments are represented as arrows in the graphical user interface.

In the following display the top Push Button control is attached from its right edge to the right edge of the frame. The lower Push Button control is attached from its center to the right edge of the frame.

As you can see in the following display, when the frame is enlarged horizontally at run time, the top Push Button (Button1) expands to the right (the edge attachment), and the bottom Push Button (Button2) moves with the right edge of the frame (the center attachment).
Attachment Direction and Type

When you start defining attachments, the Define Attachment dialog box appears so that you can select the direction and type of attachment.

Figure A1.6  The Define Attachment Dialog Box

The *direction* of the attachment governs which control changes when the anchor of the attachment is moved. The control that the attachment arrow points to responds to the moving of the anchor. Remember that an attachment can be anchored to the edge or center of another control, or to the edge of the frame. The left arrow direction is the default attachment direction, and it is the only direction that is used in the example later in this appendix.

Table A1.1  Attachment Directions

<table>
<thead>
<tr>
<th>Icon</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Bidirectional" /></td>
<td>Bidirectional</td>
<td>Both controls respond to resizing or moving either control. In effect, both ends of the attachment are anchors.</td>
</tr>
<tr>
<td><img src="image" alt="Single Direction (Right)" /></td>
<td>Single Direction (Right)</td>
<td>The control that the arrow points to responds to a move of the anchor.</td>
</tr>
<tr>
<td><img src="image" alt="Single Direction (Left)" /></td>
<td>Single Direction (Left)</td>
<td>The control that the arrow points to responds to a move of the anchor.</td>
</tr>
</tbody>
</table>

The *type* of attachment defines the distance between the anchor and the arrow either in terms of pixels between the points (an absolute attachment type), or as a percentage of space between the points (a relative attachment type). The absolute attachment type is the default attachment type, and it is the only type of attachment that is used in the example later in this appendix.

Table A1.2  Attachment Types

<table>
<thead>
<tr>
<th>Icon</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Absolute" /></td>
<td>Absolute</td>
<td>Maintains a fixed number of pixels between attachment points</td>
</tr>
</tbody>
</table>
Moving Controls That Are Attached

Before you define any attachments, you should adjust the size of the frame and arrange the controls as you want them to be displayed at run time. Once defined, attachments are honored at build time, and if you move an anchor (on a control or a frame), any attached controls are moved or resized according to the attachments.

However, you can select and move multiple controls at build time while not in attach mode without having attachments honored.

In attach mode you can both move and resize individual controls without attachments being honored. The alignment tools are not available in attach mode.

Deleting and Altering Attachments

You cannot alter the direction or type of an attachment after it has been created. To change an attachment you must delete it and then create a new attachment.

To delete an attachment, follow these steps after you are in attach mode:

1. Select the delete type (the attachment direction is unimportant).

2. Click inside the control where the attachment is pointing to the control. Do not click on the attachment line itself—doing so will not delete the attachment.

For More Information about Attachments

There are many other ways to control screen geometry using attachments. For more information, see the Guide to SAS/AF Applications Development, available at support.sas.com/documentation/onlinedoc/af/.
Define Attachments That Resize the Table Viewer

This example assumes you have completed at least the Display_data frame that was presented earlier in this document (see “Build the Display_data Frame” on page 24).

To make the Table Viewer on the Display_data frame expand and contract according to the size of the frame, define two attachments by following these steps:

1. With no control selected, select Layout ⇒ Attach ⇒ Define Attachment.

   The frame changes into attach mode, the Define Attachment dialog box appears, and the mouse cursor changes.

   *Figure A1.7  The Display_data Frame in Attach Mode*

2. In the Define Attachment dialog box, select the direction and type as indicated (both of these selections are the defaults, so they should already be selected):

3. Attach the bottom edge of the Table Viewer to the bottom edge of the frame by placing the mouse cursor inside the lower edge of the Table Viewer. Then click and drag to the bottom edge of the frame, just above the scroll bar, and then release the mouse button.
4. Make a second attachment from the right edge of the Table Viewer to the right edge of the frame.

There should now be two attachments that point to the Table Viewer control. If your attachments do not resemble the attachments below, delete the attachments that you created and start over (see “Deleting and Altering Attachments” on page 51).

5. Click **OK** in the Define Attachment dialog box.

The frame returns to the regular, build-time view.
Test the Table Viewer Attachments

To test the attachments, save the frame and then test it by selecting Build $\Rightarrow$ Test.

Resize the run-time frame horizontally and then also vertically.

*Figure A1.10  Display_data Frame Expanded Horizontally*

*Figure A1.11  Display_data Frame Expanded Vertically (with Overlapping Controls)*
Define Attachments That Move the Close Window Button

To fix the problem of the Table Viewer being overlapped, you must define two attachments that move the Close Window button as the frame size is changed.

Attachments to the center of a control cause that control to move when the attachment anchor is moved. This is in contrast to the Table Viewer attachments, which are defined on the edges of the Table Viewer, and cause it to expand or contract, but remain in the same location on the frame.

To define two attachments on the Close Window button, follow these steps:

1. With no control selected, select Layout ⇒ Attach ⇒ Define Attachment.
2. In the Define Attachment dialog box, select the direction and type as indicated (both of these selections are the defaults, so they should already be selected):

3. Attach the center of the Close Window button to the bottom edge of the frame by placing the mouse pointer in the center of the Close Window button. Then click and drag to the bottom edge of the frame, just above the scroll bar, and then release the mouse button.
Figure A1.12 Creating an Attachment to the Close Window Button

Note that the attachment arrow points to the center (not the edge) of the Close Window button.

4. Make a second attachment from the center of the Close Window button to the right edge of the frame.

Figure A1.13 The Four Completed Attachments

Save and test the frame again. As you enlarge the frame, the Table Viewer should resize and the Close Window button should move.
Appendix 2
Deploying Applications

Introduction

After you create an application, you can deploy it in such a way that a user can launch it directly, without first having to launch SAS, find the frame, and run it. You can also configure a SAS/AF application to launch as a kiosk-like application, running in the foreground without menus, a title bar, or any of the SAS display manager windows (Program Editor, Log, or Output). When you exit your application, the SAS session automatically ends.

This appendix guides you through the process of modifying a copy of the SAS configuration file and using the INITCMD system option to create an icon that launches the Start_menu frame from the desktop.

These instructions apply to a standard SAS®9.4 installation on the Windows 7 operating system. Consult your SAS host companion for information about the specifics of your operating system.

Edit a Copy of the SAS CFG File

A SAS configuration file contains instructions that are executed each time you start a SAS session. To alter a configuration file so that the Start_menu application is executed, follow these steps:

1. Find the SAS CFG file.
   Assuming a standard SAS installation on Windows, the CFG file can be found at C:\Program Files\SASHome\SASFoundation\9.4\sasv9.cfg.

2. Duplicate the CFG file, and name the new version Startmenu.CFG.

3. Open the Startmenu.cfg file and add to the bottom of the file, after all other lines, the following options:
   -awscontrol notitle
   -noawsmenu
   -initcmd "af c=sasuser.example.Start_menu.frame;
          toolclose;
          command close;
          wstatusln off;
          wwindowbar off;"

4. Save and close the CFG file.

Note that the list of commands in the INITCMD option is enclosed in quotation marks.
The following tables describe the SAS system options and SAS commands that are used in the INITCMD option.

**Table A2.1  SAS System Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>awscontrol notitle</td>
<td>Turns off the system menu, and the minimize and maximize buttons in the main SAS window</td>
</tr>
<tr>
<td>noawsmenu</td>
<td>Turns off the menu bar in the main SAS window</td>
</tr>
<tr>
<td>initcmd</td>
<td>Specifies commands to execute during the start of a SAS session, and suppresses the Log, Output, Program Editor, Enhanced Editor, and Explorer windows when executing a SAS/AF application</td>
</tr>
</tbody>
</table>

**Table A2.2  SAS Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>af c=sasuser.example.start_menu.frame</td>
<td>Executes the SAS/AF Start_menu frame</td>
</tr>
<tr>
<td>toolclose</td>
<td>Turns off the application toolbar</td>
</tr>
<tr>
<td>command close</td>
<td>Turns off the command bar</td>
</tr>
<tr>
<td>wstatusln off</td>
<td>Turns off the status line (normally at the bottom of the SAS window)</td>
</tr>
<tr>
<td>wwindowbar off</td>
<td>Turns off the window bar (normally at the bottom of the main SAS window)</td>
</tr>
</tbody>
</table>

Now you need to create a shortcut that references the Startmenu.CFG file while launching SAS.

**Create a Shortcut to SAS That References Startmenu.CFG**

To create a Windows shortcut that references the Startmenu.CFG file while starting SAS, follow these steps (changing the paths appropriately for your installation):

1. Find the SAS executable.
   Assuming a standard SAS installation on Windows, the SAS executable can be located at `C:\Program Files\SASHome\SASFoundation\9.4\sas.exe`.  
2. Create a new shortcut to the SAS executable, naming it **StartMenu**.
3. Right-click the StartMenu icon, and then select **Properties**.
4. Edit the Target of the shortcut to reference Startmenu.CFG by appending the following text to the end of the existing Target:
   ```
   -config 'C:\Program Files\SASHome\SASFoundation\9.4\Startmenu.CFG'
   ```
The complete Target line should look something like this. It is split over two lines here, but your Target should be on one line:

"C:\Program Files\SASHome\SASFoundation\9.4\sas.exe"
-config 'C:\Program Files\SASHome\SASFoundation\9.4\Startmenu.CFG'

5. Click **OK**.

Test the StartMenu shortcut. The Start_menu frame should launch, and the main SAS window should be full screen, with no title, menus, or toolbars. For more information about other start-up options, refer to your host companion and the Base SAS help.
Appendix 3
Defining a Subclass

Subclassing

Instead of defining the same five attributes every time you need a button to close a frame, you can create a button that has all the attributes that you need already defined. You do this by creating a subclass.

A subclass is derived from another class, called a parent class. The subclass inherits all the attributes and methods of that parent. This means that if you change the value of an attribute or the way a method works on the parent, the same change is propagated to the subclass.

You can also define on the subclass new values for attributes that the subclass inherits. These new values override the values that were inherited from the parent class. For example, if the parent class has a width attribute that is set to 50, you can change that value to 100 on the subclass.

This appendix guides you through the process of creating a subclass of the Push Button class. The new subclass overrides the values for the following attributes:

- commandOnClick
- height
- label
- name
- width

After making the Close Window Button subclass, you can drag and drop it onto a frame and the button will function without any further configuration. Besides eliminating repetitive work, subclassing helps to ensure conformity: you won't be able to make the mistake of labeling one button “Close Window” and another “Exit”.

Create a Close Window Button Subclass

Add the Close Window Button Class to the Components Window

Test the New Close Window Button
Create a Close Window Button Subclass

To create a Close Window Button subclass, follow these steps:

1. Create the new class by entering the following command at the SAS command line:
   ```
   build sasuser.button.closeWindowButton.class
   ```
   The Class Editor appears.

2. Replace the provided description, CLOSEWINDOWBUTTON.CLASS, with **Close Window Button**. This description will be used as the class name in the Components window.

3. For the parent class, enter **sashelp.classes.pushbutton_c.class**, and press ENTER.
   Notice how the properties on the left side of the Class Editor are now available.

**Overriding Attributes**

To customize the class and make it look and behave like the Close Window buttons that are currently on the Display_data and Create_report frames, you must change five attributes. Changing an attribute value on a class at build time to a value other than the
default is called *overriding* an attribute. The first attribute to override is the width attribute.

To override the width attribute, follow these steps:

1. Click on the Attributes for the class in the Class Properties list (on the left side of the Class Editor).
2. Scroll to the width attribute (in the list of attributes on the right side of the Class Editor).
3. Override the width attribute by right-clicking on the width attribute row and selecting **Override**.

![Screenshot of Class Editor](image)

An 'O' appears in the State column for the width attribute, indicating that the attribute is overridden.

![Screenshot of Class Editor showing overridden attribute](image)

4. In the Initial Value column for width, enter `85`.

   The default width of all new Close Window Buttons will be 85 pixels.

Using the same general procedure, override the following four attributes on the Close Window Button class and set them to the indicated value:

- **commandOnClick**: `end;`
  - Note the semicolon at the end of the command.
- **height**: `30`
- **label**: `Close Window`
- **name**: `CloseWindowButton`

When you are finished, close the Class Editor by clicking **Yes** in the confirmation dialog box to save your changes.
Add the Close Window Button Class to the Components Window

Now that you have a new Close Window Button class, you need to make it accessible at build time so that you can drag it to and drop it onto a frame. To make the new button subclass appear in the Components window, follow these steps:

1. Create a frame or open an existing frame.
   The Components window appears.

2. Right-click in the Components window and select Add Classes.

3. Enter `sasuser.button.closeWindowButton.class` or navigate to the class and select it.
   The class appears at the top of the Components window. The class will remain in the Components window, even between SAS sessions, until you remove it.

4. Close the frame without saving it.

You can also drag a class from a SAS Explorer window to a frame. Doing so means that you do not have to add the class to the Components window.

Test the New Close Window Button

To test the new Close Window Button, delete the existing Close Window button on the Display_data frame, and then add the new Close Window Button to the frame. Because you set all the attributes on the Close Window Button subclass, the button will work without any additional configuration (although you will have to redefine the attachments to the Close Window Button).

If you want to change something about the Close Window Button, in most cases you only have to make a change to the class, and the change is propagated to all the instances of the Close Window Button in all frames that use it.

A complete discussion of object-oriented development is beyond the scope of this appendix, but creating this simple subclass demonstrates how useful subclassing can be. For more information about object-oriented development in SAS/AF, see the *Guide to SAS/AF Applications Development*. 
active window
the window to which keyboard input is directed. Only one window can be active at a time.

argument
a value that is provided or returned when calling a method.

attachment
a way to control the position, size, and movement of controls on a frame.

attribute
a characteristic of a component such as its name, color, size, or the data it references. See also Property.

banner
the command prompt in the upper-left corner of a frame. Controlled by the bannerType attribute on the Frame class.

build environment
the tools and windows in SAS/AF that are used to construct frame applications.

build time
the period when a program is in the process of being built; when it is not executing.

catalog
See SAS catalog.

catalog entry
See SAS catalog entry.

class
a template for an object. A class defines all of an object's characteristics (attributes) and the operations (methods) that the object can perform. See also object.

compile
to translate SCL code and frame objects into a form that can be executed.

component
a control or a model.
control
da type of component that is visual in nature; as opposed to a model, which has no visual representation. For example, Check Boxes, List Boxes, and Push Buttons. Controls are also sometimes called visual components. See also model.

dot notation
da syntax for accessing the properties of a component. In dot notation, the object is separated from the property by a period, which is called a dot. For example, the following syntax would call the _deselectAll method on the listbox1 object:
listbox1._deselectAll( )

frame
an area that contains controls; analogous to a window.

frame entry
the SAS catalog entry type for a frame.

frame SCL
SCL that is associated with a specific frame.

handle
a graphic representation on the edge of a frame or a control that designates areas that are used to resize or move the frame or control.

INIT
a reserved SCL label that indicates the initialization code of an SCL program.

initialization
the first stage of frame execution before a frame is displayed. Typically used to declare and set variables and define data sources.

label
See SCL label.

labeled section
one or more SCL statements that are identified by an SCL label. See SCL label.

mainframe
a high-performance computer made for multi-user, processor-intensive computing. Typically used by businesses and for scientific research.

method
an action that is defined for a class. For example, the action of deselecting all items is defined as the _deselectAll method on the List Box class.

model
a type of component that provides attributes and methods for querying and modifying data sources. For example, a SAS Data Set model contains methods for reading and manipulating SAS tables.

native control
the presentation of a control such that it appears to have been built specifically for a platform. For example, a control built on a UNIX system and then ported to Windows 7 will be indistinguishable from other Windows 7 controls.
non-visual component
another term for a model.

object
a specific instantiation of a class. For example, when you drag a List Box control (a class) onto a frame, you create the listbox1 object. The terms object and instance are often used interchangeably.

parent class
the class from which another class is derived.

PROC
See SAS procedure.

procedure
See SAS procedure.

property
any of the characteristics of a component that collectively determine the component's appearance and behavior. Both attributes and methods are types of properties.

run time
the period when a program is executing.

SAS catalog
a SAS file that stores many different kinds of information in smaller units called catalog entries. A single SAS catalog can contain several different types of catalog entries. See also SAS catalog entry.

SAS catalog entry
a storage unit within a SAS catalog. Each entry has an entry type that identifies its purpose to SAS.

SAS Component Language (SCL)
See SCL (SAS Component Language).

SAS procedure
a program that is accessed with a PROC statement. SAS procedures can be used to produce reports, manage files, or analyze data. Many procedures are included with the Base SAS software.

SCL (SAS Component Language)
a programming language that is provided with SAS/AF software. You can use SCL to develop interactive applications that manipulate SAS data sets and external files.

SCL label
a word that indicates the beginning of a section of SCL code.

subclass
a class that is derived from another class. The subclass inherits the attributes and methods of its parent class, and can also possess its own unique attributes and methods. Technically, almost all SAS/AF classes are subclasses because they are derived from other classes.

subclassing
the process of deriving a new class from an existing class. See also subclass.
SUBMIT
a command that causes SAS to compile and execute a program.

TERM
a reserved SCL label that indicates the code that is run when an SCL program ends.

visual component
another term for a control.

WHERE expression
one or more criteria for retrieving data.
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore
for additional books and resources.