Overview 865
Events 866
Interfaces 866
Using the Composite Class 866
  Creating Composite Classes 866
  Using an Alternate Definition 867
  Creating a Composite Object by Subclassing 867
Methods 873
Dictionary 874

Overview

The Composite class allows you to design custom widgets that are made up of combinations of other widgets, such as text entry fields, graphics, or control objects.

A composite object consists of a container fitted around a group of widgets, called the components of the composite object. The composite object is the owner of its components.

Although individual components of a composite cannot be accessed from a composite object in a frame, the composite can take on the value of a single component. Each component of a composite automatically generates an instance variable that can be used in SCL programs to send methods to the component. In addition to assembling components, you can

- change the definition of a component. The changes appear when you edit or run a frame containing an existing composite object of that class.
- forward an existing method from the widget to the component without having to provide SCL code.
- define attachments between the components and the composite container to control the size and space between components when the composite is resized.
- create an alternate, text-based version of the composite for applications that run on nongraphics devices.

Parent:
  sashelp.fsp.widget.class
Class:
  sashelp.fsp.Composit.class
Events

There are no events defined for the Composite class.

Interfaces

There are no interfaces defined for the Composite class.

Using the Composite Class

The Composite class is not on the Selection List displayed by the Make action in the frame because the Composite class is the model with which you create composite objects. Once you create a composite object, you can add it to the RESOURCE entry so it appears on the Selection List.

Creating Composite Classes

There are two techniques to creating composite classes:

- Select Save as composite from the region pop-up menu to save a group of widgets already created in a frame. Use this technique if you have created several objects and decide to use this combination again in other applications:
  1. Create the widgets that you want the composite object to contain.
  2. Drag a region completely surrounding the objects.
  3. With the region active, select Save as Composite from the region pop-up menu. The Class Entry window opens.
  4. In the Class Entry window, specify the name and description of the new class; close the window and save the class.
  5. To change the composite object once you have saved it, edit the CLASS entry.
  6. Add the new composite class to a RESOURCE entry so it will appear on the Selection List (see SAS/AF online help for details).

  Note: When you make a composite class in this way, the original group of widgets is not itself a composite object but instead serves as a model for the new composite class with which you can create composite objects in the future.

- Create a new CLASS entry that is a subclass of the Composite class. In the CLASS entry window, define the widgets, instance variables, and methods that make up the composite object. The steps for creating a composite subclass are nearly identical to creating subclasses of other objects, except that you choose and arrange the widgets that are the components of the composite object (see SAS/AF online help for details).
  1. Create a CLASS entry for the new composite class with the BUILD command.
  2. In the Class Entry window, assign SASHELP.FSP.COMPOSIT as the parent.
  3. Define the appearance of the composite object by selecting Edit attributes to open the Composite Attributes window.
  4. Select Edit Definition to open the Composite Definition window.
5 Use the Selection List from the region pop-up menu to create the widgets inside the composite definition container.
6 Close the window to save the definition and return to the Class Entry window.
7 If desired, modify the new class:
   - Edit the class instance variables (see SAS/AF online help for details).
   - Edit the class methods (see SAS/AF online help for details).
   - Specify the name of the custom attribute window (see SAS/AF online help for details). Define a new custom attributes window if necessary.
8 Close the Class Entry window.
9 Add the new composite class to a RESOURCE entry so it will appear on the Selection List (see SAS/AF online help for details).
   For a complete example of this technique, see “Creating a Composite Object by Subclassing” on page 867.

---

Using an Alternate Definition

If your composite widget might be used in an application that runs on a nongraphics terminal, you can specify an alternate definition for the composite widget. The alternate definition is a separate set of components that is automatically substituted for the base definition when running on a nongraphics device. A one-to-one correspondence between the objects in the base and alternate definitions is not required.

For example, a base definition comprised of the graphical objects Image Icon and Extended Text Entry will not display on a nongraphics device. However, you can create an alternate definition consisting of a Block and a Text Entry field, which are text-based objects. In the Composite Definition window, select the composite object and choose Alternate Definition from the pop-up menu. The original definition disappears, leaving an empty composite definition container in which you create the new definition. To return to the original version, select Base definition from the pop-up menu.

When writing the code for the Composite class methods, you can call the _isAlternate method to determine which definition is active. Then structure the SCL statements to distinguish between the two definitions.

---

Creating a Composite Object by Subclassing

This example creates a composite class called Combo Field that consists of two components: a text entry field and a control object. The control object is associated with an SLIST entry so that when the control is selected, the Combo Field displays the list of items stored in the entry. When an item is selected from the list, it is displayed in the text field. In this example, the Combo Field object selects colors stored in an SLIST entry.

1 Create a new CLASS entry for the composite subclass by entering this command:

   build sasuser.combocat.combofld.class

   COMBOFLD.CLASS is the name of the Combo Field subclass. In the Class Entry window set the parent class to SASHELP.FSP.COMPOSIT, the name of the Composite class. COMBOFLD.CLASS will inherit methods from its parent class. Change the description to Combo Field.

2 Define the new composite object:
Create the component objects: First, select **Edit attributes** to open the Composite Attributes window. Then, select **Edit Definition** to display the Composite Definition window. In the container box, create a text entry field with a control object to its right. Assign these attributes:

<table>
<thead>
<tr>
<th>Object</th>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>for the Text Entry Field object</td>
<td>Name</td>
<td>TEXT</td>
</tr>
<tr>
<td></td>
<td>Length</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Allow field growth</td>
<td>on</td>
</tr>
<tr>
<td>for the Control object</td>
<td>Name</td>
<td>CONTROL</td>
</tr>
<tr>
<td>for the Region for each object and for the composite container</td>
<td>Outline type</td>
<td>Simple</td>
</tr>
<tr>
<td></td>
<td>Colors</td>
<td>Foreground</td>
</tr>
</tbody>
</table>

Provide attachments between the container and its components so that growing or shrinking the composite horizontally causes the text to grow or shrink with it, but leave the control object the same size (for more information on attachments, see SAS/AF online help).

- Select the container, and from the Actions menu choose **Define attachments**.
- For Type, select **Absolute**. Then apply these attachments:

<table>
<thead>
<tr>
<th>From...</th>
<th>To...</th>
<th>Apply this attachment</th>
</tr>
</thead>
<tbody>
<tr>
<td>left edge of the text entry field</td>
<td>left edge of the container region</td>
<td>Left directional</td>
</tr>
<tr>
<td>right edge of text entry field</td>
<td>the left edge of the control object</td>
<td>Right directional</td>
</tr>
<tr>
<td>right edge of the control object</td>
<td>right edge of the container region</td>
<td>Left directional</td>
</tr>
</tbody>
</table>

- On the control object, open the pop-up menu and select **Set horizontal fixed**.
- Click **OK** to close the Define Attachments window.

Shrink the container so that it fits tightly around the component objects. Select the container and from the pop-up menu select **Snug fit**. In the Snug Fit window, change **Clearance value** from 4 to 1. Click **OK** to exit the window.

Close the Composite Definition window and return to the Composite Attributes window.

Assign a value to Combo Field by specifying the text entry field component as the source of its value. When a Combo Field object is created in a frame, it will display the value of the text entry component.
In the Composite Attributes window, click the arrow in the **Source component** field and select TEXT - Text Entry. This establishes a direct, one-to-one relationship between the value of the Combo Field object and its text entry component.

Click **OK** to exit the Composite Attributes window and return to the Class Entry window.

4 Specify the Combo Field class instance variables:

- Open the Instance Variables window, which contains a list of all of the instance variables defined for this class. The new variables TEXT and CONTROL correspond to the components of the Combo Field object and were automatically added when the Composite Definition was saved. Each component's identifier list is added as an automatic instance variable of the composite class.

- Add new instance variables that pass information and attributes to the Combo Field. Select **Add Mode On** from the Actions menu. For each new variable, enter the following variable information and select **New**:

<table>
<thead>
<tr>
<th>Name</th>
<th>Initial Value</th>
<th>AutomaticType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTRY</td>
<td>blank</td>
<td>Yes</td>
<td>Char specifies the name of the SLIST catalog entry containing the list of pop-up menu items</td>
</tr>
<tr>
<td>ITEMLIST</td>
<td>blank</td>
<td>Yes</td>
<td>List specifies the SCL list containing the pop-up menu items</td>
</tr>
<tr>
<td>TCOLOR</td>
<td>9</td>
<td>Yes</td>
<td>Num specifies the color number of the text entry field; the initial value, 9, is black</td>
</tr>
<tr>
<td>CCOLOR</td>
<td>BLACK</td>
<td>Yes</td>
<td>Char specifies the color of the control object; the initial value is black</td>
</tr>
</tbody>
</table>

The values of the ENTRY, TCOLOR, and CCOLOR variables are set in the custom attribute window (see step 8).

Click **OK** to exit the Instance Variable window.

5 Customize Combo Field Class methods to handle the initialization and display of the list of items:

- Select **Methods** to bring up a list of all of the methods defined for this class.

- Select the _init method. In the **Source Entry** field, specify SASUSER.COMBOCAT.COMBOFLD.SCL as the location for the new SCL code for the method. Specify INIT as the SCL entry label.

- Repeat the last step for the _select and _bupdate methods. Specify the SCL labels SELECT and BUPDATE and use the same source entry, SASUSER.COMBOCAT.COMBOFLD.SCL.

6 Provide the SCL code for the Combo Field class methods. Because the three methods use the same source, you can enter the code for all three in the same SCL program.

In the Methods window, select any overridden method and select **Edit Source** from the Actions menu. Enter the following SCL code:
length entry $ 36;
INIT: method;
call super(_self_, '_init');

Fill ITEMLIST with the items from the SLIST entry specified by the ENTRY variable.

\[
rc = \text{fillist}(\text{‘CATALOG’, entry, itemlist});
\]

If the list population fails, output an error message.

\[
\begin{align*}
\text{if} \ (rc \ ^= 0) \ \text{then do;} \\
\quad \text{name} &= \text{getnitemc}(_\text{self_}, \text{‘NAME’}); \\
\quad \text{msg} &= \text{‘ERROR: Item list could not be created for ‘} || \text{name}; \\
\quad \text{call send(_frame_, ‘_setMsg’, msg)}; \\
\quad \text{end}; \\
\end{align*}
\]

endmethod;

SELECT runs when any component is either modified or selected.

length current $ 8;
SELECT: method;
call super(_self_, ‘_select’);

Get the name of the component that was selected and store it in the SCL variable CURRENT.

\[
\text{call send(_self_, ‘_getCurrentName’, current);} \\
\]

If the control object was selected, display the list of items in a pop-up menu.

\[
\begin{align*}
\text{if} \ (\text{current} = \text{‘CONTROL’}) \ \text{then do;} \\
\quad \text{sel} &= \text{popmenu(itemlist)}; \\
\end{align*}
\]

If an item is selected from the pop-up menu, display the text of that item (ITEM) in the text entry field (TEXT).

\[
\begin{align*}
\text{if} \ (\text{sel} > 0) \ \text{then do;} \\
\quad \text{item} &= \text{getitemc(itemlist, sel)}; \\
\quad \text{call send(text, ‘_setText’, item)}; \\
\quad \text{end}; \\
\end{align*}
\]
endmethod;

Update the colors of the text and control object after OK or End from the custom attribute window.

length ccolor $ 8;
BUPDATE: method;

Text entry and control object colors are updated based on the values of the TCOLOR and CCOLOR variables.

call send(text, '_setColorNum',
tcolor);
call send(control, '_setColor',
ccolor);
call super(_self_, '_bupdate');
endmethod;

Compile and save the SCL entry.

7 Forward color specification methods to the components:
- Forward the Text Entry method _setColorNum to the text entry component and rename it SET_textCOLOR. When you invoke the SET_textCOLOR method, it passes its arguments directly to the _setColorNum method of the Text Entry widget. Therefore, the argument list for SET_textCOLOR is identical to the argument list for _setColorNum.
  - Select Add Mode On from the pop-up menu in the Methods window.
  - Enter SET_textCOLOR in the Name field.
  - In Forward To Object, turn on Active. Then, in ID variable, click the arrow and select TEXT - Text Entry. In Method, enter _setColorNum.
  - Click NEW to add to the Methods list.
  - Repeat this process to forward the Control object method _setColor to the control component and rename it SET_COLOR.
  - Click OK to exit the Methods window.

8 Specify a custom attribute window for the Combo Field object so that the user can specify a list for the pop-up menu and select colors for the text and arrow:
- Open the Set Custom Attributes window.
- Enter SASUSER.COMBOCAT.CMBOATTR.FRAME in the Custom Attributes field, and then select Replace supplied attribute window.
- Click OK to exit the Set Custom Attributes window and again to exit the CLASS Entry window and save the COMBOFLD.CLASS entry.

9 Create the Custom Attributes Window.
- Make the frame by entering this command:
  build sasuser.combocat.cmboattr.frame
- Add four text entry fields and assign these attributes to the fields:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Color</th>
<th>Length</th>
<th>Pad Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Char</td>
<td>Foreground</td>
<td>8</td>
<td>blank</td>
</tr>
<tr>
<td>ENTRY</td>
<td>Char</td>
<td>Black</td>
<td>36</td>
<td>blank</td>
</tr>
<tr>
<td>TCOLOR</td>
<td>Num</td>
<td>Black</td>
<td>8</td>
<td>blank</td>
</tr>
<tr>
<td>CCOLOR</td>
<td>Char</td>
<td>Black</td>
<td>8</td>
<td>blank</td>
</tr>
</tbody>
</table>

In addition, in the Region Attributes window for each field assign the appropriate titles and set Outline type to Simple and Colors to Black.

- Add two command push buttons, one for the OK command and one for the CANCEL command. Assign these attributes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>4</td>
</tr>
<tr>
<td>CANCEL</td>
<td>8</td>
</tr>
</tbody>
</table>

- Provide the SCL code for the Combo Field Attributes window. Select Locals ➤ Edit SCL
to edit SASUSER.COMBOCAT.CMBOATTR.SCL. Enter the following code:

The ENTRY statement passes the _widget instance variable as an optional argument to the Custom Attributes frame.

```
entry optional = _widget 8;
```

Get the combo field attributes, NAME and ENTRY, as well as color information for the text component (TCOLOR) and color information for the control component (CCOLOR).

```
INIT:
name = getnitemc(_widget, 'NAME');
entry = getnitemc(_widget, 'ENTRY');
tcolor = getnitemn(_widget, 'TCOLOR');
ccolor = getnitemc(_widget, 'CCOLOR');
return;
```

If the user makes no changes to the fields, store the field attributes back on the _widget list.

```
TERM:
if (_status = 'C') then return;
_widget = setnitemc(_widget, name, 'NAME');
_widget = setnitemc(_widget, entry, 'ENTRY');
_widget = setnitemn(_widget, tcolor, 'TCOLOR');
_widget = setnitemc(_widget, ccolor, 'CCOLOR');
```
return;

Compile and save the SCL entry. Close the SOURCE window and the CMBOATTR.FRAME entry and return to the catalog directory.

10 Add the Combo Field Class to a RESOURCE entry so it will appear on the Selection List:
   - Copy the BUILD.RESOURCE entry from the SASHELP.FSP catalog:
     ```
     copy sashelp.fsp.build.resource
     ```
   - Edit BUILD.RESOURCE and select Add from the Actions menu to open the Resource Class List window.
   - Select SASUSER.COMBOCAT.COMBOFLD.CLASS.
   - Click OK to return to the Resource Class List window and again to return to the Catalog Directory.
   - Note: If you make any changes to the CLASS entry, be sure to resynchronize the RESOURCE entry list. Either choose Synchronize from the RESOURCE editor Actions menu, or type y in the selection field next to the entry name in the BUILD directory.

11 Test the Combo Field Class. Open a new frame and create a Combo Field widget. (The Combo Field class will be at the bottom of the Selection List.)
   - In the Combo Field Attributes window enter SASHELP.FSP.COLORNAM.SLIST in SLIST Entry. This list contains color names. Click OK.
   - In the frame, select Locals ▶ Testaf.
   - Click the control to display the list of colors:
   - Select a color. The list closes and the color name appears in the field.

## Methods

Methods specific to the Composite class are described here. Inherited methods are described in the Object class and in the Widget class.

The following Widget methods are sent to all components when received by the composite:

- _gray
- _hide
- _popups
- _protect
- _setCursorShape
- _ungray
- _unhide
- _unprotect
Dictionary

_bupdate

Updates an object upon exiting the object attribute window

Inherited from Widget

Syntax
CALL SUPER(_self_.bupdate);

Details
Because the component objects are not saved with the frame and are re-created every
time the frame is edited or invoked, _bupdate also runs during the composite's _init (or
_binit) method. Therefore, when you override the _init (or _binit) method and perform a
CALL SUPER, the code for the _bupdate method runs before the code that follows the
CALL SUPER in the _init method.

Override the _bupdate method when you need to initialize or reset component
attributes (which usually should be stored as instance variables in the Composite class).

_getCurrentName

Returns the name of the component that was last selected, deselected, or modified

Syntax
CALL NOTIFY (composite-name, '_getCurrentName', component-name);

Argument Type Description

| component-name | C | returns the name of the component |

Example
See the example for _getWidget.

_getCurrentWidget

Returns the object identifier of the component that was last selected, deselected, or modified
**Syntax**

CALL NOTIFY (composite-name, '_getCurrentWidget', object-id);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object-id</td>
<td>N</td>
<td>returns the object identifier of the component</td>
</tr>
</tbody>
</table>

**_getText**

Retrieves the character value assigned to a composite object

**Syntax**

CALL NOTIFY (composite-name, '_getText', text-value);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text-value</td>
<td>C</td>
<td>returns the character value</td>
</tr>
</tbody>
</table>

**Details**

If a composite has been assigned a numeric value in the Initial Value window, the _getText method causes a program halt. Use the _getValue method to return the value of a composite that has a numeric value.

**_getValue**

Retrieves the numeric value assigned to a composite object

**Syntax**

CALL NOTIFY (composite-name, '_getValue', composite-value);
### _getWidget

**Returns an SCL list describing a single component in a composite object**

### Syntax

CALL NOTIFY (composite-name, '_getWidget', component-name, object-id);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component-name</td>
<td>C</td>
<td>specifies the name of a component</td>
</tr>
<tr>
<td>object-id</td>
<td>N</td>
<td>returns the object identifier of the component, which is also the identifier of an SCL list containing descriptive information about the component</td>
</tr>
</tbody>
</table>

### Example

This example retrieves the name of the component and passes it to the _getWidget method to get the object identifier:

```
length currname $8;

/* Get the component’s name */
call notify ('compl',
            '_get_current_name_', currname);

/* Use the name to get the component’s object identifier */
call notify ('compl', '_get_widget_',
            currname, objid);
```

### _getWidgets

**Returns an SCL list of all components in a composite object**
### Syntax

CALL NOTIFY (composite-name, '_getWidgets', existing-list-id);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>existing-list-id</td>
<td>N</td>
<td>specifies the identifier of an SCL list that will contain the object identifiers of all the components in the composite object. In each sublist, the list item ‘NAME’ contains the name of the component.</td>
</tr>
</tbody>
</table>

### Details

You can use the list returned by _getWidgets to send a method to all components of the object at once.

### _isAlternate

Returns whether or not the alternate definition is in effect

### Syntax

CALL NOTIFY (composite-name, '_isAlternate', status);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>N</td>
<td>returns a value indicating which definition is in effect:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 alternate definition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 base definition</td>
</tr>
</tbody>
</table>

### _setText

Assigns a character value to a composite object

### Syntax

CALL NOTIFY (composite-name, '_setText', text-string);
_setValue

Assigns a numeric value to a composite object

Syntax
CALL NOTIFY (composite-name, '_setValue', composite-value);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>composite-value</td>
<td>N</td>
<td>specifies the numeric value to assign</td>
</tr>
</tbody>
</table>

Details
 setValue is valid only for a composite object whose value is numeric or whose source component is numeric. If the initial value of the composite is character, _setValue causes a program halt.

If a value is assigned to a composite object in the SCL code for a frame, then the _refresh (or _postInit) method automatically calls the _setValue method.

By overriding _setText, you can make appropriate visual changes to the composite when the user assigns the object a value in SCL code.

_snugFit

Assigns the clearance between the composite container and its component objects

Syntax
CALL NOTIFY (composite-name, '_snugFit', clearance<, units>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearance</td>
<td>N</td>
<td>specifies the numeric value to assign</td>
</tr>
</tbody>
</table>

Details
_snugFit is valid only for a composite object whose value is numeric or whose source component is numeric. If the initial value of the composite is character, _snugFit causes a program halt.

If a value is assigned to a composite object in the SCL code for a frame, then the _refresh (or _postInit) method automatically calls the _snugFit method.

By overriding _setText or _setValue method, you can make appropriate visual changes to the composite when the user assigns the object a value in SCL code.
<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearance</td>
<td>N</td>
<td>specifies the number of units to leave between the container box and the objects it contains</td>
</tr>
<tr>
<td>units</td>
<td>C</td>
<td>specifies the clearance units. Valid values are</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'CHARACTERS'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'CM'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'FONTS'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'INCHES'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'MM'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'PIXELS'</td>
</tr>
</tbody>
</table>

See also the `_new` method in the Class class.