Overview

The Widget class is a utility class that, when combined with the Object and Frame classes, provides the foundation from which all other classes are built. The Widget class is the parent class of all visual objects used to create FRAME entries. Whenever you create an object in a frame, you are creating an instance of a subclass of the Widget class.

SAS/AF software supplies subclasses that inherit the attributes, methods, events, and event handlers provided in the Widget class.

Parent:

sashelp.fsp.Object.class

Class:

sashelp.fsp.Widget.class

Using the Widget Class

You cannot subclass the widget class itself. You can, however, subclass existing visual controls to add required functionality.

Attributes

Public attributes specified for the Widget Class class are described here. See sashelp.fsp.Object.class for inherited attributes.
Dictionary

**backgroundColor**

Returns or sets the background color of a control

Type: Character

Valid Values: (All SAS-supported color values)

Editor: sashelp.classes.colorEditor.frame

---

**borderColor**

Returns or sets the border color of a control

Type: Character

Valid Values: (All SAS-supported color values)

Editor: sashelp.classes.colorEditor.frame

---

**borderLightSource**

Returns or sets the appearance of a simulated light source to provide a shading effect to the border of a control

Type: Character

Valid Values: Upper Left, Upper Right, Lower Left, Lower Right

---

**borderStyle**

Returns or sets the border style of a control

Type: Character
**borderTitle**

Returns or sets the title that appears in the border of a control

**Type:** Character

**Valid Values:**

---

**borderTitleColor**

Returns or sets the color applied to the border title for a control

**Type:** Character

**Valid Values:** (All SAS-supported color values)

**Editor:** sashelp.classes.colorEditor.frame

---

**borderTitleFont**

Returns or sets the font applied to the border title for a control

**Type:** List

**Valid Values:**

**Editor:** sashelp.fsp.fonteditor.scl

---

**borderTitleFontScaling**

Returns or sets the scaling for the border title font for a control
**borderTitleJustification**

Returns or sets the justification of the border title for a control

Type: Character
Valid Values: Left, Right, Center

**borderTitleOffset**

Returns or sets the amount of space with which the border title is offset from a control

Type: Numeric
Valid Values:

**borderWidth**

Returns or sets the width of the border for a control

Type: Numeric
Valid Values:

**CBTFrameName**

Returns or sets the CBT frame that is displayed when a CBT entry is assigned as the help for the object
commandOnClick

Returns or sets the SAS command that executes when the control is selected

Type: Character

Valid Values: 

commandProcessing

Returns or sets a flag indicating how the control processes commands entered by users in the frame

Type: Character

Valid Values: Honor frame attribute, Run object label, Ignore it

cursorTracking

Returns or sets the state of cursor tracking that occurs when the mouse pointer is moved across a control

Type: Character

Valid Values: Yes, No

description

Returns the short description for the object
**dragEnabled**

Returns or sets the state that determines whether the control can be dragged when selected

*Type: Character*

*Valid Values: Yes, No*

**dragInfo**

Returns or sets the information that is transferred from the control when a drag operation occurs

*Type: List*

*Valid Values:*

*Editor: SASHELP.FSP.DRAGDROPINFOEDITOR.FRAME*

**dragOperations**

Returns or sets the type of operations that the control can handle as a drag request

*Type: Character*

*Valid Values:*

*Editor: SASHELP.FSP.DRAGOPERATIONSEDITOR.FRAME*

**dropEnabled**

Returns or sets the state that determines whether the control can serve as a drop site
**dropInfo**

Returns or sets the information that defines the data representations that can be dropped on the control

- **Type:** List
- **Valid Values:**
- **Editor:** SASHELP.FSP.DRAGDROPINFOEDITOR.FRAME

---

**dropOperations**

Returns or sets the type of operations that the control can handle as a drop request

- **Type:** List
- **Valid Values:**
- **Editor:** SASHELP.FSP.DROPOPERATIONSEDITOR.FRAME

---

**enabled**

Returns or sets the state that determines whether the control is enabled

- **Type:** Character
- **Valid Values:** Yes, No

---

**height**

Returns or sets the height of the control (based on sizingUnits)
help

Returns or sets the help topic that appears when the user selects object help (such as What's This? help)

Type: Character
Valid Values:
Editor: sashelp.classes.objectHelpEditor.frame

helpText

Returns or sets the help text that appears on the status line when a user selects and holds a control

Type: Character
Valid Values:

horizontalPosition

Returns or sets the horizontal position (or upper left x) of the control (based on sizingUnits)

Type: Numeric
Valid Values:

mousePointer

Returns or sets the shape of the cursor or mouse pointer for the control
Widget Class

sizingUnits

Type: Character
Valid Values: \sashelp.fsp.mousepointer.slist

---

**popMenuProcessing**

Returns or sets a flag indicating how the control processes popup menu requests

Type: Character
Valid Values: Honor frame attribute, Run object label, Ignore it, Run _popup method

---

**regionOutlineBehavior**

Returns or sets the behavior of the border

Type: Character
Valid Values: None, PushButton, CheckBox, RadioBox

---

**regionOutlineState**

Returns or sets the state of the border

Type: Character
Valid Values: On, Off

---

**sizingUnits**

Returns or sets the units of measure used by sizing and positioning attributes

Type: Character
Valid Values: Pixels, Characters, Inches, CM, MM, Fonts

---

**tooltipText**

Returns or sets the text that appears as a tool tip when the cursor is positioned over the control

Type: Character

Valid Values:

---

**verticalPosition**

Returns or sets the vertical position (or upper left y) of the control (based on sizingUnits)

Type: Numeric

Valid Values:

---

**visible**

Returns or sets the state that determines whether the component is visible

Type: Character

Valid Values: Yes, No

---

**width**

Returns or sets the width of the control (based on sizing units)

Type: Numeric

Valid Values:
Methods

Methods specified for the Widget Class class are described here. See sashelp.fsp.Object.class for inherited methods.

__addDragOp

Adds to the type of operations this object can handle as a drag request

Syntax

return = objectName_addDragOp( );

Details

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>C</td>
<td>U</td>
<td>specifies the type of operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'COPY' (default), 'LINK',</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'MOVE'</td>
</tr>
</tbody>
</table>

__addDragRep

Adds or registers this representation as a valid drag representation

Syntax

return = objectName_addDragRep( );

Details

A drag site is defined the first time a representation is defined for the object. During the drag, the list of representations is matched with the drop sites. The drop is allowed when there is at least one representation match.
_addDropAction

Adds to the type of actions that this object can handle as a drop request

Syntax

return = objectName_addDropAction( );

Details

Some hosts provide special drag requests that display the list of actions when the object is dropped. The action selected is passed to the _validateDropData and _drop methods. This allows a drop site to define more than one action to be performed at drop time.

_addDropOp

Adds to the type of operations that an object can handle as a drop request

Syntax

return = objectName_addDropOp( );

Details

When data are dragged from place to place, the data can be copied or moved or some linkage can be specified. Copy implies that the data are provided with no post-processing. Move implies that the data are provided and the source is removed. Link implies that some mechanism is to be used to keep the source and destination in sync. Programmers must provide the linkage.

_addDropRep

Adds or registers a representation as a valid drop representation

Syntax

return = objectName_addDropRep( );
Details

A drag site is defined the first time a representation is defined for the object. During the drag, the list of representations is matched with the drop sites. The drop is allowed when there is at least one representation match.

_bpostinit

Performs additional processing after the _binit method at design time

Syntax

objectName_bpostinit( );

Details

This method runs automatically for all objects in a frame when you edit or browse a Frame entry. It runs after the _binit method.

_bpreterm

Performs additional processing before the _bterm method runs

Syntax

objectName_bpreterm( );

_bupdate

Updates an object upon exiting the Object Attributes window at design time

Syntax

objectName_bupdate( );
Details
This method runs automatically by the _attributesDialog method when you select OK from the Object Attributes window. The default behavior updates the attributes of an object. For example, changing the color of a text entry object, updating the icon number, or displaying a new Grseg entry for a SAS/GRAPH Output object.

_childPopup
Displays a pop-up menu containing character items in an SCL list

Syntax
objectName_childPopup(itemsList, objectID, selection);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemsList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of a list containing items to display in the pop-up</td>
</tr>
<tr>
<td>objectID</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the object identifier of the object that received the original _popup method</td>
</tr>
<tr>
<td>selection</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the index number of the selection made from the pop-up menu</td>
</tr>
</tbody>
</table>

Details
The _childPopup method is invoked by the _popup method of a child object. The list passed into this method can be modified (such as adding or removing items or making items inactive).

The sequence of events and method calls is
1. A popup event is generated on an object.
2. The object’s _popup method is invoked.
3. A _super_ in the _popup method invokes the _childPopup method on the owning object or frame. (See _getOwner in this class for more information on owning objects).
4. If an owning object was found, a _super_ in the owning object's _childPopup method invokes the _childPopup on the frame.

Since the _super eventually reaches the frame, the code following the _super_ is where the menu selection should be processed.

After the pop-up menu list is displayed, selection contains the selected menu item. If that selection belongs to this method, the selection should be handled and selection should be set to 0 before returning. The list should also be cleaned up (removing items that were added, adding items that were removed, making items active, and so on). It
is important that the list and selection returned from this method be consistent with the list that was passed into the method.

### _clearHelpMode

Clears the help environment

**Syntax**

```c
objectName._clearHelpMode();
```

**Details**

This method runs automatically when the help environment is disabled. The default behavior is to protect objects that were unprotected by _setHelpMode. See _setHelpMode in this class for a complete example.

### _column

Returns the number of a column in which a widget begins

**Syntax**

```c
objectName._column( column );
```

**Argument Type Use Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the left-most column occupied by the widget</td>
</tr>
</tbody>
</table>

**Details**

Example

_column determines the column in which button1 starts and stores the value in the variable col:

```c
button1._column('col');
```
_completeDrag

Completes a drag operation and allows the move operation to remove the object after the drop so no data are lost

Syntax

objectName_completeDrag( representation, operation, dataList, successfulDrop );
objectName_completeDrag( objectID );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>representation</td>
<td>Character</td>
<td>Input</td>
<td>specifies the representation selected for the drop</td>
</tr>
<tr>
<td>operation</td>
<td>Character</td>
<td>Input</td>
<td>specifies the operation selected for the drop</td>
</tr>
<tr>
<td>dataList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of the SCL list that contains the data defined by the representation</td>
</tr>
<tr>
<td>successfulDrop</td>
<td>Character</td>
<td>Input</td>
<td>specifies the cancel flag set in the _validateDropData method: ACCEPT indicates successful completion, CANCEL indicates drop was cancelled</td>
</tr>
<tr>
<td>objectID</td>
<td>(1)</td>
<td>Input</td>
<td>specifies the identifier of an object containing drag and drop data</td>
</tr>
</tbody>
</table>

(1) objectID is sashelp.classes.draganddrop.class

Details

The _completeDrag method is called automatically when a drag operation is completed. Action should be taken only on the source data if the operation is Move and successfulDrop is not ‘Cancel’. Remove any visual changes added in the _startDrag method. If no valid drop occurs, successfulDrop is ‘Cancel’. 

_copyRegion

Copies an object

Syntax

objectName_copyRegion( );
objectName_copyRegion( upperLeftColumn );
objectName_copyRegion( upperLeftColumn, upperLeftRow );
objectName_copyRegion( upperLeftColumn, upperLeftRow, units );
objectName_copyRegion( upperLeftColumn, upperLeftRow, units, objectID );

**Details**
You can specify the position for the copied object with upperLeftColumn and upperLeftRow. If either upperLeftColumn or upperLeftRow (but not both) is missing, upperLeftColumn or upperLeftRow of the copied object is used. If both values are missing, you get an outline of the object to move and place similar to a MAKE operation when editing a Frame entry. For more information on units, see _new in the Class class. The location for the copied object is not necessarily the same as the location of the actual copied widget.

If the object being copied is nongraphical and the row and column position for upperLeftColumn and upperLeftRow attempt to overlay another nongraphical object, you will receive the message

ERROR: Non-graphic region overlap, operator aborted.

---

**_copyToClipboard**

Copies an object to the host’s clipboard

**Syntax**

objectName_copyToClipboard( );

**Details**

The behavior of _copyToClipboard is host dependent. On hosts like the PC that have a clipboard facility, this method renders a bit-mapped image of the directed object and saves the image to the clipboard. From there the image can be manipulated using standard tools on your host.

_copyToClipboard is not supported for non-graphic (text) objects.

---

**_cursor**

Positions the cursor in a widget

**Syntax**

objectName_cursor( );
Details
The _cursor method has the same effect as the Cursor statement, except that the _cursor method may be executed from another Frame, SCL, or Program entry. The textEntry object and the extendedTextEntry object have additional parameters that are described in the _cursor method in the class.

Example

```
if field(‘error’, ‘libname’) > 0 then
    libname._cursor();
```

The field function has the same functionality for Frame entries as it does for Program entries, with the following exception: the field function may only be executed in the SCL program associated with the Frame entry; it may not be executed in other SCL methods.

_cursorTracker

Invoked when cursor tracking is enabled and the mouse pointer is moved across a widget

Syntax

```
objectName._cursorTracker( x, y );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the mouse pointer position in pixels relative to the upper left corner of the widget's region (0,0)</td>
</tr>
<tr>
<td>y</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the mouse pointer position in pixels relative to the upper left corner of the widget's region (0,0)</td>
</tr>
</tbody>
</table>

Details

As you move the mouse to the right, the x value grows. As you move the mouse pointer down, the y value grows. Because region outlines are considered part of the region but are drawn on the outside edge of the region, you can get negative values for x and y when the mouse pointer is on the top or left border of the region. When the mouse pointer is on the bottom or right border of the region, you can also get values for x and y that exceed the region size. This is more noticeable when the region outline width is large.

Some hosts do not allow the _cursorTracker method to be invoked while the mouse pointer is moving across native widgets (for example, list boxes, push buttons, and icons).

Example

Set the cursor shape for the container object, obj1, to a crosshair and enable cursor tracking. The _cursorTracker method prints whether the mouse pointer is on the
region outline or inside the region. The code for _cursorTracker is assumed to be in Sasuser.Example.Methods.scl.

```
length
msg $40; tracker: method x y 8; _self_._getObjectSize(width, height, 'pixels'); if (x < 0 or y < 0 or x > width or y > height) then msg = 'You are on the region border'; else msg = 'You are in the region'; _frame_._setMsg(msg||('|'||x||',||y|')); endmethod;
```

For this code to run, cursor tracking must be enabled. Instead of making a container box subclass, the _setInstanceMethod method is used:

```
init: obj1._setInstanceMethod._cursorTracker
(sasuser.example.methods,tracker); obj1._cursorTrackingOn();
obj1._setCursorShape(3); return;
```

When the mouse pointer is moved over the container box, the message line indicates the following or something similar:

- You are in the region (32,55)
- You are on the region border (-2,46)

---

**_cursorTrackingOff**

Disables cursor tracking for a widget

---

**Syntax**

`objectName_cursorTrackingOff();`

**Details**

Disabling the cursor tracker prevents the _cursorTracker method from being invoked while the mouse pointer is moved across a widget. Cursor tracking can be enabled using the _cursorTrackingOn method and can be queried using the _isTracking method.

---

**_cursorTrackingOn**

Enables cursor tracking for a widget
Syntax
objectManager._cursorTrackingOn();

Details
Enabling the cursor tracker allows the _cursorTracker method to be invoked while the mouse pointer is moved across the widget. Cursor tracking can be disabled using the _cursorTrackingOff method and can be queried using the _isTracking method. Some hosts do not allow the _cursorTracker method to be invoked while the mouse pointer is moving across native widgets (for example, list boxes, push buttons, and icons).

_disableDragDropSite

Completely disables a drag-and-drop site

Syntax
objectManager._disableDragDropSite();

_disableResizeNotify

Disables the _resize method from being invoked when the widget is resized

Syntax
objectManager._disableResizeNotify();

_drop

Invoked when the drop occurs and can perform any function requiring a new frame or application to be run
Syntax

objectName_drop( representation, operation, dataList, dragStart, x, y );
objectName_drop( objectID );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>representation</td>
<td>Character</td>
<td>Input</td>
<td>specifies the representation selected for the drop</td>
</tr>
<tr>
<td>operation</td>
<td>Character</td>
<td>Input</td>
<td>specifies the operation selected for the drop</td>
</tr>
<tr>
<td>dataList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of the SCL list that contains the data defined by the representation</td>
</tr>
<tr>
<td>dragStart</td>
<td>Character</td>
<td>Input</td>
<td>specifies where the drag started: INSIDE (inside the window containing the drop site) or OUTSIDE</td>
</tr>
<tr>
<td>x</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the x location of the drop in pixels</td>
</tr>
<tr>
<td>y</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the y location of the drop in pixels</td>
</tr>
<tr>
<td>objectID</td>
<td>(1)</td>
<td>Input</td>
<td>specifies the identifier of an object containing drag and drop data</td>
</tr>
</tbody>
</table>

(1) objectID is sashelp.classes.draganddrop.class

Details

The _drop method is called automatically after a drag site object has been dropped on a drop site object. The drop site is passed the data returned from the _getDragData method. The _drop method is the last method to be called in the drop sequence. It is to be used to perform any function that might require a new frame or application to be run. Make sure to test the representation and the action before responding to the drop.

X and Y are the pixel offsets of the drop within the drop zone. These values are good only for nonwidget drop zones. Op passed in may not be one of Move, Copy, or Link if a drop action was placed on the drop site. In that case, it can be one of those actions.

_enableDragDropSite

Enables a previously disabled drag-and-drop site

Syntax

objectName_enableDragDropSite( );
_enableResizeNotify

Enables the _resize method to be invoked when the widget is resized

Syntax

objectName._enableResizeNotify();

_errorOff

Turns off a widget’s error state

Syntax

objectName._errorOff();

Details

- You can also use the field function, errorOff function, or errorOff statement to turn a widget’s error attribute off. However, only the _errorOff method can be executed from another Frame, SCL, or Program entry. For example, if you override the _validate method for a subclass of the TextEntry class, the _validate method can turn off the error attribute without knowing the field name.
- For extended table widgets, you can disable the error flag for a particular row of the extended table and for any widget in the extended table. For more information, refer to _errorOff in the ExtendedTable class.
- Generally, when a field is modified, you will unconditionally turn off the error attribute, perform validation, and turn on the error attribute only if the validation fails.
- If any widget has its error attribute set when an END command is issued, afsw does not execute any widget _preterm methods or the TERM label, and the END command does not end the Frame entry. You must issue the CANCEL command instead.

Example
Use the following statement at the beginning of an overridden _validate method to turn off a text entry's error attribute before performing any validation:

```
_self_.errorOff(); /* perform validation... */
```

### _errorOn

**Turns on a widget's error state**

**Syntax**

```
objectName_errorOn();
```

**Details**

- The Field function, ErrorOn function, or ErrorOn statement can also turn on the error attribute. However, only the _errorOn method may be executed from another Frame, SCL, or Program entry. For example, if you override the _validate method for a subclass of the TextEntry class, the _validate method can turn on the error attribute without knowing the field name.
- For extended table widgets, you can enable the error flag for a particular row of the extended table and for any widget in the extended table. For more information, refer to _errorOn in the ExtendedTable class.
- If any widget has its error attribute set when an END command is issued, you must issue the CANCEL command to execute any widget _preterm methods or the TERM label. The END command does not end the Frame entry.
- If any widget has its error attribute set, the MAIN section will not execute unless Control Error is in effect.

**Example**

```
_errorOn turns the error attribute on if the widget's value is greater than 100:

MYVALID: method; if (_value_>100) then
_self_.errorOn()); endmethod;
```

### _getBackgroundColor

**Returns the background color of an object**

**Syntax**

```
objectName_getBackgroundColor( color );
```
objectName_getBackgroundColor( color, isSet );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Character</td>
<td>Output</td>
<td>returns the background color of the widget</td>
</tr>
<tr>
<td>isSet</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating whether the color is set or inherited: 1 if set, 0 if inherited</td>
</tr>
</tbody>
</table>

**Details**

SASCOLOR window element names are not returned. If the background color is set to a SASCOLOR window element, the color name is resolved when the color is set. For example, if the background color is set to the error color and error currently maps to red, then _getBackgroundColor returns red.

---

_objectBorderColor_

**Returns the border color of an object**

**Syntax**

```
objectName_getBorderColor( color );
objectName_getBorderColor( color, isSet );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Character</td>
<td>Output</td>
<td>returns the border color</td>
</tr>
<tr>
<td>isSet</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating whether the color is set or inherited: 1 if set, 0 if inherited</td>
</tr>
</tbody>
</table>

**Details**

SASCOLOR window element names are not returned. If the border color is set to a SASCOLOR window element, the color name is resolved when the color is set. For example, if the border color is set to the error color and error currently maps to red, _getBorderColor returns red.

---

_objectBorderState_

**Returns the border state of an object**

Syntax

objectName._getBorderState(state);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>Character</td>
<td>Output</td>
<td>returns a value indicating the appearance of the border: ON (pushed in), OFF (popped out)</td>
</tr>
</tbody>
</table>

Details

A return value of On means the border has a pushed-in appearance, assuming the region attribute Light Source is Upper left. This method is valid only if the object has a region attribute Outline Type of Button and the Button Behavior is not None. Otherwise, _getBorderState will always return Off.

_getCMD

Returns the SAS command that is executed when the widget is modified or selected

Syntax

objectName._getCMD(command);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>Character</td>
<td>Output</td>
<td>returns a SAS command</td>
</tr>
</tbody>
</table>

Details

The SAS command executes when the user selects or modifies the widget in a frame or if the SCL program otherwise invokes the _select method of the widget. The command is set either by the _setCMD method or through the Command Processing Attribute window.

Example

_getCMD returns the SAS command that is executed when button3 is selected and stores the information in the variable CMD3.

```sas
length cmd3 $200;
button3._getCMD(cmd3);
```
_getCursorShape

Gets the cursor shape (mouse pointer shape) of a widget

Syntax

objectName_getCursorShape( shape );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shape</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a shape number</td>
</tr>
</tbody>
</table>

_getDragData

Exports the data from a drag site object to a drop site object

Syntax

objectName_getDragData( representation, operation, dataList,x, y );
objectName_getDragData( objectID );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>representation</td>
<td>Character</td>
<td>Input</td>
<td>specifies the representation selected for the drop</td>
</tr>
<tr>
<td>operation</td>
<td>Character</td>
<td>Input</td>
<td>specifies the operation selected for the drop</td>
</tr>
<tr>
<td>dataList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of the SCL list that contains the data defined by the representation</td>
</tr>
<tr>
<td>x</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the x location of the drag in pixels</td>
</tr>
<tr>
<td>y</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the y location of the drag in pixels</td>
</tr>
<tr>
<td>objectID</td>
<td>(1)</td>
<td>Input</td>
<td>specifies the identifier of an object containing drag and drop data</td>
</tr>
</tbody>
</table>

(1) objectID is sashelp.classes.draganddrop.class
**Details**

The `_getDragData` method is called automatically after a drag site object is dropped and before the `_validateDropData` method runs on the drop site object. This method runs only on drag site objects. This method provides a way to export the data to the drop site object. The data are passed as a global list. The format of the data depends on the representation passed in. `_dndTEXT`, `_dndFILE`, and `_dndDATASET` representations require a list of character items. All three representations return one or more character strings. When you use the `_getDragData` method, each item in the returned list must be a character item.

The list passed into this method is a global list to facilitate passing data between tasks. Any lists added to `dataList` should also be global lists.

### `_getDragOps`

**Returns the drag operations currently defined for this object**

**Syntax**

```plaintext
objectName_getDragOps( operationsList );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operationsList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of the SCL list that contains the drag operations</td>
</tr>
</tbody>
</table>

**Details**

Note that `operationList` can be modified and returned to the `_setDragOp` method using `callApply` to perform global changes in the drag operations.

### `_getDragReps`

**Returns the drag representations currently defined for this object**

**Syntax**

```plaintext
objectName_getDragReps( representationsList );
```
### _getDropActions

**Description**

Returns the drop actions defined on the widget

**Syntax**

```
objectName_getDropActions( actions );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actions</td>
<td>Numeric</td>
<td>Update</td>
<td>returns the drop actions</td>
</tr>
</tbody>
</table>

### _getDropOps

**Description**

Returns the drop operations currently defined for this object

**Syntax**

```
objectName_getDropOps( operationsList );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operationsList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of the SCL list that contains the drop operations</td>
</tr>
</tbody>
</table>

**Details**

Note that you can modify and return operationsList to the _setDropOp method using CALL APPLY to perform global changes in the drop operations.
**_getDropReps**

Returns a list of the object's currently defined drop representations

**Syntax**

```plaintext
objectName._getDropReps( representationsList );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>representationsList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of the SCL list that contains the drop representations operations</td>
</tr>
</tbody>
</table>

**Details**

Note that you can modify and return representationsList to the _setDropRep method using CALL APPLY to perform global changes in the drop representations.

**_getName**

Invoked when the name attribute is queried

**Syntax**

```plaintext
objectName._getName( name );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Character</td>
<td>Output</td>
<td>returns the name of the widget</td>
</tr>
</tbody>
</table>

**_getObjectSize**

Returns the size of the object's region

**Syntax**

```plaintext
objectName._getObjectSize( width, height );
```
**_getObjectSizeIn_**

_Returns the size of the object’s region in inches_

---

**Syntax**

objectName_getObjectSizeIn( width, height );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the object’s width</td>
</tr>
<tr>
<td>height</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the object’s height</td>
</tr>
</tbody>
</table>

**_getOwner_**

_Returns the object identifier of the owner of an object_

---

**Syntax**

objectName_getOwner( objectID );

objectName_getOwner( objectID, type );
Argument | Type       | Use   | Description
---------|------------|-------|---------------------------------------------
objectID  | Numeric    | Output| returns the object identifier of the owning object; if no owner, returns 0

**Details**

An owner is typically a parent object that enforces some control over the owned object. Examples of owning objects are extended tables, work areas, composites, and organizational charts. By default, _getOwner looks for the first owner of an object.

Example

Suppose a push button is contained within a composite object and that composite object is contained within an extended table. To get the object identifier of the extended table from the push button’s _select method, a type of EXTTBL must be specified:

```plaintext
length
_method_ $40; select: method; _super_._method(); _self_._getOwner(id, 'exttbl'); id._setBorderColor('red'); endmethod;
```

If a type was not specified, the object identifier of the composite would be returned.

---

### _getProperties

**Fills a list with properties of an object**

**Syntax**

```plaintext
objectName_getProperties( propertiesList, units );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertiesList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned is filled with widget properties</td>
</tr>
<tr>
<td>units</td>
<td>Character</td>
<td>Input</td>
<td>specifies the units of the region’s size: CHARACTERS, INCHES, CM, MM, FONTS (default), PIXELS</td>
</tr>
</tbody>
</table>

**Details**

Note that propertiesList is filled from the attributes specified in the Object Attributes window plus attributes specified in the Region Attributes window.

The region attributes are contained in a sublist called _region. The format of this list is described in the _new method in the Class class.
Not all methods that modify an object also modify the values in the list. That is, the list returned by _getProperties reflects the initial data used when the object was initialized and does not necessarily reflect the current state of the object.

_getRegion

Returns an SCL list containing region attributes

Syntax

objectName_getRegion( regionList );
objectName_getRegion( regionList, units );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regionList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned is filled region attribute settings</td>
</tr>
<tr>
<td>units</td>
<td>Character</td>
<td>Input</td>
<td>specifies the units of the region’s size</td>
</tr>
</tbody>
</table>

Details

The list filled by _getRegion contains the same information as the _region list returned by _getProperties. The format of the list is explained in the _new method in the Class class.

Example

Get the region attributes for obj1 and specify the region size in inches:

```plaintext
list = makelist(); obj1.getRegion(list,'inches'); ulx = getnitemn (list, 'ulx'); uly = getnitemn (list, 'uly'); lrx = getnitemn (list, 'lrx'); lry = getnitemn (list, 'lry'); put 'region coordinates in inches is' ulx= uly= lrx= lry=;
```

_getText

Returns the character value assigned to an object

Syntax

objectName_getText( value );
### _getValue

**Argument Type Use Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Character</td>
<td>Output</td>
<td>returns the value of the object (for legacy class support)</td>
</tr>
</tbody>
</table>

**Details**

If the object represents a numeric value, the _getText method causes a program halt. Use the _getValue method to return a numeric value.

This method is mainly used within other methods because the value of an object is not directly accessible. The value of an object is available in the frame's SCL program by using the name of the object.

**Example**

A SAS/GRAPH Output object has a character value whose value represents the name of the Grseg entry being displayed. To get the name of the Grseg entry, use the _getText method:

```plaintext
length text $40; graph-id._getText(text);
put 'The name of the GRSEG entry is' text;
```

---

### _getType

**Returns the type of an object**

---

**Syntax**

`objectName._getType(type);`

**Argument Type Use Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Character</td>
<td>Output</td>
<td>returns the type of object: CHAR, NUM or EMBEDDED</td>
</tr>
</tbody>
</table>

**Details**

Based on the value returned by _getType, you can use either _getText or _getValue to get the value of an object. Objects whose values are list identifiers are numeric.

The text entry object has an optional parameter for the specific type of text entry. For example, a numeric text entry object can be of type Num, Fixed, Short, and so on.
**Syntax**

objectName._getValue( value );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Numeric</td>
<td>Input</td>
<td>returns the value of the object (for legacy class support)</td>
</tr>
</tbody>
</table>

**Details**

If the object represents a character value, the method fails. This method is mainly used within other methods because the value of an object is not directly accessible. The value of an object is available in the frame's SCL program by using the name of the object.

---

**_gray**

Deactivates (grays) a widget so it may not be selected

---

**Syntax**

objectName._gray( );

---

**_hide**

Hides a widget so it is not visible and cannot be selected

---

**Syntax**

objectName.hide( );

objectName.hide( all );

**Details**

- The _gray method is one of two mutually exclusive methods that disable a widget so users cannot modify or select the widget. The other method is _protect for this
class. You cannot use the _gray method to gray a widget that is already protected with the _protect method, and vice versa.

- Use the _gray method if users can select a widget under some circumstances but not under others. For example, the Attributes windows for the provided Widget Classes use the _gray method to inform users that two Additional Attributes options, Object links and Custom attributes, are not available for certain widgets.
- The _gray method has the same functionality as the gray function in SCL, but the _gray method is the only way to gray a widget in a different entry.
- When a widget is grayed, the default _select method does nothing, even if invoked directly from an SCL program.
- The _gray method for the RadioBox class has additional capabilities. See the RadioBox class.

Example
_gray grays a push button named update so it may not be selected by users.
update._gray();

---

**_inError**

Reports whether a widget is currently in error

----------

**Syntax**

`objectName._inError( errorState );`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorState</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating whether the widget is in error: 1 is in error, 0 is not in error</td>
</tr>
</tbody>
</table>

**Details**

This method has the same effect as the field function error check, except that it may be executed from another Frame, SCL, or Program entry.

---

**_isDisplayed**

Reports whether a widget is currently visible in the window
Syntax

```
objectName_isDisplayed( isDisplayed );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isDisplayed</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating whether the widget is displayed: 1 is displayed, 0 is not displayed</td>
</tr>
</tbody>
</table>

Details

You may also use the displayed function to return the displayed status of a widget, but the _isDisplayed method is the only way to determine a widget’s displayed status from another SCL entry such as a method.

_isDisplayed returns a value of 0 if the widget is hidden, swapped out (see _swapOut), or scrolled out of view from a work area object. It also returns a value of 0 for text entry objects if the noDisplay attribute is set.

Example

```
update._isDisplayed('isDisplayed'); if isDisplayed then put ‘the UPDATE button is displayed’; else put ‘the UPDATE button is not displayed’;
```

_isGray

Reports whether a widget in the window is currently active or inactive (grayed)

Syntax

```
objectName_isGray( isGray );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isGray</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating the state of the widget: 1 is inactive, 0 is active</td>
</tr>
</tbody>
</table>

Details

The _isGray method has the same functionality as the isGray function in SCL. However, only the _isGray method can determine whether a widget in a different entry is grayed.

The _isGray method for the RadioBox class has additional capabilities. See the RadioBox class.

Example

```
_isGray determines whether a push button named update is grayed:
```
_isHidden

Reports whether a widget is currently hidden in the window

Syntax

objectName_isHidden(ishidden);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ishidden</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating whether the widget is hidden: 1 is hidden, 0 is not hidden</td>
</tr>
</tbody>
</table>

Details

_isHidden returns only whether a widget is hidden. It does not return whether a widget is swapped out or scrolled out of view.

Example

_isHidden determines whether a widget named update is hidden:

    update._isHidden('isHidden');
    if isHidden then put 'the update button is hidden'; else put 'the update button is visible';

_isModified

Reports whether a widget is modified

Syntax

objectName_isModified(isModified);
### _isModified_

**Argument Type Use Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isModified</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating whether the widget is modified: 1 is modified, 0 is not modified</td>
</tr>
</tbody>
</table>

**Details**

- A widget is modified when users change its field contents or select it with the keyboard keys or mouse.
- When a widget is modified, its status returns to unmodified after the widget label, the _select method, and the MAIN section execute.
- The field or modified function can also return the modified status of a widget, but the _isModified method is the only way to determine if a widget is modified from a different entry such as a method.
- In general, the _isModified method is not needed because a widget’s label runs when the widget is modified.

**Example**

```plaintext
_isModified determines whether update is modified:

update._isModified('isModified'); if isModified then put 'the UPDATE button is modified'; else put 'the UPDATE button is not modified';
```

---

### _isProtected_

Reports whether a widget is currently protected in the window

**Syntax**

```
objectName_isProtected(isProtected);
```

**Argument Type Use Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isProtected</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating whether the widget is protected: 1 is protected, 0 is not protected</td>
</tr>
</tbody>
</table>

**Details**

**Example**

```plaintext
_isProtected determines whether update is protected:

update._isProtected('isProtected'); if isProtected then put 'the UPDATE button is protected'; else put 'the UPDATE button is not protected';
```
_isSwapped

Reports whether the object is swapped out

Syntax

objectName_isSwapped( isSwapped );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isSwapped</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating whether the object is swapped: 0 if swapped, 1 if not swapped</td>
</tr>
</tbody>
</table>

_isTracking

Reports whether cursor tracking is enabled for a widget

Syntax

objectName_isTracking( isTracking );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isTracking</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating whether tracking is enabled: 1 if enabled, 0 if not enabled</td>
</tr>
</tbody>
</table>

Details

Cursor tracking is enabled and disabled by the _cursorTrackingOn and _cursorTrackingOff methods, respectively.

_moveRegion

Moves or resizes an object
Syntax

ObjectName_moveRegion( );
ObjectName_moveRegion( upperLeft );
ObjectName_moveRegion( upperLeftX, upperLeftY );
ObjectName_moveRegion( upperLeftX, upperLeftY, lowerLeftX );
ObjectName_moveRegion( upperLeftX, upperLeftY, lowerLeftX, lowerLeftY );
ObjectName_moveRegion( upperLeftX, upperLeftY, lowerLeftX, lowerLeftY, units );
ObjectName_moveRegion( upperLeftX, upperLeftY, lowerLeftX, lowerLeftY, units, rc );

Details

The position for the moved and resized objects can be specified with ulx (upper left x), uly (upper left y), lrx (lower left x) and lry (lower left y).

If both upper left values and lower right values are specified, the region is positioned and sized at the new coordinates.

If upper left values are specified and lower right values are missing, the object is moved to the location specified by the upper left values. The size of the object is not changed.

If lower right values are specified and upper left values are missing, the lower right of the region is moved to the specified coordinates, causing the object to be resized.

If both upper left values and lower right values are missing, you are given an outline of the object to move and place, similar to adding a component when editing a frame.

Note: The coordinates for the new region position are not necessarily the same as the coordinates of the actual widget. For example, specifying that a push button should be moved to column 10 actually causes the push button to be moved to column 11. The push button's region starts in column 10.

_needRefresh

Marks a widget for a later refresh operation

Syntax

ObjectName_needRefresh( );

Details

If you write new methods for a Widget subclass, your methods may alter widgets such that they are not marked for refresh. For example, your new methods may change the value a widget displays. In this case, if you want the widget to be refreshed when the Frame is refreshed, invoke the _needRefresh method from your method.
Example
The following example marks the widget named stickerPrice to be refreshed and then calls the Frame entry's _refresh method. This causes the Frame to execute the _refresh method on the widget stickerPrice.

Note: The _needRefresh method itself does not execute stickerPrice's _refresh method.

```
stickerPrice._needRefresh(); _frame_. _refresh();
```

See _refresh in this class for another example of using the _needRefresh method.

---

_/noWrite_

Internal method

---

_/Syntax_

objectName_noWrite();

---

_/objectLabel_

Runs the object label section in the frame's SCL program

---

_/Syntax_

objectName_objectLabel();

---

_/Details_

The _objectLabel method, which is normally used when an object is subclassed, is not called by SCL code. It is invoked automatically by the frame when the object is modified or selected. By default, the _objectLabel method runs the label associated with the object in the frame's SCL program.

This method can be overridden if preprocessing or postprocessing is needed for the object label section.

Example
To process information both before and after the object label in the frame's SCL program runs, you can override the _objectLabel method of the object and write your method similar to this:
Chapter 5

length _method_ $ 40; objlab: method;
/*----- perform preprocessing -----*/ /*------------------*/ /* some SCL statements here; */ /* some SCL statements here; */ /* some SCL statements here; */ /*---- run the object label section */ /* in the FRAME ---*/ _super._method(); /*----- perform postprocessing ---- */ /*------------------*/ /* more SCL statements here; */ /* more SCL statements here; */ /* more SCL statements here; */ /* endmethod;

___popup

Displays a pop-up menu containing character items in an SCL list

Syntax

objectName_popup( itemsList, selection );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemsList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of a list containing items to display in the pop-up</td>
</tr>
<tr>
<td>selection</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the index number of the selection made from the pop-up menu</td>
</tr>
</tbody>
</table>

Details

The _popup method is enabled when
- you select Use _popup method in the object’s Command Processing window
- you select _runPopup method in the General Attributes Keys, Pmenu, and Commands window
- you specify ‘popup’ in the _popups method (see _popups in this class)
- the object is a work area item (see the Work Area class)

When the _popup method is enabled, it is automatically invoked when the object receives a pop-up menu event. The list passed to the _popup method is allocated for you and is initially empty. As the _popup method is passed up the class hierarchy, each Super class can modify the list (such as adding or removing items or making items inactive).

If there is not an owning object, the _childPopup method is invoked on the Frame object. There are currently five owning objects for Frame entries:
- Composite Object
- Data Form
- Extended Table
- Organizational Chart
- Work Area
After the pop-up menu list displays, selection contains the selected pop-up menu item. If that selection belongs to this method, the selection should be handled and selection should be set to 0 before returning. The list should also be cleaned up (for example, removing items that were added, adding items that were removed, and making items active). It is important that the list and selection returned from this method be consistent with the list that was passed into the method as illustrated in the example.

Example

This example adds three items to the beginning of the list and then handles the selection of the item:

```c
length _method_ $40; popup: method list sel 8; /* Insert three items on the front of */ /* the pop-up menu list. */ rc = insertc(list, "Edit", 1); rc = insertc(list, "Copy", 2); rc = insertc(list, "Remove", 3); /* CALL SUPER invokes the current */ /* method, _popup, and displays */ /* the list. */ rc = _super_.method('list', 'sel'); /* Delete the three items from the */ /* list so they are not passed to */ /* child subclasses. */ rc = delitem(list, 1); rc = delitem(list, 1); rc = delitem(list, 1); /* SEL was set by the CALL SUPER; */ /* check if the selection is one */ /* of the items that was added. */ if (sel > 0) then do; /* Since the three added items */ /* were added at the beginning */ /* of the list, SEL must be 1, */ /* 2, or 3 if one of the items */ /* was selected. */ if (sel <= 3) then do; if (sel = 1) then _self_.attributesDialog(); else if (sel = 2) then _self_.copyRegion(); else /* sel must be 3 */ _self_.term(); end; /* The selection was not one of */ /* the items that was added; */ /* however, since the three items */ /* were added at the beginning, */ /* subtract 3 from the */ /* selection. */ else sel = sel - 3; end; endmethod;
```

In this example selection is checked for greater than 0 before any work is done. This is to allow a Super class to handle the selection first and have it set selection to 0 if it handled the selection (selection is set to 0 in this example when the selection is handled).

Note that 3 is subtracted from selection if the selection was not one of the items that was added. Therefore, no subclasses are affected when a Super class adds items to the list. For example, if a class added three items to the beginning of the list, the expected value of selection would be 1, 2, or 3 when one of those items is selected. However, if a Super class also added three items to the beginning of the list, that would have the effect of pushing the three previous items further down the list, thus making a selection of item 4, 5, or 6.

__popups__

Changes the pop-up menu processing option

Syntax

```c
objectName_popups( option );
```
### Argument Type Use Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Character</td>
<td>Input</td>
<td>specifies the pop-up menu processing option: GATTR (honor general attribute), OBJECT (run object label), OFF (ignore it), POPUP (use _popup method)</td>
</tr>
</tbody>
</table>

**Details**

Popmenu processing is normally specified in the Command Processing window in the object attributes. The four option values correspond to the four radio buttons in the Command Processing window.

The _popup method is used by the work area object to force items added to the work area to use the _popup method instead of the _select method (the default). This is done because the popup interface provided by the work area object is the primary method of manipulating work area items as well as removing those items.

**Example**

Allow all objects in the frame to run the _popup method:

```
_init: list = makelist();
_frame._getWidgets(list); do i = 1 to listlen(list); object = popl(list);
object._popup('popup'); end; return;
```

---

**_postinit**

**Performs additional processing on all widgets after the _initLabel method of the frame runs**

**Syntax**

```
objectName_postinit( );
```

**Details**

Under normal circumstances, you do not call the _postinit method directly. Instead, it is intended to be overridden by a subclass and called automatically. Override the _postinit method if you want a Widget Class to perform additional processing after the INIT section of the Frame entry's associated SCL code has executed. This is useful if the Widget Class requires data that can be computed only after all other widgets in the frame have initialized or after the INIT section defines the data.

For example, if a radio box is populated from a data set ID, the data set must be open in the INIT section of the frame. Then the _postinit method of the radio box populates the radio box from that opened data set ID.

If a _refresh method is invoked on an object before the object's _postinit method has run, the _refresh is changed to a _postinit method. Conversely, if a _postinit method is invoked on an object whose _postinit method has already run, _postinit is changed to _refresh.

**Example**
The following example shows a _postinit method for a subclass of the ListBox class; assume the method has been overridden in the class editor and refers to the SCL code here. This _postinit method populates the list box with the contents of a SList entry, which is assumed to contain a list of strings. The name of the SList entry is stored in the widget's character attribute, SList. By performing this operation in the _postinit method instead of the _init method, the INIT program of a Frame entry can change the SList instance variable or change the contents of the SList entry.

INIT:
method; /* empty the list box */ _self_._deleteAll(); ename = getitemc(_self_, ‘SList’); items = makelist(); rc = fillist(‘catalog’, ename, items); if (rc = 0) then do i = 1 to listlen(items); item = getitemc(items, i); /* add the item to the end */ /* of list box */ _self_, '_add_', item, -1); end; rc = dellist(items); endmethod;

___preterm

Performs additional processing before the _termLabel method of the frame runs

Syntax

objectName_preterm( );

Details

Under normal circumstances, you do not need to call the _preterm method directly; it is called automatically for each widget in a frame immediately before executing the TERM label in the Frame entry's SCL code. Typically, the method executes when users issue an END or CANCEL command. For example, if a user issues the END command, leaving a text entry field blank (and the field has the required attribute set), the _preterm method turns on the error attribute for the blank field, preventing the TERM section from executing while the field is in error.

Note The _preterm method does not execute if the Frame entry performs a halt and _status is set to ‘H’. Δ

Although this method is executed automatically, you may need to override its default processing, for example, if you create a subclass that requires certain additional conditions to be true before terminating the program. In this case, you should override the _preterm method and set _errorOn if the condition exists, or use the _setStatus method for the Frame class to set the status to ‘R’.

Example

PRETERM: method; if (some condition is true) then _self_._errorOn(); endmethod.
_printDialog

Syntax

objectName_printDialog( );

_protect

Protects a widget without updating its data

Syntax

objectName_protect( );

Details

- The results of the _protect method vary by the object (for example, in some cases, _protect may not allow selection and in other cases, it also may not allow tabbing). The _protect method is one of two mutually exclusive methods that disable a widget so users cannot modify or select the widget. The second method is described in _gray for this class. You cannot use the _gray method to gray a widget that is already protected with the _protect method, and vice versa.

- Use the _protect method rather than the _gray method to protect a widget that is never selectable. For example, you may protect a SAS/GRAPH Output widget that acts as the background for a frame. This prevents the _select method from running when users select the widget. Icons, graphic text, and widgets used for titles or other decoration should be protected with the _protect method as well.

- You may also use the field or protect functions to protect a widget; however, the _protect method is the only way to protect a widget from another SCL entry such as a method.

Example

_protect protects a widget named libname.

libname._protect();

_refresh

Redraws a widget without updating its data
Syntax

objectName._refresh();

Details

The _refresh method is executed automatically every time the frame is refreshed. The frame is refreshed when users enter information, the frame receives the _refresh method, or the frame executes an SCL refresh statement.

When the _refresh method executes, the Widget Classes supplied with SAS/AF software perform widget-specific operations such as copying data to the window or running the GETROW section of the SCL program.

A widget receives the _refresh method if one of the following conditions is true:

- the widget is assigned a value in the associated SCL program
- the widget's value is changed through a method (for example, _setText or _setColorStr)
- the widget is in error (that is, the _inError method returns a value of 1)
- the widget has been marked as needing a refresh (for example, using the _needRefresh method)
- a text entry object has been modified.

If the _refresh method is invoked on an object before the object’s _postinit method has run, the _refresh is changed to a _postinit method. Conversely, if a _postinit method is invoked on an object whose _postinit method has already run, _postinit is changed to _refresh.

Example

In the following example, a subclass of the text label field overrides the _refresh method to update itself with the current time. The label updates its text with the formatted current time and then invokes the super _refresh method, which actually puts the new text label on the window. Before returning, this method calls its _needRefresh method so that it is refreshed the next time the Frame entry is refreshed.

REFRESH: method;
_self._setText.putn(time(),time()); _super._refresh();
_self._needRefresh(); endmethod;

_regionAttrDialog

Displays the region attributes window for an object

Syntax

objectName._regionAttrDialog();
_removeDragDropSite

Destroys a drag-and-drop site without removing the object

Syntax

\texttt{objectName\_removeDragDropSite( );}

/resetCursorShape

Resets the cursor shape (mouse pointer shape) of a widget

Syntax

\texttt{objectName\_resetCursorShape( );}

Details

By default, the widget's cursor shape is the cursor shape for the frame.

_resize

Invoked when the widget is resized

Syntax

\texttt{objectName\_resize( );}

Details

The _resize method is invoked automatically when the _resize method is enabled (see _enableResizeNotify) and the widget is resized. A widget can be resized through methods (_resizeRegion or _moveRegion) or attachments.
To enable or disable the _resize method, use the _enableResizeNotify and _disableResizeNotify methods, respectively.

**CAUTION:**

_Avoid infinite loop with _resize_ If the _resize method calls a method that causes you to resize again (for example, _resizeRegion_), the _resize method may be called again. The result may be an infinite loop. To avoid this problem, use the _disableResizeNotify method before you call the method that causes you to resize again and use _enableResizeNotify after you have called that method._

Example

Suppose you have a SAS/GRAPH Output object and you want to play the Grseg entry associated with the object (via Proc Greplay) and retain the size of the object in the played graph. You can query the region size in inches and use these values to set the graphic options HSize and VSize. Whenever the graph object is resized, you need to reset the HSize and VSize options.

```plaintext
length hsize vsize 8; resize: method;
_self_.getObjectSize(hsize, vsize, inches); submit continue; goptions hsize=&hsize vsize=&vsize; endsubmit; endmethod;
```

---

**_resizeRegion_**

Resizes an object

_____

**Syntax**

```plaintext
objectName_resizeRegion( );
objectName_resizeRegion( width );
objectName_resizeRegion( width, height );
objectName_resizeRegion( width, height, units );
objectName_resizeRegion( width, height, units, rc );
```

**Details**

Specific values can be specified for width and height, or missing values can be used either to leave the size unchanged or to allow interactive resizing.

If a standard missing value is specified (_blank or .) for width or height, the width or height is not changed, respectively. For example, to change the height of obj1 to 3 inches, specify the desired value for height and a missing value for width:

```plaintext
obj1._resizeRegion(. ,3, 'i');
```

Special missing values can be specified for width and height to allow interactive resizing. See the example. The width and height for the new region are not necessarily
the width and height for the object. For example, specifying that the width of a push button is 10 actually causes the push button to have a width of 8 or so, depending on the host. The push button's region has a width of 10.

<table>
<thead>
<tr>
<th>Width</th>
<th>Height</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>change both the width and height to 5</td>
</tr>
<tr>
<td>5</td>
<td>.</td>
<td>change the width to 5</td>
</tr>
<tr>
<td>.</td>
<td>5</td>
<td>change the height to 5</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>nothing changes</td>
</tr>
<tr>
<td>5</td>
<td>.T</td>
<td>change the width to 5 and allow the bottom edge to be interactively changed</td>
</tr>
<tr>
<td>5</td>
<td>.B</td>
<td>change the width to 5 and allow the top edge to be interactively changed</td>
</tr>
<tr>
<td>.L</td>
<td>5</td>
<td>change the height to 5 and allow the right edge to be interactively changed</td>
</tr>
<tr>
<td>.R</td>
<td>5</td>
<td>change the height to 5 and allow the left edge to be interactively changed</td>
</tr>
<tr>
<td>.L</td>
<td>.T</td>
<td>allow the lower right corner to be interactively changed</td>
</tr>
<tr>
<td>.L</td>
<td>.B</td>
<td>allow the upper right corner to be interactively changed</td>
</tr>
<tr>
<td>.R</td>
<td>.T</td>
<td>allow the lower left corner to be interactively changed</td>
</tr>
<tr>
<td>.R</td>
<td>.B</td>
<td>allow the upper left corner to be interactively changed</td>
</tr>
</tbody>
</table>

Example
Change the width of obj1 to 2 centimeters and allow the bottom edge (that is, anchor the top edge) to be interactively resized:

```javascript
obj1._resizeRegion( 2, .T, 'cm');
```

Allow the lower right corner (that is, anchor the upper left corner) to be resized interactively:

```javascript
obj1._resizeRegion( .L, .T);
```

**_respondToDragOff_**

Allows a drop site object to reverse the effects of _respondToDragOnto_

**Syntax**

```javascript
objectName_respondToDragOff( );
```
objectName_respondToDragOff(objectID);

Details
The respondToDragOff method is called when a drag site object is no longer over the drop site object.

__respondToDragOnto

Changes the drop site object's visual appearance to indicate a drop is allowed

Syntax
objectName_respondToDragOnto();
objectName_respondToDragOnto(objectID);

Details
The respondToDragOnto method is called when a drag site object is dragged over a valid drop site object. A visual change in the object is recommended to show the user that a drop is possible.

__restore

Restores the state of an object after HELPMode

Syntax
objectName_restore();

Details
The _restore method is invoked automatically after an object receives a _help method due to the HELPMode command. Some objects change state or appearance when selected, even if they are selected while in the help environment. For example, selecting a list box while in the help environment highlights the row that was selected. The _restore method restores the previous state of the list box.
_row

Returns the number of the row in which a widget begins

**Syntax**

```plaintext
objectName_row( row );
```

**Argument Type Use Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the top-most row occupied by the widget</td>
</tr>
</tbody>
</table>

**Details**

**Example**

The following example determines a widget's starting row and column position in a frame and then displays a pop-up menu in the same position:

```plaintext
button._row(‘row’); button._column(‘col’); choice = popmenu(buttonMenu, 0, row, col);
```

_scrollbarsOff

Hides the scroll bars of an object

**Syntax**

```plaintext
objectName_scrollbarsOff( );
objectName_scrollbarsOff( which );
```

**Details**

These objects support scroll bars:

<table>
<thead>
<tr>
<th>Object</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catalog Entry Viewer</td>
<td>Organizational Chart</td>
</tr>
<tr>
<td>Data Form</td>
<td>Process Flow Diagram</td>
</tr>
<tr>
<td>Data Table</td>
<td>Report Viewer</td>
</tr>
<tr>
<td>Extended Table</td>
<td>SAS/GRAPH Output</td>
</tr>
</tbody>
</table>

This method has no effect on other objects.

**_scrollbarsOn**

Unhides the scroll bars of an object

**Syntax**

```
objectName_scrollbarsOn( );
objectName_scrollbarsOn( which );
```

**Details**

For a list of objects that support scroll bars, see _scrollbarsOff. This method has no effect on other objects.

**_select**

Responds to a user selection or modification

**Syntax**

```
objectName_select( );
```

**Details**

A widget's _select method executes automatically any time users select or modify the widget. The default _select method executes the SAS command associated with the widget. If no command has been assigned to the widget, the default _select method does nothing.

If a labeled section of code in the SCL program is associated with the widget, when the user selects or modifies the widget, the labeled section executes before the _select method is invoked. However, calling the _select method does not execute the labeled section.

If the value of _EVENT_ is 'C', the _select method does not execute the associated command.

To write custom Widget Classes that perform more sophisticated processing when selected, you can override either the _select method or the _objectLabel method. The
_objectLabel method has more flexibility, so in general that method should be overridden. See the _objectLabel method for more information.

### _setBackgroundColor

Assigns the background color of an object

**Syntax**

```
objectName_setBackgroundColor( color );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Character</td>
<td>Input</td>
<td>specifies the background color of the widget</td>
</tr>
</tbody>
</table>

**Details**

Specifying inherit sets the background color to the same color as the parent region.

### _setBorderColor

Assigns a color to a region’s border

**Syntax**

```
objectName_setBorderColor( color );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Character</td>
<td>Input</td>
<td>specifies the color of the border</td>
</tr>
</tbody>
</table>

### _setBorderState

Sets the border state of an object
Syntax

objectId._setBorderStyle( state );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>Character</td>
<td>Input</td>
<td>specifies a value indicating the appearance of the border: ON (pushed in), OFF (popped out) or TOGGLE (reverse the current state)</td>
</tr>
</tbody>
</table>

Details

A value of On means the border has a pushed-in appearance, assuming the region attribute Light Source is set to upper left.

This method is valid only if the object has a region attribute Outline Type of Button and the Button Behavior is not Checkbox or Radio Button.

_setBorderStyle

Sets the border style of an object

Syntax

objectId._setBorderStyle( style );
objectId._setBorderStyle( style, width );
objectId._setBorderStyle( style, width, lightSource );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>style</td>
<td>Character</td>
<td>Input</td>
<td>specifies the border style: SIMPLE, SHADOW, BUTTON, EMBOSS, or NONE</td>
</tr>
<tr>
<td>width</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the width of the border</td>
</tr>
<tr>
<td>lightSource</td>
<td>Character</td>
<td>Input</td>
<td>specifies a light source: UPPER LEFT, UPPER RIGHT, LOWER LEFT, or LOWER RIGHT</td>
</tr>
</tbody>
</table>

Details

The light source is honored only if the shadow style is shadow, button, or emboss.

When values for style, width, or lightSource are missing, those attributes are not changed.
Example

obj1 is a container box. Clicking on the container cycles through different light sources and sets the border title to the current light source. The INIT section simply initializes CNT and does not have a return statement so the_setBorderStyle in MAIN runs:

array
light[0:3] $11 {"upper left", "upper right", "lower right", "lower left" ];
init: cnt=0; main: obj1._setBorderStyle(‘shadow’,5,light[cnt]);
obj1._setBorderTitle(light[cnt]); cnt = mod(cnt+1,4); return;

_setBorderStyle

Assigns a title to a region’s border

Syntax

objectName_setBorderTitle( title );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>Character</td>
<td>Input</td>
<td>specifies the text of the border title</td>
</tr>
</tbody>
</table>

Details

Example

_setBorderStyle assigns Financial Report as obj1’s border title:

obj1._setBorderTitle(‘Financial Report’);

_setBorderTitleColor

Sets the color of an object’s border color

Syntax

objectName_setBorderTitleColor( color );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Character</td>
<td>Input</td>
<td>specifies the color of the border</td>
</tr>
</tbody>
</table>
**_setCMD**

Assigns the command to execute when a widget is selected

----------

**Syntax**

```
objectName_setCMD( command );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>Character</td>
<td>Input</td>
<td>specifies the SAS command to execute</td>
</tr>
</tbody>
</table>

**Details**

Example

The following example changes the SAS command associated with an icon named buildIcon. The variable catalog contains the name of a SAS catalog.

```
buildIcon._setCMD(); build || catalog || msg=no;
```

**_setCursorShape**

Assigns the cursor shape (mouse pointer shape) for a widget

----------

**Syntax**

```
objectName_setCursorShape( shape );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shape</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies a shape number</td>
</tr>
</tbody>
</table>

**Details**

When you assign a cursor shape to a widget, the cursor changes from the default shape (normally an arrow, #2) to the specified shape when the mouse pointer moves across the widget. Some hosts do not allow you to invoke the _cursorTracker method while the mouse pointer is moving across native widgets (for example, list boxes, push buttons, and icons).
### Table 5.1  Cursor Shapes

<table>
<thead>
<tr>
<th>Number</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>text</td>
</tr>
<tr>
<td>1</td>
<td>wait</td>
</tr>
<tr>
<td>2</td>
<td>select (default)</td>
</tr>
<tr>
<td>3</td>
<td>crosshair</td>
</tr>
<tr>
<td>4</td>
<td>focus</td>
</tr>
<tr>
<td>5</td>
<td>move</td>
</tr>
<tr>
<td>6</td>
<td>copy</td>
</tr>
<tr>
<td>7</td>
<td>question mark</td>
</tr>
<tr>
<td>8</td>
<td>focus/zoom in</td>
</tr>
<tr>
<td>9</td>
<td>focus/zoom out</td>
</tr>
<tr>
<td>10</td>
<td>left and right edge resize</td>
</tr>
<tr>
<td>11</td>
<td>top and bottom edge resize</td>
</tr>
<tr>
<td>12</td>
<td>unconstricted region resize</td>
</tr>
<tr>
<td>13</td>
<td>top left/bottom right resize</td>
</tr>
<tr>
<td>14</td>
<td>top right/bottom left resize</td>
</tr>
<tr>
<td>15</td>
<td>action ok (allowed)</td>
</tr>
<tr>
<td>16</td>
<td>multi-object move</td>
</tr>
<tr>
<td>17</td>
<td>drop ok</td>
</tr>
<tr>
<td>18</td>
<td>action not allowed</td>
</tr>
<tr>
<td>19</td>
<td>help on a selected item</td>
</tr>
</tbody>
</table>

#### Example

Suppose you make a SAS/GRAPH Output object, `graph`, and you want the cursor shape to be a crosshair when the mouse pointer is moved across the graph. Specify

```plaintext
graph._setCursorShape(3);
```

---

**_setDragOp**

Sets the type of operations that this object can handle as a drag request

**Syntax**

```plaintext
return = objectName_setDragOp( );
```
Details
When data are dragged from place to place, the data can be copied or moved or some linkage can be specified. Copy implies that the data are provided with no post-processing. Move implies that the data are provided and the source is removed. Link implies that some mechanism is used to keep the source and destination in sync. Programmers must provide the linkage. The type of operation is matched with the drop operation of a drop site object to determine if the drop is allowed. All previous drag operations are removed from this object. This is the preferred drag operation until another _setDragOp method or an _addDragOp method is called. If you specify more than one operation, the first is considered the default. Some hosts allow you to choose which operation to use at drag or drop time.

_setDragRep
Registers the representation as a valid drag representation

Syntax
return = objectName_setDragRep( );

_setDropAction
Sets the type of actions this object can handle as a drop request

Syntax
return = objectName_setDropAction( );

Details
The action-op pair is matched with the drag operation of a drag site object to determine if the drop is allowed. All previous drop actions are removed from this object. Some hosts provide special drag requests that display the list of actions when the object is dropped. The action selected is passed to the _validateDropData and _drop methods. Copy, move, and link cannot be removed.

_setDropOp
Sets the type of operations this object can handle as a drop request
Syntax
return = objectName_setDropOp( );

Details
When data are dragged from place to place, the data can be copied or moved or some linkage can be specified. Copy implies that the data are provided with no post-processing. Move implies that the data are provided and the source is removed. Link implies that some mechanism is to be used to keep the source and destination in sync. The programmers must provide the linkage. The _setDropOp method matches the op variable value with the drag operation of a drag site object to determine if the drop is valid. All previous drop operations are removed from this object. This is the preferred drop representation until another _setDropOp method or an _addDropOp method is called. If this method is not called, the default operation is copy.

_setDropRep

Registers a representation as valid for dropping

Syntax
return = objectName_setDropRep( );

Details
A drag site is defined the first time a representation is defined for the object. During the drag, the list of representations is matched with the drop sites. The drop is allowed when there is at least one representation match. The current _setDropRep method specifies the preferred drop representation until another _setDropRep method or an _addDropRep method is called. All previous drop representations are removed from this object.

_setHelpMode

Sets the help environment

Syntax
objectName_setHelpMode( );
Details

This method runs automatically when the help environment is enabled. The default behavior is to unprotect objects that are protected so that you can click them with the help mouse pointer.

Example

Suppose you want to change the region title of your object when the help environment is enabled. Use the _setHelpMode method to change the title and the _clearHelpMode method to restore it.

```plaintext
length _method title $40; sethelp: method;
/* get the current region title and */ /* save it */ reg = makelist();
_self_. _getRegion(reg); title = getnitemc(reg, 'title'); rc =
setnitemc(_self_, title, 'previous title'); rc = dellist(reg); /* set the
new title */ _self_. _setBorderTitle('select me for help');
_super_. _method(); endmethod; clrhelp: method; /**/ /* get the previous
title and restore */ /* it */ title = getnitemc(_self_, 'previous title');
_self_. _setBorderTitle(title); rc = delnitem(_self_, 'previous title');
_super_. _method(); endmethod;
```

_setOutline

Sets the border style of an object

Syntax

```plaintext
objectName_setOutline( style );
objectName_setOutline( style, width );
objectName_setOutline( style, width, lightSource );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>style</td>
<td>Character</td>
<td>Input</td>
<td>specifies the border style: SIMPLE, SHADOW, BUTTON, EMOSS, NONE</td>
</tr>
<tr>
<td>width</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the width of the border</td>
</tr>
<tr>
<td>lightSource</td>
<td>Character</td>
<td>Input</td>
<td></td>
</tr>
</tbody>
</table>
_setRegion

Sets multiple region attributes

Syntax

objectName._setRegion( regionList );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regionList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of a list containing named items of region attributes to change</td>
</tr>
</tbody>
</table>

Details

The format of the list is described in the _new method in the Class class.

_setRegionColor

Sets the widget's background color

Syntax

objectName._setRegionColor( color );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Character</td>
<td>Input</td>
<td>specifies the color used for the widget's background</td>
</tr>
</tbody>
</table>

_setSelection

Sets the selection status of a widget. Used by the Work Area and the Extended Table objects.

Syntax

objectName._setSelection( status, color );
### Details

The `setSelection` method is used by the work area object and the extended table object. When you select an object in a work area, the object is sent a `setSelection` method with a status of ON. The object that was previously selected in the work area is sent a `setSelection` with a status of OFF. Both the selected object and the previously selected object have their border colors changed to indicate their selection status. To achieve a different visual appearance, you could override the `setSelection` method and change the object's background color or border title.

For extended table objects, the `setSelection` method is automatically called from the GETROW section of the SCL program to update the selection status of a widget. By default, when a row is selected, the border color is changed to the selection color. When a row is unselected, the region's original border color is restored. For example, suppose you have an extended table with a logical row that consists of obj1 and obj2. The table can display four rows. If the user selects row 2, the `selectRow` method and the PUTROW block will run for row 2. When the Frame entry is redisplayed, the following processing takes place:

1. The extended table invokes the `setSelection` method for obj1 on row 1 with the OFF parameter and the original color.
2. The extended table invokes the `setSelection` method for obj2 on row 1 with the OFF parameter and the original color.
3. The `getRowLabel` method runs for row 1.
4. The `getRow` method runs for row 1.
5. The extended table invokes the `setSelection` method for obj1 on row 2 with the ON parameter and the selection color.
6. The extended table invokes the `setSelection` method for obj2 on row 2 with the ON parameter and the selection color.
7. The `getRowLabel` method runs for row 2.
8. The `getRow` method runs for row 2.
9. The same steps are repeated for rows 3 and 4.

#### Example

The following example is an overridden `setSelection` method. Instead of changing the default behavior of the region's border color, this example changes the region's title.

```plaintext
sel: method status color $;
if(status='ON') then _self_.setBorderTitle(selected); else _self_.setBorderTitle(unselected); endmethod;
```

---

### _setTabbable

Enables a widget to receive the _tab method
Syntax

objectName_setTabbable( status );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Character</td>
<td>Input</td>
<td>specifies a value indicating whether the widget accepts the _tab method:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ON accepts _tab, OFF does not accept _tab</td>
</tr>
</tbody>
</table>

Details

Use _setTabbable if you want to tab to objects that do not normally receive the _tab method, such as Graphic Text or SAS/GRAPH Output objects.

Some hosts perform their own tabbing, so making a widget tabbable may have no effect.

ON sends a _tab method to the widget when the user hits the TAB key; OFF does not send a _tab method to the widget when the user hits the TAB key.

Example

_setTabbable allows the Graphic Text object, GText, to receive the _tab method.

gtext._setTabbable('ON');

_setText

Sets the character value of an object

Syntax

objectName_setText( value );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Character</td>
<td>Input</td>
<td>specifies the value of the object (for legacy class support)</td>
</tr>
</tbody>
</table>

Details

If the object represents a numeric value, the method fails. This method is mainly used within other methods because the value of an object is not directly accessible. The value of an object is available in the frame's SCL program by using the name of the object.

Example

The following two lines of code perform the same function:
obj1='hello';
obj1._setText('hello');

_setTitleStyle

Sets the title style of an object

Syntax

objectName.setTitleStyle( justification );
objectName.setTitleStyle( justification, verticalOffset );
objectName.setTitleStyle( justification, verticalOffset, font );
objectName.setTitleStyle( justification, verticalOffset,font, fontScaling );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>justification</td>
<td>Character</td>
<td>Input</td>
<td>specifies the justification: L (left), R (right), C (center) or N (none)</td>
</tr>
<tr>
<td>verticalOffset</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies vertical offset (in pixels) of the title</td>
</tr>
<tr>
<td>font</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the identifier of an SCL list containing font attributes</td>
</tr>
<tr>
<td>fontScaling</td>
<td>Character</td>
<td>Input</td>
<td>specifies whether to perform font scaling: YES or NO</td>
</tr>
</tbody>
</table>

Details

The font list can be created using the fontSel SCL function.

Example

Set the title on obj1 to 'hello there' and make it right justified and raised 9 pixels above the region outline:

obj1._setBorderTitle('hello there');
obj1._setTitleStyle('R', 9);

_setValue

Sets the numeric value of an object
### Syntax

```c
objectName ._setValue( value );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the value of the object (for legacy class support)</td>
</tr>
</tbody>
</table>

### Details

If the object represents a character value, the method fails. This method is used mainly within other methods because the value of an object is not directly accessible. The value of an object is available in the frame's SCL program by using the name of the object.

**Example**
The following two lines of code perform the same function:

```c
obj1 = 28; obj1._setValue(28);
```

---

### _snapShot

Copies a snapshot of a graphical widget into the Image Data Object

### Syntax

```c
objectName ._snapShot( objectID );
objectName ._snapShot( objectID, successful );
objectName ._snapShot( objectID, successful, snapType );
objectName ._snapShot( objectID, successful, snapType, upperLeftX );
objectName ._snapShot( objectID, successful, snapType, upperLeftX, upperRightY );
objectName ._snapShot( objectID, successful, snapType, upperLeftX, upperRightY, width );
objectName ._snapShot( objectID, successful, snapType, upperLeftX, upperRightY, width, height );
```
### Argument Types and Uses

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectID</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the object identifier of the Image Data Object that is the target of the snapshot operation</td>
</tr>
<tr>
<td>successful</td>
<td>Numeric</td>
<td>Output</td>
<td>returns whether or not the _snapshot method was successfully completed: 0 if successful</td>
</tr>
<tr>
<td>snapType</td>
<td>Character</td>
<td>Input</td>
<td>specifies the type of snapshot operation: BOUNDED (default), UNBOUNDED or MAIN</td>
</tr>
<tr>
<td>upperLeftX</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the the upper left x pixel coordinate for explicit bounding specification</td>
</tr>
<tr>
<td>upperRightY</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the the upper right y pixel coordinate for explicit bounding specification</td>
</tr>
<tr>
<td>width</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the the width in pixels for explicit bounding specification</td>
</tr>
<tr>
<td>height</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the the height in pixels for explicit bounding specification</td>
</tr>
</tbody>
</table>

### Details

An Image Data Object, which contains the results of the snapshot operation, must be provided. Not all objects will have a useful _snapshot image placed in the Image Data Object. Host widgets (such as push buttons and list boxes) are notable examples.

**Example**

The following example has a SAS/GRAPH Output object (**Original Graph**) on the left half of the display and two Image objects (**Scaled** and **Unscaled**) on the right half. The first Image object (**Scaled**) has scaling enabled and the second Image object (**Unscaled**) has scaling disabled. A radio box (snapType) contains the different snap_types. There is also a Graphic Text object nested in the SAS/GRAPH Output object that contains the string ‘Texas’. When the radio box is selected, a new snapshot is created.

The following SCL code is used:

```scl
init: scaled._getDataId(scaledId);
unscaled._getDataId(unscaledId); return; snatype:
graph._snapshot(scaledId, rc, snatype); scaled._update();
graph._snapshot(unscaledId, rc, snatype); unscaled._update(); return;
```

Before selecting a snatype, **Original Graph** is scrolled down and to the right so that parts of Montana and Idaho are in the upper left. Selecting **BOUNDED** from the radio box produces a snapshot.

Notice the Image object **Scaled** contains the entire picture visible in GRAPH, whereas **Unscaled** only contains the upper left portion. Both images contain parts of Montana and Idaho in the upper left.

When **Unbounded** is selected, **Scaled** contains the entire Grseg entry that GRAPH displays, whereas **Unscaled** contains only the upper left portion. **Unbounded** attempts to take a snapshot of the entire object, but only the upper left fits in **Unscaled**.
When `Main` is selected, you get results similar to `Bounded` except that nested objects are included in the snapshot. Notice the string 'Texas' appears in the snapshot images, whereas the string does not appear in the previous `Bounded` snapshot.

### startDrag

With `completeDrag`, performs possible visual changes to an object selected for dragging

**Syntax**

```plaintext
objectName_startDrag( xLocation, yLocation );
objectName_startDrag( objectID );
```

**Argument**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xLocation</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the X location of the drag in pixels</td>
</tr>
<tr>
<td>yLocation</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the Y location of the drag in pixels</td>
</tr>
<tr>
<td>objectID</td>
<td>(1)</td>
<td>Input</td>
<td>specifies the identifier of an object containing drag and drop data</td>
</tr>
</tbody>
</table>

(1) `objectID` is `sashelp.classes.draganddrop.class`

**Details**

The `startDrag` method is called when an object is selected for dragging. Use this method in conjunction with the `completeDrag` method to perform possible visual changes.

### swapIn

Swaps in the object so it is visible

**Syntax**

```plaintext
objectName_swapIn( );
```
**_swapOut_**

Swaps out the object so it is hidden

---

**Syntax**

`objectName_swapOut( );`

**Details**

Since textual objects (for example, push button, icons, text entry objects) cannot overlap, you can use _swapOut to temporarily remove an object so that another textual object can occupy that space. If you use the _hide method instead, the object would only be hidden. It would still occupy the location on the screen, so another textual object could not be placed in that location.

- If you attempt to swap in a textual object in the same location as another textual object, the method fails.
- See _isSwapped and _hide for more information.

---

**_tab_**

Indicates whether a widget was tabbed to

---

**Syntax**

`objectName_tab( direction, inOut );`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>direction</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies a value indicating the direction of tabbing: 1 tabs forward, -1 tabs backward</td>
</tr>
<tr>
<td>inOut</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating where the cursor has been moved: 1 tabbed to the widget, 0 tabbed out of the widget</td>
</tr>
</tbody>
</table>

---

**_tabIn_**

Invoked when object is made active
**Syntax**

```
objectName_tabIn();
```

**Details**

The _tabIn method, which is normally used when an object is subclassed, is not called by SCL code. It is invoked automatically by the frame when the object is made active. An object is made active when it is selected or tabbed to or when the _cursor method has been invoked on the object.

This method can be used to control the appearance of the active object in a frame.

**Example**

Change the border color and title when the object is made active:

```scl
length _method color $ 40; tabin: method;
_super_.method(); /* save original border color and title */ /* (used in _tabOut) */ _self_.getBorderColor(color); rc = setnitemc(_self_, color, 'original color'); list = makelist(); _self_.getRegion(list); rc = setnitemc(_self_, getnitemc(list, 'title), 'original title'); rc = dellist(list, 'y'); _self_.setBorderTitle("I'm active"); _self_.setBorderColor(green); endmethod;
```

**_tabOut**

**Invoked when object is made inactive**

**Syntax**

```
objectName_tabOut();
```

**Details**

The _tabOut method, which is normally used when an object is subclassed, is not called by SCL code; it is invoked automatically by the frame when the object is made inactive. An object is made inactive when a new object is made active.

Use this method to control the appearance of the inactive objects in a frame.

**Example**

Restore the border color and title when the object is made inactive:

```scl
length _method color title $ 40; tabout: method; _super_.method(); /* get original border color and title */ /* (set in _tabIn) */ color = getnitemc(_self_, 'original color'); title = getnitemc(_self_, 'original title'); rc = dellitem(_self_, 'original color'); rc = dellitem(_self_, 'original title'); _self_.setBorderColor(green); endmethod;
```
_term

Deletes an object and if the object contains other objects, it optionally deletes the nested objects as well.

Syntax

objectName._term(recursive);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recursive</td>
<td>Character</td>
<td>Input</td>
<td>specifies whether to recursively delete all nested objects when an object contains other objects: Y or N (default)</td>
</tr>
</tbody>
</table>

Details

If an object contains other objects, the recursive argument specifies whether the nested objects should be deleted. For example, if a container object named containerbox1 contained a text entry control and a check box control, invoking

```java
containerbox1._term('Y');
```

recursively deletes the container and the nested objects. In other words, the nested objects also receive the _term method.

_ungray

Activates (ungrays) a grayed widget

Syntax

objectName._ungray();

Details

The field function can also ungray a widget, but the _ungray method is the only way to ungray a widget from a different entry.
Example
_ungray ungrays a widget named buildIcon if the variable catalog is not in error:
   if
   field('error', 'catalog') = 0 then buildIcon._ungray();

__unhide

Displays a widget that has been hidden

Syntax

objectName_unhide( );
objectName_unhide( all );

Details

Example
_unhide displays the buildIcon object if the variable catalog is not in error:
   if
   field('error', 'catalog') = 0 then buildIcon._unhide();

__unprotect

Unprotects a widget, allowing user modification

Syntax

objectName_unprotect( );

Details

You can also use the field or unprotect function to unprotect a widget. However, the
_unprotect method is the only way to unprotect a widget from another SCL entry such as
a method.
Example
_unprotect unprotects a widget named libname.
   libname._unprotect();
_update

Updates a widget

Syntax

objectName._update();

Details

The _update method re-creates an object and updates its contents based on its current attributes, which may require rereading an external file, data set, catalog entry, or SCL variable.

The _update method differs from the _refresh method in the extent of its update. A _refresh method updates an object's display in a window. An _update method updates the object's contents, and possibly re-creates the object, before updating its display in the window.

For example, a SAS/GRAPH Output object does not attempt to reread the Grseg entry with the _refresh method. The Grseg entry is reread with the _update method. Suppose you have a SAS/GRAPH output object that displays Sasuser.Graphs.Sales.Grseg. If the Frame application regenerates the Grseg entry (for example, via a SUBMIT block), the SAS/GRAPH output object needs an _update method for the new Grseg entry to be displayed.

_updateValue

Internal method

Syntax

objectName._updateValue();

_validateAttrs

Internal method
**Syntax**

objectName_validateAttrs();

---

**_validateDropData**

Allows a drop site to validate a drag operation’s data

---

**Syntax**

objectName_validateDropData( representation, operation, dataList, dragStart, x, y, cancel );

objectName_validateDropData( objectID );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>representation</td>
<td>Character</td>
<td>Input</td>
<td>specifies the representation selected for the drop</td>
</tr>
<tr>
<td>operation</td>
<td>Character</td>
<td>Input</td>
<td>specifies the operation selected for the drop</td>
</tr>
<tr>
<td>dataList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of the SCL list that contains the data defined by the representation</td>
</tr>
<tr>
<td>dragStart</td>
<td>Character</td>
<td>Input</td>
<td>specifies where the drag started: INSIDE (inside the window containing the drop site) or OUTSIDE</td>
</tr>
<tr>
<td>x</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the x location of the drop in pixels</td>
</tr>
<tr>
<td>y</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the y location of the drop in pixels</td>
</tr>
<tr>
<td>cancel</td>
<td>Character</td>
<td>Output</td>
<td>returns a flag indicating whether to complete or cancel the drop: 'C' indicates cancel otherwise a blank indicates to continue</td>
</tr>
<tr>
<td>objectID</td>
<td>(1)</td>
<td>Input</td>
<td>specifies the identifier of an object containing drag and drop data</td>
</tr>
</tbody>
</table>

(1) objectID is sashelp.classes.draganddrop.class
Details

The _validateDropData method is called when a drag site object has been dropped on a
drop site object. The drop site is passed the data returned from the _getDragData
method. This method is called to give the drop site a chance to validate the data and
allows the drag operation to be canceled.

No new Frame entries or applications can be started from this method. The SCL
code for this method also cannot be debugged.

Note: The _validateDropData method is followed by the _drop method, so it is
advisable to test the representation and the action before responding to the drop
operation.

x and y are the pixel offsets of the drop within the drop zone. These values are valid
only for nonwidget drop zones.

Events

Events specified for the Widget Class class are described here. See
sashelp.fsp.Object.class for inherited events.

backgroundColor changed
  Occurs when the backgroundColor attribute is changed

borderColor changed
  Occurs when the borderColor attribute is changed

borderLightSource changed
  Occurs when the borderLightSource attribute is changed

borderStyle changed
  Occurs when the borderStyle attribute is changed

borderTitle changed
  Occurs when the borderTitle attribute is changed

borderTitleColor changed
  Occurs when the borderTitleColor attribute is changed

borderTitleFont changed
  Occurs when the borderTitleFont attribute is changed

borderTitleFontScaling changed
  Occurs when the borderTitleFontScaling attribute is changed

borderTitleJustification changed
  Occurs when the borderTitleJustification attribute is changed

borderTitleOffset changed
  Occurs when the borderTitleOffset attribute is changed

borderWidth changed
  Occurs when the borderWidth attribute is changed
commandOnClick changed
  Occurs when the commandOnClick attribute is changed
commandProcessing changed
  Occurs when the commandProcessing attribute is changed
cursorTracking changed
  Occurs when the cursorTracking attribute is changed
dragEnabled changed
  Occurs when the dragEnabled attribute is changed
dragInfo changed
  Occurs when the dragInfo attribute is changed
dragOperations changed
  Occurs when the dragOperations attribute is changed
dropEnabled changed
  Occurs when the dropEnabled attribute is changed
dropInfo changed
  Occurs when the dropInfo attribute is changed
dropOperations changed
  Occurs when the dropOperations attribute is changed
enabled changed
  Occurs when the enabled attribute is changed
height changed
  Occurs when the height attribute is changed
helpText changed
  Occurs when the helpText attribute is changed
horizontalPosition changed
  Occurs when the horizontalPosition attribute is changed
mousePointer changed
  Occurs when the mousePointer attribute is changed
popMenuProcessing changed
  Occurs when the popMenuProcessing attribute is changed
regionOutlineBehavior changed
  Occurs when the regionOutlineBehavior attribute is changed
regionOutlineState changed
  Occurs when the regionOutlineState attribute is changed
sizingUnits changed
  Occurs when the sizingUnits attribute is changed
toolTipText changed
  Occurs when the toolTipText attribute is changed
verticalPosition changed
  Occurs when the verticalPosition attribute is changed
visible changed
  Occurs when the visible attribute is changed
width changed
  Occurs when the width attribute is changed

Event Handlers

Event handlers specified for the Widget Class class are described here. See sashelp.fsp.Object.class for inherited event handlers.

<table>
<thead>
<tr>
<th>Event</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.sizingUnits Changed</td>
<td>_onSizingUnitsChanged</td>
<td>Occurs according to class definition</td>
</tr>
</tbody>
</table>