Frame Class

Overview

SAS/AF software uses the Frame class as the foundation for GUI-based applications. The Frame class provides windowing capabilities to SAS/AF applications and serves as a container where you can add visual controls and non-visual components to create a user interface. The Frame class enables you to create windows and dialog boxes, add menu bars or banners, and add application help.

Frame objects (and all visual controls and non-visual components placed on them) are stored in FRAME entries. When a FRAME entry is displayed in either the development environment or the execution environment, an instance of the Frame class is created.

Parent:

sashelp.fsp.Object.class

Class:

sashelp.fsp.Frame.class

Using the Frame Class

You can use a frame whenever you need to build a graphical user interface for an application. The Frame class and any subclasses of it that you create provide the properties for the windows and dialogs in your applications. Frames are stored as FRAME entries in a catalog.

By default, the frame looks for an SCL entry with the same name as its FRAME entry. You can use a different SCL entry by setting the SCLEntry attribute to the desired item.

To specify a menu bar to appear with the frame, set the pmenuEntry attribute to a valid PMENU entry.
SCL Programs for Frames

FRAME entries are primarily controlled by SCL programs. You can use SCL programs to invoke methods for objects and perform such tasks as user validation, calculation of values based on user input, manipulation and responding to controls (or widgets) on a frame, and providing messages and help to users.

Note: An SCL program is not required for a frame.

The SCL programs for frame entries can contain labeled sections in addition to the standard INIT, MAIN, and TERM labels. You can use a label of the same name as an object, and the code in the label executes when an action is performed on the object or its value is changed. For example, if you add a push button control named okButton to your frame, your SCL code could include

```scl
okButton: /* your processing instructions here */ return;
```

Attributes

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

Dictionary

**automaticCompile**

Returns or sets a flag indicating whether the frame is automatically compiled prior to a SAVE, END or TESTAF command

Type: Character

Valid Values: Yes, No

**backgroundColor**

Returns or sets the background color of the frame

Type: Character

Valid Values: Blue, Red, Pink, Green, Cyan, Yellow, White, Orange, Black, Magenta, Gray, Brown, Sysback, Syssecb, Sysfore, Background, Secondary Background
**bannerAttribute**

Returns or sets the appearance of the frame's banner

Type: Character

Valid Values: None, Blinking, Highlight, Hirev, Reverse, Underline

---

**bannerColor**

Returns or sets the color of the frame's banner

Type: Character

Valid Values: Blue, Red, Pink, Green, Cyan, Yellow, White, Orange, Black, Magenta, Gray, Brown, Sysback, Syssecb, Sysfore, Banner

---

**bannerType**

Returns or sets the type of window banner for the frame

Type: Character

Valid Values: Command, Select, None

---

**borderAttribute**

Returns or sets the appearance of the frame's border

Type: Character

Valid Values: None, Blinking, Highlight, Hirev, Reverse, Underline
**borderColor**

Returns or sets the color of the frame's border

*Type:* Character

*Valid Values:* Blue, Red, Pink, Green, Cyan, Yellow, White, Orange, Black, Magenta, Gray, Brown, Sysback, Syssecb, Sysfore, Border, Secondary Border

---

**buildTime**

Returns whether the frame is being built or executed

*Type:* Character

*Valid Values:* Yes, No

---

**buildTimeMode**

Returns whether the frame is being edited or browsed at build time

*Type:* Character

*Valid Values:* Edit, Browse

---

**CBTFrameName**

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

*Type:* Character

*Valid Values:*
**commandAttribute**

Returns or sets the appearance of the frame’s command area

**Type:** Character

**Valid Values:** None, Blinking, Highlight, Hirev, Reverse, Underline

---

**commandColor**

Returns or sets the color of text that is displayed in the command area

**Type:** Character

**Valid Values:** Blue, Red, Pink, Green, Cyan, Yellow, White, Orange, Black, Magenta, Gray, Brown, Sysback, Syssecb, Sysfore, Command

---

**commandProcessing**

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

**Type:** Character

**Valid Values:** Ignore it, Run main, Run object label

---

**confirmOnExit**

Returns or sets a flag that controls whether a confirmation dialog appears when the frame is closed

**Type:** Character

**Valid Values:** Yes, No
cursorPlacement

Returns or sets the name of the control where the cursor is placed when the frame opens

Type: Character
Valid Values: \sashelp.classes.frameWidgetValues.scl

defaultPushButton

Returns or sets the push button control in the frame that is the default button. Using this attribute will disable command line functionality.

Type: Character
Valid Values: \sashelp.classes.defaultpushbuttonvalues.scl

description

Returns or sets the short description for the frame

Type: Character
Valid Values:

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

---

**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

forcePmenuOn

Returns or sets a flag indicating whether the menu bar is displayed at the top of the frame

Type: Character
Valid Values: Yes, No

foregroundAttribute

Returns or sets the appearance of text that is displayed in the frame’s foreground

Type: Character
Valid Values: None, Blinking, Highlight, Hirev, Reverse, Underline

foregroundColor

Returns or sets the default color of region borders

Type: Character
Valid Values: Blue, Red, Pink, Green, Cyan, Yellow, White, Orange, Black, Magenta, Gray, Brown, Sysback, Syssecb, Sysfore, Foreground, Info, Label
height

Returns or sets the height of the frame

Type: Numeric
Valid Values:

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

horizontalPosition

Returns or sets the upper left column coordinate that specifies the position of the frame

Type: Numeric
Valid Values:

icon

Returns or sets the icon of the frame

Type: Numeric
Valid Values:
Editor: sashelp.classes.iconeditor.scl
**keysEntry**

Returns or sets the name of the KEYS catalog entry containing the function key assignments defined for the frame

Type: Character

Valid Values: 

Editor: SASHELP.FSP.KEYEDITOR.SCL

---

**messageAttribute**

Returns or sets the appearance of the frame’s message area

Type: Character

Valid Values: None, Blinking, Highlight, Hires, Reverse, Underline

---

**messageColor**

Returns or sets the color of text that is displayed in the message area

Type: Character

Valid Values: Blue, Red, Pink, Green, Cyan, Yellow, White, Orange, Black, Magenta, Gray, Brown, Sysback, Syssecb, Sysfore, Message

---

**mousePointer**

Returns or sets the shape of the cursor or mouse pointer for the frame
Frame Class △ popMenuSupport 37

---

### popMenuSupport

Returns or sets a flag indicating whether popup (or context) menu support occurs on the frame

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

---

### name

Returns or sets the name of the object

**Type:** Character  
**Valid Values:**

---

### pmenuEntry

Returns or sets the name of the PMENU catalog entry that contains the menu associated with the frame

**Type:** Character  
**Valid Values:**

Editor: `SASHELP.FSP.PMEDITOR.SCL`

---

### popMenuProcessing

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

**Type:** Character  
**Valid Values:** Ignore it, Run main, Run object label, Run _popup method

---

### popMenuSupport

Returns or sets a flag indicating whether popup (or context) menu support occurs on the frame
promptCharacter

Returns or sets the character used for user input prompts.
Type: Character
Valid Values: Yes, No

resource

Returns the 4-level resource name of the frame
Type: Character
Valid Values:

SCLEntry

Returns or sets the name of the SCL entry associated with the frame
Type: Character
Valid Values:
Editor: sashelp.fsp.scLEntryEditor.scl

showBlockCursor

Returns or sets whether the block cursor will be hidden
showContextHelp

Returns or sets a flag indicating whether context help is supported by the frame

Type: Character
Valid Values: Yes, No

---

title

Returns or sets the title of the frame

Type: Character

---

type

Returns or sets the type of window in which to display the frame

Type: Character
Valid Values: Standard, Dialog

---

verticalPosition

Returns or sets the upper left row coordinate that specifies the position of the frame

Type: Numeric
Valid Values:

---

**width**

Returns or sets the width of the frame

**Type:** Numeric

**Valid Values:**

---

**windowSizeIncludes**

Returns or sets whether the window size includes borders and scroll bars, or only the frame’s master region

**Type:** Character

**Valid Values:** Borders, Master

---

**Methods**

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

---

**_addDragOp**

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

**Syntax**

```plaintext
return = objectName._addDragOp();
```
**_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 the frame can handle as a drop request

**Syntax**

```
return = objectName_addDropAction( );
```

**Details**

The action operation pair is matched with the drag operation of a drag site object to determine if the drop is allowed. 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 the frame 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. The _addDropOp method matches the operation variable value with the drag operation of a drag site object to determine if the drop is valid.

_addDropRep

Registers a representation as valid for dropping

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 when you edit or browse a FRAME entry. It runs after the _binit method and after the FRAME entry creates the display. The _bPostInit of the frame sends the _bPostInit to all objects in the frame.
**_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 is run automatically by the _attributesDialog method when you select **OK** (not CANCEL) from the General Attributes window. The default behavior updates the frame's colors and window size.

**_childPopup**

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

Syntax

```
objectName__childPopup( items, objectID, selection );
```
### Argument Type Use Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</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

This method is invoked by a widget's _childPopup method (see the _childPopup method in the Widget class for more information). This method gives the frame object a chance to modify the list that is displayed.

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 (removing items that were added, adding items that had been 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. See the example of the _popup method in the Widget class.

---

### _clearHelpMode

Clears the help environment

---

### Syntax

```plaintext
objectName._clearHelpMode();
```

### Details

This method runs automatically when the help environment is disabled. The default behavior is to invoke the _clearHelpMode method on all objects in the frame. See _clearHelpMode_ in the Widget class.

---

### _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>Update</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 code is not ‘Cancel’. Remove any visual changes added in the _startDrag method. If no valid drop occurs, code is ‘Cancel’.

__convertBox

Converts region coordinates to a different set of units

Syntax

_objectName_convertBox( regionCoordinatesList, units );
### Argument Type Use Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regionCoordinatesList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of a list containing region coordinates: ULX, ULY, LRX, LRY</td>
</tr>
<tr>
<td>units</td>
<td>Character</td>
<td>Input</td>
<td>specifies the units to convert to: CHARACTERS, INCHES, CM, MM, FONTS (default), or PIXELS</td>
</tr>
</tbody>
</table>

### Details

Use the `_getRegion` and `_getProperties` methods to get a list of attributes for an object. Part of the list returned includes region attributes.

The conversion of the region coordinates converts from the units currently listed to the units passed into the method. For more information on region coordinates and units, see the `_new` method in the Class class.

---

### `_cursorTracker`

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

**Syntax**

```plaintext
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 X (horizontal) coordinate location of the mouse pointer</td>
</tr>
<tr>
<td>y</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the Y (vertical) coordinate location of the mouse pointer</td>
</tr>
</tbody>
</table>

---

### `_cursorTrackingOff`

Disables cursor tracking for the frame

**Syntax**

```plaintext
objectName._cursorTrackingOff( );
```
Details
Cursor tracking can be enabled using the _cursorTrackingOn method and can be queried using the _isTracking method.

_cursorTrackingOn

Enables cursor tracking for the frame

Syntax
objectName_cursorTrackingOn();

Details
Cursor tracking can be disabled using the _cursorTrackingOff method and can be queried using the _isTracking method.
Some hosts do not allow you to invoke the _cursorTracker method while the mouse pointer is moving across native widgets (for example, list boxes and push buttons).

_disableDragDropSite

Completely disables a drag-and-drop site

Syntax
objectName_disableDragDropSite();

_disableReenterNotify

Disables the _reenter method

Syntax
objectName_disableReenterNotify();
Details
Disabling the _reenter method prevents the _reenter method from being invoked when
the frame is re-entered. For more information, see _reenter in this class.

_disableResizeNotify
Disables the _resize method

Syntax
objectName._disableResizeNotify();

Details
Disabling the _resize method prevents the _resize method from being invoked when the
frame is resized. For more information, see _resize in this class.

_disableWindow
Disables the window for input

Syntax
objectName._disableWindow();

_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 );
### Argument Type Use Description

<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>Update</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( );
```
_enableReenterNotify

Enables the _reenter method

Syntax

objectName._enableReenterNotify();

(enableResizeNotify

Enables the _resize method

Syntax

objectName._enableResizeNotify();

(enableWindow

Enables the window for input

Syntax

objectName._enableWindow();

_getArglist

Gets an SCL list containing arguments passed to the frame
Syntax

ObjectName_getArglist( argumentsList );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argumentsList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned is filled with the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>parameters that were passed to the frame (via CALL DISPLAY)</td>
</tr>
</tbody>
</table>

Details

The_getArglist method specifies the identifier of an SCL list that will contain the actual parameters that were passed to the frame (via CALL DISPLAY). If no parameters were passed, argList is a missing value. The primary purpose of this method is to allow you access to the argument list during a FRAME _init method. The _init method runs before the INIT: section, so the ENTRY statement has not been processed. Use this method also to process parameters for FRAME entries that do not have associated SCL source code.

This method is normally called from the _init method. It cannot be used in the INIT section of the frame unless the ENTRY statement of the frame contains an ARGLIST= or REST= specification.

The list identified by argList is created automatically when the method is called and deleted automatically when the FRAME entry ends. The list has the same semantics as the ARGLIST= list. In fact, if the FRAME ENTRY statement contains an ARGLIST=L2, the list identifiers for argList and L2 are the same. The ENTRY statement, if any, must match the incoming parameters. Otherwise, the method fails, causing a program halt. If the frame does not contain a program, all actual parameters are placed in argList.

If you use the REST= option on the ENTRY statement, _getArglist returns the list identifier for the REST= list. This list may not contain all the arguments. See Example 3.

If _getArglist is invoked from the frame's _init method, you can modify the values of the items on the list. When the INIT section of the frame runs, the ENTRY statement receives the new values. See Example 2.

The TYPE of items in argList (that is, the values returned from ITEMTYPE) is always either 'C' or 'N'. If a list is passed as an argument (as in Example 1), the list identifier of that list is inserted into argList as a numeric item (ITEMTYPE of 'N') rather than as a list (ITEMTYPE of 'L').

_getCharVar

Returns the value of a character variable or array item contained in the frame's SCL entry

Syntax

ObjectName_getCharVar( variableName, value );
objectName_getCharVar( variableName, value, dimension );
objectName_getCharVar( variableName, value, dimension, lowerBound );
objectName_getCharVar( variableName, value, dimension, lowerBound, upperBound );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variableName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the SCL variable to change</td>
</tr>
<tr>
<td>value</td>
<td>Character</td>
<td>Output</td>
<td>returns the value of the SCL variable</td>
</tr>
<tr>
<td>dimension</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the dimension for which you want to know the number of elements of the array and returns the dimension of that array</td>
</tr>
<tr>
<td>lowerBound</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the lower bound of the specified dimension when variableName is an array</td>
</tr>
<tr>
<td>upperBound</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the upper bound of the specified dimension when variableName is an array</td>
</tr>
</tbody>
</table>

**Details**

Variables contained in a frame’s SCL program are usually accessed from the SCL program itself. To access a character variable or array item from within a method, use the `getCharVar` method.

The `getCharVar` method can also determine the dimensions and the upper and lower bounds of both one-dimensional and multidimensional arrays. The `dim`, `lowBound`, and `highBound` arguments are similar to the `DIM`, `LBOUND`, and `HBOUND` SAS functions.

---

**getComponents**

Fills a list of all components in the current frame

**Syntax**

objectName_getComponents( components );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>components</td>
<td>List</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned contains all components in the frame</td>
</tr>
</tbody>
</table>
_getCurrentName

Returns the name of the current widget

Syntax

objectName_getCurrentName( 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 current widget. A blank will be returned if there is no active widget in the frame.</td>
</tr>
</tbody>
</table>

Details

The current widget is the last widget that was either selected or modified or that received the last command that was issued. (When a command is issued, the current widget is the one in which the text cursor is positioned. If your system is equipped with a mouse, the last widget selected and the widget in which the text cursor is positioned are not necessarily the same.)

_getCurrentWidget

Returns the identifier of the current widget

Syntax

objectName_getCurrentWidget( widgetID );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widgetID</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the object identifier of the current widget. A zero will be returned if there is no active widget in the frame.</td>
</tr>
</tbody>
</table>

Details

The current widget is the last widget that was either selected or modified or that received the last command that was issued. (When a command is issued, the current widget is the one in which the text cursor is positioned. If your system is equipped with
a mouse, the last widget selected and the widget in which the text cursor is positioned are not necessarily the same.)

---

**_getCursorShape**

Gets the cursor shape (mouse pointer shape) of the frame

**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 the cursor 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>
</tbody>
</table>
### _getDragReps

Returns the drag representations currently defined for this object

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>Update</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 data 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 operationsList 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**

```
objectName._getDragReps( 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 drag representations operations</td>
</tr>
</tbody>
</table>

**Details**

Note that `representationsList` can be modified and sent back to the `_setDragRep` method using `callApply` to perform global changes in the drag representations.

---

**_getDropOps**

**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**

**Syntax**

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

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

---

**Syntax**

```
objectName._getDropReps( representationsList );
```
Frame Class

\_getName

<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 representationList to the _setDropRep method using call Apply to perform global changes in the drop representations.

\_getMsg

Returns the message assigned to a frame

**Syntax**

```
objectName\_getMsg( message );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Character</td>
<td>Output</td>
<td>returns the text displayed on the frame's message line</td>
</tr>
</tbody>
</table>

**Details**

The message assigned to a FRAME entry displays on the message line when the frame is refreshed. Note that _getMsg may not return all system messages written to the message line.

Use the _getMsg method in an SCL program that is not running in a frame. For example, use this method for an SCL method, where the _msg variable is not directly accessible.

You can call the _getMsg method with either the CALL NOTIFY routine or the CALL SEND routine. However, to access the _msg variable in another frame, you must use CALL SEND because CALL NOTIFY works only within a frame.

\_getName

Returns the four-level name of the FRAME entry

**Syntax**

```
objectName\_getName( entryName );
```
### _getNumVar_

Returns the value of a numeric variable or array item contained

#### Syntax

```plaintext
objectName._getNumVar( variableName, value );
objectName._getNumVar( variableName, value, dimension );
objectName._getNumVar( variableName, value, dimension, lowerBound );
objectName._getNumVar( variableName, value, dimension, lowerBound, upperBound );
```

#### Argument Type Use Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entryName</td>
<td>Character</td>
<td>Output</td>
<td>returns the catalog entry name of the frame</td>
</tr>
<tr>
<td>variableName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the SCL variable to change</td>
</tr>
<tr>
<td>value</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the value of the SCL variable</td>
</tr>
<tr>
<td>dimension</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the dimension for which you want to know the number of elements of the array and returns the dimension of that array</td>
</tr>
<tr>
<td>lowerBound</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the lower bound of the specified dimension when variableName is an array</td>
</tr>
<tr>
<td>upperBound</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the upper bound of the specified dimension when variableName is an array</td>
</tr>
</tbody>
</table>

#### Details

Variables contained in a frame's SCL program are usually accessed from the SCL program itself. To access a numeric variable or array item from within a method, use _getNumVar_ method.

_getNumVar_ can also determine the dimensions and the upper and lower bounds of both one-dimensional and multidimensional arrays. The dimension, lowerBound, and upperBound arguments are similar to the DIM, LBOUND, and HBOUND SAS functions.
ftware Document

.Frame Class

getStatus

Fills an SCL list with all widgets that are currently selected in the frame

Syntax

objectName._getSelectedWidgets( widgets );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widgets</td>
<td>List</td>
<td>Update</td>
<td>specifies the identifier of the SCL list that when returned contains all currently selected widgets in the frame</td>
</tr>
</tbody>
</table>

.getSelf

Returns the object whose method is the current frame

Syntax

objectName._getSelf( selfID );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selfID</td>
<td>Numeric</td>
<td>Update</td>
<td>returns the identifier of the current object or returns zero</td>
</tr>
</tbody>
</table>

Details

When a method on an object X is a FRAME entry rather than an SCL entry, the._getSelf method returns that object’s identifier when the frame is executing. Normally, the object’s identifier is available in the special variable_self; this is true for the SCL attached to the FRAME entry, for example. However, if you have subclassed the Frame class and overridden one or more of the Frame class methods, when those methods execute, _self becomes the identifier of the frame, not the previous object. Use _getSelf to get the object identifier for object X.

getStatus

Returns the value of a frame’s _status_ variable
**Syntax**

objectName_getStatus( 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>Output</td>
<td>returns the value to assign to the frame's <em>STATUS</em> variable</td>
</tr>
</tbody>
</table>

**Details**

When you access the _status variable from another SCL program, you must call the _getStatus method with the CALL SEND or CALL APPLY routine.

---

**_getTable**

Returns the object identifier for the extended table containing a specified widget

---

**Syntax**

objectName_getTable( objectName, extendedTableID );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of a widget in the frame</td>
</tr>
<tr>
<td>extendedTableID</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the object identifier for the extended table containing the widget or a value of zero if the widget is not in an extended table</td>
</tr>
</tbody>
</table>

---

**_getWidget**

Returns a widget’s identifier

---

**Syntax**

objectName_getWidget( widgetName, widgetID );
FrameClass △ _inError 61

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widgetName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the widget in the frame</td>
</tr>
<tr>
<td>widgetID</td>
<td>Numeric</td>
<td>Output</td>
<td>returns the object identifier for the specified widgetName</td>
</tr>
</tbody>
</table>

_getWidgets

Fills an SCL list of all widgets in the current frame

Syntax

```
objectName_getWidgets( widgetsList );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widgetsList</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the identifier of a list that when returned is filled with object identifiers of the widgets in the frame</td>
</tr>
</tbody>
</table>

Details

_getWidgets fills a list with all widgets in the frame. This includes widgets that are not available for method calls, such as CSF components, widgets in extended tables, and composite object components.

_inError

Returns whether any frame objects are in error

Syntax

```
objectName_inError( errorState );
```
### _initLabel

**Description**

The `_inError` method returns whether any frame objects are in error. See the `_inError`, `_erroron`, and `_erroroff` methods of the Widget class.

**Syntax**

objectName._initLabel();

**Details**

The `_initLabel` method, which is normally used when the Frame class is subclassed, is not called by SCL code; it is invoked automatically by the frame when the frame is initialized. By default, the `_initLabel` method runs the INIT section of the frame’s SCL program.

This method can be overridden to perform some pre- or postprocessing with respect to the INIT section. For example, to process information both before and after the INIT section of the frame’s SCL program runs, you can override the `_initLabel` method of the Frame class, and write your method similar to this:

```scl
length _method_ $ 40; initlab: method; /*---- perform preprocessing ----------------------*/ /* some SCL statements here; */ /*---- run the INIT section of the FRAME ----------------*/ call super (_self_, _method_); /*---- perform postprocessing ----------------------*/ /* more SCL statements here; */ endmethod;
```

### _isHelpMode

**Description**

Returns whether the help environment is enabled or disabled.

**Syntax**

objectName._isHelpMode( enabled );
<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Numeric</td>
<td>Output</td>
<td>returns a value indicating whether the help environment is enabled: 1 if enabled, 0 if not enabled</td>
</tr>
</tbody>
</table>

### _isTracking

**Returns whether cursor tracking is enabled for the frame**

#### Syntax

```
objectName__isTracking( tracking );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tracking</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.

### _mainLabel

**Runs the MAIN section of the frame’s SCL entry**

#### Syntax

```
objectName__mainLabel( );
```

#### Details

The _mainLabel method, which is normally used when the Frame class is subclassed, is not called by SCL code; it is invoked automatically by the frame after all _objectLabel methods have run. By default, the _mainLabel method runs the MAIN section of the frame's SCL program.

This method can be overridden to perform pre- or postprocessing with respect to the MAIN section. For example, to process some information both before and after the
MAIN section of the frame's SCL program runs, you can override the _mainLabel method of the Frame class, and write your method similar to this:

```
length _method_ $ 40; mainlab: method; /*---- perform preprocessing -----------------------*/ /* some SCL statements here; */
/*---- run the MAIN section of the FRAME -----------*/ call super (_self_, _method_); /*---- perform postprocessing ----------------------*/ /* more SCL statements here; */
endmethod;
```

---

### _make

**Syntax**

```
objectName_make( );
```

---

### _postinit

**Performs additional processing after _initLabel runs**

**Syntax**

```
objectName_postinit( );
```

**Details**

The _postInit method, which is normally used when the Frame class is subclassed, is not called by SCL code; it is invoked automatically by the frame after the _initLabel method runs. By default, the _postInit method of the frame invokes the _postInit method for all objects.

---

### _preterm

**Performs additional processing before _termLabel runs**

**Syntax**

```
objectName_preterm( );
```
Details
The _preTerm method, which is normally used when the Frame class is subclassed, is not called by SCL code; it is invoked automatically by the frame before the _termLabel method runs. By default, the _preTerm method of the frame invokes the _preTerm method for all objects.

Setting _status to 'R' in the _preTerm method prevents the _termLabel method from running and prevents the frame from terminating.

_print

Allows custom printing for an application

----------

Syntax

objectName_print( );

Details
By default, the _print method invokes the _printObject method for all objects in the frame. The _printObject method has no default behavior and must be coded.

_reenter

Invoked when the frame is re-entered

----------

Syntax

objectName_reenter( );

Details
The _reenter method is invoked automatically when the _reenter method is enabled (see _enableReenterNotify) and the frame is re-entered. A frame is considered re-entered when it receives an event after a previous window was active. For example, if you are running a FRAME application and the application submits some code that causes the OUTPUT window or the GRAPH window to display, the frame is re-entered when you end from the OUTPUT window or the GRAPH window. Another example is when you have two FRAME applications running and the focus is in one of the applications: if you pop-up the window for the other application and begin to enter data or select widgets, the newly active application is re-entered.
A frame is not considered re-entered when returning from a CALL DISPLAY or any other SCL function that displays a window (for example, LIBLIST, DIRLIST, and ACCESS).

To enable or disable the _reenter method, use the _enableReenterNotify and _disableReenterNotify methods, respectively.

### _refresh

**Refreshes the display of a frame**

**Syntax**

```plaintext
objectName._refresh();
```

**Details**

The _refresh method invokes the _refresh method of all widgets that have the _needRefresh attribute set with the _needRefresh method. See _needRefresh in the Widget class for more details. The _refresh method is the same as the REFRESH statement.

When you invoke the _refresh method (or REFRESH statement) on a frame prior to the frame receiving a _postinit method, a _postinit method is sent to the frame, and then the refresh proceeds. This can occur if you invoke _refresh from the frame’s _init method, the _initLabel method, or the INIT section.

If you call the _refresh method (or the REFRESH statement) from the frame’s INIT section, it causes the _postinit method to run for the frame and all widgets before the _refresh method is sent to all widgets. Normally, the _postinit methods are sent after the INIT section finishes.

### _removeDragDropSite

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

**Syntax**

```plaintext
objectName._removeDragDropSite();
```
_resetCursorShape

Resets the cursor shape (mouse pointer shape) of the frame

Syntax

```
objectName._resetCursorShape();
```

Details

If the cursor shape had been changed through the _setCursorShape method, the original cursor shape can be restored using _resetCursorShape.

_resize

Invoked when the frame is resized

Syntax

```
objectName._resize();
```

Details

The _resize method is invoked automatically when the _resize method is enabled (see _enableResizeNotify) and the frame is resized. Many hosts allow windows to be resized by grabbing the window borders and moving them. The frame can also be resized via SCL functions, for example, WDEF.

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), you can call the _resize method 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.

_respondToDragOff

Allows a drop site object to reverse the effects of _respondToDragOnto
**Syntax**

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.

---

**_select**

**Syntax**

objectName_select( );

**Details**

A frame's _select method executes automatically any time users select or modify the frame. The default _select method executes the SAS command associated with the frame. If no command has been assigned to the frame, the default _select method does nothing.
If a labeled section of code in the SCL program is associated with the frame, 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.

_setCharVar

Sets the value of a character variable or array item contained in the frame’s SCL entry

Syntax

objectName_setCharVar( variableName, value );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variableName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the name of the SCL variable to change</td>
</tr>
<tr>
<td>value</td>
<td>Character</td>
<td>Output</td>
<td>specifies the value to set on the SCL variable</td>
</tr>
</tbody>
</table>

Details

Variables contained in a frame’s SCL program are usually accessed from the SCL program itself. To set the value of a character variable or array item from within a method, use the _setCharVar method.

_setColor

Changes the colors of the frame

Syntax

objectName_setColor( area, color, attribute );

objectName_setColor( area, color );
### _setControl

Controls the execution of labeled program sections and formatting of SUBMIT blocks

#### Syntax

```javascript
objectName_setControl( options );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Character</td>
<td>Input</td>
<td>specifies a single, quoted string containing one or more control options for program statements</td>
</tr>
</tbody>
</table>

### _setCursorShape

Sets the cursor shape (mouse pointer shape) for the frame

#### Syntax

```javascript
objectName_setCursorShape( shape );
objectName_setCursorShape( shape, immediate );
```
<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>
<tr>
<td>immediate</td>
<td>Character</td>
<td>Input</td>
<td>specifies whether to change the shape immediately: NO (default) which changes the shape the next time the frame refreshes</td>
</tr>
</tbody>
</table>

**Details**

You can change the cursor shape or mouse pointer shape from the default arrow (shape 2) to any shape in the following table:

<table>
<thead>
<tr>
<th>Shape</th>
<th>Description</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</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>

To reset the cursor shape to the original value, use `_resetCursorShape`. Normally, the cursor shape changes the next time the frame is refreshed. However, there may be times when you want the shape to change immediately. Not all hosts support changing the cursor shape.

---

**_setDragOp**

Sets the type of operations that the frame can handle as a drag request
Syntax
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 this representation as a valid dragging representation

Syntax
return = objectName_setDragRep( );

_setDropAction

Sets the type of actions the frame can handle as a drop request

Syntax
return = objectName_setDropAction( );

Details
The action operation 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 the frame 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 operation 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.
_setHelp

Sets the HELP entry to be associated with the frame

Syntax

objectName.setHelp( entryName );
objectName.setHelp( entryName, CBTFraneName );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entryName</td>
<td>Character</td>
<td>Input</td>
<td>specifies the catalog entry name</td>
</tr>
<tr>
<td>CBTFraneName</td>
<td>Character</td>
<td>Input</td>
<td></td>
</tr>
</tbody>
</table>

Details

This method is used to change the general help associated with a frame. Help is associated with the frame in the General Help window from the General Attributes window.

If entryName is a one- or three-level name, the type is assumed to be a CBT.

If a one- or two-level name is specified, the entry is searched first in the current catalog, SASUSER.PROFILE, and then in SASHELP.FSP.

If the entry is a CBT, you can specify a CBT frame name. The CBT frame name is the name following the NAME= option on the frame indicator line.

_setHelpMode

Sets the help environment

Syntax

objectName.setHelpMode( );

Details

See the HELPMODE command. The default behavior is to invoke the _setHelpMode method on all objects in the frame. See _setHelpMode in the Widget class.

_setKeys

Sets the KEYS entry to be associated with the frame
### Syntax

```
objectName_setKeys( keysEntry );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keysEntry</td>
<td>Character</td>
<td>Input</td>
<td>specifies the one-, two-, three-, or four-level KEYS entry name</td>
</tr>
</tbody>
</table>

**Details**

A one- or two-level name assumes the keys are in the current catalog, SASUSER.PROFILE, or SASHELP.FSP.

### _setMsg

Assigns a message to a FRAME entry

```
objectName_setMsg( message );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Character</td>
<td>Input</td>
<td>specifies text to display on the message line when the frame is refreshed</td>
</tr>
</tbody>
</table>

**Details**

The message assigned to the frame is displayed on the message line when the frame is refreshed. This method is useful in SCL methods because in an SCL entry the message string is not directly accessible.

### _setNumVar

Sets the value of a numeric variable or array item contained in the frame’s SCL entry

```
objectName_setNumVar( variableName, value );
```
### _setPMENU

Assigns the PMENU entry to be associated with the frame

**Syntax**

```c
objectName_setPMENU( pmenu );
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmenu</td>
<td>Character</td>
<td>Input</td>
<td>specifies a one-, two-, three-, or four-level PMENU entry name</td>
</tr>
</tbody>
</table>

### _setStatus

Assigns a value to the _status_ variable

**Syntax**

```c
objectName_setStatus( status );
```
### _setStatus_

**Arguments**

<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 the value to assign to the frame's <em>STATUS</em> variable</td>
</tr>
</tbody>
</table>

**Details**

This method is useful with SCL methods in which the _status variable is not directly accessible.

Setting the _status to 'H' causes the application to halt the next time it returns from its processing loop. For example, if a text entry field's _validate method uses the _setStatus method to set the status to 'H', the FRAME's SCL program halts when the _validate method returns.

### _setTitle_

**Assigns the window title of the frame**

**Syntax**

```
objectName_setTitle(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 title</td>
</tr>
</tbody>
</table>

**Details**

If you are running with the SAS System's Application Work Space (AWS), the window title is contained in the inner window.

### _setWindowBanner_

**Assigns the banner of the frame**

**Syntax**

```
objectName_setWindowBanner(banner);
```
### _setWindowSize

Assigns the window size of the frame

#### Syntax

```plaintext
objectName_setWindowSize( );
objectName_setWindowSize( startingRow );
objectName_setWindowSize( startingRow, startingColumn );
objectName_setWindowSize( startingRow, startingColumn, numberOfRows );
objectName_setWindowSize( startingRow, startingColumn, numberOfRows, numberOfColumns );
objectName_setWindowSize( startingRow, startingColumn, numberOfRows, numberOfColumns, sizingOption );
```

#### Details

This method is similar to the WDEF SCL function. If a missing value is passed for any of the window sizes, that particular size will not change.

To prevent window flashing, this method should be invoked from the _init method of the frame. After the _init method, the window size of the frame is already set and changing the window size causes a flash.

To change the sizing policy of a frame, you can pass missing values for all the window sizes and specify the new sizing policy. See the following example.

### _setWindowType

Assigns the window type of the frame

#### Syntax

```plaintext
objectName_setWindowType( type );
```
Frame Class \_snapShot

<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>Input</td>
<td>specifies the window type: STANDARD or DIALOG</td>
</tr>
</tbody>
</table>

**Details**

This method is valid only from the \_init method of a frame.

\_snapShot

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

**Syntax**

```
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 );
```

<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>
</tbody>
</table>
### Chapter 3

<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>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.

---

### _snugFit

**Resizes the window to fit around the objects it contains**

#### Syntax

objectName_snugFit( n1 );

objectName_snugFit( clearance, units );

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>Numeric</td>
<td>Update</td>
<td>specifies the amount of space to leave between the window and the objects it contains</td>
</tr>
<tr>
<td>clearance</td>
<td>Numeric</td>
<td>Input</td>
<td>specifies the units to convert for clearance: CHARACTERS, INCHES, CM, MM, FONTS, or PIXELS (default)</td>
</tr>
</tbody>
</table>

#### Details

The units are honored when the new size of the frame is computed, but when the frame is resized, the snug fit size is rounded to row and column values. Only the right and bottom borders of the window are resized. The top and left borders are unchanged. That is, the number of rows and columns in the window change but the starting row and column of the window do not change. For more information on units, see the _new method in the Class class.

---

### _startDrag

**With _completeDrag, performs possible visual changes to an object selected for dragging**
**Syntax**

objectName_startDrag( );

objectName_startDrag( objectID );

**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.

---

**_termLabel**

Runs the TERM section of the frame’s SCL entry

---

**Syntax**

objectName_termLabel( );

**Details**

The _termLabel method, which is normally used when the Frame class is subclassed, is not called by SCL code; it is invoked automatically by the frame when the frame is ended. By default, the _termLabel method runs the TERM section of the frame’s SCL program.

Setting _status to ‘R’ in the _termLabel method resumes the frame.

This method can be overridden to perform pre- or postprocessing with respect to the TERM section. For example, to process information both before and after the TERM section of the frame’s SCL program runs, you can override the _termLabel method of the Frame class, and write your method similar to the example below.

---

**_update**

Updates the frame

---

**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.

_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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the representation</td>
</tr>
<tr>
<td>dragStart</td>
<td>Character</td>
<td>Input</td>
<td>specifies where the drag started:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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>
</tbody>
</table>
Argument | Type   | Use    | Description
---------|--------|--------|--------------------------------------------------
cancel    | Character | Output | returns a flag indicating whether to complete or cancel the drop: ‘C’ indicates cancel otherwise a blank indicates to continue

objectId | (1) | Update | specifies the identifier of an object containing drag and drop data

(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.

赢info

Returns information about the current frame

Syntax

objectName_winfo( windowItem, arg1 );
objectName_winfo( windowItem, arg1, arg2 );

Argument | Type   | Use    | Description
---------|--------|--------|--------------------------------------------------
windowItem | Character | Input | specifies a characteristic of the window or an action
arg1      | Numeric | Update | returns the value of windowItem
arg2      | Numeric | Output | returns the value of windowItem

Details

infoItem can be one of the following:

‘BACKCOLOR’
‘BATCH’
'COMMAND'
'CURSCREEN'
'CURSORCOL'
'CURSORROW'
'GRAPHICS'
'ICON'
'LOGON'
'MAXCOL'
'MAXROW'
'MONO'
'NSCREEN'
'NUMCOLS'
'NUMROWS'
'NUMXINCH'
'NUMXPIXEL'
'NUMYINCH'
'NUMYPixel'
'PANECCOL'
'PANECCOL'
'PANECROW'
'PANEROW'
'PMENU'
'PMENUGRAY'
'PMENUSTATE'
'POPUP'
'STARTCOL'
'STARTROW'
'UICON'
'XPIXCELL'
'XPIXEL'
'YPIXCELL'
'YPIXEL'

Because _init runs before a display is created, you cannot use 'PMENUGRAY' and 'PMENUSTATE'.

If a frame's method is overridden and points to an SCL entry, using the WINFO function within the overridden method returns information about the SCL entry rather than the frame. To get information about the frame, use _winfo.

If _winfo is invoked from the _init method of the frame, not all infoItems can be queried. The _init method of the frame runs before the display is created, so many of the infoItems that return information about the display cannot return valid values.
Events

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

SCLEntry changed
   Occurs when the SCLEntry attribute is changed
automaticCompile changed
   Occurs when the automaticCompile attribute is changed
backgroundColor changed
   Occurs when the backgroundColor attribute is changed
bannerAttribute changed
   Occurs when the bannerAttribute attribute is changed
bannerColor changed
   Occurs when the bannerColor attribute is changed
bannerType changed
   Occurs when the bannerType attribute is changed
borderAttribute changed
   Occurs when the borderAttribute attribute is changed
borderColor changed
   Occurs when the borderColor attribute is changed
commandAttribute changed
   Occurs when the commandAttribute attribute is changed
commandColor changed
   Occurs when the commandColor attribute is changed
commandProcessing changed
   Occurs when the commandProcessing attribute is changed
confirmOnExit changed
   Occurs when the confirmOnExit attribute is changed
cursorPlacement changed
   Occurs when the cursorPlacement attribute is changed
defaultPushButton changed
   Occurs when the defaultPushButton 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
forcePmenuOn changed
   Occurs when the forcePmenuOn attribute is changed
foregroundAttribute changed
   Occurs when the foregroundAttribute attribute is changed
foregroundColor changed
   Occurs when the foregroundColor attribute is changed
height changed
   Occurs when the height attribute is changed
horizontalPosition changed
   Occurs when the horizontalPosition attribute is changed
icon changed
   Occurs when the icon attribute is changed
keysEntry changed
   Occurs when the keysEntry attribute is changed
messageAttribute changed
   Occurs when the messageAttribute attribute is changed
messageColor changed
   Occurs when the messageColor attribute is changed
mousePointer changed
   Occurs when the mousePointer attribute is changed
pmenuEntry changed
   Occurs when the pmenuEntry attribute is changed
popMenuProcessing changed
   Occurs when the popMenuProcessing attribute is changed
popMenuSupport changed
   Occurs when the popMenuSupport attribute is changed
promptCharacter changed
   Occurs when the promptCharacter attribute is changed
showBlockCursor changed
   Occurs when the showBlockCursor attribute is changed
showContextHelp changed
   Occurs when the showContextHelp attribute is changed
title changed
   Occurs when the title attribute is changed
type changed
   Occurs when the type attribute is changed
verticalPosition changed
   Occurs when the verticalPosition attribute is changed
width changed
   Occurs when the width attribute is changed
windowSizeIncludes changed
   Occurs when the windowSizeIncludes attribute is changed