Overview

A toolbar consists of one or more buttons with a label or an image or both that perform some action when selected or deselected, such as opening files, running an application, or changing a font. In addition to selecting images, labels, and actions for each button, you can:

- control toolbar layout and step through buttons when they cannot all be viewed at once
- select fonts for labels
- specify descriptive messages for each button
- specify whether buttons act like push buttons, check boxes, or radio buttons
- save toolbar definitions in SCL lists or as catalog SLIST entries for reuse.

The Toolbar class also supports drag and drop in both the Build and the run-time environments. This feature allows you to copy and move toolbar buttons within and between toolbars by dragging and dropping the buttons where you want them. Moreover, it lets you change button text, images, and functionality while building the toolbar.

PARENT: SASHELP.FSP.COMPOSIT.CLASS
CLASS: sashelp.fsp.Palette.class

Using the Toolbar Class

How the Toolbar Works

There are two ways to make toolbar buttons perform actions:
1 Clicking on a toolbar button runs the command specified in the Commands section of the Enter Values window, or assigned with the _setCmd method, or specified in the Command Processing window (see the Widget class).

2 Clicking on a toolbar button runs a labeled section of the SCL program. Include these steps in your program:
   - call the _getLastSel method to determine which button has been selected
   - use conditional processing to decide what action to take, that is, which section of code to run
   - run the appropriate section of the SCL program.

   For example, this program determines which of two buttons has been selected in the toolbar named TBAR1 and runs the appropriate labeled section:

   ```scl
   length name $24;
   TBAR1:
   call notify ('tbar1', '_get_last_sel_',
               index, status, name);
   if name='PRINT' then link PRINT;
   else if name='DATASET' then link OPEN;
   return;
   PRINT:
   /* code for printing here */
   return;
   OPEN:
   /* code for opening a data set here */
   return;
   ```

---

**Scroll Buttons**

If a toolbar has more buttons than can be displayed in the space available, scroll buttons are automatically added to each end of the toolbar. Two scroll buttons (left and right or up and down) allow you to step through all the buttons in a toolbar.

Each scroll button occupies one button space in the toolbar. For example, if you specify a toolbar with six button spaces and eight buttons, the toolbar displays the left and right scroll buttons and four of the eight action buttons. Therefore, a toolbar that uses scroll buttons must have a minimum of three button places.

Toolbars with scroll buttons do not have any space between the buttons, even if space is specified in the Attributes window.

---

**Drag and Drop**

The Toolbar object supports drag and drop. By default drag and drop is enabled at Build time and disabled at run time. The toolbar uses drag and drop in three ways:

1. In the Build environment, you can move or copy a button by dragging it to the desired location, either within the same toolbar or within a different toolbar. To delete a button, use the Attributes window.

2. The same functionality is available in the run-time environment if you explicitly enable it for the toolbar. For example, to enable drag and drop for the toolbar
called TBAR1, include the _enableDragDropSite method of the Widget class in the SCL program:

```
INIT:
call notify ('tbar1',
                '_enable_drag_drop_site_');
return;
```

This allows the user of the toolbar to copy and move toolbar buttons at run time.

To turn off drag and drop at run time, call the _disableDragDropSite method of the Widget class.

3 In the run-time environment, you can drag onto a toolbar any widget that has been set up as a drag site and assigned one of the Toolbar drag representations that define the kind of data a toolbar button can accept. By default, the Toolbar class supports three drag representations:

- `_dndPaletteBUTTON` is an SCL list that contains complete information about the button.
- `_dndPaletteIMAGE` is an image stored in SASHELP.I0x0x that can be used by the button as its image.
- `_dndPaletteLABEL` is text that can be used as the label for the button.

These representations allow you to edit a toolbar by changing the text, image, or functionality of a button by entering the data in another widget and dragging that widget onto the button.

For more information on drag and drop concepts, refer to SAS/AF online help.

If you want to know if something has been dragged out of your toolbar, or if something has been dropped on your toolbar, then you need to subclass the toolbar and override one or more of the three toolbar drag-and-drop methods and possibly one or more of the Widget drag-and-drop methods.

---

**Example: Changing Button Labels and Images with Drag and Drop**

This example creates a toolbar in which the user can dynamically change a toolbar button label or image at run time. The frame contains a toolbar and two text entry fields in which the user enters new text or the name of a new image. The contents of each field can be dragged to a toolbar button where it replaces the current label or image.

1 In the SASUSER library create a catalog called FRAMECAT in which to store the FRAME entry and the associated SCL programs.

2 Create a FRAME entry containing a toolbar, TBAR1, and two text entry fields named LABEL and IMAGE. The toolbar should have one or more buttons and the Button Display attribute should be Image and label.

3 Provide the SCL program for the frame:

```
INIT:
    /* Make the toolbar a drag/drop site. */
    call notify('tbar1',
                '_enable_drag_drop_site_');
```
Example: Changing Button Labels and Images with Drag and Drop

Chapter 106

/* Make both text entry fields drag */
/* sites and assign the appropriate */
/* data representation. */
call notify('image', '_add_drag_rep_',
    '_dndPaletteIMAGE');
call notify('label', '_add_drag_rep_',
    '_dndPaletteLABEL');

/* Make the text entry fields */
/* drag/drop sites. */
call notify('image',
    '_enable_drag_drop_site_');
call notify('label',
    '_enable_drag_drop_site_');

/* Override the _getDragData */
/* methods for both fields. Point to */
/* the SCL program that implements */
/* the methods. */
call notify('image',
    '_set_instance_method_',
    '_getDragData',
    'sasuser.framecat.tbarmeth.scl',
    'getimage');
call notify('label',
    '_set_instance_method_',
    '_getDragData',
    'sasuser.framecat.tbarmeth.scl',
    'getlabel');

return;

Compile the code and close the SCL entry.

4 Provide the SCL program that defines the methods (SASUSER.FRAMECAT.TBARMETH.SCL):

    length value $200;

    /* Store the image data in a list. */
    GETIMAGE: method rep $200 op $200 data 8 x 8 y8 / RESIDENT;
        call send(_self_, '_get_text_', value);
        setnitemc(data, value, 'IMAGE');
    endmethod;
    /*newpage*/

    /* Store the label data in a list. */
    GETLABEL: method rep $200 op $200 data 8 x 8 y8 / RESIDENT;
        call send(_self_, '_get_text_', value);
        setnitemc(data, value, 'LABEL');
    endmethod;

    Compile the code and save or close the SCL entry.

5 Run TESTAF. In the TESTAF window,
   - enter new text for a button label in the Label field, press ENTER, and drag the text onto the button.
Methods

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

Dictionary

_activate

Selects or deselects the specified button and runs the appropriate commands

Syntax

CALL NOTIFY (toolbar-name, '_activate', button-name);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button-name</td>
<td>C</td>
<td>specifies the name of the button</td>
</tr>
</tbody>
</table>

Details

_activate has exactly the same effect as the user clicking on the button. For buttons with Check Box behavior, the state of the button changes. For buttons with Radio Button behavior, the button becomes active.

_buttonCompleteDrag

Called by the button in the toolbar when a drag operation is completed

Syntax

CALLSUPER(_SELF_, '_buttonCompleteDrag', index, rep, op, data);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>N</td>
<td>returns the index of the button dragged out of a toolbar</td>
</tr>
<tr>
<td>rep</td>
<td>C</td>
<td>returns the representation selected for the drop</td>
</tr>
</tbody>
</table>
### _buttonDrop

Called by the button in the toolbar when a drop operation occurs on that button

#### Syntax

```plaintext
CALLSUPER(_SELF_, '_buttonDrop', index, rep, op, data, task, x, y);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>N</td>
<td>returns the index of the button on which the drop occurred</td>
</tr>
<tr>
<td>rep</td>
<td>C</td>
<td>returns the representation selected for the drop</td>
</tr>
<tr>
<td>op</td>
<td>C</td>
<td>returns the operation selected for the drop</td>
</tr>
<tr>
<td>data</td>
<td>N</td>
<td>returns the identifier of an SCL list containing the data defined by the representation</td>
</tr>
<tr>
<td>task</td>
<td>C</td>
<td>returns a value indicating if the drag started in this window: 'INSIDE' 'OUTSIDE'</td>
</tr>
<tr>
<td>x</td>
<td>N</td>
<td>returns the X location of the drop in pixels</td>
</tr>
<tr>
<td>y</td>
<td>N</td>
<td>returns the Y location of the drop in pixels</td>
</tr>
</tbody>
</table>

#### Details

Only use this method when creating a subclass, in which case you override it to determine when a drop operation has occurred on one of the buttons in the toolbar.
Syntax
CALLSUPER(_SELF_, '_buttonGetDragData', index, data);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>N</td>
<td>returns the index of the button which the drag occurred on</td>
</tr>
<tr>
<td>data</td>
<td>N</td>
<td>returns the identifier of an SCL list containing the data defined by the representation</td>
</tr>
</tbody>
</table>

Details
Only use this method when creating a subclass, in which case you override it to add or remove information to the list returned by the Toolbar class's super method.

_getButtonIndex

Returns the index of the specified button

Syntax
CALL NOTIFY (toolbar-name, '_getButtonIndex', button-name, index);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button-name</td>
<td>C</td>
<td>specifies the name of the button</td>
</tr>
<tr>
<td>index</td>
<td>N</td>
<td>returns the index of the button</td>
</tr>
</tbody>
</table>

Details
Every button in a toolbar is assigned an index number (beginning with 1) that indicates its position in the toolbar. Index numbers are assigned beginning with the leftmost button in the first row and proceeding left to right, then row by row. Button arrangement is controlled by the _setConform method.

See Also
_getName

_getButtonWidth

Returns the width of the buttons
### Syntax

**CALL NOTIFY** (toolbar-name, `_getButtonWidth`, width);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>N</td>
<td>returns the width of the button in character cells</td>
</tr>
</tbody>
</table>

### _getCmd

Returns the command that runs when the current button is selected or deselected

#### Syntax

**CALL NOTIFY** (toolbar-name, `_getCmd`, button-name, down-cmd, up-cmd);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button-name</td>
<td>C</td>
<td>specifies the name of the button</td>
</tr>
<tr>
<td>down-cmd</td>
<td>C</td>
<td>returns the command that runs when the button is selected</td>
</tr>
<tr>
<td>up-cmd</td>
<td>C</td>
<td>returns the command that runs when the button is deselected; if the button is a push button, you must pass in a variable even though no command has been specified</td>
</tr>
</tbody>
</table>

#### Details

If you specify only one argument for this method, it is assumed to be cmd-string and _getCmd behaves as described in the Widget class and returns the command for the entire toolbar.

If the button has **Check Box** or **Radio Group** behavior, another command can be run when the button is explicitly deselected (**Check Box**) or another button is selected (**Radio Box**).

#### See Also

_getCmd in the Widget class.

### _getConformOption

Returns the way the buttons are arranged in the toolbar
Syntax

CALL NOTIFY (toolbar-name, '_getConformOption', conform<, n-rows<, n-cols>>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conform</td>
<td>C</td>
<td>returns a value indicating button arrangement:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' positioned to fit the region shape</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' positioned according to row and column specification</td>
</tr>
<tr>
<td>n-rows</td>
<td>N</td>
<td>returns the number of rows of buttons if conform is 'N'</td>
</tr>
<tr>
<td>n-cols</td>
<td>N</td>
<td>returns the number of columns of buttons if conform is 'N'</td>
</tr>
</tbody>
</table>

Details

If the buttons are laid out in rows and columns (the value of conform is N), you can specify n-rows and n-cols to find out the number of rows and columns.

_getDisplayType

Returns what the toolbar buttons display

Syntax

CALL NOTIFY (toolbar-name, '_getDisplayType', type);

<table>
<thead>
<tr>
<th>where... is type...</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>returns a value indicating what the buttons display:</td>
</tr>
<tr>
<td></td>
<td>'IMAGE' only an image</td>
</tr>
<tr>
<td></td>
<td>'LABEL' only a label</td>
</tr>
<tr>
<td></td>
<td>'BOTH' both an image and a label</td>
</tr>
</tbody>
</table>

_getFont

Returns information about the font used for the button label

Syntax

CALL NOTIFY (toolbar-name, '_getFont', font-list-id);
### `font-list-id`

Returns the identifier of an SCL list containing information about the font; returns 0 if no font list is specified for the toolbar.

**Details**

If the font list is empty, the DMS font is used.

### `_getHelp`

Returns the descriptive message (help text) associated with the specified button.

**Syntax**

```call notify```

`toolbar-name`, `_getHelp`, `button-name`, `text-string`;

**Argument Type Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>button-name</code></td>
<td>C</td>
<td>Specifies the name of the button</td>
</tr>
<tr>
<td><code>text-string</code></td>
<td>C</td>
<td>Returns the text of the message</td>
</tr>
</tbody>
</table>

**Details**

The message displays when the user selects and holds down a button.

### `_getImage`

Returns the name of the image displayed on the specified button.

**Syntax**

```call notify```

`toolbar-name`, `_getImage`, `button-name`, `image-name`;

**Argument Type Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>button-name</code></td>
<td>C</td>
<td>Specifies the name of the button</td>
</tr>
<tr>
<td><code>image-name</code></td>
<td>C</td>
<td>Returns the name of the image</td>
</tr>
</tbody>
</table>
### _getLabel

Returns the label of the specified button

**Syntax**

CALL NOTIFY (toolbar-name, '_getLabel', button-name, label);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button-name</td>
<td>C</td>
<td>specifies the name of the button</td>
</tr>
<tr>
<td>label</td>
<td>C</td>
<td>returns the label text associated with the specified button</td>
</tr>
</tbody>
</table>

### _getLastSel

Returns the name and index of the most recently selected or deselected button

**Syntax**

CALL NOTIFY (toolbar-name, '_getLastSel', index, is-sel, button-name);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>N</td>
<td>returns the index of the button or 0</td>
</tr>
<tr>
<td>is-sel</td>
<td>N</td>
<td>returns a value indicating the current status of the button:</td>
</tr>
<tr>
<td>button-name</td>
<td>C</td>
<td>returns the name of the button</td>
</tr>
</tbody>
</table>

**Details**

If a scrolling tool has been pressed or the user selected an area of the toolbar outside of a button, index contains zero.
_getName

Returns either the name of the button at the specified index position or the name of the toolbar.

Syntax

CALL NOTIFY (toolbar-name, _getName', name<, index>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>C</td>
<td>returns the name of the toolbar; if index is specified, returns the name of the button at the index position</td>
</tr>
<tr>
<td>index</td>
<td>N</td>
<td>specifies the index of the button</td>
</tr>
</tbody>
</table>

Details

Button names are used instead of the button index to access individual buttons in the toolbar because the button index changes when the button is moved. To find the name of the button that was last clicked on, use the _getLastSel method.

See Also

_getButtonIndex

_getSpaceAfter

Returns the number of separator spaces after the specified button.

Syntax

CALL NOTIFY (toolbar-name, _getSpaceAfter', button-name, space-after);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button-name</td>
<td>C</td>
<td>specifies the name of the button</td>
</tr>
<tr>
<td>space-after</td>
<td>N</td>
<td>returns a value from 0...9</td>
</tr>
</tbody>
</table>

_getToprow

Returns the index of the leftmost button in the toolbar.
Syntax

CALL NOTIFY (toolbar-name, '_getToprow', index<, visible>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>N</td>
<td>returns the index of the leftmost button. If the toolbar does not have scroll buttons, 0 is returned.</td>
</tr>
<tr>
<td>visible</td>
<td>N</td>
<td>returns the number of visible buttons that are not scroll buttons</td>
</tr>
</tbody>
</table>

Details

Note that the index value returned by _getToprow may not be the same as the index specified in the last call to _setToprow even if the toolbar has not been scrolled since the call to _setToprow. See _setToprow for more details.

_gray

Grays a single button or the entire toolbar so that it may not be selected

Syntax

CALL NOTIFY (toolbar-name, '_gray', button-name>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button-name</td>
<td>C</td>
<td>specifies the name of the button to gray. If button-name is omitted, the entire toolbar is grayed.</td>
</tr>
</tbody>
</table>

Details

Unlike _gray in the Widget class, this method has the optional argument button-name.

_isGray

Reports whether the specified button or the entire toolbar is grayed

Syntax

CALL NOTIFY (toolbar-name, '_isGray', is-gray<, button-name>);
<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is-gray</td>
<td>N</td>
<td>returns a value indicating the status of the toolbar or button:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = gray</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = not gray</td>
</tr>
<tr>
<td>button-name</td>
<td>C</td>
<td>specifies the button name. If button-name is omitted, is-gray returns the status of the entire toolbar.</td>
</tr>
</tbody>
</table>

**Details**

Unlike `isGray` in the Widget class, this method has the optional argument button-name.

---

**_repopulate**

Refills a toolbar with its original items

**Syntax**

```plaintext
CALL NOTIFY (toolbar-name, '_repopulate');
```

**Details**

 `_repopulate` re-reads the information about the toolbar from the button data source specified by the `_populate` instance variable and reconfigures and redisplays the toolbar based on this new information. Note that `_repopulate` does not reread the toolbar instance variables. Use the `_update` method to reread the instance variables as well as the button data source.

To change one attribute on one button, use a method like `setImage`, but to make significant changes to the toolbar button list, changing the list and calling `_repopulate` is more efficient. For example, you might call `_repopulate` after changing all the `IMAGE` items in the list that populates the toolbar.

**See Also**

  * `_update`

---

**_setButtonWidth**

Assigns the width of the buttons

**Syntax**

```plaintext
CALL NOTIFY (toolbar-name, '_setButtonWidth', width);
```
**Argument Type Description**

- **width**
  - Type: N
  - Description: specifies the width of the button:
    - n: the width of the button in character cells
    - 0: the default width of the button display type

**Details**

The button display type, which determines the display of labels and images, is set in the Attributes window. Use this method to make the buttons containing labels wider or narrower.

### _setCmd

Assigns the command to run when a button is selected or deselected

**Syntax**

CALL NOTIFY (toolbar-name, '_setCmd', button-name, down-cmd, up-cmd);

**Argument Type Description**

- **button-name**
  - Type: C
  - Description: specifies the name of the button
- **down-cmd**
  - Type: C
  - Description: specifies the command to run when the button is selected
- **up-cmd**
  - Type: C
  - Description: specifies the command to run when the button is deselected; if the button is a push button, enter a null value ("")

**Details**

The command that runs depends on the selection style of the button and the type of action:

<table>
<thead>
<tr>
<th>If the button is a...</th>
<th>Description</th>
<th>Then...</th>
</tr>
</thead>
<tbody>
<tr>
<td>push button</td>
<td>selects the button</td>
<td>down-cmd runs</td>
</tr>
<tr>
<td>check box</td>
<td>selects the button</td>
<td>down-cmd runs</td>
</tr>
<tr>
<td></td>
<td>deselects the button</td>
<td>up-cmd runs</td>
</tr>
</tbody>
</table>
If the button is a... | Description | Then...
---|---|---
button in a radio group | selects the button | down-cmd runs
 | selects another toolbar button in the radio group | up-cmd runs

**See Also**

_setCmd in the Widget class

### _setConformOption

Assigns the layout of the toolbar buttons and controls how many buttons are displayed

**Syntax**

CALL NOTIFY (toolbar-name, '_setConformOption', conform, <n-rows, n-cols>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conform</td>
<td>C</td>
<td>specifies a value indicating button arrangement: 'Y' arrange to fit the region 'N' arrange according to row and column specifications</td>
</tr>
<tr>
<td>n-rows</td>
<td>N</td>
<td>specifies the number of rows of buttons</td>
</tr>
<tr>
<td>n-cols</td>
<td>N</td>
<td>specifies the number of columns of buttons</td>
</tr>
</tbody>
</table>

**Details**

If conform is 'N', specifying the number of rows and columns arranges the buttons in a rectangle and resizes the object. Only the number of buttons defined by n-rows and n-cols are displayed. If all the buttons do not fit the region, scroll buttons are displayed.

### _setDisplayType

Controls what the toolbar buttons display

**Syntax**

CALL NOTIFY (toolbar-name, '_setDisplayType', type);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
Argument | Type | Description
---|---|---
type | C | specifies a value indicating what the buttons display:
\textit{IMAGE}' | only an image\n\textit{LABEL}' | only a label\n\textit{BOTH}' | both an image and a label

\_setFont

Assigns a font for the label text (if any)

**Syntax**

\texttt{CALL NOTIFY (toolbar-name, '_setFont', font-list-id);}

Argument | Type | Description
---|---|---
font-list-id | N | specifies the identifier of an SCL list created by the SCL function \texttt{FONTSEL} and containing font information.

\_setHelp

Assigns the descriptive message (help text) associated with the button

**Syntax**

\texttt{CALL NOTIFY (toolbar-name, '_setHelp', button-name, text-string);}

Argument | Type | Description
---|---|---
button-name | C | specifies the name of the button\ntext-string | C | specifies the descriptive message

details

A descriptive message appears on the status line when the user positions the cursor over the button and holds down the mouse button. (Moving the cursor off the toolbar button before releasing the mouse button ends the display of help and prevents any associated button action from occurring.)
**_setImage_**

Assigns the image to display on the specified button

---

**Syntax**

CALL NOTIFY (toolbar-name, '_setImage', button-name, image-name);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button-name</td>
<td>C</td>
<td>specifies the name of the button</td>
</tr>
<tr>
<td>image-name</td>
<td>C</td>
<td>specifies the one-level name of an image stored in the SASHELP.I0x0x catalog</td>
</tr>
</tbody>
</table>

**Details**

The image only appears when the toolbar display type is IMAGE or BOTH. See _getDisplayType and _setDisplayType in this class.

---

**_setLabel_**

Assigns a label to the specified button

---

**Syntax**

CALL NOTIFY (toolbar-name, '_setLabel', button-name, label);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button-name</td>
<td>C</td>
<td>specifies the name of the button</td>
</tr>
<tr>
<td>label</td>
<td>C</td>
<td>specifies the text of the label</td>
</tr>
</tbody>
</table>

**Details**

The label text only appears if the toolbar display type is LABEL or BOTH.

---

**_setName_**

Assigns a name to the button at the specified index

---
**Syntax**

**CALL NOTIFY** (toolbar-name, '_setName', index, button-name);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>N</td>
<td>the index of the button</td>
</tr>
<tr>
<td>button-name</td>
<td>C</td>
<td>the new name of the button at the specified index</td>
</tr>
</tbody>
</table>

**_setSpaceAfter**

Assigns the number of separator spaces after the specified button

**Syntax**

**CALL NOTIFY** (toolbar-name, '_setSpaceAfter', button-name, space-after);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button-name</td>
<td>C</td>
<td>specifies the name of the button</td>
</tr>
<tr>
<td>space-after</td>
<td>N</td>
<td>specifies a value from 0...9 indicating the number of spaces after the button. The default is 0.</td>
</tr>
</tbody>
</table>

**_setToprow**

Assigns the button which is leftmost in the toolbar

**Syntax**

**CALL NOTIFY** (toolbar-name, '_setToprow', index);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>N</td>
<td>specifies the index of the button to be leftmost</td>
</tr>
</tbody>
</table>

**Details**

Note that the specified button may not be the the leftmost button if this would leave a gap between the rightmost button and the right scroll button. If the toolbar does not have scroll buttons, calls to _setToprow are ignored.
**_ungray**

Ungrays a single grayed button or the entire toolbar

**Syntax**

CALL NOTIFY (toolbar-name, '_ungray', button-name);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button-name</td>
<td>C</td>
<td>specifies the name of the button to ungray. If button-name is omitted, the entire toolbar is ungrayed.</td>
</tr>
</tbody>
</table>

**Details**

Unlike _ungray in the Widget class, this method has the optional argument button-name.

---

**_update**

Rereads all the instance variables of the toolbar and redisplay the toolbar based on this new information

**Syntax**

CALL NOTIFY (toolbar-name, '_update');

**Details**

Note that the work _update performs is a superset of the work _repopulate performs. You might call _update on a toolbar if you changed the data source for the button information or you wanted to change more than one of the toolbar instance variables and wanted the update to occur only once. Method calls such as _setConformOption, which modify a toolbar instance variable, perform an _update as part of their processing.

**See Also**

_repopulate