Overview

The Work Area class is a tool for building dynamic applications in which the end user creates and interacts with a predefined set of items that are available from a pop-up menu. These items are defined by the application developer who builds a set of items (widgets) appropriate to the tasks of the application and assigns them all the necessary functionality through subclassing. At run time, the end user then dynamically creates and positions new widgets in the work area and performs tasks with them.

For example, a work area application for tracking stocks might provide a set of items appropriate to stock management, such as stock data, graphs, and news readers. These items would perform tasks such as reading data, creating graphs, and displaying text. The user would add items to the work area as needed and perform tasks with these items using selection, drag-and-drop functionality, or customized pop-up menus. In addition, the user can manipulate the items; for example, deleting, copying, or moving them. When the user finishes the work area session, all changes are saved so the work area can be restored to the state in which the user left it.

Using the Work Area Class

These are the basic application development steps for creating a simple work area:

1. Subclass the widgets to be used in the work area and define their functionality. The items within the work area contain all the logic to perform tasks, not the work area itself, which functions as a space manager. Therefore, the widgets should be subclassed and provided the necessary functionality before they are added to the work area.

2. Customize the work area pop-up menus to add functionality for the end user.
3 Add the new widgets to the RESOURCE entry for the frame or define a new RESOURCE entry with the work area widgets.

4 In the Build environment of a frame, create and fill a region with a work area object. Typically you don't add items to the work area in the Build environment because those items will not belong to the work area; they are simply siblings of it. Because the work area is scrollable, it can be larger than the frame.

5 In the Attributes window, assign the RESOURCE entry containing your widget subclasses.

6 Also specify a catalog entry in which to save the work area when the user leaves it. Typically the work area is stored as an SCL list in a catalog entry of type WORKAREA.

7 Close the Attributes window and run TESTAF to create and test the functionality of the work area items.

Drag and Drop and the Work Area

Drag-and-drop functionality, which has been added to the Widget class as a whole, is particularly useful for the Work Area class. Drag and drop allows you to define additional functionality for objects so that you can use a mouse to pick up an object from your work area and drag it over to another object on your work area and have an action occur when you drop it. For example, your work area might contain an item called View that runs the FSVIEW command when you drag a dataset (also a work area item) and drop it on that item.

The Work Area class automatically sets each item to be a drag site of type _afWORKAREA. The _afWORKAREA representation is an SCL list that contains the result of the _getProperties method (see the Widget class) and includes all instance variables and the region definition. It provides the work area with enough information to re-create the object if it is copied or moved to a different window. As a result, you can move or copy a work area item to a different work area in either the same frame or a different frame. In this case, the object is destroyed and re-created.

You can also move or copy a work area item within the same work area. This action uses a different work area representation named _afWORKAREA plus a unique numeric suffix. This representation allows the workarea to simply move the item instead of destroying and re-creating it.

For more information on drag and drop, see SAS/AF online help.

Example: Defining Drag and Drop by Subclassing

For an example that defines drag sites and drop sites using per-instance methods (instead of subclassing) see SAS/AF online help.

A desk is defined, generally, as a place to do work. Part of your desk may include file drawers, note pads, calculators, ledgers, etc. The arrangement and contents of the things on your desk are decided by you and not by anyone else. The Work Area Class provides an area where you can place any function you can think of, from commands to applications, from calendars to datasets.

... Or, you may have placed a Graphics Object to plot how the stock market is doing. You may have several stocks that you want to plot. You could drag the stock over the Graphic object and drop it to plot that stock. Also, just like the top of your desk, if you leave the frame containing the work area object and come back to it later, it will be in the same state as when you left it.
Example: Adding Highlighting to a Work Area

The following example shows how to add highlighting to a work area. Two methods are overridden in this example. The first is the _selectItem method. It is used to highlight the selected item and unhighlight any previously selected items. The second method is the _select method. This method is used to unhighlight any previously selected items.

```
length _self_ 8;

/* _selectItem method. Turns on */
/* outlines of all selected items */
/* Turns off outlines of previous */
/* selection */

selitem: method ids 8 action $ 8;
  l = makelist();
  /* Get the list of currently */
  /* selected items and turn off their */
```

button:
/* Create the main list and a */
/* sublist */
  l = makelist();
/* Create a sublist for class */
/* information */
l1 = makelist();
/* Fill the sublist with the class */
/* of the item*/
rc = setnitemcl(l1, "GTEXT.GTEXT",
    "_classname");
/* Add the class sublist to the main */
/* list */
rc = insertl(l, l1, 1);
/* Create a sublist for position */
/* information */
r = makelist();
/* Add the region sublist to the */
/* main list */
rc = setnitemrl(l1, r, "_region");
/* Specify the coordinates for the */
/* upper left corner of the item */
rval = setnitemdr(r, 1, "ULX");
rc = setnitemdr(r, 1, "ULY");
/* Does this do anything other */
/* than print list contents?? */
putlist(l);
/* Add the button to the work area */
/* WORK1 */
call notify("work1", "_newItem", l);
/* Delete the list */
dellist(l);
return;
```
/* current selection colors */
call send(_SELF_, "_getSelections", l);
    do i=1 to listlen(l);
        l1 = getiteml(l, i);
        call send(l1, "_setSelection", "OFF",
                        "BLACK");
    end;

    /* Let the sworkarea know about the */
    /* selected items */
call super(_SELF_, "_selectItem", ids,
                        action);

    /* Get the list of currently selected */
    /* items and turn on their */
    /* current selection colors */
call send(_SELF_, "_getSelections", l);
    do i=1 to listlen(l);   
        l1 = getiteml(l, i);
        call send(l1, "_setSelection", "ON",
                        "WHITE");
    end;
rc = dellist(l);
    if (rc ^= 0) then do;
        put "Error removing temporary list
during _selectItem";
        _msg="Error removing temporary list
during _selectItem";
    end;
endmethod;

    /* Select method to turn off outlines */
select: method;
    l = makelist();

        /* Send the select item method and */
        /* only have this item selected */
call send(_self_, "_selectItem", l,
                        "REPLACE");
rc = dellist(l);
        if (rc ^= 0) then do;
            put "Error removing temporary list
during _select";
            _msg="Error removing temporary list
during _select";
        end;
endmethod;

Work Area Run-time Menus
The user creates and modifies work area items at run time by selecting items from pop-up menus.
Work Area Class

Work Area Run-time Menus

Note: Objects placed on the work area in the Build environment are not work area items. As a result, run-time menus do not affect them, and you cannot perform actions such as growing, resizing, editing, deleting, or scrolling.

**Work Area Region Menu**

The pop-up menu in the work area region displays these choices:

- **Add Item**
  displays the list of widgets that can be added to the work area. This list is comprised of all Widget classes contained in the current resource associated with the work area. Selecting a widget adds it to the work area. This selection runs the _addItem method.

- **Modify Resource**
  allows you to edit the current resource entry. This selection runs the _editResource method.

- **Turn Growing On/Off**
  turns resize handles on and off for all regions in the work area. When growing is on, Add Item and Copy are grayed. This selection runs the _growMode method.

- **Auto Arrange**
  arranges the work area items so the middle of the upper edges are the distance apart specified in the Adjustment dialog box. Row- and column-based objects may not show text if the arrangement places the object in an invalid location. This selection runs the _arrange method.

- **Adjust Arrangement**
  displays the dialog box for adjusting the arrangement sizes. This selection runs the _adjustDialog method.

- **Disallow dragging**
- **Allow dragging**
  toggle between providing dragging or other functionality with a mouse button. When Disallow dragging is selected, the mouse button provides an alternate function (usually marking or pasting).

You can use the _popup method to add or delete items in this menu.

**Work Area Item Menu**

If you pop-up on a newly created item in the work area, the menu displays these choices:

- **Attributes**
  opens the attributes window for that item. This selection runs the _attributesDialog method.

- **Region Attributes**
  opens the Region Attributes window. This selection runs the _regionAttr method.

- **Remove**
  removes the item. This selection runs the _term method.
Copy

copies the item. This selection runs the _copyRegion method.

Move

moves the item. This selection runs the _moveRegion method.

Additional choices may be available depending on the class of the item. You can use the _childPopup method to add or delete items in this menu.

Methods

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

Dictionary

_addItem

Allows the user to add a new item to the work area by selecting a class from the current RESOURCE entry.

Syntax

CALL NOTIFY (workarea-name, '_addItem');

Details

_addItem displays a selection list similar to the one opened by the Make action in the Actions pop-up menu in the DISPLAY window. The Add Item selection of the run-time menu calls this method.

_adjustDialog

Allows the user to specify the amount of space between items in the work area.

Syntax

CALL NOTIFY (workarea-name, '_adjustDialog');
Details
_adjustDialog opens a dialog box in which the user enters distance in pixels that are used by the _arrange method. The Adjust Arrangement selection of the run-time menu calls this method.

__arrange

Arranges all items in the work area

Syntax
CALL NOTIFY (workarea-name, `_arrange');

Details
__arrange positions the items so the distance between the middle of the upper edges of each item is the amount specified by _adjustDialog. You can offset the initial position of the upper edge with the _arrangeOffsetY instance variable.

The Auto Arrange selection of the run-time pop-up calls this method.

If the work area boundary clips a widget outside the work area, such as an icon or CHECKBOX, the clipped widget is hidden.

__childPopup

Displays a pop-up menu containing choices supplied by the Work Area class or defined by the user. Overrides __childPopup in the Widget class.

Syntax
CALLSUPER(_SELF_, `__childPopup', fill-list, widget-id, sel);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fill-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list containing the pop-up menu items</td>
</tr>
<tr>
<td>widget-id</td>
<td>N</td>
<td>specifies the identifier of the widget that received the original _popup method</td>
</tr>
<tr>
<td>sel</td>
<td>N</td>
<td>returns the selected pop-up menu item</td>
</tr>
</tbody>
</table>

Details
The __childPopup method is run by the WIDGET class when the user causes the WPOPUP event (usually with a mouse button) on a work area item. The method has
the opportunity to modify the list of pop-up menu items passed in and then pass the fill-list on up the class hierarchy to be handled by the base Work Area class. After the CALL SUPER, sel is set to the index of the selected item from the pop-up menu. If that selection belongs to this method, the selection should be handled, the fill-list cleaned up and selection set to 0 before returning. It is important that the fill-list and sel returned from this method be consistent with the fill-list that was passed into the method. See the _popUp method for an example.

By default, _childPopup for a newly added item adds the following items to the menu:

- Attributes
- Region Attributes
- Remove
- Copy
- Move

For a description of these items, see “Work Area Run-time Menus” on page 1970.

__editResource__

Edits the RESOURCE entry associated with the work area

**Syntax**

CALL NOTIFY (workarea-name, '_editResource');

**Details**

The _editResource method opens the Resource Entry window in which you can modify the list of widget subclasses that can be added to the work area. If you edit an existing class, the new definitions are not used until the next invocation of the work area.

This method does nothing if no resource has been defined for the work area.

The Modify Resource selection of the run-time pop-up calls this method. This item is grayed if you have not specified a RESOURCE entry.

__getResource__

Returns the name of the current RESOURCE entry

**Syntax**

CALL NOTIFY (workarea-name, '_getResource', resource-name);
### _getWidgets

**Returns an SCL list containing all the currently displayed work area items**

**Syntax**

```
CALL NOTIFY (workarea-name, '_getWidgets', widget-list);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list that will contain descriptive information about all widgets in the work area</td>
</tr>
</tbody>
</table>

**Details**

The returned list contains a list of objects. You can use this list to send methods to individual widgets using the instance variable _SELF_.

---

### _getSelections

**Returns an SCL list containing all currently selected work area items**

**Syntax**

```
CALL NOTIFY (workarea-name, '_getSelections', selected-list);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selected-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list that will contain information about each currently selected widget in the work area</td>
</tr>
</tbody>
</table>

**Details**

The returned list contains data about the selected widgets and can be used to operate on the widgets as a group but not individually, unless you use the instance variable _SELF_ under each widget to send methods to that widget.

To access individual widgets, use the _getWidgets method.
Because this list contains all the widgets in the work area and because it is in the SLIST format (which is the same format as the WORKAREA entry type) it can be used to save the contents of the work area.

__growMode__

Specifies whether work area items are in grow mode, that is if they can be resized and moved.

**Syntax**

CALL NOTIFY (workarea-name, '_growMode', mode);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>C</td>
<td>specifies the status of grow mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'ON' allows growing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'OFF' does not allow growing</td>
</tr>
</tbody>
</table>

**Details**

_growMode_ allows the user to resize individual work area items at run time. While in grow mode, the work area items cannot be selected or copied, and new items cannot be added.

The Turn Growing On/Off selection of the run-time pop-up calls this method.

__hscroll__

Scrolls the work area horizontally.

**Syntax**

CALL NOTIFY (workarea-name, '_hscroll', unit, num-units);
<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>C</td>
<td>specifies the scrolling unit: <code>COLUMN</code>, <code>HALF</code>, <code>MAX</code>, <code>PAGE</code></td>
</tr>
<tr>
<td>num-units</td>
<td>N</td>
<td>specifies number of units to scroll. To scroll right, specify a positive number; to scroll left, specify a negative number. The default is 1.</td>
</tr>
</tbody>
</table>

**Details**

A page is the visible portion of the work area. MAX scrolls to the extent of the scroll bar.

---

**_issel**

Reports whether a specified work area item is currently selected

**Syntax**

```call notify```

(workarea-name, `_issel`, widget-id, status);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget-id</td>
<td>N</td>
<td>specifies the object identifier of a widget in the work area</td>
</tr>
<tr>
<td>status</td>
<td>C</td>
<td>returns a value indicating whether the item is currently selected: <code>Y</code> selected, <code>N</code> not selected</td>
</tr>
</tbody>
</table>

**Details**

Use the _getWidgets method in this class to return the list of widgets.

---

**_newItem**

Adds a new item to the work area

**Syntax**

```call notify```

(workarea-name, `_newItem`, item-list);
Argument | Type | Description
--- | --- | ---
item-list | N | specifies the identifier of an SCL list containing one or more sublists that define the new item. See the _new method for the Class class for more information on the format of this list.

Details

Item-list must contain a value for the _classname sublist item, which specifies the class of the item to be created. This value is the two-level name used by the work area's RESOURCE entry.

Optionally, item-list can contain the _region sublist, which provides values for the region position variables and places the item in the work area. This sublist can contain only one of the groups of items in the following table:

<table>
<thead>
<tr>
<th>If the list contains...</th>
<th>Then...</th>
</tr>
</thead>
<tbody>
<tr>
<td>coordinates for both the variables 'ULX' and 'ULY'</td>
<td>the work area item is placed at the specified position using the default size</td>
</tr>
<tr>
<td>coordinates for the four variables 'ULX', 'ULY', 'LRX', and 'LRY'</td>
<td>the work area item is placed at the position specified by ULX and ULY and is sized to LRX and LRY</td>
</tr>
<tr>
<td>no values for the region variables</td>
<td>the work area item is placed in the first available appropriate place. That is, the item is centered horizontally within a grid only if that grid slot is not occupied. The size of the grid is defined in the Attributes window or in the Adjust Arrangement dialog box.</td>
</tr>
</tbody>
</table>

Example

Create a frame with a work area. Use the following code to define a graphic text object and place it in the work area, WORK1. Use the _region list to position it in the upper left corner (1,1):

```
INIT:
/* Create an ITEM list. */
itemlst=makelist();

/* Create a sublist to contain the */
/* class information and attributes. */
classlst=makelist();
/**/
/* Add an item for _classname. */
classlst=setnitemc(classlst,'GTEXT.GTEXT',
                  '_classname_');

/* Insert attribute items for COLOR */
/* and TEXT, */
classlst=insertc(classlst,'yellow','-1,
                 'color');
classlst=insertc(classlst,'New GTEXT
Widget','-1,'TEXT');

/* insert the class list information */
/* into the item list */
itemlst=insert1(itemlst,classlst,1);

    /* Create a region sublist to contain */
    /* the positioning information */
    region=makelist();

    /* Insert the region list into the */
    /* class list */
    classlst=setniteml(classlst,region,'_region_');

    /* Add items to the region list to */
    /* position the widget */
    /* in the upper left corner of */
    /* the workarea */
    region=setnitemn(region,1,'ulx');
    region=setnitemn(region,1,'uly');

    /* Add the graphic text item list to */
    /* the work area */
    call notify('work1','_new_item_',itemlst);
    dellist(itemlst);
    return;

_popUp

Displays pop-up menu choices supplied by the Work Area class, or defined by the user. Overrides _popUp in the Widget class.

Syntax
CALLSUPER(_SELF_, '_popUp', fill-list, sel);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fill-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list containing the pop-up menu items</td>
</tr>
<tr>
<td>sel</td>
<td>N</td>
<td>returns the selected pop-up menu item</td>
</tr>
</tbody>
</table>

Details
The _popUp method runs when the user causes the WPOPUP event (usually with a mouse button). The method has the opportunity to modify the list of pop-up menu items passed in and then pass the fill-list on up the class hierarchy to be handled by the WIDGET class. After the CALL SUPER, sel is set to the index of the selected item from the pop-up menu. If that selection belongs to this method, the selection should be handled, the fill-list cleaned up and sel set to 0 before returning. It is important that the fill-list and sel returned from this method be consistent with the fill-list that was passed into the method.
By default, _popup for the work area adds the following items to the menu:
Add Item
Modify Resource
Turn Growing On/Off
Auto Arrange
Adjust Arrangement

These items are described in “Work Area Run-time Menus” on page 1970.

Example

This example adds three items to the pop-up menu. These items allow you to edit work area attributes, copy the work area, or remove it. It also passes the request on up for the Widget class to handle.

/* Override the _popup method. */
POPUP: Method plist 8 sel 8;
   /* Insert items on the front of the */
   /* pop-up menu list */
   rc = insertc(plist, "Edit", 1);
   rc = insertc(plist, "Copy", 2);
   rc = insertc(plist, "Remove", 3);
   call super(_self_, "_popup", plist, sel);

   /* Clean up the list so items are */
   /* not passed */
   /* to the child subclasses */
   rc = delitem(plist, 1);
   rc = delitem(plist, 1);
   rc = delitem(plist, 1);

   /* Test the value of SEL to see if */
   /* it is one of the 3 new items*/
   if (sel > 0) then do;
      if (sel <= 3) then do;
         /* If so, perform the requested */
         /* action and reset to 0 */
         if (sel = 1) then call send(_self_,
            ’_attributes_dialog_’);
         else if (sel = 2) then call send
            (_self_, ’_copy_region_’);
         else if (sel = 3) then call send
            (_self_, ’_delete_object_’);
         sel = 0;
      end;
   else
      /* Otherwise, since 3 items were */
      /* added to the front */
      /* of the list, subtract 3 from */
      /* the selection. */
      sel = sel-3;
   end;
endmethod;

_\_popWidgets

Replaces the list of current items with a previous list

Syntax

CALL NOTIFY (workarea-name, '_popWidgets');

Details

The _popWidgets method returns the work area to the last set of work area items that were saved using the _setWidgets method with the PUSH option.

_\_runSelections

Runs each selected work area item

Syntax

CALL NOTIFY (workarea-name, '_runSelections');

Details

The _runSelections method is run when the user double clicks on a work area item. The work area runs the _select method for each selected item with the _EVENT_ variable set to 'D'.

_\_selectItem

Adds one or more items to the list of selected work area items

Syntax

CALL NOTIFY (workarea-name, '_selectItem', selected-list, action);
<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selected-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list containing the work area items to be listed as currently selected. The list must contain object identifiers that are lists and not numbers. Each identifier needs to be named as well with the name of the object.</td>
</tr>
<tr>
<td>action</td>
<td>C</td>
<td>specifies how to handle the contents of selected-list:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'APPEND' adds item to list (caused by shift-click)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'REPLACE' replaces all selections in the list with this selections (caused by a single click)</td>
</tr>
</tbody>
</table>

**Details**

/selectItem allows users to select one item or multiple items. The method runs automatically when a work area item is selected. The APPEND and REPLACE values are used internally to maintain the list returned by _selectItem.

When you select a work area item, the _selectItem method runs, which in turn runs the _setSelection Widget method. When selected, the item is highlighted with the default border color. However, upon deselection, the work area background color is passed to the _setSelection method; as a result when the item is deselected, the background color of the item replaces the border color and the item appears to lose its border. To change this behavior, override the _setSelection method when you create the subclass.

**Example**

This example shows how to select all items within a work area. It can be provided with the frame containing the work area.

WORK1:

```plaintext
widgets = makelist();

/* Get all the items within the */
/* work area */
call notify('work1','_getWidgets'
 ,widgets);
/**/

/* Delete these items if they */
/* are on the list */
if nameditem(widgets,'ICONSIZEX') then
   rc = delnitem(widgets,'ICONSIZEX');
else if nameditem(widgets,'ICONSIZEY') then
   rc = delnitem(widgets,'ICONSIZEY');

/* Make a list of object identifiers. */
/* Note that _getWidgets */
/* does not return a list of */
/* identifiers, but is a list */
/* describing the items. _SELF_ */
/* is the identifier of that item. */
objids = makelist();
do i = 1 to listlen(widgets);
   rc = setniteml(objids,
      getniteml(getiteml
```
(widgets,i),'_SELF_'),
nameitem(widgets,i));

end;
call notify('work1','_selectItem'
,objs,'REPLACE');
rc = dellist(widgets);
rc = dellist(objids);
return;

See Also
_setSelection in the Widget class.

---

_setResource

Changes the RESOURCE entry used by the work area

Syntax

CALL NOTIFY (workarea-name, '_setResource', resource-name);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource-name</td>
<td>C</td>
<td>specifies the two- or four-level name of the resource entry. If the name specified in resource-name is invalid or does not exist, the work area uses the frame's resource.</td>
</tr>
</tbody>
</table>

Details

The RESOURCE entry is the source of all widget subclasses that can be added to the work area. The list of choices displayed by _addItem comes from the RESOURCE entry.

---

_setWidgets

Replaces the current work area items with new items

Syntax

CALL NOTIFY (workarea-name, '_setWidgets', widget-list<, 'PUSH'>);
**Argument** | **Type** | **Description**
--- | --- | ---
widget-list | N | Specifies the identifier of an SCL list containing descriptive information about the new work area items. The list is expected to be in the form returned by the _getWidgets method.

'PUSH' | C | Saves the current list of work area items so it can be restored by the _popWidgets method.

**Details**

The work area is saved as an SCL list in a catalog entry of type WORKAREA. Because this is the same format as _afWORKAREA, you can reload the saved work area with the _setWorkarea method. For more information about _afWORKAREA, see “Using the Work Area Class” on page 1967.

_setWidgets redefines the layout of the work area by replacing the existing list of work area items with the new list of items defined in widget-list. You can use this method to dynamically set the list of work area items that are displayed in the work area at run time.

You can pass the entire contents of a work area to the _setWidgets method by using the FILLIST function to create the list of work area items.

However, the list of information does not have to be complete. If the _classname list item is present in each item's sublist, a work area item is created and the defaults for the class replace any missing list items. (_classname is the two-level class name within the work area's resource.) If no region position information is passed in, an automatic placement is chosen.

**See Also**

_getWidgets in the Frame class.

---

**_setWorkarea**

Replaces the current work area with the contents of a different work area

**Syntax**

CALL NOTIFY (workarea-name, '_setWorkarea', new-workarea);

**Argument** | **Type** | **Description**
--- | --- | ---
new-workarea | C | Specifies the name of a WORKAREA entry with which to replace the current one

**Details**

The _setWorkarea method allows you to change the name of the work area entry currently used by the object. This entry holds the information currently contained within the work area. Note that the current contents of this entry replace the current Work Area contents.
_vscroll

Scrolls the work area vertically

Syntax

CALL NOTIFY (workarea-name, '_vscroll', unit, num-units);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>C</td>
<td>specifies the scrolling unit:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>’MAX’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>’PAGE’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>’HALF’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>’ROW’</td>
</tr>
</tbody>
</table>
| num-units  | N    | specifies the number of units to scroll. To scroll down, specify a positive number; to scroll up, specify a negative number.

_write

Saves the contents of the current work area

Syntax

CALL NOTIFY (workarea-name, '_write');

Details

_write saves the work area to the entry specified in the Save Location field of the Attributes window. This entry is a list that can be read in using the FILLIST function.