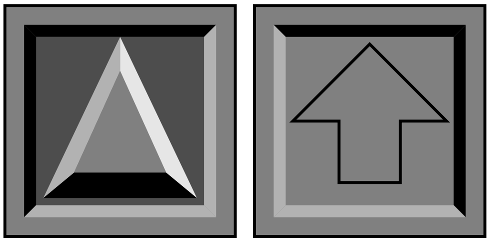
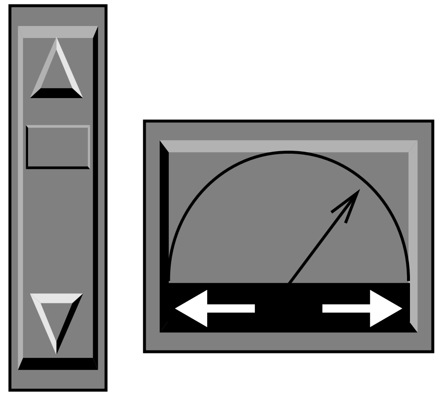
The SUIT Widget Library

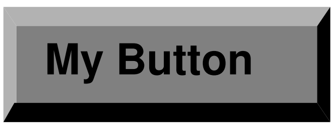
The SUIT Widgets at a Glance



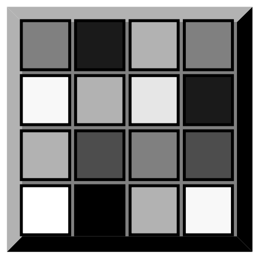
"Arrow Button" on page 114



"Bounded Value" on page 116

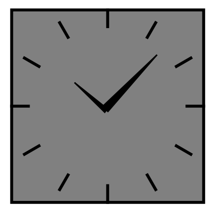


"Button Widget" on page 119

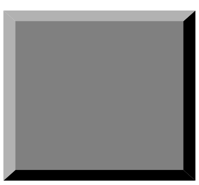


"Color Chips" on page 122

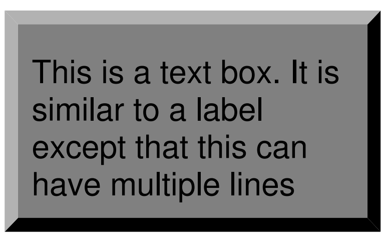
"Label" on page 129



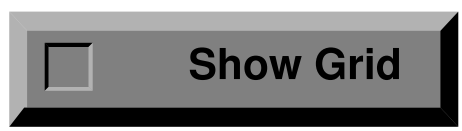
"Clock" on page 121



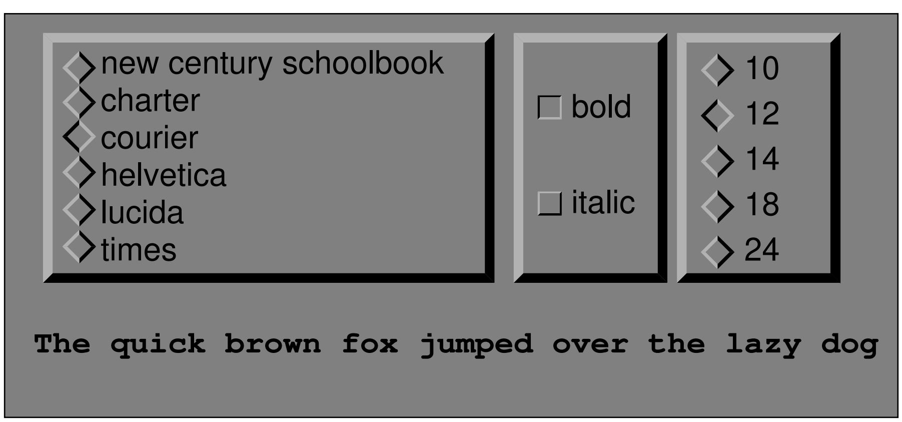
"Place Mat" on page 132



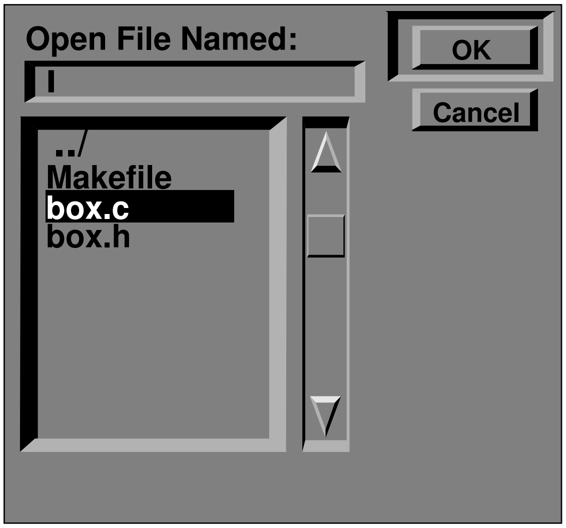
"Text Box" on page 141



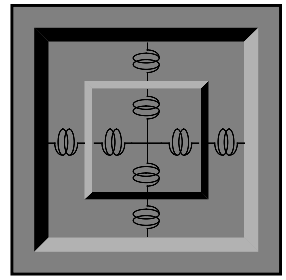
"On Off Switch" on page 130



"Font Panel" on page 128

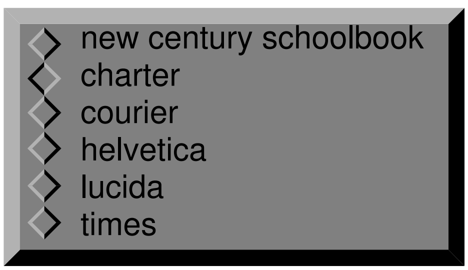


"File Browser" on page 127

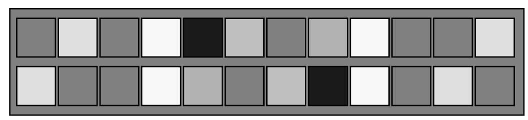


"Spring Panel" on page 139

"Trash Can" on page 145

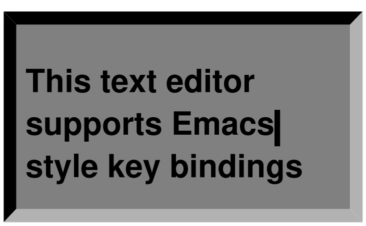


"Radio Buttons" on page 136

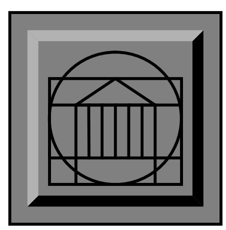


"Pattern Chips" on page 131

"Type In Box" on page 146

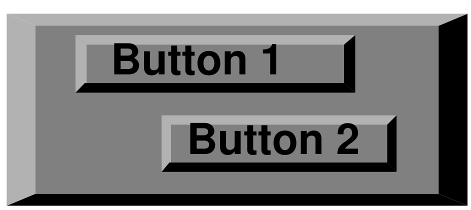


"Text Editor" on page 142



"UVA Logo" on page 148

"Scrollable List" on page 138



"Bulletin Board" on page 118

"Pulldown Menu" on page 134

Global Properties

These are the properties that all objects can inherit from the global level. Because GLOBAL level properties usually hold important, system wide information, GLOBAL properties cannot be destroyed interactively with the SUIT property editor.

Property Name	Notes
ANIMATED *boolean*	True when an object continually performs some animated function. For example, when **ANIMATED** is TRUE for the bouncing ball widget, the ball constantly moves around in its box.
BACKGROUND_COLOR *GP_color*	Default background color. For beveled display styles and GP shapes, this should be set to the same color as the border color. Also see: **FOREGROUND_COLOR** and **BORDER_COLOR**.
BORDER_COLOR *GP_color*	Default border color. For beveled display styles and GP shapes, this should be set to the same color as the background color. Also see: **FOREGROUND_COLOR** and **BORDER_COLOR**.
BORDER_RAISED *boolean*	Used in the motif border style. Setting this to FALSE makes a widget appear to be depressed into the screen, away from the viewer. Setting this to TRUE makes a widget appear to be raised out of the screen, toward the viewer. Also see: **BORDER_TYPE**, **BORDER_WIDTH**, **DRAW_BORDER_ON_INSIDE**, and **HAS_BORDER.**
BORDER_TYPE *SUIT_enum*	Three options are available: "simple", "motif" and "fancy motif." A simple border is just a rectangle drawn in the border color. Motif borders have a 3-D look about them. Fancy motif borders have the 3-D look with much more decoration added. They are usually used for SUIT Dialog Boxes which grab the input focus and thus the user's attention (hence the fanciness). Also see: **BORDER_RAISED**, **BORDER_WIDTH**, **DRAW_BORDER_ON_INSIDE**, and **HAS_BORDER**.
BORDER_WIDTH *integer*	The width of the border given in pixels. Also see: **BORDER_RAISED, BORDER_TYPE, DRAW_BORDER_ON_INSIDE**, and **HAS_BORDER**.
CAN_BE_OPENED *boolean*	Set to TRUE when the widget being examined is a pulldown menu or a bulletin board.
CLIP_TO_VIEWPORT *boolean*	When TRUE, a widget cannot draw graphics outside its own viewport. When FALSE, a widget can draw anywhere on the screen. This property is almost always TRUE. Also see: **VIEWPORT**.
DEFAULT_OBJECT_HEIGHT *integer*	Sets the height of widgets created interactively and widgets that have no **VIEWPORT** property specified(pixels).
DEFAULT_OBJECT_WIDTH *integer*	Sets the width of widgets created interactively and widgets that have no **VIEWPORT** property specified(pixels)
DRAW_BORDER_ON_INSIDE *boolean*	When **FALSE**, a widget's border is drawn to the outside of the widget's stated viewport. Also see: **BORDER_RAISED, BORDER_WIDTH, BORDER_TYPE**, and **HAS_BORDER**.

FONT *GP_font*	Default font that widgets will use if they do not specify a font at their class or object levels. The format of the string you see is: typeface, style, point size. For example 10 point bold italic Times would be written: times, bold italic, 10.
FOREGROUND_COLOR *GP_color*	The default foreground color. Widgets paint their graphics in the foreground color. For example, the color of the text on a button is drawn in **FOREGROUND_COLOR**, but the button itself is considered the **BACKGROUND_COLOR**. Also see: **BACKGROUND_COLOR** and **BORDER_COLOR**.
HAS_BACKGROUND *boolean*	When TRUE, SUIT will fill a widget's viewport by drawing a rectangle in the background color before calling that widget's paint procedure. When FALSE, SUIT will not do this. Most widgets leave this TRUE, except for those which are animated, or are performing optimized painting. Also see: **BACKGROUND_COLOR** and **ANIMATED**.
HAS_BORDER *boolean*	When TRUE, the object is drawn with a border. Also see: **BORDER_RAISED**, **BORDER_WIDTH**, **BORDER_TYPE**, and **DRAW_BORDER_ON_INSIDE**.
MARGIN *integer*	Denotes the number of pixels between the text and the widget border for those widgets that have text in them. This property applies to the top, bottom, left and right margins.
SCREEN_HEIGHT *integer*	Vertical resolution of the screen in pixels. When SUIT is running on a computer screen using a window manager, this refers to the height of the window.
SCREEN_WIDTH *integer*	Horizontal resolution of the monitor in pixels. When SUIT is running on a computer screen using a window manager, this refers to the width of the window.
SHOW_TEMPORARY_PROPERTIES *boolean*	Each SUIT property is either temporary or permanent. Temporary properties are not saved to the .sui file when the program exits, therefore any changes made to a temporary property will not be in effect the next time the program is invoked. The SUIT property editor normally displays only permanent properties, because most temporary properties are used for internal bookkeeping purposes and aren't very interesting. When **SHOW_TEMPORARY_PROPERTIES** is TRUE, the SUIT property editor shows all properties, and indicates which are temporary by using a different color to display them, or by showing "(temporary)" after their name on a monochrome display.
SHRINK_TO_FIT *boolean*	When TRUE, labels and buttons will resize themselves to be the size of the **LABEL** plus whatever **MARGIN** is set. Attempting to resize a widget that has **SHRINK_TO_FIT** set to TRUE will have no effect.
SPRINGINESS *SUIT_springiness*	If a **SUIT** application is running inside a window in a window managed environment, and the user resizes the entire window, should each button get proportionally larger? The **SPRINGINESS** property allows widgets to specify what they do if their parent (for most widgets, this just means the window or screen) changes in size. Conceptually, imagine that each widget is connected to the four walls of its parent by either springs or stiff rods - as the parent's walls move, the widget will stay the same distance from a wall if it is connected by a stiff rod. In the same fashion, widgets describe whether they get bigger by having either a stiff rod or a spring in each of the two possible directions, horizontal and vertical.

When you click on a springiness property, you interact with pictures of the springs and rods. Of course, in each of the directions (horizontal, vertical), there must be some springiness - if a widget was connected to its parents by a stiff rod on both its left and right sides, and had a stiff horizontal rod inside, it would be “overconstrained” if its parent got wider or narrower - that’s why you must always have at least one spring in each direction (vertical and horizontal). For more information, see the discussions of springiness in the heirarchy section of the reference manual.

SUIT_SYSTEM_FONT
GP_font

This is the font used by SUIT for the property editor and the SUIT menu.

VISIBLE
boolean

Denotes whether the widget is visible in the application. Widgets that have their **VISIBLE** property set to FALSE are not painted on the screen, and do not respond to mouse events. If you set **VISIBLE** to FALSE accidently and then exit the property editor, the only way to set the property back to TRUE is through program control or by hand editing the .sui file. Also see: **VISIBLE_WITHIN_PROPERTY_EDITOR**.

VISIBLE_WITHIN_PROPERTY_EDITOR
boolean

Denotes whether the widget is visible while being edited in the property editor. All the widgets except for the one being edited have their **VISIBLE_WITHIN_PROPERTY_EDITOR** set to FALSE. You should not need to worry about this property - it is used internally by SUIT.

WINDOW
SUIT_window

It is often inconvenient for widgets to draw in screen (pixel) coordinates, so most widgets use another coordinate system and they let the GP graphics package perform the calculations to figure out which pixels the graphics should end up on. The **WINDOW** property denotes the floating point coordinate system that a widget uses to display its graphics. It’s an arbitrary rectangle used as a convenience in the painting routine. For example, a widget that implemented football fields could draw in a coordinate system which was 100 yards by 40 yards. Most widgets just use from 0.0 to 1.0 in each direction. For more information, see the section on Windows and Viewports in the SUIT Reference Manual. Also see: **VIEWPORT**.

Common Properties

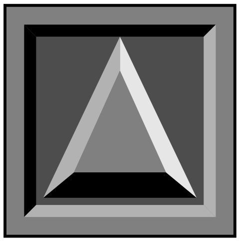
These are properties that are found at the object level for many widgets.

Property Name	Notes
ACTIVE_DISPLAY *SUIT_enum*	Each screen object implements an abstract idea, such as a clock. However, most abstract ideas can be displayed on the screen in many different ways. These are called display styles, or just "displays." A clock, for example, has two different displays: digital and analog. The **ACTIVE_DISPLAY** property tells SUIT which display style to use. By convention, if a widget only has one display style, that style is called "standard." You can change the display style for a widget without using the property editor: just type SUIT-C with the mouse over that widget to "cycle" to the next display.
CALLBACK_FUNCTION *SUIT_functionPointer*	The function to be called when the SUIT object is hit. This is a property that gets bound at runtime, but has no ASCII text representation. You should not attempt to change this temporary property interactively.
NUMBER_OF_CHILDREN *integer*	This property applies to bulletin board and pulldown menu widgets, and indicates how many different widgets are contained in the bulletin board or menu. All children of a widget must stay inside that widget on the screen. Widgets with children are sometimes called hierarchical widgets; they and all their children can be manipulated as a unit. hierarchical widgets can even have other hierarchical widgets as their children. This property should never be directly set by the user.
VIEWPORT *SUIT_viewport*	The physical location of the SUIT object on the screen. Changing this property is usually done interactively with SUIT-drag, but it is also possible to change a viewport by typing the four integers into the property editor directly. The four numbers correspond to lower-left x coordinate, lower-left y coordinate, upper-right x coordinate, upper-right y coordinate, also known as x1, y1, x2, y2. For more information, see the section on Windows and Viewports in the SUIT Reference Manual. Also see: **WINDOW**.

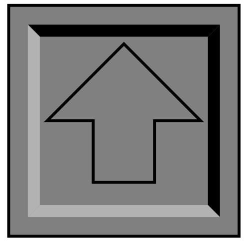
Arrow Button

Class: “arrow button”

Display Styles:



motif



simple arrow

`SUIT_object SUIT_createArrowButton(char *name, void callback (SUIT_object))`

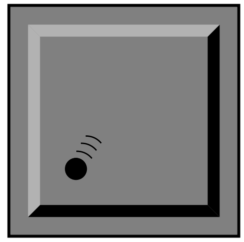
Description: Just like regular buttons, arrow buttons call functions when they are pressed. These buttons are provided as a convenience.

Property Name	Notes
DARKEN_BACKGROUND *boolean*	Determines whether the background of a motif arrow is drawn darker than the BACKGROUND_COLOR for this widget.
DIRECTION *SUIT_enum*	Sets direction of the arrow: up, down, left, or right.
INTERMEDIATE_FEEDBACK *boolean*	Determines whether the widget is currently displaying intermediate feedback -- a visual cue to let you know that the button is depressed but has not been released yet. Most users need not pay attention to this property.
SHADOW_THICKNESS *integer*	Sets the depth, in pixels, of the shadow for the motif arrow. Most users need not pay attention to this property.

Bouncing Ball

Class: “bouncing ball”

Display Styles:



standard

`SUIT_object SUIT_createBouncingBall(char *name)`

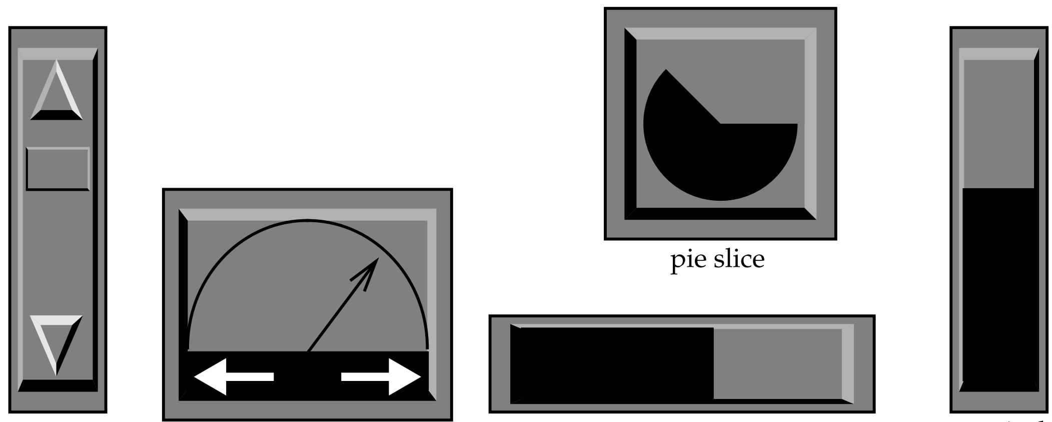
Description: This creates a ball that bounces around inside the box. Purely for demonstrating animation.

Property Name	Notes
BALL_SIZE *double*	The diameter of the ball expressed in world coordinates.
BALL_X *double*	The X position of the ball, in world coordinates.
BALL_Y *double*	The X position of the ball, in world coordinates.
FILLED *boolean*	When TRUE, the ball is filled in with the FOREGROUND_COLOR.
PIXELS_PER_SECOND *int*	The number of pixels per second (the speed) that the ball moves

Bounded Value

Class: “bounded value”

Display Styles:



SUIT_object SUIT_createBoundedValue (char *name, void callback (SUIT_object))

Description: Allows the user to select an integer or double between the minimum and maximum values specified. By setting the “granularity” property to 1.0, the current value of the widget can be constrained to integral values (i.e. doubles with no fractional part).

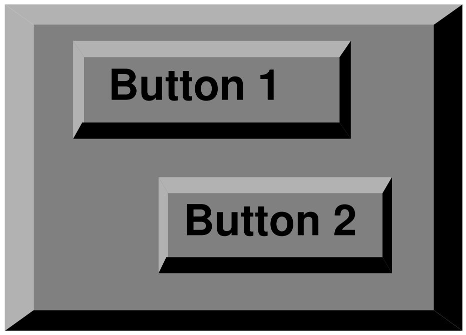
Property Name	Notes
ARROWHEAD_ANGLE *double*	The angle in degrees that the arrow head makes with the needle in the speedometer display style. A typical arrow would range from 10 to 50 degrees. 0 degrees would give no arrow at all.
ARROWHEAD_LENGTH *double*	Length of the arrowhead, expressed in world coordinates. Applies only to the speedometer display style. A typical arrowhead would range from 0.1 to 0.5 in length.
BUTTON_BACKGROUND_COLOR *GP_color*	Color of the bar on the bottom of the speedometer where the left/right arrows are shown.This only applies to the speedometer display style.
BUTTON_FOREGROUND_COLOR *GP_color*	Color of the left/right arrows shown below the speedometer. This only applies to the speedometer display style.
CURRENT_VALUE *double*	As shown by the widget, this is a floating point number between the minimum and maximum values.
GRANULARITY *double*	In the speedometer and scroll bar displays, this is the amount that the buttons will increment or decrement the current value of the bounded value. In all displays, the current value must be a multiple of the granularity - for example, setting a **GRANULARITY** of 1.0 effectively makes the bounded value an integer-only widget.

HAS_ARROW *boolean*	If TRUE, the speedometer needle ends in an arrow. Ignored in all other display styles.
HAS_TICK_MARKS *boolean*	If TRUE and if **GRANULARITY** is not zero, tick marks will be drawn on the sides of the thermometer display styles for the bounded value: one mark for each **GRANULARITY** step.
INCREASE_CLOCKWISE *boolean*	If TRUE, the pie slice display style increases in a clockwise fashion.
MAXIMUM_VALUE *double*	The highest possible value.
MINIMUM_VALUE *double*	The lowest possible value.
NEEDLE_COLOR *GP_color*	Color of speedometer needle and its arrowhead.

Bulletin Board

Class: “bulletin board”

Display Styles:



standard

For a complete understanding of this widget, read the section called "Hierarchy" on page 153.

SUIT_object SUIT_createBulletinBoard (char *name)

Description: Creates a container to hold other widgets.

See Also: A full explanation of bulletin boards begins with the section on Hierarchy on page 153. Functions that manipulate hierarchical widgets are described on page 72.

Example: To create a bulletin board that contains 2 buttons:

```
SUIT_object board, button;

board = SUIT_createBulletinBaord (“my board”);
button1 = SUIT_createButton (“my button”, callback1);
button2= SUIT_createButton (“another button”, callback2);

SUIT_addChildToObject (board, button1);
SUIT_addChildToObject (board, button2);
```

SUIT_object SUIT_createBulletinBoardWithClass (char *name, char *className)

Description: Creates a container to hold other widgets, but registers a different class name for the bulletin board so that it does not inherit the class level properties that govern all bulletin boards.

See Also: A full explanation of bulletin boards begins with the section on Hierarchy on page 153.

Example: If, in the above example, you replaced

```
SUIT_createBulletinBoard(“my board”);
```

with

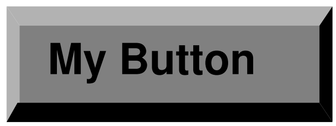
```
SUIT_createBulletinBaordWithClass(“my board” , “control panel”);
```

The widget would look class level properties up under the “control panel” class, not the “bulletin board” class, allowing control panels and bulletin boards to possess different properties.

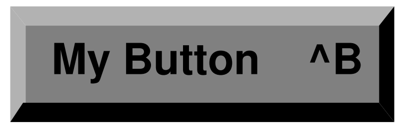
Button Widget

Class: “button”

Display Styles:



standard



button with hot key

Standard is a rectangular region.
Button with hot key shows the hot key (if any) that is associated with the button**.**

`SUIT_object SUIT_createButton (char *name, void callback (SUIT_object))`

Description: Creates a button that invokes the callback function when the button is pressed.

Parameters: **`name:`** By default, the text on the front of the button is the same as its name, but you can change that by setting the “label” property of the button to something else.

Property Name	Notes
DISABLED *boolean*	If **TRUE**, the text for the button is drawn in the **DISABLED COLOR** (rather than **FOREGROUND COLOR**) and the callback is not called when the button is pressed. This is useful for times in an application when certain functions are not valid - for example, a drawing editor might disable the “delete current object” button whenever there was no currently selected object.
DISABLED_COLOR *GP_color*	Sets the color of the button if the button is deemed to be an “inactive choice” in the application. See the **DISABLED** property at the object level.
HOTKEY *text*	The keyboard combination that can also be used to activate the button. Note that this is only a piece of text that appears alongside the LABEL. In order to actually attach the hotkey to the button requires registering an input trapper function.
LABEL *text*	The name that appears on the front of the button.

Buttons (Abort and Done)

SUIT_object SUIT_createAbortButton (SUIT_callbackFunctionPtr callback)

Description: Creates a button whose label says "abort" and whose callback calls **SUIT_done(ABORT)**, which leaves the application without saving the .sui file. If your application requires that some cleanup be done before SUIT exits, you can pass it a pointer to a function that will get called before the program ends. If you have no such function, you can pass NULL.

Example:

```
void MyCleanupFunction(SUIT_object me){
     /* "me" is the Abort button object. */
     }

obj = SUIT_createAbortButton(MyCleanupFunction);
```

See Also: **SUIT_done()** page 50

SUIT_object SUIT_createDoneButton (SUIT_callbackFunctionPtr callback)

Description: Creates a button whose label says "done" and whose callback calls **SUIT_done(NORMAL_EXIT)**, which leaves the application after saving the.sui file. If your application requires that some cleanup be done before SUIT exits, you can pass it a pointer to a function that will get called before the program ends. If you have no such function, you can pass NULL.

Example:

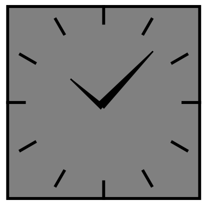
```
void MyCleanupFunction(SUIT_object me) {
     /* me is the Done button object. */
     }

obj = SUIT_createDoneButton(MyCleanupFunction);
```

See Also: **SUIT_done()** page 50

Clock

Class: “clock”

Display Styles:



analog

digital

`SUIT_object SUIT_createClock (char *name)`

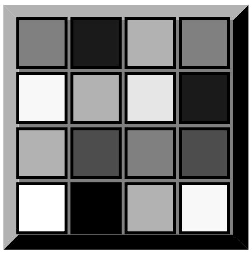
Description: This creates a widget that displays the system time from the computer’s clock.

Property Name	Notes
HAS_RIM *boolean*	Denotes whether or not the analog clock has a circular rim around it. Has no effect on the digital display style of the clock.
HAS_SECOND_HAND *boolean*	Denotes whether or not the analog display of the clock displays a second hand. This property can be toggled with the mouse. In the digital display style, if **HAS_SECOND_HAND** is TRUE, then the seconds are updated every second, otherwise they remain at zero.
MILITARY_TIME *boolean*	When TRUE, the digital display shows the time using a 24 hour clock. Unused in the analog display.

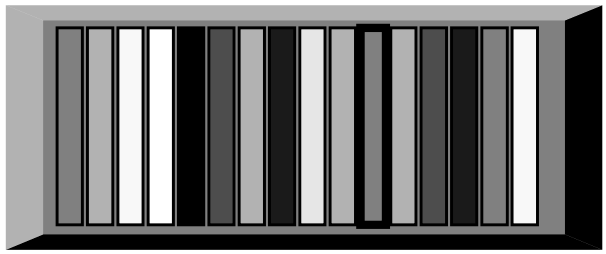
Color Chips

Class: “color chips”

Display Styles:



parquet



standard

`SUIT_object SUIT_createColorChips (char *name, void callback (SUIT_object))`

Description: Allows the user to select a color from the available palette of colors.

Property Name	Notes
CHIP_BORDER *double*	The current color is shown by drawing a highlighted border around its chip. The **CHIP BORDER** property controls the width of that border, expressed as a fraction of the width of the entire set of color chips. Typical range for this property is (0.05-0.1).
CURRENT_VALUE *GP_color*	The color currently selected.
HIGHLIGHT_COLOR *GP_color*	The color of the border used to highlight of the **CURRENT_VALUE** color.
PREVIOUS_VALUE *GP_color*	The color previously selected.

Dialog Boxes

Class: “dialog box”

All SUIT dialog boxes are “moded” meaning that once activated, they must be answered (by pressing an OK or Cancel button, for example) before you can interact with the rest of your application. For code examples using dialog boxes, see the DialogBoxes program in the SUIT examples directory.

IMPORTANT: Dialog Boxes return values of type Reply. (See page 35). Code that depends on a particular numeric value being returned from the dialog boxes may not work as expected.

Example:

WRONG:

```
if (SUIT_askYesNo(“are you there?”) )
....
```

RIGHT:

```
if (SUIT_askYesNo(“are you there?”) == REPLY_YES)
....
```

`void SUIT_inform (char *info)`

Description: Creates and activates a dialog box that contains a text string and a single button: OK. Like all dialog boxes, the **`SUIT_inform()`** box must be answered before the user can interact with the rest of the application.

Example: **`SUIT_inform ("Mail Sent");`**

`Reply SUIT_ask (char *question, char *button1Name, char *button2Name)`

Description: Asks a question and places two buttons on the screen. The buttons that appear are named by the two buttonName parameters.

Returns: **`REPLY_BUTTON1`**
`REPLY_BUTTON2`

Example:

```
if (SUIT_ask("Which units?", "Meters", "Feet") == REPLY_BUTTON1)
     printf ("user seleced Meters");
else
     printf ("user selected feet");
```

SUIT_object SUIT_createOKCancelDialogBox(char* name, SUIT_object innards, SUIT_validationFunction dataOK)

Description: Creates a dialog box that contains a single SUIT widget and offers two responses: OK and CANCEL. The widget can be a bulletin board that itself contains many widgets. This function does not actually place the dialog box on the screen, that is done with the function called **SUIT_activateDialogBox()**.

Parameters: **name** This is the name for this widget.
innards This is the SUIT_object that the dialog box is to wrap around.
dataOK This is a function that returns a boolean that indicates whether the data specified in the dialog box is valid or not and is called when the user selects the dialog box's OK button. The validation function takes a single SUIT_object as a parameter which will be the the object that the dialog box is currently wrapping. If your validation function returns TRUE, this means that the data from the wrapped widget is valid and **SUIT_activateDialogBox()** returns REPLY_OK. If the validation function returns FALSE, this means that there is something wrong with the data that the user specified, in which case, **SUIT_activateDialogBox()** does not return at all; the dialog box stays up, waiting for the user to continue interacting with the widget in the dialog box. It is not uncommon for the validation function to use **SUIT_inform()** to notify the user that the data was invalid, though this is not required.

Returns: A SUIT_object that represents the dialog box. Use it as the argument to **SUIT_activateDialogBox()**.

See Also: **SUIT_activateDialogBox()** page 124.
Example program for dialog boxes included in the SUIT distribution.

Reply SUIT_activateDialogBox (SUIT_object dbox)

Description: This function causes the dialog box created with **SUIT_createOKCancelDialogBox()** to be placed on the screen. The dialog box waits for input and exits with either **REPLY_OK** or **REPLY_CANCEL**.

See Also: **SUIT_createOKCancelDialogBox()** page 124.
Example program for dialog boxes included in the SUIT distribution.

Reply SUIT_askWithCancel(char *question, char *button1Name, char *button2Name)

Description: Asks a question and places three buttons on the screen. The first two buttons that appear are named by the two buttonName parameters. The last button is a cancel button.

Returns: **REPLY_BUTTON1**
REPLY_BUTTON2
REPLY_CANCEL

Example:

```
switch (SUIT_askWithCancel("Save changes before exiting?",
                           "Save", "Don't Save")) {
 case REPLY_BUTTON1:
     SaveTheChanges();
     break;
 case REPLY_BUTTON2:
     ExitWithoutSaving();
     break;
 case REPLY_CANCEL:
     /* Do nothing, user returns to application */
     break;
}
```

Reply SUIT_askOKCancel(char *question)

Description: Brings forth a dialog box that asks a question and offers two responses: OK and CANCEL.

Returns: **REPLY_OK**
REPLY_CANCEL

Example:

```
if (SUIT_askOKCancel ("About to transmit large file.") == REPLY_OK)
     printf("User selected OK");
else
     printf("User selected CANCEL");
```

Reply SUIT_askYesNo(char *question)

Description: Brings forth a dialog box that asks a question and offers two responses: YES and NO.

Returns: **REPLY_YES**
REPLY_NO

Example:

```
if (SUIT_askYesNo ("Are you sure?") == REPLY_YES)
     printf ("User selected YES");
else
     printf ("User selected NO");
```

Reply SUIT_askYesNoCancel (char *question)

Description: Brings forth a dialog box that asks a question with three responses: YES, NO and CANCEL

Returns: **REPLY_YES**
REPLY_NO
REPLY_CANCEL

Example:

```
switch (SUIT_askYesNoCancel("Save changes before exiting?")) {
 case REPLY_YES:
     SaveTheChanges();
     break;
 case REPLY_NO:
     ExitWithoutSaving();
     break;
 case REPLY_CANCEL:
     /* Do nothing, user returns to application */
     break;
}
```

```
Reply SUIT_getString(char *message, char *defaultString,
                        char answer[], int answerLength)
```

Description: Brings forth a dialog box with a type in box, an OK button and a CANCEL button. The string that the user typed comes back in the parameter called **`answer[]`**, which must be an allocated array of charcters of length **`answerLength`**. The **`message`** parameter is the prompt for the user and the **`defaultString`** is a character constant that appears in the type in box when the box first appears.

Returns: **`REPLY_OK`**
`REPLY_CANCEL`

Example:

```
char answer[50];

if (SUIT_getString("What is your name?",
                   "Bob the Amazing", answer, 50) == REPLY_OK){
     printf ("hello, %s\n", answer);
}
```

```
SUIT_object SUIT_createFileBrowserDialogBox (char *name, char *startDir,
                        char *label, char *typeInBoxLabel,
                        void callback(SUIT_object) )
```

Description: Brings forth a dialog box with a file browser widget, an OK button and a CANCEL button in it. As with the general **`SUIT_createOKCancelDialogBox()`**, to cause the dialog box to be visible to the user, use **`SUIT_activateDialogBox()`**. In general, it is usually easier to call **`SUIT_askForFileName()`**, which handles all the creation/activation/destruction operations, and just produces a string value for a file name as a result. Use **`SUIT_createFileBrowserDialogBox()`** call when you want more control over the dialog box (position, color, font, etc.).

NOTE: To specify the "current directory", use a NULL srting or "", not the sting "."

See Also: **`SUIT_creaateFileBrowser()`** on page 127.

```
char *SUIT_askForFileName (char *startDirectory, char *label,
                              char *typeInBoxLabel)
```

Description: Brings forth a dialog box with a file browser widget, an OK button and a CANCEL button in it in it. The value returned is the name of the file that the user selected, or NULL if the user pressed CANCEL.

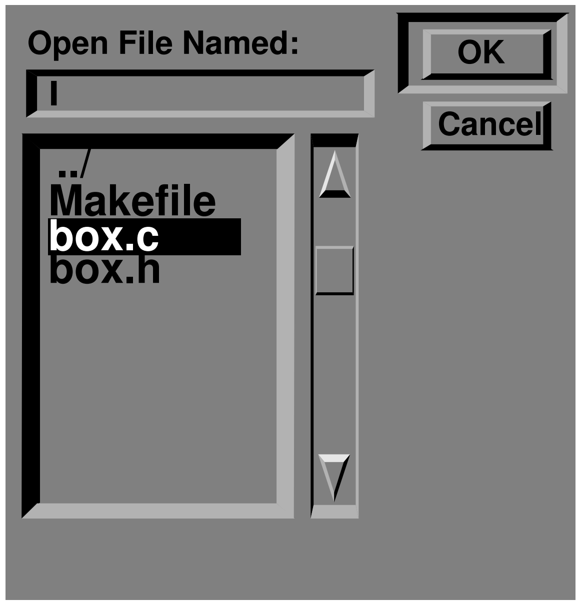
NOTE: To specify the "current directory", use a NULL srting or "", not the sting "."

See Also: **`SUIT_creaateFileBrowser()`** on page 127.

File Browser

Class: “file browser”

Display Styles:




```
SUIT_object SUIT_createFileBrowser (char *name, char *startDirectory,
                                    char *label, char *typeInBoxLabel,
                                    SUIT_callbackFunctionPointer callback))
```

Description: Creates a widget that displays the files in a given directory, starting with the given **startDirectory** . This currently only works on UNIX platforms.

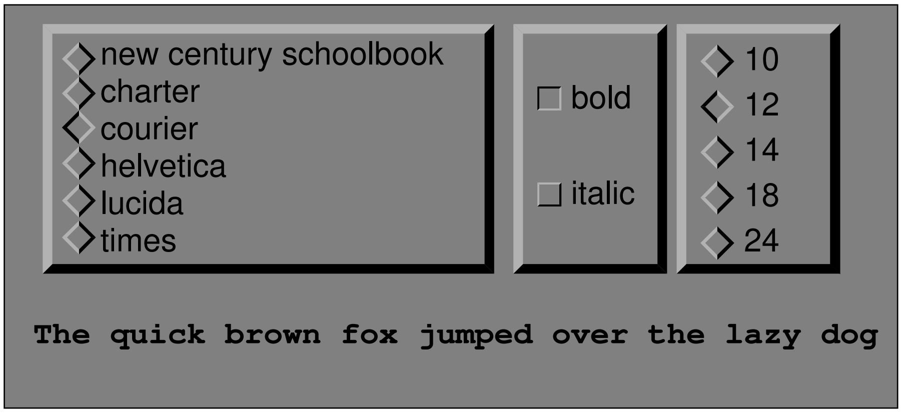
NOTE: To specify the "current directory", use a NULL srting or "", not the sting "."

Property Name	Notes
CURRENT_DIRECTORY *string*	The name of the directory (without the final filename appended) that the browser is looking at
CURRENT_FILE *string*	The name of the file (without the pathname prepended) that the browser is looking at.
CURRENT_VALUE *string*	The name of the file or directory that appears in the type-in box. This is the combination of CURRENT_DIRECTORY and CURRENT_FILE property values.
FILE_FILTER *string*	A wildcard specification (e.g. *.c or *.sui) that limits the kins of files that the browser will list. Example: if FILE_FILTER is set to *.c, the browser will only list files that end in ".c".

Font Panel

Class: “font panel”

Display Styles:



standard

`SUIT_object SUIT_createFontPanel (char *name)`

Description: This widget allows the user to select one of the standard GP fonts. The CURRENT_VALUE property of this widget is the font that is currently displayed on the panel.

Property Name	Notes
CURRENT_VALUE *GP_font*	This is the font that is currently selected on the font panel.

Label

Class: "label"

Display Styles:

standard

`SUIT_object SUIT_createLabel (char *name)`

Description: This call creates a widget that displays a text string. It has no callback. The default label is its name, though you can change that by setting the **LABEL** property. By default, labels are set to shrink to fit and to have no border, but these properties may be changed through the **SHRINK_TO_FIT** and **HAS_BORDER** properties, respectively.

Property Name	Notes
HAS_BORDER *boolean*	By default set to FALSE, meaning that it would have no border.
LABEL *text*	The string that appears on the face of the label.

On Off Switch

Class “on/off switch”

Display Styles:

motif square

check box

motif diamond

dot in circle

`SUIT_object SUIT_createOnOffSwitch (char *name, void callback (SUIT_object))`

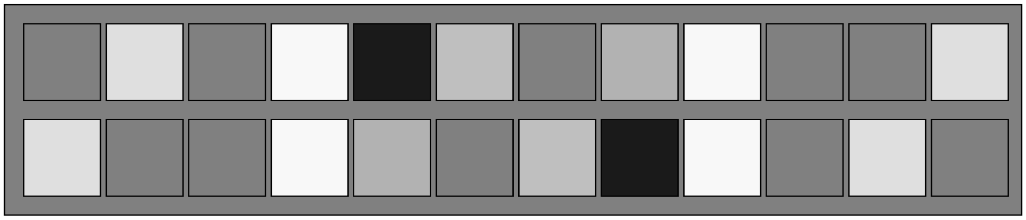
Description: Allows the user to select a boolean state.

Property Name	Notes
CURRENT_VALUE *boolean*	Set to yes if switch is on, no if the switch is off.
DISABLED *boolean*	If TRUE, the text for the button is drawn in the **DISABLED COLOR** (rather than **FOREGROUND COLOR**) and the callback is not called when the button is pressed. This is useful for times in an application when certain functions are not valid - for example, a drawing editor might disable the “delete current object” button whenever there was no currently selected object.
DISABLED_COLOR *GP_color*	The color of the text when the button is disabled (see object level property **DISABLED**).
LABEL *text*	The text to display on the face of the button.

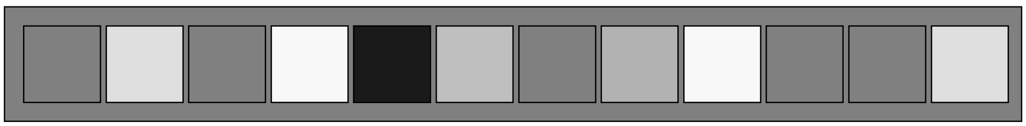
Pattern Chips

Class: "pattern chips"

Display Styles:



parquet



standard

The picture here is schematic. The real pattern chips show 40 different patterns.

`SUIT_object SUIT_createPatternChips(char *name, void callback (SUIT_object))`

Description: Allows the user to select one of the standard GP patterns.

Property Name	Notes
CHIP_BORDER *double*	The width of the border around each chip.
CURRENT_VALUE *integer*	Pattern currently selected.
HIGHLIGHT_COLOR *GP_color*	The color of the highlight around the selected pattern chip.
PREVIOUS_VALUE *integer*	Pattern previously selected.

Place Mat

Class: “place mat”

standard

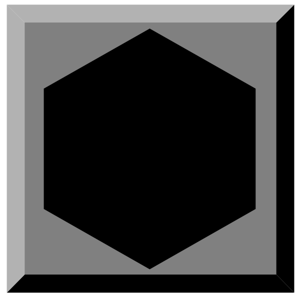
`SUIT_object SUIT_createPlaceMat (char *name)`

Description: This creates a do nothing square on the screen. It has neither a paint procedure nor a hit procedure, making it useful only for decoration. Commonly placed behind other widgets to create a particular visual effect.

Polygon

Class: “polygon”

Display Styles:



standard

`SUIT_object SUIT_createPolygon(char *name)`

Description: This is the famous polygon widget that appears in the SUIT tutorial.

Property Name	Notes
FILLED *boolean*	When TRUE, the polygon will be filled solid in the **FOREGROUND_COLOR**. When FALSE, only the outline is drawn.
NUMBER_OF_SIDES *integer*	The number of sides for the polygon. Setting this property to be a number less than 3 will draw a triangle.

Pulldown Menu

Class: "pulldown menu"

Display Styles:

standard

`SUIT_object SUIT_createPullDownMenu (char *name)`

Description: This creates a menu with a button that can be pressed, revealing choices underneath.

Example: See the example file menu.c

Property Name	Notes
LABEL *text*	This is the label that appers on the front of the pulldown menu button.

Menu Functions

`SUIT_object SUIT_createVerticalMenu (char *name)`

Description: This creates a placard style menu. (Like a pulldown menu that is always down.)

`SUIT_object SUIT_createMenuBar (char *name)`

Description: This creates a menu bar that can hold other menu buttons. This is just a horizontal stacker.

Example: See the example file menu.c

`SUIT_object SUIT_addToMenu(SUIT_object obj, char *buttonName, void callback (SUIT_object))`

Description: This adds a menu item to a menu.

Parameters: **`buttonName`**:This is the name of the button. Because this is a widget name, this string must be unique among all widget names in your application.

`callback` This is the function that you want called when this choice of the menu is used. The SUIT_object passed in as a parameter is the menu that was used.

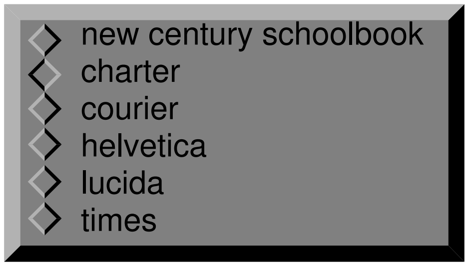
`SUIT_object SUIT_addToMenuWithHotKey (SUIT_object obj, char *button Name, void callback (SUIT_object), char *hotkey)`

Description: This adds a menu item to a menu and adds a hot key designation for the label. You will need to register a trapper function to make the hotkey active (See page 70).

Radio Buttons

Class: “radio buttons”

Display Styles:




```
SUIT_object SUIT_createRadioButtons (char *name,
                                     void (*callback)(SUIT_object))
```

Description: Allows the user to select one (and only one) item from a list of many choices in a mutually exclusive fashion.

Parameters: **`callback:`** A callback to the radio button that SUIT will execute when any of the radio buttons are pressed.

See Also: Radio Button Utility Functions on page 137.

Examples: See the example programs radio1.c and radio2.c

Property Name	Notes
CURRENT_VALUE *SUIT_enum*	This is the item that is currently selected. The string representation of the current SUIT enumeration is taken to be the label for each of the radio buttons.

Radio Button Utility Functions

void SUIT_addButtonToRadioButtons (SUIT_object radio, char *buttonName)

Description: Allows the programmer to add a button choice to a set of radio buttons. The button choice is added to the bottom of the list of choices already present in the set of radio buttons (if any).

Parameters: **radio:** A set of radio buttons created in a call to **SUIT_createRadioButtons().**

name: The name of the button to add. The name must be unique among the names of all SUIT objects in that SUIT will use this name to create a button widget for you.

Example:

```
/* Using the radio buttons */

void ChooseRadioButton(SUIT_object obj) {
printf ("button pressed is %s\n",
                   SUIT_getEnumString(obj,CURRENT_VALUE));
}

void MakeRadioButtons(void) {
     SUIT_object radio = SUIT_createRadioButtons("my buttons",
            ChooseRadioButton);
     SUIT_addButtonToRadioButtons (radio, "Underdog");
     SUIT_addButtonToRadioButtons (radio, "Bullwinkle");
     SUIT_addButtonToRadioButtons (radio, "Ren and Stimpy");
}
```

void SUIT_pressThisRadioButton (SUIT_object radio_buttons, char *buttonName)

Description: Allows the programmer to select one of the choices on the set of radio buttons. Note that this does NOT call the radio button callback, it merely changes the CURRENT_VALUE and changes the display to match.

Example:

```
/* select the Fred radio button choice under program control */

SUIT_object radio;

radio = SUIT_createRadioButtons("Flintstones", NULL);
SUIT_addButtonToRadioButtons(radio, "Fred");
SUIT_addButtonToRadioButtons(radio, "Barney");
SUIT_addButtonToRadioButtons(radio, "Wilma");

SUIT_pressThisRadioButton(radio, "Fred");
```

Scrollable List

Class: “scrollable list”

Display Styles:

standard d

```
SUIT_object SUIT_createScrollableList(char *name,
                                       void(*callback)(SUIT_object))
```

Description: Allows the user to select one string from a scrolling list of strings.

Parameters: **name**: The name of the scrollable list widget.

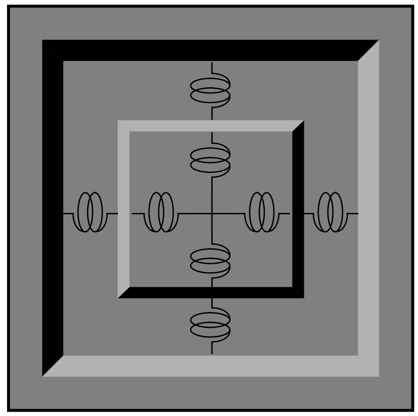
callback: A function to call when a string is selected.

Property Name	Notes
CURRENT_VALUE *text*	The currently selected string. Initially, this is the empty string.
CURRENT_ROW *integer*	The integer number of the currently selected string. The first string is string 0. If nothing is highlighted, the **CURRENT_ROW** is -1.
HAS_BACKGROUND *boolean*	When FALSE, the background can be seen between the text list and the scrollbar.
HAS_BORDER *boolean*	When FALSE, the scrollbar appears to be separate from the text list.
LABEL *text*	If not the empty string "", the scrollable list will display this **LABEL** string at the top of the list. For example, you could set this property to be the string “colors” for the list of items: “red”, “green”, and “blue”.
LIST *SUIT_textList*	This is the entire list of items in the scrollable list.

Spring Panel

Class: "spring panel"

Display Styles:



standard

`SUIT_object SUIT_createSpringPanel(char *name)`

Description: This widget allows the user to control the SUIT_springiness data type. SUIT_springiness is used to control the way a widget behaves when the widget's parent resizes (e.g. resizing the application window or a bulletin board widget).

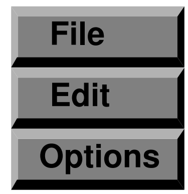
See also: There is a discussion of springiness on page 156 of the reference manual.

Stacker

Class: “stacker”

Display Styles:

horizontal stacking



vertical stacking

`SUIT_object SUIT_createStacker (char *name);`

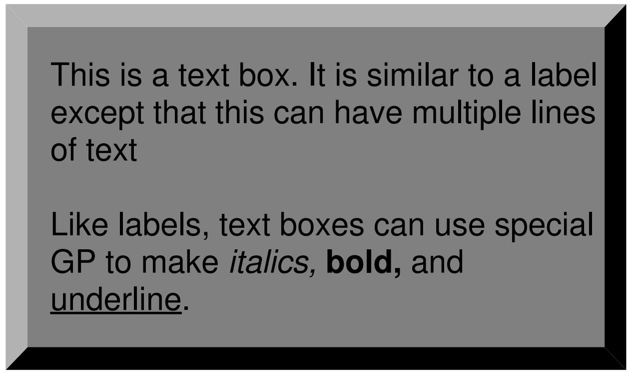
Description: This widget, like the bulletin board, is meant to hold other widgets. Stackers arrange their children in horizontal or vertical stacks, depending on the display style. The hieght of a stacker in the "horizontal stacking" display style is the height of the tallest child widget. Similarly, the width of a vertical stacker is that of its widest child. You can add children to stacker widgets as you would a bulletin board with **`SUIT_addChildToObject()`** or interactively, by opening up the stacker (SUIT-o) and SUIT-dragging the new children into the stacker. With vertical stackers,. new children are added to the stacker above the object you drop the new child on, and with horizontal stackers, new children are added to the right. Children maintain their size and shape as they are added to the stacker; the stacker changes size to accomodate them.

Property Name	Notes

Text Box

Class: "text box"

Display Styles:



standard

`SUIT_object SUIT_createTextBox(char *name, char *textToDisplay)`

Description: This is essentially a multiline label that performs wrapping of the text (no hyphenation, sorry) automatically, depending on the size of the viewport. The text box can use GP's special text attribute notation to create text boxes with underlines, bold, italics and special characters like ligatures and accents. For more information on this notation, see page 104.

Example:

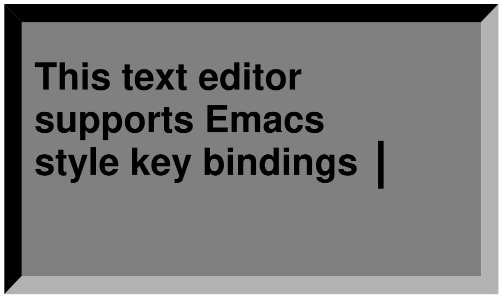
```
char *Notes = "This is a text box. It is similar to a label except that this can have multiple lines of text.\n Like labels, text boxes can use special GP to make @italic(italics), @bold(bold), and @under-line(underline).";

SUIT_createTextBox("my notes", Notes);
```

Property Name	Notes
LABEL *text*	The text that appears on the face of the Text Box. This string is allowed to contain imbedded newlines.
LINE_SPACING *double*	This number determines the space between lines of text, measured in text heights. 2.0 is double spaced.
NUMBER_OF_LINES *integer*	This number represents the number of lines of text currently being displayed. This number changes if text is added, or if the widget resizes and lines wrap differently.
STARTING_LINE *integer*	Usually, this number is zero, meaning that the text being displayed starts at the first line of the total amount of text held by this text box (the **LABEL** property). By setting **STARTING_LINE** to some number greater than zero, the text can be made to scroll off the top of the text box.

Text Editor

Class: “text editor”

Displays:



standard

`SUIT_object SUIT_createTextEditor(char *name, void callback (SUIT_object))`

Description: Creates an emacs-line text editor. The text that appears in the editor is a single string, which may contain imbedded newlines.

See Also: **`SUIT_textOfFile()`** on page 142.

```
SUIT_object SUIT_createTextEditorWithScrollBar(char *name,
                        void callback(SUIT_object))
```

Description: Creates a bulletin board which contains both an emacs-line text editor and an associated scrollbar. The text that appears in the editor is a single string, which may contain imbedded newlines. The object returned is the text editor, not the bulletin board.

`void SUIT_sendToEditor(SUIT_object o, char *command)`

Description: This sends a stream of characters to the text editor as if they were typed into the text editor directly. The command string may contain any keybinding (see page 144) defined for the text editor. Strings not recognized as commands are entered into the CURRENT_VALUE buffer exactly as they appear in the given command string.

`void SUIT_highlightBlockInTextEditor(SUIT_object o, int beginPos, int endPos)`

Description: This function highlights the section of the text between the beginPos and the endPos. First position in the text editor is position 0.

`char *SUIT_textOfFile (char *filename)`

Description: Provided as a convenience. Opens and returns, as a single string, the contents of a file. Can be used to set the **CURRENT_VALUE** property of a Text Editor widget.

Property Name	Notes
ALTERED *boolean*	FALSE until the text is edited.
ANY_KEYSTROKE_TRIGGERS *boolean*	When TRUE, the callback is called at every keystroke. When FALSE, the callback is only invoked at the DONE_EDITING_KEY key sequence
CURRENT_VALUE *text*	The text currently being edited. This string is allowed to contain newlines, tabs, but cannot use the GP_text special characters; all characters in the text editor will be the same font.
CURSOR_COLOR *GP_color*	The color of the cursor, which defaults to black.
CURSOR_INDEX *integer*	This temporary property denotes which character the cursor is in front of. For example, when CURSOR_INDEX is 0, this means that the cursor is in front of the first character.
CURSOR_STYLE *SUIT_enum*	This denotes the style of the cursor, either "vertical bar" or "i-beam."
CUT_BUFFER *text*	Contains the last text cut from the CURRENT_VALUE string. This buffer is set whenever the WIPE_BLOCK_KEY or KILL_LINE_KEY key sequence is used (CUT_BUFFER is set to be the killed text) or when the user highlights the text by dragging over the text editor with the mouse (CUT_BUFFER is set to be the text in the highlighted region).
HIGHLIGHT_BLOCK *boolean*	When true, the text that appears between the MARK_INDEX and the MARK_END_INDEX appears in reverse video (light on dark).
INPUT_SEQUENCE *text*	This buffer stores the editor commands before the editor processes them. After the editor recognizes a command, this property is reinitialized to the null string. If you register an interest in this property, you can examine the keyboard command before the editor responds to it. See also the SUIT function called **`SUIT_sendToEditor()`**.
MARK_INDEX *integer*	This number denotes the beginning position of the highlighted region in the text. To change the position of the highlighted region, use the **`SUIT_highlightBlockInTextEditor()`** call.
MARK_END_INDEX *integer*	This number denotes the ending position of the highlighted region in the text. To change the position of the highlighted region, use the **`SUIT_highlightBlockInTextEditor()`** call.
NUMBER_OF_LINES *integer*	This represents the total number of text lines being edited.
READ_ONLY *boolean*	When TRUE, commands that change the buffer are disabled and the widget becomes a text browser rather than an editor.
SPACING_GAP *int*	Number of pixels between lines of text measured from the lowest descender to the highest ascenders of the next line.
TAB_LENGTH *int*	The length of a tab, measured in characters. For example, setting tab length to 5 will provide tabs every five spaces across the width of the editor. Note that tabs will only line up with a fixed width font such as courier.Tabs are preserved in the text as tabs, not converted to spaces.

Text Editor / Type in Box Keybindings

The notation used here is similar to that used in Emacs: “C-b” is interpreted as “Control-b” and “M-<“is interpreted as “press the escape key, then press the **<** key“. The values given here are default values.

BACKWARD_CHAR_KEY Default value is C-b
text

BEGINNING_OF_LINE_KEY Default value is C-a
text

BEGINNING_OF_TEXT_KEY Default value is M-<
text

DELETE_CHAR_KEY Default value is C-d
text

DELETE_ENTIRE_LINE_KEY Default value is C-u
text

DONE_EDITING_KEY Default value is C-x. This command invokes the callback.
text

END_OF_LINE_KEY Default value is C-e
text

END_OF_TEXT_KEY Default value is M->
text

FORWARD_CHAR_KEY Default value is C-f
text

KILL_LINE_KEY Default value is C-k
text

NEWLINE_KEY Default value is C-m
text

NEXT_LINE_KEY Default value is C-n
text

OPEN_LINE_KEY Default value is C-o. This opens a line at the current cursor position.
text

PREVIOUS_LINE_KEY Default value is C-p
text

REPAINT_KEY Default value is C-l (that’s a lower case letter, not the number 1)
text

SCROLL_DOWN_KEY Default value is M-v
text

SCROLL_UP_KEY Default value is **C-v**
text

SET_MARK_KEY Default value is C-‘ which is equivalent to a C-<space> key sequence.
text

TAB_KEY Default value is C-i
text

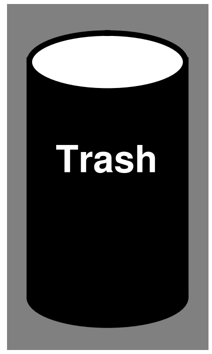
WIPE_BLOCK_KEY Default value is C-w
text

YANK_KEY Default value is C-y
text

Trash Can

Class: "trash can"

Display Styles:



standard

`SUIT_object SUIT_createTrashCan (char *name)`

Description: This widget is nothing more than a fancy placemat.

Property Name	Notes
HELP_MESSAGE *text*	This is the message that appears in a SUIT_inform() style dialog box when the user clicks on the trash can.

Type In Box

Class:"type in box"

Display Styles:

standard

`SUIT_object SUIT_createTypeInBox (char *name, void callback (SUIT_object))`

Description: Creates a one line text entry field.

See Also: The keybindings are exactly the same as those for the Text Editor Widget.

Property Name	Notes
ALTERED *boolean*	FALSE until the text is edited.
ANY_KEYSTROKE_TRIGGERS *boolean*	When TRUE, the callback is called at every keystroke. When FALSE, the callback is only invoked when the user presses RETURN.
CURRENT_VALUE *text*	The text currently being edited. This string is allowed to contain newlines, tabs, but cannot use the GP_text special characters; all characters in the text editor will be the same font.
CURSOR_COLOR *GP_color*	The color of the cursor, which defaults to black.
CURSOR_INDEX *integer*	This temporary property denotes which character the cursor is in front of. For example, when CURSOR_INDEX is 0, this means that the cursor is in front of the first character.
CURSOR_STYLE *SUIT_enum*	This denotes the style of the cursor, either "vertical bar" or "i-beam."
CUT_BUFFER *text*	Contains the last text cut from the CURRENT_VALUE string. This buffer is set whenever the WIPE_BLOCK_KEY or KILL_LINE_KEY key sequence is used (CUT_BUFFER is set to be the killed text) or when the user highlights the text by dragging over the text editor with the mouse (CUT_BUFFER is set to be the text in the highlighted region).
HIGHLIGHT_BLOCK *boolean*	When true, the text that appears between the MARK_INDEX and the MARK_END_INDEX appears in reverse video.

INPUT_SEQUENCE
text

This buffer stores the editor commands before the editor processes them. After the editor recognizes a command, this property is reinitialized to he null string. If you register an interest in this property, you can examine the keyboard command before the editor responds to it. See also the SUIT function called **`SUIT_sendToEditor()`**.

MARK_INDEX
integer

This number denotes the beginning position of the highlighted region in the text. To change the position of the highlighted region, use the **`SUIT_highlightBlockInTextEditor()`** call.

MARK_END_INDEX
integer

This number denotes the ending position of the highlighted region in the text. To change the position of the highlighted region, use the **`SUIT_highlightBlockInTextEditor()`** call.

NUMBER_OF_LINES
integer

This represents the total number of text lines being edited. This number will always be at least 1.

READ_ONLY
boolean

When TRUE, commands that change the buffer are disabled and the widget becomes a text browser rather than an editor.

SPACING_GAP
integer

Number of pixels between lines of text measured from the lowest descender to the highest ascenders of the next line.

TAB_LENGTH
integer

The length of a tab, measured in characters. For example, setting tab length to 5 will provide tabs every five spaces across the width of the editor. Note that tabs will only line up with a fixed width font such as courier.Tabs are preserved in the text as tabs, not converted to spaces.

UVA Logo

Class: “uva logo”

Display Styles:

uva big v rotunda

`SUIT_object SUIT_createUVALogo (char *name)`

Description: This widget displays several different logos for the University of Virginia. It is included in the toolkit because we are insanely proud of our institution.

Property Name	Notes
LINE_WIDTH *integer*	Sets the line thickness for the rotunda display style.