GUI API Nabla

Contenido

Widget

Atributos:

x: [real]

Coordenada píxel posición ‘x’ (esquina superior izquierda del widget)

y: [real]

Coordenada píxel posición ‘y’ (esquina superior izquierda del widget)

w: [real]

Ancho del widget

h: [real]

Largo del widget

color_bg: [list]

Color de fondo del widget (en formato RGB), por defecto es {200,200,200}

color_bo: [list]

Color de borde del widget (en formato RGB), por defecto es {0,0,0}

Funciones:

Move(x [real], y [real]):

Cambia los valores de coordenada píxel posición ‘x’ e ‘y’ del widget (esquina superior izquierda)

Size(w [real], h [real])

Cambia la dimesión (ancho w, largo h) del widget

Color(label [string], color [list])

Cambia el color especificado, por ejemplo para cambiar el color de fondo a color blanco: Color(“bg”, {255,255,255})

setText(type [string], value [undef])

Define una etiqueta que se greficará en el widget.
type es una cadena y puede definir multiples acciones, y *value* es un tipo de dato que va de acuerdo a *type*:

- “add”: Agregar etiqueta
  value [list]:
  {label [string], x [real], y [real], size [real], justify [string], color [list]}

- “addList”: Agregar lista a la lista de etiquetas existente
  value [list of list]:
  {
  {label1 [string], x [real], y [real], size [real], justify [string], color [list]},
  {label2 [string], x [real], y [real], size [real], justify [string], color [list]},
  …
  }

- “setList”: Establecer lista de etiquetas
  value [list of list]:
  {
  {label1 [string], x [real], y [real], size [real], justify [string], color [list]},
  {label2 [string], x [real], y [real], size [real], justify [string], color [list]},
  …
  }
  justify puede ser: “right”, “left” o “mid”

- “remove”: Elimina en la posición *value* definida.
  value [real]

- “clear”: Elimina todas las etiquetas definidas
  value [“string”]: Debe escribir Nill

Por ejemplo, si desea agregar una nueva etiqueta, la entrada es:
setText(“add”, {“etiqueta”, 10, 10, 3, “left”, {0,0,0}})
Por ejemplo si desea eliminar una etiqueta dada una posición, la entrada es:
setText(“remove”, 2)
Por ejemplo si desea todas las etiquetas definidas, la entrada es:
setText(“clear”, Nill)

Open()
Activa el objeto gráfico, y cuando está activo, éste se graficará en la pantalla y podrá interactuar con él. No requiere especificar ningún argumento.
*Por defecto la mayoría de los widgets están activos.

Close()
Desactiva el objeto gráfico, éste ya no se graficará en la pantalla y no podrá interactuar con él. No requiere especificar ningún argumento.

AddWidget(widget [string])
Agrega un widget o elemento gráfico. Debe especificar el nombre del widget como una cadena de texto. Por ejemplo: AddWidget(“object1”, “object2”), el objecto2 será agregado y contenido en el objeto1.

setParent(parent [string])
Similar a AddWidget(), ésta función agrega el propio widget o elemento gráfico a otro(padre) especificado.
Debe especificar el nombre del widget como una cadena de texto. Por ejemplo: setParent(“object1”, “object2”), el objecto1 será agregado y contenido en el objeto2.

setLayout(array [list of lists], prop [list])
Ejecuta un layout de objetos especificados que están contenidos en un widget. Mediante ésta función puede definir layouts horizontales y verticales. *prop* es una lista, pero si no desea definirlo, se reemplaza por Nill: setLayout(array [list of lists], Nill)

Para definir un layout vertical:

```
setLayout(
  {
        {object1 [string], type [string], stretch [real], justifyV [undef], justifyH [undef]},
        {object2 [string], type [string], stretch [real], justifyV [undef], justifyH [undef]},
        …
  },
        {dx_left [real], dy_top [real], w [real], h [real], dx_right [real], dy_bot [real]}
)
```

- *object1* es un string, debe indicar el nombre el objeto o widget
- *type* es una etiqueta que especifica como debe ajustarse el objeto gráfico a su espacio determinado, puede ser de 2 formas: "fit" ajusta el objeto de tal manera que ocupa todo el espacio definido, o puede ser "fixed" no modifica la dimensión del objeto.
- *stretch* define la proporción o porcentaje del layout que ocupará, toma un valor real entre 0 y 1, si se especifica 0, el valor que tomará será igual a la dimensión del widget (largo, ancho)
- *justifyV* define la justificación del objeto siempre y cuando *type* sea "fixed", puede ser "top", "bot" o "mid", si en lugar de una cadena se especifica un valor real positivo, ésta desplazará el objeto dicho valor desde la parte superior hacia abajo.
- *justifyH* define la justificación del objeto siempre y cuando *type* sea "fixed", puede ser "right", "left" o "mid", si en lugar de una cadena se especifica un valor real positivo, ésta desplazará el objeto dicho valor de izquierda a derecha.

El layout que se está definiendo, por defecto toma la dimensión del widget contenedor por defecto, pero es posible modificar la posición y dimensión en el widget contenedor:

- *dx_left* es un valor real positivo, desplaza el layout en píxeles hacia la derecha.
- *dy_top* es un valor real positivo, desplaza el layout en píxeles hacia abajo.
- *w* es un valor real, define el nuevo ancho del layout.
- *h* es un valor real, define el nuevo largo del layout.
- *dx_right* es un valor real, que recorta en píxeles el ancho definido del layout.
- *dy_bot* es un valor real, que recorta en píxeles el largo definido del layout.

Para definir un layout horizontal, la sintaxis es:

```
setLayout(
  {
    {
        {object1 [string], type [string], stretch [real], justifyV [undef], justifyH [undef]},
        {object2 [string], type [string], stretch [real], justifyV [undef], justifyH [undef]},
        …,
        stretch [real]
    }
  },
        {dx_left [real], dy_top [real], w [real], h [real], dx_right [real], dy_bot [real]}
)
```

Ahora bien, puede combinar layouts verticales y horizonales de la siguiente manera:

```
setLayout(
  {
    {
          {object1 [string], type [string], stretch [real], justifyV [undef], justifyH [undef]},
          {object2 [string], type [string], stretch [real], justifyV [undef], justifyH [undef]},
          …,
          stretch [real]
    },
    {object3 [string], type [string], stretch [real], justifyV [undef], justifyH [undef]},
    {object4 [string], type [string], stretch [real], justifyV [undef], justifyH [undef]},
    …
  },
          {dx_left [real], dy_top [real], w [real], h [real], dx_right [real], dy_bot [real]}
)
```

Señales

Connect(“IsClicked”, array [list [expression]])

Esta señal se ejecutará cuando el widget sea presionado. El argumento *array* especifica la función que se conectará a la señal, es una lista: {‘function’}.
Por ejemplo: Connect(“IsClicked”, {‘Function(a,b,c)’})

MainWindow

Atributos:

icon: [string]

Ícono que aparecerá en la barra de título de la ventana, parte superior izquierda. El argumento es un string, y debe especificar el nombre y formato de un archivo imagen cargado a los archivos de aplicación.
Por ejemplo: s_("ventana", "icon", "imagen1.png")

title: [string]

Establece el título de la barra de título de la ventana.
Por ejemplo: s_("ventana", "title", "Nuevo título")

color_bg: [list]

Color de fondo de la ventana (en formato RGB), por defecto es {200,200,200}

color_bo: [list]

Color de borde de la ventana (en formato RGB), por defecto es {200,200,200}

color_la: [list]

Color del título de la ventana (en formato RGB), por defecto es {220,220,220}

color_bgt: [list]

Color de barra de título de la ventana (en formato RGB), por defecto es {54,54,54}

Funciones

Todas las heredadas por la clase Widget.

setTitle(title [string])

Establece el título de la barra de título de la ventana.
Por ejemplo: setTitle("Nuevo título")

BgColor(color [list])

Establece el color de la ventana principal en formato RGB. Por ejemplo para cambiar el color a blanco: BgColor({255,255,255})

Señales

Todas las heredadas por la clase Widget.

DialogWindow

Atributos:

x: [real]

Coordenada píxel posición ‘x’ (esquina superior izquierda de la ventana)

y: [real]

Coordenada píxel posición ‘y’ (esquina superior izquierda de la ventana)

w: [real]

Ancho de la ventana.

h: [real]

Largo de la ventana.

icon: [string]

Ícono que aparecerá en la barra de título de la ventana, parte superior izquierda. El argumento es un string, y debe especificar el nombre y formato de un archivo imagen cargado a los archivos de aplicación.
Por ejemplo: s_(“ventana”, “icon”, “imagen1.png”)

title: [string]

Establece el título de la barra de título de la ventana.
Por ejemplo: s_(“ventana”, “title”, “Nuevo título”)

color_bg: [list]

Color de fondo de la ventana (en formato RGB), por defecto es {200,200,200}

color_bo: [list]

Color de borde de la ventana (en formato RGB), por defecto es {200,200,200}

color_la: [list]

Color del título de la ventana (en formato RGB), por defecto es {220,220,220}

color_bgt: [list]

Color de barra de título de la ventana (en formato RGB), por defecto es {54,54,54}

Funciones

Todas las heredadas por las clases MainWindow y Widget.

Señales

Todas las heredadas por las clases MainWindow y Widget.

TabWidget

Atributos:

x: [real]
Coordenada píxel posición ‘x’ (esquina superior izquierda del widget)

y: [real]
Coordenada píxel posición ‘y’ (esquina superior izquierda del widget)

w: [real]
Ancho del widget

h: [real]
Largo del widget

color_bg: [list]
Color de fondo del widget (en formato RGB), por defecto es {200,200,200}

color_bo: [list]
Color de borde del widget (en formato RGB), por defecto es {120,120,120}

color_la: [list]
Color de los textos de encabezados del widget (en formato RGB), por defecto es {0,0,0}

Funciones:

Todas las heredadas por la clase Widget.

AddWidget(widget [string])
Agrega un objeto de la clase Widget al TabWidget, widget no debe haber sido instaciado antes, ésta función lo realiza por sí misma, de lo contrario generará un error.
Ejemplo: AddWidget(“widget1”)

setHeaders(array [list])
Establece los textos de los encabezados de los widgets agregados anteriormente.
Por ejemplo si se agregó los widgets:
AddWidget(“widget1”)
AddWidget(“widget1”)
Se puede definir los textos de los encabezados de ambos widgets:
setHeaders({“Widget 1.”, “Widget 2.”})

setTabIndex(index [real])
Establece el nuevo índice del TabWidget, el índice indica el widget que se está visualizando.
Ejemplo: setTabIndex(2)

Señales:

Todas las heredadas por la clase Widget.

Connect(“IndexChanged”, array [list [expression]])

Esta señal se ejecutará cuando el TabWidget cambie de índice y posea el foco. El argumento *array* especifica la función que se conectará a la señal, es una lista: {‘function’}.
Por ejemplo: Connect(“IndexChanged”, {‘CualquierFunción(a,b,c)’})

GroupBox

Atributos:

x: [real]
Coordenada píxel posición ‘x’ (esquina superior izquierda del GroupBox)

y: [real]
Coordenada píxel posición ‘y’ (esquina superior izquierda del GroupBox)

w: [real]
Ancho del GroupBox

h: [real]
Largo del GroupBox

label: [string]
Etiqueta del GroupBox.

color_bg: [list]
Color de fondo (en formato RGB), por defecto es {200,200,20}

color_bo: [list]
Color de borde (en formato RGB), por defecto es {120,120,120}

color_la: [list]
Color de etiqueta (en formato RGB), por defecto es {0,0,0}

Funciones:

Todas las heredadas por la clase Widget.

setLabel(label [string])
Establece la etiqueta del GroupBox.

Señales:

Todas las heredadas por la clase Widget.

LineEdit

Atributos:

x: [real]
Coordenada píxel posición ‘x’ (esquina superior izquierda de LineEdit)

y: [real]
Coordenada píxel posición ‘y’ (esquina superior izquierda de LineEdit)

w: [real]
Ancho de LineEdit.

h: [real]
Largo de LineEdit.

label: [string]
Etiqueta de LineEdit.

text: [string]
Texto que contiene LineEdit.

type: [string]
Es el tipo de dato que contendrá el LineEdit, puede ser definido: “text” o “number”. Si es “text” entonces se tratará de un string, caso contrario será tratado como un número o expresión.

color_bg: [list]
Color de fondo (en formato RGB), por defecto es {235,235,235}

color_bo: [list]
Color de borde (en formato RGB), por defecto es {0,0,0}

color_boe: [list]
Color de borde si LineEdit tiene el foco (en formato RGB), por defecto es {0,120,215}

color_txt: [list]
Color de texto que contiene LineEdit (en formato RGB), por defecto es {0,0,0}

Funciones:

Todas las heredadas por la clase Widget

Move(x [real], y[real])
Cambia los valores de coordenada píxel posición ‘x’ e ‘y’ de LineEdit (esquina superior izquierda)

Size(w [real], h [real])
Cambia la dimesión (ancho w, largo h) de LineEdit.

Label(label [string])

Establece la etiqueta de LineEdit.

Text(text [string])

Establece el texto contenido en el LineEdit.

setTypeData(type [string])

Establece el tipo de dato que contendrá el LineEdit, puede ser definido: “text” o “number”. Si es “text” entonces se tratará de un string, caso contrario será tratado como un número o expresión.

Get()

Retorna el valor que contiene el LineEdit. Si el *type* es “text” retornará un string, caso contrario retornará un dato real. No requiere ningún argumento.

Señales:

Todas las heredadas por la clase Widget.

Connect(“EnterKey”, array [list [expression]])

Esta señal se ejecutará cuando se presione el botón ENTER de la calculadora y el LineEdit posea el enfoque. El argumento *array* especifica la función que se conectará a la señal, es una lista: {‘function’}.
Por ejemplo: Connect(“EnterKey”, {‘CualquierFunción(a,b,c)’})

Button

Atributos:

x: [real]
Coordenada píxel posición ‘x’ (esquina superior izquierda del Button)

y: [real]
Coordenada píxel posición ‘y’ (esquina superior izquierda del Button)

w: [real]
Ancho del Button.

h: [real]
Largo del Button.

label: [string]
Texto del Button, también puede definir un ícono con el formato: “icon.png”.

color_bg: [list]
Color de fondo del Button (en formato RGB), por defecto es {180,180,180}

color_bo: [list]
Color de borde del Button (en formato RGB), por defecto es {100,100,100}

color_la: [list]
Color del texto del Button (en formato RGB), por defecto es {0,0,0}

Funciones:

Todas las heredadas por la clase Widget

Move(x [real], y[real])
Cambia los valores de coordenada píxel posición ‘x’ e ‘y’ del Button (esquina superior izquierda)

Size(w [real], h [real])
Cambia la dimesión (ancho w, largo h) del Button.

Label(label [string])
Establece la etiqueta del Button, también puede definir un ícono con el formato: “icon.png”.

Señales:

Todas las heredadas por la clase Widget.

CheckBox

Atributos:

x: [real]

Coordenada píxel posición ‘x’ (esquina superior izquierda del CheckBox).
Por defecto es 0.

y: [real]

Coordenada píxel posición ‘y’ (esquina superior izquierda del CheckBox).
Por defecto es 0.

w: [real]

Ancho del CheckBox, por defecto es 20.

h: [real]

Largo del CheckBox, por defecto es 20.

mark: [string]

Etiqueta o texto del CheckBox, por defecto es "✓", también puede definir un ícono con el formato: “icon.png”.

status: [boolean]

Estado de selección del CheckBox. Si es 1 entonces está seleccionado, caso contrario no está seleccionado. Por defecto es 1.

color_bg: [list]

Color de fondo (en formato RGB), por defecto es {225,225,225}

color_bo: [list]

Color de borde (en formato RGB), por defecto es {0,0,0}

color_mark: [list]

Color de etiqueta o texto (en formato RGB), por defecto es {0,0,0}

Funciones:

Move(x [real], y[real])

Cambia los valores de coordenada píxel posición ‘x’ e ‘y’ del CheckBox (esquina superior izquierda)

Size(w [real], h [real])

Cambia la dimesión (ancho w, largo h) del CheckBox.

Mark(mark [string])

Establece la etiqueta o texto del CheckBox, también puede definir un ícono con el formato: “icon.png”.

Color(label [string], color [list])

Cambia el color especificado, por ejemplo para cambiar el color de fondo a color blanco: Color(“bg”, {255,255,255})

Señales:

Connect(“StatusChanged”, array[list [expression]])

Esta señal se ejecutará cada vez que el estado del CheckBox cambie ya sea de 1 a 0 ó de 0 a 1. El argumento *array* especifica la función que se conectará a la señal, es una lista: {‘function’}.
Por ejemplo: Connect(“StatusChanged”, {‘CualquierFunción(a,b,c)’})

RadioButton

Atributos:

x: [real]

Coordenada píxel posición ‘x’ (esquina superior izquierda del RadioButton).
Por defecto es 0.

y: [real]

Coordenada píxel posición ‘y’ (esquina superior izquierda del RadioButton).
Por defecto es 0.

w: [real]

Ancho del RadioButton, por defecto es 150.

h: [real]

Largo del RadioButton, por defecto es 100.

label: [string]

Etiqueta del RadioButton, por defecto es “RadioButton”

color_bg: [list]

Color de fondo (en formato RGB), por defecto es {200,200,200}

color_bo: [list]

Color de borde (en formato RGB), por defecto es {120,120,120}

color_la: [list]

Color de texto del encabezado (en formato RGB), por defecto es {0,0,0}

Funciones:

Todas las heredadas por las clases GroupBox y Widget.

AddWidget(widget [string])

Agrega un CheckBox definido con anterioridad al RadioButton, *widget* es un string y hace referencia a un objeto(variable de usuario definida con anterioridad) de la clase CheckBox.

Señales:

Todas la heredadas por las clases GroupBox y Widget.

ComboBox

Atributos:

x: [real]

Coordenada píxel posición ‘x’ (esquina superior izquierda).
Por defecto es 0.

y: [real]

Coordenada píxel posición ‘y’ (esquina superior izquierda).
Por defecto es 0.

w: [real]

Ancho de la barra de encabezado, por defecto es 60.

h: [real]

Largo del ComboBox, por defecto es 20.

wl: [real]

Ancho de lista del ComboBox, por defecto es 60.

hl: [real]

Largo o altura de los elementos de la lista del ComboBox, por defecto es 18.

title: [string]

Texto del título de la barra del encabezado del ComboBox, por defecto es “ComboBox”.

list: [list]

Lista de elementos a mostrar en el ComboBox, pueden ser cualquier tipo de dato, ya que éstos serán convertidos a string.

fh: [boolean]

Boleano *fh,* si es 1 entonces mostrará *title* en el encabezado, caso contrario mostrará el elemento actual seleccionado.

index: [real integer]

Posición del elemento enfocado en el ComboBox.

sb_w: [real]

Ancho de la barra de desplazamiento del ComboBox, si es 0 entonces ésta no se graficará.

sb_he: [real]

Barras de separación entre la barra de desplazamiento y su altura total.

color_bg: [list]

Color de fondo (en formato RGB), por defecto es {190,210,240}

color_bo: [list]
Color de borde (en formato RGB), por defecto es {0,0,0}

color_la: [list]
Color de texto del encabezado (en formato RGB), por defecto es {0,0,0}

color_lbg: [list]
Color de fondo de la lista (en formato RGB), por defecto es {210,210,210}

color_lla: [list]
Color de de texto de la lista (en formato RGB), por defecto es {0,0,0}

color_pbg: [list]
Color de indice de la lista (en formato RGB), por defecto es {200,100,255}

color_sbg: [list]
Color de barra de desplazamiento (en formato RGB), por defecto es {120,120,120}

Funciones:

Move(x [real], y [real])
Cambia los valores de coordenada píxel posición 'x' e 'y' (esquina superior izquierda)

Size(w [real], h [real])
Establece la nueva dimesión (ancho w, largo h).

Color(label [string], color [list])
Cambia el color especificado, por ejemplo para cambiar el color de fondo a color blanco: Color("bg", {255,255,255})

setHeightList(height [real])
Establece el largo o altura de la lista del ComboBox.

Title(title [string])
Establece un nuevo título de encabezado del ComboBox.

setList(array [list])
Establece la lista que contendrá el ComboBox.

setDimView(dimview [real integer])
Establece la dimensión de elementos que se mostrará en la lista del ComboBox.

setIndex(index [real integer])
Establece el índice del elemento enfocado en la lista del ComboBox.

setListDrop(status [boolean])

Establece si el ComboBox es desplegable. Si es 1 la lista no es desplegable, si es 0 la lista es desplegable.

Señales:

Connect("IndexChanged", array[list [expression]])

Esta señal se ejecutará cada vez que el índice de enfoque de la lista cambie a un valor diferente. El argumento *array* especifica la función que se conectará a la señal, es una lista: {'function'}.
Por ejemplo: Connect("IndexChanged", {'CualquierFunción(a,b,c)'})

Connect("CurrentIndex", array[list [expression]])

Esta señal se ejecutará solo si el ComboBox es definido como desplegable, al seleccionar un item la lista se cierra y esta señal es activada. El argumento *array* especifica la función que se conectará a la señal, es una lista: {'function'}.
Por ejemplo: Connect("CurrentIndex", {'CualquierFunción(a,b,c)'})

Connect("LoseFocus", array[list [expression]])

Esta señal se ejecutará si el ComboBox pierde el enfoque y no es seleccionado ningún item. El argumento *array* especifica la función que se conectará a la señal, es una lista: {'function'}.
Por ejemplo: Connect("LoseFocus", {'CualquierFunción(a,b,c)'})

TableWidget

Parámetros:

x: [real]

Coordenada píxel posición ‘x’ (esquina superior izquierda).
Por defecto es 0.

y: [real]

Coordenada píxel posición ‘y’ (esquina superior izquierda).
Por defecto es 0.

w: [real]

Ancho de la barra de encabezado, por defecto es 200.

h: [real]

Largo de la barra de encabezado, por defecto es 20.

title: [string]

Texto o título de la barra de encabezado de la tabla, por defecto es “TableWidget”.

ht: [real]

Largo o alto de los elementos de la tabla, por defecto es 20.

pos: [list]

Posición o elemento seleccionado de la tabla, por defecto es {1,1}

head: [string]

Texto de barra de intersección de encabezados, por defecto es “”.

row_header: [list]

Encabezados de filas de la tabla, puede tomar datos reales o string.

col_header: [list]

Encabezados de columnas de la tabla, puede tomar datos reales o string.

hrow_w: [real]

Ancho de encabezado de fila, por defecto es 25.

hcol_h: [real]

Largo o altura de encabezado de columna, por defecto es 20.

dimview: [list {rows, columns}]

Dimensión visualizada de la tabla, en una lista debes indicar la cantidad de filas y columnas, por defecto es {3, 4}

hi: [real]

Largo o altura de la barra inferior de la tabla, por defecto es 20.

sb_w: [real]
Ancho de la barra de desplazamiento del ComboBox, si es 0 entonces ésta no se graficará, por defecto es 15.

sb_he: [real]
Separación entre la barra de desplazamiento y su altura total, por defecto es 7.

color_bg: [list]
Color de fondo de barra de encabezado (en formato RGB), por defecto es {190,210,240}

color_bo: [list]
Color de borde (en formato RGB), por defecto es {0,0,0}

color_la: [list]
Color de texto o título de barra de encabezado (en formato RGB), por defecto es {0,0,0}

color_tbg: [list]
Color de fondo de la tabla (en formato RGB), por defecto es {210,210,210}

color_tla: [list]
Color de de texto de la tabla (en formato RGB), por defecto es {0,0,0}

color_hbg: [list]
Color de fondo de encabezados (en formato RGB), por defecto es {150,150,150}

color_hla: [list]
Color de texto de encabezados (en formato RGB), por defecto es {120,120,120}

color_pbg: [list]
Color de fondo de selección (en formato RGB), por defecto es {180,180,180}

color_ibg: [list]
Color de fondo de barra inferior (en formato RGB), por defecto es {255,255,255}

color_sbg: [list]
Color de barra de desplazamiento (en formato RGB), por defecto es {120,120,120}

Funciones:

Move(x [real], y [real])
Cambia los valores de coordenada píxel posición ‘x’ e ‘y’ (esquina superior izquierda)

Size(w [real], h [real])
Establece la nueva dimesión (ancho w, largo h).

Title(title [string])
Establecer el título de la barra de encabezado de TableWidget.

Color(label [string], color [list])

Cambia el color especificado, por ejemplo para cambiar el color de fondo a color blanco:
Color(“bg”, {255,255,255})

setTable(table [list o lists or matrix])

Establece la tabla del TableWidget.
Por ejemplo: setTable({{1,2}, {3,4}}) ó setTable([[1,2],[3,4]])

setHeightTable(height [real])

Establece el largo o altura de los elementos visualizado de la tabla del TableWidget.

setHead(head [string])

Establece el texto de barra de intersección de encabezados.

setRowHeader(row_header [list])

Establece los encabezados de las filas de la tabla del TableWidget.
Por ejemplo, si su tabla tiene tres filas, necesitará tres encabezados:
setRowHeader({“encabezado1”, “encabezado2”, “encabezado 3”})

setColHeader(col_header [list])

Establece los encabezados de las columnas de la tabla del TableWidget.
Por ejemplo, si su tabla tiene tres columnas, necesitará tres encabezados:
setColHeader({“encabezado1”, “encabezado2”, “encabezado 3”})

setRowHeaderWeight(weight [real])

Establece el ancho de los encabezados de filas de la tabla.

setColHeaderHeight(height [real])

Establece el largo o altura del los encabezados de columnas de la tabla.

setDimView(dimview [list {rows, columns}])

Establece la dimensión visualizada de la tabla, en una lista debes indicar la cantidad de filas y columnas.

Señales:

Ninguna.

Board

Atributos:

x: [real]

Coordenada píxel posición ‘x’ (esquina superior izquierda).
Por defecto es 0.

y: [real]

Coordenada píxel posición ‘y’ (esquina superior izquierda).
Por defecto es 0.

w: [real]

Ancho de Board, por defecto es 320.

h: [real]

Largo de Board, por defecto es 240.

lx: [list]

Lista que almacena las coordenadas ‘x’ de vértices del conjunto.

ly: [list]

Lista que almacena las coordenadas ‘y’ de vértices del conjunto.

lz: [list]

Lista que almacena las coordenadas ‘z’ de vértices del conjunto.

deltax: [list {dx_left [real], dx_right [real]}]

Recorte del ancho de Board para graficar.

deltay: [list {dy_top [real], dy_bot [real]}]

Recorte del largo o altura de Board para graficar.

li: [list]

Lista que almacena las conexiones ‘i’ de vértices del conjunto.

lj: [list]

Lista que almacena las conexiones ‘j’ de vértices del conjunto.

ls: [list of lists]

Lista que almacena las conexiones de vértices del conjunto para graficar superficies.
La sintaxis es: s_(self, “ls”, {{1,2,3}, {4,3,1,4,2}, …}), nótese que puede definir conexiones de *n* cantidad de vértices.

enumv: [boolean]

Si enumv es 1, entonces la numeración de vértices será graficada, caso contrario no se graficará. Por defecto es 0.

enume: [bolean]
Si enume es 1, entonces la enumeración de aristas será graficada, caso contrario no se graficará. Por defecto es 0.

zf: [real]
Factor de alejamiento del centro de giro, por defecto es 3.

color_bg: [list]
Color de fondo (en formato RGB), por defecto es {200,200,200}

color_bo: [list]
Color de borde (en formato RGB), por defecto es {0,0,0}

color_v: [list]
Color de vértices (en formato RGB), por defecto es {200,200,150}

color_e: [list]
Color de aristas (en formato RGB), por defecto es {50,100,150}

color_s: [list]
Color de superficies (en formato RGB), por defecto es {0,128,0}

color_vs: [list]
Color de vértices seleccionados (en formato RGB), por defecto es {130,130,130}

color_es: [list]
Color de aristas seleccionadas (en formato RGB), por defecto es {130,130,130}

Funciones:
Todas las heredadas por la clase Widget.

Move(x [real], y [real])
Cambia los valores de coordenada píxel posición 'x' e 'y' (esquina superior izquierda)

Size(w [real], h [real])
Establece la nueva dimesión (ancho w, largo h).

Vertex(type [string], value [undef])
Ésta función define y manipula los vértices. *type* es una etiqueta que puede tomar los siguientes valores:

- "add": Agrega un vértice
  value: Es una lista con valores coordenados {x [real], y [real], z [real]}
- "addList": Agrega una lista de vértices
  value: Es una lista de listas, con valores coordenados de los vértices:
  {{x1 [real], x2 [real], …}, {y1 [real], y2 [real], …}, {z1 [real], z2 [real], …}}
- "setList": Establece la lista de vértices
  value: Es una lista de listas, con valores coordenados de los vértices:

{{x1 [real], x2 [real], …}, {y1 [real], y2 [real], …}, {z1 [real], z2 [real], …}}

- "remove": Elimina un vértice dada una posición
  value: Es un real, indica la posición a eliminar
- "clear": Elimina todos los vértices definidos
  value: No es requerido, pero debe ser Nill.
- "modify": Modifica el valor de un vértice específico.
  value: Es una lista {pos [real], eje [string], newValue [real]}, donde *pos* es el índice del vértice a modificar, *eje* es un string, puede ser "x", "y" y "z", y newValue es un real, el valor nuevo.
- "newSet": Define un nuevo conjunto de vértices
  value: Es una lista
  {name [string], lx [list], ly [list], lz [list], act [boolean], color [list]} donde *name* es un string que indica el nombre del nuevo conjunto de vértices.
  lx, ly, lz, son listas de coordenadas de los nuevos vértices, éstas serán agregadas a la lista de vértices existentes, pero mantendrá sus propiedades definidas.
  act es un boleano, si es 1 entonces sus vértices serán seleccionables, caso contrario no serán seleccionables.
  color es una lista con el color RGB {r [real], g [real], b [real]} que será utilizado para el nuevo conjunto.
- "addListSet": Agrega una lista de vértices nuevos a un conjunto existente.
  value: Es una lista {name [string], lx [list], ly [list], lz [list]} donde:
  name es el nombre del conjunto al que se desea agregar nuevos vértices.
  lx, ly, lz son listas de coordenadas de los vertices que se desea agregar.
- "removeSet": Elimina un conjunto definido.
  value: Es una cadena que indica el nombre del conjunto a remover.
- "replaceSet": Reemplaza todos los vértices de un conjunto definido por una nueva lista de vértices.
  value: Es una lista {name [string], lx [list], ly [list], lz [list]} donde:
  name es una cadena que indica el nombre del conjunto a reemplazar.
  lx, ly, lz son listas de coordenadas de los vértices que se desea reemplazar.

Edge(type [string], value [undef])

Ésta función define y manipula las aristas. *type* es una etiqueta que puede tomar los siguientes valores:

- "add": Agrega una nueva arista
  value: Es una lista con valores de conexión {i [real], j [real]}
- "addList": Agrega una lista de aristas
  value: Es una lista de listas, con valores de conexión de vértices:
  {{i1, i2, …}, {j1, j2, …}}
- "setList": Establece la lista de vértices
  value: Es una lista de listas, con valores de conexión de vértices:
  {{i1, i2, …}, {j1, j2, …}}
- "remove": Elimina una conexión de vértices dada una posición
  value: Es un real, indica la posición a eliminar
- "clear": Elimina todos las aristas definidas
  value: No es requerido, pero debe ser Nill.
- "newSet": Define un nuevo conjunto de aristas
  value: Es una lista
  {name [string], li [list], lj [list], act [boolean], color [list]} donde:
  name es un string que indica el nombre del nuevo conjunto de aristas.

li, lj son listas de nuevas conexiones de aristas, éstas serán agregadas a la lista de aristas existentes, pero mantendrá sus propiedades definidas.
act es un boleano, si es 1 entonces sus aristas serán seleccionables, caso contrario no serán seleccionables.
color es una lista con el color RGB {r [real], g [real], b [real]} que será utilizado para el nuevo conjunto.

- "addListSet": Agrega una lista de aristas nuevas a un conjunto existente.
  value: Es una lista {name [string], li [list], lj [list]} donde:
  name es el nombre del conjunto al que se desea agregar nuevas aristas.
  li, lj son listas de conexiones de vértices que se desea agregar.
- "removeSet": Elimina un conjunto definido.
  value: Es una cadena que indica el nombre del conjunto a remover.
- "replaceSet": Reemplaza todos las aristas de un conjunto definido por una nueva lista de aristas.
  value: Es una lista {name [string], li [list], lj [list]} donde:
  name es una cadena que indica el nombre del conjunto a reemplazar.
  li, lj son listas de aristas de conexiones de vértices que se desea reemplazar.

Surfaces(type [string], value [undef])

Ésta función define y manipula las superficies. *type* es una etiqueta que puede tomar los siguientes valores:

- "add": Agrega una nueva superficie
  value: Es una lista con valores de conexión de aristas {v1 [real], v2 [real], …}
- "addList": Agrega una lista de superficies
  value: Es una lista de listas, con indices de vértices:
  {{v1, v2, …}, {vert1, vert2, …}, …}
- "setList": Establece la lista de superficies
  value: Es una lista de listas, con valores de superficies:
  {{v1, v2, …}, {vert1, vert2, …}, …}
- "remove": Elimina una superficie definida dada una posición
  value: Es un real, indica la posición a eliminar
- "clear": Elimina todos las superficies definidas
  value: No es requerido, pero debe ser Nill.
- "newSet": Define un nuevo conjunto de superficies
  value: Es una lista
  {name [string], ls [list], color [list]} donde:
  name es un string que indica el nombre del nuevo conjunto de superficie.
  ls, son listas de nuevas superficies, éstas serán agregadas a la lista de asuperficies existentes, pero mantendrá sus propiedades definidas.
  color es una lista con el color RGB {r [real], g [real], b [real]} que será utilizado para el nuevo conjunto.
- "addListSet": Agrega una lista de superficies nuevas a un conjunto existente.
  value: Es una lista {name [string], ls [list]} donde:
  name es el nombre del conjunto al que se desea agregar nuevas superficies.
  ls son listas de superficies que se desea agregar.
- "removeSet": Elimina un conjunto definido.
  value: Es una cadena que indica el nombre del conjunto a remover.
- "replaceSet": Reemplaza todos las superficies de un conjunto definido por una nueva lista de superficies.
  value: Es una lista {name [string], ls [list]} donde:
  name es una cadena que indica el nombre del conjunto a reemplazar.

ls son listas de aristas de superficies que se desea reemplazar.

Projection(proj [string])

Establece el tipo de proyeción, “p” para proyeción en perspectiva y “o” para proyección ortográfica.

PFactor(factor [real])

Establece el factor de alejamiento del centro de giro.

Señales:

Todas las heredadas por la clase Widget.

Connect(“Vertex”, array[list [expression]])

Esta señal se ejecutará si algún vértice ha sido presionado. El argumento *array* especifica la función que se conectará a la señal, es una lista: {‘function’}.
Por ejemplo: Connect(“Vertex”, {‘CualquierFunción(a,b,c)’})

Connect(“Edge”, array[list [expression]])

Esta señal se ejecutará si alguna arista del conjunto ha sido presionado. El argumento *array* especifica la función que se conectará a la señal, es una lista: {‘function’}.
Por ejemplo: Connect(“Edge”, {‘CualquierFunción(a,b,c)’})