RUBY/TK - MENUBUTTON WIDGET

Description:

A **menubutton** is a widget that displays a textual string, bitmap, or image and is associated with a menu widget. If text is displayed, it must all be in a single font, but it can occupy multiple lines on the screen (if it contains newlines or if wrapping occurs because of the *wraplength* option) and one of the characters may optionally be underlined using the underline option.

In normal usage, pressing mouse button 1 over the menubutton causes the associated menu to be posted just underneath the menubutton. If the mouse is moved over the menu before releasing the mouse button, the button release causes the underlying menu entry to be invoked. When the button is released, the menu is unposted.

Menubuttons are typically organized into groups called menu bars that allow scanning: if the mouse button is pressed over one menubutton and the mouse is moved over another menubutton in the same menu bar without releasing the mouse button, then the menu of the first menubutton is unposted and the menu of the new menubutton is posted instead.

Syntax:

Here is a simple syntax to create this widget:

```
TkMenubutton.new(root) {
  .....Standard Options....
  .....Widget-specific Options....
}
```

Standard Options:

- activebackground
- cursor
- highlightthickness
- takefocus
- activeforeground
- disabledforeground
- image
- text
- anchor
- font
- justify
- textvariable
- background
- foreground
- padx
- underline
- bitmap

- highlightbackground
- pady
- wraplength
- borderwidth
- highlightcolor
- relief

These options have been described in previous chapter.

Widget-specific Options:

SN	Options with Description
1	**compound** => String Specifies whether the button should display both an image and text, and if so, where the image should be placed relative to the text. Valid values for this option are **bottom**, **center**, **left**, **none**, **right** and **top**. The default value is **none**, meaning that the button will display either an image or text, depending on the values of the *image* and *bitmap* options.
2	**direction** => String Specifies where the menu is going to be popup up. **above** tries to pop the menu above the menubutton. **below** tries to pop the menu below the menubutton. **left** tries to pop the menu to the left of the menubutton. **right** tries to pop the menu to the right of the menu button. **flush** pops the menu directly over the menubutton.
3	**height** => Integer Specifies a desired height for the menubutton.
4	**indicatoron** => Boolean The value must be a proper boolean value. If it is true then a small indicator rectangle will be displayed on the right side of the menubutton and the default menu bindings will treat this as an option menubutton. If false then no indicator will be displayed.
5	**menu** => String Specifies the path name of the menu associated with this menubutton. The menu must be a child of the menubutton.
6	**state** => String Specifies one of three states for the menubutton: **normal**, **active**, or **disabled**. In normal state the menubutton is displayed using the **foreground** and **background** options.
7	**width** => Integer Specifies a desired width for the menubutton.

Event Bindings:

Ruby/Tk automatically creates class bindings for menubuttons that give them the following default

behavior:

- A menubutton activates whenever the mouse passes over it and deactivates whenever the mouse leaves it.
- Pressing mouse button 1 over a menubutton posts the menubutton: its relief changes to raised and its associated menu is posted under the menubutton. If the mouse is dragged down into the menu with the button still down, and if the mouse button is then released over an entry in the menu, the menubutton is unposted and the menu entry is invoked.
- If button 1 is pressed over a menubutton and then released over that menubutton, the menubutton stays posted: you can still move the mouse over the menu and click button 1 on an entry to invoke it. Once a menu entry has been invoked, the menubutton unposts itself.
- If button 1 is pressed over a menubutton and then dragged over some other menubutton, the original menubutton unposts itself and the new menubutton posts.
- If button 1 is pressed over a menubutton and released outside any menubutton or menu, the menubutton unposts without invoking any menu entry.
- When a menubutton is posted, its associated menu claims the input focus to allow keyboard traversal of the menu and its submenus.
- If the underline option has been specified for a menubutton then keyboard traversal may be used to post the menubutton: Alt+x, where x is the underlined character *oritslower – caseorupper – caseequivalent*, may be typed in any window under the menubutton's toplevel to post the menubutton.
- The F10 key may be typed in any window to post the first menubutton under its toplevel window that isn't disabled.
- If a menubutton has the input focus, the space and return keys post the menubutton.

If the menubutton's state is **disabled** then none of the above actions occur: the menubutton is completely non-responsive.

Examples:

```
require "tk"

mbar = TkFrame.new {
  relief 'raised'
  borderwidth 2
}
mbar.pack('fill' => 'x')

TkMenubutton.new(mbar) {|mb|
  text "File"
  underline 0
  menu TkMenu.new(mb) {
    add 'command', 'label' => 'New...', 'underline' => 0,
   'command' => proc {print "opening new file\n"}
    add 'command', 'label' => 'Quit',
   'underline' => 0, 'command' => proc{exit}
  }
  pack('side' => 'left', 'padx' => '1m')
}

TkMenubutton.new(mbar) {|mb|
  text "Help"
  underline 0
  menu TkMenu.new(mb) {
    add 'command', 'label' => 'About', 'underline' => 0,
   'command' => proc {print "This is menu example.\n"}
  }
  pack('side' => 'left', 'padx' => '1m')
}
```

```
Tk.mainloop
```

This will produce the following result: