RUBY/TK - MENU WIDGET

Description:

A **menu** is a widget that displays a collection of one-line entries arranged in one or more columns. There exist several different types of entries, each with different properties. Entries of different types may be combined in a single menu. Menu entries are not the same as entry widgets. In fact, menu entries are not even distinct widgets; the entire menu is one widget.

When first created, a new listbox has no elements. Elements may be added or deleted using provided methods. In addition, one or more elements may be selected from the listed items.

It is not necessary for all the elements to be displayed in the listbox window at once. Listboxes allow scrolling in both directions using the standard *xscrollcommand* and *yscrollcommand* options

Syntax:

Here is a simple syntax to create this widget:

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

Standard Options:

- activebackground
- background
- disabledforeground
- relief
- activeborderwidth
- borderwidth
- font
- takefocus
- activeforeground
- cursor
- foreground

These options have been described in previous chapter.

Widget-specific Options:

SN	Options with Description
1	**postcommand** => String If this option is specified then it provides a callback to execute each time the menu is posted. The callback is invoked by the post method before posting the menu.
2	**selectcolor** => String

For menu entries that are check buttons or radio buttons, this option specifies the color to display in the indicator when the check button or radio button is selected.

3 **tearoff** => Integer

This option must have a proper boolean value, which specifies whether or not the menu should include a tear-off entry at the top. If so, it will exist as entry 0 of the menu and the other entries will number starting at 1. The default menu bindings arrange for the menu to be torn off when the tear-off entry is invoked.

4 **tearoffcommand** => String

If this option has a non-empty value, then it specifies a Ruby/Tk callback to invoke whenever the menu is torn off. The actual command will consist of the value of this option, followed by a space, followed by the name of the menu window, followed by a space, followed by the name of the name of the torn off menu window. For example, if the option's is "a b" and menu .x.y is torn off to create a new menu .x.tearoff1, then the command "a b .x.y .x.tearoff1" will be invoked.

5 **title** => String

The string will be used to title the window created when this menu is torn off. If the title is NULL, then the window will have the title of the menubutton or the text of the cascade item from which this menu was invoked.

6 **type** => String

This option can be one of **menubar**,**tearoff**, or **normal**, and is set when the menu is created.

Manipulating the Menus:

There are various ways to play with a Menus:

- The **activate***index***:** method is used to change the state of the entry indicated by *index* to **active** and redisplay it using its active colors.
- The **add***type, ?option, value, option, value, . . . ?***:** method is used to add a new entry to the bottom of the menu. The new entry's type is given by *type* and must be one of **cascade**, **checkbutton**, **command**, **radiobutton**, or **separator**, or a unique abbreviation of one of the above.
- The **delete***index1?, index2?***:** method is used to delete all of the menu entries between *index1* and *index2* inclusive. If *index2* is omitted then it defaults to *index1*.
- The **index***index***:** method returns the numerical index corresponding to *index*, or **none** if *index* was specified as **none**.
- The **insert***index, type?, option => value, . . . ?***:** method is same as the **add** method except that it inserts the new entry just before the entry given by *index*, instead of appending to the end of the menu. The *type*, *option*, and *value* arguments have the same interpretation as for the **add** widget method.
- The **invoke***index***:** method is used to invoke the action of the menu entry.
- The **post***x, y***:** method is used to arrange for the menu to be displayed on the screen at the root-window coordinates given by x and y.
- The **postcascade***index***:** method posts the submenu associated with the cascade entry given by *index*, and unposts any previously posted submenu.

- The **type***index***:** method returns the type of the menu entry given by *index*. This is the *type* argument passed to the **add** widget method when the entry was created, such as **command** or **separator**, or **tearoff** for a tear-off entry.
- The **unpost** method unmaps the window so that it is no longer displayed. If a lower-level cascaded menu is posted, unpost that menu. Returns an empty string.
- The **yposition***index* method returns a decimal string giving the y-coordinate within the menu window of the topmost pixel in the entry specified by *index*.

Menu Configuration:

The default bindings support four different ways of using menus:

- **Pulldown Menus:** This is the most common case. You create one menubutton widget for each top-level menu, and typically you arrange a series of menubuttons in a row in a menubar window. You also create the top-level menus and any cascaded submenus, and tie them together with *menu* options in **menubuttons** and cascade menu entries.
- **Popup Menus:** Popup menus typically post in response to a mouse button press or keystroke. You create the popup menus and any cascaded submenus, then you call the **Popup** method at the appropriate time to post the top-level menu.
- **Option Menus:** An option menu consists of a menubutton with an associated menu that allows you to select one of several values. The current value is displayed in the menubutton and is also stored in a global variable. Use the **Optionmenu** class to create option menubuttons and their menus.
- **Torn-off Menus:** You create a torn-off menu by invoking the tear-off entry at the top of an existing menu. The default bindings will create a new menu that is a copy of the original menu and leave it permanently posted as a top-level window. The torn-off menu behaves just the same as the original menu.

Event Bindings:

Ruby/Tk automatically creates class bindings for menus that give them the following default behavior:

- When the mouse enters a menu, the entry underneath the mouse cursor activates; as the mouse moves around the menu, the active entry changes to track the mouse.
- When the mouse leaves a menu all of the entries in the menu deactivate, except in the special case where the mouse moves from a menu to a cascaded submenu.
- When a button is released over a menu, the active entry $ifany$ is invoked. The menu also unposts unless it is a torn-off menu.
- The Space and Return keys invoke the active entry and unpost the menu.
- If any of the entries in a menu have letters underlined with **underline** option, then pressing one of the underlined letters $oritsupper - caseorlower - caseequivalent$ invokes that entry and unposts the menu.
- The Escape key aborts a menu selection in progress without invoking any entry. It also unposts the menu unless it is a torn-off menu.
- The Up and Down keys activate the next higher or lower entry in the menu. When one end of the menu is reached, the active entry wraps around to the other end.
- The Left key moves to the next menu to the left. If the current menu is a cascaded submenu, then the submenu is unposted and the current menu entry becomes the cascade entry in the parent. If the current menu is a top-level menu posted from a menubutton, then the current menubutton is unposted and the next menubutton to the left is posted. Otherwise the key has no effect. The left-right order of menubuttons is determined by their stacking order: Tk assumes that the lowest menubutton $whichbydefaultisthefirstonecreated$ is on the left.
- The Right key moves to the next menu to the right. If the current entry is a cascade entry,

then the submenu is posted and the current menu entry becomes the first entry in the submenu. Otherwise, if the current menu was posted from a menubutton, then the current menubutton is unposted and the next menubutton to the right is posted.

Disabled menu entries are non-responsive: they don't activate and they ignore mouse button presses and releases.

Examples:

```
require "tk"

root = TkRoot.new
root.title = "Window"


menu_click = Proc.new {
  Tk.messageBox(
    'type'    => "ok",
    'icon'    => "info",
    'title'   => "Title",
    'message' => "Message"
  )
}

file_menu = TkMenu.new(root)

file_menu.add('command',
              'label'     => "New...",
              'command'   => menu_click,
              'underline' => 0)
file_menu.add('command',
              'label'     => "Open...",
              'command'   => menu_click,
              'underline' => 0)
file_menu.add('command',
              'label'     => "Close",
              'command'   => menu_click,
              'underline' => 0)
file_menu.add('separator')
file_menu.add('command',
              'label'     => "Save",
              'command'   => menu_click,
              'underline' => 0)
file_menu.add('command',
              'label'     => "Save As...",
              'command'   => menu_click,
              'underline' => 5)
file_menu.add('separator')
file_menu.add('command',
              'label'     => "Exit",
              'command'   => menu_click,
              'underline' => 3)

menu_bar = TkMenu.new
menu_bar.add('cascade',
             'menu'  => file_menu,
             'label' => "File")

root.menu(menu_bar)

Tk.mainloop
```

This will produce the following result: