RUBY/TK - TEXT WIDGET

Description:

A **Text** widget provides users with an area so that they can enter multiple lines of text. Text widgets are part of the classic Tk widgets, not the themed Tk widgets.

Text widgets support three different kinds of annotations on the text:

- **Tags** - allow different portions of the text to be displayed with different fonts and colors. In addition, Tcl commands can be associated with tags so that scripts are invoked when particular actions such as keystrokes and mouse button presses occur in particular ranges of the text.
- **Marks** - The second form of annotation consists of marks, which are floating markers in the text. Marks are used to keep track of various interesting positions in the text as it is edited.
- **Embedded windows** - The third form of annotation allows arbitrary windows to be embedded in a text widget.

A label can display a textual string, bitmap or image. 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.

Syntax:

Here is a simple syntax to create this widget:

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

Standard Options:

- background
- borderwidth
- cursor
- exportselection
- font
- foreground
- highlightbackground
- highlightcolor
- highlightthickness
- insertbackground
- insertborderwidth
- insertofftime
- insertontime
- insertwidth

- padx
- pady
- relief
- selectbackground
- selectborderwidth
- selectforeground
- setgrid
- takefocus
- xscrollcommand
- yscrollcommand

These options have been described in previous chapter.

Widget-specific Options:

SN	Options with Description
1	**height** => Integer Specifies the desired height for the window, in units of characters. Must be at least one.
2	**spacing1** => Integer Requests additional space above each text line in the widget, using any of the standard forms for screen distances. If a line wraps, this option only applies to the first line on the display. This option may be overriden with **spacing1** options in tags.
3	**spacing2** => Integer For lines that wrap *sothattheycovermorethanonelineonthedisplay* this option specifies additional space to provide between the display lines that represent a single line of text. The value may have any of the standard forms for screen distances. This option may be overriden with **spacing** options in tags.
4	**spacing3** => Integer Requests additional space below each text line in the widget, using any of the standard forms for screen distances. If a line wraps, this option only applies to the last line on the display. This option may be overriden with **spacing3** options in tags.
5	**state** => String Specifies one of two states for the text: **normal** or **disabled**. If the text is disabled then characters may not be inserted or deleted and no insertion cursor will be displayed, even if the input focus is in the widget.
6	**tabs** => String Specifies a set of tab stops for the window. The option's value consists of a list of screen distances giving the positions of the tab stops. Each position may optionally be followed in the next list element by one of the keywords **left**, **right**, **center**, or **numeric**, which specifies how to justify text relative to the tab stop. **Left** is the default

7 **width** => Integer

Specifies the desired width for the window in units of characters. If the font doesn't have a uniform width then the width of the character "0" is used in translating from character units to screen units.

8 **wrap** => String

Specifies how to handle lines in the text that are too long to be displayed in a single line of the text's window. The value must be **none** or **char** or **word**.

Manipulating Test:

There are following useful methods to manipulate the content of an text:

- **delete*index*1, ?*index*2?:** Delete a range of characters from the text. If both index1 and index2 are specified, then delete all the characters starting with the one given by index1 and stopping just before index2. If *index2* doesn't specify a position later in the text than *index1* then no characters are deleted. If *index2* isn't specified then the single character at *index1* is deleted.

- **get*index*1, ?*index*2?**: Return a range of characters from the text. The return value will be all the characters in the text starting with the one whose index is *index1* and ending just before the one whose index is *index2* (the character at *index2* will not be returned). If *index2* is omitted then the single character at *index1* is returned.

- **index*index*** : Returns the position corresponding to *index* in the form *line.char* where *line* is the line number and *char* is the character number.

- **insert*index, chars, ?tagList, chars, tagList, . . . ?*** : Inserts all of the *chars* arguments just before the character at *index*. If *index* refers to the end of the text *thecharacterafterthelastnewline* then the new text is inserted just before the last newline instead. If there is a single *chars* argument and no *tagList*, then the new text will receive any tags that are present on both the character before and the character after the insertion point; if a tag is present on only one of these characters then it will not be applied to the new text. If *tagList* is specified then it consists of a list of tag names; the new characters will receive all of the tags in this list and no others, regardless of the tags present around the insertion point. If multiple *chars-tagList* argument pairs are present, they produce the same effect as if a separate **insert** widget command had been issued for each pair, in order. The last *tagList* argument may be omitted.

- **xview*option, args*** : This command is used to query and change the horizontal position of the text in the widget's window.

- **yview*?args?*** : This command is used to query and change the vertical position of the text in the widget's window.

Event Bindings:

Ruby/Tk automatically creates class bindings for texts. Here are few important bindings listed.

- Clicking mouse button 1 positions the insertion cursor just before the character underneath the mouse cursor, sets the input focus to this widget, and clears any selection in the widget. Dragging with mouse button 1 strokes out a selection between the insertion cursor and the character under the mouse.

- Double-clicking with mouse button 1 selects the word under the mouse and positions the insertion cursor at the beginning of the word. Dragging after a double click will stroke out a selection consisting of whole words.

- Triple-clicking with mouse button 1 selects the line under the mouse and positions the insertion cursor at the beginning of the line. Dragging after a triple click will stroke out a selection consisting of whole lines.

- Clicking mouse button 1 with the Control key down will reposition the insertion cursor without affecting the selection.
- The Left and Right keys move the insertion cursor one character to the left or right; they also clear any selection in the text.
- The Up and Down keys move the insertion cursor one line up or down and clear any selection in the text. If Up or Right is typed with the Shift key down, then the insertion cursor moves and the selection is extended to include the new character.
- Control-x deletes whatever is selected in the text widget.
- Control-o opens a new line by inserting a newline character in front of the insertion cursor without moving the insertion cursor.
- Control-d deletes the character to the right of the insertion cursor.

Examples:

```
require 'tk'

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

text = TkText.new(root) do
  width 30
  height 20
  borderwidth 1
  font TkFont.new('times 12 bold')
   pack("side" => "right",  "padx"=> "5", "pady"=> "5")
end
text.insert 'end', "Hello!\n\ntext widget example"
Tk.mainloop
```

This will produce the following result: