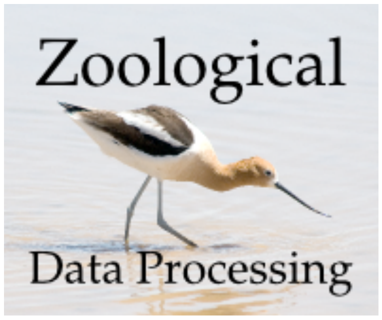
Next / Previous / Contents / Shipman's homepage

Tkinter 8.5 reference: a GUI for Python



10. The `Entry` widget

The purpose of an `Entry` widget is to let the user see and modify a *single* line of text.

- If you want to display *multiple* lines of text that can be edited, see Section 24, "The `Text` widget".
- If you want to display one or more lines of text that *cannot* be modified by the user, see Section 12, "The `Label` widget".

Some definitions:

- The *selection* is a highlighted region of the text in an `Entry` widget, if there is one.

  Typically the selection is made by the user with the mouse, and selected text is copied to the system's clipboard. However, *Tkinter* allows you to control whether or not selected text gets copied to the clipboard. You can also select text in an `Entry` under program control.

- The *insertion cursor* shows where new text will be inserted. It is displayed only when the user clicks the mouse somewhere in the widget. It usually appears as a blinking vertical line inside the widget. You can customize its appearance in several ways.

- Positions within the widget's displayed text are given as an *index*. There are several ways to specify an index:

- As normal Python indexes, starting from 0.
- The constant `tk.END` refers to the position after the existing text.
- The constant `tk.INSERT` refers to the current position of the insertion cursor.
- The constant `tk.ANCHOR` refers to the first character of the selection, if there is a selection.
- You may need to figure out which character position in the widget corresponds to a given mouse position. To simplify that process, you can use as an index a string of the form `'@n'`, where *n* is the horizontal distance in pixels between the left edge of the `Entry` widget and the mouse. Such an index will specify the character at that horizontal mouse position.

To create a new `Entry` widget in a root window or frame named *`parent`*:

```
    w = tk.Entry(parent, option, ...)
```

This constructor returns the new `Entry` widget. Options include:

Table 17. `Entry` widget options

`bg` or `background`	The background color inside the entry area. Default is a light gray.
`bd` or `borderwidth`	The width of the border around the entry area; see Section 5.1, “Dimensions”. The default is two pixels.
`cursor`	The cursor used when the mouse is within the entry widget; see Section 5.8, “Cursors”.

`disabledbackground`	The background color to be displayed when the widget is in the `tk.DISABLED` state. For option values, see `bg` above.
`disabledforeground`	The foreground color to be displayed when the widget is in the `tk.DISABLED` state. For option values, see `fg` below.
`exportselection`	By default, if you select text within an `Entry` widget, it is automatically exported to the clipboard. To avoid this exportation, use `exportselection=0`.
`fg` or `foreground`	The color used to render the text. Default is black.
`font`	The font used for text entered in the widget by the user. See Section 5.4, “Type fonts”.
`highlightbackground`	Color of the focus highlight when the widget does not have focus. See Section 53, “Focus: routing keyboard input”.
`highlightcolor`	Color shown in the focus highlight when the widget has the focus.
`highlightthickness`	Thickness of the focus highlight.
`insertbackground`	By default, the insertion cursor (which shows the point within the text where new keyboard input will be inserted) is black. To get a different color of insertion cursor, set `insertbackground` to any color; see Section 5.3, “Colors”.
`insertborderwidth`	By default, the insertion cursor is a simple rectangle. You can get the cursor with the `tk.RAISED` relief effect (see Section 5.6, “Relief styles”) by setting

	insertborderwidth to the dimension of the 3-d border. If you do, make sure that the insertwidth option is at least twice that value.
insertofftime	By default, the insertion cursor blinks. You can set insertofftime to a value in milliseconds to specify how much time the insertion cursor spends off. Default is 300. If you use insertofftime=0, the insertion cursor won't blink at all.
insertontime	Similar to insertofftime, this option specifies how much time the cursor spends on per blink. Default is 600 (milliseconds).
insertwidth	By default, the insertion cursor is 2 pixels wide. You can adjust this by setting insertwidth to any dimension.
justify	This option controls how the text is justified when the text doesn't fill the widget's width. The value can be tk.LEFT (the default), tk.CENTER, or tk.RIGHT.
readonlybackground	The background color to be displayed when the widget's state option is 'readonly'.
relief	Selects three-dimensional shading effects around the text entry. See Section 5.6, “Relief styles”. The default is relief=tk.SUNKEN.
selectbackground	The background color to use displaying selected text. See Section 5.3, “Colors”.
selectborderwidth	The width of the border to use around selected text. The default is one pixel.
selectforeground	The foreground (text) color of selected text.

`show`	Normally, the characters that the user types appear in the entry. To make a “password” entry that echoes each character as an asterisk, set `show='*'`.
`state`	Use this option to disable the `Entry` widget so that the user can't type anything into it. Use `state=tk.DISABLED` to disable the widget, `state=tk.NORMAL` to allow user input again. Your program can also find out whether the cursor is currently over the widget by interrogating this option; it will have the value `tk.ACTIVE` when the mouse is over it. You can also set this option to `'disabled'`, which is like the `tk.DISABLED` state, but the contents of the widget can still be selected or copied.
`takefocus`	By default, the focus will tab through entry widgets. Set this option to 0 to take the widget out of the sequence. For a discussion of focus, see Section 53, “Focus: routing keyboard input”.
`textvariable`	In order to be able to retrieve the current text from your entry widget, you must set this option to an instance of the `StringVar` class; see Section 52, “Control variables: the values behind the widgets”. You can retrieve the text using `*v*.get()`, or set it using `*v*.set()`, where *v* is the associated control variable.
`validate`	You can use this option to set up the widget so that its contents are checked by a validation function at certain times. See Section 10.2, “Adding validation to an `Entry` widget”.

`validatecommand`	A callback that validates the text of the widget. See Section 10.2, “Adding validation to an `Entry` widget”.
`width`	The size of the entry in characters. The default is 20. For proportional fonts, the physical length of the widget will be based on the average width of a character times the value of the `width` option.
`xscrollcommand`	If you expect that users will often enter more text than the onscreen size of the widget, you can link your entry widget to a scrollbar. Set this option to the `.set` method of the scrollbar. For more information, see Section 10.1, “Scrolling an `Entry` widget”.

Methods on `Entry` objects include:

`.delete(`*`first`*`, `*`last`*`=None)`

Deletes characters from the widget, starting with the one at index *`first`*, up to but *not* including the character at position *`last`*. If the second argument is omitted, only the single character at position *`first`* is deleted.

`.get()`

Returns the entry's current text as a string.

`.icursor(`*`index`*`)`

Set the insertion cursor just before the character at the given *`index`*.

`.index(`*`index`*`)`

Shift the contents of the entry so that the character at the given *`index`* is the leftmost visible character. Has no effect if the text fits entirely within the entry.

`.insert(`*`index`*`, `*`s`*`)`

Inserts string *`s`* before the character at the given *`index`*.

`.scan_dragto(`*`x`*`)`

See the `scan_mark` method below.

`.scan_mark(`*`x`*`)`

Use this option to set up fast scanning of the contents of the `Entry` widget that has a scrollbar that supports horizontal scrolling.

To implement this feature, bind the mouse's button-down event to a handler that calls `scan_mark(`*`x`*`)`, where *`x`* is the current mouse *`x`* position. Then bind the `<Motion>` event to a handler that calls `scan_dragto(`*`x`*`)`, where *`x`* is the current mouse *`x`* position. The `scan_dragto` method scrolls the contents of the `Entry` widget continuously at a rate proportional to the horizontal distance between the position at the time of the `scan_mark` call and the current position.

`.select_adjust(`*`index`*`)`

This method is used to make sure that the selection includes the character at the specified *`index`*. If the selection already includes that character, nothing happens. If not, the selection is expanded from its current position (if any) to include position *`index`*.

`.select_clear()`

Clears the selection. If there isn't currently a selection, has no effect.

`.select_from(`*`index`*`)`

Sets the `tk.ANCHOR` index position to the character selected by *`index`*, and selects that character.

`.select_present()`

If there is a selection, returns true, else returns false.

`.select_range(`*`start`*`, `*`end`*`)`

Sets the selection under program control. Selects the text starting at the *`start`* *`index`*, up to but *not* including the character at the *`end`* index. The *`start`* position must be before the *`end`* position.

To select all the text in an entry widget `e`, use `e.select_range(0, tk.END)`.

`.select_to(`*`index`*`)`

Selects all the text from the `tk.ANCHOR` position up to but not including the character at the given *`index`*.

`.xview(`*`index`*`)`

Same as `.xview()`. This method is useful in linking the `Entry` widget to a horizontal scrollbar. See Section 10.1, "Scrolling an `Entry` widget".

`.xview_moveto(`*`f`*`)`

Positions the text in the entry so that the character at position *`f`*, relative to the entire text, is positioned at the left edge of the window. The *`f`* argument must be in the range [0,1], where 0 means the left end of the text and 1 the right end.

`.xview_scroll(`*`number`*`, `*`what`*`)`

Used to scroll the entry horizontally. The *`what`* argument must be either `tk.UNITS`, to scroll by character widths, or `tk.PAGES`, to scroll by chunks the size of the entry widget. The *`number`* is positive to scroll left to right, negative to scroll right to left. For example, for an entry widget `e`, `e.xview_scroll(-1, tk.PAGES)` would move the text one “page” to the right, and `e.xview_scroll(4, tk.UNITS)` would move the text four characters to the left.

10.1. Scrolling an `Entry` widget

Making an `Entry` widget scrollable requires a little extra code on your part to adapt the `Scrollbar` widget's callback to the methods available on the `Entry` widget. Here are some code fragments illustrating the setup. First, the creation and linking of the `Entry` and `Scrollbar` widgets:

```
    self.entry = tk.Entry(self, width=10)
    self.entry.grid(row=0, sticky=tk.E+tk.W)

    self.entryScroll = tk.Scrollbar(self, orient=tk.HORIZONTAL,
        command=self.__scrollHandler)
    self.entryScroll.grid(row=1, sticky=tk.E+tk.W)
    self.entry['xscrollcommand'] = self.entryScroll.set
```

Here's the adapter function referred to above:

```
    def __scrollHandler(self, *L):
        op, howMany = L[0], L[1]

        if op == 'scroll':
            units = L[2]
            self.entry.xview_scroll(howMany, units)
        elif op == 'moveto':
            self.entry.xview_moveto(howMany)
```

Next: 10.2. Adding validation to an `Entry` widget
Contents: *Tkinter* 8.5 reference: a GUI for Python
Previous: 9. The `Checkbutton` widget
Home: John Shipman's Home Sweet Homepage

John W. Shipman
Comments welcome: john@nmt.edu

Last updated: 2013-09-09 22:58
URL: `http://www.nmt.edu/~shipman/soft/tkinter/web/entry.html`