Tkinter 8.5 reference: a GUI for  
Python

John W. Shipman

2013-12-31 17:59

Abstract

Describes the Tkinter widget set for constructing graphical user interfaces (GUIs) in the Python  
programming language. Includes coverage of the ttk themed widgets.  
This publication is available in Web form1 and also as a PDF document2. Please forward any

comments to tcc-doc@nmt.edu.

Table of Contents

1. A cross-platform graphical user interface builder for Python . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3  
2. A minimal application . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4  
3. Definitions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5  
4. Layout management . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5  
4.1. The .grid() method . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6  
4.2. Other grid management methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7  
4.3. Configuring column and row sizes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7  
4.4. Making the root window resizeable . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8  
5. Standard attributes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9  
5.1. Dimensions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9  
5.2. The coordinate system . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10  
5.3. Colors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10  
5.4. Type fonts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10  
5.5. Anchors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12  
5.6. Relief styles . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12  
5.7. Bitmaps . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12  
5.8. Cursors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13  
5.9. Images . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14  
5.10. Geometry strings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15  
5.11. Window names . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16  
5.12. Cap and join styles . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16  
5.13. Dash patterns . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17  
5.14. Matching stipple patterns . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17  
6. Exception handling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18  
7. The Button widget . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18  
8. The Canvas widget . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20  
8.1. Canvas coordinates . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22  
8.2. The Canvas display list . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22  
8.3. Canvas object IDs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

1 http://www.nmt.edu/tcc/help/pubs/tkinter/  
2 http://www.nmt.edu/tcc/help/pubs/tkinter/tkinter.pdf

New Mexico Tech Computer Center

Tkinter 8.5 reference

1

8.4. Canvas tags . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22  
8.5. Canvas tagOrId arguments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22  
8.6. Methods on Canvas widgets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22  
8.7. Canvas arc objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28  
8.8. Canvas bitmap objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29  
8.9. Canvas image objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30  
8.10. Canvas line objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30  
8.11. Canvas oval objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32  
8.12. Canvas polygon objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33  
8.13. Canvas rectangle objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35  
8.14. Canvas text objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37  
8.15. Canvas window objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38  
9. The Checkbutton widget . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38  
10. The

28.1. Importing ttk . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 109  
28.2. The ttk widget set . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 110  
29. ttk.Button . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 110  
30. ttk.Checkbutton . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112  
31. ttk.Combobox . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115  
32. ttk

Pertinent references:  
• Fredrik Lundh, who wrote Tkinter, has two versions of his An Introduction to Tkinter: a more complete  
1999 version3 and a 2005 version4 that presents a few newer features.  
• Python 2.7 quick reference5: general information about the Python language.  
• For an example of a sizeable working application (around 1000 lines of code), see huey: A color and  
font selection tool6. The design of this application demonstrates how to build your own compound  
widgets.  
We'll start by looking at the visible part of Tkinter: creating the widgets and arranging them on the  
screen. Later we will talk about how to connect the face—the “front panel”—of the application to the  
logic behind it.

2. A minimal application

Here is a trivial Tkinter program containing only a Quit button:

#!/usr/bin/env python  
import Tkinter as tk

class Application(tk.Frame):  
def \_\_init\_\_(self, master=None):  
tk.Frame.\_\_init\_\_(self, master)  
self.grid()  
self.createWidgets()

def createWidgets(self):  
self.quitButton = tk.Button(self, text='Quit',  
command=self.quit)  
self.quitButton.grid()

app = Application()  
app.master.title('Sample application')  
app.mainloop()

This line makes the script self-executing, assuming that your system has Python correctly installed.  
This line imports the Tkinter module into your program's namespace, but renames it as tk.  
Your application class must inherit from Tkinter's Frame class.  
Calls the constructor for the parent class, Frame.  
Necessary to make the application actually appear on the screen.  
Creates a button labeled “Quit”.  
Places the button on the application.  
The main program starts here by instantiating the Application class.  
This method call sets the title of the window to “Sample application”.  
Starts the application's main loop, waiting for mouse and keyboard events.

3 http://www.pythonware.com/library/tkinter/introduction/  
4 http://effbot.org/tkinterbook/  
5 http://www.nmt.edu/tcc/help/pubs/python/  
6 http://www.nmt.edu/tcc/help/lang/python/examples/huey/

4

Tkinter 8.5 reference

New Mexico Tech Computer Center

3. Definitions

Before we proceed, let's define some of the common terms.  
window  
This term has different meanings in different contexts, but in general it refers to a rectangular area  
somewhere on your display screen.  
top-level window  
A window that exists independently on your screen. It will be decorated with the standard frame  
and controls for your system's desktop manager. You can move it around on your desktop. You  
can generally resize it, although your application can prevent this  
widget  
The generic term for any of the building blocks that make up an application in a graphical user in-  
terface. Examples of widgets: buttons, radiobuttons, text fields, frames, and text labels.  
frame  
In Tkinter, the Frame widget is the basic unit of organization for complex layouts. A frame is a  
rectangular area that can contain other widgets.  
child, parent  
When any widget is created, a parent-child relationship is created. For example, if you place a text  
label inside a frame, the frame is the parent of the label.

4. Layout management

Later we will discuss the widgets, the building blocks of your GUI application. How do widgets get  
arranged in a window?  
Although there are three different “geometry managers” in Tkinter, the author strongly prefers the  
.grid() geometry manager for pretty much everything. This manager treats every window or frame  
as a table—a gridwork of rows and columns.  
• A cell is the area at the intersection of one row and one column.  
• The width of each column is the width of the widest cell in that column.  
• The height of each row is the height of the largest cell in that row.  
• For widgets that do not fill the entire cell, you can specify what happens to the extra space. You can  
either leave the extra space outside the widget, or stretch the widget to fit it, in either the horizontal  
or vertical dimension.  
• You can combine multiple cells into one larger area, a process called spanning.  
When you create a widget, it does not appear until you register it with a geometry manager. Hence,  
construction and placing of a widget is a two-step process that goes something like this:

self.thing = tk.Constructor(parent, ...)  
self.thing.grid(...)

where Constructor is one of the widget classes like Button, Frame, and so on, and parent is the  
parent widget in which this child widget is being constructed. All widgets have a .grid() method  
that you can use to tell the geometry manager where to put it.

New Mexico Tech Computer Center

Tkinter 8.5 reference

5

4.1. The .grid() method

To display a widget w on your application screen:

w.grid(option=value, ...)

This method registers a widget w with the grid geometry manager—if you don't do this, the widget will  
exist internally, but it will not be visible on the screen. For the options, see Table 1, “Arguments of the  
.grid() geometry manager” (p. 6).

Table 1. Arguments of the .grid() geometry manager

column

columnspan

in\_

ipadx

The column number where you want the widget gridded, counting from zero. The default  
value is zero.  
Normally a widget occupies only one cell in the grid. However, you can grab multiple  
cells of a row and merge them into one larger cell by setting the columnspan option to

the number of cells. For example, w.grid(row=0, column=2, columnspan=3)

would place widget w in a cell that spans columns 2, 3, and 4 of row 0.  
To register w as a child of some widget w2, use in\_=w2. The new parent w2 must be a  
descendant of the parent widget used when w was created.  
Internal x padding. This dimension is added inside the widget inside its left and right  
sides.  
Internal y padding. This dimension is added inside the widget inside its top and bottom  
borders.

• The other combinations will also work. For example, sticky=tk.N+tk.S+tk.W will stretch the  
widget vertically and place it against the west (left) wall.

4.2. Other grid management methods

These grid-related methods are defined on all widgets:

w.grid\_bbox(column=None, row=None, col2=None, row2=None)

Returns a 4-tuple describing the bounding box of some or all of the grid system in widget w. The  
first two numbers returned are the x and

w.columnconfigure(N, option=value, ...)

In the grid layout inside widget w, configure column N so that the given option has the given  
value. For options, see the table below.

w.rowconfigure(N, option=value, ...)

In the grid layout inside widget w, configure row N so that the given option has the given value.  
For options, see the table below.  
Here are the options used for configuring column and row sizes.

Table 2. Column and row configuration options for the .grid() geometry manager

minsize

pad

weight

The column or row's minimum size in pixels. If there is nothing in the given column or  
row, it will not appear, even if you use this option.  
A number of pixels that will be added to the given column or row, over and above the  
largest cell in the column or row.  
To make a column or row stretchable, use this option and supply a value that gives the  
relative weight of this column or row when distributing the extra space. For example, if  
a widget w contains a grid layout, these lines will distribute three-fourths of the extra  
space to the first column and one-fourth to the second column:

w.columnconfigure(0, weight=3)  
w.columnconfigure(1, weight=1)

If this option is not used, the column or row will not stretch.

4.4. Making the root window resizeable

Do you want to let the user resize your entire application window, and distribute the extra space among  
its internal widgets? This requires some operations that are not obvious.  
It's necessary to use the techniques for row and column size management, described in Section 4.3,  
“Configuring column and row sizes” (p. 7), to make your Application widget's grid stretchable.  
However, that alone is not sufficient.  
Consider the trivial application discussed in Section 2, “A minimal application” (p. 4), which contains  
only a Quit button. If you run this application, and resize the window, the button stays the same size,  
centered within the window.  
Here is a replacement version of the .\_\_createWidgets() method in the minimal application. In  
this version, the Quit button always fills all the available space.

def createWidgets(self):  
top=self.winfo\_toplevel()  
top.rowconfigure(0, weight=1)  
top.columnconfigure(0, weight=1)  
self.rowconfigure(0, weight=1)  
self.columnconfigure(0, weight=1)  
self.quit = Button(self, text='Quit', command=self.quit)  
self.quit.grid(row=0, column=0,  
sticky=tk.N+tk.S+tk.E+tk.W)

The “top level window” is the outermost window on the screen. However, this window is not your  
Application window—it is the parent of the Application instance. To get the top-level window,

8

Tkinter 8.5 reference

New Mexico Tech Computer Center

call the .winfo\_toplevel() method on any widget in your application; see Section 26, “Universal  
widget methods” (p. 97).  
This line makes row 0 of the top level window's grid stretchable.  
This line makes column 0 of the top level window's grid stretchable.  
Makes row 0 of the Application widget's grid stretchable.  
Makes column 0 of the Application widget's grid stretchable.  
The argument sticky=tk.N+tk.S+tk.E+tk.W makes the button expand to fill its cell of the  
grid.  
There is one more change that must be made. In the constructor, change the second line as shown:

def \_\_init\_\_(self, master=None):  
tk.Frame.\_\_init\_\_(self, master)  
self.grid(sticky=tk.N+tk.S+tk.E+tk.W)  
self.createWidgets()

The argument sticky=tk.N+tk.S+tk.E+tk.aW to self.grid() is necessary so that the Applic-

ation widget will expand to fill its cell of the top-level window's grid.

5. Standard attributes

Before we look at the widgets, let's take a look at how some of their common attributes—such as sizes,  
colors and fonts—are specified.  
• Each widget has a set of options that affect its appearance and behavior—attributes such as fonts,  
colors, sizes, text labels, and such.  
• You can specify options when calling the widget's constructor using keyword arguments such as

text='PANIC!' or height=20.

• After you have created a widget, you can later change any option by using the widget's .config()  
method. You can retrieve the current setting of any option by using the widget's .cget() method.  
See Section 26, “Universal widget methods” (p. 97) for more on these methods.

5.1. Dimensions

Various lengths, widths, and other dimensions of widgets can be described in many different units.  
• If you set a dimension to an integer, it is assumed to be in pixels.  
• You can specify units by setting a dimension to a string containing a number followed by:

Table 3. Dimensional units

c  
i  
m  
p

Centimeters  
Inches  
Millimeters  
Printer's points (about 1/72″)

New Mexico Tech Computer Center

Tkinter 8.5 reference

9

5.2. The coordinate system

As in most contemporary display systems, the origin of each coordinate system is at its upper left corner,  
with the x coordinate increasing toward the right, and the y coordinate increasing toward the bottom:

The base unit is the pixel, with the top left pixel having coordinates (0,0). Coordinates that you specify  
as integers are always expressed in pixels, but any coordinate may be specified as a dimensioned  
quantity; see Section 5.1, “Dimensions” (p. 9).

5.3. Colors

There are two general ways to specify colors in Tkinter.  
• You can use a string specifying the proportion of red, green, and blue in hexadecimal digits:

#rgb  
#rrggbb  
#rrrgggbbb

Four bits per color  
Eight bits per color  
Twelve bits per color

For example, '#fff' is white, '#000000' is black, '#000fff000' is pure green, and '#00ffff'  
is pure cyan (green plus blue).  
• You can also use any locally defined standard color name. The colors 'white', 'black', 'red',  
'green', 'blue', 'cyan', 'yellow', and 'magenta' will always be available. Other names may  
work, depending on your local installation.

5.4. Type fonts

Depending on your platform, there may be up to three ways to specify type style.  
• As a tuple whose first element is the font family, followed by a size (in points if positive, in pixels if  
negative), optionally followed by a string containing one or more of the style modifiers bold, italic,

underline, and overstrike.

Examples: ('Helvetica', '16') for a 16-point Helvetica regular; ('Times', '24', 'bold  
italic') for a 24-point Times bold italic. For a 20-pixel Times bold font, use ('Times', -20,

'bold').

• You can create a “font object” by importing the tkFont module and using its Font class constructor:

import tkFont

font = tkFont.Font(option, ...)

where the options include:

The font family name as a string.

size  
weight  
slant  
underline  
overstrike

The font height as an integer in points. To get a font n pixels high, use -n.  
'bold' for boldface, 'normal' for regular weight.  
'italic' for italic, 'roman' for unslanted.  
1 for underlined text, 0 for normal.  
1 for overstruck text, 0 for normal.

For example, to get a 36-point bold Helvetica italic face:

helv36 = tkFont.Font(family='Helvetica',  
size=36, weight='bold')

• If you are running under the X Window System, you can use any of the X font names. For example,

the font named '-\*-lucidatypewriter-medium-r-\*-\*-\*-140-\*-\*-\*-\*-\*-\*' is a good

fixed-width font for onscreen use. Use the xfontsel program to help you select pleasing fonts.  
To get a list of all the families of fonts available on your platform, call this function:

tkFont.families()

The return value is a list of strings. Note: You must create your root window before calling this function.  
These methods are defined on all Font objects:

.actual(option=None)

If you pass no arguments, you get back a dictionary of the font's actual attributes, which may differ  
from the ones you requested. To get back the value of an attribute, pass its name as an argument.

.cget(option)

Returns the value of the given option.

.configure(option, ...)

Use this method to change one or more options on a font. For example, if you have a Font object

called titleFont, if you call titleFont.configure(family='times', size=18), that

font will change to 18pt Times and any widgets that use that font will change too.

.copy()

Returns a copy of a Font object.

.measure(text)

Pass this method a string, and it will return the number of pixels of width that string will take in  
the font. Warning: some slanted characters may extend outside this area.

.metrics(option)

If you call this method with no arguments, it returns a dictionary of all the font metrics. You can re-  
trieve the value of just one metric by passing its name as an argument. Metrics include:

ascent  
descent  
fixed  
linespace

Number of pixels of height between the baseline and the top of the highest ascender.  
Number of pixels of height between the baseline and the bottom of the lowest ascender.  
This value is 0 for a variable-width font and 1 for a monospaced font.  
Number of pixels of height total. This is the leading of type set solid in the given font.

New Mexico Tech Computer Center

Tkinter 8.5 reference

11

5.5. Anchors

The Tkinter module defines a number of anchor constants that you can use to control where items are  
positioned relative to their context. For example, anchors can specify where a widget is located inside  
a frame when the frame is bigger than the widget.  
These constants are given as compass points, where north is up and west is to the left. We apologize to  
our Southern Hemisphere readers for this Northern Hemisphere chauvinism7.  
The anchor constants are shown in this diagram:

For example, if you create a small widget inside a large frame and use the anchor=tk.SE option, the  
widget will be placed in the bottom right corner of the frame. If you used anchor=tk.N instead, the  
widget would be centered along the top edge.  
Anchors are also used to define where text is positioned relative to a reference point. For example, if  
you use tk.CENTER as a text anchor, the text will be centered horizontally and vertically around the  
reference point. Anchor tk.NW will position the text so that the reference point coincides with the  
northwest (top left) corner of the box containing the text. Anchor tk.W will center the text vertically  
around the reference point, with the left edge of the text box passing through that point, and so on.

5.6. Relief styles

The relief style of a widget refers to certain simulated 3-D effects around the outside of the widget. Here  
is a screen shot of a row of buttons exhibiting all the possible relief styles:

The width of these borders depends on the borderwidth option of the widget. The above graphic  
shows what they look like with a 5-pixel border; the default border width is 2.

5.7. Bitmaps

For bitmap options in widgets, these bitmaps are guaranteed to be available:

7 http://flourish.org/upsidedownmap/

12

Tkinter 8.5 reference

New Mexico Tech Computer Center

The graphic above shows Button widgets bearing the standard bitmaps. From left to right, they are

'error', 'gray75', 'gray50', 'gray25', 'gray12', 'hourglass', 'info', 'questhead',  
'question', and 'warning'.

You can use your own bitmaps. Any file in .xbm (X bit map) format will work. In place of a standard  
bitmap name, use the string '@' followed by the pathname of the .xbm file.

5.8. Cursors

There are quite a number of different mouse cursors available. Their names and graphics are shown  
here. The exact graphic may vary according to your operating system.

Table 4. Values of the cursor option

arrow

man

based\_arrow\_down

middlebutton

based\_arrow\_up

boat

mouse

pencil

pirate

dotbox

double\_arrow

draft\_large

draft\_small

draped\_box

exchange

fleur

gobbler

shuttle

sizing

spider

spraycan

star

target

tcross

top\_left\_arrow

top\_left\_corner

• The Python Imaging Library (PIL) supports images in a much wider variety of formats. Its ImageTk  
class is specifically designed for displaying images within Tkinter applications. See the author's  
companion document for PIL documentation: Python Imaging Library (PIL) quick reference8.

5.9.1. The BitmapImage class  
To display a two-color image in the .xbm format, you will need this constructor:

tk.BitmapImage(file=f[, background=b][, foreground=c])

where f is the name of the .xbm image file.  
Normally, foreground (1) bits in the image will be displayed as black pixels, and background (0) bits  
in the image will be transparent. To change this behavior, use the optional background=b option to  
set the background to color b, and the optional foreground=c option to set the foreground to color  
c. For color specification, see Section 5.3, “Colors” (p. 10).  
This constructor returns a value that can be used anywhere Tkinter expects an image. For example, to  
display an image as a label, use a Label widget (see Section 12, “The Label widget” (p. 48)) and  
supply the BitmapImage object as the value of the image option:

logo = tk.BitmapImage('logo.xbm', foreground='red')  
Label(image=logo).grid()

5.9.2. The PhotoImage class  
To display a color image in .gif, .pgm, or .ppm format, you will need this constructor:

tk.PhotoImage(file=f)

where f is the name of the image file. The constructor returns a value that can be used anywhere Tkinter  
expects an image.

5.10. Geometry strings

A geometry string is a standard way of describing the size and location of a top-level window on a  
desktop.  
A geometry string has this general form:

'wxh±x±

For example, a window created with geometry='120x50-0+20' would be 120 pixels wide by 50  
pixels high, and its top right corner will be along the right edge of the desktop and 20 pixels below the  
top edge.

5.11. Window names

The term window describes a rectangular area on the desktop.  
• A top-level or root window is a window that has an independent existence under the window manager.  
It is decorated with the window manager's decorations, and can be moved and resized independently.  
Your application can use any number of top-level windows.  
• The term “window” also applies to any widget that is part of a top-level window.  
Tkinter names all these windows using a hierarchical window path name.  
• The root window's name is '.'.  
• Child windows have names of the form '.n', where n is some integer in string form. For example,  
a window named '.135932060' is a child of the root window ('.').  
• Child windows within child windows have names of the form 'p.n' where p is the name of the  
parent window and n is some integer. For example, a window named '.135932060.137304468'  
has parent window '.135932060', so it is a grandchild of the root window.  
• The relative name of a window is the part past the last '.' in the path name. To continue the previous  
example, the grandchild window has a relative name '137304468'.  
To get the path name for a widget w, use str(w).  
See also Section 26, “Universal widget methods” (p. 97) for methods you can use to operate on window

names, especially the .winfo\_name, .winfo\_parent, and .winfo\_pathname methods.

5.12. Cap and join styles

For pleasant and effective rendering of diagrams, sometimes it is a good idea to pay attention to cap  
and join styles.  
• The cap style of a line is the shape of the end of the line. Styles are:  
• tk.BUTT: The end of the line is cut off square at a line that passes through the endpoint.  
• tk.PROJECTING: The end of the line is cut off square, but the cut line projects past the endpoint  
a distance equal to half the line's width.  
• tk.ROUND: The end describes a semicircle centered on the endpoint.  
• The join style describes the shape where two line segments meet at an angle.  
• tk.ROUND: The join is a circle centered on the point where the adjacent line segments meet.  
• tk.BEVEL: A flat facet is drawn at an angle intermediate between the angles of the adjacent lines.  
• tk.MITER: The edges of the adjacent line segments are continued to meet at a sharp point.  
This illustration shows how Tkinter's cap and join options work with a line made of two connected line  
segments. Small red circles show the location of the points that define this line.

16

Tkinter 8.5 reference

New Mexico Tech Computer Center

5.13. Dash patterns

A number of widgets allow you to specify a dashed outline. The dash and dashoffset options give  
you fine control over the exact pattern of the dashes.

dash

This option is specified as a tuple of integers. The first integer specifies how many pixels should be  
drawn. The second integer specifies how many pixels should be skipped before starting to draw  
again, and so on. When all the integers in the tuple are exhausted, they are reused in the same order  
until the border is complete.  
For example, dash=(3,5) produces alternating 3-pixel dashes separated by 5-pixel gaps. A value  
of dash=(7,1,1,1) produces a dash-and-dot pattern, with the dash seven times as long as the  
dot or the gaps around the dot. A value of dash=(5,) produces alternating five-pixel dashes and  
five-pixel gaps.

dashoff

To start the dash pattern in a different point of cycle instead of at the beginning, use an option of  
dashoff=n, where n is the number of pixels to skip at the beginning of the pattern.  
For example, for options dash=(5, 1, 2, 1) and dashoff=3, the first pattern produced will  
be: 2 on, 1 off, 2 on, and 1 off. Subsequent patterns will be 5 on, 1 off, 2 on, and 1 off. Here is a screen  
shot of a line drawn with this combination of options:

5.14. Matching stipple patterns

This may seem like an incredibly picky style point, but if you draw a graphic that has two objects with  
stippled patterns, a real professional will make sure that the patterns align along their boundary.

New Mexico Tech Computer Center

Tkinter 8.5 reference

17

Here is an example. The left-hand screen shot shows two adjacent 100×100 squares stippled with the  
“gray12” pattern, but the right-hand square is offset vertically by one pixel. The short black line in the  
center of the figure is drawn along the boundary of the two figures.

The second screen shot is the same, except that the two 100×100 squares have their stipple patterns lined  
up.  
In practice, this arises in two situations. The alignment of large stippled areas is controlled by an option  
named offset. For figures with stippled outlines, the outlineoffset option controls their alignment.  
Both options have values of one of these forms:

• 'x,

bd or borderwidth

bg or background  
bitmap

command  
cursor  
default

disabledforeground  
fg or foreground  
font  
height  
highlightbackground  
highlightcolor  
highlightthickness  
image

Width of the border around the outside of the button; see Section 5.1, “Di-  
mensions” (p. 9). The default is two pixels.  
Normal background color.  
Name of one of the standard bitmaps to display on the button (instead of  
text).  
Function or method to be called when the button is clicked.  
Selects the cursor to be shown when the mouse is over the button.  
tk.NORMAL is the default; use tk.DISABLED if the button is to be initially  
disabled (grayed out, unresponsive to mouse clicks).  
Foreground color used when the button is disabled.  
Normal foreground (text) color.  
Text font to be used for the button's label.  
Height of the button in text lines (for textual buttons) or pixels (for images).  
Color of the focus highlight when the widget does not have focus.  
The color of the focus highlight when the widget has focus.  
Thickness of the focus highlight.  
Image to be displayed on the button (instead of text).  
How to show multiple text lines: tk.LEFT to left-justify each line;  
tk.CENTER to center them; or tk.RIGHT to right-justify.

text

textvariable

underline

width

wraplength

Text displayed on the button. Use internal newlines to display multiple text  
lines.  
An instance of StringVar() that is associated with the text on this button.  
If the variable is changed, the new value will be displayed on the button.  
See Section 52, “Control variables: the values behind the widgets” (p. 153).  
Default is -1, meaning that no character of the text on the button will be  
underlined. If nonnegative, the corresponding text character will be under-  
lined. For example, underline=1 would underline the second character  
of the button's text.  
Width of the button in letters (if displaying text) or pixels (if displaying an  
image).  
If this value is set to a positive number, the text lines will be wrapped to fit  
within this length. For possible values, see Section 5.1, “Dimensions” (p. 9).

Methods on Button objects:

.flash()

Causes the button to flash several times between active and normal colors. Leaves the button in the  
state it was in originally. Ignored if the button is disabled.

.invoke()

Calls the button's command callback, and returns what that function returns. Has no effect if the  
button is disabled or there is no callback.

8. The Canvas widget

A canvas is a rectangular area intended for drawing pictures or other complex layouts. On it you can  
place graphics, text, widgets, or frames. See the following sections for methods that create objects on  
canvases:  
• .create\_arc(): A slice out of an ellipse. See Section 8.7, “Canvas arc objects” (p. 28).  
• .create\_bitmap(): An image as a bitmap. See Section 8.8, “Canvas bitmap objects” (p. 29).  
• .create\_image(): A graphic image. See Section 8.9, “Canvas image objects” (p. 30).  
• .create\_line(): One or more line segments. See Section 8.10, “Canvas line objects” (p. 30).  
• .create\_oval(): An ellipse; use this also for drawing circles, which are a special case of an ellipse.  
See Section 8.11, “Canvas oval objects” (p. 32).  
• .create\_polygon(): A polygon. See Section 8.12, “Canvas polygon objects” (p. 33).  
• .create\_rectangle(): A rectangle. See Section 8.13, “Canvas rectangle objects” (p. 35).  
• .create\_text(): Text annotation. See Section 8.14, “Canvas text objects” (p. 37).  
• .create\_window(): A rectangular window. See Section 8.15, “Canvas window objects” (p. 38).  
To create a Canvas object:

w = tk.Canvas(parent, option=value, ...)

The constructor returns the new Canvas widget. Supported options include:

20

Tkinter 8.5 reference

New Mexico Tech Computer Center

Table 6. Canvas widget options

bd or borderwidth

bg or background  
closeenough

confine

cursor  
height  
highlightback-  
ground  
highlightcolor  
highlightthickness  
relief

scrollregion

selectbackground  
selectborderwidth  
selectforeground  
takefocus

width  
xscrollincrement

xscrollcommand

yscrollincrement  
yscrollcommand

Width of the border around the outside of the canvas; see Section 5.1, “Di-  
mensions” (p. 9). The default is two pixels.  
Background color of the canvas. Default is a light gray, about '#E4E4E4'.  
A float that specifies how close the mouse must be to an item to be con-  
sidered inside it. Default is 1.0.  
If true (the default), the canvas cannot be scrolled outside of the scrollre-  
gion (see below).  
Cursor used in the canvas. See Section 5.8, “Cursors” (p. 13).  
Size of the canvas in the Y dimension. See Section 5.1, “Dimensions” (p. 9).  
Color of the focus highlight when the widget does not have focus. See Sec-  
tion 53, “Focus: routing keyboard input” (p. 155).  
Color shown in the focus highlight.  
Thickness of the focus highlight. The default value is 1.  
The relief style of the canvas. Default is tk.FLAT. See Section 5.6, “Relief  
styles” (p. 12).  
A tuple (w, n, e, s) that defines over how large an area the canvas can  
be scrolled, where w is the left side, n the top, e the right side, and s the bot-  
tom.  
The background color to use displaying selected items.  
The width of the border to use around selected items.  
The foreground color to use displaying selected items.  
Normally, focus (see Section 53, “Focus: routing keyboard input” (p. 155))  
will cycle through this widget with the tab key only if there are keyboard  
bindings set for it (see Section 54, “Events” (p. 157) for an overview of key-  
board bindings). If you set this option to 1, focus will always visit this widget.  
Set it to '' to get the default behavior.  
Size of the canvas in the X dimension. See Section 5.1, “Dimensions” (p. 9).  
Normally, canvases can be scrolled horizontally to any position. You can get  
this behavior by setting xscrollincrement to zero. If you set this option  
to some positive dimension, the canvas can be positioned only on multiples  
of that distance, and the value will be used for scrolling by scrolling units,  
such as when the user clicks on the arrows at the ends of a scrollbar. For more  
information on scrolling units, see Section 22, “The Scrollbar wid-  
get” (p. 74).  
If the canvas is scrollable, set this option to the .set() method of the hori-  
zontal scrollbar.  
Works like xscrollincrement, but governs vertical movement.  
If the canvas is scrollable, this option should be the .set() method of the  
vertical scrollbar.

New Mexico Tech Computer Center

Tkinter 8.5 reference

21

8.1. Canvas coordinates

Because the canvas may be larger than the window, and equipped with scrollbars to move the overall  
canvas around in the window, there are two coordinate systems for each canvas:  
• The window coordinates of a point are relative to the top left corner of the area on the display where  
the canvas appears.  
• The canvas coordinates of a point are relative to the top left corner of the total canvas.

8.2. The Canvas display list

The display list refers to the sequence of all the objects on the canvas, from background (the “bottom”  
of the display list) to foreground (the “top”).  
If two objects overlap, the one above the other in the display list means the one closer to the foreground,  
which will appear in the area of overlap and obscure the one below. By default, new objects are always  
created at the top of the display list (and hence in front of all other objects), but you can re-order the  
display list.

8.3. Canvas object IDs

The object ID of an object on the canvas is the value returned by the constructor for that object. All object  
ID values are simple integers, and the object ID of an object is unique within that canvas.

8.4. Canvas tags

A tag is a string that you can associate with objects on the canvas.  
• A tag can be associated with any number of objects on the canvas, including zero.  
• An object can have any number of tags associated with it, including zero.  
Tags have many uses. For example, if you are drawing a map on a canvas, and there are text objects for  
the labels on rivers, you could attach the tag 'riverLabel' to all those text objects. This would allow  
you to perform operations on all the objects with that tag, such as changing their color or deleting them.

8.5. Canvas tagOrId arguments

A tagOrId argument specifies one or more objects on the canvas.  
• If a tagOrId argument is an integer, it is treated as an object ID, and it applies only to the unique  
object with that ID. See Section 8.3, “Canvas object IDs” (p. 22).  
• If such an argument is a string, it is interpreted as a tag, and selects all the objects that have that tag  
(if there are any). See Section 8.4, “Canvas tags” (p. 22).

8.6. Methods on Canvas widgets

All Canvas objects support these methods:

.addtag\_above(newTag, tagOrId)

Attaches a new tag to the object just above the one specified by tagOrId in the display list. The  
newTag argument is the tag you want to attach, as a string.

22

Tkinter 8.5 reference

New Mexico Tech Computer Center

.addtag\_all(newTag)

Attaches the given tag newTag to all the objects on the canvas.

.addtag\_below(newTag, tagOrID)

Attaches a new tag to the object just below the one specified by tagOrId in the display list. The  
newTag argument is a tag string.

.addtag\_closest(newTag, x,

.find\_above(tagOrId)

Returns the ID number of the object just above the object specified by tagOrId. If multiple objects  
match, you get the highest one. Returns an empty tuple if you pass it the object ID of the highest  
object.

.find\_all()

Returns a list of the object ID numbers for all objects on the canvas, from lowest to highest.

.find\_below(tagOrId)

Returns the object ID of the object just below the one specified by tagOrId. If multiple objects  
match, you get the lowest one. Returns an empty tuple if you pass it the object ID of the lowest object.

.find\_closest(x,

• tk.SEL\_LAST, to return the position after the end of the current text selection, or raise  
tk.TclError if the item does not currently contain the selection.  
• A string of the form “@x,

.scan\_dragto(x,

.tag\_lower(tagOrId, belowThis)

Moves the object or objects selected by tagOrId within the display list to a position just below the  
first or only object specied by the tag or ID belowThis.  
If there are multiple items with tag tagOrId, their relative stacking order is preserved.  
This method does not affect canvas window items. To change a window item's stacking order, use  
a lower or lift method on the window.

.tag\_raise(tagOrId, aboveThis)

Moves the object or objects selected by tagOrId within the display list to a position just above the  
first or only object specied by the tag or ID aboveThis.  
If there are multiple items with tag tagOrId, their relative stacking order is preserved.  
This method does not affect canvas window items. To change a window item's stacking order, use  
a lower or lift method on the window.

.tag\_unbind(tagOrId, sequence, funcId=None)

Removes bindings for handler funcId and event sequence from the canvas object or objects  
specified by tagOrId. See Section 54, “Events” (p. 157).

.type(tagOrId)

Returns the type of the first or only object specified by tagOrId. The return value will be one of

the strings 'arc', 'bitmap', 'image', 'line', 'oval', 'polygon', 'rectangle', 'text',  
or 'window'.  
.xview(tk.MOVETO, fraction)

This method scrolls the canvas relative to its image, and is intended for binding to the command  
option of a related scrollbar. The canvas is scrolled horizontally to a position given by offset,  
where 0.0 moves the canvas to its leftmost position and 1.0 to its rightmost position.

.xview(tk.SCROLL, n, what)

This method moves the canvas left or right: the what argument specifies how much to move and  
can be either tk.UNITS or tk.PAGES, and n tells how many units to move the canvas to the right  
relative to its image (or left, if negative).  
The size of the move for tk.UNITS is given by the value of the canvas's xscrollincrement option;  
see Section 22, “The Scrollbar widget” (p. 74).  
For movements by tk.PAGES, n is multiplied by nine-tenths of the width of the canvas.

.xview\_moveto(fraction)

This method scrolls the canvas in the same way as .xview(tk.MOVETO, fraction).

.xview\_scroll(n, what)  
Same as .xview(tk.SCROLL, n, what).  
.yview(tk.MOVETO, fraction)

The vertical scrolling equivalent of .xview(tk.MOVETO,…).

.yview(tk.SCROLL, n, what)

The vertical scrolling equivalent of .xview(tk.SCROLL,…).

.yview\_moveto(fraction)

The vertical scrolling equivalent of .xview().

.yview\_scroll(n, what)

The vertical scrolling equivalents of .xview(), .xview\_moveto(), and .xview\_scroll().

New Mexico Tech Computer Center

Tkinter 8.5 reference

27

8.7. Canvas arc objects

An arc object on a canvas, in its most general form, is a wedge-shaped slice taken out of an ellipse. This  
includes whole ellipses and circles as special cases. See Section 8.11, “Canvas oval objects” (p. 32) for  
more on the geometry of the ellipse drawn.  
To create an arc object on a canvas C, use:

id = C.create\_arc(x0, y0, x1, y1, option, ...)

The constructor returns the object ID of the new arc object on canvas C.  
Point (x0, y0) is the top left corner and (x1, y1) the lower right corner of a rectangle into which the ellipse  
is fit. If this rectangle is square, you get a circle.  
The various options include:

Table 7. Canvas arc options

activedash  
activefill  
activeoutline  
activeoutlinestipple  
activestipple  
activewidth  
dash  
dashoffset

disableddash  
disabledfill  
disabledoutline  
disabledoutlinestipple  
disabledstipple  
disabledwidth  
extent

fill

offset

outline  
outlineoffset

outlinestipple

These options apply when the arc is in the tk.ACTIVE state, that is,  
when the mouse is over the arc. For example, the activefill option  
specifies the interior color when the arc is active. For option values,

see dash, fill, outline, outlinestipple, stipple, and width,

respectively.

Dash pattern for the outline. See Section 5.13, “Dash patterns” (p. 17).  
Dash pattern offset for the outline. See Section 5.13, “Dash pat-  
terns” (p. 17).  
These options apply when the arc's state is tk.DISABLED.

Width of the slice in degrees. The slice starts at the angle given by the  
start option and extends counterclockwise for extent degrees.  
By default, the interior of an arc is transparent, and fill='' will select  
this behavior. You can also set this option to any color and the interior  
of the arc will be filled with that color.  
Stipple pattern offset for the interior of the arc. See Section 5.14,  
“Matching stipple patterns” (p. 17).  
The color of the border around the outside of the slice. Default is black.  
Stipple pattern offset for the outline. See Section 5.14, “Matching stipple  
patterns” (p. 17).  
If the outline option is used, this option specifies a bitmap used to  
stipple the border. Default is black, and that default can be specified

by setting outlinestipple=''.

28

Tkinter 8.5 reference

New Mexico Tech Computer Center

start

state

stipple

style

Starting angle for the slice, in degrees, measured from +x direction. If  
omitted, you get the entire ellipse.  
This option is tk.NORMAL by default. It may be set to tk.HIDDEN to  
make the arc invisible or to tk.DISABLED to gray out the arc and make  
it unresponsive to events.  
A bitmap indicating how the interior fill of the arc will be stippled.  
Default is stipple='' (solid). You'll probably want something like  
stipple='gray25'. Has no effect unless fill has been set to some  
color.  
The default is to draw the whole arc; use style=tk.PIESLICE for  
this style. To draw only the circular arc at the edge of the slice, use  
style=tk.ARC. To draw the circular arc and the chord (a straight line  
connecting the endpoints of the arc), use style=tk.CHORD.

tags

width

If a single string, the arc is tagged with that string. Use a tuple of strings  
to tag the arc with multiple tags. See Section 8.4, “Canvas tags” (p. 22).  
Width of the border around the outside of the arc. Default is 1 pixel.

8.8. Canvas bitmap objects

A bitmap object on a canvas is shown as two colors, the background color (for 0 data values) and the  
foreground color (for 1 values).  
To create a bitmap object on a canvas C, use:

id = C.create\_bitmap(x,

bitmap  
disabledbackground  
disabledbitmap  
disabledforeground  
foreground

state

tags

The bitmap to be displayed; see Section 5.7, “Bitmaps” (p. 12).  
These options specify the background, bitmap, and foreground to be used  
when the bitmap's state is tk.DISABLED.

The color that will appear where there are 1 values in the bitmap. The default

is foreground='black'.

By default, items are created with state=tk.NORMAL. Use tk.DISABLED  
to make the item grayed out and unresponsive to events; use tk.HIDDEN  
to make the item invisible.  
If a single string, the bitmap is tagged with that string. Use a tuple of strings  
to tag the bitmap with multiple tags. See Section 8.4, “Canvas tags” (p. 22).

8.9. Canvas image objects

To display a graphics image on a canvas C, use:

id = C.create\_image(x,

Table 10. Canvas line options

activedash  
activefill  
activestipple  
activewidth  
arrow

arrowshape

capstyle

dash

dashoffset

disableddash  
disabledfill  
disabledstipple  
disabledwidth  
fill  
joinstyle

offset

smooth

splinesteps

These options specify the dash, fill, stipple, and width values to be used  
when the line is active, that is, when the mouse is over it.

The default is for the line to have no arrowheads. Use arrow=tk.FIRST to get  
an arrowhead at the (x0, y0) end of the line. Use arrow=tk.LAST to get an  
arrowhead at the far end. Use arrow=tk.BOTH for arrowheads at both ends.  
A tuple (d1, d2, d3) that describes the shape of the arrowheads added by  
the arrow option. Default is (8,10,3).

You can specify the shape of the ends of the line with this option; see Section 5.12,  
“Cap and join styles” (p. 16). The default option is tk.BUTT.  
To produce a dashed line, specify this option; see Section 5.13, “Dash pat-  
terns” (p. 17). The default appearance is a solid line.  
If you specify a dash pattern, the default is to start the specified pattern at the  
beginning of the line. The dashoffset option allows you to specify that the  
start of the dash pattern occurs at a given distance after the start of the line. See  
Section 5.13, “Dash patterns” (p. 17).  
The dash, fill, stipple, and width values to be used when the item is in

the tk.DISABLED state.

The color to use in drawing the line. Default is fill='black'.  
For lines that are made up of more than one line segment, this option controls  
the appearance of the junction between segments. For more details, see Sec-  
tion 5.12, “Cap and join styles” (p. 16). The default style is ROUND  
For stippled lines, the purpose of this option is to match the item's stippling  
pattern with those of adjacent objects. See Section 5.14, “Matching stipple pat-  
terns” (p. 17)..  
If true, the line is drawn as a series of parabolic splines fitting the point set. De-  
fault is false, which renders the line as a set of straight segments.  
If the smooth option is true, each spline is rendered as a number of straight line  
segments. The splinesteps option specifies the number of segments used to  
approximate each section of the line; the default is splinesteps=12.

New Mexico Tech Computer Center

Tkinter 8.5 reference

31

state

stipple

tags

width

Normally, line items are created in state tk.NORMAL. Set this option to  
tk.HIDDEN to make the line invisible; set it to tk.DISABLED to make it unre-  
sponsive to the mouse.  
To draw a stippled line, set this option to a bitmap that specifies the stippling  
pattern, such as stipple='gray25'. See Section 5.7, “Bitmaps” (p. 12) for  
the possible values.  
If a single string, the line is tagged with that string. Use a tuple of strings to tag  
the line with multiple tags. See Section 8.4, “Canvas tags” (p. 22).  
The line's width. Default is 1 pixel. See Section 5.1, “Dimensions” (p. 9) for  
possible values.

8.11. Canvas oval objects

Ovals, mathematically, are ellipses, including circles as a special case. The ellipse is fit into a rectangle  
defined by the coordinates (x0, y0) of the top left corner and the coordinates (x1, y1) of a point just  
outside of the bottom right corner.

The oval will coincide with the top and left-hand lines of this box, but will fit just inside the bottom and  
right-hand sides.  
To create an ellipse on a canvas C, use:

id = C.create\_oval(x0, y0, x1, y1, option, ...)

which returns the object ID of the new oval object on canvas C.  
Options for ovals:

Table 11. Canvas oval options

activedash  
activefill  
activeoutline  
activeoutlinestipple

These options specify the dash pattern, fill color, outline color, outline  
stipple pattern, interior stipple pattern, and outline width values to be  
used when the oval is in the tk.ACTIVE state, that is, when the mouse  
is over the oval. For option values, see dash, fill, outline, out-

linestipple, stipple, and width.

32

Tkinter 8.5 reference

New Mexico Tech Computer Center

activestipple  
activewidth  
dash

dashoffset

disableddash  
disabledfill  
disabledoutline  
disabledoutlinestipple  
disabledstipple  
disabledwidth  
fill

offset

outline

outlineoffset

stipple

outlinestipple

state

tags

width

To produce a dashed border around the oval, set this option to a dash  
pattern; see Section 5.13, “Dash patterns” (p. 17)  
When using the dash option, the dashoffset option is used to change  
the alignment of the border's dash pattern relative to the oval. See  
Section 5.14, “Matching stipple patterns” (p. 17).  
These options specify the appearance of the oval when the item's state

is tk.DISABLED.

The default appearance of an oval's interior is transparent, and a value  
of fill='' will select this behavior. You can also set this option to  
any color and the interior of the ellipse will be filled with that color;  
see Section 5.3, “Colors” (p. 10).  
Stipple pattern offset of the interior. See Section 5.14, “Matching stipple  
patterns” (p. 17).  
The color of the border around the outside of the ellipse. Default is

outline='black'.

Stipple pattern offset of the border. See Section 5.14, “Matching stipple  
patterns” (p. 17).  
A bitmap indicating how the interior of the ellipse will be stippled.  
Default is stipple='', which means a solid color. A typical value  
would be stipple='gray25'. Has no effect unless the fill has  
been set to some color. See Section 5.7, “Bitmaps” (p. 12).  
Stipple pattern to be used for the border. For option values, see

stipple below.

By default, oval items are created in state tk.NORMAL. Set this option  
to tk.DISABLED to make the oval unresponsive to mouse actions. Set  
it to tk.HIDDEN to make the item invisible.  
If a single string, the oval is tagged with that string. Use a tuple of  
strings to tag the oval with multiple tags. See Section 8.4, “Canvas  
tags” (p. 22).  
Width of the border around the outside of the ellipse. Default is 1 pixel;  
see Section 5.1, “Dimensions” (p. 9) for possible values. If you set this  
to zero, the border will not appear. If you set this to zero and make the  
fill transparent, you can make the entire oval disappear.

8.12. Canvas polygon objects

As displayed, a polygon has two parts: its outline and its interior. Its geometry is specified as a series  
of vertices [(x0, y0), (x1, y1), … (xn, yn)], but the actual perimeter includes one more segment from (xn,  
yn) back to (x0, y0). In this example, there are five vertices:

New Mexico Tech Computer Center

Tkinter 8.5 reference

33

To create a new polygon object on a canvas C:

id = C.create\_polygon(x0, y0, x1, y1, ..., option, ...)

The constructor returns the object ID for that object. Options:

Table 12. Canvas polygon options

activedash  
activefill  
activeoutline  
activeoutlinestipple  
activestipple  
activewidth  
dash

dashoffset

disableddash  
disabledfill  
disabledoutline  
disabledoutlinestipple  
disabledstipple  
disabledwidth  
fill

joinstyle

offset

outline

These options specify the appearance of the polygon when it is in the  
tk.ACTIVE state, that is, when the mouse is over it. For option values,

see dash, fill, outline, outlinestipple, stipple, and width.

Use this option to produce a dashed border around the polygon. See  
Section 5.13, “Dash patterns” (p. 17).  
Use this option to start the dash pattern at some point in its cycle other  
than the beginning. See Section 5.13, “Dash patterns” (p. 17).  
These options specify the appearance of the polygon when its state

is tk.DISABLED.

You can color the interior by setting this option to a color. The default  
appearance for the interior of a polygon is transparent, and you can  
set fill='' to get this behavior. See Section 5.3, “Colors” (p. 10).  
This option controls the appearance of the intersections between adja-  
cent sides of the polygon. See Section 5.12, “Cap and join styles” (p. 16).  
Offset of the stipple pattern in the interior of the polygon. See Sec-  
tion 5.14, “Matching stipple patterns” (p. 17).  
Color of the outline; defaults to outline='', which makes the outline  
transparent.

34

Tkinter 8.5 reference

New Mexico Tech Computer Center

outlineoffset

outlinestipple

smooth

splinesteps

state

stipple

tags

width

Stipple offset for the border. See Section 5.14, “Matching stipple pat-  
terns” (p. 17).  
Use this option to get a stippled border around the polygon. The option  
value must be a bitmap; see Section 5.7, “Bitmaps” (p. 12).  
The default outline uses straight lines to connect the vertices; use  
smooth=0 to get that behavior. If you use smooth=1, you get a con-  
tinuous spline curve. Moreover, if you set smooth=1, you can make  
any segment straight by duplicating the coordinates at each end of that  
segment.  
If the smooth option is true, each spline is rendered as a number of  
straight line segments. The splinesteps option specifies the number  
of segments used to approximate each section of the line; the default

is splinesteps=12.

By default, polygons are created in the tk.NORMAL state. Set this option  
to tk.HIDDEN to make the polygon invisible, or set it to tk.DISABLED  
to make it unresponsive to the mouse.  
A bitmap indicating how the interior of the polygon will be stippled.  
Default is stipple='', which means a solid color. A typical value  
would be stipple='gray25'. Has no effect unless the fill has  
been set to some color. See Section 5.7, “Bitmaps” (p. 12).  
If a single string, the polygon is tagged with that string. Use a tuple of  
strings to tag the polygon with multiple tags. See Section 8.4, “Canvas  
tags” (p. 22).  
Width of the outline; defaults to 1. See Section 5.1, “Dimen-  
sions” (p. 9).

8.13. Canvas rectangle objects

Each rectangle is specified as two points: (x0, y0) is the top left corner, and (x1, y1) is the location of  
the pixel just outside of the bottom right corner.  
For example, the rectangle specified by top left corner (100,100) and bottom right corner (102,102) is a  
square two pixels by two pixels, including pixel (101,101) but not including (102,102).  
Rectangles are drawn in two parts:  
• The outline lies inside the rectangle on its top and left sides, but outside the rectangle on its bottom  
and right side. The default appearance is a one-pixel-wide black border.  
For example, consider a rectangle with top left corner (10,10) and bottom right corner (11,11). If you  
request no border (width=0) and green fill (fill='green'), you will get one green pixel at (10,10).  
However, if you request the same options with a black border (width=1), you will get four black  
pixels at (10,10), (10,11), (11,10), and (11,11).  
• The fill is the area inside the outline. Its default appearance is transparent.  
To create a rectangle object on canvas C:

id = C.create\_rectangle(x0, y0, x1, y1, option, ...)

This constructor returns the object ID of the rectangle on that canvas. Options include:

New Mexico Tech Computer Center

Tkinter 8.5 reference

35

Table 13. Canvas rectangle options

activedash  
activefill  
activeoutline  
activeoutlinestipple  
activestipple  
activewidth  
dash

dashoffset

disableddash  
disabledfill  
disabledoutline  
disabledoutlinestipple  
disabledstipple  
disabledwidth  
fill

offset

outline  
outlineoffset

outlinestipple

state

stipple

tags

width

These options specify the appearance of the rectangle when its state  
is tk.ACTIVE, that is, when the mouse is on top of the rectangle. For

option values, refer to dash, fill, outline, outlinestipple,  
stipple, and width below.

To produce a dashed border around the rectangle, use this option to  
specify a dash pattern. See Section 5.13, “Dash patterns” (p. 17).  
Use this option to start the border's dash pattern at a different point in  
the cycle; see Section 5.13, “Dash patterns” (p. 17).  
These options specify the appearance of the rectangle when its state

is tk.DISABLED.

By default, the interior of a rectangle is empty, and you can get this  
behavior with fill=''. You can also set the option to a color; see  
Section 5.3, “Colors” (p. 10).  
Use this option to change the offset of the interior stipple pattern. See  
Section 5.14, “Matching stipple patterns” (p. 17).  
The color of the border. Default is outline='black'.  
Use this option to adjust the offset of the stipple pattern in the outline;  
see Section 5.14, “Matching stipple patterns” (p. 17).  
Use this option to produce a stippled outline. The pattern is specified  
by a bitmap; see Section 5.7, “Bitmaps” (p. 12).  
By default, rectangles are created in the tk.NORMAL state. The state is  
tk.ACTIVE when the mouse is over the rectangle. Set this option to  
tk.DISABLED to gray out the rectangle and make it unresponsive to  
mouse events.  
A bitmap indicating how the interior of the rectangle will be stippled.  
Default is stipple='', which means a solid color. A typical value  
would be stipple='gray25'. Has no effect unless the fill has  
been set to some color. See Section 5.7, “Bitmaps” (p. 12).  
If a single string, the rectangle is tagged with that string. Use a tuple  
of strings to tag the rectangle with multiple tags. See Section 8.4,  
“Canvas tags” (p. 22).  
Width of the border. Default is 1 pixel. Use width=0 to make the border  
invisible. See Section 5.1, “Dimensions” (p. 9).

36

Tkinter 8.5 reference

New Mexico Tech Computer Center

8.14. Canvas text objects

You can display one or more lines of text on a canvas C by creating a text object:

id = C.create\_text(x,

• To replace the text in an item with object ID I on a canvas C with the text from a string S, call

C.itemconfigure(I, text=S).

A number of canvas methods allow you to manipulate text items. See Section 8.6, “Methods on Canvas  
widgets” (p. 22), especially dchars, focus, icursor, index, and insert.

8.15. Canvas window objects

You can place any Tkinter widget onto a canvas by using a canvas window object. A window is a rectan-  
gular area that can hold one Tkinter widget. The widget must be the child of the same top-level window  
as the canvas, or the child of some widget located in the same top-level window.  
If you want to put complex multi-widget objects on a canvas, you can use this method to place a Frame  
widget on the canvas, and then place other widgets inside that frame.  
To create a new canvas window object on a canvas C:

id = C.create\_window(x,

• You will need to create a control variable, an instance of the IntVar class, so your program can query  
and set the state of the checkbutton. See Section 52, “Control variables: the values behind the wid-  
gets” (p. 153), below.  
• You can also use event bindings to react to user actions on the checkbutton; see Section 54,  
“Events” (p. 157), below.  
• You can disable a checkbutton. This changes its appearance to “grayed out” and makes it unresponsive  
to the mouse.  
• You can get rid of the checkbutton indicator and make the whole widget a “push-push” button that  
looks recessed when it is set, and looks raised when it is cleared.  
To create a checkbutton in an existing parent window or frame parent:

w = tk.Checkbutton(parent, option, ...)

The constructor returns a new Checkbutton widget. Options include:

Table 16. Checkbutton widget options

activebackground

activeforeground  
anchor

bg or background

bitmap

bd or borderwidth

command

compound

cursor

disabledforeground

font  
fg or foreground

height

Background color when the checkbutton is under the cursor. See Section 5.3,  
“Colors” (p. 10).  
Foreground color when the checkbutton is under the cursor.  
If the widget inhabits a space larger than it needs, this option specifies  
where the checkbutton will sit in that space. The default is anchor=tk.CEN-  
TER. See Section 5.5, “Anchors” (p. 12) for the allowable values. For ex-  
ample, if you use anchor=NW, the widget will be placed in the upper left  
corner of the space.  
The normal background color displayed behind the label and indicator. See  
Section 5.3, “Colors” (p. 10). For the bitmap option, this specifies the color  
displayed for 0-bits in the bitmap.  
To display a monochrome image on a button, set this option to a bitmap;  
see Section 5.7, “Bitmaps” (p. 12).  
The size of the border around the indicator. Default is two pixels. For pos-  
sible values, see Section 5.1, “Dimensions” (p. 9).  
A procedure to be called every time the user changes the state of this  
checkbutton.  
Use this option to display both text and a graphic, which may be either a  
bitmap or an image, on the button. Allowable values describe the position  
of the graphic relative to the text, and may be any of tk.BOTTOM, tk.TOP,

tk.LEFT, tk.RIGHT, or tk.CENTER. For example, compound=tk.LEFT

would position the graphic to the left of the text.  
If you set this option to a cursor name (see Section 5.8, “Cursors” (p. 13)),  
the mouse cursor will change to that pattern when it is over the checkbutton.  
The foreground color used to render the text of a disabled checkbutton. The  
default is a stippled version of the default foreground color.  
The font used for the text. See Section 5.4, “Type fonts” (p. 10).  
The color used to render the text. For the bitmap option, this specifies  
the color displayed for 1-bits in the bitmap.  
The number of lines of text on the checkbutton. Default is 1.

New Mexico Tech Computer Center

Tkinter 8.5 reference

39

highlightbackground

highlightcolor  
highlightthickness

image

indicatoron

The color of the focus highlight when the checkbutton does not have focus.  
See Section 53, “Focus: routing keyboard input” (p. 155).  
The color of the focus highlight when the checkbutton has the focus.  
The thickness of the focus highlight. Default is 1. Set to 0 to suppress display  
of the focus highlight.  
To display a graphic image on the button, set this option to an image object.  
See Section 5.9, “Images” (p. 14).  
Normally a checkbutton displays as its indicator a box that shows whether  
the checkbutton is set or not. You can get this behavior by setting indic-  
atoron=1. However, if you set indicatoron=0, the indicator disappears,  
and the entire widget becomes a push-push button that looks raised when  
it is cleared and sunken when it is set. You may want to increase the bor-  
derwidth value to make it easier to see the state of such a control.  
If the text contains multiple lines, this option controls how the text is jus-

tified: tk.CENTER, tk.LEFT, or tk.RIGHT.

textvariable

underline

variable

width

wraplength

If you need to change the label on a checkbutton during execution, create  
a StringVar (see Section 52, “Control variables: the values behind the wid-  
gets” (p. 153)) to manage the current value, and set this option to that control  
variable. Whenever the control variable's value changes, the checkbutton's  
annotation will automatically change as well.  
With the default value of -1, none of the characters of the text label are un-  
derlined. Set this option to the index of a character in the text (counting  
from zero) to underline that character.  
The control variable that tracks the current state of the checkbutton; see  
Section 52, “Control variables: the values behind the widgets” (p. 153).  
Normally this variable is an IntVar, and 0 means cleared and 1 means  
set, but see the offvalue and onvalue options above.  
The default width of a checkbutton is determined by the size of the displayed  
image or text. You can set this option to a number of characters and the  
checkbutton will always have room for that many characters.  
Normally, lines are not wrapped. You can set this option to a number of  
characters and all lines will be broken into pieces no longer than that  
number.

Methods on checkbuttons include:

.deselect()

Clears (turns off) the checkbutton.

.flash()

Flashes the checkbutton a few times between its active and normal colors, but leaves it the way it  
started.

.invoke()

You can call this method to get the same actions that would occur if the user clicked on the check-  
button to change its state.

.select()

Sets (turns on) the checkbutton.

.toggle()

Clears the checkbutton if set, sets it if cleared.

10. The

• 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

insertborderwidth

insertofftime

insertontime

insertwidth

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” (p. 12))  
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.  
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.  
Similar to insertofftime, this option specifies how much time the  
cursor spends on per blink. Default is 600 (milliseconds).  
By default, the insertion cursor is 2 pixels wide. You can adjust this by setting  
insertwidth to any dimension.  
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.

width

xscrollcommand

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.  
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

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

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

.select\_to(index)

index.  
.xview(index)

Same as .xview(). This method is useful in linking the

1. Write a callback function that checks the text in the

'%i'

'%P'  
'%s'  
'%S'

'%v'  
'%V'

When the user attempts to insert or delete text, this argument will be the index of the beginning  
of the insertion or deletion. If the callback was due to focus in, focus out, or a change to the  
textvariable, the argument will be -1.  
The value that the text will have if the change is allowed.  
The text in the entry before the change.  
If the call was due to an insertion or deletion, this argument will be the text being inserted or  
deleted.  
The current value of the widget's validate option.  
The reason for this callback: one of 'focusin', 'focusout', 'key', or 'forced' if the

textvariable was changed.

'%W'

The name of the widget.

Here is a small example. Suppose you want your callback to receive the '%d' to find out why it was  
called; '%i' to find out where the insertion or deletion would occur; and '%S' to find out what is to  
be inserted or deleted. Your method might look like this:

def isOkay(self, why, where, what):  
...

Next you use the universal .register() method to wrap this function. We assume that self is some  
widget.

okayCommand = self.register(isOkay)

To set up this callback, you would use these two options in the

class that inherits from Frame, adding your own interface to it. This is a good way to hide the details  
of interactions within a group of related widgets from the outside world.  
To create a new frame widget in a root window or frame named parent:

w = Frame(parent, option, ...)

The constructor returns the new Frame widget. Options:

Table 19. Frame widget options

bg or background  
bd or borderwidth

cursor

height

highlightbackground

highlightcolor  
highlightthickness  
padx

The frame's background color. See Section 5.3, “Colors” (p. 10).  
Width of the frame's border. The default is 0 (no border). For permitted  
values, see Section 5.1, “Dimensions” (p. 9).  
The cursor used when the mouse is within the frame widget; see Section 5.8,  
“Cursors” (p. 13).  
The vertical dimension of the new frame. This will be ignored unless you  
also call .grid\_propagate(0) on the frame; see Section 4.2, “Other grid  
management methods” (p. 7).  
Color of the focus highlight when the frame does not have focus. See Sec-  
tion 53, “Focus: routing keyboard input” (p. 155).  
Color shown in the focus highlight when the frame has the focus.  
Thickness of the focus highlight.  
Normally, a Frame fits tightly around its contents. To add N pixels of hori-  
zontal space inside the frame, set padx=N.  
Used to add vertical space inside a frame. See padx above.

Table 20. Label widget options

activebackground  
activeforeground  
anchor

bg or background  
bitmap

bd or borderwidth

compound

cursor

disabledforeground

font

fg or foreground

height

highlightbackground  
highlightcolor  
highlightthickness  
image

Background color to be displayed when the mouse is over the widget.  
Foreground color to be displayed when the mouse is over the widget.  
This options controls where the text is positioned if the widget has more  
space than the text needs. The default is anchor=tk.CENTER, which centers  
the text in the available space. For other values, see Section 5.5, “An-  
chors” (p. 12). For example, if you use anchor=tk.NW, the text would be  
positioned in the upper left-hand corner of the available space.  
The background color of the label area. See Section 5.3, “Colors” (p. 10).  
Set this option equal to a bitmap or image object and the label will display  
that graphic. See Section 5.7, “Bitmaps” (p. 12) and Section 5.9, “Im-  
ages” (p. 14).  
Width of the border around the label; see Section 5.1, “Dimensions” (p. 9).  
The default value is two pixels.  
If you would like the Label widget to display both text and a graphic  
(either a bitmap or an image), the compound option specifies the relative  
orientation of the graphic relative to the text. Values may be any of tk.LEFT,

tk.RIGHT, tk.CENTER, tk.BOTTOM, or tk.TOP. For example, if you

specify compound=BOTTOM, the graphic will be displayed below the text.  
Cursor that appears when the mouse is over this label. See Section 5.8,  
“Cursors” (p. 13).  
The foreground color to be displayed when the widget's state is

tk.DISABLED.

If you are displaying text in this label (with the text or textvariable  
option, the font option specifies in what font that text will be displayed.  
See Section 5.4, “Type fonts” (p. 10).  
If you are displaying text or a bitmap in this label, this option specifies the  
color of the text. If you are displaying a bitmap, this is the color that will  
appear at the position of the 1-bits in the bitmap. See Section 5.3, “Col-  
ors” (p. 10).  
Height of the label in lines (not pixels!). If this option is not set, the label  
will be sized to fit its contents.  
Color of the focus highlight when the widget does not have focus.  
The color of the focus highlight when the widget has focus.  
Thickness of the focus highlight.  
To display a static image in the label widget, set this option to an image  
object. See Section 5.9, “Images” (p. 14).  
Specifies how multiple lines of text will be aligned with respect to each  
other: tk.LEFT for flush left, tk.CENTER for centered (the default), or  
tk.RIGHT for right-justified.

relief

Specifies the appearance of a decorative border around the label. The default  
is tk.FLAT; for other values, see Section 5.6, “Relief styles” (p. 12).  
By default, an

This constructor returns the new LabelFrame widget. Options:

Table 21. LabelFrame widget options

bg or background

bd or borderwidth

cursor

fg or foreground  
height

highlightbackground  
highlightcolor  
highlightthickness  
labelanchor

The background color to be displayed inside the widget; see Section 5.3,  
“Colors” (p. 10).  
Width of the border drawn around the perimeter of the widget; see Sec-  
tion 5.1, “Dimensions” (p. 9). The default value is two pixels.  
Selects the cursor that appears when the mouse is over the widget; see  
Section 5.8, “Cursors” (p. 13).  
Color to be used for the label text.  
The vertical dimension of the new frame. This will be ignored unless you  
also call .grid\_propagate(0) on the frame; see Section 4.2, “Other  
grid management methods” (p. 7).  
Color of the focus highlight when the widget does not have focus.  
The color of the focus highlight when the widget has focus.  
Thickness of the focus highlight.  
Use this option to specify the position of the label on the widget's border.  
The default position is 'nw', which places the label at the left end of the  
top border. For the nine possible label positions, refer to this diagram:

labelwidget

padx

Instead of a text label, you can use any widget as the label by passing  
that widget as the value of this option. If you supply both labelwidget  
and text options, the text option is ignored.  
Use this option to add additional padding inside the left and right sides  
of the widget's frame. The value is in pixels.  
Use this option to add additional padding inside the top and bottom of  
the widget's frame. The value is in pixels.

14. The Listbox widget

The purpose of a listbox widget is to display a set of lines of text. Generally they are intended to allow  
the user to select one or more items from a list. All the lines of text use the same font. If you need  
something more like a text editor, see Section 24, “The Text widget” (p. 82).  
To create a new listbox widget inside a root window or frame parent:

w = tk.Listbox(parent, option, ...)

This constructor returns the new Listbox widget. Options:

Table 22. Listbox widget options

activestyle

This option specifies the appearance of the active line. It may have any of  
these values:

bg or background  
bd or borderwidth

cursor

disabledforeground  
exportselection

font  
fg or foreground  
height  
highlightbackground

highlightcolor  
highlightthickness  
listvariable

'underline'

The active line is underlined. This is the default option.

'dotbox'

The active line is enclosed in a dotted line on all four sides.

'none'

The active line is given no special appearance.

The background color in the listbox.  
The width of the border around the listbox. Default is two pixels. For pos-  
sible values, see Section 5.1, “Dimensions” (p. 9).  
The cursor that appears when the mouse is over the listbox. See Section 5.8,  
“Cursors” (p. 13).  
The color of the text in the listbox when its state is tk.DISABLED.  
By default, the user may select text with the mouse, and the selected text  
will be exported to the clipboard. To disable this behavior, use exportse-

lection=0.

The font used for the text in the listbox. See Section 5.4, “Type fonts” (p. 10).  
The color used for the text in the listbox. See Section 5.3, “Colors” (p. 10).  
Number of lines (not pixels!) shown in the listbox. Default is 10.  
Color of the focus highlight when the widget does not have focus. See Sec-  
tion 53, “Focus: routing keyboard input” (p. 155).  
Color shown in the focus highlight when the widget has the focus.  
Thickness of the focus highlight.  
A StringVar that is connected to the complete list of values in the listbox  
(see Section 52, “Control variables: the values behind the widgets” (p. 153).  
If you call the .get() method of the listvariable, you will get back a  
string of the form "('v0', 'v1', ...)", where each vi is the contents  
of one line of the listbox.  
To change the entire set of lines in the listbox at once, call .set(s) on the  
listvariable, where s is a string containing the line values with spaces  
between them.

52

Tkinter 8.5 reference

New Mexico Tech Computer Center

For example, if listCon is a StringVar associated with a listbox's  
listvariable option, this call would set the listbox to contain three lines:

listCon.set('ant bee cicada')

This call would return the string "('ant', 'bee', 'cicada')":

listCon.get()

Selects three-dimensional border shading effects. The default is tk.SUNKEN.  
For other values, see Section 5.6, “Relief styles” (p. 12).  
The background color to use displaying selected text.  
The width of the border to use around selected text. The default is that the  
selected item is shown in a solid block of color selectbackground; if you  
increase the selectborderwidth, the entries are moved farther apart  
and the selected entry shows tk.RAISED relief (see Section 5.6, “Relief  
styles” (p. 12)).  
The foreground color to use displaying selected text.  
Determines how many items can be selected, and how mouse drags affect  
the selection:  
• tk.BROWSE: Normally, you can only select one line out of a listbox. If  
you click on an item and then drag to a different line, the selection will  
follow the mouse. This is the default.  
• tk.SINGLE: You can only select one line, and you can't drag the  
mouse—wherever you click button 1, that line is selected.  
• tk.MULTIPLE: You can select any number of lines at once. Clicking on  
any line toggles whether or not it is selected.  
• tk.EXTENDED: You can select any adjacent group of lines at once by  
clicking on the first line and dragging to the last line.

By default, a listbox is in the tk.NORMAL state. To make the listbox unre-  
sponsive to mouse events, set this option to tk.DISABLED.  
Normally, the focus will tab through listbox widgets. Set this option to 0 to  
take the widget out of the sequence. See Section 53, “Focus: routing keyboard  
input” (p. 155).  
The width of the widget in characters (not pixels!). The width is based on  
an average character, so some strings of this length in proportional fonts  
may not fit. The default is 20.  
If you want to allow the user to scroll the listbox horizontally, you can link  
your listbox widget to a horizontal scrollbar. Set this option to the .set  
method of the scrollbar. See Section 14.1, “Scrolling a Listbox wid-  
get” (p. 56) for more on scrollable listbox widgets.  
If you want to allow the user to scroll the listbox vertically, you can link  
your listbox widget to a vertical scrollbar. Set this option to the .set  
method of the scrollbar. See Section 14.1, “Scrolling a Listbox wid-  
get” (p. 56).

relief

selectbackground  
selectborderwidth

selectforeground  
selectmode

state

takefocus

width

xscrollcommand

yscrollcommand

A special set of index forms is used for many of the methods on listbox objects:

New Mexico Tech Computer Center

Tkinter 8.5 reference

53