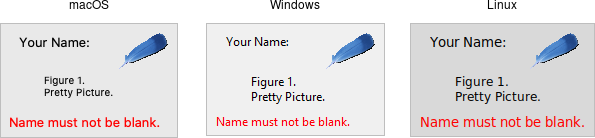
## #Label

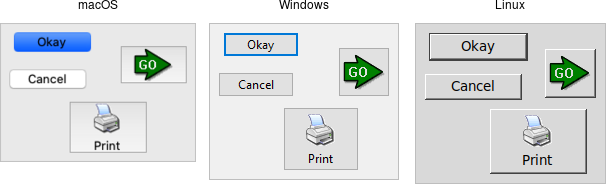
#A **label** is a widget that displays text or images, typically that users will just view but not otherwise interact with. Labels are used for to identify controls or other parts of the user interface, provide textual feedback or results, etc.

**  
Label widgets.**

Like frames, labels can take several different configuration options, which can alter how they are displayed.

## #Button

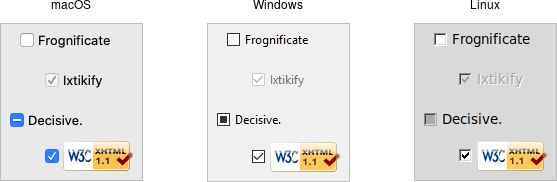
A **button**, unlike a frame or label, is very much there to interact with. Users press a button to perform an action. Like labels, they can display text or images, but accept additional options to change their behavior.

**  
Button widgets.**

Typically, their contents and command callback are specified at the same time. As with other widgets, buttons can take several different configuration options, including the standard style option, which can alter their appearance and behavior.

# Checkbutton

# A **checkbutton** widget is like a regular button that also holds a binary value of some kind (i.e., a toggle). When pressed, a checkbutton flips the toggle and then invokes its callback. Checkbutton widgets are frequently used to allow users to turn an option on or off.

**  
Checkbutton widgets.**

Checkbuttons use many of the same options as regular buttons but add a few more. The text, textvariable, image, and compound configuration options control the display of the label (next to the checkbox itself). Similarly, the command option lets you specify a command to be called every time a user toggles the checkbutton; and the invoke method will also execute the same command. The state and instate methods allow you to manipulate the disabled state flag to enable or disable the checkbutton.

## #Entry

* [Widget Roundup](https://tkdocs.com/widgets/entry.html)
* [Reference Manual](https://tcl.tk/man/tcl8.6/TkCmd/ttk_entry.htm)

An **entry** widget presents users with a single line text field where they can type in a string value. These can be just about anything: a name, a city, a password, social security number, etc.

**  
Entry widgets.**

A width configuration option may be specified to provide the number of characters wide the entry should be. This allows you, for example, to display a shorter entry for a zip or postal code.

# #**The Grid Geometry Manager**

We'll take a bit of a break from talking about different widgets (what to put onscreen) and focus instead on geometry management (where to put those widgets). We introduced the general idea of geometry management in the "Tk Concepts" chapter. Here, we focus on one specific geometry manager: grid.

As we've seen, grid lets you layout widgets in columns and rows. If you're familiar with using HTML tables to do layout, you'll feel right at home here. This chapter illustrates the various ways you can tweak grid to give you all the control you need for your user interface.

Grid is one of several geometry managers available in Tk, but its mix of power, flexibility, and ease of use make it the best choice for general use. Its constraint model is a natural fit with today's layouts that rely on the alignment of widgets. There are other geometry managers in Tk: pack is also quite powerful, but harder to use and understand, while place gives you complete control of positioning each element. Even widgets like paned windows, notebooks, canvas, and text that we'll explore later can act as geometry managers.

## Columns and Rows

In grid, widgets are assigned a column number and a row number. These indicate each widget's position relative to other widgets. All widgets in the same column will be above or below each other. Those in the same row will be to the left or right of each other.

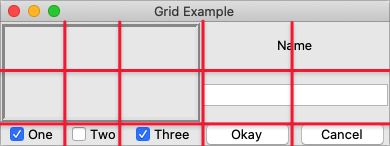
Column and row numbers must be positive integers (i.e., 0, 1, 2, ...). You don't have to start at 0 and can leave gaps in column and row numbers (e.g., column 1, 2, 10, 11, 12, 20, 21). This is useful if you plan to add more widgets in the middle of the user interface later.

The width of each column will vary depending on the width of the widgets contained within the column. Ditto for the height of each row. This means when sketching out your user interface and dividing it into rows and columns, you don't need to worry about each column or row being equal width.

## Spanning Multiple Cells

Widgets can take up more than a single cell in the grid; to do this, we'll use the columnspan and rowspan options when gridding the widget. These are analogous to the "colspan" and "rowspan" attribute of HTML tables.

Here is an example of creating a user interface with multiple widgets, some that take up more than a single cell.

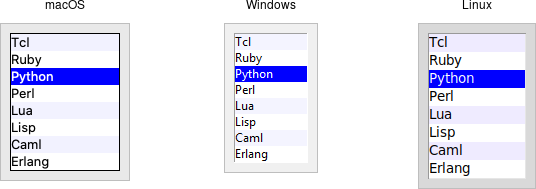
**  
Gridding multiple widgets.**

## #Listbox

A **listbox** widget displays a list of single-line text items, usually lengthy, and allows users to browse through the list, selecting one or more.

Listboxes are part of the classic Tk widgets; there is not presently a listbox in the themed Tk widget set.

*Tk's treeview widget (which is themed) can also be used as a listbox (a one level deep tree), allowing you to use icons and styles with the list. It's also likely that a multi-column (table) list widget will make it into Tk at some point, whether based on treeview or one of the available extensions.*

**  
Listbox widgets.**

Listboxes are created using the **Listbox** class. A height configuration option can specify the number of lines the listbox will display at a time without scrolling:

## VideoCapture() [4/5]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| cv::VideoCapture::VideoCapture | ( | int | index | ) |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Python:** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | cv.VideoCapture( |  | ) -> | <VideoCapture object> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | cv.VideoCapture( | filename | ) -> | <VideoCapture object> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | cv.VideoCapture( | filename, apiPreference | ) -> | <VideoCapture object> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | cv.VideoCapture( | index | ) -> | <VideoCapture object> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | cv.VideoCapture( | index, apiPreference | ) -> | <VideoCapture object> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Open a camera for video capturing.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**Parameters**

|  |  |
| --- | --- |
| **index** | camera\_id + domain\_offset (CAP\_\*) id of the video capturing device to open. To open default camera using default backend just pass 0. Use a domain\_offset to enforce a specific reader implementation if multiple are available like [**cv::CAP\_FFMPEG**](https://docs.opencv.org/3.4/d4/d15/group__videoio__flags__base.html#gga023786be1ee68a9105bf2e48c700294dacf10e9692c4166f74de62b7d00c377d0) or [**cv::CAP\_IMAGES**](https://docs.opencv.org/3.4/d4/d15/group__videoio__flags__base.html#gga023786be1ee68a9105bf2e48c700294dabe73da7a65e66cdd47c87733a4ea6ef0) or [**cv::CAP\_DSHOW**](https://docs.opencv.org/3.4/d4/d15/group__videoio__flags__base.html#gga023786be1ee68a9105bf2e48c700294dab6ac3effa04f41ed5470375c85a23504). e.g. to open Camera 1 using the MS Media Foundation API use index = 1 + [**cv::CAP\_MSMF**](https://docs.opencv.org/3.4/d4/d15/group__videoio__flags__base.html#gga023786be1ee68a9105bf2e48c700294da278d5ad4907c9c0fe6d1c6104b746019) |