back next

The Tkinter Grid Geometry Manager

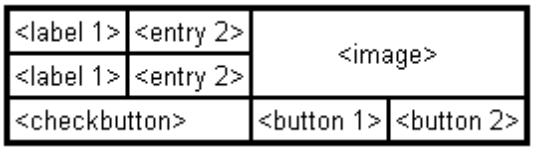
The **Grid** geometry manager puts the widgets in a 2-dimensional table. The master widget is split into a number of rows and columns, and each “cell” in the resulting table can hold a widget.

When to use the Grid Manager

The grid manager is the most flexible of the geometry managers in Tkinter. If you don’t want to learn how and when to use all three managers, you should at least make sure to learn this one.

The grid manager is especially convenient to use when designing dialog boxes. If you’re using the packer for that purpose today, you’ll be surprised how much easier it is to use the grid manager instead. Instead of using lots of extra frames to get the packing to work, you can in most cases simply pour all the widgets into a single container widget, and use the grid manager to get them all where you want them. (I tend to use two containers; one for the dialog body, and one for the button box at the bottom.)

Consider the following example:



Creating this layout using the pack manager is possible, but it takes a number of extra frame widgets, and a lot of work to make things look good. If you use the grid manager instead, you only need one call per widget to get everything laid out properly (see next section for the code needed to create this layout).

Warning: Never mix grid and pack in the same master window. Tkinter will happily spend the rest of your lifetime trying to negotiate a solution that both managers are happy with. Instead of waiting, kill the application, and take another look at your code. A common mistake is to use the wrong parent for some of the widgets.

Patterns

Using the grid manager is easy. Just create the widgets, and use the **grid** method to tell the manager in which row and column to place them. You don’t have to specify the size of the grid beforehand; the manager automatically determines that from the widgets in it.

```
Label(master, text="First").grid(row=0)
Label(master, text="Second").grid(row=1)

e1 = Entry(master)
e2 = Entry(master)

e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
```

Note that the column number defaults to 0 if not given.

Running the above example produces the following window:

Simple grid example

Empty rows and columns are ignored. The result would have been the same if you had placed the widgets in row 10 and 20 instead.

Note that the widgets are centered in their cells. You can use the **sticky** option to change this; this option takes one or more values from the set **N**, **S**, **E**, **W**. To align the labels to the left border, you could use **W** (west):

```
Label(master, text="First").grid(row=0, sticky=W)
Label(master, text="Second").grid(row=1, sticky=W)

e1 = Entry(master)
e2 = Entry(master)

e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
```

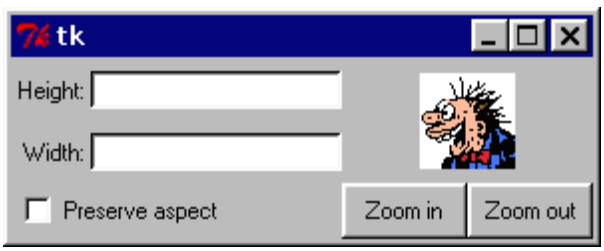
Using the sticky option

You can also have the widgets span more than one cell. The **columnspan** option is used to let a widget span more than one column, and the **rowspan** option lets it span more than one row. The following code creates the layout shown in the previous section:

```
label1.grid(sticky=E)
label2.grid(sticky=E)

entry1.grid(row=0, column=1)
entry2.grid(row=1, column=1)

checkbutton.grid(columnspan=2, sticky=W)

image.grid(row=0, column=2, columnspan=2, rowspan=2,
           sticky=W+E+N+S, padx=5, pady=5)

button1.grid(row=2, column=2)
button2.grid(row=2, column=3)
```

There are plenty of things to note in this example. First, no position is specified for the label widgets. In this case, the column defaults to 0, and the row to the *first unused row in the grid*. Next, the entry widgets are positioned as usual, but the checkbutton widget is placed on the next empty row (row 2, in this case), and is configured to span two columns. The resulting cell will be as wide as the label and entry columns combined. The image widget is configured to span both columns and rows at the same time. The buttons, finally, is packed each in a single cell:

Using column and row spans



Reference

Grid (class) [#]

Grid geometry manager. This is an implementation class; all the methods described below are available on all widget classes.

grid(options)** [#]

Place the widget in a grid as described by the options.

***options*
Geometry options.

column=
Insert the widget at this column. Column numbers start with 0. If omitted, defaults to 0.

columnspan=
If given, indicates that the widget cell should span multiple columns. The default is 1.

in=

Place widget inside to the given widget. You can only place a widget inside its parent, or in any decendant of its parent. If this option is not given, it defaults to the parent.

Note that **in** is a reserved word in Python. To use it as a keyword option, append an underscore (**in_**).

in_=
Same as in. See above.

ipadx=
Optional horizontal internal padding. Works like **padx**, but the padding is added *inside* the widget borders. Default is 0.

ipady=
Optional vertical internal padding. Works like **pady**, but the padding is added *inside* the widget borders. Default is 0.

padx=
Optional horizontal padding to place around the widget in a cell. Default is 0.

pady=
Optional vertical padding to place around the widget in a cell. Default is 0.

row=
Insert the widget at this row. Row numbers start with 0. If omitted, defaults to the first empty row in the grid.

rowspan=
If given, indicates that the widget cell should span multiple rows. Default is 1.

sticky=

Defines how to expand the widget if the resulting cell is larger than the widget itself. This can be any combination of the constants **S**, **N**, **E**, and **W**, or **NW**, **NE**, **SW**, and **SE**.

For example, **W** (west) means that the widget should be aligned to the left cell border. **W+E** means that the widget should be stretched horizontally to fill the whole cell. **W+E+N+S** means that the widget should be expanded in both directions. Default is to center the widget in the cell.

grid_bbox(column=None, row=None, col2=None, row2=None) [#]

The grid_bbox method.

column
row
col2
row2

grid_columnconfigure(index, **options) [#]

Set options for a cell column.

To change this for a given widget, you have to call this method on the widget's parent.

index
: Column index.

***options*
: Column options.

minsize=
: Defines the minimum size for the column. Note that if a column is completely empty, it will not be displayed, even if this option is set.

pad=
: Padding to add to the size of the largest widget in the column when setting the size of the whole column.

weight=
: A relative weight used to distribute additional space between columns. A column with the weight 2 will grow twice as fast as a column with weight 1. The default is 0, which means that the column will not grow at all.

grid_configure(options)** [#]

Same as **grid**.

grid_forget() [#]

Remove this widget from the grid manager. The widget is not destroyed, and can be displayed again by **grid** or any other manager.

grid_info() [#]

Return a dictionary containing the current cell options for the cell used by this widget.

Returns:
: A dictionary containing grid grid management options.

grid_location(x, y) [#]

Returns the grid cell under (or closest to) a given pixel.

x

y

Returns:
: A tuple containing the column and row index.

grid_propagate(flag) [#]

Enables or disables geometry propagation. When enabled, a grid manager connected to this widget attempts to change the size of the widget whenever a child widget changes size. Propagation is always enabled by default.

flag
: True to enable propagation.

grid_remove() [#]

Remove this widget from the grid manager. The widget is not destroyed, and can be displayed again by **grid** or any other manager.

grid_rowconfigure(index, **options) [#]

Set options for a row of cells.

To change this for a given widget, you have to call this method on the widget's parent.

index
: Row index.

***options*
: Row options.

minsize=
: Defines the minimum size for the row. Note that if a row is completely empty, it will not be displayed, even if this option is set.

pad=
: Padding to add to the size of the largest widget in the row when setting the size of the whole row.

weight=
: A relative weight used to distribute additional space between rows. A row with the weight 2 will grow twice as fast as a row with weight 1. The default is 0, which means that the row will not grow at all.

grid_size() [#]

Returns the current grid size for the geometry manager attached to this widget. This is defined as indexes of the first empty column and row in the grid, in that order.

Returns:
: A 2-tuple containing the number of columns and rows.

grid_slaves(row=None, column=None) [#]

Returns a list of the “slave” widgets managed by this widget. The widgets are returned as Tkinter widget references.

Returns:
: A list of widgets.

back next