TkLayout Documentation

Release 1.0.0

Dreas Nielsen

Mar 17, 2018

Contents:

TkLayout is a Python library that simplifies the development of Tkinter applications that have a complex layout of widgets. Simplification is achieved by separating the description of the interface structure from the construction of the nested frames that implement that structure.

CHAPTER 1

Purpose

The purpose of the *tklayout* library is to allow the developer of a Tkinter GUI to easily specify the layout of UI widgets, or groups of widgets, and then to automatically create and populate the nested frames necessary to implement the specified layout.

CHAPTER 2

Usage

The usage pattern to be followed when using this library is as follows:

1. Design the layout and assign a name to each of the layout elements. An example is shown in Figure 1.

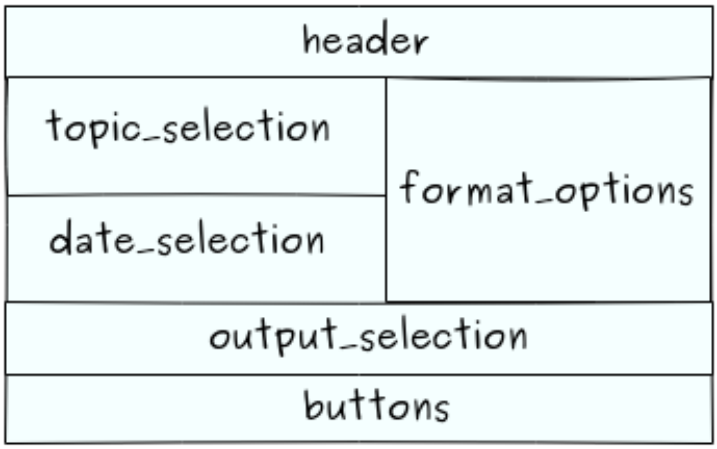

Fig. 2.1: Figure 1. Layout sketch with named elements

2. Instantiate an object of the AppLayout class from the *tklayout* library.

3. Identify layout elements that will appear in the same column or row, and, working from the innermost grouping to the outermost, tell the AppLayout object about all of those relationships, using the `column_elements()` and `row_elements()` methods.

4. Tell the AppLayout object to create a set of nested frames that conform to the specified layout groupings, using the `create_layout()` method.

5. Tell the AppLayout object which functions to run to populate the named frames with the appropriate widgets, using the `build_elements()` method.

In all interactions with the AppLayout object, layout elements are referred to by the names assigned in step 1 above, and as shown in *Figure 1*.

In step 3 above, the first elements in Figure 1 that would be identified as being grouped together would be the *topic_selection* and *date_selection* elements. The second grouping to be identified would be the combination of the previous two with the *format_options* element. Although Tkinter frames must be created from the outside in (i.e., the largest enclosing frame must be created first), the "inside-out" method of describing the layout of user interface elements may be easier to visualize and to use.

The UI implementer does nto need to keep track of all of the frame objects that are used in the layout. The frame objects that enclose the user-specified elements can be obtained from the AppLayout object using the `frame()` method, identifying the element of interest by name.

Other methods that support access to frames and their elements after the nested set of frames has been built are:

- `build_element()`: This allows a build function to be specified (and run) for a single named UI element.
- `frame_widgets()`: This returns the list of all of the elements within the frame enclosing a user-specified element.

CHAPTER 3

The AppLayout Class

Creating a Tkinter layout is done with the `AppLayout` class in the `tklayout` module.

class tklayout.**AppLayout**

Represents the structure of Tkinter widgets (e.g., frames) and provides methods to create and populate the nested set of frames.

class Element(*components, config_dict=None, grid_opts=None, arrangement='vertical', component_weights=[1], element_weight=1*)

An Element object describes one Tkinter element–a Frame or other widget.

An Element object consists of:

- A list of component names (names are strings).
- A dictionary of configuration values for the frame that will contain the components.
- A dictionary of gridding options for the frame that will contain the components.
- A string indicating whether the component elements are arranged horizontally, vertically, or on parallel pages.
- A weight or list of weights for the rows or columns represented by the components. If this list is shorter than the list of components, its members will be recycled as many times as necessary so that there is a weight for each component. The default is [1].
- The weight for the column or row in which all of the elements appear.

build_element(*element_name*, *build_function*)

Runs the specified build_function to populate the frame identified by element_name.

The 'create_layout()' method must have been called so that elements have frames assigned. A build_function cannot be applied to an element that is synthesized by the AppLayout object (i.e., a name returned by 'column_elements()' or 'row_elements()').

Parameters

- **element_name** – A user-assigned name for one of the layout elements.

- **build_function** – A callback function that takes a Frame as an argument and populates that frame with widgets to implement the specific layout element.

build_elements(*build_functions*)
Populates a group of UI elements with widgets.

Parameters **build_functions** – A dictionary where the keys are element names and the values are functions that take a frame as an argument and adds widgets or otherwise prepares that frame for display and use.

The 'create_layout()' method must have been previously called so that elements have frames assigned.

column_elements(*element_names, config_dict=None, grid_dict=None, row_weights=[1], column_weight=1*)
Takes a list of names of elements to be vertically arranged, creates a new element that encloses those, and returns a synthesized name for the element that is created.

Parameters

- **element_names** – A list of names (strings) of elements.
- **config_dict** – A dictionary of configuration specifications for the frame that will contain the named elements.
- **grid_dict** – A dictionary of gridding options for the frame that will contain the named elements.
- **row_weights** – A list of weights for the rows of the frame that will contain the named elements. If this list is shorter than the *element_names* list, its members will be recycled when assigning weights.
- **column_weight** – A weight for the single column in the frame that will contain the named elements.

Returns A name (string) that is automatically assigned to the element that is created to contain the named elements. This name may be used in the *element_names* argument to subsequent calls to *column_elements()*, *row_elements()*. or *page_elements*.

create_layout(*master_widget*, *master_element_name*, *row=0*, *column=0*, *row_weight=1*, *column_weight=1*)
Creates a nested set of frames corresponding to the application layout that has been specified.

The application layout must have been previously described by calls to the `column_elements`, `row_elements`, and `page_elements` methods.

Parameters

- **master_widget** – A container widget (e.g., Frame) that will serve as the root for all the frames to be created.
- **master_element_name** – The name of the element to be placed in the top-level frame. This name must be one of those supplied to 'row_elements()' or 'column_elements()', or returned by those methods.
- **row** – The row number within the master_widget for the top-level frame that will be created. Optional; defaults to 0.
- **column** – The column number within the master widget for the top-level frame that will be created. Optional; defaults to 0.
- **row_weight** – The row weight for the frame's row in the master_widget. Optional; defaults to 1.

- **column_weight** – The column weight for the frame's column in the master widget. Optional; defaults to 1.

frame (*element_name*)
Returns the frame for the given element name.

The frame will only be valid after the 'create_layout()' method has been called.

frame_widgets (*element_name*)
Returns a list of all widgets within the frame for the given element name. The list will only be populated if the 'create_layout()' method has been called and then a suitable 'build' function has been called for this element.

layout_as_json (*show_attributes=False*)
Return a representation of the structure as a JSON string. Attribute names values may optionally be included.

This is intended primarily for debugging.

Returns A string formatted as JSON.

page_elements (*element_names*, *config_dict=None*, *grid_dict=None*)
Takes a list of names of elements to be arranged on separate pages or panes (e.g., pages of a Notebook widget), creates a new element that contains those, and returns a synthesized name for the element that is created.

Parameters

- **element_names** – A list of names (strings) of elements.
- **config_dict** – A dictionary of configuration specifications for the frame that will contain the named elements.
- **grid_dict** – A dictionary of gridding options for the frame that will contain the named elements.

Returns A name (string) that is automatically assigned to the element that is created to contain the named elements. This name may be used in the *element_names* argument to subsequent calls to *column_elements()*, *row_elements()*, or *page_elements*.

row_elements (*element_names, config_dict=None, grid_dict=None, column_weights=[1], row_weight=1*)
Takes a list of names of elements to be horizontally arranged, creates a new element that encloses those, and returns a synthesized name for the element that is created.

Parameters

- **element_names** – A list of names (strings) of elements.
- **config_dict** – A dictionary of configuration specifications for the frame that will contain the named elements.
- **grid_dict** – A dictionary of gridding options for the frame that will contain the named elements.
- **column_weights** – A list of weights for the columns of the frame that will contain the named elements. If this list is shorter than the *element_names* list, its members will be recycled when assigning weights.
- **row_weight** – A weight for the single row in the frame that will contain the named elements.

Returns A name (string) that is automatically assigned to the element that is created to contain the named elements. This name may be used in the *element_names* argument to subsequent calls to *column_elements()*, *row_elements()*. or *page_elements*.

CHAPTER 4

Example

The following code is a simple usage of the `tklayout` module to demonstrate the use of the `AppLayout` class and its methods.

```
try:
    import Tkinter as tk
except:
    import tkinter as tk

import tklayout as tkb


def test():
    # Define functions to build each of the user-defined elements
    # that will appear in the application.  (These 'build' functions
    # are nested within the 'test' function, but need not be.)

    # Build element A.
    def build_a(parent):
        w = tk.Label(parent, text="Element A", justify=tk.CENTER)
        w.grid(row=0, column=0, padx=10, pady=5, sticky=tk.NSEW)
    # Build element B.
    def build_b(parent):
        w = tk.Label(parent, text="Element B", justify=tk.CENTER, fg="blue")
        w.grid(row=0, column=0, padx=10, pady=5, sticky=tk.NSEW)
        parent.rowconfigure(0, weight=1)
        parent.columnconfigure(0, weight=1)
    # Build element C.
    def build_c(parent):
        w =  tk.Label(parent, text="Element C", justify=tk.CENTER)
        w.grid(row=0, column=0, padx=10, pady=5, sticky=tk.NSEW)
        parent.rowconfigure(0, weight=1)
        parent.columnconfigure(0, weight=1)
    # Build element D.
    def build_d(parent):
        w = tk.Label(parent, text="Element D", justify=tk.CENTER, fg="green")
```

```
        w.grid(row=0, column=0, padx=5, pady=5, sticky=tk.EW)
    # Build element E.
    def build_e(parent):html copyright entity
        w = tk.Label(parent, text="Element E", justify=tk.CENTER)
        w.grid(row=0, column=0, padx=5, pady=5, sticky=tk.NSEW)

    # Initialize the application layout object.
    lo = tkb.AppLayout()

    # Define configuration and gridding options that will be used
    # for the frames that enclose the user-defined elements.
    config_opts = {"borderwidth": 3, "relief": tk.GROOVE}
    grid_opts = {"sticky": tk.NSEW}

    # Define the structure of the application elements.
    ab = lo.column_elements(["A", "B"], config_opts, grid_opts)
    abc = lo.row_elements([ab, "C"], config_opts, grid_opts)
    app = lo.column_elements(["D", abc, "E"], config_opts, grid_opts, row_weights=[0,
↪1,1])

    # Create the Tkinter root object.
    root = tk.Tk()

    # Create the frames implementing the specified layout.
    lo.create_layout(root, app, row=0, column=0, row_weight=1, column_weight=1)

    # Fill in the user-defined element frames with widgets.
    lo.build_elements({"A": build_a, "B": build_b, "C": build_c, "D": build_d, "E":␣
↪build_e})

    # Run the application.
    root.mainloop()

test()
```

This will produce an application window with the layout shown below.

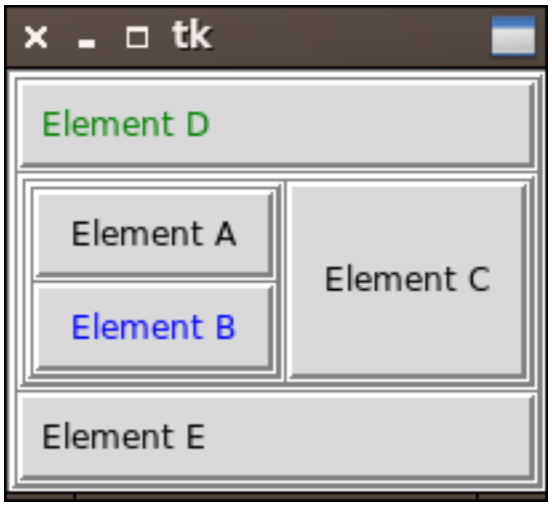

Fig. 4.1: Figure 2. Example Layout

Frame borders are used in this example to illustrate the nested set of frames that is created by `create_layout()`.

The layout of the UI elements can be easily altered just by changing the calls to `column_elements()` and `row_elements()`. For example, changing the calls to the `row_elements()` and `column_elements()` methods in the previous script with this set of method calls:

```
cb = lo.column_elements(["C", "B"], config_opts, grid_opts)
ae = lo.column_elements(["A", "E"], config_opts, grid_opts)
cbae = lo.row_elements([cb, ae], config_opts, grid_opts)
app = lo.column_elements([cbae, "D"], config_opts, grid_opts)
```

will produce a different layout for the same application, as shown below.

Fig. 4.2: Figure 3. Alternative Layout

CHAPTER 5

Notes

5.1 Geometry Managers

The `build_layout()` method uses the *grid* geometry manager for all frames that it creates. However, the *pack* and *place* geometry managers can be used instead to populate any of the frames that enclose user-defined elements.

5.2 Resizing

By default, all frames are made resizeable. Every row and column is given a weight of 1, and every frame is made sticky to its N, S, E, and W enclosing frame. This default differs from Tkinter's: Tkinter by default does not not create or grid frames so that they are automatically resizeable. The default for the AppLayout class differs because resizing is frequently desirable, and it is generally easier to suppress resizing than it is to enable it.

The default weights and configuration options can be changed with the optional arguments to the `row_elements()`, `column_elements()`, and `build_elements()` methods.

CHAPTER 6

Availability

The TkLayout library is available on PyPi. It can be installed with:

```
pip install tklayout
```

The latest code is available from the Bibucket repository.

CHAPTER 7

Change Log

Date	Version	Revision
2018-03-17	1.0.0	Revised documentation.
2018-01-26	0.8.0	Removed borders from automatically created frames.
2018-01-24	0.7.0	Added the 'frame_widgets()' method to the AppLayout class. Eliminated redundant gridding when no grid options. Made frames created by 'create_layout()' sticky to NSEW by default. Modified Tkinter import to run under Python 3.
2018-01-22	0.6.0	Added a version number and added/edited docstrings.
2018-01-21	0.5.0	Completed initial draft of all necessary (maybe) methods of the AppLayout class.
2018-01-17	0.1.0	Created, incomplete.

CHAPTER 8

Copyright and License

CHAPTER 9

Index

- genindex

Python Module Index

t

Index