An Introduction To Graphical User Interface With Python's Tkinter

By: Brad Garrod

11/8/13

I. ABSTRACT

This application note dives into the use of a GUI using Tkinter and Python. The use of a GUI is very important to user and computer system interaction, and is a necessity for ease of use in complex situations. Inside of this application note there are many widgets as well as layout design and complete construction of a graphical user interface.

II. BODY

A. INTRODUCTION

A graphical user interface, or GUI for short, is a visual way for a user to interact with an electronic system through visual icons rather than a command or text based interaction. The purpose of a GUI is to increase the simplicity of controlling an electronic system and make the time to learn of such system much smaller. GUIs are used nowadays on every computer system in almost every program, such as an operating system's buttons and icons to ease navigation throughout the system or starting up other applications, which have their own GUI associated to such.

The "Xerox Alto" was the first claimed computer to "pull together all of the elements of the modern Graphical User Interface" (www.toastytech.com/guis/alto.html). The Alto was one of the first personal computers that could present graphics and share information. The first model of the Alto included a mouse, keyboard, large removable disks, and a microprocessor. Although looking at it now most wouldn't consider the graphics involved much of an interface, the interaction it users to enter and execute commands much easier with simple buttons and textual interface.

While programming a GUI there are many considerations involved, however the basic concept always encompassing the project is ease of use and for the idea of anyone to be able to open and use it. This makes descriptive icons, placement inside of a window, and organization some of the highest priorities. Understanding these concepts not only allows for a GUI to be used easily upon opening, but also for future renditions to grow and become familiar, such as Microsoft has been able to do with their office applications as well as the operating systems in both Macintosh and Windows systems.

B. OBJECTIVE

In this application note, the focus is creating and designing a GUI using Python and its included GUI writing module called Tkinter (will be referenced as Tk). Since GUIs can get very complex with a multitude of challenges as well as tricks to accomplish certain feats, the attention here is understanding the main concepts and how to create a simple GUI. This will include everything from the first basic window, different widgets to incorporate into a GUI, how to associate said widgets to certain actions, and efficiently laying out all of the widgets inside of that first window.

Issues

Steps

To begin with, this application note assumes that Python 2.7 or higher is currently installed on the computer that is in use, and that the user has at least a basic understanding of the Python programming language.

C. *BASIC WINDOW*

The first step in any Tk program is to import the Tkinter module itself. By importing the Tk module you create a path inside of your program that can now reference the Tk folder created upon installation of Python. This allows for use of any of the prewritten modules or widgets in the Tk library. There are a few ways to do this, and each can greatly change the way the program is written.

The most simple way is simply:

```
import Tkinter
```

This creates that reference, however it still makes the functions still need to be qualified by Tkinter (i.e. Tkinter.Tk(), Tkinter.button(), etc.).

To import all of the functions at once from Tk:

```
from Tkinter import *
```

Using this call allows Tk to import everything it contains into the local memory as keywords, so the functions can simply be called instead of qualified (i.e. Tk(), button()). Although this seems the most reasonable, this method limits the variable names that are allowed to be used, and overwriting Tk functions is more common. For this application note, the assumption will be made that "from Tkinter import *" was used.

The next step is to create the basic window that will hold all of the widgets applied into the GUI. This is sometimes referenced as a root window, but the basic behind it is that it is a container and the starting point of actual GUI creation. To create this container, the following command must be executed:

```
rootWindow = Tk()
```

To run all Tkinter applications, you must invoke a command that runs the module root Tk module itself, whether that is in the GUI program or a program that calls the GUI. This is done as follows:

```
rootWindow.mainloop()
```

D. WIDGETS

Widgets are basically extra graphical options to insert inside of the root window that has been previously created. There are a multitude of widgets, but the focus will be on the main ones: label, button, checkbutton, and entry widgets. These will allow the user to visually see and interact with the function running behind the GUI, and also after binding these to events we can create functions associated with each widget.

Note: All of the following widget examples will use the following skeleton code to show the widget:

```
from Tkinter import *
rootWin = Tk()
… widget to be shown …
widgetVariable.pack()
mainloop()
```

The *widgetVariable.pack()* is described in the packing section of this application note.

I) LABEL

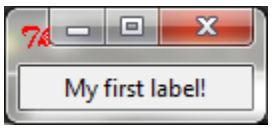
The label widget is a very simple widget and allows for text or image to be entered into the root window for labeling or descriptive purpose. Additionally, there are many options that can be adjusted from this widget including the font, background, foreground, anchor, and many additional options (http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/label.html).

First, to create a label a user must invoke the Label() function with the minimum of what window to apply this function to and the text to include, with any additional options added after:

```
myLabel = Label(parent, option, …)
```

The following is an example of a simple box with a label:

```
firstLabel = Label(rootWin, text = "My first label!")
```



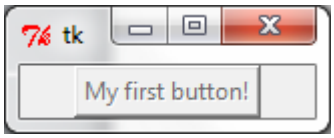
If no other controls are specified, everything will take on the default values and the label will be shown as above if no other labels are present. For further options, please see the Tkinter reference material (http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/label.html).

II) BUTTON

The button widget allow for a clickable button to be displayed in the root window, usually calling a function associated to that button. The button can contain either text or an image, formatted similarly to the label widget described above. The additional options can be found from the Tkinter reference material http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/button.html.

A simple button function is as follows, this time we want to disable the button since there is not a function associated to it by inserting *state=DISABLED* inside of the button call.

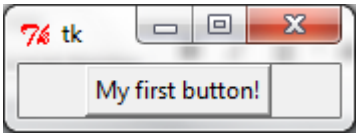
```
   firstButton = Button(rootWin, text=”My first button!”,
state=disabled)
```



Notice that this button is not clickable because the state is disabled. Let’s insert a simple function associated to the button so that it can be clickable. If a print statement is inserted inside of a function, then called by the button we can see the reaction to the button, like follows:

```
def printButton():
      print "BUTTON WORKED!"

firstButton = Button(rootWin, text = "My first button!",
command = printButton)
```



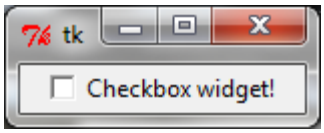
Which returns the output “*BUTTON WORKED!*” when the button is clicked. The function call that is shown is called event binding, and will be explained later.

III) Checkbutton Widget

The checkbutton widget allows for a variable to be declared true or false depending on the state of a check box (either checked or unchecked, respectively). Similarly to all text descriptors, this can be either an image or text entry next to the checkbox with many corresponding options associated with such.

The checkButton is mostly required to have an event binding to have much meaning to it unless grabbing all of the variable states at a later time in the program lifetime, such as a finished state. The following is a simple checkbutton widget application where the state of the checkbutton is stored into *firstCheckState*, which must be declared first.

```
firstCheckBut = Checkbutton(rootWin, text = “Checkbox widget!”,
variable = firstCheckState)
```



To retrieve the state of the checkbox, a *varName.get()* function must be called, such as: firstCheckState.get()

IV) Entry Widget

Another widget in the Tkinter module allows for a user to enter a string, which then, similar to the checkbox, is stored as a widget to be used at the appropriate time. The entry widget allows for a single line of entered text that can contain be set and cleared as wanted inside of the program as well. Usually a button is accompanied with the text entry widget to apply the text itself, although the text entry can be binded to an event within itself.

The following is a simple example of creating a text entry widget and setting the default value:

```
firstEntry = Entry(rootWin,  textvariable = firstEntryValue)
firstEntryValue.set("DEFAULT VALUE")
```

E. LAYOUT MANAGEMENT

There are three effective methods of placing widgets into the root window or an internal frame. These include packing, gridding, and placing. Packing and gridding are similar, whereas they are in relative spots in an area and essentially placed into the frame that it is called, whereas placing is an insertion on a coordinate based system. Essentially the use of each one has particular purposes and advantages, but is mostly user preference. This application note will cover the packing and gridding systems, since they are generally the most applicable and easily understood, and the coordinate based system of packing can get very confusing.

I) Packing

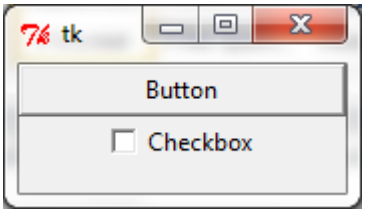
Packing allows the user to place widgets either vertically or horizontally, in the order specified by the program. Additional options allow widgets to fill up either the horizontal or vertical space, or have a certain amount of horizontal or vertical space between widgets in combination to packing them certain ways.

In the first situation we will simply pack a button and a checkbox in one main root window both vertically and horizontally. Also in the horizontal pack we will add vertical and horizontal packing. Later in the example we will see this method placed inside internal frames, which are then packed into the root window.

In addition to the vertical packing we will extend the button widget to extend to the perimeter of the root window. Note packing vertically is the default so it doesn't have to be declared, however being descriptive is a good habit to get into when programming, especially with GUIs. Also, when using the fill option it must be noted that when the window is resized the button is resized as well.

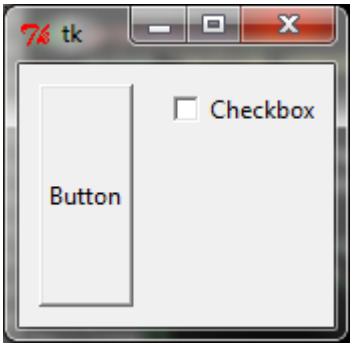
The side option inside of the packing command declares from what direction the item is packed. For example, if *TOP* is the option, it is packed from the top down, and *LEFT* goes from left to right.

```
… two widgets were previously created with variable names
   buttonWidget and checkWidget
buttonWidget.pack(side = TOP, fill=X)
checkWidget.pack(side = TOP)
```



Horizontal packing is very similar to vertical packing, but now the addition of padding will be in place. The idea of padding is to allow space between widgets to clean up the GUI and make it look better. This next example will combine a button and checkbox horizontally, filling the button in the vertical direction as well as adding horizontal and vertical padding.

```
buttonWidget.pack(side=LEFT, fill = Y, fill=X, pady = 10,
   padx = 10)
checkWidget.pack(side = LEFT, pady = 10, padx = 10)
```

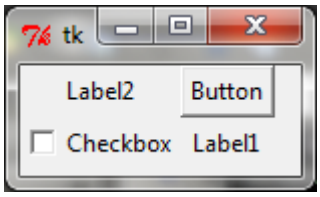


II) *GRID MANAGER*

The second way to place widgets is in a grid layout, where the root window can be divided into rows and columns, and then widgets can be placed inside of those grid spaces. Similar to the method above, multiple widgets can be placed into a single frame which then can be inserted into the grid space.

We will show this simply by entering the same widgets as used before along with two labels to be inserted into the grid space in no particular order to show the ease of use of the grid manager.

```
… two more label widgets were added named label1 and label2
Label1.grid(row=1,column=1)
buttonWidget.grid(row=0,column=1)
checkWidget.grid(row=1, column=0)
Label2.grid(row = 0, column = 0)
```



III. CONCLUSION

All of the examples shown below in combination with each other, especially with further implementation and advancement of techniques can allow for a complex but effective user interface for allow for ease of use. Python is a very simple programming language and Tkinter, although not as robust as some other GUI modules associated with Python, is simple and effective. Many other modules associated with Python can allow for data manipulation, effective communication between a model and an interface, and also gain more information about the background program.

References

Chris Meyers, “Buildling a GUI Application with Tkinter”, 2007,
Website: “Openbookproject.net/py4fun/gui/tkPhone.html”
Website: “Effbot.org/tkinterbook/”, “An Introduction To Tkinter”