Tkinter (Part Two)

Message Boxes

Dialog and message boxes are used to allow the pop-up of:

- **error** window boxes—to alert the user that a selection just made is not functional yet
- **warning** window boxes—to alert the user of the consequences of a selection just made (i.e., "are you sure you want to quit?" etc.)
- **widget selection** boxes—to allow the user to select among a list of options, based on the previous selection
- **dialog boxes**—to allow text entry (strings, integers, or floats)

The message dialogs are provided by the `tkMessageBox` module.

The `tkMessageBox` has the following syntax:

```
tkMessageBox.FunctionName(title, message, **options)
```

Arguments:
`title`—the text filling the title bar
`message`—the message text
`options`—one of the options presented in the options table

The function is one of the following:

Function	Description
`askokcancel`	Asks if operation should proceed. Return TRUE if answer is OK.
`askquestion`	Asks a question.
`askretrycancel`	Ask if operation should be retried. Returns TRUE is answer is YES.
`askyesno`	Asks a YES/NO question. Returns TRUE if answer is YES.

Function	Description
askyesnocancel	Asks a YES/NO/CANCEL question. Returns TRUE if answer is YES.
showerror	Shows an error message.
showinfo	Shows an info message.
showwarning	Shows a warning message.

The option is one of the following:

Option	Description and choices
type	The choices for this option are: **'ok'**, **'okcancel'**, **'yesno'**, **'yesnocancel'**, **'retrycancel'**, **'abortretryignore'**.
message	The message displayed inside the alert.
detail	If specified, a secondary message, displayed in a smaller font under the main message.
title	Title for the dialog window. Not available on MacOS X.
icon	Choices: **'info'** (default), **'error'**, **'question'**, **'warning'**.
default	Specifies which button, **'ok'** or **'cancel'**, for an **okcancel** dialog, is the default.
parent	Which window of the application this message box applies to.

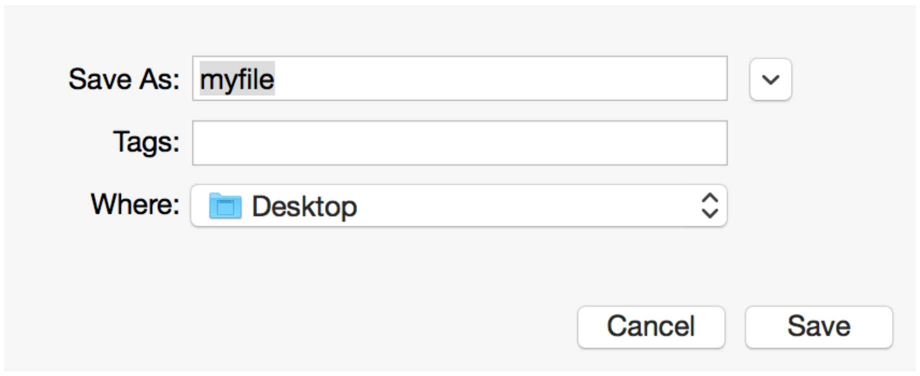
File Dialogs

Tkinter provides the ability to read from, and write into a file, with the tkFileDialog module. The following example shows how to use tkFileDialog and one of its functions to generate the file dialog:

```
import Tkinter, Tkconstants, tkFileDialog
from Tkinter import *
from tkFileDialog import *

class TkFileDialogExample:
    def __init__(self, master):
        frame=Frame(master)
        frame.pack()
        button_opt={'fill': Tkconstants.BOTH, 'padx': 5, 'pady':
5}
        Button(frame,
              text='Save As',
              command=self.callback).pack(**button_opt)

        self.file_opt=options={}
        options['filetypes']=[('all files', '.*'), ('text
files', '.txt')]
        options['initialfile']='myfile.txt'
        options['parent']=master

    def callback(self):
        filename=tkFileDialog.asksaveasfilename(**self.file_opt)
        if filename:
            return open(filename, 'w')

root=Tk()
TkFileDialogExample(root)
root.mainloop()
```

This example uses the method `asksaveasfilename()` of the `filename` object of class `tkFileDialog`, with a dictionary passed in as option arguments. The option arguments were defined as dictionary variable `options` in the class constructor. If the variable `filename` (to be supplied from the file dialog window by the user) is non-null, the method `open()` with filename and option `'w'` (for write) as arguments will open for writing and save that file in the specified directory.

The output window for this source code is:


Save As: myfile

Tags:

Where: Desktop

The general syntax for the methods available in `tkFileDialog` is:

```
FunctionName([mode, ]**options)
```

where:
`FunctionName`—is one of the functions provided in the following table
`mode`—one of `'r'` (for read) or `'w'` (for write)
`**options`—a dictionary of options

The following table gives a list of the methods available from the tkFileDialog module:

Function	Description
`askdirectory(**options)`	Chooses a directory.
`askopenfile(mode='r', **options)`	Asks for a filename to open, and returns the opened file.
`askopenfilename(**options)`	Asks for a filename to open, but returns nothing.
`asksaveasfile(mode='w', **options)`	Asks for a filename to save as, and returns the opened file.
`asksaveasfilename(**options)`	Asks for a filename to save as, but returns nothing.

The list of options is shown in the following table:

Options	Description
`defaultextension [extension]`	The default file extension if the user does not specify one.
`filetypes [filePatternList]`	Returns the file types available on a system's file types listbox.
`initialdir [directory]`	Specifies that the files in the directory should be displayed when the dialog pops up.
`initialfile [filename]`	Specifies the filename to be displayed in the dialog when it pops up.
`message [string]`	A message to include in the client area of the dialog.
`multiple [Boolean]`	Allows the user to specify multiple files from the Open dialog.
`mustexist [Boolean]`	Specifies whether the user may specify non-existen directories.
`parent [window]`	Makes supplied window the parent of the dialog.
`title [titleString]`	A string to display as the title of the dialog.

Layout Management

Tkinter has the following three layout managers:
- pack
- grid
- place

These geometry managers serve the following functions:
- arranging the widgets in the window, which includes determining the size and placement of components: although each type of widget has options specifying size and alignment, it is the geometry manager that determines the size and alignment of all components in the context of the current window
- registering the widget with the underlying windowing system

Pack

The easiest to use of the three geometry managers, Pack allows the size and alignment of widgets to be declared relative to each other, instead of in absolute value. This is a layout manager most appropriate for simple applications, where the arrangement of widgets does not follow sophisticated patterns, but rather only requires vertical or horizontal placement, for example.

Option	Description
`anchor`	Placement of widget inside the packing area. Default is CENTER.
`expand`	Whether the widget should be expanded to fill any extra space in the packing area. Default is FALSE, meaning the widget is not expanded.
`fill`	Whether the widget should occupy all the space provided to it by the master window. Default is NONE. Other option values are X (fill horizontally), Y (fill vertically), or BOTH (scale up in both directions to fill the space).
`in_`	Pack the widget inside the specified widget. The widget can be packed inside its parent (widget) or inside a descendant of its parent.
`ipadx`	Internal padding along the x direction. Default is 0.
`ipady`	Internal padding along the y direction. Default is 0.
`padx`	External padding along the x direction. Default is 0.
`pady`	External padding along the y direction. Default is 0.
`side`	Specifies which side of the parent (widget) to pack the widget against. Option values are TOP, BOTTOM, LEFT, RIGHT. For more complex layouts, rather than use nested Frame widgets in conjunction with options to the Pack geometry manager and its method **`pack()`**, it is recommended to use the Grid geometry manager instead.

For example, in our database table, we want to display differently sized fonts for the table header versus the table body. In order to force each body column to remain aligned with its corresponding header, we could declare a Frame widget for each entire column—including header and subsequent rows—and then force the contents of each table row widget to expand to the size of the parent frame, thus expanding to the width of the header. The question is how to add each new table cell to its corresponding frame.

Grid

The Grid geometry manager places the widgets in a 2-dimensional table, consisting of a number of rows and columns. The position of a widget within the grid is determined by the row and column number. The size of the grid does not need to be specified, because, depending on the number of widgets it contains, and on the size of the largest widget, Grid will automatically determine the best size for each widget used.

However, in our example, because the width of the Label widgets used for the table headers is defined in units dependent on the font size (that is, in number of characters), this results in unevenly sized headers versus table rows.

The following table gives the options for the Grid geometry manager:

Option	Description
column	Column number of the widget within the grid.
columnspan	Allows a widget to span multiple columns, if specified.
in	Places the widget inside the given widget, which can be its parent, or a descendant of its parent. The default is the parent.
in_	Same as in.
ipadx	Internal horizontal padding inside the widget borders.
ipady	Internal vertical padding inside the widget borders.
padx	External horizontal padding around the widget within its cell.
pady	External vertical padding around the widget within its cell.
row	Row number of the widget within the grid.
rowspan	Allows a widget to span multiple rows, if specified.
sticky	Defines how to expand the widget if the resulting cell is larger than the widget itself. Any combination of values S, N, E, W.

Place

Frame

A frame is a rectangular region on the screen. The Frame widget is mainly used:

- as a geometry master for other widgets
- to provide padding between widgets what would otherwise be adjacent
- to group other widgets into complex layouts
- as a place holder for video overlays and other external processes
- even as a separator between two widgets!

The following table shows the options available to Frame and their descriptions:

Option	Description
background	The background color for this frame.
bg	Idem.
borderwidth	Width of frame border. Default is 0.
bd	Idem.
class	Default is Frame.
colormap	For displays using 256 colors or less, this option allows the user to specify which 256 colors to use. By default, each new frame uses the same color map as its parent.
container	
cursor	The shape of the cursor when the mouse is hovering over this area of the window.
height	The height of the frame. Default is 0.
highlightbackground	The background when in focus.
highlightcolor	The color of the text when in focus.
highlightthickness	The thickness of the highlight.
padx	Horizontal padding. Default is 0.

Option	Description
`pady`	Vertical padding. Default is 0.
`relief`	Border appearance. Options are FLAT, SUNKEN, RAISED, GROOVE, and RIDGE. This option is only visible when the border width is non-zero.
`takefocus`	If TRUE, the user can use Tab to move this widget. The default value is FALSE.
`visual`	
`width`	Width of the frame. Default is 0.