Pykinter

1. Instead of drag and drop , the widgets are placed once they are clicked.
2. Taskbar can be edited with colors, icon sizes, icon shapes from given selections
3. The center\_frame(the window that is created by program) ‘s size can be adjusted and can be stretched out or stretched in to zoom in or zoom out but the aspect ratio will remain same.
4. A feature to copy the properties of the last widget in the new widget
5. Allows user to put values in listbox,combobox,dropmenu,optionmenu,
6. Allow to insert image by doubleclicking on image widget
7. Allows to create out own widget and select particular bindings for it
8. Put bindings selection in properties\_tab
9. Put function command property in properties\_tab
10. Proper Linking between different frames , windows
11. Show code of the widget by selecting in right click option
12. Web integration
13. Gives u live visual of Result GUI and almost all interaction can be seen while building it

Undo redo on left side

Custom widgets on left side

Object inspector on right side

Command = function import functionality on right side for widgets

**Normal Flow**

Layout\_main.py

Backend\_properties.py

Functions.py

First\_click.py

Properties\_tab.py

Widgets\_tab.py

**Exact Flow (Import wise)**

Layout\_main.py

Backend\_properties.py

Functions.py

First\_click.py

Properties\_tab.py

Widgets\_tab.py

**Global variables used**

Layout\_main.py

Backend\_properties.py

Functions.py

First\_click.py

Properties\_tab.py

Widgets\_tab.py

**Write Points that are left to do that you feel while using it**

**Annu Task**

1. **In properties\_tab.py, provide bindings to all as per the first one (i.e name\_enter); there are 3 bindings**
2. **Put all the widgets in the (first\_click.py) with the bindings as per the initial widgets.**
3. **Make taskbar in tb() in layout\_main.py according to the details provided.**

**Properties**

**Window**

* **Window name**
* **Window icon**
* **Window opacity**
* **Widget state enabled or disabled**

**Taskbar**

* **Color**
* **border**
* **height**
* **width**
* **title with align**
* **buttons**
* **icons (list of icons to chose)**
* **X**
* **Y**

**Cursor**

* **Cursor size**
* **Cursor icon**
* **Cursors diffent types**

**Basic**

* **Name**
* **X**
* **Y**
* **Height**
* **Width**
* **Max height**
* **Max width**
* **Min width**
* **Min height**

**Background**

* **Background**
* **Border color**
* **Border type (dashed,solid,circle)**
* **Border size**

**Font**

* **Font family**
* **Font size**
* **Bold**
* **Italic**
* **Underline**
* **Strikeout**
* **Antialiasing**

**ON Focus change list bar ( Same for Off Focus, On Click )**

* **Font family**
* **Font size**
* **Bold**
* **Italic**
* **Underline**
* **Strikeout**
* **Antialiasing**
* **Background**
* **Border color**
* **Border size**
* **Name**
* **X**
* **Y**
* **Height**
* **Width**
* **Max height , max width**
* **Min height min width**
* **Cursor size , cursor icon**
* **Window name**
* **Window icon**
* **Window opacity**

**Bindings**

|  |  |
| --- | --- |
| <Button> | A mouse button is pressed with the mouse pointer over the widget. The detail part specifies which button, e.g. The left mouse button is defined by the event <Button-1>, the middle button by <Button-2>, and the rightmost mouse button by <Button-3>.  <Button-4> defines the scroll up event on mice with wheel support and and <Button-5> the scroll down.  If you press down a mouse button over a widget and keep it pressed, Tkinter will automatically "grab" the mouse pointer. Further mouse events like Motion and Release events will be sent to the current widget, even if the mouse is moved outside the current widget. The current position, relative to the widget, of the mouse pointer is provided in the x and y members of the event object passed to the callback. You can use ButtonPress instead of Button, or even leave it out completely: , , and <1> are all synonyms. |
| <Motion> | The mouse is moved with a mouse button being held down. To specify the left, middle or right mouse button use <B1-Motion>, <B2-Motion> and <B3-Motion> respectively. The current position of the mouse pointer is provided in the x and y members of the event object passed to the callback, i.e. event.x, event.y |
| <ButtonRelease> | Event, if a button is released. To specify the left, middle or right mouse button use <ButtonRelease-1>, <ButtonRelease-2>, and <ButtonRelease-3> respectively. The current position of the mouse pointer is provided in the x and y members of the event object passed to the callback, i.e. event.x, event.y |
| <Double-Button> | Similar to the Button event, see above, but the button is double clicked instead of a single click. To specify the left, middle or right mouse button use <Double-Button-1>, <Double-Button-2>, and <Double-Button-3> respectively. You can use Double or Triple as prefixes. Note that if you bind to both a single click (<Button-1>) and a double click (<Double-Button-1>), both bindings will be called. |
| <Enter> | The mouse pointer entered the widget. Attention: This doesn't mean that the user pressed the Enter key!. <Return> is used for this purpose. |
| <Leave> | The mouse pointer left the widget. |
| <FocusIn> | Keyboard focus was moved to this widget, or to a child of this widget. |
| <FocusOut> | Keyboard focus was moved from this widget to another widget. |
| <Return> | The user pressed the Enter key. You can bind to virtually all keys on the keyboard: The special keys are Cancel (the Break key), BackSpace, Tab, Return(the Enter key), Shift\_L (any Shift key), Control\_L (any Control key), Alt\_L (any Alt key), Pause, Caps\_Lock, Escape, Prior (Page Up), Next (Page Down), End, Home, Left, Up, Right, Down, Print, Insert, Delete, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, Num\_Lock, and Scroll\_Lock. |
| <Key> | The user pressed any key. The key is provided in the char member of the event object passed to the callback (this is an empty string for special keys). |
| a | The user typed an "a" key. Most printable characters can be used as is. The exceptions are space (<space>) and less than (<less>). Note that 1 is a keyboard binding, while <1> is a button binding. |
| <Shift-Up> | The user pressed the Up arrow, while holding the Shift key pressed. You can use prefixes like Alt, Shift, and Control. |
| <Configure> | The size of the widget changed. The new size is provided in the width and height attributes of the event object passed to the callback. On some platforms, it can mean that the location changed. |

**Bindings properties**

**Event Attributes**

**widget**

The widget which generated this event. This is a valid Tkinter widget instance, not a name. This attribute is set for all events.

**x, y**

The current mouse position, in pixels.

**x\_root, y\_root**

The current mouse position relative to the upper left corner of the screen, in pixels.

**char**

The character code (keyboard events only), as a string.

**keysym**

The key symbol (keyboard events only).

**keycode**

The key code (keyboard events only).

**num**

The button number (mouse button events only).

**width, height**

The new size of the widget, in pixels (Configure events only).

**type**

The event type.

Bind,bind\_all

**Protocols**[**#**](http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm#protocols)

In addition to event bindings, Tkinter also supports a mechanism called *protocol handlers.* Here, the term protocol refers to the interaction between the application and the window manager. The most commonly used protocol is called **WM\_DELETE\_WINDOW**, and is used to define what happens when the user explicitly closes a window using the window manager.

You can use the **protocol** method to install a handler for this protocol (the widget must be a root or **Toplevel** widget):

widget.protocol("WM\_DELETE\_WINDOW", handler)

Once you have installed your own handler, Tkinter will no longer automatically close the window. Instead, you could for example display a message box asking the user if the current data should be saved, or in some cases, simply ignore the request. To close the window from this handler, simply call the **destroy** method of the window:

**Capturing destroy events**

from Tkinter import \*

import tkMessageBox

def **callback**():

if tkMessageBox.askokcancel("Quit", "Do you really wish to quit?"):

root.destroy()

root = Tk()

root.protocol("WM\_DELETE\_WINDOW", callback)

root.mainloop()

Note that even you don’t register an handler for **WM\_DELETE\_WINDOW**on a toplevel window, the window itself will be destroyed as usual (in a controlled fashion, unlike X). However, as of Python 1.5.2, Tkinter will not destroy the corresponding widget instance hierarchy, so it is a good idea to always register a handler yourself:

top = Toplevel(...)

*# make sure widget instances are deleted*

top.protocol("WM\_DELETE\_WINDOW", top.destroy)

Future versions of Tkinter will most likely do this by default.

**Other Protocols**

Window manager protocols were originally part of the X window system (they are defined in a document titled *Inter-Client Communication Conventions Manual*, or ICCCM). On that platform, you can install handlers for other protocols as well, like **WM\_TAKE\_FOCUS** and **WM\_SAVE\_YOURSELF**. See the ICCCM documentation for details.

**Basic default windows**

* **File choser**
* **Dialog box**
* **Basic Form**

**Widgets**

* **Button**
* **Tool Button**
* **Radio Button**
* **Check Button**
* **Entry Box**
* **Label**
* **ScrollBar**
* **Menu Bar (checkbox,radioButton)**
* **Popup bar**
* **Popup menu**
* **Progress bar**
* **Spin box**
* **Combo box**
* **Drop view**
* **List view**
* **Canvas**
* **Frame**
* **Spiner**
* **Shapes**
* **Image**
* **Text**

**Task**

**Create scrollbar**

**Create dropmenu**

**Create shapes**

**Create toolbutton**

**Create different title bars**

**Make separate if else condition for each widget in update.py**

**Test the application**

**Prepare whole properties\_tab.py**

**Display All the attributes when widget selected**

**Right Click Context Menu**

**1.Button**

**Button Id**

**Copy**

**Last widget properties**

**Delete**

**Change Name**

**Command**

**Select All Buttons**

**Background Image**

**Default size**

**The problem in the right click menu was when i was initialising the widget in first click with other selection dots then, i needed to change the right click menu for different widgets , which was making first click.py lengthy which was not needed. And the menu were continuously added whenever i was right clicking as all the time the function invokes and the menu items get added i.e appended**

**That is the reason i put all the right click functions in right click.py and all its bindings in functions.py**

**But as the context menu is not working in middle frame2 hence finally we put it in alltime() in first\_click.py and then referenced it further to right\_click.py**

**1.Getting the variable of the parent widget is important in creating a GUI system. As it reduces the number of arguements of the functions we pass.**

**When we import files , make sure the importing process don’t form a loop or else it wont workout because code wont return back to mainloop()**

**Using the canvas.delete(‘tag’) which deletes the widgets of the tag ‘tag’ to remove the repetations and hence adding the elements in the context menu each time when right click is invoked.**

**What is parent of the widget?**

**How to use widget.\_nametowidget()**

**Widget.master**