**SCRAPBOOK**

from Tkinter import \*

import Image, ImageTk, os

class Scrapbook:

def \_\_init\_\_(self, master=None):

self.master = master

self.frame = Frame(master, width=400, height=420, bg='gray50',

relief=RAISED, bd=4)

self.lbl = Label(self.frame)

self.lbl.place(relx=0.5, rely=0.48, anchor=CENTER)

self.images = []

images = os.listdir("images")

xpos = 0.05

for i in range(10):

Button(self.frame, text='%d'%(i+1), bg='gray10',

fg='white', command=lambda s=self, img=i: \

s.getImg(img)).place(relx=xpos, rely=0.99, anchor=S)

xpos = xpos + 0.08

self.images.append(images[i])

Button(self.frame, text='Done', command=self.exit,

bg='red', fg='yellow').place(relx=0.99, rely=0.99, anchor=SE)

self.frame.pack()

self.getImg(0)

def getImg(self, img):

self.masterImg = Image.open(os.path.join("images",

self.images[img]))

self.masterImg.thumbnail((400, 400))

self.img = ImageTk.PhotoImage(self.masterImg)

self.lbl['image'] = self.im

def exit(self):

self.master.destroy()

root = Tk()

**BROWSER**

from Tkinter import \*

import Pmw

import os

import AppShell

import Image, ImageTk

path = "./icons/"

imgs = "./images/"class Node:

def \_\_init\_\_(self, master, tree, icon=None,

openicon=None, name=None, action=None):

self.master, self.tree = master, tree

self.icon = PhotoImage(file=icon)

if openicon:

self.openicon = PhotoImage(file=openicon)

else:

self.openicon = None

self.width, self.height = 1.5\*self.icon.width(), \

1.5\*self.icon.height()

self.name = name

self.var = StringVar()

self.var.set(name)

self.text = Entry(tree, textvariable=self.var, bg=tree.bg,

bd=0, width=len(name)+2, font=tree.font,

fg=tree.textcolor, insertwidth=1,

highlightthickness=1,

highlightbackground=tree.bg,

selectbackground="#044484",

selectborderwidth=0,

selectforeground='white')

self.action = action

self.x = self.y = 0 #drawing location

self.child = []

self.state = 'collapsed'

self.selected = 0

def addChild(self, tree, icon=None, openicon=None, name=None,

action=None):

child = Node(self, tree, icon, openicon, name, action)

self.child.append(child)

self.tree.display()

return child

def deleteChild(self, child):

self.child.remove(child)

self.tree.display()

def textForget(self):

self.text.place\_forget()

for child in self.child:

child.textForget()

def deselect(self):

self.selected = 0

for child in self.child:

child.deselect()

def boxpress(self, event=None):

if self.state == 'expanded':

self.state = 'collapsed'

elif self.state == 'collapsed':

self.state = 'expanded'

self.tree.display()

def invoke(self, event=None):

if not self.selected:

self.tree.deselectall()

self.selected = 1

self.tree.display()

if self.action:

self.action(self.name)

self.name = self.text.get()

self.text.config(width=len(self.name)+2)

def displayIconText(self):

tree, text = self.tree, self.text

if self.selected and self.openicon:

self.pic = tree.create\_image(self.x, self.y,

image=self.openicon)

else:

self.pic = tree.create\_image(self.x, self.y,

image=self.icon)

text.place(x=self.x+self.width/2, y=self.y, anchor=W)

text.bind("<ButtonPress-1>", self.invoke)

tree.tag\_bind(self.pic, "<ButtonPress-1>", self.invoke, "+")

text.bind("<Double-Button-1>", self.boxpress)

tree.tag\_bind(self.pic, "<Double-Button-1>",

self.boxpress, "+")

def displayRoot(self):

if self.state == 'expanded':

for child in self.child:

child.display()

self.displayIconText()

def displayLeaf(self):

self.tree.hline(self.y, self.master.x+1, self.x)

self.tree.vline(self.master.x, self.master.y, self.y)

self.displayIconText()

def displayBranch(self):

master, tree = self.master, self.tree

x, y = self.x, self.y

tree.hline(y, master.x, x)

tree.vline(master.x, master.y, y)

if self.state == 'expanded' and self.child != []:

for child in self.child:

child.display()

box = tree.create\_image(master.x, y,

image=tree.minusnode)

elif self.state == 'collapsed' and self.child != []:

box = tree.create\_image(master.x, y,

image=tree.plusnode)

tree.tag\_bind(box, "<ButtonPress-1>", self.boxpress, "+")

self.displayIconText()

def findLowestChild(self, node):

if node.state == 'expanded' and node.child != []:

return self.findLowestChild(node.child[-1])

else:

return node

def display(self):

master, tree = self.master, self.tree

n = master.child.index(self)

self.x = master.x + self.width

if n == 0:

self.y = master.y + (n+1)\*self.height

else:

previous = master.child[n-1]

self.y = self.findLowestChild(previous).y + self.height

if master == tree:

self.displayRoot()

elif master.state == 'expanded':

if self.child == []:

self.displayLeaf()

else:

self.displayBranch()

tree.lower('line')

class Tree(Canvas):

def \_\_init\_\_(self, master, icon, openicon, treename, action,

bg='white', relief='sunken', bd=2,

linecolor='#808080', textcolor='black',

font=('MS Sans Serif', 8)):

Canvas.\_\_init\_\_(self, master, bg=bg, relief=relief, bd=bd,

highlightthickness=0)

self.pack(side='left', anchor=NW, fill='both', expand=1)

self.bg, self.font= bg, font

self.linecolor, self.textcolor= linecolor, textcolor

self.master = master

self.plusnode = PhotoImage(file=os.path.join(path, 'plusnode.gif'))

self.minusnode = PhotoImage(file=os.path.join(path, 'minusnode.gif'))

self.inhibitDraw = 1

self.imageLabel = None

self.imageData = None

self.child = []

self.x = self.y = -10

self.child.append( Node( self, self, action=action,

icon=icon, openicon=openicon, name=treename) )

def display(self):

if self.inhibitDraw: return

self.delete(ALL)

for child in self.child:

child.textForget()

child.display()

def deselectall(self):

for child in self.child:

child.deselect()

def vline(self, x, y, y1):

for i in range(0, abs(y-y1), 2):

self.create\_line(x, y+i, x, y+i+1, fill=self.linecolor,

tags='line')

def hline(self, y, x, x1):

for i in range(0, abs(x-x1), 2):

self.create\_line(x+i, y, x+i+1, y, fill=self.linecolor,

tags='line')

class ImageBrowser(AppShell.AppShell):

usecommandarea=1

appname = 'Image Browser'

def createButtons(self):

self.buttonAdd('Ok',

helpMessage='Exit',

statusMessage='Exit',

command=self.quit)

def createMain(self):

self.panes = self.createcomponent('panes', (), None,

Pmw.PanedWidget,

(self.interior(),),

orient='horizontal')

self.panes.add('browserpane', min=150, size=160)

self.panes.add('displaypane', min=.1)

f = os.path.join(path, 'folder.gif')

of = os.path.join(path, 'openfolder.gif')

self.browser = self.createcomponent('browse', (), None,

Tree,

(self.panes.pane('browserpane'),),

icon=f,

openicon=of,

treename='Multimedia',

action=None)

self.browser.pack(side=TOP, expand=YES, fill=Y)

self.datasite = self.createcomponent('datasite', (), None,

Frame,

(self.panes.pane('displaypane'),))

self.datasite.pack(side=TOP, expand=YES, fill=BOTH)

f = os.path.join(path, 'folder.gif')

of = os.path.join(path, 'openfolder.gif')

gf = os.path.join(path, 'gif.gif')

jf = os.path.join(path, 'jpg.gif')

xf = os.path.join(path, 'other.gif')

self.browser.inhibitDraw = 1

top=self.browser.child[0]

top.state='expanded'

jpeg=top.addChild(self.browser, icon=f, openicon=of,

name='Jpeg',action=None)

gif=top.addChild(self.browser, icon=f, openicon=of,

name='GIF', action=None)

other=top.addChild(self.browser, icon=f, openicon=of,

name='Other', action=None)

imageDir = { '.jpg': (jpeg, jf), '.jpeg': (jpeg, jf),

'.gif': (gif, gf), '.bmp': (other, xf),

'.ppm': (other, xf)}

files = os.listdir(imgs)

for file in files:

r, ext = os.path.splitext(file)

cont, icon = imageDir.get(ext, (None, None))

if cont:

cont.addChild(self.browser, icon=icon,

name=file, action=self.showMe)

self.browser.inhibitDraw = 0

self.browser.display()

self.panes.pack(side=TOP, expand=YES, fill=BOTH)

def createImageDisplay(self):

self.imageDisplay = self.createcomponent('image', (), None,

Label,

(self.datasite,))

self.browser.imageLabel = self.imageDisplay

self.browser.imageData= None

self.imageDisplay.place(relx=0.5, rely=0.5, anchor=CENTER)

def createInterface(self):

AppShell.AppShell.createInterface(self)

self.createButtons()

self.createMain()

self.createImageDisplay()

def showMe(self, dofile)

if self.browser.imageData: del self.browser.imageData

self.browser.imageData = ImageTk.PhotoImage(\

Image.open('%s%s' % \

(imgs, dofile)))

self.browser.imageLabel['image'] = self.browser.imageData

if \_\_name\_\_ == '\_\_main\_\_':

imageBrowser = ImageBrowser()

imageBrowser.run()