



<b>ID</b>		<b>Creation date</b>	June 08, 2009
<b>Platform</b>	S60 3rd and 5th Editions	<b>Tested on devices</b>	E71, XpressMusic 5800
<b>Category</b>	Python	<b>Subcategory</b>	appuifw

**Keywords (APIs, classes, methods, functions):** appuifw, toolbar, canvas, graphics

## Introduction

Toolbars are a common components in modern operating systems, in general represented as a panel with small icons or similar components that can be moved and enabled/disabled. Toolbars add functionality and facilities for users, making enjoyable the user experience. However, few S60 applications have support for toolbars, even in touch enabled devices, like XpressMusic 5800, where touch screen is an invite for toolbars.

In this article it is presented a toolbar implementation for PyS60 applications called CanvasToolbar. It can be used with S60 3rd and 5th devices, with support for moving and enabling/disabling functions. Moreover, it is possible to create toolbars with transparency and to choose the orientation (vertical or horizontal).

You can see a complete demo application using this toolbar called [\*\*Scribble demo revisited\*\*](#).

## Usage

CanvasToolbar is a **remote controlled** component. It does not have its own event loop for processing key presses or touches. Instead, a callback function is used to notify selections or focus changes. This design decision keeps the component small and simple to use while leaving all control actions to the main loop of canvas based applications.

For using it, create a set of icons with same size and load them as a list of Image objects. After, create a CanvasToolbar with the following parameters:

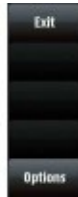
- **canvas** (Image): image buffer where toolbar will be drawn. In general, this is our screen buffer.
- **sel\_cbk** (function): a callback to be called when an icon is selected or toolbar is touched.
- **redraw\_cbk** (function): a callback to be called when redrawing is necessary.
- **imgs** (list of Images): image list with icons.
- **position** ((int,int)): initial position for drawing the toolbar. Default is (0,0).
- **background\_color** ((int,int,int)): toolbar background color. Default is light yellow (255,255,128).
- **orientation** (int): O\_HORIZONTAL or O\_VERTICAL for horizontal or vertical (default) toolbars
- **transparency** (int): transparency percentage (0 = no transparent, 100 = totally transparent). Default is 0.
- **margin** (int): external margin. Default is 3px.
- **img\_border** (int): border to be added to images. Default is 2px.

## Toolbar\_on\_canvas\_for\_touch\_and\_non\_touch\_S60\_devices

It will be necessary to write some event loop for touch devices or to create key binds for non touch devices. In next sections, CanvasToolbar source code and usage examples are presented. Below you can see it in action, running in XpressMusic 5800 and E71. See the following [video](#) as well.



240px Toolbar running on XpressMusic 5800



240px Toolbar running on E71

## Source code

CanvasToolbar is below. It can be download from [this link](#). Copy this code to e:\python\lib.

```
# -*- coding: utf-8 -*-
# (c) Marcelo Barros de Almeida
# marcelobarrosalmeida@gmail.com
# License: GPL3
import graphics
import key_codes

try:
    a = key_codes.EButton1Down
except:
    # adding missing key_codes just to avoid tests related
    # to touch enabled or not
    key_codes.EButton1Down=0x101
    key_codes.EButton1Up=0x102
    key_codes.EDrag=0x107
    key_codes.EModifierDoubleClick=0x00080000

O_HORIZONTAL = 1
O_VERTICAL = 2

class CanvasToolbar(object):
    """ Creates a toolbar given a set of square images with same size.
        Toolbar is draw as below:

                img_border  margin
                |---|---|
+-----+-----+ --
|         |         | | <-- margin (external margin)
| +-----+         | | --
| |         |         | | <-- img_border (image border, for drawing selection)
| | +-----+         | | --
| | | ICON |         | |
| | |         |         | |
| | +-----+         | | --
| |         |         | | <-- 2*img_border
| | +-----+         | | --
| | | ICON |         | |
| | |         |         | |
```

## Toolbar\_on\_canvas\_for\_touch\_and\_non\_touch\_S60\_devices

```
| | +-----+ | |
| |           | |
| +-----+ | |
|
+-----+
```

```
"""
# Possible orientations
def __init__(self, canvas, sel_cbk, redraw_cbk, imgs, position=(0,0), bg=(255,255,128),
             orientation=O_VERTICAL, transparency=0, margin=3, img_border=2):
    """ Created the toolbar object.
        Parameters:
            - canvas (Image): image buffer where toolbar will be drawn
            - sel_cbk (function): callback to be called when an icon is selected toolbar is t
            - redraw_cbk (function): callback to be called when redrawing is necessary
            - imgs (list of Images): image list with icons
            - position ((int,int)): initial position for drawing the toolbar
            - background color ((int,int,int)): toolbar background color
            - orientation (int): O_HORIZONTAL or O_VERTICAL
            - transparency (int): transparency percentage (0 = no transparent, 100 = totally
            - margin (int): external margin
            - img_border (int): border to be added to images
    """
    self.canvas = canvas
    self.sel_cbk = sel_cbk
    self.redraw_cbk = redraw_cbk
    self.imgs = imgs
    self.position = [position[0],position[1],0,0]
    self.bcolor = bg
    self.orientation = orientation
    self.transparency = (255*(100-transparency)/100,)*3
    self.margin = margin
    self.img_border = img_border
    self.img_selected = 0
    self.selected = -1
    self.last_img_selected = -1
    self.visible = False
    self.calc_size()
    self.redraw_cbk()

def calc_size(self):
    """ Using the list of images, calculate toolbar size and selection image
    """
    n = len(self.imgs)
    self.img_size = self.imgs[0].size[0]
    if self.orientation == O_HORIZONTAL:
        my = self.img_size + 2*(self.img_border + self.margin)
        mx = self.img_size*n + 2*(self.img_border*n + self.margin)
    else:
        mx = self.img_size + 2*(self.img_border + self.margin)
        my = self.img_size*n + 2*(self.img_border*n + self.margin)
    self.size = (mx,my)
    self.position[2] = self.position[0] + mx
    self.position[3] = self.position[1] + my
    self.create_sel_img(self.img_size+2*self.img_border)
    self.tlb_img = graphics.Image.new(self.size)
    self.msk_img = graphics.Image.new(self.size, 'L')
    self.msk_img.clear(self.transparency)

def create_sel_img(self, sz):
    """ Creates selection image (small square with dashed border)
    """
```

## Toolbar\_on\_canvas\_for\_touch\_and\_non\_touch\_S60\_devices

```
self.sel_img = graphics.Image.new((sz,sz))
self.sel_img.clear(self.bcolor)
cb = self.bcolor
cf = (0,0,0)
c = cb
step = 7
for p in range(sz):
    if p%step == 0:
        if c == cb:
            c = cf
        else:
            c = cb
    for b in range(self.img_border):
        self.sel_img.point((p,b),outline=c)
        self.sel_img.point((p,sz-self.img_border+b),outline=c)
        self.sel_img.point((b,p),outline=c)
        self.sel_img.point((sz-self.img_border+b,p),outline=c)

def move(self,pos):
    """ Move toolbar to a new position given by pos (int,int)
    """
    self.position = [pos[0],pos[1],pos[0]+self.size[0],pos[1]+self.size[1]]
    self.redraw_cbk()

def is_visible(self):
    """ Check if toolbar is visible (True) or not (False)
    """
    return self.visible

def is_inside(self,p):
    """ Checks if a given point p (int,int) is inside toolbar area or not
    """
    if p[0] > self.position[0] and \
        p[0] < self.position[2] and \
        p[1] > self.position[1] and \
        p[1] < self.position[3]:
        return True
    else:
        return False

def redraw(self,rect=None):
    """ Redraw the toolbar
    """
    if not self.visible:
        return
    self.tlb_img.clear(self.bcolor)
    x = self.margin + self.img_border
    y = self.margin + self.img_border
    for n in range(len(self.imgs)):
        img = self.imgs[n]
        if self.img_selected == n:
            self.tlb_img.blit(self.sel_img,
                             target=(x-self.img_border,y-self.img_border),
                             source=((0,0),self.sel_img.size))
        self.tlb_img.blit(img,target=(x,y),source=((0,0),img.size))
    if self.orientation == O_HORIZONTAL:
        x += self.img_size + 2*self.img_border
    else:
        y += self.img_size + 2*self.img_border
    self.canvas.blit(self.tlb_img,
                     target=self.position[:2],
                     source=((0,0),self.tlb_img.size),
```

## Toolbar\_on\_canvas\_for\_touch\_and\_non\_touch\_S60\_devices

```
        mask=self.msk_img)

def redraw2(self,rect=None):
    """ Redraw the toolbar
    """
    if not self.visible:
        return
    self.canvas.rectangle(self.position,
                          outline = self.bcolor,
                          fill = self.bcolor)
    x = self.position[0] + self.margin + self.img_border
    y = self.position[1] + self.margin + self.img_border
    for n in range(len(self.imgs)):
        img = self.imgs[n]
        if self.img_selected == n:
            self.canvas.blit(self.sel_img,
                             target=(x-self.img_border,y-self.img_border),
                             source=((0,0),self.sel_img.size))
        self.canvas.blit(img,target=(x,y),source=((0,0),img.size))
        if self.orientation == O_HORIZONTAL:
            x += self.img_size + 2*self.img_border
        else:
            y += self.img_size + 2*self.img_border

def show(self):
    """ Make the toolbar visible
    """
    if not self.visible:
        self.visible = True
        self.img_selected = 0
        self.selected = -1
        self.last_img_selected = -1
    self.redraw_cbk()

def hide(self):
    """ Make the toolbar invisible
    """
    self.visible = False
    self.redraw_cbk()

def next(self):
    """ Move the focus to the next icon
    """
    self.img_selected = (self.img_selected + 1)%len(self.imgs)
    self.redraw_cbk()

def prev(self):
    """ Move the focus to the previous icon
    """
    self.img_selected = (self.img_selected - 1)%len(self.imgs)
    self.redraw_cbk()

def set_sel(self,pos=None):
    """ Call this function for ensuring the selection.
        For non touch UI, the selection callback is called immediately.
        For touch UI, pos (int,int) argument is used to check if we
        have a selection (typing over an icon with focus)
        or if we are just changing the focus (typing over a new icon)
    """
    if pos:
        if self.orientation == O_HORIZONTAL:
            n = int((pos[0] - self.margin - self.position[0])/(self.img_size+2*self.img_borde
```

## Toolbar\_on\_canvas\_for\_touch\_and\_non\_touch\_S60\_devices

```
else:
    n = int((pos[1] - self.margin - self.position[1])/(self.img_size+2*self.img_borde
self.img_selected = min(max(n,0),len(self.imgs)-1)
if self.img_selected == self.last_img_selected:
    self.selected = self.img_selected
    self.sel_cbk()
else:
    self.last_img_selected = self.img_selected
    self.redraw_cbk()
else:
    self.selected = self.img_selected
    self.sel_cbk()

def get_sel(self):
    """ Return the selected icon or -1 (no selection yet)
    """
    return self.selected
```

## CanvasToolbar and non touch devices

In the following code snippet, CanvasToolbar is used with E71. Left navi key was programmed to show/hide the toolbar and up/down navi keys are used to change the focus. Selection key is used to select. It can be download from [this link](#). Copy code and icons to e:\python\.

```
# -*- coding: utf-8 -*-
# (c) Marcelo Barros de Almeida
# marcelobarrosalmeida@gmail.com
# License: GPL3
import e32
import graphics
import key_codes
from appuifw import *
import sysinfo
from ctoolbar import CanvasToolbar

class ToolbarDemo(object):
    """ Toolbar demo for non touch UI
    """
    def __init__(self):
        self.lock = e32.Ao_lock()
        app.title = u"ToolbarDemo"
        app.screen = "full"
        self.toolbar = None
        self.load_imgs()
        sz = max(sysinfo.display_pixels())
        self.scr_buf = graphics.Image.new((sz,sz))
        self.canvas = Canvas(redraw_callback=self.redraw)
        self.toolbar = CanvasToolbar(self.scr_buf,
                                   self.item_selected,
                                   self.redraw,
                                   self.imgs,
                                   (0,0))

        app.body = self.canvas
        app.menu = [(u"Show toolbar", lambda: self.toolbar.show()),
                   (u"Hide toolbar", lambda: self.toolbar.hide()),
                   (u"Help", self.help),
                   (u>About", self.about),
                   (u"Quit", self.close_app)]
        self.toolbar.show()
```

## Toolbar\_on\_canvas\_for\_touch\_and\_non\_touch\_S60\_devices

```
self.canvas.bind(key_codes.EKeyLeftArrow, self.left_key)
self.canvas.bind(key_codes.EKeyUpArrow, self.up_key)
self.canvas.bind(key_codes.EKeyDownArrow, self.down_key)
self.canvas.bind(key_codes.EKeySelect, self.sel_key)
self.lock.wait()

def left_key(self):
    if not self.toolbar.is_visible():
        self.toolbar.show()
    else:
        self.toolbar.hide()

def up_key(self):
    if self.toolbar.is_visible():
        self.toolbar.prev()

def down_key(self):
    if self.toolbar.is_visible():
        self.toolbar.next()

def sel_key(self):
    if self.toolbar.is_visible():
        self.toolbar.set_sel()

def redraw(self, rect=None):
    """ Erase your canvas, draw your stuff and
        ask toolbar for redrawing if it is visible.
        Finally, blit your buffer in canvas.
    """
    self.scr_buf.clear((255,255,255))
    if self.toolbar:
        self.toolbar.redraw()
    self.canvas.blit(self.scr_buf)

def load_imgs(self):
    imgs_file = [ u"e:\\python\\refresh22.png",
                  u"e:\\python\\day22.png",
                  u"e:\\python\\week22.png",
                  u"e:\\python\\month22.png",
                  u"e:\\python\\setup22.png" ]

    self.imgs = []
    for fn in imgs_file:
        self.imgs.append(graphics.Image.open(fn))

def item_selected(self):
    item = self.toolbar.get_sel()
    if item == -1:
        note(u"No item selected", "info")
    else:
        note(u"Item %d selected" % item, "info")

def help(self):
    note(u"Left arrow to show/hide the toolbar", "info")

def about(self):
    note(u"Canvas toolbar demo by Marcelo Barros (marcelobarrosalmeida@gmail.com)", "info")

def close_app(self):
    self.lock.signal()
    app.set_exit()
```

ToolbarDemo()

## CanvasToolbar and touch devices

In the following code snippet, CanvasToolbar is used with XpressMusic 5800. Double touch show/hide the toolbar. It can be download from [this link](#). Copy code and icons to e:\python\.

```
# -*- coding: utf-8 -*-
# (c) Marcelo Barros de Almeida
# marcelobarrosalmeida@gmail.com
# License: GPL3
import e32
import graphics
import key_codes
from appuifw import *
import sysinfo
from ctoolbar import CanvasToolbar

class ToolbarDemo(object):
    """ Toolbar demo for touch UI
    """
    def __init__(self):
        self.lock = e32.Ao_lock()
        app.title = u"ToolbarDemo"
        app.screen = "full"
        self.toolbar = None
        self.drag_filter_cnt = 0
        self.load_imgs()
        sz = max(sysinfo.display_pixels())
        self.scr_buf = graphics.Image.new((sz, sz))
        self.canvas = Canvas(redraw_callback=self.redraw,
                             event_callback=self.event)
        self.toolbar = CanvasToolbar(self.scr_buf,
                                     self.item_selected,
                                     self.redraw,
                                     self.imgs,
                                     (0, 0))

        app.body = self.canvas
        app.menu = [(u"Show toolbar", lambda: self.toolbar.show()),
                    (u"Hide toolbar", lambda: self.toolbar.hide()),
                    (u"Help", self.help),
                    (u>About", self.about),
                    (u"Quit", self.close_app)]
        self.toolbar.show()
        self.lock.wait()

    def event(self, ev):
        # just checking touch events
        if ev['type'] not in [key_codes.EButton1Up,
                              key_codes.EButton1Down,
                              key_codes.EDrag]:
            return
        # checking double click (show/hide toolbar)
        if ev['modifiers'] == key_codes.EModifierDoubleClick and \
            ev['type'] == key_codes.EButton1Down and \
            not self.toolbar.is_inside(ev['pos']):
            if not self.toolbar.is_visible():
                self.toolbar.show()
            else:
```

## Toolbar\_on\_canvas\_for\_touch\_and\_non\_touch\_S60\_devices

```
        self.toolbar.hide()
    return
# filtering drag event before moving (we need at least
# five consecutive drag events to move
if ev['type'] == key_codes.EDrag and self.toolbar.is_visible():
    if self.toolbar.is_inside(ev['pos']):
        self.drag_filter_cnt = 1
    elif self.drag_filter_cnt >= 1:
        self.drag_filter_cnt += 1
    else:
        self.drag_filter_cnt = 0
    if self.drag_filter_cnt >= 5:
        self.toolbar.move(ev['pos'])
    return
else:
    self.drag_filter_cnt = 0
# touch UI selection
if ev['type'] == key_codes.EButton1Down and \
    self.toolbar.is_inside(ev['pos']) and \
    self.toolbar.is_visible():
    self.toolbar.set_sel(ev['pos'])

def redraw(self, rect=None):
    """ Erase your canvas, draw your stuff and
    ask toolbar for redrawing if it is visible.
    Finally, blit your buffer in canvas.
    """
    self.scr_buf.clear((255,255,255))
    if self.toolbar:
        self.toolbar.redraw()
    self.canvas.blit(self.scr_buf)

def load_imgs(self):
    imgs_file = [ u"e:\\python\\refresh44.png",
                  u"e:\\python\\day44.png",
                  u"e:\\python\\week44.png",
                  u"e:\\python\\month44.png",
                  u"e:\\python\\setup44.png" ]

    self.imgs = []
    for fn in imgs_file:
        self.imgs.append(graphics.Image.open(fn))

def item_selected(self):
    item = self.toolbar.get_sel()
    if item == -1:
        note(u"No item selected", "info")
    else:
        note(u"Item %d selected" % item, "info")

def help(self):
    note(u"Type twice in canvas to show/hide the toolbar", "info")

def about(self):
    note(u"Canvas toolbar demo by Marcelo Barros (marcelobarrosalmeida@gmail.com)", "info")

def close_app(self):
    self.lock.signal()
    app.set_exit()

ToolbarDemo()
```

## References

- [Source code \(zip file\)](#)
- [Scribble demo revisited](#). An application using transparent toolbars.
- [How to display transparent PNG on canvas with masks](#). Interesting article that helped a lot when implementing transparency.