

A_100%_Python_implemented_Listbox_base_class

When getting the specifications for my Python framework for S60, it appeared that the listbox wrappers provided by PyS60 did not satisfy the requirements.

I needed for example to have a listbox style with three lines of text items displaying the description of some articles.

Another one was to have the same 3 lines description item but also with a picture thumbnail on the left part of it.

In addition I needed to have a callback called every time the current focused object changed in order to set/dim the menu depending on one condition to another.

The easiest way (not the best performances) was to re-implement the listbox object in Python.

The source below is the base class I'm using for all the listboxes styles. I freshly modified it today since I had a chance to have a N93. Switching the view to portrait or landscape is now handled as well as the recalculation of the elements to be displayed on the canvas.

Contents

- [1](#)
[Source](#)
- [2](#) [Usage](#)
- [3](#)
[Screenshots](#)
- [4](#)
[Remarks](#)

Source

```
import appuifw, e32
from key_codes import EScancodeUpArrow, EScancodeDownArrow, EScancodeSelect
from graphics import Image
try:
    from akntextutils import wrap_text_to_array
except:
    import sys
    sys.exit("akntextutils module isn't installed!")

## Keyboard handler class.
class Keyboard( object ):
    ## The constructor
    # @param self The object pointer
    # @param aOnevent An optional function that can be called on events
    def __init__( self, aOnevent=lambda:None ):
        ## Keyboard state dictionary
        self._keyboard_state = { }
        ## Key down dictionary
        self._downs = { }
        ## User defined callback function run when an event occurs
        self._onevent = aOnevent
```

A_100%_Python_implemented_Listbox_base_class

```
## Event handler
# @param self The object pointer
# @param aEvent Event detected
def handle_event( self, aEvent ):
    if aEvent['type'] == appuifw.EEventKeyDown:
        code = aEvent['scancode']
        if not self.is_down( code ):
            self._downs[code] = self._downs.get( code, 0 ) + 1
            self._keyboard_state[code] = 1
        elif aEvent['type'] == appuifw.EEventKeyUp:
            self._keyboard_state[aEvent['scancode']] = 0
        self._onevent( )

## Detects if the given keycode key is down
# @param self The object pointer
# @param aScancode key code
def is_down( self, aScancode ):
    return self._keyboard_state.get( aScancode, 0 )

## Returns true if the given keycode key has been pressed
# @param self The object pointer
# @param aScancode Key code
def pressed( self, aScancode ):
    if self._downs.get( aScancode, 0 ):
        self._downs[aScancode] -= 1
        return True
    return False

## Base class for creating custom listboxes.
# @todo Find a way to automatically calculate the scroll factor and the line
# spacing with scallable fonts.
class ListBaseClass( object, appuifw.Canvas ):
    # private const class variables
    ## Background color for the list
    _iBackgroundColor = 0xffffffff
    ## Current item focus
    _iCurrentFocus = 0
    ## Default text font
    _iDefaultFont = 'dense'
    ## Value used to display the selection rectangle at the right line
    _iFactor = 1
    ## Selection background color
    _iFocusBackgroundColor = 0xc3d9ff
    ## Selection outline color
    _iFocusOutlineColor = 0x0000ff
    ## Line space
    _iLineSpace = 13
    ## Lines color ( here black )
    _iLineColor = 0x000000
    ## Maximum item to be displayed. By default 5 it is recalculated for each
    # new view.
    _iMaxItem = 5
    ## When True, elements are drawn
    _iReady = False
    ## Scrollbar outline color
    _iScrollbarOutlineColor = 0x000000
    ## Scrollbar cursor outline color
    _iScrollbarCursorOutlineColor = 0x0000ff
    ## Scrollbar cursor fill color
    _iScrollbarCursorFillColor = 0xc3d9ff
    ## Scroll translation factor in pixel
```

A_100%_Python_implemented_Listbox_base_class

```
_iScrollFactor = 25
## Text color
_iTextColor = 0x000000
## Text x origin in pixel
_iXText = 25
## Text y origin in pixel
_iYText = 20

# public methods
#####

## The constructor.
# @param self The object pointer
# @param aUserCallback User defined callback method or function. Return
# current focus id when select key it pressed
# @param aItems List of items to display
# @param aScrollY By default True. Display scrollbar or not
# @param aMenuDimmer Send signal to the menu dimmer callback if set.
def __init__( self, aUserCallback, aItems, aMenuDimmer=lambda:None,
              aScrollY=True ):
    ## User defined callback method
    self._mUserCallback = aUserCallback
    ## List of unformatted items to be displayed
    self._iRawItems = aItems
    ## List of formatted items to be displayed
    self._iItems = None
    ## If True, the scrollbar will be displayed
    self._iScrollY = aScrollY
    ## Send signal to the menu dimmer callback if set
    # added on 2007-03-22: missing feature
    self._mMenuDimmer = aMenuDimmer
    ## Current item focus
    self._iCurrentFocus = 0
    ## Keyboard handler instance
    self._iKeyboard = Keyboard ( self._eventCallback)
    appuifw.Canvas.__init__( self,
                             self._redrawCallback,
                             self._iKeyboard.handle_event,
                             self._resizeCallback)

    self._formatData( )
    self._iReady = True

## Returns the current focus number.
# @param self The object pointer
def current( self ):
    return self._iCurrentFocus

## Sets new content for the list class object.
# @param self The object pointer
# @param aItems set a new list to the listbox object
def set( self, aItems ):
    self._iItems = aItems
    self._formatData( )
    self._redrawCallback( )

# Shows the list on the application body.
# @param self reference
def show( self ):
    appuifw.app.body = self
    self._redrawCallback()
```

A_100%_Python_implemented_Listbox_base_class

```
# private methods
#####
## Calculate the focus factor (_iFactor) depending on the set
# self._iScrollFactor. It shouldn't need to be overridden.
# @param self The object pointer
def _calculateFocus( self ):
    if self._iCurrentFocus == 0:
        ## Value used to start displaying the items
        self._iScroll = 0
        ## Value used to display the selection rectangle at the right line
        self._iFactor = 0
    # last item with bottom caption when selecting up
    elif self._iCurrentFocus == len( self._iItems ) - 1 and \
        self._iScroll == 0:
        if len( self._iItems ) < self._iMaxItem:
            self._iFactor = ( len( self._iItems ) - 1 ) * \
                self._iScrollFactor
        else:
            self._iScroll = self._iCurrentFocus - ( self._iMaxItem - 1 )
            self._iFactor = ( self._iMaxItem - 1 ) * self._iScrollFactor
    # no scroll made and 0 <= focus < max item
    elif self._iCurrentFocus < self._iMaxItem and self._iScroll == 0:
        self._iScroll = 0
        self._iFactor = self._iCurrentFocus * self._iScrollFactor
    # first scroll down caption down
    elif self._iCurrentFocus == self._iMaxItem and self._iScroll == 0:
        self._iFactor = ( self._iMaxItem - 1 ) * self._iScrollFactor
        self._iScroll += 1
    # increment scroll
    elif self._iScroll != 0 and \
        ( self._iCurrentFocus - self._iMaxItem ) == self._iScroll:
        self._iFactor = ( self._iMaxItem - 1 ) * self._iScrollFactor
        self._iScroll += 1
    elif self._iCurrentFocus >= self._iMaxItem and self._iScroll != 0:
        if ( self._iCurrentFocus - ( self._iMaxItem - 1 ) ) == 0:
            self._iScroll -= 1
            self._iFactor = 0
        else:
            self._iFactor = ( self._iCurrentFocus - self._iScroll ) * \
                self._iScrollFactor
            if self._iFactor < 0:
                self._iFactor = 0
                self._iScroll -= 1
    elif self._iCurrentFocus < self._iMaxItem and self._iScroll != 0:
        if self._iCurrentFocus == ( self._iScroll - 1 ):
            self._iScroll -= 1
            self._iFactor = 0
        elif self._iCurrentFocus == self._iScroll:
            self._iFactor = 0
        else:
            self._iFactor = ( self._iCurrentFocus - self._iScroll ) * \
                self._iScrollFactor

## Key down, scrolls down. Shouldn't need to be overridden.
# @param self The object pointer
def _down( self ):
    self._iCurrentFocus += 1
    if self._iCurrentFocus == len( self._iItems ):
        self._iCurrentFocus = 0
    # send signal to menu dimmer
    if self._mMenuDimmer:
        self._mMenuDimmer( )
```

A_100%_Python_implemented_Listbox_base_class

```
self._redrawCallback( )

## Draw the focus; here a rectangle caption. Overwrite it for your need.
# @param self The object pointer
def _drawFocus( self ):
    self.rectangle( ( 1, 5 + self._iFactor, ( self.size[0] - 4 ), 25 + \
        self._iFactor ),
                    outline=self._iFocusOutlineColor,
                    fill=self._iFocusBackgroundColor )

## Method for redrawing elements on the screen. Overwrite it for your need
# @param self The object pointer
def _drawItems( self ):
    self.clear( self._iBackgroundColor )
    if self._iReady:
        yText = self._iYText
        # vertical left side line
        self.line( ( 20, 0, 20, self.size[1] ), outline=self._iLineColor)
        # horizontal bottom line
        self.line( ( 20, self.size[1]-1, self.size[0], self.size[1]-1 ), \
            outline=self._iLineColor)
        self._drawFocus( )
        for item in self._iItems[self._iScroll : self._iScroll + \
            self._iMaxItem]:
            self.text( ( self._iXText, yText ), item,
                      font = self._iDefaultFont )
            yText += 25

## Method for drawing the scrollbar. Overwrite it for your need.
# @param self The object pointer
def _drawScrollbar( self ):
    if self._iScrollY and len( self._iItems ) > self._iMaxItem:
        height = float ( ( self.size[1] - 1 ) / ( len( self._iItems ) ) )
        y = float( ( self._iCurrentFocus * height ) + 1 )
        self.line( ( self.size[0] - 2, 0, self.size[0] - 2, self.size[1]),
                  outline=self._iScrollbarOutlineColor)
        self.rectangle( ( self.size[0] - 3, y, ( self.size[0] ), y + \
            height),
                       outline=self._iScrollbarCursorOutlineColor,
                       fill=self._iScrollbarCursorFillColor)

## Event callback method ( does not support repeat when long press ).
# Shouldn't need to be overwritten except if you want to add a binding
# method for keyboard.
# @param self The object pointer
# @param aEvent Event code
def _eventCallback( self ):
    if self._iKeyboard.pressed( EScancodeUpArrow ):
        self._up( )
    elif self._iKeyboard.pressed( EScancodeDownArrow ):
        self._down( )
    elif self._iKeyboard.pressed( EScancodeSelect ):
        if self._mUserCallback != None:
            self._mUserCallback( self.current( ) )

## Format data to the appropriate form you want. Might need to be
# overridden.
# @param self The object pointer
def _formatData( self ):
    tempList = []
    for item in self._iRawItems:
        lines = wrap_text_to_array( item, self._iDefaultFont,
```

A_100%_Python_implemented_Listbox_base_class

```

        self.size[0] - self._iXText - 5 )
# here I want to keep only the first line if oversized line
if len(lines) > 1:
    tempList.append( lines[0] + u'...' )
else:
    tempList.append( lines[0] )
self._iItems = tempList

## Redraw callback method. Shouldn't need to be overwritten.
# @param self The object pointer
# @param aRect Attribute value sent by the Canvas object
def _redrawCallback( self, aRect=None ):
    self._calculateFocus( )
    self._drawItems( )
    self._drawScrollbar( )

## Key up, scrolls up. Shouldn't need to be overwritten.
# @param self The object pointer
def _up( self ):
    self._iCurrentFocus -= 1
    # it means we want the last item of the list
    if self._iCurrentFocus == -1:
        self._iCurrentFocus = len( self._iItems ) - 1
    # send signal to menu dimmer
    if self._mMenuDimmer:
        self._mMenuDimmer( )
    self._redrawCallback( )

## Recalculate how many items can be displayed on the screen.
# @param self The object pointer.
# @param aClientRect two-element tuple that contains the new clientRect
# width and height sent by the canvas object.
def _resizeCallback( self, aClientRect=None ):
    try:
        oldValue = self._iMaxItem
        self._formatData()
        self._iMaxItem = int(self.size[1]/self._iScrollFactor)
        # if the screen is switched from portrait to landscape and, the item
        # the current item may become focused but invisible. A scroll
        # adjustment is needed
        self._iScroll += oldValue - self._iMaxItem
        if self._iScroll < 0 :
            self._iScroll = 0
    except:
        pass
```

Usage

```
SCRIPT_LOCK = e32.Ao_lock( )
def mainCallback( aId ):
    print aId

def menuDimmerCallback():
    if lb :
        appuifw.app.menu = [(u'item %s menu'%str(lb.current()+1), lambda:None)]

def __exit__( ):

```

A_100%_Python_implemented_Listbox_base_class

```
SCRIPT_LOCK.signal( )

appuifw.app.exit_key_handler = __exit__
appuifw.app.title= u'ListBoxBaseClass'

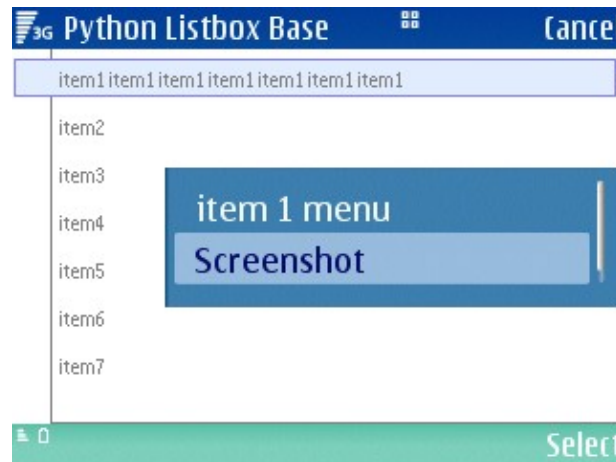
l1list = [u'item1 item1 item1 item1 item1 item1 item1', u'item2', u'item3',
          u'item4', u'item5', u'item6', u'item7', u'item8', u'item9',
          u'item10',u'item11', u'item12', u'item13',u'item14', u'item15',
          u'item16', u'item17' ]

lb = ListBoxBaseClass( mainCallback, l1list, menuDimmerCallback )
lb.show( )
menuDimmerCallback( )
SCRIPT_LOCK.wait( )
```

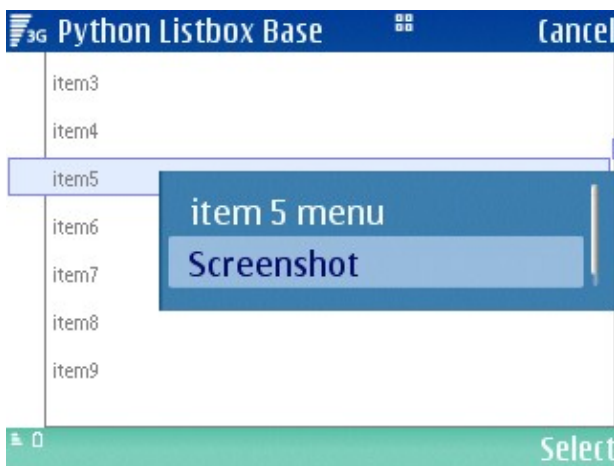
Screenshots



Python ListBox Portrait: the first line is too long to be displayed completely; marquees are added (**small issue with wrap_text_to_array**: the line should be displayed completely...)



Python Listbox Landscape: the first line is displayed completely



Python Listbox Menu Dimmer: in this screenshot pay attention to "item 5 menu" updated during the navigation

Remarks

You can easily create new listbox style with this base class by inheriting this base class and overwriting a few methods. I could post some examples later on.

You can see other styles I made on [my thesis client](#), maybe it could give you some ideas.