



ID	...	Creation date	18 May 2007
Platform	S60	Tested on devices	Emulator
Category	Python	Subcategory	Camera

Keywords (APIs, classes, methods, functions): Camera, viewfinder

Here is a simple camera application that uses the camera view finder.

There are no zooming or special option. It is free to you to add them depending on your needs.

Code Snippet

```
import appuifw, e32, camera, time
from key_codes import EScancodeSelect, EScancode5, EScancode1, EScancode3

if e32.in_emulator( ):
    IMAGEPATH = 'D:\\Images\\%s.png'
else:
    IMAGEPATH = 'E:\\Images\\%s.png'

IMAGERESOLUTION = (640, 480)

# I added this class in the source if you haven't already added it
# to the libraries.
class Keyboard( object ):
    def __init__( self, onevent=lambda:None ):
        self._keyboard_state = {}
        self._downs = {}
        self._onevent = onevent

    def handle_event( self, event ):
        if event['type'] == appuifw.EEventKeyDown:
            code = event['scancode']
            if not self.is_down( code ):
                self._downs[code] = self._downs.get( code, 0 ) + 1
                self._keyboard_state[code] = 1
        elif event['type'] == appuifw.EEventKeyUp:
            self._keyboard_state[event['scancode']] = 0
        self._onevent( )

    def is_down( self, scancode ):
        return self._keyboard_state.get( scancode, 0 )
```

A_simple_camera_application_using_the_view_finder

```
def pressed( self, scancode ):
    if self._downs.get( scancode, 0 ):
        self._downs[scancode] -= 1
        return True
    return False

# Here is the camera base class I use in few applications.
# This class could be added to your libraries so that you only have to import it.
class Camera( object, appuifw.Canvas ):
    _iImageWidth = 160
    _iImageHeight = 120
    _iSartColor = 0x000000
    _iCapturedColor = 0x0000ff
    _iSavingColor = 0x00ff00
    _iStopedColor = 0xff0000

    def __init__( self, aHandleEvent ):
        self._iReady = False
        self._iTempImage = None
        self._iSaving = False
        self._iSaved = False
        self._iViewerActive = False
        appuifw.Canvas.__init__( self, self._redrawCallback, aHandleEvent )
        appuifw.app.body = self
        self._iReady = True

    def _redrawCallback( self, aRect=None ):
        self.clear( )
        x = ( self.size[0] - self._iImageWidth ) / 2
        y = ( self.size[1] - self._iImageHeight ) / 2
        if self._iTempImage:
            self.blit( self._iTempImage, target=( x+1, y ) )

        if not self._iTempImage:
            color = self._iSartColor
        elif self._iSaving:
            color = self._iSavingColor
        elif self._iSaved:
            color = self._iCapturedColor
        else:
            color = self._iStopedColor

        if self._iReady:
            self.line( ( x-2, y-3,x+5, y-3 ), outline=color )
            self.line( ( x-2, y-3,x-2, y+4 ), outline=color )
            self.line( ( self.size[0]-x+2, y-3,self.size[0]-x-4, y-3 ),
                outline=color )
            self.line( ( self.size[0]-x+3, y-3,self.size[0]-x+3, y+4 ),
                outline=color )
            self.line( ( x-2, self.size[1]-y+2,x+5, self.size[1]-y+2 ),
                outline=color )
            self.line( ( x-2, self.size[1]-y-4,x-2, self.size[1]-y+2 ),
                outline=color )
            self.line( ( self.size[0]-x+2, self.size[1]-y+2,self.size[0]-x-4,
                self.size[1]-y+2 ), outline=color )
            self.line( ( self.size[0]-x+3, self.size[1]-y-4, self.size[0]-x+3,
                self.size[1]-y+3 ), outline=color )

    def _viewFinderRedrawCallback( self, aImage ):
        self.blit( aImage, target=( (self.size[0] - aImage.size[0])/2 +1,
            (self.size[1] - aImage.size[1])/2 ) )
```

A_simple_camera_application_using_the_view_finder

```
self._iTempImage = aImage

def setActive( self ):
    appuifw.app.body = self
    self._redrawCallback( )

def startViewFinder( self ):
    self._iTempImage = None
    self.showCapture( )
    camera.start_finder( self._viewFinderRedrawCallback,
                        size = ( self._iImageWidth, self._iImageHeight ) )
    self._iSaved = False
    self._iViewerActive = True

def stopViewFinder( self ):
    if self._iViewerActive:
        camera.stop_finder( )
        self.showCapture( )
        self._iViewerActive = False

def showCapture(self):
    self._redrawCallback( )

def capture( self ):
    try:
        self._iSaving = True
        self.stopViewFinder( )
        e32.ao_yield( )
        ctime = time.localtime()
        # Filename format: YYYYMMDDHHMMSS
        picName = "%4i%02i%02i%02i%02i%02i" %+ \
                ((ctime[0]), (ctime[1]), (ctime[2]),\
                (ctime[3]), (ctime[4]), (ctime[5]))
        img = camera.take_photo( size=IMAGERESOLUTION )
        img.save( IMAGEPATH%picName )

        # if thumbnail size is good enough for your application just save
        # the last image:
        # self._iTempImage.save( IMAGEPATH )
        self._iSaved = True
    except Exception, error:
        print error
        self._iSaved = False
    self._iSaving = False
    self.showCapture( )

def isSaved( self ):
    return self._iSaved

def isViewerActive( self ):
    return self._iViewerActive

def __del__( self ):
    if self._iViewerActive:
        camera.stop_finder( )

# This class, where all the handler are defined, uses both of the classes above.
class CameraView( object ):
    def __init__( self ):
        self._iKeyboard = Keyboard( self._keyObserver )
        self._iCamera = Camera( self._iKeyboard.handle_event )
```

A_simple_camera_application_using_the_view_finder

```
def setActive( self ):
    self._iCamera.setActive( )
    self._menuDimmer( )

def _keyObserver( self ):
    # select and 5 keys for capture
    if self._iKeyboard.pressed( ESCancodeSelect ) or \
        self._iKeyboard.pressed( ESCancode5 ):
        if self._iCamera.isViewerActive():
            self._handleCommands( "CAPTURE" )

    # key 1 to start the viewer
    elif self._iKeyboard.pressed( ESCancode1 ):
        if not self._iCamera.isViewerActive( ):
            self._handleCommands( "NEW" )

    # 3 key for stoping the viewer
    elif self._iKeyboard.pressed( ESCancode3 ):
        if self._iCamera.isViewerActive( ) and \
            not self._iCamera.isSaved( ) :
            self._handleCommands( "STOP" )

def _handleCommands( self, aCommand ):
    if aCommand == "NEW":
        self._iCamera.startViewFinder( )
        self._menuDimmer( )

    elif aCommand == "STOP":
        self._iCamera.stopViewFinder( )
        self._menuDimmer( )

    elif aCommand == "CAPTURE":
        appuifw.note(u'Please wait', 'info', 1)
        self._iCamera.capture( )
        self._menuDimmer( )
        if self._iCamera.isSaved( ):
            appuifw.note(u'Saved', 'info')
        else:
            appuifw.note(u'Failed', 'error')

    elif aCommand == "EXIT":
        __exit__()

def _menuDimmer( self ):
    menu = []
    if self._iCamera.isSaved( ):
        menu.append( ( u'New...', lambda:self._handleCommands( "NEW" ) ) )

    elif self._iCamera.isViewerActive( ):
        menu.append( ( u'Capture', lambda:self._handleCommands( "CAPTURE" ) ) )
        menu.append( ( u'Stop', lambda:self._handleCommands( "STOP" ) ) )

    else:
        menu.append( ( u'New...', lambda:self._handleCommands( "NEW" ) ) )
        menu.append( ( u'Exit', lambda:self._handleCommands( "EXIT" ) ) )

    appuifw.app.menu = menu

def __del__( self ):
    # clean exit
```

A_simple_camera_application_using_the_view_finder

```
self._iCamera.__del__( )

def __exit__():
    camView.__del__( )
    APP_LOCK.signal()

if __name__ == '__main__':
    APP_LOCK = e32.Ao_lock( )
    appuifw.app.title = u"Cam using View Finder "
    appuifw.app.exit_key_handler = __exit__
    camView = CameraView( )
    camView.setActive( )
    APP_LOCK.wait( )
```