



ID		Creation date	April 27, 2009
Platform	S60 3rd Edition	Tested on devices	Nokia N82
Category	Python	Subcategory	graphics

Keywords (APIs, classes, methods, functions): Canvas, graphics, Image

Introduction

Although PyS60 is not able to deal with images with alpha information (32-bit or RGBA), there is one way to show transparent GIF and PNG images using the mask parameter on the Image class.

Instructions

Suppose you have drawn a very cool clock with Inkscape having a transparent background and that you have exported it to PNG and would like to overlay this PNG on a gradient (green to yellow, for example) background.

The first natural move would be the following code (supposing you have the image files in E:\python\lib\ directory):

```
self.gradient = Image.open(u'E:\\python\\lib\\gradient.png')
self.clock_img = Image.open(u'E:\\python\\lib\\clock.png')
self.canvas.clear()
self.canvas.blit(self.gradient)
self.canvas.blit(self.clock_img)
```

But the result isn't what you would like to see:

How_to_display_transparent_PNG_on_canvas_with_masks



To solve this problem there's a mask parameter for the `blit` method to help graphics to know where the transparent pixels are on the image.

The mask can be a 1-bit image (black and white) or an 8-bit image (grayscale). The `blit` method **doesn't** accept other image modes on mask image (12, 16 or 24 bits).

There is an example of an automask function, that automatically creates an image mask from another image, using the first pixel as transparent color on the Forum Nokia Wiki:

[How to display transparent image on canvas.](#)

```
def automask(im):
    width, height = im.size          # get image size
    mask = Image.new(im.size, '1')  # black and white
    tran = im.getpixel((0,0))[0]    # transparent top-left
    for y in range(height):
        line = im.getpixel([(x, y) for x in range(width)])
        for x in range(width):
            if line[x] == tran:
                mask.point((x,y), 0) # mask on the point
    return mask
```

This function creates a mask image to be used with the `blit` method as shown below:

```
self.gradient = Image.open(u'E:\\python\\lib\\gradient.png')
self.clock_img = Image.open(u'E:\\python\\lib\\clock.png')
self.mask = automask(self.clock_img)
self.canvas.clear()
self.canvas.blit(self.gradient)
self.canvas.blit(self.clock_img, mask=self.mask)
```

The result of applying this function is a lot better than the black background replacing transparent pixels, but it doesn't support anti-alias and partially transparent pixels.

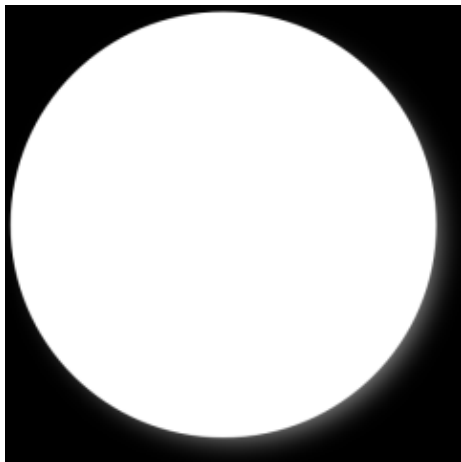
The result is shown below (note the absence of anti-alias on clock boundaries):

How_to_display_transparent_PNG_on_canvas_with_masks



The solution for the anti-alias and partially transparent pixels is a mask with 8-bit depth, where black is 100% transparent and white is 0% transparent. The grayscale gives the percentage of opacity of each pixel.

Using your favorite image processing program (for example [Gimp](#)), you can extract only the alpha channel and export it as grayscale. You'll have to invert the image to have white instead of black and vice-versa, like the image below.



With this image mask, you can use the code below to apply the mask and use the PNG transparency:

```
self.gradient = Image.open(u'E:\\python\\lib\\gradient.png')
self.clock_img = Image.open(u'E:\\python\\lib\\clock.png')
mask = Image.open(u'E:\\python\\lib\\clock-mask.png')
self.mask = Image.new(mask.size, 'L') #grayscale 8-bit
self.mask.blit(mask) #convert the png to 8-bit grayscale
self.canvas.clear()
self.canvas.blit(self.gradient)
self.canvas.blit(self.clock_img, mask=self.mask)
```

The result is shown below:



You can notice that the image is perfectly merged with gradient background with a little shadow on the right side of clock.

Portuguese Version

[Como exibir imagens PNG no canvas](#)