Phatch Contributors Guide

Table of Contents

Phatch = Photo & Batch!

Phatch is a simple to use cross-platform GUI Photo Batch Processor which handles all popular image formats and can duplicate (sub)folder hierarchies. Phatch can batch resize, rotate, rename, … and more in minutes instead of hours or days if you do it manually. Phatch will also support a console version in the future to batch photos on webservers.

Introduction

Welcome

Thanks for contributing to Phatch!

- You might have exciting ideas how to extend the functionality with new features. The good news is that Phatch was designed from the ground up to make the development of actions as easy as possible.
- Or you might want to write some documentation or tutorial.

Required Skills

Developping Action plugins

To develop actions for Phatch you only need to know Python [http://www.python.org] and PIL [http://www.pythonware.com/products/pil] (Python Image Library). Although Phatch uses wxPython [http://www.wxpython.org] for its cross-platform GUI, you don't need at all wxPython to write actions. Phatch will do all the GUI work automatically for you behind the scenes, so you can fully concentrate on the image manipulation. Also don't worry about internationalisation, Phatch will take care of that too.

Writing documentation

Can you work with notepad, gedit or kate? Fine, that is all you need. The documentation is generated from plain text files, which follow the asciidoc [http://www.methods.co.nz/asciidoc] format. They can be translated afterwards in a lot of different formats, such as html and pdf. You don't need to install asciidoc to write documentation. You can send the plain text files to the Phatch development team and we will take care of it. However if you are curious, you will find instructions in this manual on how to install asciidoc on Ubuntu Feisty to generate pdfs. This might be usefull as well to install asciidoc on other operating systems.

Translating

- Translating the application Phatch is done through launchpad [https://translations.launchpad.net/phatch/trunk/+pots/phatch].
- Translating of documentation is done with plain text asciidoc files. Be sure to keep a copy of the source file on which you base your translation. This will allow you to quickly see what has changed with a diff viewer when new documentation is available for translation.

Developping Action Plugins

Getting Started: Invert

Introduction

The best way to start is first to develop an PIL function which handles the manipulation. Afterwards you create input fields in order to pass the required parameters to the PIL function. *Invert* is taken as a first case study as it doesn't require any parameters except for the image itself. Let's look at the source code:

Example 1. Invert Action Source Code

```
from core import models
from core.translation import _t

def invert(image):
    """PIL function"""
    return ImageChops.invert(image)                         ❶

class Action(models.Action):
    label       = _t('Invert')
    author      = 'Stani'
    email       = 'spe.stani.be@gmail.com'
    version     = '0.1'
    tags        = ['Colours']
    __doc__     = _t('Invert the colors of an image')       ❷

    def import_modules(self):
        global ImageChops
        import ImageChops                                   ❸

    def apply(self,photo,setting,cache):
        return self.apply_to_current_layer_image(photo)     ❹

    description = _t("""Invert the colors of the image.""") ❺

    #icon        = 'icon data'                              ❻
```

❶❸ Importing Modules
❷ Defining The Action
❹ Applying The Action
❺ Describing The Action
❻ Adding An Icon

Importing Modules

For every action you need to add the first two lines which import the basic functionality for writing action plugins. The module models provide the architecture for the plugin: the `class Action` and the input fields. (As invert doesn't require any input, there are no fields here.) Every other module you need to import with the method `import_modules`, in which you declare them as global. For example to invert an image with PIL you need the `ImageChops` module.

Why do modules have to be imported in a seperate method? The reason is that Phatch at startup imports all actions to extract the information it needs about the actions. If the imports would be declared normally, the startup would be delayed by maybe unneeded modules.

Defining The Action

You need to create a new `class Action` which is derived from `models.Action`. You need to define the action with the `label`, `author`, `email`, `version`, `tags` and `__doc__` properties. `label` and `__doc__` will appear translated in the *Add Action* dialog box. That is why you need to mark them with the `_t` function. At the moment `tags` and `description`, which can contain a longer description than the `__doc__` one-liner, are not exposed yet.

Applying The Action

Internally Phatch works with *photos*. *Photos* consist of multiple *layers*. Every *layer* contains an *image*, but also has its own offset postion. Phatch doesn't expose this functionality yet, but will later support full layered photos, just like in Gimp. The hierarchy to remember is: Photo>Layer>Image. Luckily you don't have to worry about this in the beginning as Phatch provides you an easy method to apply a PIL function the current layer image: `apply_to_current_layer_image`.

Describing The Action

In the description property you can describe the action more elaborately. Use triple quotes for multi-line text.

Adding An Icon

As in the example no specific icon was added, Phatch will use a default one. In the folder `phatch/pyWx/lib` there is an utility `img2py.py`. With the following command you can convert any image (eg.png) to a python file:

```
$ python img2py.py fileName icon.py
```

This will generate an `icon.py` file, in which you will find the following code:

```
def getData():
    return zlib.decompress('icon data')
```

You can add now the *icon data* to your action source file to define the icon:

```
class Action(models.Action):
    description = """Invert the colors of the image."""
    icon        = 'icon data'
```

Advanced: Drop Shadow

Introduction

The following code was taken from the Python Cookbook: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/474116 It demonstrates how easy it is to integrate existing PIL code into Phatch.

Source Code

Example 2. Shadow Action Source Code

```
from core import models
from core.translation import _t

def dropShadow(image, x_offset=5, y_offset=5, back_colour=(255,255,255,0),
                shadow_colour=0x444444, border=8, iterations=3,
                force_back=False, cache={}):
    """
    Add a gaussian blur drop shadow to an image.

    image           - The image to overlay on top of the shadow.
    offset          - Offset of the shadow from the image as an (x,y) tuple.
                        Can be positive or negative.
    back_colour     - Background colour behind the image.
    shadow_colour   - Shadow colour (darkness).
    border          - Width of the border around the image.  This must be wide
                        enough to account for the blurring of the shadow.
    iterations      - Number of times to apply the filter.  More iterations
                        produce a more blurred shadow, but increase processing
                        time.
    """
    #get info
    size        = image.size
    mode        = image.mode

    back        = None

    #assert image is RGBA
    if mode != 'RGBA':
        #create cache id
        id  = ''.join([str(x) for x in ['shadow_',size,x_offset,y_offset,
                    border, iterations,back_colour,shadow_colour]])

        #look up in cache
        if cache.has_key(id):
            #retrieve from cache
            back, back_size = cache[id]

    if back is None:
        #size of backdrop
        back_size       = (size[0] + abs(x_offset) + 2*border,
                        size[1] + abs(y_offset) + 2*border)

        #create shadow mask
        if mode == 'RGBA':
            image_mask  = image.split()[-1]
            shadow      = Image.new('L',back_size,0)
        else:
            image_mask  = Image.new(mode,size,shadow_colour)
            shadow      = Image.new(mode,back_size,back_colour)

        shadow_left     = border + max(x_offset, 0)
        shadow_top      = border + max(y_offset, 0)
        shadow.paste(image_mask, (shadow_left, shadow_top,
                                shadow_left + size[0], shadow_top + size[1]))
        del image_mask #free up memory

        #blur shadow mask

        #Apply the filter to blur the edges of the shadow.  Since a small
        #kernel is used, the filter must be applied repeatedly to get a decent
        #blur.
        n   = 0
        while n < iterations:
            shadow = shadow.filter(ImageFilter.BLUR)
            n += 1

        #create back
        if mode == 'RGBA':
            back  = Image.new('RGBA',back_size,shadow_colour)
            back.putalpha(shadow)
            del shadow #free up memory
        else:
            back        = shadow
            cache[id]   = back, back_size

    #Paste the input image onto the shadow backdrop
    image_left  = border - min(x_offset, 0)
    image_top   = border - min(y_offset, 0)
    if mode == 'RGBA':
        back.paste(image, (image_left, image_top),image)
        if force_back:
            mask    = back.split()[-1]
            back.paste(Image.new('RGB',back.size,back_colour),(0,0),
                ImageChops.invert(mask))
            back.putalpha(mask)
    else:
        back.paste(image, (image_left, image_top))

    return back

class Action(models.Action):
    """Drops shadow"""

    label       = _t('Shadow')
    author      = 'Stani'
    email       = 'spe.stani.be@gmail.com'
    version     = '0.1'
    tags        = [_t('filter')]
    __doc__     = _t('Drops a blurred shadow under a photo')

    def __init__(self):
        fields = models.Fields()

        fields[_t('Horizontal Offset')]        = models.PixelField('5')
        fields[_t('Vertical Offset')]          = models.PixelField('5')
        fields[_t('Border')]                   = models.PixelField('8')
        fields[_t('Shadow Blur')]              = models.SliderField(3,1,20)
        fields[_t('Background Colour')]        = models.ColourField('#FFFFFF')
        fields[_t('Shadow Colour')]            = models.ColourField('#444444')
        fields[_t('Force Background Colour')]  = models.BooleanField(True)

        super(Action,self).__init__(fields)

    def apply(self,photo,setting,cache):
        #get info
        info        = photo.get_info()
        #size
        x0, y0      = info['new_size']
        dpi         = info['new_dpi']
        #dpi
        parameters  = {
            'x_offset'      : self.get_field_size('Horizontal Offset',
                                info,x0,dpi),
            'y_offset'      : self.get_field_size('Vertical Offset',
                                info,x0,dpi),
            'border'        : self.get_field_size('Border', info,x0,dpi),
            'iterations'    : self.get_field_value('Shadow Blur', info,),
            'force_back'    : self.get_field_value('Force Background Colour',
                                info),
            'back_colour'   : self.get_field_value('Background Colour', info),
            'shadow_colour' : self.get_field_value('Shadow Colour', info),
            'cache'         : cache,
        }

        #manipulate layer
        return self.apply_to_current_layer_image(photo,dropShadow,**parameters)

    def import_modules(self):
        #lazily import
        global Image, ImageChops, ImageFilter
        import Image, ImageChops, ImageFilter
```

Further Study

Probably looking at the source code of the actions, will teach you the most. You find all the actions in the folder `phatch/actions`

If you installed Phatch on Ubuntu, probably the actions are in the folder: `/usr/lib/python2.5/site-packages/phatch/actions`

Writing Documentation

Introduction

The mechanism used for documentation is asciidoc. You can find more information here: http://www.methods.co.nz/asciidoc/

You can send plain text files which will be translated into pdf an html. The html will be published on the Phatch website, which might include some advertisements. The pdf version will be a present to anyone who donates. If you contribute documentation, you agree with these conditions

Installing AsciiDoc

Ubuntu Feisty

You need to install these packages:

```
$ sudo apt-get install asciidoc docbook-xml docbook-xsl source-highlight
```

To generate pdfs you will need FOP [http://xmlgraphics.apache.org/fop/], which unfortunately is not available as a package. Also we need an older version, as the current one is not compatible with the Ubuntu packages. Therefore download fop-0.20.5-bin.zip [http://archive.apache.org/dist/xmlgraphics/fop/binaries/fop-0.20.5-bin.zip].

Unzip this folder to where you want, for example `/opt/fop`. Make a symbolic link so fop.sh is recognized as a command:

```
$ sudo ln -s /opt/fop/fop.sh /usr/local/bin/fop.sh
```

Add this line to fop.sh so that it can find your java version:

```
JAVA_HOME=/usr/lib/jvm/java-6-sun
```

Copy the filter (`./examples/source-highlight-filter/source-highlight-filter.conf`) to one of the standard AsciiDoc filter locations — typically `/etc/asciidoc/filters/` or `~/.asciidoc/filters/`.