

Plugins für Inkscape

In diesem Tutorial wird das Schreiben und Installieren von Plugins für Inkscape behandelt.

Liste von nützlichen Plugins

- Polygon <https://github.com/ThoreMehr/inkscape-polygon>
- Mehr_BoxMaker https://github.com/ThoreMehr/Mehr_BoxMaker
- Gear-dev <https://github.com/jnweiger/inkscape-gears-dev>

Installieren

Die .py und die .inx Datei muss direkt in den Extentions-Ordner kopiert werden. Welcher Ordner das ist, hängt vom Betriebssystem ab. Es sollten die Dateien direkt sein, nicht ein Ordner, der diese Dateien enthält.

Windows

Die Erweiterungen liegen im Ordner `\Inkscape\share\extensions`. In der Standardinstallation also in `C:\Program Files\Inkscape\share\extensions`

Ubuntu und viele andere Linux Varianten

Erweiterungen nach `~/.config/inkscape/extensions/` kopieren und Inkscape neu starten.

Mac OS

Erweiterungen nach `/Applications/inkscape.app/Contents/Resources/share/inkscape/extensions`

Schreiben von Plugins

Die Inkscape- Plugins bestehen aus zwei Teilen, einer GUI und dem eigentlichen Code. Dieses Tutorial beschränkt sich auf die Verwendung der Standart-GUI und Code in Python. Ein Plugin besteht aus zwei Dateien, einer .inx und einer .py, da in Python programmiert wird. Außerdem kann ein plugin zu einer von vier Klassen gehören, import, export, print und effect. Davon ist effect am interessantesten, da es Geometrie erzeugen und verändern kann. Das heißt, damit kann man Zeichnen. Dazu sollten SVG-Pfade verstanden werden. Hier ist ein [tutorial](#) dafür.

GUI

Die GUI wird in Form einer XML-Datei mit der Endung .inx definiert. Der aufbau der Datei ist wie folgt:

```
<?xml version="1.0" encoding="UTF-8"?>
<inkscape-extension
xmlns="http://www.inkscape.org/namespace/inkscape/extension">
  <_name>$NAME</_name>
  <id>$ID</id>
```

\$NAME sollte der Name des Plugins sein. \$ID ist die ID des Plugins und muss einzigartig sein. Ich verwende momentan de.mehr.flk.\$name.

Danach folgt dann eine Anzahl Parameter. Wie diese funktionieren kann [hier](#) nachgeschlagen werden. Die ganzen Anführungszeichen sind ein Muss, wenn diese nicht benutzt werden kann die GUI nicht verarbeitet werden und das Plugin erscheint nicht im Menu. Wenn viele Parameter verwendet werden, sind tabs durch den „notebook“ Parameter zwingend, da sonst, vorallem bei Bildschirmen mit niedriger Auflösung, die Anwenden Taste außerhalb des Bildschirms liegt.

Auf die Parameter folgt dann noch die Informationen zum Aufruff des Python Programs:

```
<effect>
<object-type>all</object-type>
  <effects-menu>
    <submenu _name="$SUBMENUENAME"/>
  </effects-menu>
</effect>
```

Dieser Block ist spezifisch für die effect Klasse.

```
<script>
  <command reldir="extensions"
interpreter="python">$PLUGINNAME.py</command>
</script>
</inkscape-extension>
```

In \$SUBMENUENAME kommt der Name des Submenues rein, ich verwende Laser Tools. In

\$PLUGINNAME sollte der Name der .py rein, normalerweise der Name der Erweiterung.

Es ist nicht möglich aus dem Code Einfluss auf die GUI zu nehmen, die GUI ist nur dazu da, eine Oberfläche auf ein Comandozeilenwerkzeug zu setzten.

Python Code

Der Code beginnt mit dem Import der benötigten Bibliothken. Ich empfehle inkex und simplestyle zu importieren und je nach bedarf weitere Bibliotheken.

Danach folgen Funktionen. Einige nützliche Funktionen sind

```
def textElement(text, positon, self):
    t = inkex.etree.Element(inkex.addNS('text', 'svg'))
    t.text=text
```

```
t.set('x',str(self.unittouu(str(positon[0])+positon[2])))
t.set('y',str(self.unittouu(str(positon[1])+positon[2])))
parent.append(t)
return t
```

```
def drawS(XYstring,color):          # Draw lines from a list
    name='part'
    style = { 'stroke': color, 'fill': 'none' }
    drw =
{'style':simplestyle.formatStyle(style),inkex.addNS('label','inkscape'):name
,'d':XYstring}
    inkex.etree.SubElement(parent, inkex.addNS('path','svg'), drw )
    return
```

```
def groupdraw(XYstrings,colors)  :
    if len(XYstrings)==1:
        drawS(XYstrings[0],colors[0])
        return
    grp_name = 'Group'
    grp_attribs = {inkex.addNS('label','inkscape'):grp_name}
    grp = inkex.etree.SubElement(parent, 'g', grp_attribs)#the group to put
everything in
    name='part'
    for i in range(len(XYstrings)):
        style = { 'stroke': colors[i%len(colors)], 'fill': 'none' }
        drw =
{'style':simplestyle.formatStyle(style),inkex.addNS('label','inkscape'):name
+str(i),'d':XYstrings[i]}
        inkex.etree.SubElement(grp, inkex.addNS('path','svg'), drw )
    return
```

In der ersten wird ein Textelement mit dem Text text an der Position (x,y,unit) erzeugt. In self wird eine Instance von inkex.Effect übergeben. diese ist nach der Initialisierung in self gespeichert. Ich habe noch nicht herausgefunden, wie ich ohne diese Instance unittouu aufrufe. drawS zeichnet einen SVG-Pfad in der übergebenen Fabe. Diese wird in HEX im Format '#RRGGBB' übergeben. groupdraw macht das gleiche für ein Array an SVG-Pfaden. Wenn mehr Strings als Faben verwendet übergeben wird werden Faben wiederholt. Darauf hin kann man mit der Definition der Klasse beginnen.

```
class $CLASSNAME(inkex.Effect):
def __init__(self):
    # Call the base class constructor.
    inkex.Effect.__init__(self)
```

Dieser Block leitet die Definition der Klasse vor. In \$Classname kommt der Name der Klasse, ich verwende da den Namen des Pugins. Danach werden die Optionen nach folgeneden Schema übernommen:

```
self.OptionParser.add_option('--
$OPTIONNAME',action='store',type='$OPTIONTYPE',
dest='$DESTINATIONVAR',default='$DEFAULTVAL',help='$HELPTXT')
```

In \$OPTIONNAME kommt der Name der Option rein, genau so geschrieben, wie in der .inx. In \$OPTIONSTYPE kommt entweder string, float, int oder inkbool rein, je nach dem, was ihr verwendet. Wenn ihr ein enum oder optionsgroup Parameter als string speichert, müssen vergleiche mit strings gemacht werden, daher z.B. == '0' und nicht == 0. Danach wird mit

```
def effect(self):  
    global parent, $OTHERVARS
```

der Effect definiert. Hier kommt der eigentliche Code rein. Auf Variablen im global bereich kann aus allen Funktionen in der .py zugegriffen werden. Auf die zuvor übernommenen Parameter kann mit

```
self.options.$DESTINATIONVAR
```

zugegriffen werden. Danach erstellen wir eine neue Ebene auf die wir Zeichnen wollen. Dies tun wir mit

```
svg = self.document.getroot()  
    # Get the attributes:  
widthDoc = self.unittouu(svg.get('width'))  
heightDoc = self.unittouu(svg.get('height'))  
    # Create a new layer.  
layer = inkex.etree.SubElement(svg, 'g')  
layer.set(inkex.addNS('label', 'inkscape'), 'newlayer')  
layer.set(inkex.addNS('groupmode', 'inkscape'), 'layer')  
parent=self.current_layer
```

Damit wird ein neuer Layer mit Namen parent erschaffen, auf den gezeichnet wird. Dann können die Pfade als String geschrieben werden und mit den Drawfunktionen gezeichnet werden. Den Abschluss des Programms bildet

```
effect = $CLASSNAME()  
effect.affect()
```

womit Inkscape das Programm aufruft.

Nützliche Funktionen

```
self.unittouu(str(value)+unit)
```

gibt Strecken in einer Einheit in der von Inkscape genutzte universale Einheit, Pixel, zurück.