Inkscape and Python

Alex Valavanis

Institute of Microwaves and Photonics,
University of Leeds

July 11, 2014

Overview

Inkscape: an intro to the project

How I got involved

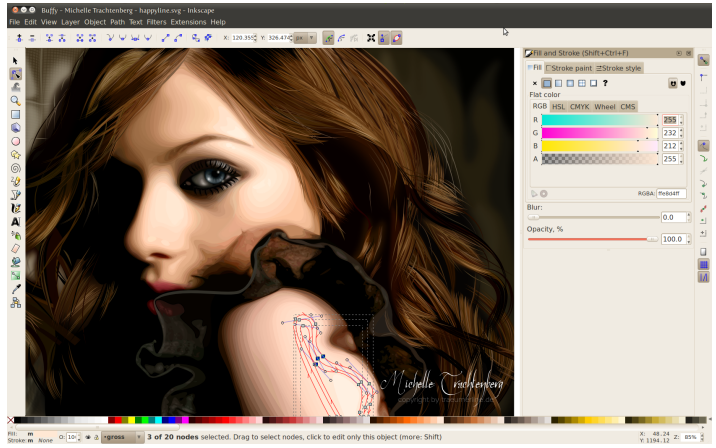
Inkscape development

My involvement

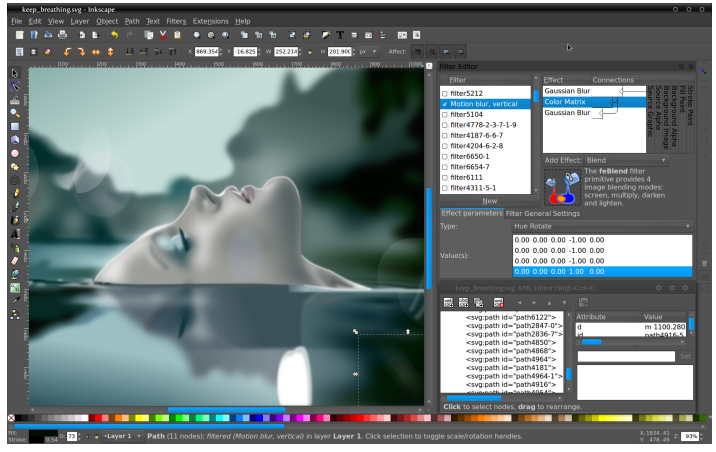
Extension subsystem

What is Inkscape?

- Scalable Vector Graphics (SVG) editor
- Forked from Sodipodi in 2003. . . which was forked from Gill in 1999
- Used by graphic designers, artists, scientists, engineers. . .

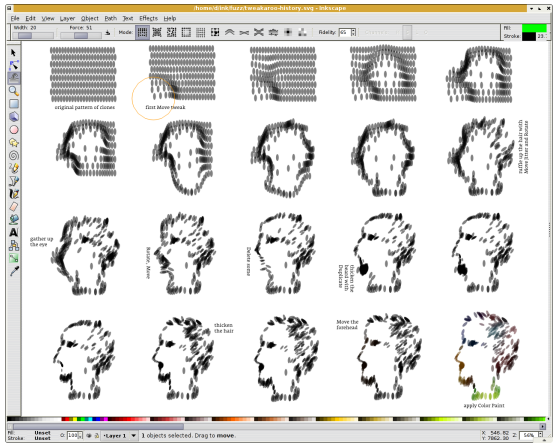


Multi-path editing.

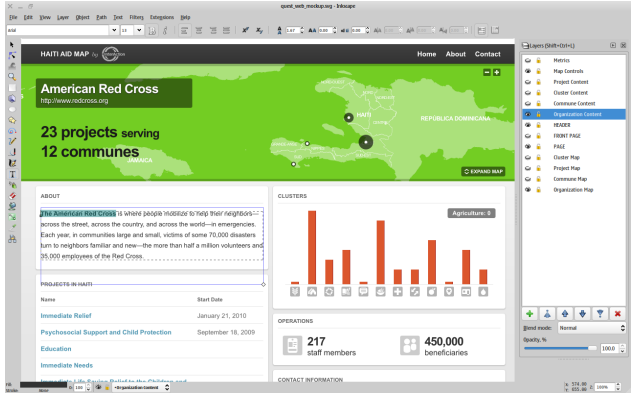


Gaussian blur.

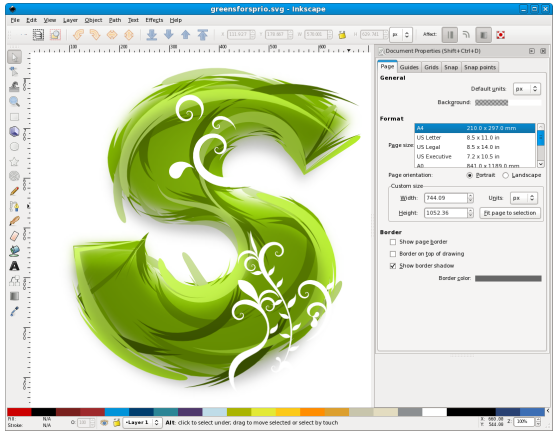
A peek at a few more features...



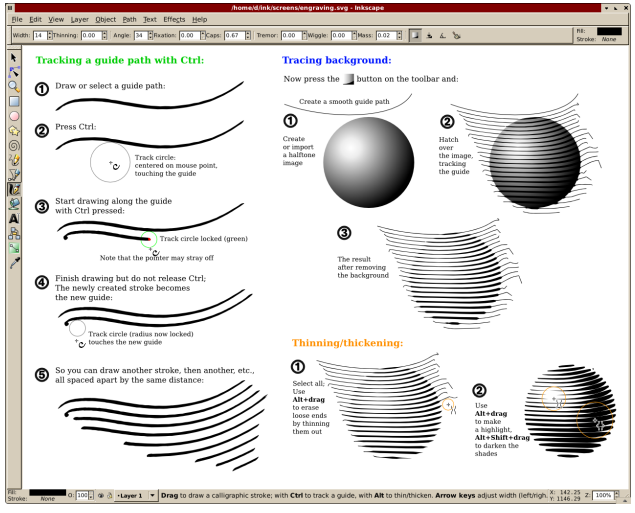
Tweak tool



Text editing



Spiro typography

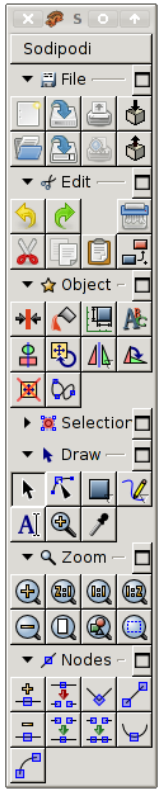


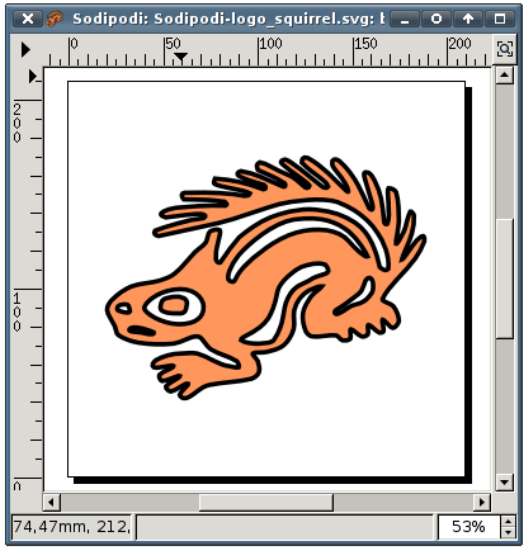
Caligraphy and engraving

Why the fork from Sodipodi?

Inkscape aims to do a few things fundamentally differently from its predecessor:

- Full SVG standards compliance
- Open developer access
- Contribute to 3rd-party libraries rather than DIY solutions
- Simpler GUI
- C++/GtkMM



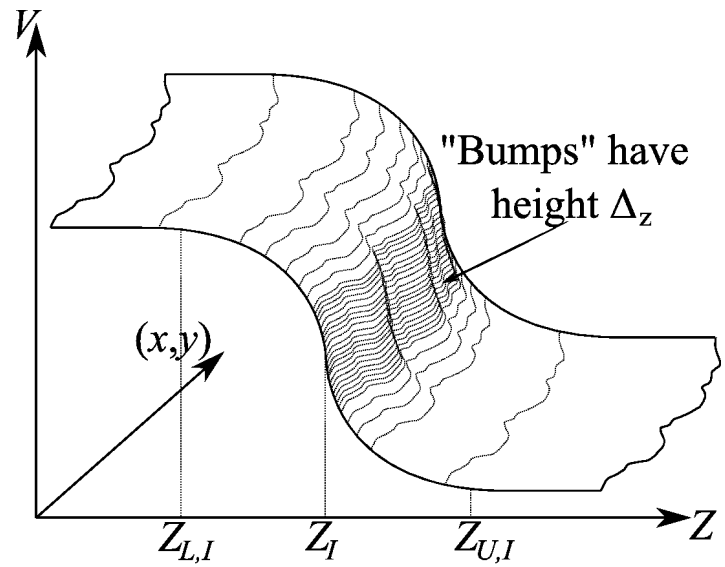


Sodipodi 0.34 (2004)

Inkscape for scientific illustrations

First encountered Inkscape while looking for a nice illustration package

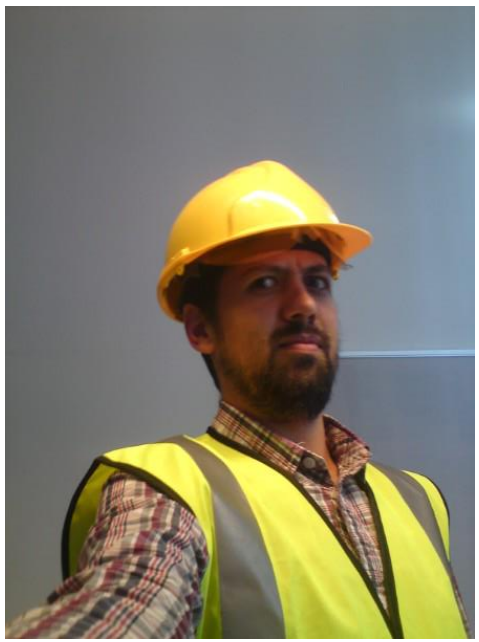
- Same vector image for posters, presentations and papers
- Free software
- Debian package available
- Fairly simple GUI
- LaTeX support
- EPS/PDF/bitmap support



Debian/Ubuntu packaging

- Upstream work-in-progress (v0.48.4)
- Lots of bugs still exist!!
  - 3610 upstream bugs
  - 93 Debian package bugs
  - 130 Ubuntu package bugs
- Slow Debian maintenance created a bottleneck
- I started Ubuntu bug triage, forwarding patches etc. . .

Deeper into the rabbit hole



- Grew tired of waiting for Debian package upgrades
- . . . Started working on Ubuntu package (2010)
- Upstream developer since May 2011
- Started off reviewing/applying patches from Ubuntu/Debian
- Pretty active over last 12 months

Project organisation

- 69 developers
  - 7 board members
  - 11 *active* developers ($\geq$ 10 commits in last year)
- Representation in W3C SVG working group
- Software Freedom Conservancy membership

Development policy

- Bazaar version control
- Launchpad for code, bugs, blueprints & answers but not translations
- Very open trunk access: 2-patch rule
- “patch first, discuss later”
- Not using time-based releases (yet!)
- Nightly & stable build PPAs provided

Architecture

A fairly outdated architecture diagram. . .

Architectural changes

Since the architecture diagram was made. . .

- Extensions subsystem
- Cairo graphics instead of custom renderer
- Pango for (some) text layout
- 2Geom library for geometry
- Dockable dialogs (GDL)
- C++/GtkMM migration

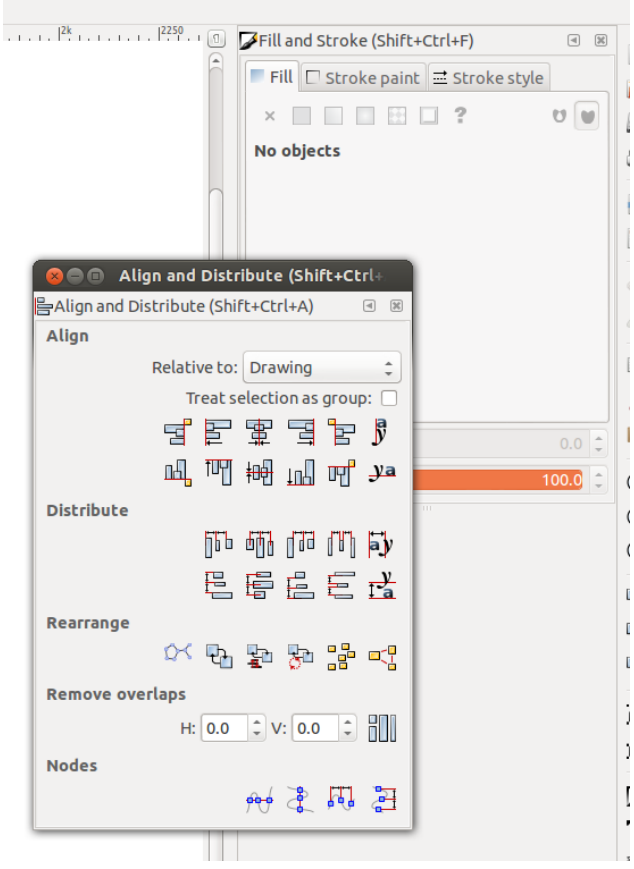
Gtk+ 3 migration

- Sodipodi GUI written with Gtk+ 1.x. No serious migration effort since then!
- **Major** API changes in Gtk+ 3.x:
  - Cairo rendering
  - Object heirarchy changed
  - Signalling changes
- *Hundreds* of changes needed in Inkscape
- Not just find-and-replace!
- 675 conditional build blocks!

Conclusion: Stay on top of library deprecations!

Gnome Docking Library (GDL)

- Gtk+ dockable dialog widget-set
- Fork copied into Inkscape trunk long ago
  - A couple of minor patches applied in fork
  - Major architectural changes upstream (Gtk+ 3)
  - Forked code Gtk+ 3 incompatible!
- I spent a **long** time untangling diffs, forwarding patches
- **Conclusion**: Don't adopt libraries unless you really have to!



Other work

Activity mostly involves build-testing and cleaning

- Launchpad PPAs
  - Nightly and stable builds
  - Inkscape 0.49 preview/testing for users
  - Early-warning system for build failures
- Deprecation testing/hardening
- cppcheck

Extension subsystem

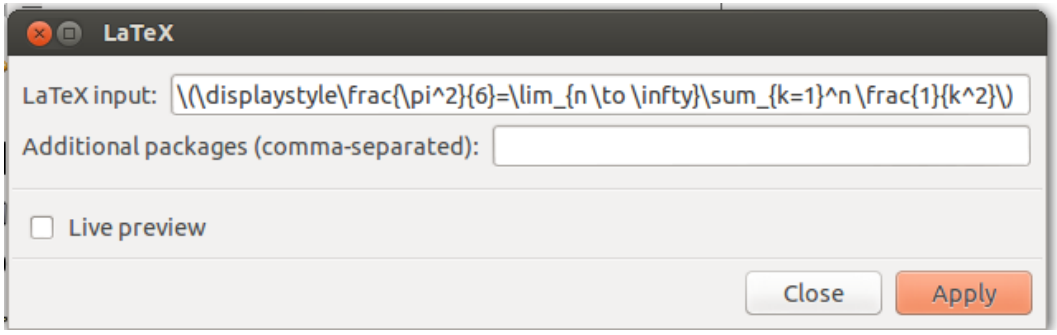
Inkscape extensions can provide various forms of functionality:

- **Input**: Translate an input file format into an SVG document
- **Output**: Translate an SVG document into an output file format
- **Effect**: Manipulate an SVG document

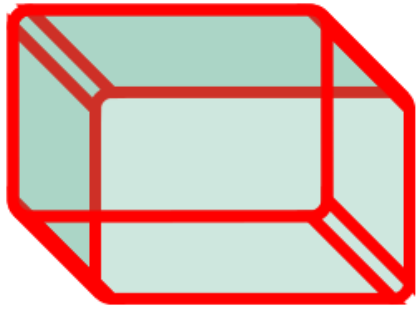
Extensions can be implemented in a number of ways, including:

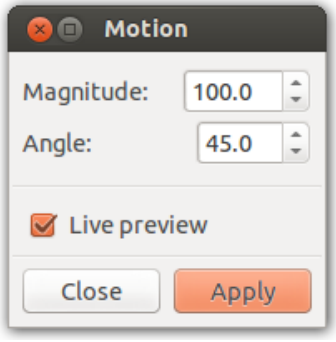
- **Internal**: C/C++ files that link directly into Inkscape
- **External Scripts**: Send SVG data via STDIN/STDOUT

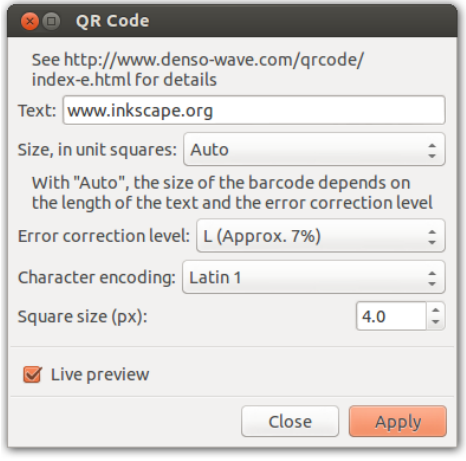
Example extensions



$$\frac{\pi^2}{6} = \lim_{n\to\infty} \sum_{k=1}^{n} \frac{1}{k^2}$$







Extension scripts

Inkscape runs extension scripts using the following syntax

```
(interpreter)? your_script (--param=value)* /path/to/input[[/SVGfile]] | inkscape
```

Note that:

- Any interpreter can be used (including Python!)
- Any number of parameters can be passed to the script
- Data is received by the script through STDIN
- Data is sent back to Inkscape through STDOUT

Extension descriptor file

- Specify parameters (to generate a GUI dialog)
- Define a menu entry in Inkscape

Extension descriptor file

- Define the script file and dependencies

```
1 <inkscape-extension>
2   <_name>Hello World!</_name>
3   <id>org.ekips.filter.hello_world</id>
4   <dependency type="executable" location="extensions">
        hello_world.py</dependency>
5   <dependency type="executable" location="extensions">
        inkex.py</dependency>
```

- "hello_world.py" is our effect script
- "inkex.py" provides an Effect base class
- Only loads if dependencies are met

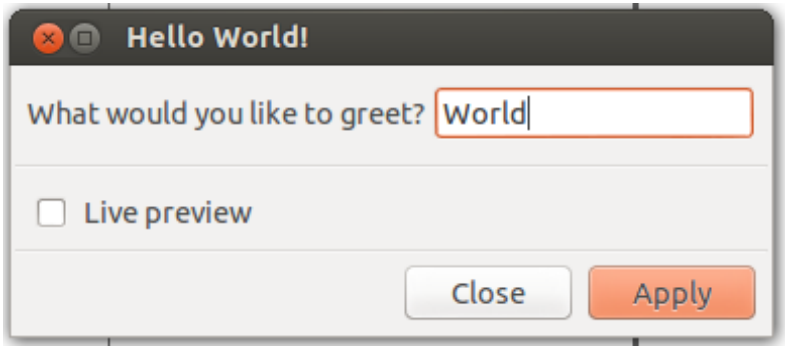
Extension descriptor file

An XML file “hello_world.inx” that provides a description of the extension script:

- Define parameters needed by the script

```
1  <param name="what" type="string" _gui-text="What
       would you like to greet?">World</param>
```

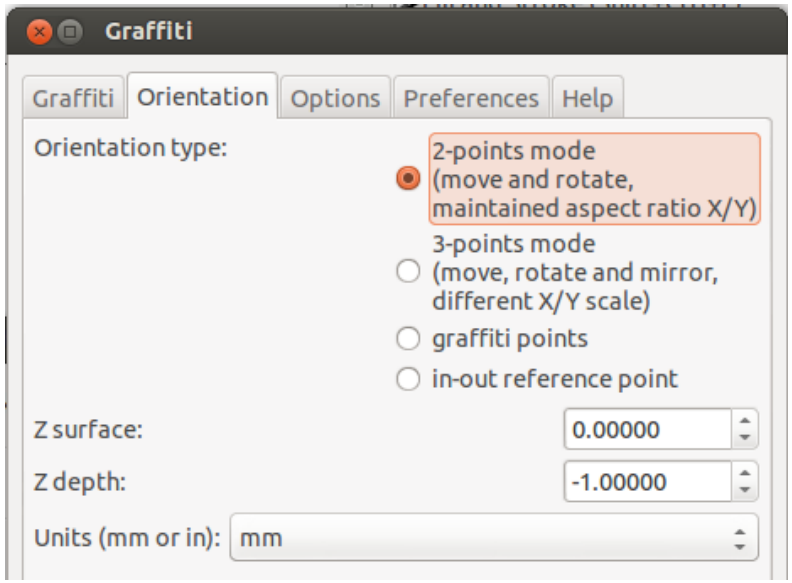
- Inkscape creates input widgets in an effect dialog



Extension descriptor file

Much fancier dialogs can be generated using param elements

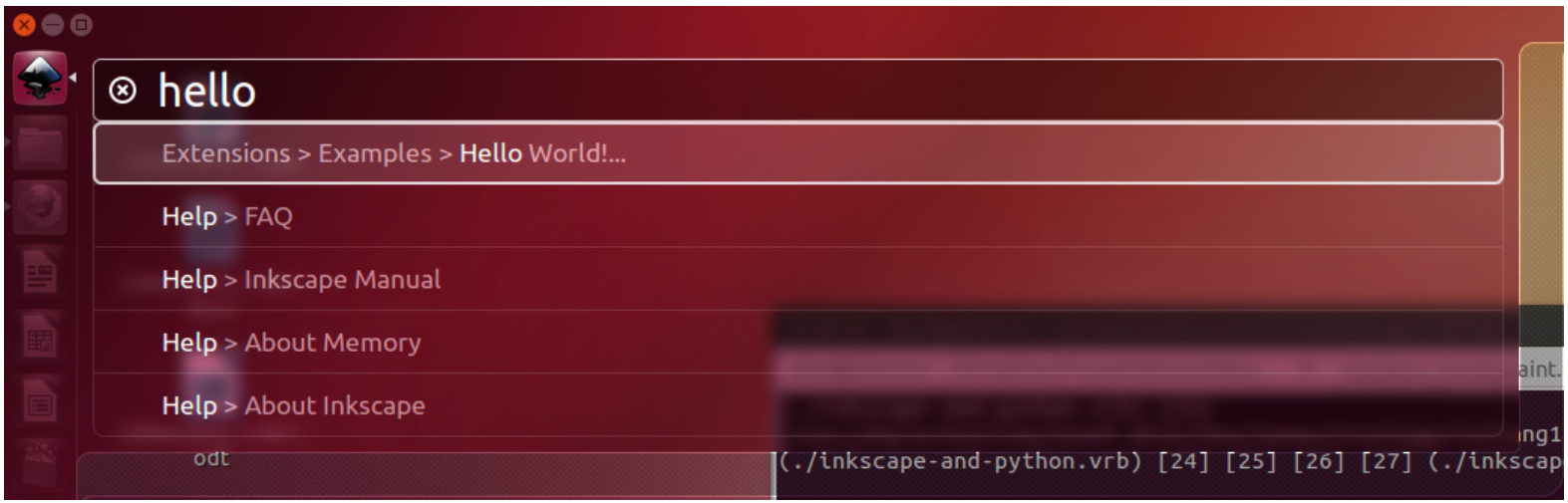
- Notebooks and pages
- Enumerations (drop-down lists)
- Option groups (radio buttons)
- Strings (text boxes)
- Numerics (spin-button)
- Colours (colour selector)



Extension descriptor file

- Define type of extension and menu location

```
1  <effect>
2    <object-type>all</object-type>
3    <effects-menu>
4        <submenu _name="Examples" />
5    </effects-menu>
6  </effect>
```



Extension descriptor file

- Define script command, location and interpreter

```
1  <script>
2    <command reldir="extensions" interpreter="python">
         hello_world.py</command>
3  </script>
4 </inkscape-extension>
```

- Python is most popular language for bundled extensions
- A couple of contributions in Perl and Ruby

Extension script

- inkex provides an effect base class and XML handling (lxml)
- simplestyle provides CSS style-handling

```
1 #!/usr/bin/env python
2
3 import sys
4 sys.path.append('/usr/share/inkscape/extensions')
5
6 import inkex
7 from simplestyle import *
```

Extension script

- Extend the Effect base class
- Handle the options passed from Inkscape

```
class HelloWorldEffect(inkex.Effect):
  def __init__(self):
    inkex.Effect.__init__(self)
    self.OptionParser.add_option('-w', '--what',
        action = 'store', type    = 'string',
        dest   = 'what',  default = 'World',
        help   = 'What would you like to greet?')
```

Extension script

- Override the effect method to modify SVG file as needed

```
1  def effect(self):
2    # Use options as required
3    what = self.options.what
4
5    # Get access to main SVG document element and get
         its dimensions.
6    svg    = self.document.getroot()
7    width  = inkex.unittouu(svg.get('width'))
8    height = inkex.unittouu(svg.get('height'))
```

Extension script

- Override the effect method to modify SVG file as needed

```
    # Create a new layer
    layer = inkex.etree.SubElement(svg, 'g')
    layer.set(inkex.addNS('label', 'inkscape'), 'Hello
        %s Layer' % (what))
    layer.set(inkex.addNS('groupmode', 'inkscape'), '
        layer')

    # Create text element
    text = inkex.etree.Element(inkex.addNS('text','svg'
        ))
    text.text = 'Hello %s!' % (what)
```

Extension script

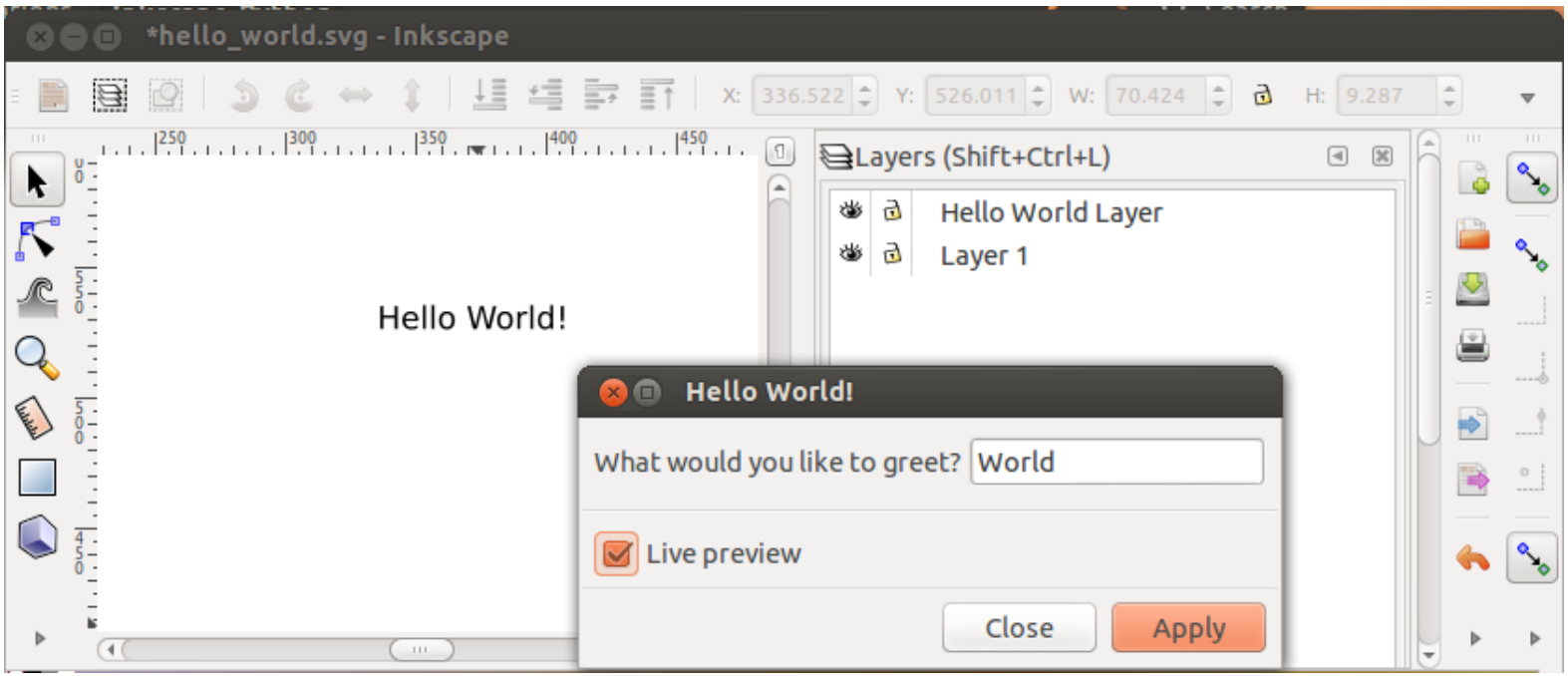
- Override the effect method to modify SVG file as needed

```
    # Set text position to center of document.
    text.set('x', str(width / 2))
    text.set('y', str(height / 2))

    # Center text horizontally with CSS style.
    style = {'text-align' : 'center', 'text-anchor': '
        middle'}
    text.set('style', formatStyle(style))

    # Connect elements together.
    layer.append(text)
```

Extension script

- Finally, instantiate effect class and apply it!

```
1 effect = HelloWorldEffect()
2 effect.affect()
```



New SVG elements

- Inkscape receives modified SVG file from script
- Inkscape redraws the file

```
<g
    id="g905"
    inkscape:groupmode="layer"
    inkscape:label="Hello World Layer">
  <text
      id="text907"
      style="text-anchor:middle;text-align:center"
      y="526.18110235"
      x="372.047244095">Hello World!</text>
</g>
```

Future directions

- Complete SVG 1.1 compliance
- Migrate to C++/GtkMM
- Replace custom text rendering with Pango
- Improve testing
- Python bindings for 2Geom library

Reflections

A few thoughts after 18 months with Inkscape:

- **Open development**: Good for new features, bad for consistency?
- **Forks of forks**: Is a messy codebase inevitable?
- **"Hands-off" management**: Contributors work well on pet projects. Slow progress on major project goals.
- **Language migration**: Is it worth it? Weird hybrid GObject/C++ code can be confusing! Unclear class heirarchies.
- **Testing**: Not enough of it! Integration with Debian build system?

Conclusions

- Inkscape provides a very welcoming development environment
- Code-base in need of a little spring cleaning!
- . . . but light is at the end of the tunnel
- Extensions subsystem is flexible and simple to use. Lots of fun for keen Python developers (hint!)

Acknowledgements

- Inkscape developers
- Peter Russell, WyPy
- Jonny Cooper, Inst. Microwaves and Photonics