aafigure README

Overview

The original idea was to parse ASCII art images, embedded in reST documents and output an image. This would mean that simple illustrations could be embedded as ASCII art in the reST source and still look nice when converted to e.g. HTML.

Since then the aafigure application also grew into a standalone application providing a command line tool for ASCII art to image conversion.

Installation

The package

To install aafigure, you need to have administrator rights on your system (be root). Type `python setup.py install` to install aafigure.

This installs a package that can be used from python (`import aafigure`) and a command line script called `aafigure`.

The Python Imaging Library (PIL) needs to be installed when support for bitmap formats is desired and it will need ReportLab for PDF output.

The docutils plugin

The docutils-aafigure extension depends on the aafigure package also requires `setuptools` (often packaged as `python-setuptools`) and docutils itself (0.5 or newer) must be installed.

After that, the `aafigure` directive will be available.

Implementation

Files in the `aafigure` package:

`aafigure.py`
: ASCII art parser. This is the main module.

`aa.py`
: ASCII art output backend. Intended for tests, not for the end user.

`pdf.py`
: PDF output backend. Depends on reportlab.

`pil.py`
: Bitmap output backend. Using PIL, it can write PNG, JPEG and more formats.

`svg.py`
: SVG output backend.

Files in the `docutils` directory:

`aafigure_directive.py`
: Implements the `aafigure` Docutils directive that takes these ASCII art figures and generates a drawing.

The `aafigure` module contains code to parse ASCII art figures and create a list of of shapes. The different output modules can walk through a list of shapes and write image files.

Usage

Command line tool

```
aafigure test.txt -t png -o test.png
```

The tool can also read from standard in and supports many options. Please look at the command's help:

```
aafigure --help
```

Within reStructured text

```
./rst2html.py README.txt >README.html
```

This results in the `README.html` file and a `.svg` file for each `aafigure`.

Display the resulting `README.html` file in a SVG capable browser. It has been tested with Firefox 1.5, 2.0 and 3.0.

Short introduction

This code in a reST document that is processed with the enhanced `rst2html.py` looks like this:

```
.. aafigure::

    -->
```

Which results in an image like this:

The `aafigure` directive has the following options:

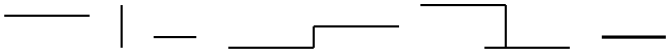
- `:scale: <float>` enlarge or shrink image
- `:line_width: <float>` change line with (svg only currently)
- `:format: <str>` choose backend/output format: 'svg', 'png', all bitmap formats that PIL supports can be used but only few make sense. Line drawings have a good compression and better quality when saved as PNG rather than a JPEG. The best quality will be achieved with SVG, tough not all browsers support this vector image format at this time.
- `:foreground: <str>` foreground color in the form `#rgb` or `#rrggbb`
- `:background: <str>` background color in the form `#rgb` or `#rrggbb` (*not* for SVG output)
- `:fill: <str>` fill color in the form `#rgb` or `#rrggbb`
- `:name: <str>` use this as filename instead of the automatic generated name
- `:aspect: <float>` change aspect ratio. Effectively it is the width of the image that is multiplied by this factor. The default setting `1` is useful when shapes must have the same look when drawn horizontally or vertically. However, `:aspect: 0.5` looks more like the original ASCII and even smaller factors may be useful for timing diagrams and such. But there is a risk that text is cropped or is draw over an object beside it.

  The stretching is done before drawing arrows or circles, so that they are still good looking.
- `:proportional: <flag>` use a proportional font instead of a mono-spaced one.

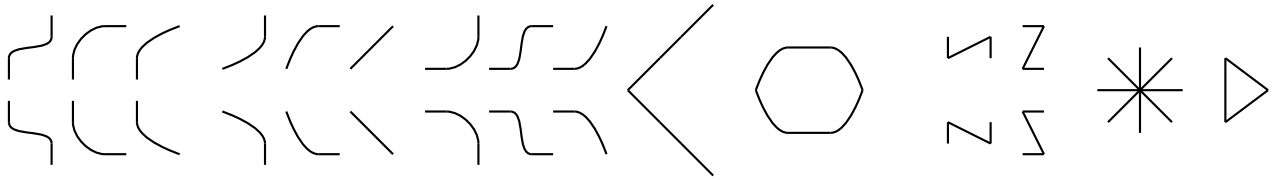
Lines

The - and | are normally used for lines. _ and ~ can also be used. They are slightly longer lines than the -. _ is drawn a bit lower and ~ a bit upper. = gives a thicker line. The later three line types can only be drawn horizontally.

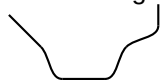
```
---- |          ___  ~~~|
     | --  ___|         |    ===
                        ~~~
```



It is also possible to draw diagonal lines. Their use is somewhat restricted tough. Not all cases work as expected.



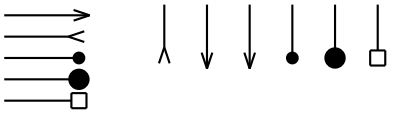
And drawing longer diagonal lines with different angles looks ugly...



Arrows

Arrow styles are:

```
--->   | | | | | |
---<   | | | | | |
---o   ^ V v o 0 #
---0
---#
```



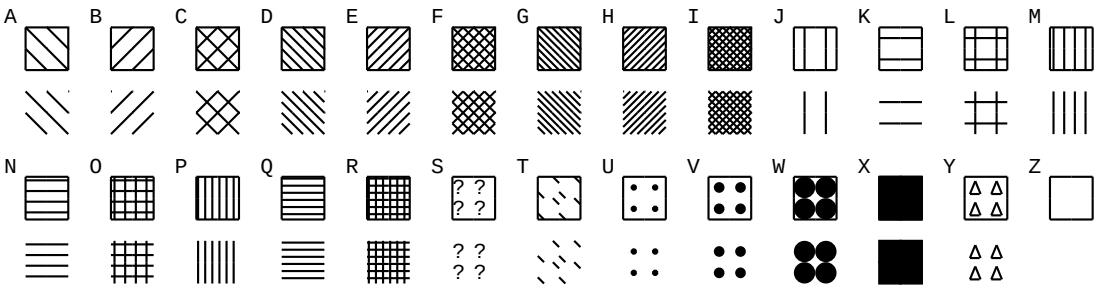
Boxes

Boxes are automatically draw when the edges are made with +, filled boxes are made with X (must be at least two units high or wide). It is also possible to make rounded edges in two ways:

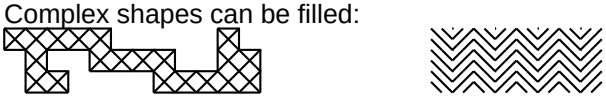
```
+-----+   XXX  /--\     --   |
|     |   XXX  |  |    /    /
+-----+   XXX  \--/   |   --
```

Fills

Upper case characters generate shapes with borders, lower case without border. Fills must be at least two characters wide or high. (This reduces the chance that it is detected as Fill instead of a string)



Complex shapes can be filled:



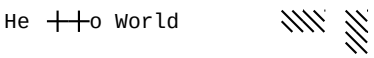
Text

The images may contain text too. There are different styles to enter text:

direct

By default are repeated characters detected as fill:

```
Hello World  dd d
               d
```



quoted

Text between quotes has priority over any graphical meaning:

```
"Hello World"  dd d
                 d
```

`"`, `'` and `\`` are all valid quotation marks. The quotes are not visible in the resulting image. This not only disables fills (see below), it also treats `-`, `|` etc. as text.

textual option

The `:textual:` option disables horizontal fill detection. Fills are only detected when they are vertically at least 2 characters high:

```
Hello World  dd d
               d
```

Other

```
* { }
```

● < >

TODO

- Symbol detection: scan for predefined shapes in the ASCII image and output them as symbol from a library
- Symbol libraries for UML, flowchart, electronic schematics, ...
- The way the image is embedded is a hack (inserting a tag trough a raw node...)
- Search for ways to bring in color. Ideas:

    - have an :option: to set color tags. Shapes that touch such a tag inherit it's color. The tag would be visible in the ASCII source tough:

      ```
      .. aafigure::
          :colortag: 1:red, 2:blue

          1--->  --->2
      ```

    - `:color: x,y,color` but counting coordinates is no so fun

  drawback: both are complex to implement, searching for shapes that belong together. It's also not always wanted that e.g. when a line touches a box, both have the same color
- aafigure probably needs arguments like `font-family`, ...
- Punctuation not included in strings (now a bit improved but if it has a graphical meaning , then that is chooses, even if it makes no sense), underlines in strings are tricky to detect...
- Dotted lines? `...`
- Group shapes that belong to an object, so that it's easier to import and change the graphics in a vector drawing program.
- Path optimizer, it happens that many small lines are output where a long line could be used.

Authors and Contact

- Chris Liechti: original author
- Leandro Lucarella: provided many patches

The project page is at https://launchpad.net/aafigure It should be used to report bugs and feature requests.

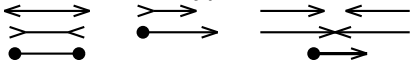
License

BSD

Tests

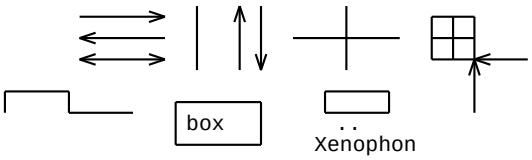
Simple tests

Different arrow types:

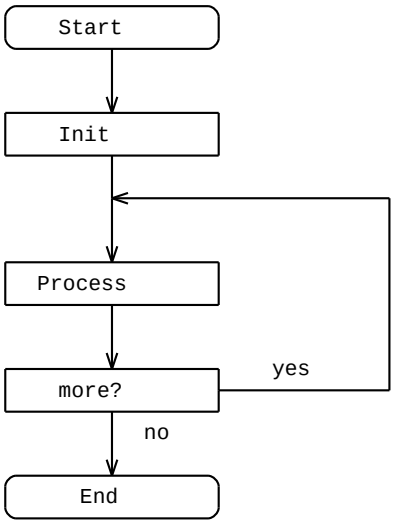


Boxes and shapes:

```
A box with text
```

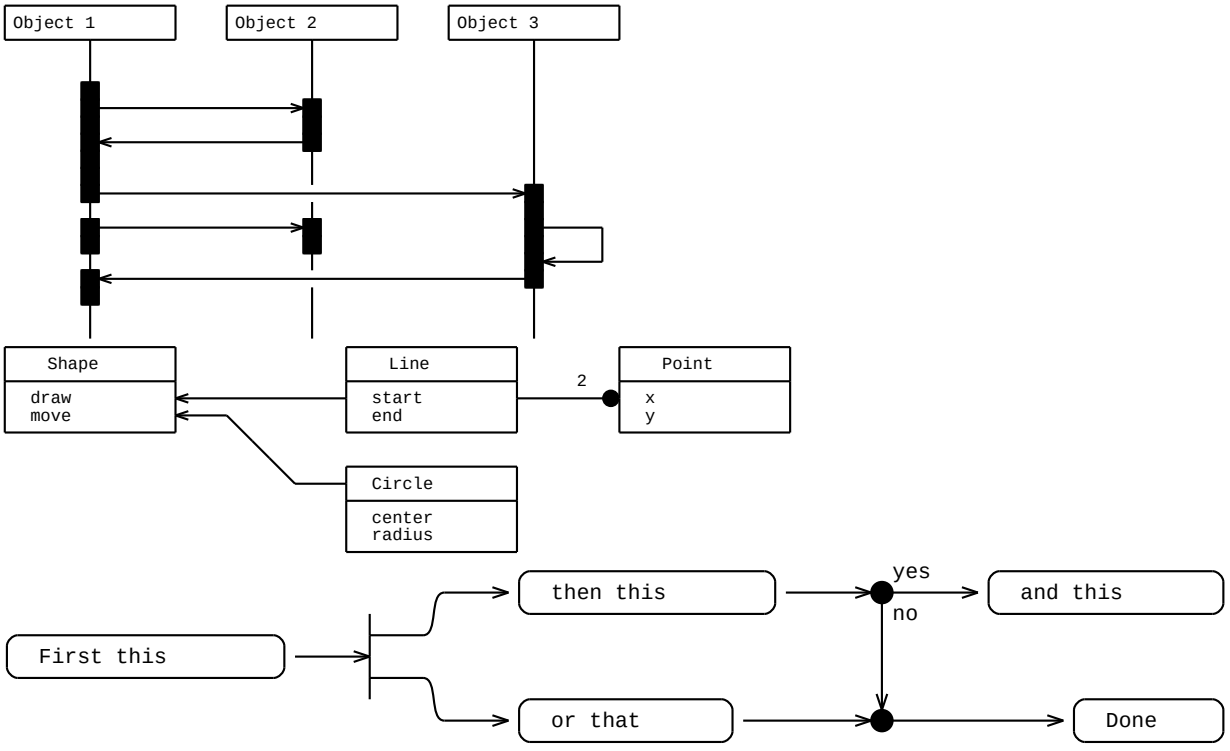


Flow chart



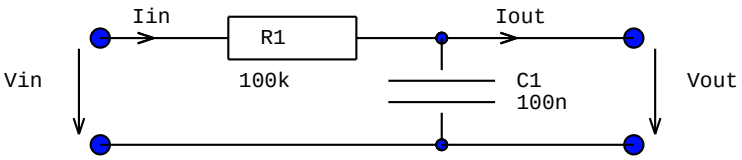
UML

No not really, yet. But you get the idea.

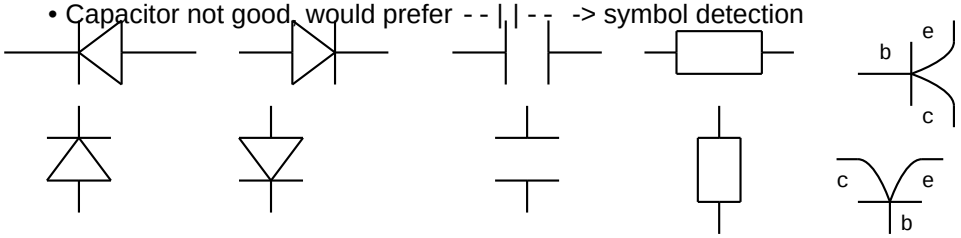


Electronics

It would be cool if it could display simple schematics.

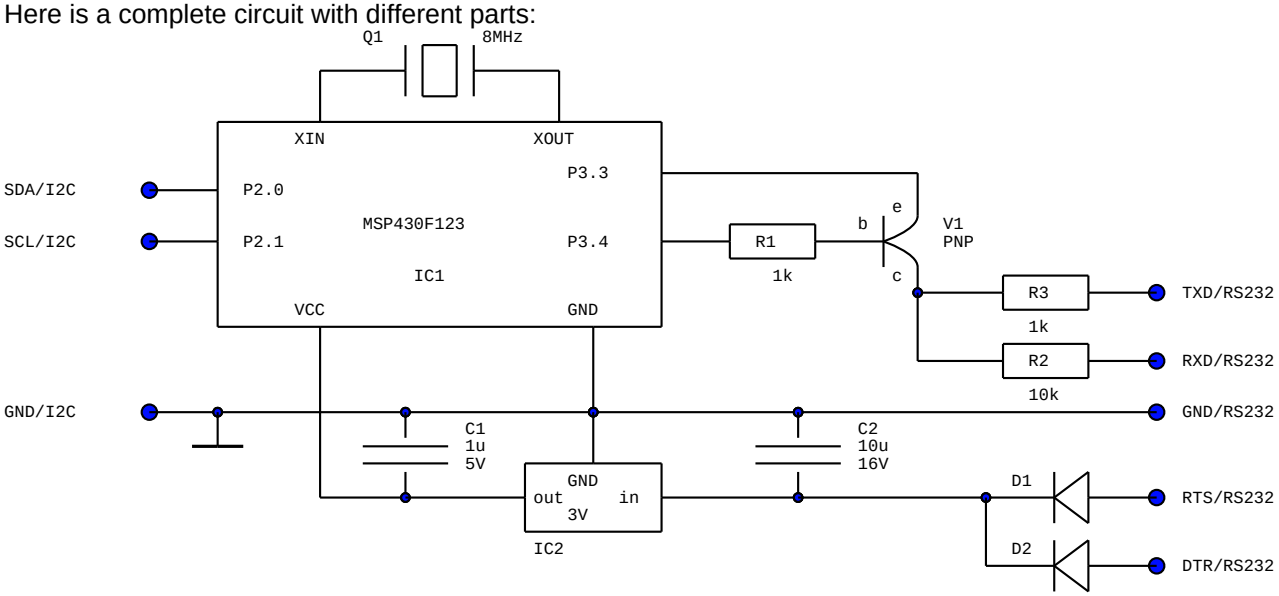


- Capacitor not good, would prefer - - | | - - -> symbol detection

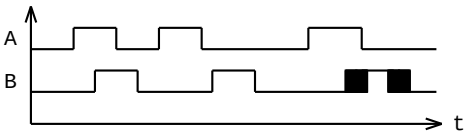


- Diodes OK
- Caps not optimal. Too far apart in image, not very good recognisable in ASCII. Space cannot be removed as the two + signs would be connected otherwise. The schematic below uses an other style.
- Arrows in transistor symbols can not be drawn

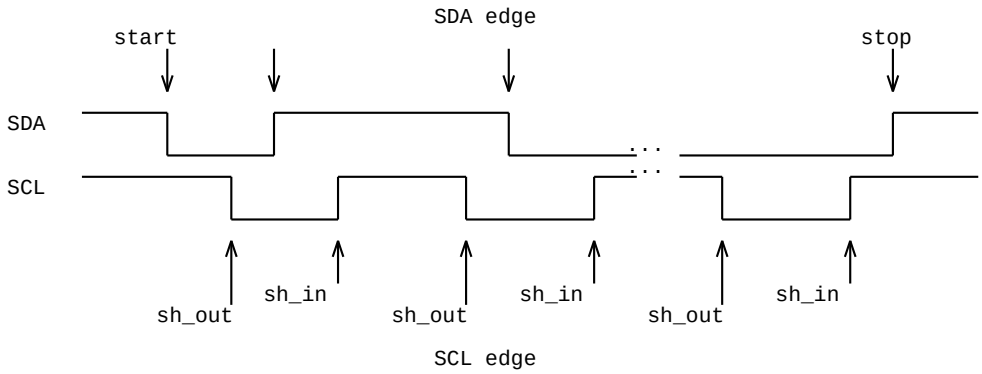
Here is a complete circuit with different parts:



Timing diagrams

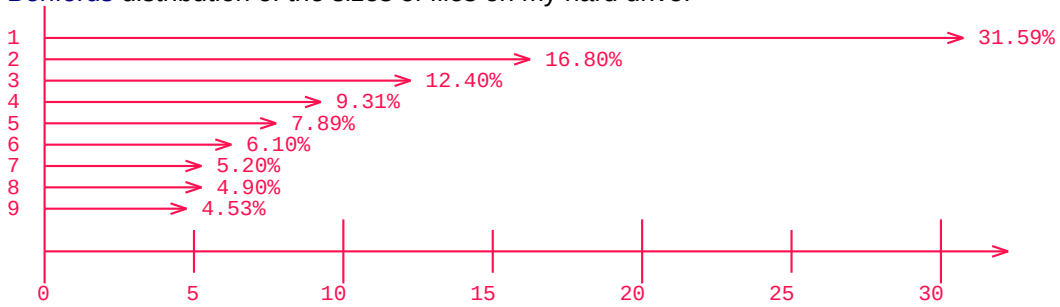


Here is one with descriptions:

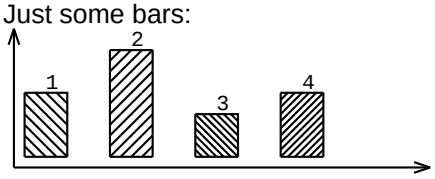


Statistical diagrams

Benfords distribution of the sizes of files on my hard drive:



Just some bars:



Schedules