Shoebot

Release 1.3.1

Feb 10, 2021

Contents

Shoebot is a tool to automate the process of drawing vector graphics using a minimal and easy-to-understand syntax.

It is a rewrite of Nodebox 1 by Frederik de Bleser and Tom de Smedt, with the purpose of having an equivalent tool in GNU/Linux systems. Nodebox 1 is itself based on DrawBot by Just van Rossum. Shoebot draws using the cross-platform Cairo graphics engine.

It follows a rich lineage of tools dedicated to generative creativity:

- Design By Numbers
- Processing
- Scriptographer
- Paper.js

For more about the nature of creative coding and generative design, be sure to read The Nodebox 1 theoretical introduction.

CHAPTER 1

Purpose

Shoebot is a useful tool for many use cases:

- creating generative and procedural works, either for screen or print output
- teaching code to non-developers by means of immediate visual feedback
- prototyping visualizations and design concepts
- automatically generating sets of vector images
- live-coding and real-time tweaking of animated graphics
- making Cairo-based tools and experiments using a simpler language

CHAPTER 2

Features

Originally built as a GNU/Linux version of Nodebox 1, Shoebot comes with many batteries included:

- supports most of Nodebox 1's functionality
- user-friendly code editor
- headless mode for quick execution without the GUI parts
- run on a window or output files in PDF, PNG, SVG or PS formats
- tweak running scripts via a simple GUI, a socket server or a text-based shell
- can be loaded as a Python module to work inside existing programs

CHAPTER 3

Links

- Main site
- GitHub repository
- Issue tracker

CHAPTER 4

Authors

Shoebot is currently maintained by Stuart Axon and Ricardo Lafuente.

A good part of the code has also been contributed to by Francesco Fantoni, Sebastian Oliva, Paulo Silva, Dave Crossland, Gabor Papp, Julien Deswaef , Pedro Ângelo and Tetsuya Saito. Examples also contributed to by Artem Popov, Barak Itkin and Simon Budig.

CHAPTER 5

Related projects

- Drawbot, the tool that started it all, by Just van Rossum from Letterror. It's now been reinvigorated with new features and experimental Python 3 support.
- Nodebox 3, the current node-based incarnation of Nodebox, running on Java.
- Cairo DrawBot, a GNU/Linux compatible fork of Drawbot by Eli Heuer.
- Nodebox-pyobjc, an active fork of Nodebox 1 for Mac, maintained by Karsten Wolf.
- PlotDevice, another fork of Nodebox 1 for Mac, maintained by Christian Swinehart.
- Canvas.js by Nodebox's co-author Tom de Smedt, allows you to create browser-based scenes and animations with JS and HTML5.
- Rapydbox – A JavaScript version of Nodebox, based on RapydScript.

CHAPTER 6

Documentation Index

6.1 Installation

6.1.1 GNU/Linux

Shoebot runs on Python 3.7 and above.

You need a few software packages on your system before installing Shoebot. There is a small handy script that will take care of this for you:

```
./install/install_dependencies.sh
```

It is recommended to install Shoebot locally, although it can be also be installed system-wide.

If the script does not support your operating system, skip to *Add support for another operating system*.

Local install

Installing shoebot for the current user.

```
python3 setup.py install
```

Local install using virtualenvwrapper

If you're using the handy virtualenvwrapper, these are the necessary commands:

```
mkvirtualenv shoebot -p $(which python3)
python3 setup.py install
```

To use Shoebot in the future, you will need to activate the environment first:

```
workon shoebot
```

Local install using a plain virtualenv

If you don't use virtualenvwrapper, run these commands after installing the dependencies.

```
virtualenv .env
source .env/bin/activate
python3 setup.py install
```

To use shoebot in the future, remember to activate the environment first.

```
source .env/bin/activate
```

6.1.2 System wide install

```
sudo python3 setup.py install
```

6.1.3 Mac OS X

Installation on Mac OS X is identical to GNU/Linux based distributions.

Dependencies are installed via *Homebrew <https://brew.sh/>_* through the `install/install_dependencies.sh` script.

Python 3.8 is supported on Homebrew, since that is what is currently supported by pygobject3 there.

6.1.4 Windows

Windows is currently untested. There used to be a purpose-built Windows version of Shoebot (Spryte) but it has been unmaintained for a long while.

If you try your hand at running Shoebot on Windows and can get *anything* running, let us know in our issue tracker so we can improve this documentation.

6.1.5 Add support for another operating system

To add support for another OS you will need to install the libraries that Shoebot depends on:

Core:

```
Python3 Pycairo Pygobject3 Pango
```

GUI:

```
Gtk3 Gtksourceview
```

The community for your operating system may be able to offer help here. Looking at how the *install_dependencies.sh* script works for may help.

Check progress with diagnose

Shoebot provides a *diagnose* command as part of setup to check if things are working.

```
python3 setup.py diagnose
```

It's usually easiest to start with Python3 and Pycairo, then move on to PyGobject, Pango and Gtk3.

PGI with CairoCFFI and Gtk3

Shoebot can run under PGI and CairoCFFI, which may be easier to install than the recommened setup with pygobject and cairo.

In this setup Shoebot can work with the GUI, but text output is not available.

Open a bug on the Shoebot issue tracker

Open a bug on the issue tracker to track progress on adding your OS.

https://github.com/shoebot/shoebot/issues

6.2 Getting Started

6.2.1 Running an example script

For your first time with Shoebot, try out the included code editor by running

```
shoebot
```

Open one of the example scripts and Run it through the *Run* -> *Run script* menu option or the `Ctrl-R` keyboard shortcut.

A window should open with an image. Shoebot reads scripts written in the Nodebox language and translates them into images or animations.

You can right click anywhere on the image window to view the output in full screen or export it as an image or vector file.

6.2.2 Using the console runner

If you prefer using your own text editor to edit files and just want something to run the scripts, the *sbot* command is what you want. Head over to the *examples/* directory and try running:

```
sbot grid/balls.bot
```

6.2.3 Headless operation

There is also a mode where the result is directly rendered into an image or vector file, without a window. We can do this with the `-o` option, short for `--outputfile`:

```
sbot grid/balls.bot -o balls.png
```

This will skip the window view and create the `balls.png` image file. You can also create SVG, PDF and PostScript (`.ps`) files.

For a list of extra options, see the *Command-line options* section.

6.3 Tutorial

The original Nodebox tutorial pages are an excellent introduction to the concepts that you'll also find in Shoebot.

- Introduction – generative art, computational design, and getting started
- Basics
  - Environment – the basics about the Nodebox working environment
  - Primitives – command parameters and drawing shapes
  - Graphics state – how shapes can be rotated and transformed
- Data
  - Variables – saving values and reusing them
  - Lists – sequences of variables
  - Strings – manipulating and working with text
- Strategy
  - Repetition – doing things over and over
  - Commands – creating custom behaviors
  - Classes – more advanced coding concepts
  - Libraries – going beyond the core
- Specifics
  - Animation
  - Interaction – working with the keyboard and mouse
  - Color – grasping color objects
  - Math – geometry and other applied math concepts
- Bézier Paths
  - Paths – the basics of Béziers
  - Manipulating paths – tweaking points on a path
  - Path Mathematics finding points in a path, inserting new ones
  - Path Filters applying effects on a path
  - Compound paths – path operations (unsupported?)
  - Clamping paths – limiting paths to a box (unsupported?)
- Advanced
  - Console

We've omitted the tutorials that relate to Mac-specific details or non-implemented features (like Psyco support). See the Nodebox tutorial page for all of them.

6.4 Live Variables

Shoebot is not limited to what you write inside your .bot file; you can open your sketch for receiving and reacting to data from outside applications and environments.

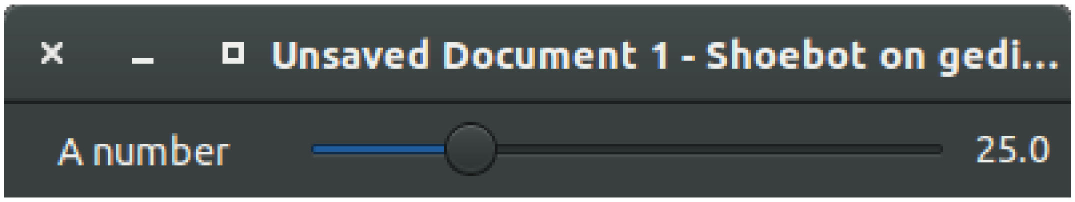
These can be created using the `var()` command.

6.4.1 Using the live variables GUI

When a script uses the *var* keyword, the corresponding widget will appear in the live variables GUI.

The following code will make a slider with a minimum value of 0, a maximum value of 100, and an initial (default) value of 25.

```
var('a_number', NUMBER, 25., 0., 100.)
```



For readability names are capitalized and underscores replaced by spaces, so 'a_number' is displayed as 'A number'.

6.4.2 Setting variables in the command line

Live variables can be set from the commandline using JSON syntax:

```
$ sbot --vars='{ "hue": 32 }' examples/vars/circle_circle.bot
```

This means that a Shoebot script can be called with different parameters without having to change the script itself – just specify the live variable initial values with `--vars`.

6.4.3 Socket server

If shoebot is run with the –*serverport* option, a socket server will also be started. Socket servers can accept connections from other applications or even across the network in order to set variables inside running Shoebot scripts.

```
$ sbot -s examples/animation/hypnoval.bot
Listening on port 7777...
```

Once it's running, it's easy to connect with telnet:

```
$ telnet 127.0.0.1 7777
```

This gets you into the shell, where you can use the commands below to list and set variables, rewind and go to frames.

The following commands can also be sent through other applications like Pure Data; there is a simple PD example in `examples/socketcontrol/helloworld.pd`. Be sure to take a look at the socket server examples in action inside the *examples/socketserver* directory.

Socketserver commands

Command	Description
goto 100	go to frame 100
pause	pause playback
rewind	set FRAME to 0
restart	set FRAME to 0 and reset all variables
vars	show content of all live variables
set n=1	set variable 'n' to value 1
n=1	set variable 'n' to value 1
help	show list of all commands

6.4.4 Using the live shell

Shoebot provides a live shell for communication with text editors, as well as livecoding.

All the socket server commands above are available, along with `load_base64` which allows livecoding.

To experiment with the shell, run an example with the `-l` option:

```
$ sbot -l examples/animation/hypnoval.bot
```

The shell accepts all the socket server commands, along with a couple more; these can be useful for an editor or IDE.

Live Shell commands

Command	Description
quit	quit shoebot
load_base64	used by IDE/Editor to send new code to Shoebot

6.5 Libraries

6.5.1 Nodebox libraries

Nodebox has a rich set of external libraries to enable new tools and commands.

Shoebot comes with some of these external libraries included. Read the original Nodebox documentation to get these going:

- Knowledge: Database, Graph, Web
- Bitmap: Photobot
- Paths: Bezier, Cornu, SVG Supershape, Bezier Editor
- Systems: Boids, L-system

- Design: Colors
- Tangible: TUIO

6.5.2 Shoebot libraries

Shoebot also includes a set of libraries of its own. We still have to document them. Ping us on the issue tracker if you need help.

Audio

Grab audio frequencies to create visualizations.

Import it with `ximport("sbaudio")`.

Requires the `pysoundcard` Python module to work.

`fft_bandpassfilter`(*data*, *fs*, *lowcut*, *highcut*)

`flatten_fft`(*scale=1.0*)

`scaled_fft`(*fft*, *scale=1.0*)

`triple`(*spectrogram*)

`fuzzydevices`(*match=""*, *min_ratio=30*)

`firstfuzzydevice`(*match=""*)

Video

This is a **very** basic video library. The library itself is built upon OpenCV, a library of programming functions mainly aimed at real time computer vision.

You need `opencv` and its python bindings in order to use this library, so on debian-based Linux you will have to install `python-opencv` and `libhighgui1` or similar.

The Video Library is meant to grab frames from webcams, surveillance cameras or video files, and to make them available to Shoebot for displaying and elaboration.

For cameras, two camera interfaces can be used on Windows: Video for Windows (VFW) and Matrox Imaging Library (MIL). On Linux, we can use V4L and FireWire (IEEE1394).

For video files, this library uses specific backends on each OS:

- ffmpeg on Linux
- Video for Windoes (VfW) on Windows
- QuickTime on Mac OS X

Up to now only camera and video frame grabbing is supported, but future development could introduce features like face and object recognition or video output to file for Shoebot animation. Future development will depend on evolution of the SWIG python bindings of OpenCV, which at present is still in-progress.

Basic usage

First, import the library:

```
videolib = ximport("sbvideo")
```

Then you can create two different types of capture devices:

```
# for video files
video = videolib.movie(file_path)
# or for webcam
camera = videolib.camera(index, width, height)
```

If you have only one camera, you can omit `index` and the first camera should be picked. Parameters `width` and `height` are optional, and they're not guaranteed to work with your camera. Now you can grab a video frame with:

```
frame = video.frame() or frame = camera.frame()
```

frame has some properties: frame.width frame.height frame.time # strange results at present frame.data

You need `frame.data` in order to pass your frame image to the Shoebot canvas using the *`image()`* command:

```
image(None, xpos, ypos, data=frame.data)
```

You can manipulate and use the image as any other image in Shoebot.

When you call the video-file capture constructor, you're supposed to be able to set a starting point in seconds from file beginning, but at present this feature is disabled as I could not get it to work properly on my linux system If you want to test it you can uncomment the relative lines in SVL. (If you succeed, please report it to Shoebot dev mailing list).

Commands

`sbvideo.`**`movie`**(*path*, *start=0*, *stop=None*)

`sbvideo.`**`camera`**(*cam=0*, *width=None*, *height=None*)

OpenCV

`sbopencv.`**`movie`**(*path*, *start=0*, *stop=None*)

`sbopencv.`**`camera`**(*cam=0*, *width=None*, *height=None*)

`sbopencv.`**`image`**(*path=None*)

`sbopencv.`**`ipl2cairo`**(*iplimage*)

`sbopencv.`**`detectHaar`**(*iplimage*, *classifier*)

`sbopencv.`**`findcontours`**(*iplimage*, *threshold=100*)

6.6 Example scripts

This is a set of example scripts from the `examples` directory.

6.6.1 How curves work

_static/how_curves_work.png

6.7 Advanced usage

This section is aimed at Python dabblers and hackers who want to get into the more involved features of Shoebot.

6.7.1 Using Shoebot as a Python module

Shoebot can be easily loaded as a module inside a Python script.

```
import shoebot

# set up a canvas
bot = shoebot.create_bot(outputfile="output.svg")

# size() needs to be called first
bot.size(400,400)

# now we can draw!
bot.rect(10,10,100,100)

# finish() should be called after all drawing commands
bot.finish()
```

Take a snapshot of the current state:

```
bot.snapshot("snap.png")
```

Run a Shoebot/Nodebox script:

```
bot.run("example.bot")
```

6.7.2 Running in Jupyter

*Jupyter notebooks <https://jupyter.org>*_ are fantastic, and Shoebot runs pretty well inside them!

First, you need to have Jupyter installed, as well as the development version of Shoebot. Using `virtualenvwrapper` for this is heavily recommended.

```
# create the virtualenv
mkvirtualenv jupytershoebot -p $(which python3)
# install jupyter dependencies
pip3 install jupyter jupyter-pip
# clone the Shoebot repository, enter it and install
git clone https://github.com/shoebot/shoebot
```

(continues on next page)

(continued from previous page)

```
cd shoebot
python3 setup.py install
```

After ensuring both packages are available, install the extension after cloning the jupyter-shoebot repository:

```
# leave the shoebot/ dir
cd ..
# clone the jupyter-shoebot repository, enter it and install
git clone https://github.com/shoebot/jupyter-shoebot
cd jupyter-shoebot
python3 setup.py install
```

And finally, while still on the `jupyter-shoebot/` directory, run

```
jupyter kernelspec install shoebot_kernel --sys-prefix
```

All done! Now you can run `jupyter notebook`, go to the `Kernel` menu, select `Change kernel` and select `Shoebot`.

Be sure to try the *notebook examples <https://github.com/shoebot/jupyter-shoebot/tree/master/example-notebooks>*_ in the Jupyter Shoebot repository.

6.7.3 Running with PyPy

To get better performance, you can run Shoebot using PyPy3, which is experimental.

When installing Shoebot, you have to point to PyPy3 when creating your virtualenv. Instead of the first command in the *Virtualenvwrapper install example*, do:

```
mkvirtualenv shoebot -p $(which pypy3)
```

For the plain virtualenv approach, try:

```
virtualenv .env -p $(which pypy3)
```

6.7.4 Using with Django

See the shoebot-django for an example of integrating Shoebot into a Django application.

6.8 Extensions

There are a few plug-ins and extensions that make for a much better experience.

6.8.1 Gedit

This Gedit plugin adds a menu to allow for running Shoebot scripts. It will open a Shoebot window using the current document as the Shoebot code.

Run the following commands to add the plugin to Gedit:

```
cd shoebot/extensions/gedit
python3 install.py
```

Now restart Gedit. Navigate to `Edit > Preferences > Plugins` and activate the Shoebotit plugin.

TODO: document plugin usage

6.9 Troubleshooting

6.9.1 The Gedit plugin does not activate

Try running Gedit from the command line so that you can see debug messages. If you see one of these warnings:

```
** (gedit:3830): WARNING **: Could not load Gedit repository: Typelib file for␣
↪namespace 'GtkSource', version '3.0' not found
```

or:

```
ImportError: cannot import name Gedit
```

then try installing the *gir1.2-gtksource-3.0* package:

```
sudo apt-get install gir1.2-gtksource-3.0
```

This StackOverflow answer helped on finding this solution. However, if you see this:

```
(gedit:6027): libpeas-WARNING **: Could not find loader 'python3' for plugin
↪'shoebotit'
```

It might be because you're using an outdated version of Gedit. We've found this issue on Gedit 3.4.x, and it disappeared after updating to version 3.8.

6.9.2 TypeError: Couldn't find foreign struct converter for 'cairo.Context'

If you see this error, it means you're missing the Python GObject interface for cairo. On Debian/Ubuntu, this should be fixed with:

```
sudo apt-get install python3-gi-cairo
```

6.10 Command reference

This documentation is still missing some parts. Refer to the Nodebox documentation for the best reference in the meantime.

6.10.1 Drawing shapes

`rect` (*x*, *y*, *width*, *height*, *roundness=0*, *draw=True*, *fill=None*)
Draw a rectangle.

Parameters

- **`x`** – top left x-coordinate
- **`y`** – top left y-coordinate
- **`width`** – rectangle width
- **`height`** – rectangle height
- **`roundness`** – rounded corner radius
- **`draw`** (*`boolean`*) – whether to draw the shape on the canvas or not
- **`fill`** – fill color

`_static/drawing_shapes__rect.png`

`ellipse` (*x*, *y*, *width*, *height*, *draw=True*)
Draw an ellipse. Same as `oval()`.

Parameters

- **`x`** – top left x-coordinate
- **`y`** – top left y-coordinate
- **`width`** – ellipse width
- **`height`** – ellipse height
- **`draw`** (*`boolean`*) – whether to draw the shape on the canvas or not

`_static/drawing_shapes__ellipse.png`

`arrow` (*x*, *y*, *width*, *type=NORMAL*, *draw=True*)
Draw an arrow.

Parameters

- **`x`** – arrow tip x-coordinate
- **`y`** – arrow tip y-coordinate
- **`width`** – arrow width (also sets height)
- **`type`** (*`NORMAL or FORTYFIVE`*) – arrow type
- **`draw`** (*`boolean`*) – whether to draw the shape on the canvas or not

`_static/drawing_shapes__arrows.png`

`star` (*startx*, *starty*, *points=20*, *outer=100*, *inner=50*, *draw=True*)
Draw a star-like polygon.

Parameters

- **`startx`** – center x-coordinate
- **`starty`** – center y-coordinate
- **`points`** – amount of points
- **`outer`** – outer radius
- **`inner`** – inner radius
- **`draw`** (*`boolean`*) – whether to draw the shape on the canvas or not

`_static/drawing_shapes__stars.png`

`line` (*x1*, *y1*, *x2*, *y2*, *draw=True*)
Draw a line from (x1,y1) to (x2,y2).

Parameters

- **`x1`** – x-coordinate of the first point
- **`y1`** – y-coordinate of the first point
- **`x2`** – x-coordinate of the second point
- **`y2`** – y-coordinate of the second point
- **`draw`** (*`boolean`*) – whether to draw the shape on the canvas or not

`_static/drawing_shapes__line.png`

`rectmode`(*mode=None*)

Change the way rectangles are specified. Each mode alters the parameters necessary to draw a rectangle using the `rect()` function.

Parameters **`mode`** (*`CORNER, CENTER or CORNERS`*) – the mode to draw new rectangles in

There are 3 different modes available:

- **CORNER mode (default)**
  - x-value of the top left corner
  - y-value of the top left corner
  - width
  - height
- **CENTER mode**
  - x-coordinate of the rectangle's center point
  - y-coordinate of the rectangle's center point
  - width
  - height
- **CORNERS mode**
  - x-coordinate of the top left corner
  - y-coordinate of the top left corner
  - x-coordinate of the bottom right corner
  - y-coordinate of the bottom right corner

So while you always specify 4 parameters to the `rect()` function, you can use `rectmode()` to change the function's behaviour according to what might suit your script's needs.

`_static/drawing_shapes__rectmode.png`

`ellipsemode`(*mode=None*)

Change the way ellipses are specified. Each mode alters the parameters necessary to draw an ellipse using the `ellipse()` function.

It works exactly the same as the `rectmode()` command.

_static/drawing_shapes__ellipsemode.png

6.10.2 Bézier paths

beginpath(*x=None*, *y=None*)

Begin drawing a Bézier path. If x and y are not specified, this command should be followed by a `moveto()` call.

Parameters

- **x** (*float or None*) – x-coordinate of the starting point
- **y** (*float or None*) – y-coordinate of the starting point

moveto(*x*, *y*)

Move the Bézier "pen" to the specified point without drawing; coordinates are absolute.

Parameters

- **x** (*float*) – x-coordinate of the point to move to
- **y** (*float*) – y-coordinate of the point to move to

relmoveto(*x*, *y*)

Move the Bézier "pen" to the specified point without drawing; coordinates are relative to the pen's current location.

Parameters

- **x** (*float*) – x-coordinate of the point to move to, relative to the pen's current point
- **y** (*float*) – y-coordinate of the point to move to, relative to the pen's current point

lineto(*x*, *y*)

Draw a line from the pen's current point; coordinates are absolute.

Parameters

- **x** (*float*) – x-coordinate of the point to draw to, relative to the pen's current point
- **y** (*float*) – y-coordinate of the point to draw to, relative to the pen's current point

rellineto(*x*, *y*)

Draw a line from the pen's current point; coordinates are relative to the pen's current location.

Parameters

- **x** (*float*) – x-coordinate of the point to draw to, relative to the pen's current point
- **y** (*float*) – y-coordinate of the point to draw to, relative to the pen's current point

curveto(*x1*, *y1*, *x2*, *y2*, *x3*, *y3*)

arc(*x*, *y*, *radius*, *angle1*, *angle2*)

closepath()

Close the path; in case the current point is not the path's starting point, a line will be drawn between them.

endpath(*draw=True*)

drawpath(*path*)

autoclosepath(*close=True*)

findpath(*points*, *curvature=1.0*)

6.10.3 Images

image(*path*, *x=0*, *y=0*, *width=None*, *height=None*, *alpha=1.0*, *data=None*, *draw=True*)

Place a bitmap image on the canvas.

Parameters

- **path** (*str*) – location of the image on disk
- **x** (*float*) – x-coordinate of the top left corner
- **y** (*float*) – y-coordinate of the top left corner
- **width** (*float or None*) – image width (leave blank to use its original width)
- **height** (*float or None*) – image height (leave blank to use its original height)
- **alpha** (*float*) – opacity
- **data** (*binary data*) – image data to load. Use this instead of `path` if you want to load an image from memory or have another source (e.g. using the *web* library)
- **draw** (*bool*) – whether to place the image immediately on the canvas or not

6.10.4 Clipping paths

beginclip(*path*)

endclip()

6.10.5 Transforms

transform(*mode=None*)

Parameters **mode** (*CORNER or CENTER*) – the mode to base new transformations on

translate(*xt*, *yt*, *mode=None*)

rotate(*degrees=0*, *radians=0*)

scale(*x=1*, *y=None*)

skew(*x=1*, *y=0*)

push()

pop()

reset()

6.10.6 Colors

Colors can be specified in a few ways:

- grayscale: `(value)`
- grayscale with alpha: `(value, alpha)`
- RGB: `(red, green, blue)`
- RGBA: `(red, green, blue, alpha)`
- hex: `('#FFFFFF')`
- hex with alpha: `('#FFFFFFFF')`

You can use any of these formats to specify a colour; for example, *fill(1,0,0)* and *fill('#FF0000')* yield the same result.

`background`(**args*)

Set background to any valid color

`outputmode`()

Not implemented yet (Nodebox API)

`colormode`(*mode=None*, *crange=None*)

Set the current colormode (can be RGB or HSB) and eventually the color range.

Parameters

- **`mode`** (*`RGB or HSB`*) – Color mode to use
- **`crange`** – Maximum value for the new color range to use. See *colorrange*.

Return type Current color mode (if called without arguments)

`colorrange`(*crange=1.0*)

Set the numeric range for color values. By default colors range from 0.0 - 1.0; use this to set a different range, e.g. with `colorrange(255)` values will range between 0 and 255.

Parameters **`crange`** (*`float`*) – Maximum value for the new color range to use

`fill`(**args*)

Sets a fill color, applying it to new paths.

Parameters **`args`** – color in supported format

`stroke`(**args*)

Set a stroke color, applying it to new paths.

Parameters **`args`** – color in supported format

`nofill`()

Stop applying fills to new paths.

`nostroke`()

Stop applying strokes to new paths.

`strokewidth`(*w=None*)

Parameters **`w`** – Stroke width

Return type Current width (if no width was specified)

`color`(**args*)

Parameters **`args`** – color in a supported format

Return type Color object

6.10.7 Text

`text` (*txt*, *x*, *y*, *width=None*, *height=1000000*, *outline=False*, *draw=True*)

Draws a string of text according to current font settings.

Parameters

- **`txt`** – Text to output
- **`x`** – x-coordinate of the top left corner
- **`y`** – y-coordinate of the top left corner
- **`width`** – text box width. When set, text will wrap to the next line if it would exceed this width. If unset, there will be no line breaks.
- **`height`** – text box height
- **`outline`** (*`bool`*) – whether to draw as an outline.
- **`draw`** (*`bool`*) – if False, the object won't be immediately drawn to canvas.

Return type BezierPath object representing the text

`font` (*fontpath=None*, *fontsize=None*)

Set the font to be used with new text instances.

Accepts TrueType and OpenType files. Depends on FreeType being installed.

Parameters

- **`fontpath`** – path to TrueType or OpenType font
- **`fontsize`** – font size in points

Return type current font path (if `fontpath` was not set)

`fontsize` (*fontsize=None*)

Set or return size of current font.

Parameters **`fontsize`** – Font size in points (pt)

Return type Font size in points (if `fontsize` was not specified)

`textpath` (*txt*, *x*, *y*, *width=None*, *height=1000000*, *draw=False*)

Generates an outlined path of the input text.

Parameters

- **`txt`** – Text to output
- **`x`** – x-coordinate of the top left corner
- **`y`** – y-coordinate of the top left corner
- **`width`** – text width
- **`height`** – text height
- **`draw`** – Set to False to inhibit immediate drawing (defaults to False)

Return type Path object representing the text.

`textmetrics` (*txt*, *width=None*, *height=None*)

Return type the width and height of a string of text as a tuple (according to current font settings)

textwidth(*txt*, *width=None*)

> **Parameters** **text** – the text to test for its dimensions
>
> **Return type** the width of a string of text according to the current font settings

textheight(*txt*, *width=None*)

> **Parameters** **text** – the text to test for its dimensions
>
> **Return type** the height of a string of text according to the current font settings

lineheight(*height=None*)
Set the space between lines of text.

> **Parameters** **height** – line height

align(*align=LEFT*)
Set the way lines of text align with each other.

> **Parameters** **align** (`LEFT, CENTER or RIGHT`) – Text alignment rule

fontoptions(*hintstyle=None*, *hintmetrics=None*, *subpixelorder=None*, *antialias=None*)
Not implemented.

6.10.8 Dynamic variables

var(*name*, *type*, *default=None*, *min=0*, *max=255*, *value=None*, *step=None*, *steps=256.0*)
Create a live variable.

> **Parameters**
>
> - **name** – Variable name
> - **type** (`NUMBER, TEXT, BOOLEAN or BUTTON`) – Variable type
> - **default** – Default value
> - **min** – Minimum value (NUMBER only)
> - **max** – Maximum value (NUMBER only)
> - **value** – Initial value (if not defined, use `default`)
> - **step** – Step length for the variables GUI (use this or `steps`, not both)
> - **steps** – Number of steps in the variables GUI (use this or `step`, not both)

6.10.9 Utility functions

random(*v1=None*, *v2=None*)

grid(*cols*, *rows*, *colSize=1*, *rowSize=1*, *shuffled=False*)

files(*path="*"*)
You can use wildcards to specify which files to pick, e.g. `f = files('*.gif')`

> **Parameters** **path** – wildcard to use in file list

autotext(*sourceFile*)
Generates mock philosophy based on a context-free grammar.

> **Parameters** **sourcefile** – file path to use as source
>
> **Return type** the generated text

`snapshot` (*filename=None*, *surface=None*, *defer=None*, *autonumber=False*)
Save the contents of current surface into a file or cairo surface/context.

Parameters

- **`filename`** – File name to output to. The file type will be deduced from the extension.
- **`surface`** – If specified will output snapshot to the supplied cairo surface.
- **`defer`** (*`boolean`*) – Decides whether the action needs to happen now or can happen later. When set to False, it ensures that a file is written before returning, but can hamper performance. Usually you won't want to do this. For files defer defaults to True, and for Surfaces to False, this means writing files won't stop execution, while the surface will be ready when snapshot returns. The drawqueue will have to stop and render everything up until this point.
- **`autonumber`** (*`boolean`*) – If true then a number will be appended to the filename.

6.10.10 Core

`ximport` (*libName*)
Import nodebox libraries.

The libraries get _ctx, which provides them with the nodebox API.

Parameters **`libName`** – Library name to import

`size` (*w=None*, *h=None*)
Sets the size of the canvas, and creates a Cairo surface and context. Only the first call will actually be effective.

`speed` (*framerate*)
Set the framerate on windowed mode.

Parameters **`framerate`** – Frames per second

Return type Current framerate

`run` (*inputcode*, *iterations=None*, *run_forever=False*, *frame_limiter=False*)
Executes the contents of a Nodebox or Shoebot script in the current surface's context.

6.11 Command-line options

This is a list of all the flags you can specify to the `sbot` command-line runner.

`--outputfile <FILENAME>, -o <FILENAME>`
Run in headless mode, outputting the result directly to a file.

Supported extensions are `.png`, `.svg`, `.pdf` and `.ps`.

`--socketserver, -s`
Run a socket server for external variable control.

`--serverport <PORT>, -s <PORT>`
Set the socket server port to listen for connections (default is 7777).

`--vars <VARS>, -v <VARS>`
Initial variables, in JSON format.

Use single quotes **outside**, and double quotes **inside**, e.g.: `--vars='{"variable1": 1}'`

`--namespace <NAMESPACE>, -ns <NAMESPACE>`
Initial namespace, in JSON format.

Use single quotes **outside**, and double quotes **inside**, e.g.: `--namespace='{"variable1": 1}'`

`--args <ARGS>, -a <ARGS>`
Pass arguments to bot. [TODO: explain better]

`--l, -l`
Open a shell to control the bot as it runs. See the *Shell mode* documentation for the available commands.

`--repeat <TIMES>, -r <TIMES>`
Set number of iterations to run the script, producing multiple images.

`--grammar <LANG>, -g <LANG>`
Select the language to use: `nodebox` (default) or `drawbot`.

Note that Drawbot support is experimental.

`--window, -w`
Run the script in a GTK window (default).

`--fullscreen, -f`
Run the script in fullscreen mode.

`--title <TITLE>, -t <TITLE>`
Set the window title.

`--close, -c`
Close the window after running the script. Use with `--repeat` for benchmarking.

`--disable-vars, -dv`
Disable the variables pane when in windowed mode.

`--disable-background-thread, -dt`
Don't run code in a background thread.

`--verbose, -V`
Show internal error information in tracebacks.

6.12 Compatibility with Nodebox

Shoebot was originally developed as a rewrite of Nodebox, attempting to follow its behaviour as close as possible. However, the developers eventually wanted some functionality that was not in Nodebox, and there are many aspects of Nodebox that, for one reason or other, were not ported over.

Now that the original Nodebox isn't being developed further, we have decided to go on and keep implementing original features whenever appropriate.

In this page, you'll find the features and behavior that differs between Nodebox and Shoebot.

If you find any difference that isn't documented here, please file an issue.

6.12.1 Unsupported Nodebox features

These are the Nodebox bits that aren't available in Shoebot.

CMYK color

Nodebox was implemented using Mac OS X's Cocoa toolkit, which supports CMYK color. Shoebot runs on the Cairo graphics backend, which does not. Nodebox commands that deal with CMYK color are therefore unsupported:

- outputmode() is not available.
- colormode() is implemented, but `CMYK` is not an accepted argument – only `RGB` or `HSB`.

Path operations

Operations between bézier paths are not supported yet. These are:

- `path.union()`
- `path.intersect()`
- `path.difference()`

Fitting a path using `path.fit()` isn't supported either.

Animation

While animations work well in a window, Shoebot does not support exporting them to GIF or video formats.

Unported libraries

- Knowledge: WordNet, Keywords, Linguistics
- Bitmap: Core Image, iSight, Quicktime
- Systems: Ants, Noise
- Design: Grid
- Typography: Pixie, Fatpath
- Tangible: WiiNode, OSC
- Other: Flowerewolf, twyg

6.12.2 Additional Shoebot features

These are features that were created in Shoebot and are not available in Nodebox, or where Shoebot behavior is distinct.

New functions

- Drawing
  - *`rectmode()`* and *`ellipsemode()`* are inspired by Processing and cater to the preferences of users who expect alternative ways to draw primitives. These are also useful for specific situations where coordinate calculation would be needed.
  - Relative path commands are available: *`relmoveto()`* and *`rellineto()`*.
  - *`arc()`* is provided by Cairo, so we support it as well.

- Others
  - `snapshot()` is a simple way to output to an image in the middle of the script execution.
  - `run()` can execute another script in the current context. This is mostly useful when using Shoebot as a Python module inside another application.

Bundled libraries

Nodebox provides a set of external libraries that can be downloaded and added to a project. Shoebot comes with ported versions of those libraries already included and available.

- Knowledge: Database, Graph, Web
- Bitmap: Photobot
- Paths: Bezier, Cornu, SVG Supershape, Bezier Editor
- Systems: Boids, L-system
- Design: Colors
- Tangible: TUIO

New libraries

Additional external libraries were developed for Shoebot:

- *Audio*
- *Video*
- *OpenCV*

6.13 Contributing

6.13.1 Development tasks

Make new examples or port existing ones

We're always eager to welcome new examples that explain a concept or show off an interesting technique.

You can either contribute your own examples, or help port existing scripts:

- the nodebox-pyobjc examples, which are more current than those in the old Nodebox 1 repository
- the scripts in the Nodebox gallery

They should work mostly without modifications – we need help testing them. Try them out and post any issues you find on our issue tracker in case you hit a wall.

Be sure to also check the brief guidelines in *Coding style for examples* so that your efforts can be included in Shoebot.

Help port libraries

We're missing a few Nodebox libraries; can you help us port them to Shoebot?

See the full list of *Unported libraries*.

Incidentally, we're also missing documentation to explain how to port Nodebox libraries. If you're interested but stuck, file an issue and we'll help you.

Look for 'Help Out' issues

The issues tagged 'Help Out' don't need a deep knowledge of Shoebot internals, and there's a good variety of tasks to be done.

Make text editor plugins

While our simple editor is around, power-users will be using their favourite text editor to hack on Shoebot scripts. Having plugins for any popular text editor would be a fantastic addition.

Integrate Shoebot with other software

Shoebot can be a great tool to complement other software, be it for

- SVG, PDF or bitmap generation
- simple visualizations
- interact in real-time with the socket server

If you see a use case where Shoebot could be helpful, we'll be more than happy to support you in implementing it.

6.13.2 Non-development tasks

Find bugs in our documentation and fix them

We're missing many details and we'd definitely welcome some help here. While actual contributions to the documentation would be the best, we'd be more than happy with pointing out the parts that are missing or plain wrong. Use the `documentation` label on the issue tracker to help us on this.

6.13.3 Tips for Developers

Coding style for the Shoebot core code

We're not picky here, other than following PEP8 style guidelines. We use flake8 extensions in our code editors to keep us strict, and recommend it.

Coding style for examples

When creating examples for including in Shoebot, we try to adhere to a set of writing guidelines to make it easy for newcomers to understand what's going on.

- Do not use one-letter variables (other than `x` and `y`), and avoid two-letter names as well (things like `dx` can be expanded to `deltax`). It will look less compact, but really helps understanding what's going on.

- Start the example with a docstring specifying the title of the example, author info and some details about the script and its workings. If you want to format this text, use Markdown.
- Use Flake8 or similar linter plugin to find necessary style fixes.
- Comments in English.
- Variables and functions are in `lowercase` and `underscored_lowercase`, class names are in `CamelCase`.

Making a release

This is our checklist to be sure we don't miss any detail when we put out a release.

- update the version number in these files:
  - `Makefile`
  - `VERSION`
  - `setup.py`
  - `doc/source/conf.py`
  - `shoebot/ide/ide.py`
- update the changelogs
  - `CHANGELOG`
  - `debian/changelog`
- tag the release commit
- publish release on GitHub
- push to PyPI
  - register on PyPI and place your credentials in `~/.pypirc`
  - install Twine
  - make a source build with `python setup.py sdist`
  - make a test upload to TestPyPI with `twine upload --repository-url https://test.pypi.org/legacy/ dist/shoebot-1.3.tar.gz`
  - if all is good, upload to PyPI with `twine upload dist/shoebot-1.3.tar.gz`
  - be sure to change the version numbers in the previous commands according to the current Shoebot version

Building Debian packages

There are some dependencies to look out for:

```
sudo apt-get install rename dh-python cdbs
```

Be sure to go through this checklist:

- update the debian/changelog file

Then, generate the Debian packages with the *make builddeb* command.

Index