
nimbusinator

Release 0.2.2

Mar 13, 2020

Contents:

1	About	3
2	Usage	5
3	Installation	7
3.1	Ubuntu 19.04	7
4	Quick-start	9
5	Links	11
5.1	nimbusinator.nimbus.Nimbus	11
5.2	nimbusinator.command.Command	12
5.3	Release Notes	17
6	Indices and tables	21
	Python Module Index	23
	Index	25

RM Nimbus GUI for Python

CHAPTER 1

About

Nimbusinator is a graphical user-interface package for Python that mimicks the graphics and text drivers of the RM Nimbus PC186. It is *not* an emulation of the Nimbus itself. This means you get the best of both worlds: Cutting-edge Python computing power, beautifully presented in up to 16 colours!

Disclaimer: Nimbusinator is a tribute project and is in no way linked to or endorsed by RM plc.

CHAPTER 2

Usage

To implement a Nimbus user interface all you need to do is import the *Nimbus* and *Command* classes, like this:

```
from nimbusinator.nimbus import Nimbus
from nimbusinator.command import Command
```

Then create one object of each, and bind the *Command* object to the *Nimbus* object:

```
nim = Nimbus()
cmd = Command(nim)
```

To display the screen, call the *boot* method on the *Nimbus* object. By default, you'll see the famous blue "Welcome Screen" before control is released back to your program. To skip the Welcome Screen simply pass the argument *skip_welcome_screen=True* when calling *boot*, like this:

```
# Boot the Nimbus with Welcome Screen
nim.boot()
# Boot the Nimbus without Welcome Screen
nim.boot(skip_welcome_screen=True)
```

Note that the original Welcome Screen reported the memory status of the computer (in kilobytes!), alongside the RM firmware version and machine serial number. Nimbusinator also displays memory status (but in units of Megabytes) and uses your Python version as the firmware version and your OS release number as the serial number. It also uses information from your runtime environment to simulate a short DOS-like boot sequence before finally releasing control back to your application. All the while you can enjoy the dulcet tones of an imaginary floppy drive.

To bring the Nimbus programming experience into the 21st century, the API - although Pythonic - has been modelled on the syntax of RM Basic. For example, in RM Basic to write a greeting in big, red letters in the bottom-left corner of the screen you would use the *PLOT* command:

```
PLOT "Hi kittens!", 10, 10 BRUSH 2 SIZE 4
```

And in Python with Nimbusinator you can write the same instruction like this:

```
cmd.plot('Hi kittens!', (10, 10), brush=2, size=4)
```

It is recommended to read the RM Basic manual to get familiar with the original commands and how graphics and text were handled on the Nimbus (see links below).

To cleanly exit your application, call the *shutdown* method on the *Nimbus* object:

```
# Always do this before your app quits:  
nim.shutdown()
```

CHAPTER 3

Installation

Nimbusinator is technically cross-platform but there are differences in the way PyGame's dependencies are installed between operating systems. So far I have only tested installation on Ubuntu 19.04 (see below). If you have tested PyGame successfully on other platforms please drop the magic formula in [the issues](#) and I'll add it below.

3.1 Ubuntu 19.04

```
# Install dependencies for pygame and simpleaudio:
sudo apt-get install -y python3-dev libasound2-dev python3-setuptools python3-numpy
↳python3-opengl libsdl-image1.2-dev libsdl-mixer1.2-dev libsdl-ttf2.0-dev libsmpeg-
↳dev libsdl1.2-dev libportmidi-dev libswscale-dev libavformat-dev libavcodec-dev
↳libtiff5-dev libx11-6 libx11-dev fluid-soundfont-gm timgm6mb-soundfont xfonts-base
↳xfonts-100dpi xfonts-75dpi xfonts-cyrillic fontconfig fonts-freefont-ttf
↳libfreetype6-dev

# Then activate your Python env and install:
pip install nimbusinator
```


CHAPTER 4

Quick-start

```
from nimbusinator.nimbus import Nimbus
from nimbusinator.command import Command

if __name__ == '__main__':
    # Create and bind nimbusinator objects:
    nim = Nimbus()
    cmd = Command(nim)
    nim.boot()           # Boot the Nimbus
    cmd.set_mode(40)     # Low resolution mode
    cmd.set_border(1)    # Dark blue border
    cmd.set_paper(9)     # Light blue paper
    cmd.cls()            # Clear screen
    cmd.plonk_logo((8, 110)) # Show Nimbus logo
    # Display a message in cyan with shadowing
    cmd.plot('Greetings!!!', (65, 155), size=2, brush=0)
    cmd.plot('Greetings!!!', (66, 156), size=2, brush=13)
    # Wait 5 seconds then shutdown
    nim.sleep(5)
    nim.shutdown()
```


- [Github](#) - Nimbusinator github repository
- [facebook](#) - RM Nimbus facebook group
- [Center for Computing History](#) - original RM Nimbus manuals and technical data
- [Center for Computing History](#) - RM Nimbus PC (Later Beige Model) - online exhibit
- [The Nimbus Museum](#) - online museum that looks like the Welcome Disk!
- [RM Nimbus](#) - Wikipedia article
- [mame](#) - comprehensive retro computer emulation project
- [Freesound pack: Floppy disk drive](#) - source of the floppy drive sounds
- [Ironstone Innovation](#) - what I do for a living

5.1 nimbusinator.nimbus.Nimbus

```
class nimbusinator.nimbus.Nimbus (full_screen=False, title='Nimbusinator', border_size=40,  
                                  silent=False)
```

Nimbus video display class.

This class represents the Nimbus video display that will host the user interface for your application. When created, the new Nimbus object will not be visible until the `boot()` method has been called.

Parameters

- **full_screen** (*bool, optional*) – Full screen mode
- **title** (*str, optional*) – The title of the display window
- **border_size** (*int, optional*) – The thickness of the border in pixels (default is 40)
- **silent** (*bool, optional*) – Run Nimbusinator in silent mode (default is False)

boot (*skip_welcome_screen=False*)

Boot the Nimbus

This will reveal the display screen and start all the Nimbus-related processes. By default the Nimbus will go through the Welcome Screen and a mock boot-up sequence before returning control to your app. You can bypass the Welcome Screen by passing `skip_welcome_screen=True`. Once `boot()` has been called, the Nimbus can be interrupted and stopped programmatically by calling the `shutdown()` method, or during runtime by the user pressing CTRL-C.

Parameters `skip_welcome_screen` (*bool, optional*) – Bypass the Welcome Screen and boot sequence

empty_paper ()

Return empty paper filled with the current paper colour

plonk_image_on_paper (*img, coord, transparent=False*)

Plonk a PIL image somewhere on the paper

The passed coordinates are consistent with RM Nimbus implemented, i.e. bottom-left of screen = (0, 0) and any images are located relative to their bottom-left corner

Parameters

- **img** (*PIL image*) – The image to be plonked
- **coord** (*tuple*) – The coordinate tuple (x, y)
- **transparent** (*bool, optional*) – True if image contains an alpha layer for transparency

run_floppy (*flag*)

Run or stop the floppy drive sound effects

Augment your user experience with the industrial melodies of a 1980s PC floppy drive

Parameters **flag** (*boolean*) – True to run the drive, False to stop

shutdown ()

Shut down the Nimbus

Stops all commands from executing, stops all Nimbus-related processes, closes the display window and halts program execution.

sleep (*sleep_time*)

Pause execution like `time.sleep()`

Unlike `time.sleep()`, this built-in sleep method will be interrupted if the user hits CTRL-C. Sleep time measured in seconds.

5.2 nimbusinator.command.Command

class `nimbusinator.command.Command` (*nimbus*)

Nimbus commands

This class contains the commands to control the Nimbus display. The commands mimic the syntax found in RM Basic but re-written Pythonically. In creating an object of this class it must be bound to a pre-existing Nimbus object. It is highly recommended to read the original Nimbus manuals (particularly RM Basic) to get a deeper understanding of how these commands originally worked.

Parameters **nimbus** (*Nimbus*) – The Nimbus object to bind to

area (*coord_list*, *brush=None*, *scale=1*)

Draw a filled polygon

Parameters

- **coord_list** (*list*) – A list of (x, y) tuples
- **brush** (*int*, *optional*) – Colour value (High-resolution: 0-3, low-resolution: 0-15)
- **scale** (*int*, *optional*) – Scale factor. To elongate pass a tuple (x_size, y_size)

ask_brush ()

Return the current brush colour

Returns int

ask_charset ()

Return the current charset for text

0 is the standard font, 1 is the other font

Returns int

ask_curpos ()

Gets the current cursor position as column index, row index

Returns (int, int)

ask_paper ()

Return the current paper colour

Returns int

ask_pen ()

Return the current pen colour

Returns int

ask_plot_font ()

Return the current plot_font

0 is the standard font, 1 is the other font

Returns int

ask_points_style ()

Gets the current points style

Returns int

circle (*radius*, *coord_list*, *brush=None*)

Draw one or more circles

Parameters

- **radius** (*int*) – The radius of the circle
- **coord_list** (*list*) – A list of (x, y) tuples
- **brush** (*int*, *optional*) – Colour value (High-resolution: 0-3, low-resolution: 0-15)

cls ()

Clear the screen of all text and graphics and reset cursor position

fetch (*block_number*, *filename*)

Fetch an image from disk and allocate it to an image block

Equivalent to FETCH in the ANIMATE extension. All major image formats are supported but in this loading images with alpha layers may break. It is also advisable to resize your image to fit the Nimbus screen mode using an image editor beforehand.

Parameters

- **block_number** (*int*) – The block number to store the image in
- **filename** (*str*) – The filename of the image to fetch

flush()

Clears the keyboard buffer

gets()

Get the oldest char in the keyboard buffer

Equivalent to GET\$ in RM Basic except it will also return the following values if a control key is pressed: ‘__F1__’, ‘__F2__’, ... , ‘__F10__’, ‘__CTRL_L__’, ‘__CTRL_R__’, ‘__ENTER__’, ‘__BACKSPACE__’, ‘__ESC__’, ‘__DELETE__’, ‘__HOME__’, ‘__PAGE_UP__’, ‘__END__’, ‘__PAGE_DOWN__’, ‘__INSERT__’, ‘__UP__’, ‘__DOWN__’, ‘__LEFT__’, ‘__RIGHT__’, ‘__TAB__’

Returns str**input** (*prompt*)

Collects keyboard input until user presses ENTER then returns the input as a string

Parameters **prompt** (*str*) – Message to be printed

Returns str**line** (*coord_list*, *brush=None*)

Draw one or more connected straight lines

Parameters

- **coord_list** (*list*) – A list of (x, y) tuples
- **brush** (*int*, *optional*) – Colour value (High-resolution: 0-3, low-resolution: 0-15)

plonk_logo (*coord*)

Plonk the RM Nimbus logo on screen

Parameters **coord** (*tuple*) – The (x, y) position to plonk the logo

plot (*text*, *coord*, *size=1*, *brush=None*, *direction=0*, *font=None*)

Plot text on the screen

Parameters

- **text** (*str*) – The text to be plotted
- **coord** (*tuple*) – The (x, y) position of the text
- **size** (*int*, *optional*) – Font size. To elongate pass a tuple (x_size, y_size)
- **brush** (*int*, *optional*) – Brush colour
- **direction** (*int*, *optional*) – 0=normal, 1=-90deg, 2=180deg, 3=-270deg
- **font** (*int*, *optional*) – 0 is the standard font, 1 is the other font

points (*coord_list*, *brush=None*, *size=1*, *style=None*)

Draw points on the screen

Parameters

- **coord_list** (*list*) –
- **coord_list** – A list of (x, y) tuples
- **brush** (*int*, *optional*) – Colour value (High-resolution: 0-3, low-resolution: 0-15)
- **size** (*int*, *optional*) – Font size. To elongate pass a tuple (x_size, y_size)
- **style** (*int*, *optional*) – The points style number (0 255)

print (*text*)

Print a string with carriage return at end

Parameters text (*str*) – The text to be printed

put (*ascii_data*)

Put a single character or string at the current cursor position

Parameters ascii_data (*int/str*) – If an int is passed the corresponding ASCII character will be plotted. If a string is passed then the string will be printed without a terminating carriage return.

set_border (*colour*)

Set the border colour

Parameters colour (*int*) – Colour value (High-resolution: 0-3, low-resolution: 0-15)

set_brush (*colour*)

Set the brush colour

Parameters colour (*int*) – Colour value (High-resolution: 0-3, low-resolution: 0-15)

set_charset (*charset*)

Set the charset for text

Parameters charset (*int*) – 0 is the standard font, 1 is the other font!

set_colour (*colour1*, *colour2*)

Set a colour to a new colour

Parameters

- **colour1** (*int*) – The colour code to be changed
- **colour2** (*int*) – The new colour to be assigned to colour1

set_curpos (*cursor_position*)

Set the cursor position

Parameters cursor_position (*tuple*) – The new cursor position (column, row)

set_cursor (*flag*)

Show or hide cursor

Parameters flag (*boolean*) – True to show cursor, False to hide

set_mode (*columns*)

Select either high-resolution or low-resolution screen mode

In RM Basic the screen resolution was set by the number of columns: 40 for low-resolution and 80 for high-resolution. Any other values had no effect. Nimbusinator is more strict and will yield an error if any other values are entered. Check the original RM Basic manual for a description of how screen resolutions worked on the Nimbus.

MODE 40: 40 columns, 25 rows; 320 pixels wide, 250 pixels high; 16 colours

MODE 80: 80 columns, 25 rows; 640 pixels wide, 250 pixels high (but doubled along vertical axis); 4 colours

Parameters **columns** (*int*) – The number of columns (40 or 80)

set_paper (*colour*)

Set the paper colour

Parameters **colour** (*int*) – Colour value (High-resolution: 0-3, low-resolution: 0-15)

set_pen (*colour*)

Set the pen colour

Parameters **colour** (*int*) – Colour value (High-resolution: 0-3, low-resolution: 0-15)

set_plot_font (*plot_font*)

Set the plot font

Parameters **plot_font** (*int*) – 0 is the standard font, 1 is the other font!

set_points_style (*points_style*, *points_list*)

Set a points style to that define in a points list

You can define up to 256 points styles (0 - 255). The points list is similar to the way patterns are define in RM Basic. Each row in the points style is define by a string, with full-stop chars (.) indicating a filled pixel, and spaces () indicating a filled pixel. The size of a points style is 8x8 pixels.

Parameters

- **points_style** (*int*) – The points style number (0 255)
- **points_list** (*list*) – The points list as defined above

slice_ (*radius*, *from_angle*, *to_angle*, *coord_list*, *brush=None*)

Draw one or more pie slices

Parameters

- **radius** (*int*) – The radius of the slice
- **from_angle** (*int*) – The starting angle (degrees)
- **to_angle** (*int*) – The finishing angle (degrees)
- **coord_list** (*list*) – A list of (x, y) tuples
- **brush** (*int*, *optional*) – Colour value (High-resolution: 0-3, low-resolution: 0-15)

writeblock (*block_number*, *coord*)

Write an image block to the screen

Equivalent to WRITEBLOCK in the ANIMATE extension. Alpha layers are not yet supported - see documentation for fetch command.

Parameters

- **block_number** (*int*) – The block number of the image block
- **coord** (*tuple*) – The (x, y) position of the image

5.3 Release Notes

5.3.1 0.2.2

Better handling of non-Roman characters, much faster fetch(), and you can now run Nimbusinator in silent mode by passing silent=True when instantiating a Nimbus object.

5.3.2 0.2.1

More graphics stuff including import and down-conversion of full-colour images, plus control key detection in gets().

Commands updated:

- slice has been renamed to **slice_** to avoid stomping on Python keyword
- area now has a scale optional parameter
- gets now detects control key presses

Commands added:

- set_points_style(points_style, points_list)
- ask_points_style()
- points(coord_list, brush=None, size=None, style=None)
- fetch(block_number, filename)
- writeblock(block_number, coord)

Bug fixes:

- y-axis still had a bug but now fixed
- some input validation on optional parameters would break for default values

5.3.3 0.1.0

Removed dependency on OpenCV, fixed bugs, crisper rendering and added 8 new commands.

Commands added:

- circle(radius, coord_list, brush=None)
- slice(radius, from_angle, to_angle, coord_list, brush=None)
- set_plot_font(plot_font)
- ask_plot_font()
- ask_paper()
- ask_pen()
- ask_brush()
- ask_charset()

Bug fixes:

- No longer draws everything one pixel higher
- More precise and faster rendering

- Cursor now matches pen colour
- Fixed wonky font images
- Fixed RM Nimbus logo image
- Fixed intermittent floppy drive
- Shutdown by CTRL-C interrupt doesn't get delayed

5.3.4 0.0.2

Revised package structure to solve module import errors with Sphinx.

Nimbus and Command classes are now imported like this:

```
from nimbusinator.nimbus import Nimbus
from nimbusinator.command import Command
```

5.3.5 0.0.1

First public release!

Key features:

- High-resolution (640x250) 4 colour mode
- Low-resolution (320x250) 16 colour mode
- Welcome Screen and DOS boot sequence
- Full screen mode
- Nimbus sleep method
- Keyboard input with cursor
- Nimbus colours
- Both Nimbus charsets

Commands added:

- `area(coord_list, brush=None)`
- `ask_charset()`
- `ask_curpos()`
- `cursor_position (tuple)`
- `cls()`
- `flush()`
- `gets()`
- `input(prompt)`
- `line(coord_list, brush=None)`
- `plonk_logo(coord)`
- `plot(text, coord, size=1, brush=None, direction=0, font=None)`
- `print(text)`

- `put(ascii_data)`
- `set_border(colour)`
- `set_brush(colour)`
- `set_charset(charset)`
- `set_colour(colour1, colour2)`
- `set_curpos(cursor_position)`
- `set_cursor(flag)`
- `set_mode(columns)`
- `set_paper(colour)`
- `set_pen(colour)`

CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`

n

`nimbusinator.command`, [12](#)
`nimbusinator.nimbus`, [11](#)

A

`area()` (*nimbusinator.command.Command method*), 12
`ask_brush()` (*nimbusinator.command.Command method*), 13
`ask_charset()` (*nimbusinator.command.Command method*), 13
`ask_curpos()` (*nimbusinator.command.Command method*), 13
`ask_paper()` (*nimbusinator.command.Command method*), 13
`ask_pen()` (*nimbusinator.command.Command method*), 13
`ask_plot_font()` (*nimbusinator.command.Command method*), 13
`ask_points_style()` (*nimbusinator.command.Command method*), 13

B

`boot()` (*nimbusinator.nimbus.Nimbus method*), 11

C

`circle()` (*nimbusinator.command.Command method*), 13
`cls()` (*nimbusinator.command.Command method*), 13
`Command` (*class in nimbusinator.command*), 12

E

`empty_paper()` (*nimbusinator.nimbus.Nimbus method*), 12

F

`fetch()` (*nimbusinator.command.Command method*), 13
`flush()` (*nimbusinator.command.Command method*), 14

G

`gets()` (*nimbusinator.command.Command method*), 14

I

`input()` (*nimbusinator.command.Command method*), 14

L

`line()` (*nimbusinator.command.Command method*), 14

N

`Nimbus` (*class in nimbusinator.nimbus*), 11
`nimbusinator.command` (*module*), 12
`nimbusinator.nimbus` (*module*), 11

P

`plonk_image_on_paper()` (*nimbusinator.nimbus.Nimbus method*), 12
`plonk_logo()` (*nimbusinator.command.Command method*), 14
`plot()` (*nimbusinator.command.Command method*), 14
`points()` (*nimbusinator.command.Command method*), 14
`print()` (*nimbusinator.command.Command method*), 15
`put()` (*nimbusinator.command.Command method*), 15

R

`run_floppy()` (*nimbusinator.nimbus.Nimbus method*), 12

S

`set_border()` (*nimbusinator.command.Command method*), 15
`set_brush()` (*nimbusinator.command.Command method*), 15
`set_charset()` (*nimbusinator.command.Command method*), 15
`set_colour()` (*nimbusinator.command.Command method*), 15
`set_curpos()` (*nimbusinator.command.Command method*), 15

`set_cursor()` (*nimbusinator.command.Command method*), 15
`set_mode()` (*nimbusinator.command.Command method*), 15
`set_paper()` (*nimbusinator.command.Command method*), 16
`set_pen()` (*nimbusinator.command.Command method*), 16
`set_plot_font()` (*nimbusinator.command.Command method*), 16
`set_points_style()` (*nimbusinator.command.Command method*), 16
`shutdown()` (*nimbusinator.nimbus.Nimbus method*), 12
`sleep()` (*nimbusinator.nimbus.Nimbus method*), 12
`slice_()` (*nimbusinator.command.Command method*), 16

W

`writeblock()` (*nimbusinator.command.Command method*), 16