

---

# **Pycairo**

*Release*

**Sep 17, 2017**



---

## Contents

---

<b>1</b>	<b>1.15.3 - 2017-09-17</b>	<b>3</b>
<b>2</b>	<b>1.15.2 - 2017-09-03</b>	<b>5</b>
<b>3</b>	<b>1.15.1 - 2017-08-19</b>	<b>7</b>
<b>4</b>	<b>1.15.0 - 2017-07-24</b>	<b>9</b>
<b>5</b>	<b>1.14.1 - 2017-07-24</b>	<b>11</b>
<b>6</b>	<b>1.14.0 - 2017-07-12</b>	<b>13</b>
<b>7</b>	<b>1.13.4 - 2017-07-12</b>	<b>15</b>
<b>8</b>	<b>1.13.3 - 2017-06-01</b>	<b>17</b>
<b>9</b>	<b>1.13.2 - 2017-05-21</b>	<b>19</b>
<b>10</b>	<b>1.13.1 - 2017-05-07</b>	<b>21</b>
<b>11</b>	<b>1.13.0 - 2017-05-03</b>	<b>23</b>
<b>12</b>	<b>1.12.0 - 2017-04-18</b>	<b>25</b>
<b>13</b>	<b>1.11.1 - 2017-04-12</b>	<b>27</b>
<b>14</b>	<b>1.11.0 - 2017-04-09</b>	<b>29</b>
<b>15</b>	<b>1.10.0 - 2011-05-01</b>	<b>31</b>
<b>16</b>	<b>1.8.10 - 2010-05-20</b>	<b>33</b>
<b>17</b>	<b>1.8.8 - 2009-08-26</b>	<b>35</b>
<b>18</b>	<b>1.8.6 - 2009-06-25</b>	<b>37</b>
<b>19</b>	<b>1.8.4 - 2009-03-19</b>	<b>39</b>
<b>20</b>	<b>1.8.2 - 2009-01-15</b>	<b>41</b>

<b>21</b>	<b>1.8.0 - 2008-12-15</b>	<b>43</b>
<b>22</b>	<b>1.6.4 - 2008-08-18</b>	<b>45</b>
<b>23</b>	<b>1.4.12 - 2007-12-13</b>	<b>47</b>
<b>24</b>	<b>1.4.0 - 2007-03-14</b>	<b>49</b>
<b>25</b>	<b>1.2.6 - 2006-11-27</b>	<b>51</b>
<b>26</b>	<b>1.2.2 - 2006-08-21</b>	<b>53</b>
<b>27</b>	<b>1.2.0 - 2006-07-03</b>	<b>55</b>
<b>28</b>	<b>1.1.6 - 2006-05-29</b>	<b>57</b>
<b>29</b>	<b>1.0.2 - 2005-10-11</b>	<b>59</b>
<b>30</b>	<b>1.0.0 - 2005-08-31</b>	<b>61</b>
<b>31</b>	<b>0.9.0 - 2005-08-10</b>	<b>63</b>
<b>32</b>	<b>0.6.0 - 2005-08-01</b>	<b>65</b>
<b>33</b>	<b>0.5.1 - 2005-06-22</b>	<b>67</b>
<b>34</b>	<b>0.4.0 - 2005-03-10</b>	<b>69</b>
<b>35</b>	<b>0.1.4 - 2005-01-14</b>	<b>71</b>
<b>36</b>	<b>0.1.3 - 2004-11-24</b>	<b>73</b>
<b>37</b>	<b>An Introduction to Cairo with Python</b>	<b>75</b>
<b>38</b>	<b>API Reference</b>	<b>81</b>
<b>39</b>	<b>C API Reference</b>	<b>161</b>
<b>40</b>	<b>Resources</b>	<b>169</b>

Since version 1.11.0 Pycairo uses [Semantic Versioning](#) i.e. the newest version is the latest stable one.



# CHAPTER 1

---

1.15.3 - 2017-09-17

---

## Fixes:

- setup.py: Install pkgconfig file into /usr/share/pkgconfig to work around JHBuild on Fedora not picking it up.
- Fix tests on big endian machines. [#75](#)
- Support building with MSVC [#pr-72](#) (Chun-wei Fan)

## Tests:

- Test MSVC builds on appveyor





## CHAPTER 2

---

1.15.2 - 2017-09-03

---

### Fixes:

- setup.py: Install pkgconfig file to the same library prefix that Python uses. (/usr/lib64 instead of /usr/lib under Fedora for example) #70 #pr-71 (Sander Sweers)



## CHAPTER 3

---

1.15.1 - 2017-08-19

---

### Fixes:

- Improved support for Python filesystem paths including `os.PathLike`. See *pathlike* for details.
- Various minor fixes

### Changes:

- Expose `cairo.Path`

### Tests:

- Improved test coverage from ~70% to ~90%



## New Features:

- Add `Surface.map_to_image()` and `Surface.unmap_image()` #51
- Add `RasterSourcePattern` #48
- Add `Glyph` #53
- Add `Rectangle` #54
- Add `TextCluster` #61
- Add `ScaledFont.text_to_glyphs()` and `ScaledFont.glyph_extents()`
- Add `Context.show_text_glyphs()`
- Add `TextExtents` #62

## Changes:

- Pycairo instances wrapping the same underlying cairo object now hash and compare equally e.g. `context.get_target() == context.get_target()`
- Functions which returned a cairo error with `Status.NO_MEMORY` no longer raise `MemoryError`, but a subclass of `Error` and `MemoryError`. Similarly errors with `Status.READ_ERROR` and `Status.WRITE_ERROR` no longer raise `IOError`, but a subclass of `Error` and `IOError`. #55
- Some functions which previously returned a tuple now return a tuple subclass like `Rectangle`, `Glyph`, `TextCluster` and `TextExtents`



## CHAPTER 5

---

1.14.1 - 2017-07-24

---

### Fixes:

- Fix a crash with `Surface.get_device()` #57





# CHAPTER 6

---

1.14.0 - 2017-07-12

---

## General:

- Requires at least cairo 1.13.1 (The snapshot in Ubuntu 14.04)

## Tests:

- Optional Hypothesis tests.

## New Features:

- Add `Surface.set_device_scale()` and `Surface.get_device_scale()`. #pr-44 (Sander Sweers)
- Add `Device` #pr-45
- Add `Surface.get_device()` #pr-45
- Add `ScriptDevice` and `ScriptMode` #pr-46
- Add `ScriptSurface` #17
- Add `Status.JBIG2_GLOBAL_MISSING`
- Add `Format.stride_for_width()`
- Add `TextClusterFlags` and `SurfaceObserverMode`
- Add `Gradient.get_color_stops_rgba()`
- Add `TeeSurface`
- Add `MeshPattern`



## CHAPTER 7

---

1.13.4 - 2017-07-12

---

### Fixes:

- Fix a rare crash with `get_data()` under Python 3 (1.13.3 regression).



## CHAPTER 8

---

1.13.3 - 2017-06-01

---

### Fixes:

- Fix ImageSurface leaking in case `get_data()` is used under Python 3. #41

### Documentation:

- Add Pillow to ImageSurface example. #pr-40 (Stuart Axon)
- Describe Freetype-py intergration. #25 #pr-43 (Hin-Tak Leung)



## CHAPTER 9

---

1.13.2 - 2017-05-21

---

### Fixes:

- Fix pip failing to install pycairo in some cases. #39

### Testing:

- Added continuous testing for Windows using MSYS2 and appveyor. #19





# CHAPTER 10

---

1.13.1 - 2017-05-07

---

## Fixes:

- `setup.py` install: Fix generated `pkg-config` file if `--home` or `--user` is specified. #34
- Fix a build error on macOS Sierra. #pr-36 (Nicolas P. Rougier)
- examples: Fix snippet examples when `.pyc` files are present. #35

## Documentation:

- Add Pyglet integration example. #pr-33 (Stuart Axon)



## New Features:

- The buffer returned by *ImageSurface.get\_data()* under Python 2 now implements the character buffer interface to make it work with `pygame.image.frombuffer()`. #pr-29
- All C enum types now have their own corresponding Python enum type: *Antialias*, *Content*, *Extend*, *FillRule*, *Filter*, *FontSlant*, *FontWeight*, *Format*, *HintMetrics*, *HintStyle*, *LineCap*, *LineJoin*, *Operator*, *PDFVersion*, *PSLevel*, *PathDataType*, *RegionOverlap*, *SVGVersion*, *Status*, *SubpixelOrder*. #26

All relevant constants are now an alias to attributes of those types e.g. *ANTIALIAS\_DEFAULT* is the same as *Antialias.DEFAULT*.

All functions returning enum values now return instances of the new types e.g. *Context.get\_antialias()* returns a *Antialias*.

*Error.status* is now a *Status*.

- All included examples now work with Python 2 & 3
- All included examples using GTK+ have been ported to GTK+ 3/PyGObject 3

## Fixes:

- Fix the signature of the *ImageSurface* buffer interface for Python 2 (int -> Py\_ssize\_t)
- setup.py: Ensure “-fno-strict-aliasing” is used with Python 2.

## Testing:

- Added travis-ci tests for flake8 and sphinx. #pr-30, #pr-32
- The test suite now has optional tests for numpy and pygame integration.



# CHAPTER 12

---

1.12.0 - 2017-04-18

---

## General:

- Require cairo 1.12.0
- Use C90 and enforce it on travis-ci. #5, #fdo-22940

## Constants:

- Add various new `cairo.OPERATOR_*`, `cairo.ANTIALIAS_*` and `cairo.FORMAT_*` constants. #1
- Add `HAS_MIME_SURFACE` and `cairo.MIME_TYPE_*`. #7, #fdo-58771
- Add `cairo.PDF_VERSION_*`. #pr-16
- Add `cairo.SVG_VERSION_*`

## Error:

- Add a `Error.status` attribute exposing `cairo.STATUS_*`
- Add `CairoError` alias for `Error` for `cairoffi` compatibility

## Matrix:

- Expose matrix components as read/write properties. e.g. `Matrix.xx`
- Fix type checking of the multiplication operator under Python 3. #8, #fdo-89162 (Lawrence D'Oliveiro)

## Surface:

- Add `Surface.set_mime_data()`. #7, #fdo-58771
- Add `Surface.get_mime_data()`. #7, #fdo-58771
- Add `Surface.supports_mime_type()`. #7, #fdo-58771
- Add `Surface.create_for_rectangle()`. #pr-13
- Add `Surface.create_similar_image()`. #pr-15
- Add `Surface.has_show_text_glyphs()`
- Fix crash when the surface wrapper gets deallocated before the surface object. #11

### **Context:**

- Add `Context.in_clip()`. #pr-14

### **PDFSurface:**

- Add `PDFSurface.restrict_to_version()`. #pr-16
- Add `PDFSurface.get_versions()`. #pr-16
- Add `PDFSurface.version_to_string()`. #pr-16

### **SVGSurface:**

- Add `SVGSurface.restrict_to_version()`
- Add `SVGSurface.get_versions()`
- Add `SVGSurface.version_to_string()`

### **XCBSurface:**

- Add `XCBSurface.set_size()`

### **PSSurface:**

- Add `PSSurface.get_levels()`
- Add `PSSurface.level_to_string()`

### **Pattern:**

- Add `Pattern.set_filter()`
- Add `Pattern.get_filter()`

### **RecordingSurface:**

- Add `RecordingSurface.get_extents()`

### **FontOptions:**

- Implement `__eq__` and `__ne__`
- Add `FontOptions.copy()`
- Add `FontOptions.hash()`
- Add `FontOptions.equal()`
- Add `FontOptions.merge()`

### **ScaledFont:**

- Add `ScaledFont.get_ctm()`
- Add `ScaledFont.get_font_matrix()`
- Add `ScaledFont.get_font_options()`

## CHAPTER 13

---

1.11.1 - 2017-04-12

---

This release fixes an ABI breakage. I missed that the original pycairo master had already broken ABI compared to 1.10.0.





# CHAPTER 14

---

1.11.0 - 2017-04-09

---

This version is based on the Python 2 version of pycairo 1.10.0 and is API/ABI compatible with both py2cairo 1.10.0 and py3cairo 1.10.0.

## General Changes:

- Requires cairo 1.10.2+
- Switch to semantic versioning
- Switch build system to distutils/setup.py (xpyb integration can be enabled with passing `--enable-xpyb` to setup.py build)
- Moved to GitHub: <https://github.com/pygobject/pycairo>

## New Features:

- Python 3 support (API/ABI compatible with py3cairo 1.10.0) including support for `cairo.Error`, `cairo.ImageSurface.get_data()` and `cairo.ImageSurface.create_for_data()`, which were missing in py3cairo.
- `cairo.RecordingSurface` (#fdo-36854, Torsten Landschoff)
- `cairo.Region`, `cairo.RectangleInt` and `cairo.REGION_OVERLAP_*` (#fdo-44336, Bug Fly)

## Bug Fixes:

- Fix crash when read()/write() methods of file objects passed to pycairo raise exceptions.
- Fix possible value truncation of handles passed to Win32Surface and Win32PrintingSurface on 64bit Windows. #fdo-57493



# CHAPTER 15

---

1.10.0 - 2011-05-01

---

**General Changes:** py2cairo 1.10.0 requires cairo 1.10.0 (or later).

**New Constants:** cairo.FORMAT\_RGB16\_565

**Bug Fixes:**

- context.get\_source().get\_surface() fails #fdo-33013
- Add support for './waf configure --libdir=XXX' #fdo-30230

**Documentation Changes:**

- Upgrade to using Sphinx 1.0.7.
- Include html documentation in the pycairo archive file.

**Build Changes:**

- Update waf to 1.6.3
- Remove setup.py

**Other Changes:**

- Improve/simplify unicode filename support.
- Improve/simplify unicode text support.



# CHAPTER 16

---

1.8.10 - 2010-05-20

---

**General Changes:** Pycairo 1.8.10 requires cairo 1.8.10 (or later).

**New Classes/Types:**

- Win32PrintingSurface
- XCBSurface - add XCB support using xpyb

**Bug Fixes:**

- Fix for libtool 2.2 ([#fdo-27974](#)).
- Mingw32 and pypy fixes ([#fdo-25203](#)).

**Other Changes:** Tests updated.

The Win32PrintingSurface and XCBSurface changes mean that pycairo 1.8.10 is not binary compatible with pycairo 1.8.8. So modules that use the pycairo C API (like pygtk) will need to be recompiled to use pycairo 1.8.10.



# CHAPTER 17

---

1.8.8 - 2009-08-26

---

## **General Changes:**

- Pycairo 1.8.8 requires cairo 1.8.8 (or later).
- Move from CVS to git.
- Add support for the waf build tool.

## **Updated Methods:**

- The PDF/PS/SVGSurface constructors now accept None as a filename.





# CHAPTER 18

---

1.8.6 - 2009-06-25

---

**General Changes:** Pycairo 1.8.6 requires cairo 1.8.6 (or later)

**Bug Fixes:**

- `ImageSurface.create_from_png_read_func` fix
- `ToyFontFace` type fix
- [#fdo-19221](#): restore `cairo.Matrix` `*` operator to the way it originally worked.

**Other Changes:** Documentation completed.



# CHAPTER 19

---

1.8.4 - 2009-03-19

---

**General Changes:** Pycairo 1.8.4 requires cairo 1.8.4 (or later) and Python 2.6

**Bug Fixes:**

- 20674: Add get/set\_extend for Gradient Patterns

**New Classes:** cairo.ToyFontFace

**New Methods:**

Pattern.get\_extend  
Pattern.set\_extend  
ToyFontFace.get\_family  
ToyFontFace.get\_slant  
ToyFontFace.get\_weight

**Deleted Methods:**

SurfacePattern.get\_extend  
SurfacePattern.set\_extend

**Other Changes:** Threading for surfaces with stream functions has been reenabled. Documentation updates.



## CHAPTER 20

---

1.8.2 - 2009-01-15

---

Pycairo 1.8.0 resulted in crashes for some applications using threads. So upgrading to 1.8.2 is recommended for threaded applications.

**Bug Fixes:**

- [#fdo-19287](#): Threading support results in crashes in `cairo.ImageSurface`

**New Methods:** `Context.set_scaled_font`

**API Changes:** Matrix multiplication:

```
old code: matrix3 = matrix1 * matrix2
new equivalent code: matrix3 = matrix1.multiply(matrix2)
matrix3 = matrix1 * matrix2
is now equivalent to matrix3 = matrix2.multiply(matrix1)
which is consistent with standard matrix multiplication.
```



# CHAPTER 21

---

1.8.0 - 2008-12-15

---

**General Changes:** Pycairo 1.8.0 requires cairo 1.8.0 (or later). Add documentation (available separately)

**Bug Fixes:**

- #fdo-18101: Add support for threading
- #fdo-18947: cairo.SurfacePattern should INCREMENT the used surface

**New Methods:**

ScaledFont.get\_scale\_matrix  
Surface.mark\_dirty\_rectangle  
Surface.set\_fallback\_resolution

**New Constants:**

cairo.EXTEND\_PAD  
cairo.HAS\_IMAGE\_SURFACE  
cairo.HAS\_USER\_FONT

**API Changes:**

- Surface.mark\_dirty: no longer accepts keyword arguments with default values.
- PycairoPattern\_FromPattern (C API): has a new 'base' argument - to fix #fdo-18947.

**Other Changes:** Allow unknown cairo Pattern/Surface types to use the pycairo base Pattern/Surface type.





**General changes:**

**Pycairo 1.6.4 requires cairo 1.6.4 (or later).** requires Python 2.5 (or later).

**Bug fixes:** #fdo-16112: Fix win32 'python setup.py ...' build – use double quotes

**New Methods:**

- Context.has\_current\_point
- Context.path\_extents
- ImageSurface.format\_stride\_for\_width
- PSSurface.get\_eps
- PSSurface.set\_eps
- PSSurface.ps\_level\_to\_string
- PSSurface.restrict\_to\_level
- Surface.copy\_page
- Surface.show\_page

**New Constants:** cairo.PS\_LEVEL\_2, cairo.PS\_LEVEL\_3

**Other changes:** test/pygame-test1.py, test/pygame-test2.py : pygame tests

examples/cairo\_snippets/snippets/ellipse.py : Update so line-width is a constant width in device-space not user-space



## CHAPTER 23

---

1.4.12 - 2007-12-13

---

**General changes:** Pycairo 1.4.12 requires cairo 1.4.12 (or later). requires Python 2.4 (or later).

**Bug fixes:**

- #fdo-10006: update autogen.sh to support automake >= 1.10
- #fdo-13460: use python-config to get python includes

**Other changes:**

- allow cairo.Context to be subclassed
- create a 'doc' subdirectory and start a FAQ file



## CHAPTER 24

---

1.4.0 - 2007-03-14

---

**General changes:** Pycairo 1.4.0 requires cairo 1.4.0 (or later).

**New methods:**

- Context.clip\_extents
- Context.copy\_clip\_rectangles
- Context.get\_dash
- Context.get\_dash\_count
- Context.get\_scaled\_font
- Context.glyph\_extents
- Context.glyph\_path
- Context.show\_glyphs
- LinearGradient.get\_linear\_points
- RadialGradient.get\_radial\_circles
- SolidPattern.get\_rgba
- SurfacePattern.get\_surface

**Deleted methods:** ImageSurface.create\_for\_array Remove Numeric Python support, since Numeric has been made obsolete by numpy, and numpy data can be read using ImageSurface.create\_for\_data.

**Other changes:** the module cairo.gtk has been removed (pygtk 2.7.0 onwards has cairo support built in).



## CHAPTER 25

---

1.2.6 - 2006-11-27

---

- Pycairo 1.2.6 requires cairo 1.2.6 (or later).
- mingw32 compiler fixes (Cedric Gustin)
- setup.py improvements (Cedric Gustin)
- `ImageSurface.get_data()` new method added `ImageSurface.get_data_as_rgba()` method removed





## CHAPTER 26

---

1.2.2 - 2006-08-21

---

- Pycairo requires cairo 1.2.2 (or later).
- setup.py has been updated to allow installation by executing `$ python setup.py install`
- examples/cairo\_snippets/snippets/gradient\_mask.py A new example to demonstrate pattern masks.
- The cairo.svg module has been removed because:
  1. Cairo does not include SVG parsing, so this module does not belong in pycairo.
  2. libsvg-cairo (the underlying C library) is unmaintained.



## CHAPTER 27

---

1.2.0 - 2006-07-03

---

**General changes:** Pycairo has been updated to work with cairo 1.2.0.

**New methods:**

Surface.set\_fallback\_resolution  
Surface.get\_content  
ImageSurface.get\_format  
Image\_surface.get\_stride

**Deleted methods:**

**PDFSurface.set\_dpi, PSSurface.set\_dpi, SVGSurface.set\_dpi**

- replaced by Surface.set\_fallback\_resolution

**Other changes:** cairo.FORMAT\_RGB16\_565 added



## CHAPTER 28

---

1.1.6 - 2006-05-29

---

**General changes:** Pycairo has been updated to work with cairo 1.1.6.

**New objects:** SVGSurface

**New methods:**

- Context.get\_group\_target
- Context.new\_sub\_path
- Context.pop\_group
- Context.pop\_group\_to\_source
- Context.push\_group
- Context.push\_group\_with\_content
- FontOptions.get\_antialias
- FontOptions.get\_hint\_metrics
- FontOptions.get\_hint\_style
- FontOptions.get\_subpixel\_order
- FontOptions.set\_antialias
- FontOptions.set\_hint\_metrics
- FontOptions.set\_hint\_style
- FontOptions.set\_subpixel\_order
- PDFSurface.set\_size
- PSSurface.dsc\_begin\_page\_setup
- PSSurface.dsc\_begin\_setup
- PSSurface.dsc\_comment
- PSSurface.set\_size
- ScaledFont.get\_font\_face
- ScaledFont.text\_extents
- Surface.get\_device\_offset
- XlibSurface.get\_depth

**Updated methods:** PDFSurface()/PSSurface() - can now write to file-like objects (like StringIO).

`surface.write_to_png()` and `ImageSurface.create_from_png()` can now write to file-like objects (like `StringIO`).  
`select_font_face`, `show_text`, `text_extents` and `text_path` now accept unicode objects.

**Other changes:** misc bug fixes.

**New examples:**

`examples/cairo_snippets/snippets_svg.py`  
`examples/cairo_snippets/snippets_ellipse.py`  
`examples/cairo_snippets/snippets_group.py`  
`examples/svg/svgconvert.py`

## CHAPTER 29

---

1.0.2 - 2005-10-11

---

**General changes:** Pycairo has been updated to work with cairo 1.0.2.

**New cairo functions supported:** `cairo.ImageSurface.create_for_data()`

**Updated functions:** `ctx.set_source_rgba(r, g, b, a=1.0)` now supports a default alpha argument

**Other changes:** `cairo.Matrix` now supports the Python sequence protocol, so you can do: `xx, yx, xy, yy, x0, y0 = matrix`





## CHAPTER 30

---

1.0.0 - 2005-08-31

---

**General changes:** Pycairo has been updated to work with cairo 1.0.0.

**New cairo functions supported:**

```
cairo.cairo_version()
cairo.cairo_version_string()
PSSurface.set_dpi()
```

**Patterns are now implemented in a class hierarchy, the new constructors are:**

```
cairo.SolidPattern (r, g, b, a=1.0)
cairo.SurfacePattern (surface)
cairo.LinearGradient (x0, y0, x1, y1)
cairo.RadialGradient (cx0, cy0, radius0, cx1, cy1, radius1)
```

**Updated functions:** Surface.write\_to\_png() now accepts a file object as well as a filename

**Updated examples:** The gtk examples now work with pygtk >= 2.7.0 without requiring the cairo.gtk module

**Bug Fixes:** fix “initializer element is not constant” compiler warnings



# CHAPTER 31

---

0.9.0 - 2005-08-10

---

**General changes:** Pycairo has been updated to work with cairo 0.9.0.

**New cairo functions supported:**

cairo\_get\_antialias  
cairo\_set\_antialias  
cairo\_surface\_mark\_dirty\_rectangle  
cairo\_surface\_flush

**Bug Fixes:**

- double buffering now works with the cairo.gtk module



## CHAPTER 32

---

0.6.0 - 2005-08-01

---

This version has many changes which update Pycairo to the new cairo API. The change list is not duplicated here, instead see the `cairo/NEWS` file for full details of all these API changes.

Pycairo method names that were different from the underlying cairo function names have been changed to make Pycairo more closely follow cairo and so enable the cairo documentation to be used for writing Pycairo programs. NOTES has been updated to list the differences between the C API and the Pycairo API.

`Context.copy_path()` has been implemented, it returns a `Path` instance which supports the iterator protocol.

Python 2.3 is now required.

**New examples:** `examples/warpedtext.py`: shows usage of the `Path` iterator

`examples/cairo_snippets/`: shows many of the ‘`cairo-demo/cairo_snippets`’ examples

`examples/gtk/png_view.py`: example using `cairo.ImageSurface.create_from_png()`

**General changes:** Pycairo has been updated to work with cairo 0.6.0, including using cairo’s new error handling scheme.

**New features:** `cairo.CONTENT_COLOR`, `cairo.ALPHA`, `cairo.COLOR_ALPHA` have been added for working with surfaces.

A new class `cairo.FontOptions` has been added.

`cairo.ImageSurface.create_from_png()` now accepts a filename string or a file object

New wrapper functions have been added for `cairo_get_font_options`, `cairo_set_font_options` and `cairo_surface_get_font_options`.



## CHAPTER 33

---

0.5.1 - 2005-06-22

---

### **New features:**

- new class `cairo.Win32Surface` (Niki Spahiev)
- `cairo.HAS_WIN32_SURFACE`, `cairo.HAS_PS_SURFACE` etc are defined to give access to the values from `cairo-features.h`

### **Fixes:**

- fix `cairo_mask`, `cairo_mask_surface` and `cairo_stroke_preserve` wrappers
- compile properly against GTK+ 2.7 (Gustavo Carneiro)
- other small fixes, including fixes for gcc 4.0 warnings





## CHAPTER 34

---

0.4.0 - 2005-03-10

---

**New cairo bindings:**

cairo\_font\_extents

**Bindings removed:**

cairo\_font\_set\_transform

cairo\_font\_current\_transform

**New examples:** gtk/hangman.py

**Other:** Changed version numbering to correspond directly with the Cairo version Pycairo was developed to work with. So, for example, Pycairo version 0.4.0 represents the Pycairo version that has been developed and tested with Cairo 0.4.0.



The Pycairo license has changed so that it is now dual-licensed under the LGPL and the MPL, the same as Cairo itself. For details see the COPYING file as well as COPYING-LGPL-2.1 and COPYING-MPL-1.1.

**New cairo bindings:**

cairo\_pdf\_surface\_create  
cairo\_set\_target\_pdf

**New libsvg-cairo bindings:**

svg\_cairo\_parse  
svg\_cairo\_parse\_buffer  
svg\_cairo\_render  
svg\_cairo\_get\_size

**Other:**

- Added `--without-pygtk` configure option.
- Renamed the Pycairo API `_new()` functions to `_wrap()` to allow `_new()` to be used for python `__new__` functions.
- New examples: `svg2png.py` and `svgview.py`.



## CHAPTER 36

---

0.1.3 - 2004-11-24

---

After the recent server compromise we discarded all unsigned snapshots. That left us without a pycairo snapshot.

Additionally, there were no tags in the source repository so I couldn't recreate the 0.1.2 snapshot, so here's a new 0.1.3 snapshot.

I apologize if I botched the version number or left something significant out of this announcement—I'm not the one who will usually be doing pycairo maintenance.

**New bindings:**

- current\_path
- current\_path\_flat
- current\_font\_extents

**Changes:** fill\_extents,stroke\_extents: Remove unnecessary args and change from a method to an attribute.

**Other:** Added two new examples: context-subclass.py and warpedtext.py



---

## An Introduction to Cairo with Python

---

Cairo is a library for drawing vector graphics. Vector graphics are interesting because they don't lose clarity when resized or transformed.

Pycairo is a set of bindings for cairo. It provides the cairo module which can be used to call cairo commands from Python.

### Integration with other Libraries

#### NumPy & ImageSurface

**Creating an ImageSurface from a NumPy array:**

```
import numpy
import cairo

width, height = 255, 255
data = numpy.ndarray(shape=(height, width), dtype=numpy.uint32)
surface = cairo.ImageSurface.create_for_data(
    data, cairo.FORMAT_ARGB32, width, height)
```

**Creating a NumPy array from an ImageSurface:**

```
import numpy
import cairo

width, height = 255, 255
surface = cairo.ImageSurface(cairo.FORMAT_ARGB32, width, height)
buf = surface.get_data()
data = numpy.ndarray(shape=(width, height),
                    dtype=numpy.uint32,
                    buffer=buf)
```

## Pygame & ImageSurface

### Creating a pygame.image from an ImageSurface:

```
import pygame
import cairo

width, height = 255, 255
surface = cairo.ImageSurface(cairo.FORMAT_ARGB32, width, height)
buf = surface.get_data()
image = pygame.image.frombuffer(buf, (width, height), "ARGB")
```

## Pyglet & ImageSurface as Texture

### Creating a pyglet.Texture from an ImageSurface

```
import ctypes
import cairo

from pyglet import app, clock, gl, image, window

# create data shared by ImageSurface and Texture
width, height = 400, 400

surface_data = (ctypes.c_ubyte * (width * height * 4))()
surface = cairo.ImageSurface.create_for_data (surface_data, cairo.FORMAT_ARGB32,
width, height, width * 4);
texture = image.Texture.create_for_size(gl.GL_TEXTURE_2D, width, height, gl.GL_
↳RGBA)
```

### Draw pyglet.Texture bound to ImageSurface

```
window = window.Window(width=width, height=height)

@window.event
def on_draw():
    window.clear()

    # Draw texture backed by ImageSurface
    gl.glEnable(gl.GL_TEXTURE_2D)

    gl.glBindTexture(gl.GL_TEXTURE_2D, texture.id)
    gl.glTexImage2D(gl.GL_TEXTURE_2D, 0, gl.GL_RGBA, width, height, 0, gl.GL_BGRA,
gl.GL_UNSIGNED_BYTE,
    surface_data)

    gl.glBegin(gl.GL_QUADS)
    gl.glTexCoord2f(0.0, 1.0)
    gl.glVertex2i(0, 0)
    gl.glTexCoord2f(1.0, 1.0)
    gl.glVertex2i(width, 0)
    gl.glTexCoord2f(1.0, 0.0)
    gl.glVertex2i(width, height)
    gl.glTexCoord2f(0.0, 0.0)
    gl.glVertex2i(0, height)
    gl.glEnd()
```



```
# call clock.schedule_update here to update the ImageSurface every frame
app.run()
```

## Pillow (PIL) & Cairo

### Creating an ImageSurface from a PIL Image:

```
import PIL.Image as Image

def from_pil(im, alpha=1.0, format=cairo.FORMAT_ARGB32):
    """
    :param im: Pillow Image
    :param alpha: 0..1 alpha to add to non-alpha images
    :param format: Pixel format for output surface
    """
    assert format in (cairo.FORMAT_RGB24, cairo.FORMAT_ARGB32), "Unsupported_
    ↳pixel format: %s" % format
    if 'A' not in im.getbands():
        im.putalpha(int(alpha * 256.))
    arr = bytearray(im.tobytes('raw', 'BGRA'))
    surface = cairo.ImageSurface.create_for_data(arr, format, im.width, im.height)
    return surface

filename = 'test.jpeg'

# Open image to an ARGB32 ImageSurface
im = Image.open(filename)
surface1 = from_pil(im)

# Open image to an RGB24 ImageSurface
im = Image.open(filename)
surface2 = from_pil(im, format=cairo.FORMAT_RGB24)

# Open image to an ARGB32 ImageSurface, 50% opacity
im = Image.open(filename)
surface3 = from_pil(im, alpha=0.5, format=cairo.FORMAT_ARGB32)
```

## Freetype-py & Cairo

See <https://github.com/rougier/freetype-py/tree/master/examples> for examples. Most of the `*-cairo.py` examples illustrate conversion from FreeType bitmaps to Cairo surfaces; the two examples, `glyph-vector-cairo.py` and `glyph-vector-2-cairo.py`, illustrate conversion from FreeType glyph contours to Cairo paths.

## Examples

The Git repository and release tarball contain various examples showing various features of cairo and integration with pygame and GTK+ in the “examples” directory:

<https://github.com/pygobject/pycairo/tree/master/examples>

## Understanding How to use Cairo

The best way to understand how to use cairo is to imagine that you are an artist using a paintbrush to draw out a shape on canvas.

To begin, you can choose a few characteristics of your brush. You can choose the thickness of your brush and the colour you want to paint with. You can also choose the shape of your brush tip - You can choose either a circle or a square.

Once you have chosen your brush, you are ready to start painting. You have to be quite precise when describing what you want to appear.

Firstly, decide where you want to place your brush on the canvas. You do this by supplying an x & y coordinate. Next you define how you want your brush stroke to look - an arc, a straight line etc. Finally you define the point where you want your stroke to end, again by supplying an x & y coordinate. Triangles and squares are very easy to do!

More complex graphics are generated using variations of the above theme with a few additions such as Fills (colouring in), transformations (zooming in, moving) etc. Using the Python interface to cairo

Nearly all the work revolves around using the `cairo.Context` (or `cairo_t` in the cairo C API). This is the object that you send your drawing commands to. There are a few options available to initialize this object in different ways.

## Initializing the `cairo.Context` Object

- One Very Important thing to realize is there is a difference between the coordinates you are describing your graphics on and the coordinates you will be displaying your graphic on.

(Ex - When giving a presentation you draw on your transparent acetate before hand, and then display it on your overhead projector - cairo calls the transparent acetate the user space coordinates and the projected image the device space coordinates)

On initializing the cairo context object, we tell it how to transform our description to how it should be displayed. To do this we supply a transformation matrix. Modifying the transformation matrix can lead to some very interesting results.

- One of cairo's most powerful features is that it can output graphics in many different formats (it can use multiple back ends). For printing, we can have cairo translate our graphics into Postscript to be sent off to the printer. For on screen display, we can have cairo translate our graphics into something glitz can understand for hardware accelerated rendering! It has many more important and useful target back ends. On initializing the `cairo.Context`, we set its target back end, supplying a few details (such as colour depth and size), as in the example below.

## Example

```
#!/usr/bin/env python

import math
import cairo

WIDTH, HEIGHT = 256, 256

surface = cairo.ImageSurface (cairo.FORMAT_ARGB32, WIDTH, HEIGHT)
ctx = cairo.Context (surface)
```

```
ctx.scale (WIDTH, HEIGHT) # Normalizing the canvas

pat = cairo.LinearGradient (0.0, 0.0, 0.0, 1.0)
pat.add_color_stop_rgba (1, 0.7, 0, 0, 0.5) # First stop, 50% opacity
pat.add_color_stop_rgba (0, 0.9, 0.7, 0.2, 1) # Last stop, 100% opacity

ctx.rectangle (0, 0, 1, 1) # Rectangle(x0, y0, x1, y1)
ctx.set_source (pat)
ctx.fill ()

ctx.translate (0.1, 0.1) # Changing the current transformation matrix

ctx.move_to (0, 0)
# Arc(cx, cy, radius, start_angle, stop_angle)
ctx.arc (0.2, 0.1, 0.1, -math.pi/2, 0)
ctx.line_to (0.5, 0.1) # Line to (x,y)
# Curve(x1, y1, x2, y2, x3, y3)
ctx.curve_to (0.5, 0.2, 0.5, 0.4, 0.2, 0.8)
ctx.close_path ()

ctx.set_source_rgb (0.3, 0.2, 0.5) # Solid color
ctx.set_line_width (0.02)
ctx.stroke ()

surface.write_to_png ("example.png") # Output to PNG
```



## Module Functions and Constants

### Module Functions

`cairo.cairo_version()`

**Returns** the encoded version

**Return type** `int`

Returns the version of the underlying C cairo library, encoded in a single integer.

`cairo.cairo_version_string()`

**Returns** the encoded version

**Return type** `str`

Returns the version of the underlying C cairo library as a human-readable string of the form “X.Y.Z”.

### Module Constants

`cairo.version`

the pycairo version, as a string

`cairo.version_info`

the pycairo version, as a tuple

### `cairo.HAS`

1 if the feature is present in the underlying C cairo library, 0 otherwise.

`cairo.HAS_ATSUI_FONT`

`cairo.HAS_FT_FONT`

`cairo.HAS_GLITZ_SURFACE`  
`cairo.HAS_IMAGE_SURFACE`  
`cairo.HAS_PDF_SURFACE`  
`cairo.HAS_PNG_FUNCTIONS`  
`cairo.HAS_PS_SURFACE`  
`cairo.HAS_RECORDING_SURFACE`  
`cairo.HAS_SVG_SURFACE`  
`cairo.HAS_USER_FONT`  
`cairo.HAS_QUARTZ_SURFACE`  
`cairo.HAS_WIN32_FONT`  
`cairo.HAS_WIN32_SURFACE`  
`cairo.HAS_XCB_SURFACE`  
`cairo.HAS_XLIB_SURFACE`

`cairo.HAS_MIME_SURFACE`  
New in version 1.12.0.

`cairo.HAS_SCRIPT_SURFACE`  
New in version 1.12.0.

`cairo.HAS_TEE_SURFACE`  
New in version 1.15.3.

### cairo.MIME\_TYPE

`cairo.MIME_TYPE_JP2` = “image/jp2”  
The Joint Photographic Experts Group (JPEG) 2000 image coding standard (ISO/IEC 15444-1).  
New in version 1.12.0.

`cairo.MIME_TYPE_JPEG` = “image/jpeg”  
The Joint Photographic Experts Group (JPEG) image coding standard (ISO/IEC 10918-1).  
New in version 1.12.0.

`cairo.MIME_TYPE_PNG` = “image/png”  
The Portable Network Graphics image file format (ISO/IEC 15948).  
New in version 1.12.0.

`cairo.MIME_TYPE_URI` = “text/x-uri”  
URI for an image file (unofficial MIME type).  
New in version 1.12.0.

`cairo.MIME_TYPE_UNIQUE_ID` = “application/x-cairo.uuid”  
Unique identifier for a surface (cairo specific MIME type). All surfaces with the same unique identifier will only be embedded once.  
New in version 1.12.0.

## Other Classes and Functions

**class** `cairo.text`

This type only exists for documentation purposes. It represents `str/unicode` under Python 2 and `str` under Python 3.

**class** `cairo.pathlike`

This type only exists for documentation purposes. It represents everything Python allows as a filesystem path except on Windows where only ANSI paths are supported. To use Unicode paths on Windows most functions take an already open file object which you can create from a Unicode path and then pass to `pycairo` instead.

New in version 1.15.1: Older versions only supported a subset of `str` paths

## Enums

Before Pycairo 1.13 most of the enum values defined here were only available as constants on the module level. See *Legacy Constants*.

**class** `cairo.Antialias`

Specifies the type of antialiasing to do when rendering text or shapes.

New in version 1.13.

**DEFAULT**

Use the default antialiasing for the subsystem and target device

**NONE**

Use a bilevel alpha mask

**GRAY**

Perform single-color antialiasing (using shades of gray for black text on a white background, for example).

**SUBPIXEL**

Perform antialiasing by taking advantage of the order of subpixel elements on devices such as LCD panels.

**FAST**

Hint that the backend should perform some antialiasing but prefer speed over quality.

**GOOD**

The backend should balance quality against performance.

**BEST**

Hint that the backend should render at the highest quality, sacrificing speed if necessary.

**class** `cairo.Content`

These constants are used to describe the content that a *Surface* will contain, whether color information, alpha information (translucence vs. opacity), or both.

New in version 1.13.

**COLOR**

The surface will hold color content only.

**ALPHA**

The surface will hold alpha content only.

**COLOR\_ALPHA**

The surface will hold color and alpha content.

**class** `cairo.Extend`

These constants are used to describe how *Pattern* color/alpha will be determined for areas “outside” the pattern’s natural area, (for example, outside the surface bounds or outside the gradient geometry).

The default extend mode is *NONE* for *SurfacePattern* and *PAD* for *Gradient* patterns.

New in version 1.13.

**NONE**

pixels outside of the source pattern are fully transparent

**REPEAT**

the pattern is tiled by repeating

**REFLECT**

the pattern is tiled by reflecting at the edges (Implemented for surface patterns since 1.6)

**PAD**

pixels outside of the pattern copy the closest pixel from the source (Since 1.2; but only implemented for surface patterns since 1.6)

**class** `cairo.FillRule`

These constants are used to select how paths are filled. For both fill rules, whether or not a point is included in the fill is determined by taking a ray from that point to infinity and looking at intersections with the path. The ray can be in any direction, as long as it doesn't pass through the end point of a segment or have a tricky intersection such as intersecting tangent to the path. (Note that filling is not actually implemented in this way. This is just a description of the rule that is applied.)

The default fill rule is *WINDING*.

New in version 1.13.

**WINDING**

If the path crosses the ray from left-to-right, counts +1. If the path crosses the ray from right to left, counts -1. (Left and right are determined from the perspective of looking along the ray from the starting point.) If the total count is non-zero, the point will be filled.

**EVEN\_ODD**

Counts the total number of intersections, without regard to the orientation of the contour. If the total number of intersections is odd, the point will be filled.

**class** `cairo.Filter`

These constants are used to indicate what filtering should be applied when reading pixel values from patterns. See *Pattern.set\_filter()* for indicating the desired filter to be used with a particular pattern.

New in version 1.13.

**FAST**

A high-performance filter, with quality similar *FILTER\_NEAREST*

**GOOD**

A reasonable-performance filter, with quality similar to *FILTER\_BILINEAR*

**BEST**

The highest-quality available, performance may not be suitable for interactive use.

**NEAREST**

Nearest-neighbor filtering

**BILINEAR**

Linear interpolation in two dimensions

**GAUSSIAN**

This filter value is currently unimplemented, and should not be used in current code.

**class** `cairo.FontSlant`

These constants specify variants of a *FontFace* based on their slant.

New in version 1.13.



**NORMAL**

Upright font style

**ITALIC**

Italic font style

**OBLIQUE**

Oblique font style

**class** `cairo.FontWeight`

These constants specify variants of a *FontFace* based on their weight.

New in version 1.13.

**NORMAL**

Normal font weight

**BOLD**

Bold font weight

**class** `cairo.Format`

These constants are used to identify the memory format of *ImageSurface* data.

New entries may be added in future versions.

New in version 1.13.

**stride\_for\_width** (*width*)

**Parameters** *width* (*int*) – the desired width of an *ImageSurface* to be created.

**Returns** the appropriate stride to use given the desired format and width, or -1 if either the format is invalid or the width too large.

**Return type** *int*

This method provides a stride value that will respect all alignment requirements of the accelerated image-rendering code within cairo. Typical usage will be of the form:

```
format = cairo.Format.RGB24
stride = format.stride_for_width(width)
surface = cairo.ImageSurface.create_for_data(
    data, format, width, height, stride)
```

Also available under `cairo.ImageSurface.format_stride_for_width()`.

New in version 1.14.

**INVALID**

no such format exists or is supported.

**ARGB32**

each pixel is a 32-bit quantity, with alpha in the upper 8 bits, then red, then green, then blue. The 32-bit quantities are stored native-endian. Pre-multiplied alpha is used. (That is, 50% transparent red is 0x80800000, not 0x80ff0000.)

**RGB24**

each pixel is a 32-bit quantity, with the upper 8 bits unused. Red, Green, and Blue are stored in the remaining 24 bits in that order.

**A8**

each pixel is a 8-bit quantity holding an alpha value.

**A1**

each pixel is a 1-bit quantity holding an alpha value. Pixels are packed together into 32-bit quantities. The ordering of the bits matches the endianness of the platform. On a big-endian machine, the first pixel is in the uppermost bit, on a little-endian machine the first pixel is in the least-significant bit.

**RGB16\_565**

each pixel is a 16-bit quantity with red in the upper 5 bits, then green in the middle 6 bits, and blue in the lower 5 bits.

**RGB30**

like *RGB24* but with 10bpc.

**class** `cairo.HintMetrics`

These constants specify whether to hint font metrics; hinting font metrics means quantizing them so that they are integer values in device space. Doing this improves the consistency of letter and line spacing, however it also means that text will be laid out differently at different zoom factors.

New in version 1.13.

**DEFAULT**

Hint metrics in the default manner for the font backend and target device

**OFF**

Do not hint font metrics

**ON**

Hint font metrics

**class** `cairo.HintStyle`

These constants specify the type of hinting to do on font outlines. Hinting is the process of fitting outlines to the pixel grid in order to improve the appearance of the result. Since hinting outlines involves distorting them, it also reduces the faithfulness to the original outline shapes. Not all of the outline hinting styles are supported by all font backends.

New entries may be added in future versions.

New in version 1.13.

**DEFAULT**

Use the default hint style for font backend and target device

**NONE**

Do not hint outlines

**SLIGHT**

Hint outlines slightly to improve contrast while retaining good fidelity to the original shapes.

**MEDIUM**

Hint outlines with medium strength giving a compromise between fidelity to the original shapes and contrast

**FULL**

Hint outlines to maximize contrast

**class** `cairo.LineCap`

These constants specify how to render the endpoints of the path when stroking.

The default line cap style is *BUTT*

New in version 1.13.

**BUTT**

start(stop) the line exactly at the start(end) point

**ROUND**

use a round ending, the center of the circle is the end point

**SQUARE**

use squared ending, the center of the square is the end point

**class** `cairo.LineJoin`

These constants specify how to render the junction of two lines when stroking.

The default line join style is *MITER*

New in version 1.13.

**MITER**

use a sharp (angled) corner, see `Context.set_miter_limit()`

**ROUND**

use a rounded join, the center of the circle is the joint point

**BEVEL**

use a cut-off join, the join is cut off at half the line width from the joint point

**class** `cairo.Operator`

These constants are used to set the compositing operator for all cairo drawing operations.

The default operator is *OVER*.

The operators marked as *unbounded* modify their destination even outside of the mask layer (that is, their effect is not bound by the mask layer). However, their effect can still be limited by way of clipping.

To keep things simple, the operator descriptions here document the behavior for when both source and destination are either fully transparent or fully opaque. The actual implementation works for translucent layers too.

For a more detailed explanation of the effects of each operator, including the mathematical definitions, see <https://cairographics.org/operators>.

New in version 1.13.

**CLEAR**

clear destination layer (bounded)

**SOURCE**

replace destination layer (bounded)

**OVER**

draw source layer on top of destination layer (bounded)

**IN**

draw source where there was destination content (unbounded)

**OUT**

draw source where there was no destination content (unbounded)

**ATOP**

draw source on top of destination content and only there

**DEST**

ignore the source

**DEST\_OVER**

draw destination on top of source

**DEST\_IN**

leave destination only where there was source content (unbounded)

**DEST\_OUT**

leave destination only where there was no source content

**DEST\_ATOP**

leave destination on top of source content and only there (unbounded)

**XOR**

source and destination are shown where there is only one of them

**ADD**

source and destination layers are accumulated

**SATURATE**

like over, but assuming source and dest are disjoint geometries

**MULTIPLY**

source and destination layers are multiplied. This causes the result to be at least as dark as the darker inputs.

**SCREEN**

source and destination are complemented and multiplied. This causes the result to be at least as light as the lighter inputs.

**OVERLAY**

multiplies or screens, depending on the lightness of the destination color.

**DARKEN**

replaces the destination with the source if it is darker, otherwise keeps the source.

**LIGHTEN**

replaces the destination with the source if it is lighter, otherwise keeps the source.

**COLOR\_DODGE**

brightens the destination color to reflect the source color.

**COLOR\_BURN**

darkens the destination color to reflect the source color.

**HARD\_LIGHT**

Multiplies or screens, dependent on source color.

**SOFT\_LIGHT**

Darkens or lightens, dependent on source color.

**DIFFERENCE**

Takes the difference of the source and destination color.

**EXCLUSION**

Produces an effect similar to difference, but with lower contrast.

**HSL\_HUE**

Creates a color with the hue of the source and the saturation and luminosity of the target.

**HSL\_SATURATION**

Creates a color with the saturation of the source and the hue and luminosity of the target. Painting with this mode onto a gray area produces no change.

**HSL\_COLOR**

Creates a color with the hue and saturation of the source and the luminosity of the target. This preserves the gray levels of the target and is useful for coloring monochrome images or tinting color images.

**HSL\_LUMINOSITY**

Creates a color with the luminosity of the source and the hue and saturation of the target. This produces an inverse effect to *HSL\_COLOR*

**class** `cairo.PathDataType`

These constants are used to describe the type of one portion of a path when represented as a *Path*.

New in version 1.13.

**MOVE\_TO**

A move-to operation

**LINE\_TO**

A line-to operation

**CURVE\_TO**

A curve-to operation

**CLOSE\_PATH**

A close-path operation

**class** `cairo.PSLevel`

These constants are used to describe the language level of the PostScript Language Reference that a generated PostScript file will conform to. Note: the constants are only defined when cairo has been compiled with PS support enabled.

New in version 1.13.

**LEVEL\_2**

The language level 2 of the PostScript specification.

**LEVEL\_3**

The language level 3 of the PostScript specification.

**class** `cairo.PDFVersion`

These constants are used to describe the version number of the PDF specification that a generated PDF file will conform to.

New in version 1.13.

**VERSION\_1\_4**

The version 1.4 of the PDF specification.

**VERSION\_1\_5**

The version 1.5 of the PDF specification.

**class** `cairo.SVGVersion`

These constants are used to describe the version number of the SVG specification that a generated SVG file will conform to.

New in version 1.13.

**VERSION\_1\_1**

The version 1.1 of the SVG specification.

**VERSION\_1\_2**

The version 1.2 of the SVG specification.

**class** `cairo.SubpixelOrder`

The subpixel order specifies the order of color elements within each pixel on the display device when rendering with an antialiasing mode of *Antialias.SUBPIXEL*.

New in version 1.13.

**DEFAULT**

Use the default subpixel order for for the target device

**RGB**

Subpixel elements are arranged horizontally with red at the left

**BGR**

Subpixel elements are arranged horizontally with blue at the left

**VRGB**

Subpixel elements are arranged vertically with red at the top

**VBGR**

Subpixel elements are arranged vertically with blue at the top

**class** `cairo.RegionOverlap`

New in version 1.13.

**IN**

The contents are entirely inside the region.

**OUT**

The contents are entirely outside the region.

**PART**

The contents are partially inside and partially outside the region.

**class** `cairo.Status`

New in version 1.13.

**SUCCESS**

**NO\_MEMORY**

**INVALID\_RESTORE**

**INVALID\_POP\_GROUP**

**NO\_CURRENT\_POINT**

**INVALID\_MATRIX**

**INVALID\_STATUS**

**NULL\_POINTER**

**INVALID\_STRING**

**INVALID\_PATH\_DATA**

**READ\_ERROR**

**WRITE\_ERROR**

**SURFACE\_FINISHED**

**SURFACE\_TYPE\_MISMATCH**

**PATTERN\_TYPE\_MISMATCH**

**INVALID\_CONTENT**

**INVALID\_FORMAT**

**INVALID\_VISUAL**

**FILE\_NOT\_FOUND**

**INVALID\_DASH**

**INVALID\_DSC\_COMMENT**

**INVALID\_INDEX**

**CLIP\_NOT\_REPRESENTABLE**

**TEMP\_FILE\_ERROR**

**INVALID\_STRIDE**

**FONT\_TYPE\_MISMATCH**

**USER\_FONT\_IMMUTABLE**

**USER\_FONT\_ERROR**

**NEGATIVE\_COUNT**

**INVALID\_CLUSTERS**  
**INVALID\_SLANT**  
**INVALID\_WEIGHT**  
**INVALID\_SIZE**  
**USER\_FONT\_NOT\_IMPLEMENTED**  
**DEVICE\_TYPE\_MISMATCH**  
**DEVICE\_ERROR**  
**INVALID\_MESH\_CONSTRUCTION**  
**DEVICE\_FINISHED**  
**LAST\_STATUS**

**JBIG2\_GLOBAL\_MISSING**

New in version 1.14.

**class** `cairo.ScriptMode`

A set of script output variants.

New in version 1.14.

**ASCII**

the output will be in readable text (default)

**BINARY**

the output will use byte codes.

**class** `cairo.TextClusterFlags`

Specifies properties of a text cluster mapping.

New in version 1.14.

**BACKWARD**

The clusters in the cluster array map to glyphs in the glyph array from end to start.

**class** `cairo.SurfaceObserverMode`

Whether operations should be recorded.

New in version 1.14.

**NORMAL**

no recording is done

**RECORD\_OPERATIONS**

operations are recorded

## Cairo Context

### class `Context()`

*Context* is the main object used when drawing with cairo. To draw with cairo, you create a *Context*, set the target surface, and drawing options for the *Context*, create shapes with functions like *Context.move\_to()* and *Context.line\_to()*, and then draw shapes with *Context.stroke()* or *Context.fill()*.

*Contexts* can be pushed to a stack via *Context.save()*. They may then safely be changed, without losing the current state. Use *Context.restore()* to restore to the saved state.

**class** `cairo.Context` (*target*)

**Parameters** *target* – target *Surface* for the context

**Returns** a newly allocated *Context*

**Raises** *MemoryError* in case of no memory

Creates a new *Context* with all graphics state parameters set to default values and with *target* as a target surface. The target surface should be constructed with a backend-specific function such as *ImageSurface* (or any other cairo backend surface create variant).

**append\_path** (*path*)

**Parameters** *path* – *Path* to be appended

Append the *path* onto the current path. The *path* may be either the return value from one of *Context.copy\_path()* or *Context.copy\_path\_flat()* or it may be constructed manually (in C).

**arc** (*xc, yc, radius, angle1, angle2*)

**Parameters**

- **xc** (*float*) – X position of the center of the arc
- **yc** (*float*) – Y position of the center of the arc
- **radius** (*float*) – the radius of the arc
- **angle1** (*float*) – the start angle, in radians
- **angle2** (*float*) – the end angle, in radians

Adds a circular arc of the given *radius* to the current path. The arc is centered at (*xc, yc*), begins at *angle1* and proceeds in the direction of increasing angles to end at *angle2*. If *angle2* is less than *angle1* it will be progressively increased by  $2\pi$  until it is greater than *angle1*.

If there is a current point, an initial line segment will be added to the path to connect the current point to the beginning of the arc. If this initial line is undesired, it can be avoided by calling *Context.new\_sub\_path()* before calling *Context.arc()*.

Angles are measured in radians. An angle of 0.0 is in the direction of the positive X axis (in user space). An angle of  $\pi/2.0$  radians (90 degrees) is in the direction of the positive Y axis (in user space). Angles increase in the direction from the positive X axis toward the positive Y axis. So with the default transformation matrix, angles increase in a clockwise direction.

To convert from degrees to radians, use `degrees * (math.pi / 180)`.

This function gives the arc in the direction of increasing angles; see *Context.arc\_negative()* to get the arc in the direction of decreasing angles.

The arc is circular in user space. To achieve an elliptical arc, you can scale the current transformation matrix by different amounts in the X and Y directions. For example, to draw an ellipse in the box given by *x, y, width, height*:

```
ctx.save()
ctx.translate(x + width / 2., y + height / 2.)
ctx.scale(width / 2., height / 2.)
ctx.arc(0., 0., 1., 0., 2 * math.pi)
ctx.restore()
```

**arc\_negative** (*xc, yc, radius, angle1, angle2*)

**Parameters**

- **xc** (*float*) – X position of the center of the arc
- **yc** (*float*) – Y position of the center of the arc
- **radius** (*float*) – the radius of the arc
- **angle1** (*float*) – the start angle, in radians



- **angle2** (*float*) – the end angle, in radians

Adds a circular arc of the given *radius* to the current path. The arc is centered at (*xc*, *yc*), begins at *angle1* and proceeds in the direction of decreasing angles to end at *angle2*. If *angle2* is greater than *angle1* it will be progressively decreased by  $2\pi$  until it is less than *angle1*.

See `Context.arc()` for more details. This function differs only in the direction of the arc between the two angles.

#### **clip()**

Establishes a new clip region by intersecting the current clip region with the current path as it would be filled by `Context.fill()` and according to the current *fill rule* (see `Context.set_fill_rule()`).

After `clip()`, the current path will be cleared from the `Context`.

The current clip region affects all drawing operations by effectively masking out any changes to the surface that are outside the current clip region.

Calling `clip()` can only make the clip region smaller, never larger. But the current clip is part of the graphics state, so a temporary restriction of the clip region can be achieved by calling `clip()` within a `Context.save()/Context.restore()` pair. The only other means of increasing the size of the clip region is `Context.reset_clip()`.

#### **clip\_extents()**

**Returns** (*x1*, *y1*, *x2*, *y2*), all float

**Return type** tuple

- *x1*: left of the resulting extents
- *y1*: top of the resulting extents
- *x2*: right of the resulting extents
- *y2*: bottom of the resulting extents

Computes a bounding box in user coordinates covering the area inside the current clip.

New in version 1.4.

#### **clip\_preserve()**

Establishes a new clip region by intersecting the current clip region with the current path as it would be filled by `Context.fill()` and according to the current *fill rule* (see `Context.set_fill_rule()`).

Unlike `Context.clip()`, `clip_preserve()` preserves the path within the `Context`.

The current clip region affects all drawing operations by effectively masking out any changes to the surface that are outside the current clip region.

Calling `clip_preserve()` can only make the clip region smaller, never larger. But the current clip is part of the graphics state, so a temporary restriction of the clip region can be achieved by calling `clip_preserve()` within a `Context.save()/Context.restore()` pair. The only other means of increasing the size of the clip region is `Context.reset_clip()`.

#### **close\_path()**

Adds a line segment to the path from the current point to the beginning of the current sub-path, (the most recent point passed to `Context.move_to()`), and closes this sub-path. After this call the current point will be at the joined endpoint of the sub-path.

The behavior of `close_path()` is distinct from simply calling `Context.line_to()` with the equivalent coordinate in the case of stroking. When a closed sub-path is stroked, there are no caps on the ends of the sub-path. Instead, there is a line join connecting the final and initial segments of the sub-path.

If there is no current point before the call to `close_path()`, this function will have no effect.

Note: As of cairo version 1.2.4 any call to `close_path()` will place an explicit `MOVE_TO` element into the path immediately after the `CLOSE_PATH` element, (which can be seen in `Context.copy_path()` for example). This can simplify path processing in some cases as it may not be necessary to save the “last move\_to point” during processing as the `MOVE_TO` immediately after the `CLOSE_PATH` will provide that point.

#### `copy_clip_rectangle_list()`

**Returns** the current clip region as a list of rectangles in user coordinates. Returns a list of *Rectangle*

**Return type** *list*

New in version 1.4.

#### `copy_page()`

Emits the current page for backends that support multiple pages, but doesn't clear it, so, the contents of the current page will be retained for the next page too. Use `Context.show_page()` if you want to get an empty page after the emission.

This is a convenience function that simply calls `Surface.copy_page()` on `Context's` target.

#### `copy_path()`

**Returns** *Path*

**Raises** *MemoryError* in case of no memory

Creates a copy of the current path and returns it to the user as a *Path*.

#### `copy_path_flat()`

**Returns** *Path*

**Raises** *MemoryError* in case of no memory

Gets a flattened copy of the current path and returns it to the user as a *Path*.

This function is like `Context.copy_path()` except that any curves in the path will be approximated with piecewise-linear approximations, (accurate to within the current tolerance value). That is, the result is guaranteed to not have any elements of type `CAIRO_PATH_CURVE_TO` which will instead be replaced by a series of `CAIRO_PATH_LINE_TO` elements.

#### `curve_to(x1, y1, x2, y2, x3, y3)`

##### **Parameters**

- **x1** (*float*) – the X coordinate of the first control point
- **y1** (*float*) – the Y coordinate of the first control point
- **x2** (*float*) – the X coordinate of the second control point
- **y2** (*float*) – the Y coordinate of the second control point
- **x3** (*float*) – the X coordinate of the end of the curve
- **y3** (*float*) – the Y coordinate of the end of the curve

Adds a cubic Bézier spline to the path from the current point to position  $(x_3, y_3)$  in user-space coordinates, using  $(x_1, y_1)$  and  $(x_2, y_2)$  as the control points. After this call the current point will be  $(x_3, y_3)$ .

If there is no current point before the call to `curve_to()` this function will behave as if preceded by a call to `ctx.move_to(x1, y1)`.

**device\_to\_user**  $(x, y)$

**Parameters**

- **x** (*float*) – X value of coordinate
- **y** (*float*) – Y value of coordinate

**Returns**  $(x, y)$ , both float

**Return type** `tuple`

Transform a coordinate from device space to user space by multiplying the given point by the inverse of the current transformation matrix (CTM).

**device\_to\_user\_distance**  $(dx, dy)$

**Parameters**

- **dx** (*float*) – X component of a distance vector
- **dy** (*float*) – Y component of a distance vector

**Returns**  $(dx, dy)$ , both float

**Return type** `tuple`

Transform a distance vector from device space to user space. This function is similar to `Context.device_to_user()` except that the translation components of the inverse CTM will be ignored when transforming  $(dx, dy)$ .

**fill**  $()$

A drawing operator that fills the current path according to the current *fill rule*, (each sub-path is implicitly closed before being filled). After `fill()`, the current path will be cleared from the `Context`. See `Context.set_fill_rule()` and `Context.fill_preserve()`.

**fill\_extents**  $()$

**Returns**  $(x1, y1, x2, y2)$ , all float

**Return type** `tuple`

- *x1*: left of the resulting extents
- *y1*: top of the resulting extents
- *x2*: right of the resulting extents
- *y2*: bottom of the resulting extents

Computes a bounding box in user coordinates covering the area that would be affected, (the “inked” area), by a `Context.fill()` operation given the current path and fill parameters. If the current path is empty, returns an empty rectangle  $(0,0,0,0)$ . Surface dimensions and clipping are not taken into account.

Contrast with `Context.path_extents()`, which is similar, but returns non-zero extents for some paths with no inked area, (such as a simple line segment).

Note that `fill_extents()` must necessarily do more work to compute the precise inked areas in light of the fill rule, so `Context.path_extents()` may be more desirable for sake of performance if the non-inked path extents are desired.

See `Context.fill()`, `Context.set_fill_rule()` and `Context.fill_preserve()`.

**fill\_preserve()**

A drawing operator that fills the current path according to the current *fill rule*, (each sub-path is implicitly closed before being filled). Unlike `Context.fill()`, `fill_preserve()` preserves the path within the *Context*.

See `Context.set_fill_rule()` and `Context.fill()`.

**font\_extents()**

**Returns** (ascent, descent, height, max\_x\_advance, max\_y\_advance), all float

**Return type** tuple

Gets the font extents for the currently selected font.

**get\_antialias()**

**Returns** the current antialias mode, as set by `Context.set_antialias()`.

**Return type** `cairo.Antialias`

**get\_current\_point()**

**Returns** (x, y), both float

**Return type** tuple

- x: X coordinate of the current point
- y: Y coordinate of the current point

Gets the current point of the current path, which is conceptually the final point reached by the path so far.

The current point is returned in the user-space coordinate system. If there is no defined current point or if *Context* is in an error status, x and y will both be set to 0.0. It is possible to check this in advance with `Context.has_current_point()`.

Most path construction functions alter the current point. See the following for details on how they affect the current point: `Context.new_path()`, `Context.new_sub_path()`, `Context.append_path()`, `Context.close_path()`, `Context.move_to()`, `Context.line_to()`, `Context.curve_to()`, `Context.rel_move_to()`, `Context.rel_line_to()`, `Context.rel_curve_to()`, `Context.arc()`, `Context.arc_negative()`, `Context.rectangle()`, `Context.text_path()`, `Context.glyph_path()`, `Context.stroke_to_path()`.

Some functions use and alter the current point but do not otherwise change current path: `Context.show_text()`.

Some functions unset the current path and as a result, current point: `Context.fill()`, `Context.stroke()`.

**get\_dash()**

**Returns** (dashes, offset)

**Return type** tuple

- dashes*: return value as a tuple for the dash array
- offset*: return value as float for the current dash offset

Gets the current dash array.

New in version 1.4.

**get\_dash\_count** ()

**Returns** the length of the dash array, or 0 if no dash array set.

**Return type** int

See also `Context.set_dash()` and `Context.get_dash()`.

New in version 1.4.

**get\_fill\_rule** ()

**Returns** the current fill rule, as set by `Context.set_fill_rule()`.

**Return type** `cairo.FillRule`

**get\_font\_face** ()

**Returns** the current `FontFace` for the `Context`.

**get\_font\_matrix** ()

**Returns** the current `Matrix` for the `Context`.

See `Context.set_font_matrix()`.

**get\_font\_options** ()

**Returns** the current `FontOptions` for the `Context`.

Retrieves font rendering options set via `Context.set_font_options()`. Note that the returned options do not include any options derived from the underlying surface; they are literally the options passed to `Context.set_font_options()`.

**get\_group\_target** ()

**Returns** the target `Surface`.

Gets the current destination `Surface` for the `Context`. This is either the original target surface as passed to `Context` or the target surface for the current group as started by the most recent call to `Context.push_group()` or `Context.push_group_with_content()`.

New in version 1.2.

**get\_line\_cap** ()

**Returns** the current line cap style, as set by `Context.set_line_cap()`.

**Return type** `cairo.LineCap`

**get\_line\_join** ()

**Returns** the current line join style, as set by `Context.set_line_join()`.

**Return type** `cairo.LineJoin`

**get\_line\_width** ()

**Returns** the current line width

**Return type** float

This function returns the current line width value exactly as set by `Context.set_line_width()`. Note that the value is unchanged even if the CTM has changed between the calls to `Context.set_line_width()` and `get_line_width()`.

**get\_matrix()**

**Returns** the current transformation *Matrix* (CTM)

**get\_miter\_limit()**

**Returns** the current miter limit, as set by *Context.set\_miter\_limit()*.

**Return type** float

**get\_operator()**

**Returns** the current compositing operator for a *Context*.

**Return type** *cairo.Operator*

**get\_scaled\_font()**

**Returns** the current *ScaledFont* for a *Context*.

New in version 1.4.

**get\_source()**

**Returns** the current source *Pattern* for a *Context*.

**get\_target()**

**Returns** the target *Surface* for the *Context*

**get\_tolerance()**

**Returns** the current tolerance value, as set by *Context.set\_tolerance()*

**Return type** float

**glyph\_extents(glyphs)**

**Parameters** **glyphs** – glyphs, a sequence of *Glyph*

**Return type** *TextExtents*

Gets the extents for an array of glyphs. The extents describe a user-space rectangle that encloses the “inked” portion of the glyphs, (as they would be drawn by *Context.show\_glyphs()*). Additionally, the *x\_advance* and *y\_advance* values indicate the amount by which the current point would be advanced by *Context.show\_glyphs()*.

Note that whitespace glyphs do not contribute to the size of the rectangle (*extents.width* and *extents.height*).

**glyph\_path(glyphs)**

**Parameters** **glyphs** – glyphs to show, a sequence of *Glyph*

Adds closed paths for the glyphs to the current path. The generated path if filled, achieves an effect similar to that of *Context.show\_glyphs()*.

**has\_current\_point()**

**returns:** True iff a current point is defined on the current path. See *Context.get\_current\_point()* for details on the current point.

New in version 1.6.

**identity\_matrix()**

Resets the current transformation *Matrix* (CTM) by setting it equal to the identity matrix. That is, the user-space and device-space axes will be aligned and one user-space unit will transform to one device-space unit.

**in\_fill(x, y)**

**Parameters**

- **x** (*float*) – X coordinate of the point to test
- **y** (*float*) – Y coordinate of the point to test

**Returns** True iff the point is inside the area that would be affected by a `Context.fill()` operation given the current path and filling parameters. Surface dimensions and clipping are not taken into account.

See `Context.fill()`, `Context.set_fill_rule()` and `Context.fill_preserve()`.

**in\_stroke** (*x*, *y*)

**Parameters**

- **x** (*float*) – X coordinate of the point to test
- **y** (*float*) – Y coordinate of the point to test

**Returns** True iff the point is inside the area that would be affected by a `Context.stroke()` operation given the current path and stroking parameters. Surface dimensions and clipping are not taken into account.

See `Context.stroke()`, `Context.set_line_width()`, `Context.set_line_join()`, `Context.set_line_cap()`, `Context.set_dash()`, and `Context.stroke_preserve()`.

**line\_to** (*x*, *y*)

**Parameters**

- **x** (*float*) – the X coordinate of the end of the new line
- **y** (*float*) – the Y coordinate of the end of the new line

Adds a line to the path from the current point to position (*x*, *y*) in user-space coordinates. After this call the current point will be (*x*, *y*).

If there is no current point before the call to `line_to()` this function will behave as `ctx.move_to(x, y)`.

**mask** (*pattern*)

**Parameters** *pattern* – a *Pattern*

A drawing operator that paints the current source using the alpha channel of *pattern* as a mask. (Opaque areas of *pattern* are painted with the source, transparent areas are not painted.)

**mask\_surface** (*surface*, *x=0.0*, *y=0.0*)

**Parameters**

- **surface** – a *Surface*
- **x** (*float*) – X coordinate at which to place the origin of *surface*
- **y** (*float*) – Y coordinate at which to place the origin of *surface*

A drawing operator that paints the current source using the alpha channel of *surface* as a mask. (Opaque areas of *surface* are painted with the source, transparent areas are not painted.)

**move\_to** (*x*, *y*)

**Parameters**

- **x** (*float*) – the X coordinate of the new position
- **y** (*float*) – the Y coordinate of the new position

Begin a new sub-path. After this call the current point will be  $(x, y)$ .

**new\_path()**

Clears the current path. After this call there will be no path and no current point.

**new\_sub\_path()**

Begin a new sub-path. Note that the existing path is not affected. After this call there will be no current point.

In many cases, this call is not needed since new sub-paths are frequently started with `Context.move_to()`.

A call to `new_sub_path()` is particularly useful when beginning a new sub-path with one of the `Context.arc()` calls. This makes things easier as it is no longer necessary to manually compute the arc's initial coordinates for a call to `Context.move_to()`.

New in version 1.6.

**paint()**

A drawing operator that paints the current source everywhere within the current clip region.

**paint\_with\_alpha(alpha)**

**Parameters** **alpha** (*float*) – alpha value, between 0 (transparent) and 1 (opaque)

A drawing operator that paints the current source everywhere within the current clip region using a mask of constant alpha value *alpha*. The effect is similar to `Context.paint()`, but the drawing is faded out using the alpha value.

**path\_extents()**

**Returns**  $(x1, y1, x2, y2)$ , all float

**Return type** tuple

- x1*: left of the resulting extents
- y1*: top of the resulting extents
- x2*: right of the resulting extents
- y2*: bottom of the resulting extents

Computes a bounding box in user-space coordinates covering the points on the current path. If the current path is empty, returns an empty rectangle  $(0, 0, 0, 0)$ . Stroke parameters, fill rule, surface dimensions and clipping are not taken into account.

Contrast with `Context.fill_extents()` and `Context.stroke_extents()` which return the extents of only the area that would be “inked” by the corresponding drawing operations.

The result of `path_extents()` is defined as equivalent to the limit of `Context.stroke_extents()` with `cairo.LINE_CAP_ROUND` as the line width approaches 0.0, (but never reaching the empty-rectangle returned by `Context.stroke_extents()` for a line width of 0.0).

Specifically, this means that zero-area sub-paths such as `Context.move_to()`; `Context.line_to()` segments, (even degenerate cases where the coordinates to both calls are identical), will be considered as contributing to the extents. However, a lone `Context.move_to()` will not contribute to the results of `Context.path_extents()`.

New in version 1.6.

**pop\_group()**



**Returns** a newly created *SurfacePattern* containing the results of all drawing operations performed to the group.

Terminates the redirection begun by a call to *Context.push\_group()* or *Context.push\_group\_with\_content()* and returns a new pattern containing the results of all drawing operations performed to the group.

The *pop\_group()* function calls *Context.restore()*, (balancing a call to *Context.save()* by the *Context.push\_group()* function), so that any changes to the graphics state will not be visible outside the group.

New in version 1.2.

#### **pop\_group\_to\_source()**

Terminates the redirection begun by a call to *Context.push\_group()* or *Context.push\_group\_with\_content()* and installs the resulting pattern as the source *Pattern* in the given *Context*.

The behavior of this function is equivalent to the sequence of operations:

```
group = cairo_pop_group()
ctx.set_source(group)
```

but is more convenient as there is no need for a variable to store the short-lived pointer to the pattern.

The *Context.pop\_group\_to\_source()* function calls *Context.restore()*, (balancing a call to *Context.save()* by the *Context.push\_group()* function), so that any changes to the graphics state will not be visible outside the group.

New in version 1.2.

#### **push\_group()**

Temporarily redirects drawing to an intermediate surface known as a group. The redirection lasts until the group is completed by a call to *Context.pop\_group()* or *Context.pop\_group\_to\_source()*. These calls provide the result of any drawing to the group as a pattern, (either as an explicit object, or set as the source pattern).

This group functionality can be convenient for performing intermediate compositing. One common use of a group is to render objects as opaque within the group, (so that they occlude each other), and then blend the result with translucence onto the destination.

Groups can be nested arbitrarily deep by making balanced calls to *Context.push\_group()/Context.pop\_group()*. Each call pushes/pops the new target group onto/from a stack.

The *push\_group()* function calls *Context.save()* so that any changes to the graphics state will not be visible outside the group, (the *pop\_group* functions call *Context.restore()*).

By default the intermediate group will have a *cairo.Content* type of *cairo.Content.COLOR\_ALPHA*. Other content types can be chosen for the group by using *Context.push\_group\_with\_content()* instead.

As an example, here is how one might fill and stroke a path with translucence, but without any portion of the fill being visible under the stroke:

```
ctx.push_group()
ctx.set_source(fill_pattern)
ctx.fill_preserve()
ctx.set_source(stroke_pattern)
ctx.stroke()
```

```
ctx.pop_group_to_source()
ctx.paint_with_alpha(alpha)
```

New in version 1.2.

### **push\_group\_with\_content** (*content*)

**Parameters** **content** (`cairo.Content`) – a content indicating the type of group that will be created

Temporarily redirects drawing to an intermediate surface known as a group. The redirection lasts until the group is completed by a call to `Context.pop_group()` or `Context.pop_group_to_source()`. These calls provide the result of any drawing to the group as a pattern, (either as an explicit object, or set as the source pattern).

The group will have a content type of *content*. The ability to control this content type is the only distinction between this function and `Context.push_group()` which you should see for a more detailed description of group rendering.

New in version 1.2.

### **rectangle** (*x*, *y*, *width*, *height*)

#### **Parameters**

- **x** (*float*) – the X coordinate of the top left corner of the rectangle
- **y** (*float*) – the Y coordinate to the top left corner of the rectangle
- **width** (*float*) – the width of the rectangle
- **height** (*float*) – the height of the rectangle

Adds a closed sub-path rectangle of the given size to the current path at position (*x*, *y*) in user-space coordinates.

This function is logically equivalent to:

```
ctx.move_to(x, y)
ctx.rel_line_to(width, 0)
ctx.rel_line_to(0, height)
ctx.rel_line_to(-width, 0)
ctx.close_path()
```

### **rel\_curve\_to** (*dx1*, *dy1*, *dx2*, *dy2*, *dx3*, *dy4*)

#### **Parameters**

- **dx1** (*float*) – the X offset to the first control point
- **dy1** (*float*) – the Y offset to the first control point
- **dx2** (*float*) – the X offset to the second control point
- **dy2** (*float*) – the Y offset to the second control point
- **dx3** (*float*) – the X offset to the end of the curve
- **dy3** (*float*) – the Y offset to the end of the curve

**Raises** `cairo.Error` if called with no current point.

Relative-coordinate version of `Context.curve_to()`. All offsets are relative to the current point. Adds a cubic Bézier spline to the path from the current point to a point offset from the current point by

$(dx3, dy3)$ , using points offset by  $(dx1, dy1)$  and  $(dx2, dy2)$  as the control points. After this call the current point will be offset by  $(dx3, dy3)$ .

Given a current point of  $(x, y)$ , `ctx.rel_curve_to(dx1, dy1, dx2, dy2, dx3, dy3)` is logically equivalent to `ctx.curve_to(x+dx1, y+dy1, x+dx2, y+dy2, x+dx3, y+dy3)`.

**rel\_line\_to** (*dx*, *dy*)

**Parameters**

- **dx** (*float*) – the X offset to the end of the new line
- **dy** (*float*) – the Y offset to the end of the new line

**Raises** `cairo.Error` if called with no current point.

Relative-coordinate version of `Context.line_to()`. Adds a line to the path from the current point to a point that is offset from the current point by  $(dx, dy)$  in user space. After this call the current point will be offset by  $(dx, dy)$ .

Given a current point of  $(x, y)$ , `ctx.rel_line_to(dx, dy)` is logically equivalent to `ctx.line_to(x + dx, y + dy)`.

**rel\_move\_to** (*dx*, *dy*)

**Parameters**

- **dx** (*float*) – the X offset
- **dy** (*float*) – the Y offset

**Raises** `cairo.Error` if called with no current point.

Begin a new sub-path. After this call the current point will offset by  $(dx, dy)$ .

Given a current point of  $(x, y)$ , `ctx.rel_move_to(dx, dy)` is logically equivalent to `ctx.(x + dx, y + dy)`.

**reset\_clip** ()

Reset the current clip region to its original, unrestricted state. That is, set the clip region to an infinitely large shape containing the target surface. Equivalently, if infinity is too hard to grasp, one can imagine the clip region being reset to the exact bounds of the target surface.

Note that code meant to be reusable should not call `reset_clip()` as it will cause results unexpected by higher-level code which calls `clip()`. Consider using `save()` and `restore()` around `clip()` as a more robust means of temporarily restricting the clip region.

**restore** ()

Restores `Context` to the state saved by a preceding call to `save()` and removes that state from the stack of saved states.

**rotate** (*angle*)

**Parameters** **angle** (*float*) – angle (in radians) by which the user-space axes will be rotated

Modifies the current transformation matrix (CTM) by rotating the user-space axes by *angle* radians. The rotation of the axes takes places after any existing transformation of user space. The rotation direction for positive angles is from the positive X axis toward the positive Y axis.

**save** ()

Makes a copy of the current state of `Context` and saves it on an internal stack of saved states. When `restore()` is called, `Context` will be restored to the saved state. Multiple calls to `save()` and `restore()` can be nested; each call to `restore()` restores the state from the matching paired `save()`.

**scale** (*sx*, *sy*)

**Parameters**

- **sx** (*float*) – scale factor for the X dimension
- **sy** (*float*) – scale factor for the Y dimension

Modifies the current transformation matrix (CTM) by scaling the X and Y user-space axes by *sx* and *sy* respectively. The scaling of the axes takes place after any existing transformation of user space.

**select\_font\_face** (*family*[, *slant*[, *weight* ] ])

**Parameters**

- **family** (*text*) – a font family name
- **slant** (*cairo.FontSlant*) – the font slant of the font, defaults to *cairo.FontSlant.NORMAL*.
- **weight** (*cairo.FontWeight*) – the font weight of the font, defaults to *cairo.FontWeight.NORMAL*.

Note: The *select\_font\_face()* function call is part of what the cairo designers call the “toy” text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications.

Selects a family and style of font from a simplified description as a family name, slant and weight. Cairo provides no operation to list available family names on the system (this is a “toy”, remember), but the standard CSS2 generic family names, (“serif”, “sans-serif”, “cursive”, “fantasy”, “monospace”), are likely to work as expected.

For “real” font selection, see the font-backend-specific *font\_face\_create* functions for the font backend you are using. (For example, if you are using the freetype-based *cairo-ft* font backend, see *cairo\_ft\_font\_face\_create\_for\_ft\_face()* or *cairo\_ft\_font\_face\_create\_for\_pattern()*.) The resulting font face could then be used with *cairo\_scaled\_font\_create()* and *cairo\_set\_scaled\_font()*.

Similarly, when using the “real” font support, you can call directly into the underlying font system, (such as *fontconfig* or *freetype*), for operations such as listing available fonts, etc.

It is expected that most applications will need to use a more comprehensive font handling and text layout library, (for example, *pango*), in conjunction with *cairo*.

If text is drawn without a call to *select\_font\_face()*, (nor *set\_font\_face()* nor *set\_scaled\_font()*), the default family is platform-specific, but is essentially “sans-serif”. Default slant is *cairo.FontSlant.NORMAL*, and default weight is *cairo.FontWeight.NORMAL*.

This function is equivalent to a call to *ToyFontFace* followed by *set\_font\_face()*.

**set\_antialias** (*antialias*)

**Parameters** **antialias** (*cairo.Antialias*) – the new antialias mode

Set the antialiasing mode of the rasterizer used for drawing shapes. This value is a hint, and a particular backend may or may not support a particular value. At the current time, no backend supports *cairo.Antialias.SUBPIXEL* when drawing shapes.

Note that this option does not affect text rendering, instead see *FontOptions.set\_antialias()*.

**set\_dash** (*dashes*[, *offset=0* ])

**Parameters**

- **dashes** – a sequence specifying alternate lengths of on and off stroke portions as float.

- **offset** (*int*) – an offset into the dash pattern at which the stroke should start, defaults to 0.

Raises `cairo.Error` if any value in *dashes* is negative, or if all values are 0.

Sets the dash pattern to be used by `stroke()`. A dash pattern is specified by *dashes* - a sequence of positive values. Each value provides the length of alternate “on” and “off” portions of the stroke. The *offset* specifies an offset into the pattern at which the stroke begins.

Each “on” segment will have caps applied as if the segment were a separate sub-path. In particular, it is valid to use an “on” length of 0.0 with `cairo.LineCap.ROUND` or `cairo.LineCap.SQUARE` in order to distributed dots or squares along a path.

Note: The length values are in user-space units as evaluated at the time of stroking. This is not necessarily the same as the user space at the time of `set_dash()`.

If the number of dashes is 0 dashing is disabled.

If the number of dashes is 1 a symmetric pattern is assumed with alternating on and off portions of the size specified by the single value in *dashes*.

**set\_fill\_rule** (*fill\_rule*)

**Parameters** *fill\_rule* (`cairo.FillRule`) – a fill rule to set the within the cairo context. The fill rule is used to determine which regions are inside or outside a complex (potentially self-intersecting) path. The current fill rule affects both `fill()` and `clip()`.

The default fill rule is `cairo.FillRule.WINDING`.

**set\_font\_face** (*font\_face*)

**Parameters** *font\_face* – a `FontFace`, or None to restore to the default `FontFace`

Replaces the current `FontFace` object in the `Context` with *font\_face*.

**set\_font\_matrix** (*matrix*)

**Parameters** *matrix* – a `Matrix` describing a transform to be applied to the current font.

Sets the current font matrix to *matrix*. The font matrix gives a transformation from the design space of the font (in this space, the em-square is 1 unit by 1 unit) to user space. Normally, a simple scale is used (see `set_font_size()`), but a more complex font matrix can be used to shear the font or stretch it unequally along the two axes

**set\_font\_options** (*options*)

**Parameters** *options* – `FontOptions` to use

Sets a set of custom font rendering options for the `Context`. Rendering options are derived by merging these options with the options derived from underlying surface; if the value in *options* has a default value (like `cairo.Antialias.DEFAULT`), then the value from the surface is used.

**set\_font\_size** (*size*)

**Parameters** *size* (*float*) – the new font size, in user space units

Sets the current font matrix to a scale by a factor of *size*, replacing any font matrix previously set with `set_font_size()` or `set_font_matrix()`. This results in a font size of *size* user space units. (More precisely, this matrix will result in the font’s em-square being a *size* by *size* square in user space.)

If text is drawn without a call to `set_font_size()`, (nor `set_font_matrix()` nor `set_scaled_font()`), the default font size is 10.0.

**set\_line\_cap** (*line\_cap*)

**Parameters** *line\_cap* (`cairo.LineCap`) – a line cap style

Sets the current line cap style within the *Context*.

As with the other stroke parameters, the current line cap style is examined by *stroke()*, *stroke\_extents()*, and *stroke\_to\_path()*, but does not have any effect during path construction.

The default line cap style is *cairo.LineCap.BUTT*.

**set\_line\_join** (*line\_join*)

**Parameters** *line\_join* (*cairo.LineJoin*) – a line join style

Sets the current line join style within the *Context*.

As with the other stroke parameters, the current line join style is examined by *stroke()*, *stroke\_extents()*, and *stroke\_to\_path()*, but does not have any effect during path construction.

The default line join style is *cairo.LineJoin.MITER*.

**set\_line\_width** (*width*)

**Parameters** *width* (*float*) – a line width

Sets the current line width within the *Context*. The line width value specifies the diameter of a pen that is circular in user space, (though device-space pen may be an ellipse in general due to scaling/shear/rotation of the CTM).

Note: When the description above refers to user space and CTM it refers to the user space and CTM in effect at the time of the stroking operation, not the user space and CTM in effect at the time of the call to *set\_line\_width()*. The simplest usage makes both of these spaces identical. That is, if there is no change to the CTM between a call to *set\_line\_width()* and the stroking operation, then one can just pass user-space values to *set\_line\_width()* and ignore this note.

As with the other stroke parameters, the current line width is examined by *stroke()*, *stroke\_extents()*, and *stroke\_to\_path()*, but does not have any effect during path construction.

The default line width value is 2.0.

**set\_matrix** (*matrix*)

**Parameters** *matrix* – a transformation *Matrix* from user space to device space.

Modifies the current transformation matrix (CTM) by setting it equal to *matrix*.

**set\_miter\_limit** (*limit*)

**Parameters** *limit* – miter limit to set

Sets the current miter limit within the *Context*.

If the current line join style is set to *cairo.LineJoin.MITER* (see *set\_line\_join()*), the miter limit is used to determine whether the lines should be joined with a bevel instead of a miter. Cairo divides the length of the miter by the line width. If the result is greater than the miter limit, the style is converted to a bevel.

As with the other stroke parameters, the current line miter limit is examined by *stroke()*, *stroke\_extents()*, and *stroke\_to\_path()*, but does not have any effect during path construction.

The default miter limit value is 10.0, which will convert joins with interior angles less than 11 degrees to bevels instead of miters. For reference, a miter limit of 2.0 makes the miter cutoff at 60 degrees, and a miter limit of 1.414 makes the cutoff at 90 degrees.

A miter limit for a desired angle can be computed as:

```
miter limit = 1/math.sin(angle/2)
```

**set\_operator** (*op*)

**Parameters** **op** (`cairo.Operator`) – the compositing operator to set for use in all drawing operations.

The default operator is `cairo.Operator.OVER`.

**set\_scaled\_font** (*scaled\_font*)

**Parameters** **scaled\_font** – a `ScaledFont`

Replaces the current font face, font matrix, and font options in the `Context` with those of the `ScaledFont`. Except for some translation, the current CTM of the `Context` should be the same as that of the `ScaledFont`, which can be accessed using `ScaledFont.get_ctm()`.

New in version 1.2.

**set\_source** (*source*)

**Parameters** **source** – a `Pattern` to be used as the source for subsequent drawing operations.

Sets the source pattern within `Context` to `source`. This pattern will then be used for any subsequent drawing operation until a new source pattern is set.

Note: The pattern's transformation matrix will be locked to the user space in effect at the time of `set_source()`. This means that further modifications of the current transformation matrix will not affect the source pattern. See `Pattern.set_matrix()`.

The default source pattern is a solid pattern that is opaque black, (that is, it is equivalent to `set_source_rgb(0.0, 0.0, 0.0)`).

**set\_source\_rgb** (*red, green, blue*)

**Parameters**

- **red** (*float*) – red component of color
- **green** (*float*) – green component of color
- **blue** (*float*) – blue component of color

Sets the source pattern within `Context` to an opaque color. This opaque color will then be used for any subsequent drawing operation until a new source pattern is set.

The color components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

The default source pattern is opaque black, (that is, it is equivalent to `set_source_rgb(0.0, 0.0, 0.0)`).

**set\_source\_rgba** (*red, green, blue* [, *alpha=1.0* ])

**Parameters**

- **red** (*float*) – red component of color
- **green** (*float*) – green component of color
- **blue** (*float*) – blue component of color
- **alpha** (*float*) – alpha component of color

Sets the source pattern within *Context* to a translucent color. This color will then be used for any subsequent drawing operation until a new source pattern is set.

The color and alpha components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

The default source pattern is opaque black, (that is, it is equivalent to `set_source_rgba(0.0, 0.0, 0.0, 1.0)`).

**set\_source\_surface** (*surface*[, *x*=0.0[, *y*=0.0]])

**Parameters**

- **surface** – a *Surface* to be used to set the source pattern
- **x** (*float*) – User-space X coordinate for surface origin
- **y** (*float*) – User-space Y coordinate for surface origin

This is a convenience function for creating a pattern from a *Surface* and setting it as the source in *Context* with `set_source()`.

The *x* and *y* parameters give the user-space coordinate at which the surface origin should appear. (The surface origin is its upper-left corner before any transformation has been applied.) The *x* and *y* patterns are negated and then set as translation values in the pattern matrix.

Other than the initial translation pattern matrix, as described above, all other pattern attributes, (such as its extend mode), are set to the default values as in *SurfacePattern*. The resulting pattern can be queried with `get_source()` so that these attributes can be modified if desired, (eg. to create a repeating pattern with `Pattern.set_extend()`).

**set\_tolerance** (*tolerance*)

**Parameters** **tolerance** (*float*) – the tolerance, in device units (typically pixels)

Sets the tolerance used when converting paths into trapezoids. Curved segments of the path will be subdivided until the maximum deviation between the original path and the polygonal approximation is less than *tolerance*. The default value is 0.1. A larger value will give better performance, a smaller value, better appearance. (Reducing the value from the default value of 0.1 is unlikely to improve appearance significantly.) The accuracy of paths within Cairo is limited by the precision of its internal arithmetic, and the prescribed *tolerance* is restricted to the smallest representable internal value.

**show\_glyphs** (*glyphs*)

**Parameters** **glyphs** – glyphs to show as a sequence of *Glyph*

A drawing operator that generates the shape from an array of glyphs, rendered according to the current font face, font size (font matrix), and font options.

**show\_page** ()

Emits and clears the current page for backends that support multiple pages. Use `copy_page()` if you don't want to clear the page.

This is a convenience function that simply calls `ctx.get_target() . show_page()`

**show\_text** (*text*)

**Parameters** **text** (*text*) – text

A drawing operator that generates the shape from a string of text, rendered according to the current font\_face, font\_size (font\_matrix), and font\_options.

This function first computes a set of glyphs for the string of text. The first glyph is placed so that its origin is at the current point. The origin of each subsequent glyph is offset from that of the previous glyph by the advance values of the previous glyph.



After this call the current point is moved to the origin of where the next glyph would be placed in this same progression. That is, the current point will be at the origin of the final glyph offset by its advance values. This allows for easy display of a single logical string with multiple calls to `show_text()`.

Note: The `show_text()` function call is part of what the cairo designers call the “toy” text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications. See `show_glyphs()` for the “real” text display API in cairo.

#### **stroke()**

A drawing operator that strokes the current path according to the current line width, line join, line cap, and dash settings. After `stroke()`, the current path will be cleared from the cairo context. See `set_line_width()`, `set_line_join()`, `set_line_cap()`, `set_dash()`, and `stroke_preserve()`.

Note: Degenerate segments and sub-paths are treated specially and provide a useful result. These can result in two different situations:

1. Zero-length “on” segments set in `set_dash()`. If the cap style is `cairo.LineCap.ROUND` or `cairo.LineCap.SQUARE` then these segments will be drawn as circular dots or squares respectively. In the case of `cairo.LineCap.SQUARE`, the orientation of the squares is determined by the direction of the underlying path.
2. A sub-path created by `move_to()` followed by either a `close_path()` or one or more calls to `line_to()` to the same coordinate as the `move_to()`. If the cap style is `cairo.LineCap.ROUND` then these sub-paths will be drawn as circular dots. Note that in the case of `cairo.LineCap.SQUARE` a degenerate sub-path will not be drawn at all, (since the correct orientation is indeterminate).

In no case will a cap style of `cairo.LineCap.BUTT` cause anything to be drawn in the case of either degenerate segments or sub-paths.

#### **stroke\_extents()**

**Returns** (x1, y1, x2, y2), all float

**Return type** tuple

- x1: left of the resulting extents
- y1: top of the resulting extents
- x2: right of the resulting extents
- y2: bottom of the resulting extents

Computes a bounding box in user coordinates covering the area that would be affected, (the “inked” area), by a `stroke()` operation given the current path and stroke parameters. If the current path is empty, returns an empty rectangle (0, 0, 0, 0). Surface dimensions and clipping are not taken into account.

Note that if the line width is set to exactly zero, then `stroke_extents()` will return an empty rectangle. Contrast with `path_extents()` which can be used to compute the non-empty bounds as the line width approaches zero.

Note that `stroke_extents()` must necessarily do more work to compute the precise inked areas in light of the stroke parameters, so `path_extents()` may be more desirable for sake of performance if non-inked path extents are desired.

See `stroke()`, `set_line_width()`, `set_line_join()`, `set_line_cap()`, `set_dash()`, and `stroke_preserve()`.

#### **stroke\_preserve()**

A drawing operator that strokes the current path according to the current line width, line join, line cap, and dash settings. Unlike `stroke()`, `stroke_preserve()` preserves the path within the cairo context.

See `set_line_width()`, `set_line_join()`, `set_line_cap()`, `set_dash()`, and `stroke_preserve()`.

**text\_extents** (*text*)

**Parameters** **text** (*text*) – text to get extents for

**Return type** *TextExtents*

Gets the extents for a string of text. The extents describe a user-space rectangle that encloses the “inked” portion of the text, (as it would be drawn by `Context.show_text()`). Additionally, the `x_advance` and `y_advance` values indicate the amount by which the current point would be advanced by `Context.show_text()`.

Note that whitespace characters do not directly contribute to the size of the rectangle (`extents.width` and `extents.height`). They do contribute indirectly by changing the position of non-whitespace characters. In particular, trailing whitespace characters are likely to not affect the size of the rectangle, though they will affect the `x_advance` and `y_advance` values.

**text\_path** (*text*)

**Parameters** **text** (*text*) – text

Adds closed paths for text to the current path. The generated path if filled, achieves an effect similar to that of `Context.show_text()`.

Text conversion and positioning is done similar to `Context.show_text()`.

Like `Context.show_text()`, After this call the current point is moved to the origin of where the next glyph would be placed in this same progression. That is, the current point will be at the origin of the final glyph offset by its advance values. This allows for chaining multiple calls to `Context.text_path()` without having to set current point in between.

Note: The `text_path()` function call is part of what the cairo designers call the “toy” text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications. See `Context.glyph_path()` for the “real” text path API in cairo.

**transform** (*matrix*)

**Parameters** **matrix** – a transformation *Matrix* to be applied to the user-space axes

Modifies the current transformation matrix (CTM) by applying *matrix* as an additional transformation. The new transformation of user space takes place after any existing transformation.

**translate** (*tx*, *ty*)

**Parameters**

- **tx** (*float*) – amount to translate in the X direction
- **ty** (*float*) – amount to translate in the Y direction

Modifies the current transformation matrix (CTM) by translating the user-space origin by (*tx*, *ty*). This offset is interpreted as a user-space coordinate according to the CTM in place before the new call to `translate()`. In other words, the translation of the user-space origin takes place after any existing transformation.

**user\_to\_device** (*x*, *y*)

**Parameters**

- **x** (*float*) – X value of coordinate
- **y** (*float*) – Y value of coordinate

**Returns** (*x*, *y*), both float

**Return type** `tuple`

- `x`: X value of coordinate
- `y`: Y value of coordinate

Transform a coordinate from user space to device space by multiplying the given point by the current transformation matrix (CTM).

**`user_to_device_distance`** (`dx`, `dy`)

**Parameters**

- `dx` (`float`) – X value of a distance vector
- `dy` (`float`) – Y value of a distance vector

**Returns** (`dx`, `dy`), both float

**Return type** `tuple`

- `dx`: X value of a distance vector
- `dy`: Y value of a distance vector

Transform a distance vector from user space to device space. This function is similar to `Context.user_to_device()` except that the translation components of the CTM will be ignored when transforming (`dx`, `dy`).

**`in_clip`** (`x`, `y`)

**param float x** X coordinate of the point to test

**param float y** Y coordinate of the point to test

**returns** `True` if the point is inside, or `False` if outside.

**rtype** `bool`

Tests whether the given point is inside the area that would be visible through the current clip, i.e. the area that would be filled by a `paint()` operation.

See `clip()`, and `clip_preserve()`.

New in version 1.12.0.

**`show_text_glyphs`** (`utf8`, `glyphs`, `clusters`, `cluster_flags`)

**Parameters**

- `utf8` (`text`) – a string of text
- `glyphs` (`list`) – list of glyphs to show
- `clusters` (`list`) – list of cluster mapping information
- `cluster_flags` (`TextClusterFlags`) – cluster mapping flags

**Raises** `Error` –

New in version 1.15.

This operation has rendering effects similar to `Context.show_glyphs()` but, if the target surface supports it, uses the provided text and cluster mapping to embed the text for the glyphs shown in the output. If the target does not support the extended attributes, this function acts like the basic `Context.show_glyphs()` as if it had been passed `glyphs`.

The mapping between `utf8` and `glyphs` is provided by a list of clusters. Each cluster covers a number of text bytes and `glyphs`, and neighboring clusters cover neighboring areas of `utf8` and `glyphs`. The clusters should collectively cover `utf8` and `glyphs` in entirety.

The first cluster always covers bytes from the beginning of utf8 . If `cluster_flags` do not have the `TextClusterFlags.BACKWARD` set, the first cluster also covers the beginning of glyphs , otherwise it covers the end of the glyphs array and following clusters move backward.

See `TextCluster` for constraints on valid clusters.

`stroke_to_path()`

---

**Note:** This function is not implemented in cairo, but still mentioned in the documentation.

---

## Exceptions

When a cairo function or method call fails an `cairo.Error` exception, or a subclass thereof, is raised.

### `cairo.Error()`

**exception** `cairo.Error`

This exception is raised when a cairo object returns an error status.

**status**

Type `cairo.Status`

**exception** `cairo.CairoError`

An alias for `Error`

New in version 1.12.0.

**exception** `cairo.MemoryError`

**Bases** `Error, MemoryError`

New in version 1.15: Prior to 1.15 `MemoryError` was raised instead of this type.

**exception** `cairo.IOError`

**Bases** `Error, IOError`

New in version 1.15: Prior to 1.15 `IOError` was raised instead of this type.

## Matrix

### `class Matrix()`

*Matrix* is used throughout cairo to convert between different coordinate spaces. A *Matrix* holds an affine transformation, such as a scale, rotation, shear, or a combination of these. The transformation of a point (x,y) is given by:

```
x_new = xx * x + xy * y + x0
y_new = yx * x + yy * y + y0
```

The current transformation matrix of a `Context`, represented as a *Matrix*, defines the transformation from user-space coordinates to device-space coordinates.

Some standard Python operators can be used with matrices:

To read the values from a *Matrix*:

```
xx, yx, xy, yy, x0, y0 = matrix
```

To multiply two matrices:

```
matrix3 = matrix1.multiply(matrix2)
# or equivalently
matrix3 = matrix1 * matrix2
```

To compare two matrices:

```
matrix1 == matrix2
matrix1 != matrix2
```

For more information on matrix transformation see [https://www.cairographics.org/cookbook/matrix\\_transform/](https://www.cairographics.org/cookbook/matrix_transform/)

**class** cairo.**Matrix** (*xx = 1.0, yx = 0.0, xy = 0.0, yy = 1.0, x0 = 0.0, y0 = 0.0*)

#### Parameters

- **xx** (*float*) – xx component of the affine transformation
- **yx** (*float*) – yx component of the affine transformation
- **xy** (*float*) – xy component of the affine transformation
- **yy** (*float*) – yy component of the affine transformation
- **x0** (*float*) – X translation component of the affine transformation
- **y0** (*float*) – Y translation component of the affine transformation

Create a new *Matrix* with the affine transformation given by *xx, yx, xy, yy, x0, y0*. The transformation is given by:

```
x_new = xx * x + xy * y + x0
y_new = yx * x + yy * y + y0
```

To create a new identity matrix:

```
matrix = cairo.Matrix()
```

To create a matrix with a transformation which translates by *tx* and *ty* in the X and Y dimensions, respectively:

```
matrix = cairo.Matrix(x0=tx, y0=ty)
```

To create a matrix with a transformation that scales by *sx* and *sy* in the X and Y dimensions, respectively:

```
matrix = cairo.Matrix(xx=sx, yy=sy)
```

**classmethod** **init\_rotate** (*radians*)

**Parameters** **radians** (*float*) – angle of rotation, in radians. The direction of rotation is defined such that positive angles rotate in the direction from the positive X axis toward the positive Y axis. With the default axis orientation of cairo, positive angles rotate in a clockwise direction.

**Returns** a new *Matrix* set to a transformation that rotates by *radians*.

**invert** ()

**Returns** If *Matrix* has an inverse, modifies *Matrix* to be the inverse matrix and returns *None*

**Raises** `cairo.Error` if the *Matrix* as no inverse

Changes *Matrix* to be the inverse of it's original value. Not all transformation matrices have inverses; if the matrix collapses points together (it is *degenerate*), then it has no inverse and this function will fail.

**multiply** (*matrix2*)

**Parameters** **matrix2** (`cairo.Matrix`) – a second matrix

**Returns** a new *Matrix*

Multiplies the affine transformations in *Matrix* and *matrix2* together. The effect of the resulting transformation is to first apply the transformation in *Matrix* to the coordinates and then apply the transformation in *matrix2* to the coordinates.

It is allowable for result to be identical to either *Matrix* or *matrix2*.

**rotate** (*radians*)

**Parameters** **radians** (`float`) – angle of rotation, in radians. The direction of rotation is defined such that positive angles rotate in the direction from the positive X axis toward the positive Y axis. With the default axis orientation of cairo, positive angles rotate in a clockwise direction.

Initialize *Matrix* to a transformation that rotates by *radians*.

**scale** (*sx*, *sy*)

**Parameters**

- **sx** (`float`) – scale factor in the X direction
- **sy** (`float`) – scale factor in the Y direction

Applies scaling by *sx*, *sy* to the transformation in *Matrix*. The effect of the new transformation is to first scale the coordinates by *sx* and *sy*, then apply the original transformation to the coordinates.

**transform\_distance** (*dx*, *dy*)

**Parameters**

- **dx** (`float`) – X component of a distance vector.
- **dy** (`float`) – Y component of a distance vector.

**Returns** the transformed distance vector (dx,dy), both float

**Return type** tuple

Transforms the distance vector (*dx*,*dy*) by *Matrix*. This is similar to `transform_point()` except that the translation components of the transformation are ignored. The calculation of the returned vector is as follows:

$$\begin{aligned} dx2 &= dx1 * a + dy1 * c \\ dy2 &= dx1 * b + dy1 * d \end{aligned}$$

Affine transformations are position invariant, so the same vector always transforms to the same vector. If (*x1*,*y1*) transforms to (*x2*,*y2*) then (*x1*+*dx1*,*y1*+*dy1*) will transform to (*x1*+*dx2*,*y1*+*dy2*) for all values of *x1* and *x2*.

**transform\_point** (*x*, *y*)

**Parameters**

- **x** (`float`) – X position.
- **y** (`float`) – Y position.

**Returns** the transformed point (x,y), both float

**Return type** tuple

Transforms the point (x, y) by *Matrix*.

**translate** (tx, ty)

**Parameters**

- **tx** (*float*) – amount to translate in the X direction
- **ty** (*float*) – amount to translate in the Y direction

Applies a transformation by *tx*, *ty* to the transformation in *Matrix*. The effect of the new transformation is to first translate the coordinates by *tx* and *ty*, then apply the original transformation to the coordinates.

**xx**

*float*: xx component of the affine transformation

New in version 1.12.0.

**yx**

*float*: yx component of the affine transformation

New in version 1.12.0.

**xy**

*float*: xy component of the affine transformation

New in version 1.12.0.

**yy**

*float*: yy component of the affine transformation

New in version 1.12.0.

**x0**

*float*: X translation component of the affine transformation

New in version 1.12.0.

**y0**

*float*: Y translation component of the affine transformation

New in version 1.12.0.

## Paths

### class Path()

**class** cairo.**Path**

*Path* cannot be instantiated directly, it is created by calling *Context.copy\_path()* and *Context.copy\_path\_flat()*.

str(path) lists the path elements.

See *path attributes*

Path is an iterator.

See examples/warpedtext.py for example usage.

## Patterns

Patterns are the paint with which cairo draws. The primary use of patterns is as the source for all cairo drawing operations, although they can also be used as masks, that is, as the brush too.

A cairo *Pattern* is created by using one of the *PatternType* constructors listed below, or implicitly through *Context.set\_source\_<type>()* methods.

### class Pattern()

*Pattern* is the abstract base class from which all the other pattern classes derive. It cannot be instantiated directly.

**class** `cairo.Pattern`

**get\_extend()**

**Returns** the current extend strategy used for drawing the *Pattern*.

**Return type** *cairo.Extend*

Gets the current extend mode for the *Pattern*. See *cairo.Extend* attributes for details on the semantics of each extend strategy.

**get\_matrix()**

**Returns** a new *Matrix* which stores a copy of the *Pattern*'s transformation matrix

**get\_filter()**

**Returns** the current filter used for resizing the pattern.

**Return type** *cairo.Filter*

New in version 1.12.0: Used to be a method of *SurfacePattern* before

**set\_filter(*filter*)**

**Parameters** **filter** (*cairo.Filter*) – a filter describing the filter to use for resizing the pattern

Note that you might want to control filtering even when you do not have an explicit *Pattern* object, (for example when using *Context.set\_source\_surface()*). In these cases, it is convenient to use *Context.get\_source()* to get access to the pattern that cairo creates implicitly. For example:

```
context.set_source_surface(image, x, y)
surfacepattern.set_filter(context.get_source(), cairo.FILTER_NEAREST)
```

New in version 1.12.0: Used to be a method of *SurfacePattern* before

**set\_extend(*extend*)**

**Parameters** **extend** (*cairo.Extend*) – an extend describing how the area outside of the *Pattern* will be drawn

Sets the mode to be used for drawing outside the area of a *Pattern*.

The default extend mode is *cairo.Extend.NONE* for *SurfacePattern* and *cairo.Extend.PAD* for *Gradient* Patterns.

**set\_matrix(*matrix*)**

**Parameters** **matrix** – a *Matrix*



Sets the *Pattern*'s transformation matrix to *matrix*. This matrix is a transformation from user space to pattern space.

When a *Pattern* is first created it always has the identity matrix for its transformation matrix, which means that pattern space is initially identical to user space.

Important: Please note that the direction of this transformation matrix is from user space to pattern space. This means that if you imagine the flow from a *Pattern* to user space (and on to device space), then coordinates in that flow will be transformed by the inverse of the *Pattern* matrix.

For example, if you want to make a *Pattern* appear twice as large as it does by default the correct code to use is:

```
matrix = cairo.Matrix(xx=0.5, yy=0.5)
pattern.set_matrix(matrix)
```

Meanwhile, using values of 2.0 rather than 0.5 in the code above would cause the *Pattern* to appear at half of its default size.

Also, please note the discussion of the user-space locking semantics of *Context.set\_source*.

## class SolidPattern(Pattern)

```
class cairo.SolidPattern (red, green, blue, alpha=1.0)
```

### Parameters

- **red** (*float*) – red component of the color
- **green** (*float*) – green component of the color
- **blue** (*float*) – blue component of the color
- **alpha** (*float*) – alpha component of the color

**Returns** a new *SolidPattern*

**Raises** *MemoryError* in case of no memory

Creates a new *SolidPattern* corresponding to a translucent color. The color components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

**get\_rgba** ()

**Returns** (red, green, blue, alpha) a tuple of float

Gets the solid color for a *SolidPattern*.

New in version 1.4.

## class SurfacePattern(Pattern)

```
class cairo.SurfacePattern (surface)
```

**Parameters** **surface** – a cairo *Surface*

**Returns** a newly created *SurfacePattern* for the given surface.

**Raises** *MemoryError* in case of no memory.

**get\_surface** ()

**Returns** the *Surface* of the *SurfacePattern*.

New in version 1.4.

## class Gradient(Pattern)

*Gradient* is an abstract base class from which other *Pattern* classes derive. It cannot be instantiated directly.

**class** cairo.**Gradient**

**add\_color\_stop\_rgb** (*offset, red, green, blue*)

### Parameters

- **offset** (*float*) – an offset in the range [0.0 .. 1.0]
- **red** (*float*) – red component of color
- **green** (*float*) – green component of color
- **blue** (*float*) – blue component of color

Adds an opaque color stop to a *Gradient* pattern. The offset specifies the location along the gradient's control vector. For example, a *LinearGradient*'s control vector is from (x0,y0) to (x1,y1) while a *RadialGradient*'s control vector is from any point on the start circle to the corresponding point on the end circle.

The color is specified in the same way as in `Context.set_source_rgb()`.

If two (or more) stops are specified with identical offset values, they will be sorted according to the order in which the stops are added, (stops added earlier will compare less than stops added later). This can be useful for reliably making sharp color transitions instead of the typical blend.

**add\_color\_stop\_rgba** (*offset, red, green, blue, alpha*)

### Parameters

- **offset** (*float*) – an offset in the range [0.0 .. 1.0]
- **red** (*float*) – red component of color
- **green** (*float*) – green component of color
- **blue** (*float*) – blue component of color
- **alpha** (*float*) – alpha component of color

Adds an opaque color stop to a *Gradient* pattern. The offset specifies the location along the gradient's control vector. For example, a *LinearGradient*'s control vector is from (x0,y0) to (x1,y1) while a *RadialGradient*'s control vector is from any point on the start circle to the corresponding point on the end circle.

The color is specified in the same way as in `Context.set_source_rgb()`.

If two (or more) stops are specified with identical offset values, they will be sorted according to the order in which the stops are added, (stops added earlier will compare less than stops added later). This can be useful for reliably making sharp color transitions instead of the typical blend.

**get\_color\_stops\_rgba** ()

**Returns** a list of (offset, red, green, blue, alpha) tuples of float

**Return type** list

Gets the color and offset information for all color stops specified in the given gradient pattern.

New in version 1.14.

## class LinearGradient(Gradient)

`class cairo.LinearGradient (x0, y0, x1, y1)`

### Parameters

- **x0** (*float*) – x coordinate of the start point
- **y0** (*float*) – y coordinate of the start point
- **x1** (*float*) – x coordinate of the end point
- **y1** (*float*) – y coordinate of the end point

**Returns** a new *LinearGradient*

**Raises** *MemoryError* in case of no memory

Create a new *LinearGradient* along the line defined by (x0, y0) and (x1, y1). Before using the *Gradient* pattern, a number of color stops should be defined using *Gradient.add\_color\_stop\_rgb()* or *Gradient.add\_color\_stop\_rgba()*

Note: The coordinates here are in pattern space. For a new *Pattern*, pattern space is identical to user space, but the relationship between the spaces can be changed with *Pattern.set\_matrix()*

**get\_linear\_points ()**

### Returns

(x0, y0, x1, y1) - a tuple of float

- x0: return value for the x coordinate of the first point
- y0: return value for the y coordinate of the first point
- x1: return value for the x coordinate of the second point
- y1: return value for the y coordinate of the second point

Gets the gradient endpoints for a *LinearGradient*.

New in version 1.4.

## class RadialGradient(Gradient)

`class cairo.RadialGradient (cx0, cy0, radius0, cx1, cy1, radius1)`

### Parameters

- **cx0** (*float*) – x coordinate for the center of the start circle
- **cy0** (*float*) – y coordinate for the center of the start circle
- **radius0** (*float*) – radius of the start circle
- **cx1** (*float*) – x coordinate for the center of the end circle
- **cy1** (*float*) – y coordinate for the center of the end circle
- **radius1** (*float*) – radius of the end circle

**Returns** the newly created *RadialGradient*

**Raises** *MemoryError* in case of no memory

Creates a new *RadialGradient* pattern between the two circles defined by (cx0, cy0, radius0) and (cx1, cy1, radius1). Before using the gradient pattern, a number of color stops should be defined using *Gradient.add\_color\_stop\_rgb()* or *Gradient.add\_color\_stop\_rgba()*.

Note: The coordinates here are in pattern space. For a new pattern, pattern space is identical to user space, but the relationship between the spaces can be changed with *Pattern.set\_matrix()*.

**get\_radial\_circles()**

**Returns**

(x0, y0, r0, x1, y1, r1) - a tuple of float

- x0: return value for the x coordinate of the center of the first circle
- y0: return value for the y coordinate of the center of the first circle
- r0: return value for the radius of the first circle
- x1: return value for the x coordinate of the center of the second circle
- y1: return value for the y coordinate of the center of the second circle
- r1: return value for the radius of the second circle

Gets the *Gradient* endpoint circles for a *RadialGradient*, each specified as a center coordinate and a radius.

New in version 1.4.

## class MeshPattern(Pattern)

class cairo.MeshPattern

Raises *Error* -

Return type *MeshPattern*

New in version 1.14.

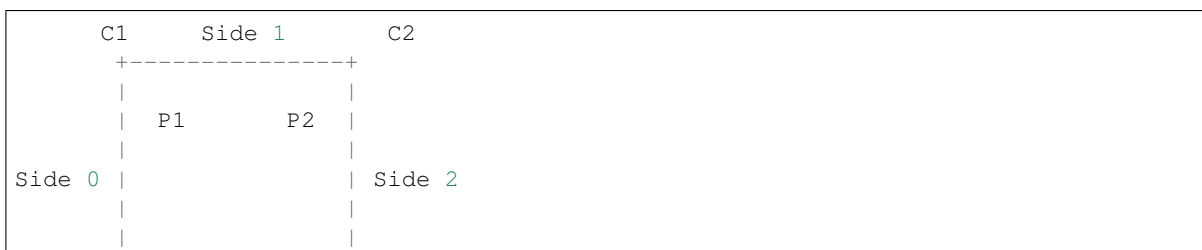
Create a new mesh pattern.

Mesh patterns are tensor-product patch meshes (type 7 shadings in PDF). Mesh patterns may also be used to create other types of shadings that are special cases of tensor-product patch meshes such as Coons patch meshes (type 6 shading in PDF) and Gouraud-shaded triangle meshes (type 4 and 5 shadings in PDF).

Mesh patterns consist of one or more tensor-product patches, which should be defined before using the mesh pattern. Using a mesh pattern with a partially defined patch as source or mask will put the context in an error status with a status of *cairo.Status.INVALID\_MESH\_CONSTRUCTION*.

A tensor-product patch is defined by 4 Bézier curves (side 0, 1, 2, 3) and by 4 additional control points (P0, P1, P2, P3) that provide further control over the patch and complete the definition of the tensor-product patch. The corner C0 is the first point of the patch.

Degenerate sides are permitted so straight lines may be used. A zero length line on one side may be used to create 3 sided patches.





Each patch is constructed by first calling `begin_patch()`, then `move_to()` to specify the first point in the patch (C0). Then the sides are specified with calls to `curve_to()` and `line_to()`.

The four additional control points (P0, P1, P2, P3) in a patch can be specified with `set_control_point()`.

At each corner of the patch (C0, C1, C2, C3) a color may be specified with `set_corner_color_rgb()` or `set_corner_color_rgba()`. Any corner whose color is not explicitly specified defaults to transparent black.

A Coons patch is a special case of the tensor-product patch where the control points are implicitly defined by the sides of the patch. The default value for any control point not specified is the implicit value for a Coons patch, i.e. if no control points are specified the patch is a Coons patch.

A triangle is a special case of the tensor-product patch where the control points are implicitly defined by the sides of the patch, all the sides are lines and one of them has length 0, i.e. if the patch is specified using just 3 lines, it is a triangle. If the corners connected by the 0-length side have the same color, the patch is a Gouraud-shaded triangle.

Patches may be oriented differently to the above diagram. For example the first point could be at the top left. The diagram only shows the relationship between the sides, corners and control points. Regardless of where the first point is located, when specifying colors, corner 0 will always be the first point, corner 1 the point between side 0 and side 1 etc.

Calling `end_patch()` completes the current patch. If less than 4 sides have been defined, the first missing side is defined as a line from the current point to the first point of the patch (C0) and the other sides are degenerate lines from C0 to C0. The corners between the added sides will all be coincident with C0 of the patch and their color will be set to be the same as the color of C0.

Additional patches may be added with additional calls to `begin_patch()/end_patch()`.

```
# Add a Coons patch
pattern = cairo.MeshPattern()
pattern.begin_patch()
pattern.move_to(0, 0)
pattern.curve_to(30, -30, 60, 30, 100, 0)
pattern.curve_to(60, 30, 130, 60, 100, 100)
pattern.curve_to(60, 70, 30, 130, 0, 100)
pattern.curve_to(30, 70, -30, 30, 0, 0)
pattern.set_corner_color_rgb(0, 1, 0, 0)
pattern.set_corner_color_rgb(1, 0, 1, 0)
pattern.set_corner_color_rgb(2, 0, 0, 1)
pattern.set_corner_color_rgb(3, 1, 1, 0)
pattern.end_patch()

# Add a Gouraud-shaded triangle
pattern = cairo.MeshPattern()
pattern.begin_patch()
pattern.move_to(100, 100)
pattern.line_to(130, 130)
pattern.line_to(130, 70)
pattern.set_corner_color_rgb(0, 1, 0, 0)
pattern.set_corner_color_rgb(1, 0, 1, 0)
pattern.set_corner_color_rgb(2, 0, 0, 1)
pattern.end_patch()
```

When two patches overlap, the last one that has been added is drawn over the first one.

When a patch folds over itself, points are sorted depending on their parameter coordinates inside the patch. The *v* coordinate ranges from 0 to 1 when moving from side 3 to side 1; the *u* coordinate ranges from 0 to 1 when going from side 0 to side

Points with higher *v* coordinate hide points with lower *v* coordinate. When two points have the same *v* coordinate, the one with higher *u* coordinate is above. This means that points nearer to side 1 are above points nearer to side 3; when this is not sufficient to decide which point is above (for example when both points belong to side 1 or side 3) points nearer to side 2 are above points nearer to side 0.

For a complete definition of tensor-product patches, see the PDF specification (ISO32000), which describes the parametrization in detail.

Note: The coordinates are always in pattern space. For a new pattern, pattern space is identical to user space, but the relationship between the spaces can be changed with `Pattern.set_matrix()`.

**begin\_patch()**

**Raises *Error*** –

Begin a patch in a mesh pattern.

After calling this function, the patch shape should be defined with `move_to()`, `line_to()` and `curve_to()`.

After defining the patch, `end_patch()` must be called before using pattern as a source or mask.

**curve\_to(x1, y1, x2, y2, x3, y3)**

**Parameters**

- **x1** (*float*) – the X coordinate of the first control point
- **y1** (*float*) – the Y coordinate of the first control point
- **x2** (*float*) – the X coordinate of the second control point
- **y2** (*float*) – the Y coordinate of the second control point
- **x3** (*float*) – the X coordinate of the end of the curve
- **y3** (*float*) – the Y coordinate of the end of the curve

**Raises *Error*** –

Adds a cubic Bézier spline to the current patch from the current point to position (x3 , y3 ) in pattern-space coordinates, using (x1 , y1 ) and (x2 , y2 ) as the control points.

If the current patch has no current point before the call to `curve_to()`, this function will behave as if preceded by a call to `pattern.move_to(x1, y1)`.

After this call the current point will be (x3 , y3 ).

**end\_patch()**

**Raises *Error*** –

Indicates the end of the current patch in a mesh pattern.

If the current patch has less than 4 sides, it is closed with a straight line from the current point to the first point of the patch as if `line_to()` was used.

**get\_control\_point(patch\_num, point\_num)**

**Parameters**

- **patch\_num** (*int*) – the patch number to return data for

- `point_num` (*int*) – the control point number to return data for

**Returns** a (x, y) tuple of float - coordinates of the control point

**Return type** *tuple*

**Raises** *Error* –

Gets the control point `point_num` of patch `patch_num` for a mesh pattern.

`patch_num` can range from 0 to n-1 where n is the number returned by `get_patch_count()`.

Valid values for `point_num` are from 0 to 3 and identify the control points as explained in *MeshPattern*.

`get_corner_color_rgba` (*patch\_num*, *corner\_num*)

**Parameters**

- `patch_num` (*int*) – the patch number to return data for
- `corner_num` (*int*) – the corner number to return data for

**Returns** a (red, green, blue, alpha) tuple of float

**Return type** *tuple*

**Raises** *Error* –

Gets the color information in corner `corner_num` of patch `patch_num` for a mesh pattern.

`patch_num` can range from 0 to n-1 where n is the number returned by `get_patch_count()`.

Valid values for `corner_num` are from 0 to 3 and identify the corners as explained in *MeshPattern*.

`get_patch_count` ()

**Returns** number of patches

**Return type** *int*

Gets the number of patches specified in the given mesh pattern.

The number only includes patches which have been finished by calling `end_patch()`. For example it will be 0 during the definition of the first patch.

`get_path` (*patch\_num*)

**Parameters** `patch_num` (*int*) – the patch number to return data for

**Returns** the path defining the patch

**Return type** *Path*

**Raises** *Error* –

Gets path defining the patch `patch_num` for a mesh pattern.

`patch_num` can range from 0 to n-1 where n is the number returned by `get_patch_count()`.

`line_to` (*x*, *y*)

**Parameters**

- `x` (*float*) – the X coordinate of the end of the new line
- `y` (*float*) – the Y coordinate of the end of the new line

**Raises** *Error* –

Adds a line to the current patch from the current point to position (x , y ) in pattern-space coordinates.

If there is no current point before the call to `line_to()` this function will behave as `pattern.move_to(x , y)`.

After this call the current point will be (x , y).

**move\_to** (x, y)

**Parameters**

- **x** (*float*) – the X coordinate of the new position
- **y** (*float*) – the Y coordinate of the new position

**Raises** *Error* –

Define the first point of the current patch in a mesh pattern.

After this call the current point will be (x , y).

**set\_control\_point** (point\_num, x, y)

**Parameters**

- **point\_num** (*int*) – the control point to set the position for
- **x** (*float*) – the X coordinate of the control point
- **y** (*float*) – the Y coordinate of the control point

**Raises** *Error* –

Set an internal control point of the current patch.

Valid values for point\_num are from 0 to 3 and identify the control points as explained in *MeshPattern*.

**set\_corner\_color\_rgb** (corner\_num, red, green, blue)

**Parameters**

- **corner\_num** (*int*) – the corner to set the color for
- **red** (*float*) – red component of color
- **green** (*float*) – green component of color
- **blue** (*float*) – blue component of color

**Raises** *Error* –

Sets the color of a corner of the current patch in a mesh pattern.

The color is specified in the same way as in `Context.set_source_rgb()`.

Valid values for corner\_num are from 0 to 3 and identify the corners as explained in *MeshPattern*.

**set\_corner\_color\_rgba** (corner\_num, red, green, blue, alpha)

**Parameters**

- **corner\_num** (*int*) – the corner to set the color for
- **red** (*float*) – red component of color
- **green** (*float*) – green component of color
- **blue** (*float*) – blue component of color
- **alpha** (*float*) – alpha component of color



**Raises *Error* –**

Sets the color of a corner of the current patch in a mesh pattern.

The color is specified in the same way as in `Context.set_source_rgba()`.

Valid values for `corner_num` are from 0 to 3 and identify the corners as explained in `MeshPattern`.

**class `RasterSourcePattern(Pattern)`**

The raster source provides the ability to supply arbitrary pixel data whilst rendering. The pixels are queried at the time of rasterisation by means of user callback functions, allowing for the ultimate flexibility. For example, in handling compressed image sources, you may keep a MRU cache of decompressed images and decompress sources on the fly and discard old ones to conserve memory.

For the raster source to be effective, you must at least specify the acquire and release callbacks which are used to retrieve the pixel data for the region of interest and demark when it can be freed afterwards. Other callbacks are provided for when the pattern is copied temporarily during rasterisation, or more permanently as a snapshot in order to keep the pixel data available for printing.

**class** `cairo.RasterSourcePattern` (*content*, *width*, *height*)

**Parameters**

- **content** (`Content`) – content type for the pixel data that will be returned. Knowing the content type ahead of time is used for analysing the operation and picking the appropriate rendering path.
- **width** (`int`) – maximum size of the sample area
- **height** (`int`) – maximum size of the sample area

**Raises *Error* –**

**Return type** `RasterSourcePattern`

Creates a new user pattern for providing pixel data.

Use the setter functions to associate callbacks with the returned pattern.

New in version 1.15.

**set\_acquire** (*acquire*, *release*)

**Parameters**

- **acquire** (`callable`) – acquire callback or `None` to unset it
- **release** (`callable`) – (optional) release callback or `None`

**Raises *Error* –**

Specifies the callbacks used to generate the image surface for a rendering operation (acquire) and the function used to cleanup that surface afterwards.

The acquire callback should create a surface (preferably an image surface created to match the target using `Surface.create_similar_image()`) that defines at least the region of interest specified by extents. The surface is allowed to be the entire sample area, but if it does contain a subsection of the sample area, the surface extents should be provided by setting the device offset (along with its width and height) using `Surface.set_device_offset()`.

**acquire** (*target*, *extents*)

**Parameters**

- **target** (`Surface`) – the rendering target surface

- **extents** (`RectangleInt`) – rectangular region of interest in pixels in sample space

**Return type** `Surface`

This function is called when a pattern is being rendered from. It should create a surface that provides the pixel data for the region of interest as defined by extents, though the surface itself does not have to be limited to that area. For convenience the surface should probably be of image type, created with `Surface.create_similar_image()` for the target (which enables the number of copies to be reduced during transfer to the device). Another option, might be to return a similar surface to the target for explicit handling by the application of a set of cached sources on the device. The region of sample data provided should be defined using `Surface.set_device_offset()` to specify the top-left corner of the sample data (along with width and height of the surface).

**release** (`surface`)

**Parameters** **surface** (`Surface`) – the surface created during acquire

This function is called when the pixel data is no longer being accessed by the pattern for the rendering operation.

New in version 1.15.

**get\_acquire** ()

**Returns** a (acquire, release) tuple of callables or None as set through `set_acquire()`

Queries the current acquire and release callbacks.

New in version 1.15.

## Region

Region — Representing a pixel-aligned area

### class Region()

*Region* is a simple graphical data type representing an area of integer-aligned rectangles. They are often used on raster surfaces to track areas of interest, such as change or clip areas.

**class** `cairo.Region` (`[rectangle_int|rectangle_ints]`)

**Parameters** **rectangle\_int** (`RectangleInt` or `[RectangleInt]`) – a rectangle or a list of rectangle

Allocates a new empty region object or a region object with the containing rectangle(s).

New in version 1.11.0.

**copy** ()

**Returns** A newly allocated *Region*.

**Raises** *Error* – if memory cannot be allocated.

Allocates a new *Region* object copying the area from original.

**get\_extents** ()

**Returns** The bounding rectangle of region

**Return type** `RectangleInt`

**num\_rectangles** ()

**Returns** The number of rectangles contained in region

**Return type** `int`

**get\_rectangle** (*nth*)

**Parameters** **nth** (*int*) – a number indicating which rectangle should be returned

**Returns** The *nth* rectangle from the region

**Return type** *RectangleInt*

**is\_empty** ()

**Returns** Whether region is empty

**Return type** `bool`

**contains\_point** (*x*, *y*)

**Parameters**

- **x** (*int*) – The x coordinate of a point
- **y** (*int*) – The y coordinate of a point

**Returns** Whether (*x*, *y*) is contained in the region

**Return type** `bool`

**contains\_rectangle** (*rectangle*)

**Parameters** **rectangle** (*RectangleInt*) –

**Returns** region overlap

**Return type** *cairo.RegionOverlap*

Checks whether rectangle is inside, outside or partially contained in region

**equal** (*region*)

**Parameters** **region** (*Region*) –

**Returns** Whether both regions contained the same coverage

**Return type** `bool`

**translate** (*dx*, *dy*)

**Parameters**

- **dx** (*int*) – Amount to translate in the x direction
- **dy** (*int*) – Amount to translate in the y direction

Translates region by (*dx*, *dy*).

**intersect** (*other*)

**Parameters** **other** (*Region* or *RectangleInt*) –

**Returns** The intersection of the region and the passed region or rectangle

**Return type** *Region*

**subtract** (*other*)

**Parameters** **other** (*Region* or *RectangleInt*) –

**Returns** The result of the subtraction of the region and the passed region or rectangle

**Return type** *Region*

**union** (*other*)

**Parameters** **other** (*Region* or *RectangleInt*) –

**Returns** The union of the region and the passed region or rectangle

**Return type** *Region*

**xor** (*other*)

**Parameters** **other** (*Region* or *RectangleInt*) –

**Returns** The exclusive difference of the region and the passed region or rectangle

**Return type** *Region*

## class RectangleInt()

*RectangleInt* is a data structure for holding a rectangle with integer coordinates.

**class** `cairo.RectangleInt` (*x=0, y=0, width=0, height=0*)

### Parameters

- **x** (*int*) – X coordinate of the left side of the rectangle
- **y** (*int*) – Y coordinate of the the top side of the rectangle
- **width** (*int*) – width of the rectangle
- **height** (*int*) – height of the rectangle

Allocates a new *RectangleInt* object.

New in version 1.11.0.

```
x
    int
y
    int
width
    int
height
    int
```

## Surfaces

`cairo.Surface` is the abstract type representing all different drawing targets that cairo can render to. The actual drawings are performed using a *Context*.

A `cairo.Surface` is created by using backend-specific constructors of the form `cairo.<XXX>Surface()`.

## class Surface()

**class** `cairo.Surface`

*Surface* is the abstract base class from which all the other surface classes derive. It cannot be instantiated directly.

**copy\_page ()**

Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page. Use `show_page ()` if you want to get an empty page after the emission.

`Context.copy_page ()` is a convenience function for this.

New in version 1.6.

**create\_similar (content, width, height)****Parameters**

- **content** (`cairo.Content`) – the content for the new surface
- **width** (`int`) – width of the new surface, (in device-space units)
- **height** – height of the new surface (in device-space units)

**Returns** a newly allocated *Surface*.

Create a *Surface* that is as compatible as possible with the existing surface. For example the new surface will have the same fallback resolution and *FontOptions*. Generally, the new surface will also use the same backend, unless that is not possible for some reason.

Initially the surface contents are all 0 (transparent if contents have transparency, black otherwise.)

**finish ()**

This method finishes the *Surface* and drops all references to external resources. For example, for the Xlib backend it means that cairo will no longer access the drawable, which can be freed. After calling `finish()` the only valid operations on a *Surface* are flushing and finishing it. Further drawing to the surface will not affect the surface but will instead trigger a `cairo.Error` exception.

**flush ()**

Do any pending drawing for the *Surface* and also restore any temporary modification's cairo has made to the *Surface's* state. This method must be called before switching from drawing on the *Surface* with cairo to drawing on it directly with native APIs. If the *Surface* doesn't support direct access, then this function does nothing.

**get\_content ()**

**Returns** The content type of *Surface*, which indicates whether the *Surface* contains color and/or alpha information.

**Return type** `cairo.Content`

New in version 1.2.

**get\_device\_offset ()****Returns**

(`x_offset`, `y_offset`) a tuple of float

- `x_offset`: the offset in the X direction, in device units
- `y_offset`: the offset in the Y direction, in device units

This method returns the previous device offset set by `set_device_offset ()`.

New in version 1.2.

**get\_fallback\_resolution ()****Returns**

(`x_pixels_per_inch`, `y_pixels_per_inch`) a tuple of float

- `x_pixels_per_inch`: horizontal pixels per inch
- `y_pixels_per_inch`: vertical pixels per inch

This method returns the previous fallback resolution set by `set_fallback_resolution()`, or default fallback resolution if never set.

New in version 1.8.

#### `get_font_options()`

**Returns** a `FontOptions`

Retrieves the default font rendering options for the `Surface`. This allows display surfaces to report the correct subpixel order for rendering on them, print surfaces to disable hinting of metrics and so forth. The result can then be used with `ScaledFont`.

#### `supports_mime_type(mime_type)`

**Parameters** `mime_type` (`str`) – the mime type (`cairo.MIME_TYPE`)

**Returns** `True` if surface supports `mime_type`, `False` otherwise

**Return type** `bool`

Return whether surface supports `mime_type`.

New in version 1.12.0.

#### `set_mime_data(mime_type, data)`

**Parameters**

- `mime_type` (`str`) – the MIME type of the image data (`cairo.MIME_TYPE`)
- `data` (`bytes`) – the image data to attach to the surface

Attach an image in the format `mime_type` to `Surface`. To remove the data from a surface, call this function with same mime type and `None` for data.

The attached image (or filename) data can later be used by backends which support it (currently: PDF, PS, SVG and Win32 Printing surfaces) to emit this data instead of making a snapshot of the surface. This approach tends to be faster and requires less memory and disk space.

The recognized MIME types are listed under `cairo.MIME_TYPE`.

See corresponding backend surface docs for details about which MIME types it can handle. Caution: the associated MIME data will be discarded if you draw on the surface afterwards. Use this function with care.

New in version 1.12.0.

#### `get_mime_data(mime_type)`

**Parameters** `mime_type` (`str`) – the MIME type of the image data (`cairo.MIME_TYPE`)

**Returns** `bytes` or `None`

Return mime data previously attached to surface with `set_mime_data()` using the specified mime type. If no data has been attached with the given mime type, `None` is returned.

New in version 1.12.0.

#### `mark_dirty()`

Tells cairo that drawing has been done to `Surface` using means other than cairo, and that cairo should reread any cached areas. Note that you must call `flush()` before doing such drawing.

#### `mark_dirty_rectangle(x, y, width, height)`

**Parameters**

- **x** (*int*) – X coordinate of dirty rectangle
- **y** (*int*) – Y coordinate of dirty rectangle
- **width** (*int*) – width of dirty rectangle
- **height** (*int*) – height of dirty rectangle

Like `mark_dirty()`, but drawing has been done only to the specified rectangle, so that cairo can retain cached contents for other parts of the surface.

Any cached clip set on the *Surface* will be reset by this function, to make sure that future cairo calls have the clip set that they expect.

**set\_device\_offset** (*x\_offset, y\_offset*)

**Parameters**

- **x\_offset** (*float*) – the offset in the X direction, in device units
- **y\_offset** (*float*) – the offset in the Y direction, in device units

Sets an offset that is added to the device coordinates determined by the CTM when drawing to *Surface*. One use case for this function is when we want to create a *Surface* that redirects drawing for a portion of an onscreen surface to an offscreen surface in a way that is completely invisible to the user of the cairo API. Setting a transformation via `Context.translate()` isn't sufficient to do this, since functions like `Context.device_to_user()` will expose the hidden offset.

Note that the offset affects drawing to the surface as well as using the surface in a source pattern.

**set\_fallback\_resolution** (*x\_pixels\_per\_inch, y\_pixels\_per\_inch*)

**Parameters**

- **x\_pixels\_per\_inch** (*float*) – horizontal setting for pixels per inch
- **y\_pixels\_per\_inch** (*float*) – vertical setting for pixels per inch

Set the horizontal and vertical resolution for image fallbacks.

When certain operations aren't supported natively by a backend, cairo will fallback by rendering operations to an image and then overlaying that image onto the output. For backends that are natively vector-oriented, this function can be used to set the resolution used for these image fallbacks, (larger values will result in more detailed images, but also larger file sizes).

Some examples of natively vector-oriented backends are the ps, pdf, and svg backends.

For backends that are natively raster-oriented, image fallbacks are still possible, but they are always performed at the native device resolution. So this function has no effect on those backends.

Note: The fallback resolution only takes effect at the time of completing a page (with `Context.show_page()` or `Context.copy_page()`) so there is currently no way to have more than one fallback resolution in effect on a single page.

The default fallback resolution is 300 pixels per inch in both dimensions.

New in version 1.2.

**show\_page** ()

Emits and clears the current page for backends that support multiple pages. Use `copy_page()` if you don't want to clear the page.

There is a convenience function for this that takes a `Context.show_page()`.

New in version 1.6.

**write\_to\_png** (*fobj*)

**Parameters** **fobj** (filename (*pathlike*), file or file-like object) – the file to write to

**Raises** *MemoryError* if memory could not be allocated for the operation

*IOError* if an I/O error occurs while attempting to write the file

Writes the contents of *Surface* to *fobj* as a PNG image.

**create\_for\_rectangle** (*x*, *y*, *width*, *height*)

**Parameters**

- **x** (*float*) – the x-origin of the sub-surface from the top-left of the target surface (in device-space units)
- **y** (*float*) – the y-origin of the sub-surface from the top-left of the target surface (in device-space units)
- **width** (*float*) – width of the sub-surface (in device-space units)
- **height** (*float*) – height of the sub-surface (in device-space units)

**Returns** a new surface

**Return type** *cairo.Surface*

Create a new surface that is a rectangle within the target surface. All operations drawn to this surface are then clipped and translated onto the target surface. Nothing drawn via this sub-surface outside of its bounds is drawn onto the target surface, making this a useful method for passing constrained child surfaces to library routines that draw directly onto the parent surface, i.e. with no further backend allocations, double buffering or copies.

---

**Note:** The semantics of subsurfaces have not been finalized yet unless the rectangle is in full device units, is contained within the extents of the target surface, and the target or subsurface’s device transforms are not changed.

---

New in version 1.12.0.

**create\_similar\_image** (*format*, *width*, *height*)

**Parameters**

- **format** (*cairo.Format*) – the format for the new surface
- **width** (*int*) – width of the new surface, (in device-space units)
- **height** (*int*) – height of the new surface, (in device-space units)

**Returns** a new image surface

**Return type** *cairo.ImageSurface*

Create a new image surface that is as compatible as possible for uploading to and the use in conjunction with an existing surface. However, this surface can still be used like any normal image surface.

Initially the surface contents are all 0 (transparent if contents have transparency, black otherwise.)

New in version 1.12.0.

**has\_show\_text\_glyphs** ()

**Returns** *True* if surface supports *Context.show\_text\_glyphs()*, *False* otherwise

**Return type** *bool*



Returns whether the surface supports sophisticated `Context.show_text_glyphs()` operations. That is, whether it actually uses the provided text and cluster data to a `Context.show_text_glyphs()` call.

Note: Even if this function returns `False`, a `Context.show_text_glyphs()` operation targeted at surface will still succeed. It just will act like a `Context.show_glyphs()` operation. Users can use this function to avoid computing UTF-8 text and cluster mapping if the target surface does not use it.

New in version 1.12.0.

**set\_device\_scale** (*x\_scale*, *y\_scale*)

**Parameters**

- **x\_scale** (*float*) – a scale factor in the X direction
- **y\_scale** (*float*) – a scale factor in the Y direction

Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface . One common use for this is to render to very high resolution display devices at a scale factor, so that code that assumes 1 pixel will be a certain size will still work. Setting a transformation via `Context.translate()` isn't sufficient to do this, since functions like `Context.device_to_user()` will expose the hidden scale.

New in version 1.14.0.

**get\_device\_scale** ()

**Returns** (*x\_scale*,*y\_scale*) a 2-tuple of float

This function returns the previous device offset set by `Surface.set_device_scale()`.

New in version 1.14.0.

**get\_device** ()

**Returns** the device or `None` if the surface does not have an associated device

**Return type** *Device*

This function returns the device for a surface.

New in version 1.14.0.

**map\_to\_image** (*extents*)

**Parameters** **extents** (`RectangleInt`) – limit the extraction to an rectangular region or `None` for the whole surface

**Returns** newly allocated image surface

**Return type** *ImageSurface*

**Raises** *Error* –

Returns an image surface that is the most efficient mechanism for modifying the backing store of the target surface.

Note, the use of the original surface as a target or source whilst it is mapped is undefined. The result of mapping the surface multiple times is undefined. Calling `Surface.finish()` on the resulting image surface results in undefined behavior. Changing the device transform of the image surface or of surface before the image surface is unmapped results in undefined behavior.

The caller must use `Surface.unmap_image()` to destroy this image surface.

New in version 1.15.0.

`unmap_image` (*image*)

**Parameters** `image` (`ImageSurface`) – the currently mapped image

Unmaps the image surface as returned from `Surface.map_to_image()`.

The content of the image will be uploaded to the target surface. Afterwards, the image is destroyed.

Using an image surface which wasn't returned by `Surface.map_to_image()` results in undefined behavior.

New in version 1.15.0.

## class `ImageSurface`(`Surface`)

A `cairo.ImageSurface` provides the ability to render to memory buffers either allocated by cairo or by the calling code. The supported image formats are those defined in `cairo.Format`.

class `cairo.ImageSurface` (*format, width, height*)

### Parameters

- **format** (`cairo.Format`) – format of pixels in the surface to create
- **width** – width of the surface, in pixels
- **height** – height of the surface, in pixels

**Returns** a new `ImageSurface`

**Raises** `MemoryError` in case of no memory

Creates an `ImageSurface` of the specified format and dimensions. Initially the surface contents are all 0. (Specifically, within each pixel, each color or alpha channel belonging to format will be 0. The contents of bits within a pixel, but not belonging to the given format are undefined).

classmethod `create_for_data` (*data, format, width, height*[, *stride* ])

### Parameters

- **data** – a writable Python buffer/memoryview object
- **format** (`cairo.Format`) – the format of pixels in the buffer
- **width** – the width of the image to be stored in the buffer
- **height** – the height of the image to be stored in the buffer
- **stride** – the number of bytes between the start of rows in the buffer as allocated. If not given the value from `cairo.Format.stride_for_width()` is used.

**Returns** a new `ImageSurface`

**Raises** `MemoryError` in case of no memory.

`cairo.Error` in case of invalid *stride* value.

Creates an `ImageSurface` for the provided pixel data. The initial contents of buffer will be used as the initial image contents; you must explicitly clear the buffer, using, for example, `cairo_rectangle()` and `cairo_fill()` if you want it cleared.

Note that the *stride* may be larger than `width*bytes_per_pixel` to provide proper alignment for each pixel and row. This alignment is required to allow high-performance rendering within cairo. The correct way to obtain a legal stride value is to call `cairo.Format.stride_for_width()` with the desired format and maximum image width value, and use the resulting stride value to allocate the data and to create the `ImageSurface`. See `cairo.Format.stride_for_width()` for example code.

**classmethod** `create_from_png` (*fobj*)

**Parameters** *fobj* – a *pathlike*, file, or file-like object of the PNG to load.

**Returns** a new *ImageSurface* initialized the contents to the given PNG file.

**static** `format_stride_for_width` (*format*, *width*)

See `cairo.Format.stride_for_width()`.

New in version 1.6.

**get\_data** ()

**Returns** a Python buffer object for the data of the *ImageSurface*, for direct inspection or modification. On Python 3 a memoryview object is returned.

New in version 1.2.

**get\_format** ()

**Returns** the format of the *ImageSurface*.

**Return type** *cairo.Format*

New in version 1.2.

**get\_height** ()

**Returns** the height of the *ImageSurface* in pixels.

**get\_stride** ()

**Returns** the stride of the *ImageSurface* in bytes. The stride is the distance in bytes from the beginning of one row of the image data to the beginning of the next row.

**get\_width** ()

**Returns** the width of the *ImageSurface* in pixels.

## class PDFSurface(Surface)

The PDFSurface is used to render cairo graphics to Adobe PDF files and is a multi-page vector surface backend.

**class** `cairo.PDFSurface` (*fobj*, *width\_in\_points*, *height\_in\_points*)

**Parameters**

- **fobj** (None, *pathlike*, file or file-like object) – a filename or writable file object. None may be used to specify no output. This will generate a *PDFSurface* that may be queried and used as a source, without generating a temporary file.
- **width\_in\_points** (*float*) – width of the surface, in points (1 point == 1/72.0 inch)
- **height\_in\_points** (*float*) – height of the surface, in points (1 point == 1/72.0 inch)

**Returns** a new *PDFSurface* of the specified size in points to be written to *fobj*.

**Raises** *MemoryError* in case of no memory

New in version 1.2.

**set\_size** ()

**Parameters**

- **width\_in\_points** (*float*) – new surface width, in points (1 point == 1/72.0 inch)
- **height\_in\_points** (*float*) – new surface height, in points (1 point == 1/72.0 inch)

Changes the size of a *PDFSurface* for the current (and subsequent) pages.

This function should only be called before any drawing operations have been performed on the current page. The simplest way to do this is to call this function immediately after creating the surface or immediately after completing a page with either `Context.show_page()` or `Context.copy_page()`.

New in version 1.2.

**restrict\_to\_version** (*version*)

**Parameters** *version* – PDF version

Restricts the generated PDF file to version . See `get_versions()` for a list of available version values that can be used here.

This function should only be called before any drawing operations have been performed on the given surface. The simplest way to do this is to call this function immediately after creating the surface.

New in version 1.12.0.

**static get\_versions** ()

**Returns** supported version list

**Return type** list

Retrieve the list of supported versions. See `restrict_to_version()`.

New in version 1.12.0.

**static version\_to\_string** (*version*)

**Parameters** *version* – PDF version

**Returns** the string associated to the given version

**Return type** str

**Raises** `ValueError` – if version isn't valid

Get the string representation of the given version id. See `get_versions()` for a way to get the list of valid version ids.

New in version 1.12.0.

## class `PSSurface(Surface)`

The *PSSurface* is used to render cairo graphics to Adobe PostScript files and is a multi-page vector surface backend.

**class** `cairo.PSSurface` (*fobj*, *width\_in\_points*, *height\_in\_points*)

**Parameters**

- **fobj** (None, *pathlike*, file or file-like object) – a filename or writable file object. None may be used to specify no output. This will generate a *PSSurface* that may be queried and used as a source, without generating a temporary file.
- **width\_in\_points** (*float*) – width of the surface, in points (1 point == 1/72.0 inch)
- **height\_in\_points** (*float*) – height of the surface, in points (1 point == 1/72.0 inch)

**Returns** a new *PDFSurface* of the specified size in points to be written to *fobj*.

**Raises** `MemoryError` in case of no memory

Note that the size of individual pages of the PostScript output can vary. See `set_size()`.

**dsc\_begin\_page\_setup()**

This method indicates that subsequent calls to *dsc\_comment()* should direct comments to the Page-Setup section of the PostScript output.

This method call is only needed for the first page of a surface. It should be called after any call to *dsc\_begin\_setup()* and before any drawing is performed to the surface.

See *dsc\_comment()* for more details.

New in version 1.2.

**dsc\_begin\_setup()**

This function indicates that subsequent calls to *dsc\_comment()* should direct comments to the Setup section of the PostScript output.

This function should be called at most once per surface, and must be called before any call to *dsc\_begin\_page\_setup()* and before any drawing is performed to the surface.

See *dsc\_comment()* for more details.

New in version 1.2.

**dsc\_comment(*comment*)**

**Parameters** **comment** (*str*) – a comment string to be emitted into the PostScript output

Emit a comment into the PostScript output for the given surface.

The comment is expected to conform to the PostScript Language Document Structuring Conventions (DSC). Please see that manual for details on the available comments and their meanings. In particular, the %%IncludeFeature comment allows a device-independent means of controlling printer device features. So the PostScript Printer Description Files Specification will also be a useful reference.

The comment string must begin with a percent character (%) and the total length of the string (including any initial percent characters) must not exceed 255 characters. Violating either of these conditions will place *PSSurface* into an error state. But beyond these two conditions, this function will not enforce conformance of the comment with any particular specification.

The comment string should not have a trailing newline.

The DSC specifies different sections in which particular comments can appear. This function provides for comments to be emitted within three sections: the header, the Setup section, and the PageSetup section. Comments appearing in the first two sections apply to the entire document while comments in the BeginPageSetup section apply only to a single page.

For comments to appear in the header section, this function should be called after the surface is created, but before a call to *dsc\_begin\_setup()*.

For comments to appear in the Setup section, this function should be called after a call to *dsc\_begin\_setup()* but before a call to *dsc\_begin\_page\_setup()*.

For comments to appear in the PageSetup section, this function should be called after a call to *dsc\_begin\_page\_setup()*.

Note that it is only necessary to call *dsc\_begin\_page\_setup()* for the first page of any surface. After a call to *Context.show\_page()* or *Context.copy\_page()* comments are unambiguously directed to the PageSetup section of the current page. But it doesn't hurt to call this function at the beginning of every page as that consistency may make the calling code simpler.

As a final note, cairo automatically generates several comments on its own. As such, applications must not manually generate any of the following comments:

Header section: %!PS-Adobe-3.0, %Creator, %CreationDate, %Pages, %BoundingBox, %Document-Data, %LanguageLevel, %EndComments.

Setup section: %BeginSetup, %EndSetup

PageSetup section: %BeginPageSetup, %PageBoundingBox, %EndPageSetup.

Other sections: %BeginProlog, %EndProlog, %Page, %Trailer, %EOF

Here is an example sequence showing how this function might be used:

```
surface = PSSurface (filename, width, height)
...
surface.dsc_comment (surface, "%%Title: My excellent document")
surface.dsc_comment (surface, "%%Copyright: Copyright (C) 2006 Cairo Lover")
...
surface.dsc_begin_setup (surface)
surface.dsc_comment (surface, "%%IncludeFeature: *MediaColor White")
...
surface.dsc_begin_page_setup (surface)
surface.dsc_comment (surface, "%%IncludeFeature: *PageSize A3")
surface.dsc_comment (surface, "%%IncludeFeature: *InputSlot LargeCapacity")
surface.dsc_comment (surface, "%%IncludeFeature: *MediaType Glossy")
surface.dsc_comment (surface, "%%IncludeFeature: *MediaColor Blue")
... draw to first page here ..
ctx.show_page (cr)
...
surface.dsc_comment (surface, "%%IncludeFeature: PageSize A5");
...
```

New in version 1.2.

**get\_eps ()**

**Returns** True iff the *PSSurface* will output Encapsulated PostScript.

New in version 1.6.

**static level\_to\_string (level)**

**Parameters** **level** (*cairo.PSLevel*) – a PS level

**Returns** the string associated to given level.

**Return type** *str*

Get the string representation of the given *level*. See *get\_levels ()* for a way to get the list of valid level ids.

---

**Note:** Prior to 1.12 this was available under *ps\_level\_to\_string()*

---

New in version 1.12.0.

**ps\_level\_to\_string**

Alias for *level\_to\_string()*

New in version 1.6.

**restrict\_to\_level (level)**

**Parameters** **level** (*cairo.PSLevel*) – a PS level

Restricts the generated PostScript file to *level*. See *get\_levels ()* for a list of available level values that can be used here.

This function should only be called before any drawing operations have been performed on the given surface. The simplest way to do this is to call this function immediately after creating the surface.

New in version 1.6.

**set\_eps** (*eps*)

**Parameters** *eps* (*bool*) – True to output EPS format PostScript

If *eps* is True, the PostScript surface will output Encapsulated PostScript.

This function should only be called before any drawing operations have been performed on the current page. The simplest way to do this is to call this function immediately after creating the surface. An Encapsulated PostScript file should never contain more than one page.

New in version 1.6.

**set\_size** (*width\_in\_points*, *height\_in\_points*)

**Parameters**

- **width\_in\_points** (*float*) – new surface width, in points (1 point == 1/72.0 inch)
- **height\_in\_points** (*float*) – new surface height, in points (1 point == 1/72.0 inch)

Changes the size of a PostScript surface for the current (and subsequent) pages.

This function should only be called before any drawing operations have been performed on the current page. The simplest way to do this is to call this function immediately after creating the surface or immediately after completing a page with either `Context.show_page()` or `Context.copy_page()`.

New in version 1.2.

**static get\_levels** ()

**Returns** supported level list

**Return type** *list*

Retrieve the list of supported levels. See `restrict_to_level()`.

New in version 1.12.0.

## class RecordingSurface(Surface)

A *RecordingSurface* is a surface that records all drawing operations at the highest level of the surface backend interface, (that is, the level of paint, mask, stroke, fill, and `show_text_glyphs`). The recording surface can then be “replayed” against any target surface by using it as a source surface.

If you want to replay a surface so that the results in target will be identical to the results that would have been obtained if the original operations applied to the recording surface had instead been applied to the target surface, you can use code like this:

```
cr = cairo.Context(target)
cr.set_source_surface(recording_surface, 0.0, 0.0)
cr.paint()
```

A *RecordingSurface* is logically unbounded, i.e. it has no implicit constraint on the size of the drawing surface. However, in practice this is rarely useful as you wish to replay against a particular target surface with known bounds. For this case, it is more efficient to specify the target extents to the recording surface upon creation.

The recording phase of the recording surface is careful to snapshot all necessary objects (paths, patterns, etc.), in order to achieve accurate replay.

**class** `cairo.RecordingSurface` (*content, rectangle*)

**Parameters**

- **content** (`cairo.Content`) – the content for the new surface
- **rectangle** (`cairo.Rectangle`) – or `None` to record unbounded operations.

**Returns** a new *RecordingSurface*

Creates a *RecordingSurface* which can be used to record all drawing operations at the highest level (that is, the level of paint, mask, stroke, fill and `show_text_glyphs`). The *RecordingSurface* can then be “replayed” against any target surface by using it as a source to drawing operations.

The recording phase of the *RecordingSurface* is careful to snapshot all necessary objects (paths, patterns, etc.), in order to achieve accurate replay.

New in version 1.11.0.

**ink\_extents** ()

:::rtype: (x0,y0,width,height) a 4-tuple of float

- **x0**: the x-coordinate of the top-left of the ink bounding box
- **y0**: the y-coordinate of the top-left of the ink bounding box
- **width**: the width of the ink bounding box
- **height**: the height of the ink bounding box

Measures the extents of the operations stored within the *RecordingSurface*. This is useful to compute the required size of an *ImageSurface* (or equivalent) into which to replay the full sequence of drawing operations.

New in version 1.11.0.

**get\_extents** ()

**Returns** a rectangle or `None` if the surface is unbounded.

**Return type** *Rectangle*

Get the extents of the recording-surface.

New in version 1.12.0.

## **class** `SVGSurface`(`Surface`)

The *SVGSurface* is used to render cairo graphics to SVG files and is a multi-page vector surface backend

**class** `cairo.SVGSurface` (*fobj, width\_in\_points, height\_in\_points*)

**Parameters**

- **fobj** (`None`, *pathlike*, file or file-like object) – a filename or writable file object. `None` may be used to specify no output. This will generate a *SVGSurface* that may be queried and used as a source, without generating a temporary file.
- **width\_in\_points** (*float*) – width of the surface, in points (1 point == 1/72.0 inch)
- **height\_in\_points** (*float*) – height of the surface, in points (1 point == 1/72.0 inch)

**Returns** a new *SVGSurface* of the specified size in points to be written to *fobj*.

**Raises** *MemoryError* in case of no memory

**restrict\_to\_version** (*version*)



**Parameters `version`** – SVG version

Restricts the generated SVG file to version . See `get_versions()` for a list of available version values that can be used here.

This function should only be called before any drawing operations have been performed on the given surface. The simplest way to do this is to call this function immediately after creating the surface.

New in version 1.12.0.

**static `get_versions()`**

**Returns** supported version list

**Return type** `list`

Retrieve the list of supported versions. See `restrict_to_version()`.

New in version 1.12.0.

**static `version_to_string(version)`**

**Parameters `version`** – SVG version

**Returns** the string associated to the given version

**Return type** `str`

**Raises `ValueError`** – if version isn't valid

Get the string representation of the given version id. See `get_versions()` for a way to get the list of valid version ids.

New in version 1.12.0.

**class `Win32Surface(Surface)`**

The Microsoft Windows surface is used to render cairo graphics to Microsoft Windows windows, bitmaps, and printing device contexts.

**class `cairo.Win32Surface(hdc)`**

**Parameters `hdc` (`int`)** – the DC to create a surface for

**Returns** the newly created surface

Creates a cairo surface that targets the given DC. The DC will be queried for its initial clip extents, and this will be used as the size of the cairo surface. The resulting surface will always be of format `cairo.FORMAT_RGB24`, see `cairo.Format`.

**class `Win32PrintingSurface(Surface)`**

The `Win32PrintingSurface` is a multi-page vector surface type.

**class `cairo.Win32PrintingSurface(hdc)`**

**Parameters `hdc` (`int`)** – the DC to create a surface for

**Returns** the newly created surface

Creates a cairo surface that targets the given DC. The DC will be queried for its initial clip extents, and this will be used as the size of the cairo surface. The DC should be a printing DC; antialiasing will be ignored, and GDI will be used as much as possible to draw to the surface.

The returned surface will be wrapped using the paginated surface to provide correct complex rendering behaviour; `cairo.Surface.show_page()` and associated methods must be used for correct output.

### class `XCBSurface(Surface)`

The XCB surface is used to render cairo graphics to X Window System windows and pixmaps using the XCB library. Note that the XCB surface automatically takes advantage of the X render extension if it is available.

**class** `cairo.XCBSurface`

#### Parameters

- **connection** – an XCB connection
- **drawable** – a X drawable
- **visualtype** – a X visualtype
- **width** – The surface width
- **height** – The surface height

Creates a cairo surface that targets the given drawable (pixmap or window).

---

**Note:** This methods works using xpyb.

---

**set\_size** (*width, height*)

#### Parameters

- **width** – The width of the surface
- **height** – The height of the surface

Informs cairo of the new size of the X Drawable underlying the surface. For a surface created for a Window (rather than a Pixmap), this function must be called each time the size of the window changes. (For a sub-window, you are normally resizing the window yourself, but for a toplevel window, it is necessary to listen for `ConfigureNotify` events.)

A Pixmap can never change size, so it is never necessary to call this function on a surface created for a Pixmap.

### class `XlibSurface(Surface)`

The XLib surface is used to render cairo graphics to X Window System windows and pixmaps using the XLib library. Note that the XLib surface automatically takes advantage of X render extension if it is available.

**class** `cairo.XlibSurface`

---

**Note:** *XlibSurface* cannot be instantiated directly because Python interaction with Xlib would require open source Python bindings to Xlib which provided a C API. However, an *XlibSurface* instance can be returned from a function call when using pygtk <http://www.pygtk.org/>.

---

**get\_depth** ()

**Returns** the number of bits used to represent each pixel value.

New in version 1.2.

**get\_height** ()

**Returns** the height of the X Drawable underlying the surface in pixels.

New in version 1.2.

**get\_width** ()

**Returns** the width of the X Drawable underlying the surface in pixels.

New in version 1.2.

## class ScriptSurface(Surface)

The script surface provides the ability to render to a native script that matches the cairo drawing model. The scripts can be replayed using tools under the util/cairo-script directory, or with cairo-perf-trace.

**class** `cairo.ScriptSurface` (*script, content, width, height*)

**Parameters**

- **script** (`cairo.ScriptDevice`) – the script (output device)
- **content** (`cairo.Content`) – the content of the surface
- **width** (*float*) – width in pixels
- **height** (*float*) – height in pixels

**Return type** `cairo.ScriptSurface`

**Raises** `cairo.Error` –

Create a new surface that will emit its rendering through `script`.

New in version 1.14.

**classmethod** `create_for_target` (*script, target*)

**Parameters**

- **script** (`cairo.ScriptDevice`) – the script (output device)
- **target** (`cairo.Surface`) – a target surface to wrap

**Return type** `cairo.ScriptSurface`

**Raises** `cairo.Error` –

Create a proxy surface that will render to `target` and record the operations to device.

New in version 1.14.

## class TeeSurface(Surface)

This surface supports redirecting all its input to multiple surfaces.

**class** `cairo.TeeSurface` (*master*)

**Parameters** **master** (`cairo.Surface`) –

**Return type** `cairo.TeeSurface`

**Raises** `cairo.Error` –

New in version 1.14.

**add** (*target*)

**Parameters** *target* (`cairo.Surface`) –

**Raises** `cairo.Error` –

Add the surface

New in version 1.14.

**remove** (*target*)

**Parameters** *target* (`cairo.Surface`) –

**Raises** `cairo.Error` –

Remove the surface

New in version 1.14.

**index** (*index*)

**Parameters** *index* (`int`) –

**Return type** `cairo.Surface`

**Raises** `cairo.Error` –

Returns the surface at index *index*. The master surface is at index 0.

New in version 1.14.

## Text

Cairo has two sets of text rendering capabilities:

- The functions with text in their name form cairo’s toy text API. The toy API takes UTF-8 encoded text and is limited in its functionality to rendering simple left-to-right text with no advanced features. That means for example that most complex scripts like Hebrew, Arabic, and Indic scripts are out of question. No kerning or correct positioning of diacritical marks either. The font selection is pretty limited too and doesn’t handle the case that the selected font does not cover the characters in the text. This set of functions are really that, a toy text API, for testing and demonstration purposes. Any serious application should avoid them.
- The functions with glyphs in their name form cairo’s low-level text API. The low-level API relies on the user to convert text to a set of glyph indexes and positions. This is a very hard problem and is best handled by external libraries, like the pangocairo that is part of the Pango text layout and rendering library. Pango is available from <http://www.pango.org/>.

## class `FontFace()`

A `cairo.FontFace` specifies all aspects of a font other than the size or font matrix (a font matrix is used to distort a font by sheering it or scaling it unequally in the two directions). A `FontFace` can be set on a `Context` by using `Context.set_font_face()` the size and font matrix are set with `Context.set_font_size()` and `Context.set_font_matrix()`.

There are various types of `FontFace`, depending on the font backend they use.

---

**class** `cairo.FontFace`

---

**Note:** This class cannot be instantiated directly, it is returned by `Context.get_font_face()`.

---

## **class** `FreeTypeFontFace(FontFace)`

FreeType Fonts - Font support for FreeType.

The FreeType font backend is primarily used to render text on GNU/Linux systems, but can be used on other platforms too.

---

**Note:** FreeType Fonts are not implemented in pycairo because there is no open source Python bindings to FreeType (and fontconfig) that provides a C API. This a possible project idea for anyone interested in adding FreeType support to pycairo.

---

## **class** `ToyFontFace(FontFace)`

The `cairo.ToyFontFace` class can be used instead of `Context.select_font_face()` to create a toy font independently of a context.

**class** `cairo.ToyFontFace` (*family*[, *slant*[, *weight* ]])

### **Parameters**

- **family** (*text*) – a font family name
- **slant** (`cairo.FontSlant`) – the font slant of the font, defaults to `cairo.FontSlant.NORMAL`.
- **weight** (`cairo.FontWeight`) – the font weight of the font, defaults to `cairo.FontWeight.NORMAL`.

**Returns** a new `ToyFontFace`

Creates a `ToyFontFace` from a triplet of family, slant, and weight. These font faces are used in implementation of the the “toy” font API.

If family is the zero-length string “”, the platform-specific default family is assumed. The default family then can be queried using `get_family()`.

The `Context.select_font_face()` method uses this to create font faces. See that function for limitations of toy font faces.

New in version 1.8.4.

**get\_family()**

**Returns** the family name of a toy font

**Return type** `str`

New in version 1.8.4.

**get\_slant()**

**Returns** the font slant value

**Return type** *cairo.FontSlant*

New in version 1.8.4.

**get\_weight** ()

**Returns** the font weight value

**Return type** *cairo.FontWeight*

New in version 1.8.4.

## class UserFontFace(FontFace)

The user-font feature allows the cairo user to provide drawings for glyphs in a font. This is most useful in implementing fonts in non-standard formats, like SVG fonts and Flash fonts, but can also be used by games and other application to draw “funky” fonts.

---

**Note:** UserFontFace support has not (yet) been added to pycairo. If you need this feature in pycairo register your interest by sending a message to the cairo mailing list, or by opening a pycairo bug report.

---

## class ScaledFont()

A *ScaledFont* is a font scaled to a particular size and device resolution. A *ScaledFont* is most useful for low-level font usage where a library or application wants to cache a reference to a scaled font to speed up the computation of metrics.

There are various types of scaled fonts, depending on the font backend they use.

**class** `cairo.ScaledFont` (*font\_face, font\_matrix, ctm, options*)

### Parameters

- **font\_face** – a *FontFace* instance
- **font\_matrix** – font space to user space transformation *Matrix* for the font. In the simplest case of a N point font, this matrix is just a scale by N, but it can also be used to shear the font or stretch it unequally along the two axes. See *Context.set\_font\_matrix()*.
- **ctm** – user to device transformation *Matrix* with which the font will be used.
- **options** – a *FontOptions* instance to use when getting metrics for the font and rendering with it.

Creates a *ScaledFont* object from a *FontFace* and matrices that describe the size of the font and the environment in which it will be used.

**extents** ()

**Returns** (ascent, descent, height, max\_x\_advance, max\_y\_advance), a tuple of float values.

Gets the metrics for a *ScaledFont*.

**get\_ctm** ()

**Returns** the CTM

**Return type** *cairo.Matrix*

Returns the CTM with which `scaled_font` was created into `ctm`. Note that the translation offsets (`x0`, `y0`) of the CTM are ignored by `ScaledFont()`. So, the matrix this function returns always has 0, 0 as `x0`, `y0`.

New in version 1.12.0.

**get\_font\_face()**

**Returns** the *FontFace* that this *ScaledFont* was created for.

New in version 1.2.

**get\_font\_matrix()**

**Returns** the matrix

**Return type** *cairo.Matrix*

Returns the font matrix with which `scaled_font` was created.

New in version 1.12.0.

**get\_font\_options()**

**Returns** font options

**Return type** *cairo.FontOptions*

Returns the font options with which `scaled_font` was created.

New in version 1.12.0.

**get\_scale\_matrix()**

**Returns** the scale *Matrix*

The scale matrix is product of the font matrix and the `ctm` associated with the scaled font, and hence is the matrix mapping from font space to device space.

New in version 1.8.

**glyph\_extents(glyphs)**

**Parameters** **glyphs** – glyphs, a sequence of *Glyph*

**Return type** *TextExtents*

New in version 1.15.

Gets the extents for a list of glyphs. The extents describe a user-space rectangle that encloses the “inked” portion of the glyphs, (as they would be drawn by `Context.show_glyphs()` if the cairo graphics state were set to the same `font_face`, `font_matrix`, `ctm`, and `font_options` as `scaled_font`). Additionally, the `x_advance` and `y_advance` values indicate the amount by which the current point would be advanced by `cairo_show_glyphs()`.

Note that whitespace glyphs do not contribute to the size of the rectangle (`extents.width` and `extents.height`).

**text\_extents(text)**

**Parameters** **text** (`text`) – text

**Return type** *TextExtents*

Gets the extents for a string of text. The extents describe a user-space rectangle that encloses the “inked” portion of the text drawn at the origin (0,0) (as it would be drawn by `Context.show_text()` if the cairo graphics state were set to the same `font_face`, `font_matrix`, `ctm`, and `font_options` as *ScaledFont*). Additionally, the `x_advance` and `y_advance` values indicate the amount by which the current point would be advanced by `Context.show_text()`.

Note that whitespace characters do not directly contribute to the size of the rectangle (width and height). They do contribute indirectly by changing the position of non-whitespace characters. In particular, trailing whitespace characters are likely to not affect the size of the rectangle, though they will affect the `x_advance` and `y_advance` values.

New in version 1.2.

**text\_to\_glyphs** (*x*, *y*, *utf8*[, *with\_clusters=True* ])

**Parameters**

- **x** (*float*) – X position to place first glyph
- **y** (*float*) – Y position to place first glyph
- **utf8** (*text*) – a string of text
- **with\_clusters** (*bool*) – If `False` only the glyph list will computed and returned

**Returns** a tuple of (*[Glyph]*, *[TextCluster]*, *TextClusterFlags*)

**Return type** *tuple*

**Raises** *Error* –

New in version 1.15.

Converts UTF-8 text to a list of glyphs, with cluster mapping, that can be used to render later.

For details of how clusters, and `cluster_flags` map input UTF-8 text to the output glyphs see `Context.show_text_glyphs()`.

The output values can be readily passed to `Context.show_text_glyphs()` `Context.show_glyphs()`, or related functions, assuming that the exact same scaled font is used for the operation.

## class FontOptions()

An opaque structure holding all options that are used when rendering fonts.

Individual features of a `FontOptions` can be set or accessed using functions named `FontOptions.set_<feature_name>` and `FontOptions.get_<feature_name>`, like `FontOptions.set_antialias()` and `FontOptions.get_antialias()`.

New features may be added to a `FontOptions` in the future. For this reason, `FontOptions.copy()`, `FontOptions.equal()`, `FontOptions.merge()`, and `FontOptions.hash()` should be used to copy, check for equality, merge, or compute a hash value of `FontOptions` objects.

**class** `cairo.FontOptions`

**Returns** a newly allocated `FontOptions`.

Allocates a new `FontOptions` object with all options initialized to default values.

Implements `__eq__` and `__ne__` using `equal()` since 1.12.0.

**get\_antialias** ()

**Returns** the antialias mode for the `FontOptions` object

**Return type** `cairo.Antialias`

**get\_hint\_metrics** ()

**Returns** the hint metrics mode for the `FontOptions` object



**Return type** *cairo.HintMetrics*

**get\_hint\_style** ()

**Returns** the hint style for the *FontOptions* object

**Return type** *cairo.HintStyle*

**get\_subpixel\_order** ()

**Returns** the subpixel order for the *FontOptions* object

**Return type** *cairo.SubpixelOrder*

**set\_antialias** (*antialias*)

**Parameters** **antialias** (*cairo.Antialias*) – the antialias mode

This specifies the type of antialiasing to do when rendering text.

**set\_hint\_metrics** (*hint\_metrics*)

**Parameters** **hint\_metrics** (*cairo.HintMetrics*) – the hint metrics mode

This controls whether metrics are quantized to integer values in device units.

**set\_hint\_style** (*hint\_style*)

**Parameters** **hint\_style** (*cairo.HintStyle*) – the hint style

This controls whether to fit font outlines to the pixel grid, and if so, whether to optimize for fidelity or contrast.

**set\_subpixel\_order** (*subpixel\_order*)

**Parameters** **subpixel\_order** (*cairo.SubpixelOrder*) – the subpixel order

The subpixel order specifies the order of color elements within each pixel on the display device when rendering with an antialiasing mode of *cairo.Antialias.SUBPIXEL*.

**merge** (*other*)

**Parameters** **other** (*FontOptions*) – another *FontOptions*

Merges non-default options from *other* into options, replacing existing values. This operation can be thought of as somewhat similar to compositing *other* onto options with the operation of *Operator.OVER*.

New in version 1.12.0.

**copy** ()

**Returns** a new *FontOptions*

Returns a new font options object copying the option values from original.

New in version 1.12.0.

**hash** ()

**Returns** the hash value for the font options object

**Return type** *int*

Compute a hash for the font options object; this value will be useful when storing an object containing a *FontOptions* in a hash table.

New in version 1.12.0.

**equal** (*other*)

**Parameters** `other` (`FontOptions`) – another `FontOptions`

**Returns** `True` if all fields of the two font options objects match. Note that this function will return `False` if either object is in error.

**Return type** `bool`

Compares two font options objects for equality.

New in version 1.12.0.

## Devices

### class `Device()`

**class** `cairo.Device`

A `Device` represents the driver interface for drawing operations to a `Surface`.

New in version 1.14.

**finish** ()

This function finishes the device and drops all references to external resources. All surfaces, fonts and other objects created for this device will be finished, too. Further operations on the device will not affect the device but will instead trigger a `Status.DEVICE_FINISHED` error.

This function may acquire devices.

New in version 1.14.

**flush** ()

Finish any pending operations for the device and also restore any temporary modifications cairo has made to the device's state. This function must be called before switching from using the device with Cairo to operating on it directly with native APIs. If the device doesn't support direct access, then this function does nothing.

This function may acquire devices.

New in version 1.14.

**acquire** ()

**Raises** `cairo.Error` – If the device is in an error state and could not be acquired.

Acquires the device for the current thread. This function will block until no other thread has acquired the device.

If the does not raise, you successfully acquired the device. From now on your thread owns the device and no other thread will be able to acquire it until a matching call to `release()`. It is allowed to recursively acquire the device multiple times from the same thread.

After a successful call to `acquire()`, a matching call to `release()` is required.

---

**Note:** You must never acquire two different devices at the same time unless this is explicitly allowed. Otherwise the possibility of deadlocks exist. As various Cairo functions can acquire devices when called, these functions may also cause deadlocks when you call them with an acquired device. So you must not have a device acquired when calling them. These functions are marked in the documentation.

---

New in version 1.14.

**release ()**

Releases a device previously acquired using `acquire ()`. See that function for details.

New in version 1.14.

## class ScriptDevice(Device)

class `cairo.ScriptDevice (fobj)`

**Parameters** `fobj` (*pathlike*, file or file-like object) – a filename or writable file object.

Creates a output device for emitting the script, used when creating the individual surfaces.

New in version 1.14.

**set\_mode (mode)**

**Parameters** `mode` (`cairo.ScriptMode`) – the new mode

Change the output mode of the script

**get\_mode ()**

**Returns** the current output mode of the script

**Return type** `cairo.ScriptMode`

Queries the script for its current output mode.

**write\_comment (comment)**

**Parameters** `comment` (*text*) – the string to emit

Emit a string verbatim into the script.

**from\_recording\_surface (recording\_surface)**

**Parameters** `recording_surface` (`cairo.RecordingSurface`) – the recording surface to replay

**Raises** `cairo.Error` –

Converts the record operations in `recording_surface` into a script.

## Glyph

### class Glyph(tuple)

class `cairo.Glyph (index, x, y)`

**Parameters**

- **index** (*int*) – glyph index in the font. The exact interpretation of the glyph index depends on the font technology being used.
- **x** (*float*) – the offset in the X direction between the origin used for drawing or measuring the string and the origin of this glyph.
- **y** (*float*) – the offset in the Y direction between the origin used for drawing or measuring the string and the origin of this glyph.

**Return type** `Glyph`

New in version 1.15: In prior versions a (int, float, float) tuple was used instead of *Glyph*.

The *Glyph* holds information about a single glyph when drawing or measuring text. A font is (in simple terms) a collection of shapes used to draw text. A glyph is one of these shapes. There can be multiple glyphs for a single character (alternates to be used in different contexts, for example), or a glyph can be a ligature of multiple characters. Cairo doesn't expose any way of converting input text into glyphs, so in order to use the Cairo interfaces that take arrays of glyphs, you must directly access the appropriate underlying font system.

Note that the offsets given by *x* and *y* are not cumulative. When drawing or measuring text, each glyph is individually positioned with respect to the overall origin

```
index
    int
x
    float
y
    float
```

## Rectangle

### class Rectangle(tuple)

```
class cairo.Rectangle (x, y, width, height)
```

#### Parameters

- **x** (*float*) – X coordinate of the left side of the rectangle
- **y** (*float*) – Y coordinate of the the top side of the rectangle
- **width** (*float*) – width of the rectangle
- **height** (*float*) – height of the rectangle

#### Return type *Rectangle*

New in version 1.15: In prior versions a (float, float, float, float) tuple was used instead of *Rectangle*.

A data structure for holding a rectangle.

```
x
    float
y
    float
width
    float
height
    float
```

## Text Cluster

### class TextCluster(tuple)

```
class cairo.TextCluster (num_bytes, num_glyphs)
```

**Parameters**

- **num\_bytes** (*int*) – the number of bytes of UTF-8 text covered by cluster
- **num\_glyphs** (*int*) – the number of glyphs covered by cluster

New in version 1.15.

The *TextCluster* structure holds information about a single text cluster. A text cluster is a minimal mapping of some glyphs corresponding to some UTF-8 text.

For a cluster to be valid, both `num_bytes` and `num_glyphs` should be non-negative, and at least one should be non-zero. Note that clusters with zero glyphs are not as well supported as normal clusters. For example, PDF rendering applications typically ignore those clusters when PDF text is being selected.

See `Context.show_text_glyphs()` for how clusters are used in advanced text operations.

**num\_bytes**  
*int*

**num\_glyphs**  
*int*

## TextExtents

### class TextExtents(tuple)

class `cairo.TextExtents` (*x\_bearing*, *y\_bearing*, *width*, *height*, *x\_advance*, *y\_advance*)

**Parameters**

- **x\_bearing** (*float*) – the horizontal distance from the origin to the leftmost part of the glyphs as drawn. Positive if the glyphs lie entirely to the right of the origin.
- **y\_bearing** (*float*) – the vertical distance from the origin to the topmost part of the glyphs as drawn. Positive only if the glyphs lie completely below the origin; will usually be negative.
- **width** (*float*) – width of the glyphs as drawn
- **height** (*float*) – height of the glyphs as drawn
- **x\_advance** (*float*) – distance to advance in the X direction after drawing these glyphs
- **y\_advance** (*float*) – distance to advance in the Y direction after drawing these glyphs. Will typically be zero except for vertical text layout as found in East-Asian languages.

**Return type** *TextExtents*

New in version 1.15: In prior versions a (float, float, float, float, float, float) tuple was used instead of *TextExtents*.

The *TextExtents* class stores the extents of a single glyph or a string of glyphs in user-space coordinates. Because text extents are in user-space coordinates, they are mostly, but not entirely, independent of the current transformation matrix. If you call `context.scale(2.0, 2.0)`, text will be drawn twice as big, but the reported text extents will not be doubled. They will change slightly due to hinting (so you can't assume that metrics are independent of the transformation matrix), but otherwise will remain unchanged.

**x\_bearing**  
*float*

**y\_bearing**  
float

**width**  
float

**height**  
float

**x\_advance**  
float

**y\_advance**  
float

## Legacy Constants

These constants are aliases for enum attributes in newer versions of Pycairo. They might still be useful if you need to support Pycairo versions older than 1.13.

`cairo.ANTIALIAS_DEFAULT`  
See *Antialias.DEFAULT*

`cairo.ANTIALIAS_NONE`  
See *Antialias.NONE*

`cairo.ANTIALIAS_GRAY`  
See *Antialias.GRAY*

`cairo.ANTIALIAS_SUBPIXEL`  
See *Antialias.SUBPIXEL*

`cairo.ANTIALIAS_FAST`  
See *Antialias.FAST*

New in version 1.12.0.

`cairo.ANTIALIAS_GOOD`  
See *Antialias.GOOD*

New in version 1.12.0.

`cairo.ANTIALIAS_BEST`  
See *Antialias.BEST*

New in version 1.12.0.

`cairo.CONTENT_COLOR`  
See *Content.COLOR*

`cairo.CONTENT_ALPHA`  
See *Content.ALPHA*

`cairo.CONTENT_COLOR_ALPHA`  
See *Content.COLOR\_ALPHA*

`cairo.EXTEND_NONE`  
See *Extend.NONE*

`cairo.EXTEND_REPEAT`  
See *Extend.REPEAT*

---

**cairo.EXTEND\_REFLECT**  
See *Extend.REFLECT*

**cairo.EXTEND\_PAD**  
See *Extend.PAD*

**cairo.FILL\_RULE\_WINDING**  
See *FillRule.WINDING*

**cairo.FILL\_RULE\_EVEN\_ODD**  
See *FillRule.EVEN\_ODD*

**cairo.FILTER\_FAST**  
See *Filter.FAST*

**cairo.FILTER\_GOOD**  
See *Filter.GOOD*

**cairo.FILTER\_BEST**  
See *Filter.BEST*

**cairo.FILTER\_NEAREST**  
See *Filter.NEAREST*

**cairo.FILTER\_BILINEAR**  
See *Filter.BILINEAR*

**cairo.FILTER\_GAUSSIAN**  
See *Filter.GAUSSIAN*

**cairo.FONT\_SLANT\_NORMAL**  
See *FontSlant.NORMAL*

**cairo.FONT\_SLANT\_ITALIC**  
See *FontSlant.ITALIC*

**cairo.FONT\_SLANT\_OBLIQUE**  
See *FontSlant.OBLIQUE*

**cairo.FONT\_WEIGHT\_NORMAL**  
See *FontWeight.NORMAL*

**cairo.FONT\_WEIGHT\_BOLD**  
See *FontWeight.BOLD*

**cairo.FORMAT\_INVALID**  
See *Format.INVALID*  
New in version 1.12.0.

**cairo.FORMAT\_ARGB32**  
See *Format.ARGB32*

**cairo.FORMAT\_RGB24**  
See *Format.RGB24*

**cairo.FORMAT\_A8**  
See *Format.A8*

**cairo.FORMAT\_A1**  
See *Format.A1*

**cairo.FORMAT\_RGB16\_565**  
See *Format.RGB16\_565*

`cairo.FORMAT_RGB30`  
See *Format.RGB30*  
New in version 1.12.0.

`cairo.HINT_METRICS_DEFAULT`  
See *HintMetrics.DEFAULT*

`cairo.HINT_METRICS_OFF`  
See *HintMetrics.OFF*

`cairo.HINT_METRICS_ON`  
See *HintMetrics.ON*

`cairo.HINT_STYLE_DEFAULT`  
See *HintStyle.DEFAULT*

`cairo.HINT_STYLE_NONE`  
See *HintStyle.NONE*

`cairo.HINT_STYLE_SLIGHT`  
See *HintStyle.SLIGHT*

`cairo.HINT_STYLE_MEDIUM`  
See *HintStyle.MEDIUM*

`cairo.HINT_STYLE_FULL`  
See *HintStyle.FULL*

`cairo.LINE_CAP_BUTT`  
See *LineCap.BUTT*

`cairo.LINE_CAP_ROUND`  
See *LineCap.ROUND*

`cairo.LINE_CAP_SQUARE`  
See *LineCap.SQUARE*

`cairo.LINE_JOIN_MITER`  
See *LineJoin.MITER*

`cairo.LINE_JOIN_ROUND`  
See *LineJoin.ROUND*

`cairo.LINE_JOIN_BEVEL`  
See *LineJoin.BEVEL*

`cairo.OPERATOR_CLEAR`  
See *Operator.CLEAR*

`cairo.OPERATOR_SOURCE`  
See *Operator.SOURCE*

`cairo.OPERATOR_OVER`  
See *Operator.OVER*

`cairo.OPERATOR_IN`  
See *Operator.IN*

`cairo.OPERATOR_OUT`  
See *Operator.OUT*

`cairo.OPERATOR_ATOP`  
See *Operator.ATOP*



---

`cairo.OPERATOR_DEST`  
See *Operator.DEST*

`cairo.OPERATOR_DEST_OVER`  
See *Operator.DEST\_OVER*

`cairo.OPERATOR_DEST_IN`  
See *Operator.DEST\_IN*

`cairo.OPERATOR_DEST_OUT`  
See *Operator.DEST\_OUT*

`cairo.OPERATOR_DEST_ATOP`  
See *Operator.DEST\_ATOP*

`cairo.OPERATOR_XOR`  
See *Operator.XOR*

`cairo.OPERATOR_ADD`  
See *Operator.ADD*

`cairo.OPERATOR_SATURATE`  
See *Operator.SATURATE*

`cairo.OPERATOR_MULTIPLY`  
See *Operator.MULTIPLY*

New in version 1.12.0.

`cairo.OPERATOR_SCREEN`  
See *Operator.SCREEN*

New in version 1.12.0.

`cairo.OPERATOR_OVERLAY`  
See *Operator.OVERLAY*

New in version 1.12.0.

`cairo.OPERATOR_DARKEN`  
See *Operator.DARKEN*

New in version 1.12.0.

`cairo.OPERATOR_LIGHTEN`  
See *Operator.LIGHTEN*

New in version 1.12.0.

`cairo.OPERATOR_COLOR_DODGE`  
See *Operator.COLOR\_DODGE*

New in version 1.12.0.

`cairo.OPERATOR_COLOR_BURN`  
See *Operator.COLOR\_BURN*

New in version 1.12.0.

`cairo.OPERATOR_HARD_LIGHT`  
See *Operator.HARD\_LIGHT*

New in version 1.12.0.

**cairo.OPERATOR\_SOFT\_LIGHT**  
See *Operator.SOFT\_LIGHT*  
New in version 1.12.0.

**cairo.OPERATOR\_DIFFERENCE**  
See *Operator.DIFFERENCE*  
New in version 1.12.0.

**cairo.OPERATOR\_EXCLUSION**  
See *Operator.EXCLUSION*  
New in version 1.12.0.

**cairo.OPERATOR\_HSL\_HUE**  
See *Operator.HSL\_HUE*  
New in version 1.12.0.

**cairo.OPERATOR\_HSL\_SATURATION**  
See *Operator.HSL\_SATURATION*  
New in version 1.12.0.

**cairo.OPERATOR\_HSL\_COLOR**  
See *Operator.HSL\_COLOR*  
New in version 1.12.0.

**cairo.OPERATOR\_HSL\_LUMINOSITY**  
See *Operator.HSL\_LUMINOSITY*  
New in version 1.12.0.

**cairo.PATH\_MOVE\_TO**  
See *PathDataType.MOVE\_TO*

**cairo.PATH\_LINE\_TO**  
See *PathDataType.LINE\_TO*

**cairo.PATH\_CURVE\_TO**  
See *PathDataType.CURVE\_TO*

**cairo.PATH\_CLOSE\_PATH**  
See *PathDataType.CLOSE\_PATH*

**cairo.PS\_LEVEL\_2**  
See *PSLevel.LEVEL\_2*

**cairo.PS\_LEVEL\_3**  
See *PSLevel.LEVEL\_3*

**cairo.PDF\_VERSION\_1\_4**  
See *PDFVersion.VERSION\_1\_4*  
New in version 1.12.0.

**cairo.PDF\_VERSION\_1\_5**  
See *PDFVersion.VERSION\_1\_5*  
New in version 1.12.0.

**cairo.SVG\_VERSION\_1\_1**  
See *SVGVersion.VERSION\_1\_1*

New in version 1.12.0.

`cairo.SVG_VERSION_1_2`

See `SVGVersion.VERSION_1_2`

New in version 1.12.0.

`cairo.SUBPIXEL_ORDER_DEFAULT`

See `SubpixelOrder.DEFAULT`

`cairo.SUBPIXEL_ORDER_RGB`

See `SubpixelOrder.RGB`

`cairo.SUBPIXEL_ORDER_BGR`

See `SubpixelOrder.BGR`

`cairo.SUBPIXEL_ORDER_VRGB`

See `SubpixelOrder.VRGB`

`cairo.SUBPIXEL_ORDER_VBGR`

See `SubpixelOrder.VBGR`

`cairo.REGION_OVERLAP_IN`

See `RegionOverlap.IN`

New in version 1.11.

`cairo.REGION_OVERLAP_OUT`

See `RegionOverlap.OUT`

New in version 1.11.

`cairo.REGION_OVERLAP_PART`

See `RegionOverlap.PART`

New in version 1.11.

`cairo.STATUS_SUCCESS`

`cairo.STATUS_NO_MEMORY`

`cairo.STATUS_INVALID_RESTORE`

`cairo.STATUS_INVALID_POP_GROUP`

`cairo.STATUS_NO_CURRENT_POINT`

`cairo.STATUS_INVALID_MATRIX`

`cairo.STATUS_INVALID_STATUS`

`cairo.STATUS_NULL_POINTER`

`cairo.STATUS_INVALID_STRING`

`cairo.STATUS_INVALID_PATH_DATA`

`cairo.STATUS_READ_ERROR`

`cairo.STATUS_WRITE_ERROR`

`cairo.STATUS_SURFACE_FINISHED`

`cairo.STATUS_SURFACE_TYPE_MISMATCH`

`cairo.STATUS_PATTERN_TYPE_MISMATCH`

`cairo.STATUS_INVALID_CONTENT`

`cairo.STATUS_INVALID_FORMAT`

`cairo.STATUS_INVALID_VISUAL`

`cairo.STATUS_FILE_NOT_FOUND`

`cairo.STATUS_INVALID_DASH`

`cairo.STATUS_INVALID_DSC_COMMENT`

`cairo.STATUS_INVALID_INDEX`

`cairo.STATUS_CLIP_NOT_REPRESENTABLE`

`cairo.STATUS_TEMP_FILE_ERROR`

cairo.**STATUS\_INVALID\_STRIDE**  
cairo.**STATUS\_FONT\_TYPE\_MISMATCH**  
cairo.**STATUS\_USER\_FONT\_IMMUTABLE**  
cairo.**STATUS\_USER\_FONT\_ERROR**  
cairo.**STATUS\_NEGATIVE\_COUNT**  
cairo.**STATUS\_INVALID\_CLUSTERS**  
cairo.**STATUS\_INVALID\_SLANT**  
cairo.**STATUS\_INVALID\_WEIGHT**  
cairo.**STATUS\_INVALID\_SIZE**  
cairo.**STATUS\_USER\_FONT\_NOT\_IMPLEMENTED**  
cairo.**STATUS\_DEVICE\_TYPE\_MISMATCH**  
cairo.**STATUS\_DEVICE\_ERROR**  
cairo.**STATUS\_INVALID\_MESH\_CONSTRUCTION**  
cairo.**STATUS\_DEVICE\_FINISHED**  
cairo.**STATUS\_LAST\_STATUS**

See *Status*

New in version 1.12.

This manual documents the API used by C and C++ programmers who want to write extension modules that use pycairo.

## To access the Pycairo C API under Python 2

Edit the client module file to add the following lines:

```
/* All function, type and macro definitions needed to use the Pycairo/C API
 * are included in your code by the following line
 */
#include "Pycairo.h"

/* define a variable for the C API */
static Pycairo_CAPI_t *Pycairo_CAPI;

/* import pycairo - add to the init<module> function */
Pycairo_IMPORT;
```

## To access the Pycairo C API under Python 3

Example showing how to import the pycairo API:

```
#include "py3cairo.h"

PyMODINIT_FUNC
PyInit_client(void)
{
    PyObject *m;

    m = PyModule_Create(&clientmodule);
```

```
if (m == NULL)
    return NULL;
if (import_cairo() < 0)
    return NULL;
/* additional initialization can happen here */
return m;
}
```

## Pkg-Config Setup

pycairo installs “pycairo.pc” or “py3cairo.pc” in case of a Python 3 install:

```
> pkg-config --libs --cflags pycairo
> pkg-config --libs --cflags py3cairo
-I/usr/include/cairo -I/usr/include/glib-2.0
-I/usr/lib/x86_64-linux-gnu/glib-2.0/include -I/usr/include/pixman-1
-I/usr/include/freetype2 -I/usr/include/libpng16 -I/usr/include/pycairo
-lcairo
```

## Misc Functions

int **Pycairo\_Check\_Status** (*cairo\_status\_t status*)

### Parameters

- **status** (*cairo\_status\_t*) –

**Returns** -1 in case of an error, otherwise 0. Sets an exception in case of an error.

Takes a status value and converts it to an exception if it represents an error status.

## Cairo Context

PyObject **PycairoContext**

*cairo\_t*\* **PycairoContext.ctx**

The wrapped *cairo\_t*

PyTypeObject\* **PycairoContext\_Type**

*cairo\_t*\* **PycairoContext\_GET** (*PycairoContext*\*obj)

### Parameters

- **obj** (*PycairoContext*) –

**Returns** *cairo\_t* [transfer none]

Get the *cairo\_t* object out of the *PycairoContext*.

PyObject\* **PycairoContext\_FromContext** (*cairo\_t*\*ctx, PyTypeObject\*type, PyObject\*base)

### Parameters

- **ctx** (*cairo\_t*) – a *cairo\_t* to ‘wrap’ into a Python object. It is unreferenced if the `PycairoContext` creation fails, or if the *cairo\_t* has an error status. [transfer full]
- **type** (*PyObject*) – a pointer to the type to instantiate. It can be `&PycairoContext_Type`, or a `PycairoContext_Type` subtype. (`cairo.Context` or a `cairo.Context` subclass) [transfer none]
- **base** (*PyObject*) – the base object used to create the context, or `NULL`. it is referenced to keep it alive while the *cairo\_t* is being used [transfer none]

**Returns** New reference or `NULL` on failure and sets an exception [transfer full]

Create a new `PycairoContext` from a *cairo\_t*

## Cairo Font Face

`PyObject` `PycairoFontFace`

*cairo\_font\_face\_t*\* `PycairoFontFace.font_face`

The wrapped *cairo\_font\_face\_t*

`PyObject` \*`PycairoFontFace_Type`

`PyObject` \* `PycairoFontFace_FromFontFace` (*cairo\_font\_face\_t* \**font\_face*)

### Parameters

- **font\_face** (*cairo\_font\_face\_t*) – a *cairo\_font\_face\_t* to ‘wrap’ into a Python object. it is unreferenced if the `PycairoFontFace` creation fails [transfer full]

**Returns** New reference or `NULL` on failure and sets an exception [transfer full]

Create a new `PycairoFontFace` from a *cairo\_font\_face\_t*

*PycairoFontFace* `PycairoToyFontFace`

`PyObject` \*`PycairoToyFontFace_Type`

## Cairo Font Options

`PyObject` `PycairoFontOptions`

*cairo\_font\_options\_t*\* `PycairoFontOptions.font_options`

`PyObject` \*`PycairoFontOptions_Type`

`PyObject` \* `PycairoFontOptions_FromFontOptions` (*cairo\_font\_options\_t* \**font\_options*)

### Parameters

- **font\_options** (*cairo\_font\_options\_t*) – a *cairo\_font\_options\_t* to ‘wrap’ into a Python object. it is unreferenced if the `PycairoFontOptions` creation fails [transfer full]

**Returns** New reference or `NULL` on failure and sets an exception [transfer full]

Create a new `PycairoFontOptions` from a *cairo\_font\_options\_t*

## Cairo Matrix

PyObject **PycairoMatrix**

*cairo\_matrix\_t* **PycairoMatrix.matrix**

PyTypeObject \***PycairoMatrix\_Type**

PyObject \* **PycairoMatrix\_FromMatrix** (const *cairo\_matrix\_t* \**matrix*)

### Parameters

- **matrix** (*cairo\_matrix\_t*) – a *cairo\_matrix\_t* to ‘wrap’ into a Python object. the *cairo\_matrix\_t* values are copied. [transfer none]

**Returns** New reference or NULL on failure and sets an exception [transfer full]

Create a new PycairoMatrix from a *cairo\_matrix\_t*

## Cairo Path

PyObject **PycairoPath**

*cairo\_path\_t*\* **PycairoPath.path**

PyTypeObject \***PycairoPath\_Type**

PyObject \* **PycairoPath\_FromPath** (*cairo\_path\_t* \**path*)

### Parameters

- **path** (*cairo\_path\_t*) – a *cairo\_path\_t* to ‘wrap’ into a Python object. *path* is unreferenced if the PycairoPath creation fails, or if *path* is in an error status. [transfer full]

**Returns** New reference or NULL on failure and sets an exception [transfer full]

Create a new PycairoPath from a *cairo\_path\_t*

## Cairo Pattern

PyObject **PycairoPattern**

*cairo\_pattern\_t*\* **PycairoPattern.pattern**

PyTypeObject \***PycairoPattern\_Type**

*PycairoPattern* **PycairoSolidPattern**

PyTypeObject \***PycairoSolidPattern\_Type**

*PycairoPattern* **PycairoSurfacePattern**

PyTypeObject \***PycairoSurfacePattern\_Type**

*PycairoPattern* **PycairoGradient**

PyTypeObject \***PycairoGradient\_Type**



*PycairoGradient* **PycairoLinearGradient**

PyTypeObject \***PycairoLinearGradient\_Type**

*PycairoGradient* **PycairoRadialGradient**

PyTypeObject \***PycairoRadialGradient\_Type**

PyObject \* **PycairoPattern\_FromPattern** (*cairo\_pattern\_t* \**pattern*, PyObject \**base*)

#### Parameters

- **pattern** (*cairo\_pattern\_t*) – a *cairo\_pattern\_t* to ‘wrap’ into a Python object. It is unreferenced if the PycairoPattern creation fails, or if the pattern has an error status. [transfer full]
- **base** (*PyObject*) – the base object used to create the pattern, or NULL. It is referenced to keep it alive while the *cairo\_pattern\_t* is being used. [transfer none]

**Returns** New reference or NULL on failure and sets an exception [transfer full]

Create a new PycairoSolidPattern, PycairoSurfacePattern, PycairoLinearGradient, or PycairoRadialGradient from a *cairo\_pattern\_t*.

## Cairo Region

PyObject **PycairoRegion**

*cairo\_region\_t*\* **PycairoRegion.region**

PyTypeObject \***PycairoRegion\_Type**

PyObject \* **PycairoRegion\_FromRegion** (*cairo\_region\_t* \**region*)

#### Parameters

- **region** (*cairo\_region\_t*) – a *cairo\_region\_t* to ‘wrap’ into a Python object. *region* is unreferenced if the PycairoRegion creation fails, or if *region* is in an error status.

**Returns** New reference or NULL on failure and sets an exception [transfer full]

Create a new PycairoRegion from a *cairo\_region\_t*

## Cairo RectangleInt

PyObject **PycairoRectangleInt**

*cairo\_rectangle\_int\_t*\* **PycairoRectangleInt.rectangle\_int**

PyTypeObject \***PycairoRectangleInt\_Type**

PyObject \* **PycairoRectangleInt\_FromRectangleInt** (const *cairo\_rectangle\_int\_t* \**rectangle\_int*)

#### Parameters

- **rectangle\_int** (*cairo\_rectangle\_int\_t*) – a *cairo\_rectangle\_int\_t* to ‘wrap’ into a Python object. *rectangle\_int* is unreferenced if the PycairoRectangleInt creation fails. [transfer none]

**Returns** New reference or NULL on failure and sets an exception [transfer full]

Create a new PycairoRectangleInt from a `cairo_rectangle_int_t`

## Scaled Font

PyObject **PycairoScaledFont**

*cairo\_scaled\_font\_t*\* **PycairoScaledFont.scaled\_font**

PyTypeObject \***PycairoScaledFont\_Type**

PyObject \* **PycairoScaledFont\_FromScaledFont** (*cairo\_scaled\_font\_t* \*scaled\_font)

### Parameters

- **scaled\_font** (*cairo\_scaled\_font\_t*) – a `cairo_scaled_font_t` to ‘wrap’ into a Python object. it is unreferenced if the `PycairoScaledFont` creation fails [transfer full]

**Returns** New reference or NULL on failure and sets an exception [transfer full]

Create a new `PycairoScaledFont` from a `cairo_scaled_font_t`

## Cairo Surface

PyObject **PycairoSurface**

*cairo\_surface\_t*\* **PycairoSurface.surface**

PyTypeObject \***PycairoSurface\_Type**

*PycairoSurface* **PycairoImageSurface**

PyTypeObject \***PycairoImageSurface\_Type**

*PycairoSurface* **PycairoPDFSurface**

PyTypeObject \***PycairoPDFSurface\_Type**

*PycairoSurface* **PycairoPSSurface**

PyTypeObject \***PycairoPSSurface\_Type**

*PycairoSurface* **PycairoRecordingSurface**

PyTypeObject \***PycairoRecordingSurface\_Type**

*PycairoSurface* **PycairoSVGSurface**

PyTypeObject \***PycairoSVGSurface\_Type**

*PycairoSurface* **PycairoWin32Surface**

PyTypeObject \***PycairoWin32Surface\_Type**

*PycairoSurface* **PycairoXCBSurface**

PyTypeObject \***PycairoXCBSurface\_Type**

*PycairoSurface* **PycairoXlibSurface**

PyTypeObject \***PycairoXlibSurface\_Type**

PyObject \* **PycairoSurface\_FromSurface** (*cairo\_surface\_t* \**surface*, PyObject \**base*)

#### Parameters

- **surface** (*cairo\_surface\_t*) – a *cairo\_surface\_t* to ‘wrap’ into a Python object. It is unreferenced if the PycairoSurface creation fails, or if the *cairo\_surface\_t* has an error status. [transfer full]
- **base** (*PyObject*) – the base object used to create the surface, or NULL. It is referenced to keep it alive while the *cairo\_surface\_t* is being used. [transfer none]

**Returns** New reference or NULL on failure and sets an exception [transfer full]

Create a new PycairoImageSurface, PycairoPDFSurface, PycairoPSSurface, PycairoRecordingSurface, PycairoSVGSurface, PycairoWin32Surface, PycairoWin32PrintingSurface, PycairoXCBSurface, or PycairoXlibSurface from a *cairo\_surface\_t*.

## Cairo Types

These are only listed here so they can be referenced in the documentation.

See <https://www.cairographics.org/manual/> for details.

```

cairo_t
cairo_status_t
cairo_surface_t
cairo_scaled_font_t
cairo_rectangle_int_t
cairo_region_t
cairo_pattern_t
cairo_matrix_t
cairo_font_options_t
cairo_path_t
cairo_font_face_t

```



This section is for listing various useful pycairo resources, feel free to contribute !

**Windows Binary Packages (unofficial)** Precompiled binaries for the Microsoft Windows platform can be obtained from the following sources:

[Precompiled PyCairo for Python 2.x from Uri Shaked](#)

### Some Libraries/Modules Using pycairo

- [Cairo Plot](#): a module to plot graphics
- [hamster graphics library](#) - a sprite styled abstraction library for drawing and animation in PyGTK
- [matplotlib](#): a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.
- [PyCha](#): PYthon CHArts - a Python package for drawing charts
- [PyGoocanvas](#): python bindings for GooCanvas which is a canvas widget for GTK+
- [PyGTK](#): GTK+ for Python.
- [rsvg](#): part of [gnome-python-desktop](#), it provides Python bindings for librsvg

### Some Applications Using pycairo

- [A Shogiban for Gnushogi](#).
- [gPodder](#): a podcatcher.
- [Miro](#): Internet TV, HD video player.
- [pycairo projects at Google Code](#).
- [PyChess](#).
- [Pyroute - OpenStreetMap](#).
- [Shoebot](#) - a pure Python graphics robot.

### Tutorials

- [Cairo Tutorial for Python \(and other\) Programmers](#): Generic introduction to cairo concepts oriented to python.
- [Cairo Tutorial for PyGTK Programmers](#): Tutorial about how to use cairo for drawing in PyGTK.
- [Writing a widget using cairo and PyGTK 2.8 Part 1, Part 2](#): A translation of the GNOME Journal tutorial by Davyd Madeley from C to Python.

### Demos

- [A Basic Cairo-clock in Python using XShape](#).
- [A simple clock implemented in pygtk and cairo](#).

**Recipies** See the main [Cairo Cookbook](#).

Pycairo is a Python module providing bindings for the [cairo graphics library](#).

The Pycairo bindings are designed to match the cairo C API as closely as possible, and to deviate only in cases which are clearly better implemented in a more ‘Pythonic’ way.

Features of the Pycairo bindings:

- Provides an object oriented interface to cairo.
- Queries the error status of objects and translates them to exceptions.
- Provides a C API that can be used by other Python extensions.

Pycairo depends on **cairo**  $\geq$  **1.14** (the 1.13 git snapshot present in Ubuntu 14.04 works as well) and works with **Python 2.7+** as well as **Python 3.3+**. Pycairo, including this documentation, is licensed under the  **LGPLv2.1**  as well as the  **MPLv1.1** .

If Pycairo is not what you need, have a look at [cairocffi](#), which is an API compatible package using [cffi](#) or [Qahirah](#), which is using [ctypes](#) and provides a more “pythonic” API with less focus on matching the cairo C API.

For more information visit <https://pycairo.readthedocs.io>

---

**Tarballs:** <https://github.com/pygobject/pycairo/releases>

**Git repo:** <https://github.com/pygobject/pycairo>

**Bug tracker:** <https://github.com/pygobject/pycairo/issues>

**Mailing list:** <https://lists.cairographics.org/cgi-bin/mailman/listinfo/cairo>

See the “*API Reference*” for further details.

To use the pycairo library:

```
import cairo
```

To build/install the library:

```
python2/3 setup.py build
python2/3 setup.py install
```

To run the tests:

```
python2/3 setup.py test
```

The Python 2 version supports [xpyb](#) integration which is disabled by default. To enable, build as follows:

```
python2 setup.py build --enable-xpyb  
# and for running tests:  
python2 setup.py test --enable-xpyb
```

For examples of pycairo code see the 'examples' directory that comes with the pycairo distribution.

For author information see the git history as well as the now deleted "ChangeLog" file in the git history.





**A**

A1 (cairo.Format attribute), 85  
 A8 (cairo.Format attribute), 85  
 acquire() (cairo.Device method), 150  
 ADD (cairo.Operator attribute), 88  
 add() (cairo.TeeSurface method), 144  
 add\_color\_stop\_rgb() (cairo.Gradient method), 118  
 add\_color\_stop\_rgba() (cairo.Gradient method), 118  
 ALPHA (cairo.Content attribute), 83  
 Antialias (class in cairo), 83  
 ANTIALIAS\_BEST (in module cairo), 154  
 ANTIALIAS\_DEFAULT (in module cairo), 154  
 ANTIALIAS\_FAST (in module cairo), 154  
 ANTIALIAS\_GOOD (in module cairo), 154  
 ANTIALIAS\_GRAY (in module cairo), 154  
 ANTIALIAS\_NONE (in module cairo), 154  
 ANTIALIAS\_SUBPIXEL (in module cairo), 154  
 append\_path() (cairo.Context method), 92  
 arc() (cairo.Context method), 92  
 arc\_negative() (cairo.Context method), 92  
 ARGB32 (cairo.Format attribute), 85  
 ASCII (cairo.ScriptMode attribute), 91  
 ATOP (cairo.Operator attribute), 87

**B**

BACKWARD (cairo.TextClusterFlags attribute), 91  
 begin\_patch() (cairo.MeshPattern method), 122  
 BEST (cairo.Antialias attribute), 83  
 BEST (cairo.Filter attribute), 84  
 BEVEL (cairo.LineJoin attribute), 87  
 BGR (cairo.SubpixelOrder attribute), 90  
 BILINEAR (cairo.Filter attribute), 84  
 BINARY (cairo.ScriptMode attribute), 91  
 BOLD (cairo.FontWeight attribute), 85  
 BUTT (cairo.LineCap attribute), 86

**C**

cairo\_font\_face\_t (C type), 167  
 cairo\_font\_options\_t (C type), 167

cairo\_matrix\_t (C type), 167  
 cairo\_path\_t (C type), 167  
 cairo\_pattern\_t (C type), 167  
 cairo\_rectangle\_int\_t (C type), 167  
 cairo\_region\_t (C type), 167  
 cairo\_scaled\_font\_t (C type), 167  
 cairo\_status\_t (C type), 167  
 cairo\_surface\_t (C type), 167  
 cairo\_t (C type), 167  
 cairo\_version() (in module cairo), 81  
 cairo\_version\_string() (in module cairo), 81  
 CairoError, 112  
 CLEAR (cairo.Operator attribute), 87  
 clip() (cairo.Context method), 93  
 clip\_extents() (cairo.Context method), 93  
 CLIP\_NOT\_REPRESENTABLE (cairo.Status attribute), 90  
 clip\_preserve() (cairo.Context method), 93  
 CLOSE\_PATH (cairo.PathDataType attribute), 89  
 close\_path() (cairo.Context method), 93  
 COLOR (cairo.Content attribute), 83  
 COLOR\_ALPHA (cairo.Content attribute), 83  
 COLOR\_BURN (cairo.Operator attribute), 88  
 COLOR\_DODGE (cairo.Operator attribute), 88  
 contains\_point() (cairo.Region method), 127  
 contains\_rectangle() (cairo.Region method), 127  
 Content (class in cairo), 83  
 CONTENT\_ALPHA (in module cairo), 154  
 CONTENT\_COLOR (in module cairo), 154  
 CONTENT\_COLOR\_ALPHA (in module cairo), 154  
 Context (class in cairo), 91  
 copy() (cairo.FontOptions method), 149  
 copy() (cairo.Region method), 126  
 copy\_clip\_rectangle\_list() (cairo.Context method), 94  
 copy\_page() (cairo.Context method), 94  
 copy\_page() (cairo.Surface method), 128  
 copy\_path() (cairo.Context method), 94  
 copy\_path\_flat() (cairo.Context method), 94  
 create\_for\_data() (cairo.ImageSurface class method), 134  
 create\_for\_rectangle() (cairo.Surface method), 132

create\_for\_target() (cairo.ScriptSurface class method), 143  
create\_from\_png() (cairo.ImageSurface class method), 135  
create\_similar() (cairo.Surface method), 129  
create\_similar\_image() (cairo.Surface method), 132  
CURVE\_TO (cairo.PathDataType attribute), 89  
curve\_to() (cairo.Context method), 94  
curve\_to() (cairo.MeshPattern method), 122

## D

DARKEN (cairo.Operator attribute), 88  
DEFAULT (cairo.Antialias attribute), 83  
DEFAULT (cairo.HintMetrics attribute), 86  
DEFAULT (cairo.HintStyle attribute), 86  
DEFAULT (cairo.SubpixelOrder attribute), 89  
DEST (cairo.Operator attribute), 87  
DEST\_ATOP (cairo.Operator attribute), 88  
DEST\_IN (cairo.Operator attribute), 87  
DEST\_OUT (cairo.Operator attribute), 87  
DEST\_OVER (cairo.Operator attribute), 87  
Device (class in cairo), 150  
DEVICE\_ERROR (cairo.Status attribute), 90  
DEVICE\_FINISHED (cairo.Status attribute), 90  
device\_to\_user() (cairo.Context method), 95  
device\_to\_user\_distance() (cairo.Context method), 95  
DEVICE\_TYPE\_MISMATCH (cairo.Status attribute), 90  
DIFFERENCE (cairo.Operator attribute), 88  
dsc\_begin\_page\_setup() (cairo.PSSurface method), 136  
dsc\_begin\_setup() (cairo.PSSurface method), 137  
dsc\_comment() (cairo.PSSurface method), 137

## E

end\_patch() (cairo.MeshPattern method), 122  
equal() (cairo.FontOptions method), 149  
equal() (cairo.Region method), 127  
Error, 112  
EVEN\_ODD (cairo.FillRule attribute), 84  
EXCLUSION (cairo.Operator attribute), 88  
Extend (class in cairo), 83  
EXTEND\_NONE (in module cairo), 154  
EXTEND\_PAD (in module cairo), 155  
EXTEND\_REFLECT (in module cairo), 154  
EXTEND\_REPEAT (in module cairo), 154  
extents() (cairo.ScaledFont method), 146

## F

FAST (cairo.Antialias attribute), 83  
FAST (cairo.Filter attribute), 84  
FILE\_NOT\_FOUND (cairo.Status attribute), 90  
fill() (cairo.Context method), 95  
fill\_extents() (cairo.Context method), 95  
fill\_preserve() (cairo.Context method), 96  
FILL\_RULE\_EVEN\_ODD (in module cairo), 155

FILL\_RULE\_WINDING (in module cairo), 155  
FillRule (class in cairo), 84  
Filter (class in cairo), 84  
FILTER\_BEST (in module cairo), 155  
FILTER\_BILINEAR (in module cairo), 155  
FILTER\_FAST (in module cairo), 155  
FILTER\_GAUSSIAN (in module cairo), 155  
FILTER\_GOOD (in module cairo), 155  
FILTER\_NEAREST (in module cairo), 155  
finish() (cairo.Device method), 150  
finish() (cairo.Surface method), 129  
flush() (cairo.Device method), 150  
flush() (cairo.Surface method), 129  
font\_extents() (cairo.Context method), 96  
FONT\_SLANT\_ITALIC (in module cairo), 155  
FONT\_SLANT\_NORMAL (in module cairo), 155  
FONT\_SLANT\_OBLIQUE (in module cairo), 155  
FONT\_TYPE\_MISMATCH (cairo.Status attribute), 90  
FONT\_WEIGHT\_BOLD (in module cairo), 155  
FONT\_WEIGHT\_NORMAL (in module cairo), 155  
FontFace (class in cairo), 144  
FontOptions (class in cairo), 148  
FontSlant (class in cairo), 84  
FontWeight (class in cairo), 85  
Format (class in cairo), 85  
FORMAT\_A1 (in module cairo), 155  
FORMAT\_A8 (in module cairo), 155  
FORMAT\_ARGB32 (in module cairo), 155  
FORMAT\_INVALID (in module cairo), 155  
FORMAT\_RGB16\_565 (in module cairo), 155  
FORMAT\_RGB24 (in module cairo), 155  
FORMAT\_RGB30 (in module cairo), 155  
format\_stride\_for\_width() (cairo.ImageSurface static method), 135  
from\_recording\_surface() (cairo.ScriptDevice method), 151  
FULL (cairo.HintStyle attribute), 86

## G

GAUSSIAN (cairo.Filter attribute), 84  
get\_acquire() (cairo.RasterSourcePattern method), 126  
get\_antialias() (cairo.Context method), 96  
get\_antialias() (cairo.FontOptions method), 148  
get\_color\_stops\_rgba() (cairo.Gradient method), 118  
get\_content() (cairo.Surface method), 129  
get\_control\_point() (cairo.MeshPattern method), 122  
get\_corner\_color\_rgba() (cairo.MeshPattern method), 123  
get\_ctm() (cairo.ScaledFont method), 146  
get\_current\_point() (cairo.Context method), 96  
get\_dash() (cairo.Context method), 96  
get\_dash\_count() (cairo.Context method), 97  
get\_data() (cairo.ImageSurface method), 135  
get\_depth() (cairo.XlibSurface method), 142

get\_device() (cairo.Surface method), 133  
 get\_device\_offset() (cairo.Surface method), 129  
 get\_device\_scale() (cairo.Surface method), 133  
 get\_eps() (cairo.PSSurface method), 138  
 get\_extend() (cairo.Pattern method), 116  
 get\_extents() (cairo.RecordingSurface method), 140  
 get\_extents() (cairo.Region method), 126  
 get\_fallback\_resolution() (cairo.Surface method), 129  
 get\_family() (cairo.ToyFontFace method), 145  
 get\_fill\_rule() (cairo.Context method), 97  
 get\_filter() (cairo.Pattern method), 116  
 get\_font\_face() (cairo.Context method), 97  
 get\_font\_face() (cairo.ScaledFont method), 147  
 get\_font\_matrix() (cairo.Context method), 97  
 get\_font\_matrix() (cairo.ScaledFont method), 147  
 get\_font\_options() (cairo.Context method), 97  
 get\_font\_options() (cairo.ScaledFont method), 147  
 get\_font\_options() (cairo.Surface method), 130  
 get\_format() (cairo.ImageSurface method), 135  
 get\_group\_target() (cairo.Context method), 97  
 get\_height() (cairo.ImageSurface method), 135  
 get\_height() (cairo.XlibSurface method), 143  
 get\_hint\_metrics() (cairo.FontOptions method), 148  
 get\_hint\_style() (cairo.FontOptions method), 149  
 get\_levels() (cairo.PSSurface static method), 139  
 get\_line\_cap() (cairo.Context method), 97  
 get\_line\_join() (cairo.Context method), 97  
 get\_line\_width() (cairo.Context method), 97  
 get\_linear\_points() (cairo.LinearGradient method), 119  
 get\_matrix() (cairo.Context method), 97  
 get\_matrix() (cairo.Pattern method), 116  
 get\_mime\_data() (cairo.Surface method), 130  
 get\_miter\_limit() (cairo.Context method), 98  
 get\_mode() (cairo.ScriptDevice method), 151  
 get\_operator() (cairo.Context method), 98  
 get\_patch\_count() (cairo.MeshPattern method), 123  
 get\_path() (cairo.MeshPattern method), 123  
 get\_radial\_circles() (cairo.RadialGradient method), 120  
 get\_rectangle() (cairo.Region method), 127  
 get\_rgba() (cairo.SolidPattern method), 117  
 get\_scale\_matrix() (cairo.ScaledFont method), 147  
 get\_scaled\_font() (cairo.Context method), 98  
 get\_slant() (cairo.ToyFontFace method), 145  
 get\_source() (cairo.Context method), 98  
 get\_stride() (cairo.ImageSurface method), 135  
 get\_subpixel\_order() (cairo.FontOptions method), 149  
 get\_surface() (cairo.SurfacePattern method), 117  
 get\_target() (cairo.Context method), 98  
 get\_tolerance() (cairo.Context method), 98  
 get\_versions() (cairo.PDFSurface static method), 136  
 get\_versions() (cairo.SVGSurface static method), 141  
 get\_weight() (cairo.ToyFontFace method), 146  
 get\_width() (cairo.ImageSurface method), 135  
 get\_width() (cairo.XlibSurface method), 143

Glyph (class in cairo), 151  
 glyph\_extents() (cairo.Context method), 98  
 glyph\_extents() (cairo.ScaledFont method), 147  
 glyph\_path() (cairo.Context method), 98  
 GOOD (cairo.Antialias attribute), 83  
 GOOD (cairo.Filter attribute), 84  
 Gradient (class in cairo), 118  
 GRAY (cairo.Antialias attribute), 83

## H

HARD\_LIGHT (cairo.Operator attribute), 88  
 HAS\_ATSUI\_FONT (in module cairo), 81  
 has\_current\_point() (cairo.Context method), 98  
 HAS\_FT\_FONT (in module cairo), 81  
 HAS\_GLITZ\_SURFACE (in module cairo), 81  
 HAS\_IMAGE\_SURFACE (in module cairo), 81  
 HAS\_MIME\_SURFACE (in module cairo), 82  
 HAS\_PDF\_SURFACE (in module cairo), 81  
 HAS\_PNG\_FUNCTIONS (in module cairo), 81  
 HAS\_PS\_SURFACE (in module cairo), 81  
 HAS\_QUARTZ\_SURFACE (in module cairo), 81  
 HAS\_RECORDING\_SURFACE (in module cairo), 81  
 HAS\_SCRIPT\_SURFACE (in module cairo), 82  
 has\_show\_text\_glyphs() (cairo.Surface method), 132  
 HAS\_SVG\_SURFACE (in module cairo), 81  
 HAS\_TEE\_SURFACE (in module cairo), 82  
 HAS\_USER\_FONT (in module cairo), 81  
 HAS\_WIN32\_FONT (in module cairo), 81  
 HAS\_WIN32\_SURFACE (in module cairo), 81  
 HAS\_XCB\_SURFACE (in module cairo), 81  
 HAS\_XLIB\_SURFACE (in module cairo), 81  
 hash() (cairo.FontOptions method), 149  
 height (cairo.Rectangle attribute), 152  
 height (cairo.RectangleInt attribute), 128  
 height (cairo.TextExtents attribute), 154  
 HINT\_METRICS\_DEFAULT (in module cairo), 156  
 HINT\_METRICS\_OFF (in module cairo), 156  
 HINT\_METRICS\_ON (in module cairo), 156  
 HINT\_STYLE\_DEFAULT (in module cairo), 156  
 HINT\_STYLE\_FULL (in module cairo), 156  
 HINT\_STYLE\_MEDIUM (in module cairo), 156  
 HINT\_STYLE\_NONE (in module cairo), 156  
 HINT\_STYLE\_SLIGHT (in module cairo), 156  
 HintMetrics (class in cairo), 86  
 HintStyle (class in cairo), 86  
 HSL\_COLOR (cairo.Operator attribute), 88  
 HSL\_HUE (cairo.Operator attribute), 88  
 HSL\_LUMINOSITY (cairo.Operator attribute), 88  
 HSL\_SATURATION (cairo.Operator attribute), 88

## I

identity\_matrix() (cairo.Context method), 98  
 ImageSurface (class in cairo), 134  
 IN (cairo.Operator attribute), 87

IN (cairo.RegionOverlap attribute), 90  
in\_clip() (cairo.Context method), 111  
in\_fill() (cairo.Context method), 98  
in\_stroke() (cairo.Context method), 99  
index (cairo.Glyph attribute), 152  
index() (cairo.TeeSurface method), 144  
init\_rotate() (cairo.Matrix class method), 113  
ink\_extents() (cairo.RecordingSurface method), 140  
intersect() (cairo.Region method), 127  
INVALID (cairo.Format attribute), 85  
INVALID\_CLUSTERS (cairo.Status attribute), 90  
INVALID\_CONTENT (cairo.Status attribute), 90  
INVALID\_DASH (cairo.Status attribute), 90  
INVALID\_DSC\_COMMENT (cairo.Status attribute), 90  
INVALID\_FORMAT (cairo.Status attribute), 90  
INVALID\_INDEX (cairo.Status attribute), 90  
INVALID\_MATRIX (cairo.Status attribute), 90  
INVALID\_MESH\_CONSTRUCTION (cairo.Status attribute), 90  
INVALID\_PATH\_DATA (cairo.Status attribute), 90  
INVALID\_POP\_GROUP (cairo.Status attribute), 90  
INVALID\_RESTORE (cairo.Status attribute), 90  
INVALID\_SIZE (cairo.Status attribute), 90  
INVALID\_SLANT (cairo.Status attribute), 90  
INVALID\_STATUS (cairo.Status attribute), 90  
INVALID\_STRIDE (cairo.Status attribute), 90  
INVALID\_STRING (cairo.Status attribute), 90  
INVALID\_VISUAL (cairo.Status attribute), 90  
INVALID\_WEIGHT (cairo.Status attribute), 90  
invert() (cairo.Matrix method), 113  
IOError, 112  
is\_empty() (cairo.Region method), 127  
ITALIC (cairo.FontSlant attribute), 85

## J

JBIG2\_GLOBAL\_MISSING (cairo.Status attribute), 91

## L

LAST\_STATUS (cairo.Status attribute), 90  
LEVEL\_2 (cairo.PSLevel attribute), 89  
LEVEL\_3 (cairo.PSLevel attribute), 89  
level\_to\_string() (cairo.PSSurface static method), 138  
LIGHTEN (cairo.Operator attribute), 88  
LINE\_CAP\_BUTT (in module cairo), 156  
LINE\_CAP\_ROUND (in module cairo), 156  
LINE\_CAP\_SQUARE (in module cairo), 156  
LINE\_JOIN\_BEVEL (in module cairo), 156  
LINE\_JOIN\_MITER (in module cairo), 156  
LINE\_JOIN\_ROUND (in module cairo), 156  
LINE\_TO (cairo.PathDataType attribute), 89  
line\_to() (cairo.Context method), 99  
line\_to() (cairo.MeshPattern method), 123  
LinearGradient (class in cairo), 119  
LineCap (class in cairo), 86

LineJoin (class in cairo), 87

## M

map\_to\_image() (cairo.Surface method), 133  
mark\_dirty() (cairo.Surface method), 130  
mark\_dirty\_rectangle() (cairo.Surface method), 130  
mask() (cairo.Context method), 99  
mask\_surface() (cairo.Context method), 99  
Matrix (class in cairo), 113  
Matrix.x0 (in module cairo), 115  
Matrix.xx (in module cairo), 115  
Matrix.xy (in module cairo), 115  
Matrix.y0 (in module cairo), 115  
Matrix.yx (in module cairo), 115  
Matrix.yy (in module cairo), 115  
MEDIUM (cairo.HintStyle attribute), 86  
MemoryError, 112  
merge() (cairo.FontOptions method), 149  
MeshPattern (class in cairo), 120  
MIME\_TYPE\_JP2 (in module cairo), 82  
MIME\_TYPE\_JPEG (in module cairo), 82  
MIME\_TYPE\_PNG (in module cairo), 82  
MIME\_TYPE\_UNIQUE\_ID (in module cairo), 82  
MIME\_TYPE\_URI (in module cairo), 82  
MITER (cairo.LineJoin attribute), 87  
MOVE\_TO (cairo.PathDataType attribute), 89  
move\_to() (cairo.Context method), 99  
move\_to() (cairo.MeshPattern method), 124  
MULTIPLY (cairo.Operator attribute), 88  
multiply() (cairo.Matrix method), 114

## N

NEAREST (cairo.Filter attribute), 84  
NEGATIVE\_COUNT (cairo.Status attribute), 90  
new\_path() (cairo.Context method), 100  
new\_sub\_path() (cairo.Context method), 100  
NO\_CURRENT\_POINT (cairo.Status attribute), 90  
NO\_MEMORY (cairo.Status attribute), 90  
NONE (cairo.Antialias attribute), 83  
NONE (cairo.Extend attribute), 83  
NONE (cairo.HintStyle attribute), 86  
NORMAL (cairo.FontSlant attribute), 84  
NORMAL (cairo.FontWeight attribute), 85  
NORMAL (cairo.SurfaceObserverMode attribute), 91  
NULL\_POINTER (cairo.Status attribute), 90  
num\_bytes (cairo.TextCluster attribute), 153  
num\_glyphs (cairo.TextCluster attribute), 153  
num\_rectangles() (cairo.Region method), 126

## O

OBLIQUE (cairo.FontSlant attribute), 85  
OFF (cairo.HintMetrics attribute), 86  
ON (cairo.HintMetrics attribute), 86  
Operator (class in cairo), 87



- OPERATOR\_ADD (in module cairo), 157  
 OPERATOR\_ATOP (in module cairo), 156  
 OPERATOR\_CLEAR (in module cairo), 156  
 OPERATOR\_COLOR\_BURN (in module cairo), 157  
 OPERATOR\_COLOR\_DODGE (in module cairo), 157  
 OPERATOR\_DARKEN (in module cairo), 157  
 OPERATOR\_DEST (in module cairo), 156  
 OPERATOR\_DEST\_ATOP (in module cairo), 157  
 OPERATOR\_DEST\_IN (in module cairo), 157  
 OPERATOR\_DEST\_OUT (in module cairo), 157  
 OPERATOR\_DEST\_OVER (in module cairo), 157  
 OPERATOR\_DIFFERENCE (in module cairo), 158  
 OPERATOR\_EXCLUSION (in module cairo), 158  
 OPERATOR\_HARD\_LIGHT (in module cairo), 157  
 OPERATOR\_HSL\_COLOR (in module cairo), 158  
 OPERATOR\_HSL\_HUE (in module cairo), 158  
 OPERATOR\_HSL\_LUMINOSITY (in module cairo), 158  
 OPERATOR\_HSL\_SATURATION (in module cairo), 158  
 OPERATOR\_IN (in module cairo), 156  
 OPERATOR\_LIGHTEN (in module cairo), 157  
 OPERATOR\_MULTIPLY (in module cairo), 157  
 OPERATOR\_OUT (in module cairo), 156  
 OPERATOR\_OVER (in module cairo), 156  
 OPERATOR\_OVERLAY (in module cairo), 157  
 OPERATOR\_SATURATE (in module cairo), 157  
 OPERATOR\_SCREEN (in module cairo), 157  
 OPERATOR\_SOFT\_LIGHT (in module cairo), 157  
 OPERATOR\_SOURCE (in module cairo), 156  
 OPERATOR\_XOR (in module cairo), 157  
 OUT (cairo.Operator attribute), 87  
 OUT (cairo.RegionOverlap attribute), 90  
 OVER (cairo.Operator attribute), 87  
 OVERLAY (cairo.Operator attribute), 88
- ## P
- PAD (cairo.Extend attribute), 84  
 paint() (cairo.Context method), 100  
 paint\_with\_alpha() (cairo.Context method), 100  
 PART (cairo.RegionOverlap attribute), 90  
 Path (class in cairo), 115  
 PATH\_CLOSE\_PATH (in module cairo), 158  
 PATH\_CURVE\_TO (in module cairo), 158  
 path\_extents() (cairo.Context method), 100  
 PATH\_LINE\_TO (in module cairo), 158  
 PATH\_MOVE\_TO (in module cairo), 158  
 PathDataType (class in cairo), 89  
 pathlike (class in cairo), 82  
 Pattern (class in cairo), 116  
 PATTERN\_TYPE\_MISMATCH (cairo.Status attribute), 90  
 PDF\_VERSION\_1\_4 (in module cairo), 158  
 PDF\_VERSION\_1\_5 (in module cairo), 158  
 PDFSurface (class in cairo), 135  
 PDFVersion (class in cairo), 89  
 pop\_group() (cairo.Context method), 100  
 pop\_group\_to\_source() (cairo.Context method), 101  
 PS\_LEVEL\_2 (in module cairo), 158  
 PS\_LEVEL\_3 (in module cairo), 158  
 PSLevel (class in cairo), 89  
 PSSurface (class in cairo), 136  
 PSSurface.ps\_level\_to\_string (in module cairo), 138  
 push\_group() (cairo.Context method), 101  
 push\_group\_with\_content() (cairo.Context method), 102  
 Pycairo\_Check\_Status (C function), 162  
 PycairoContext (C type), 162  
 PycairoContext.PycairoContext.ctx (C member), 162  
 PycairoContext\_FromContext (C function), 162  
 PycairoContext\_GET (C macro), 162  
 PycairoContext\_Type (C type), 162  
 PycairoFontFace (C type), 163  
 PycairoFontFace.PycairoFontFace.font\_face (C member), 163  
 PycairoFontFace\_FromFontFace (C function), 163  
 PycairoFontFace\_Type (C type), 163  
 PycairoFontOptions (C type), 163  
 PycairoFontOptions.PycairoFontOptions.font\_options (C member), 163  
 PycairoFontOptions\_FromFontOptions (C function), 163  
 PycairoFontOptions\_Type (C type), 163  
 PycairoGradient (C type), 164  
 PycairoGradient\_Type (C type), 164  
 PycairoImageSurface (C type), 166  
 PycairoImageSurface\_Type (C type), 166  
 PycairoLinearGradient (C type), 164  
 PycairoLinearGradient\_Type (C type), 165  
 PycairoMatrix (C type), 164  
 PycairoMatrix.PycairoMatrix.matrix (C member), 164  
 PycairoMatrix\_FromMatrix (C function), 164  
 PycairoMatrix\_Type (C type), 164  
 PycairoPath (C type), 164  
 PycairoPath.PycairoPath.path (C member), 164  
 PycairoPath\_FromPath (C function), 164  
 PycairoPath\_Type (C type), 164  
 PycairoPattern (C type), 164  
 PycairoPattern.PycairoPattern.pattern (C member), 164  
 PycairoPattern\_FromPattern (C function), 165  
 PycairoPattern\_Type (C type), 164  
 PycairoPDFSurface (C type), 166  
 PycairoPDFSurface\_Type (C type), 166  
 PycairoPSSurface (C type), 166  
 PycairoPSSurface\_Type (C type), 166  
 PycairoRadialGradient (C type), 165  
 PycairoRadialGradient\_Type (C type), 165  
 PycairoRecordingSurface (C type), 166  
 PycairoRecordingSurface\_Type (C type), 166  
 PycairoRectangleInt (C type), 165

PycairoRectangleInt.PycairoRectangleInt.rectangle\_int (C member), 165  
PycairoRectangleInt\_FromRectangleInt (C function), 165  
PycairoRectangleInt\_Type (C type), 165  
PycairoRegion (C type), 165  
PycairoRegion.PycairoRegion.region (C member), 165  
PycairoRegion\_FromRegion (C function), 165  
PycairoRegion\_Type (C type), 165  
PycairoScaledFont (C type), 166  
PycairoScaledFont.PycairoScaledFont.scaled\_font (C member), 166  
PycairoScaledFont\_FromScaledFont (C function), 166  
PycairoScaledFont\_Type (C type), 166  
PycairoSolidPattern (C type), 164  
PycairoSolidPattern\_Type (C type), 164  
PycairoSurface (C type), 166  
PycairoSurface.PycairoSurface.surface (C member), 166  
PycairoSurface\_FromSurface (C function), 167  
PycairoSurface\_Type (C type), 166  
PycairoSurfacePattern (C type), 164  
PycairoSurfacePattern\_Type (C type), 164  
PycairoSVGSurface (C type), 166  
PycairoSVGSurface\_Type (C type), 166  
PycairoToyFontFace (C type), 163  
PycairoToyFontFace\_Type (C type), 163  
PycairoWin32Surface (C type), 166  
PycairoWin32Surface\_Type (C type), 166  
PycairoXCBSurface (C type), 166  
PycairoXCBSurface\_Type (C type), 166  
PycairoXlibSurface (C type), 166  
PycairoXlibSurface\_Type (C type), 166

## R

RadialGradient (class in cairo), 119  
RasterSourcePattern (class in cairo), 125  
RasterSourcePattern.acquire() (in module cairo), 125  
RasterSourcePattern.release() (in module cairo), 126  
READ\_ERROR (cairo.Status attribute), 90  
RECORD\_OPERATIONS (cairo.SurfaceObserverMode attribute), 91  
RecordingSurface (class in cairo), 139  
Rectangle (class in cairo), 152  
rectangle() (cairo.Context method), 102  
RectangleInt (class in cairo), 128  
REFLECT (cairo.Extend attribute), 84  
Region (class in cairo), 126  
REGION\_OVERLAP\_IN (in module cairo), 159  
REGION\_OVERLAP\_OUT (in module cairo), 159  
REGION\_OVERLAP\_PART (in module cairo), 159  
RegionOverlap (class in cairo), 90  
rel\_curve\_to() (cairo.Context method), 102  
rel\_line\_to() (cairo.Context method), 103  
rel\_move\_to() (cairo.Context method), 103  
release() (cairo.Device method), 150

remove() (cairo.TeeSurface method), 144  
REPEAT (cairo.Extend attribute), 84  
reset\_clip() (cairo.Context method), 103  
restore() (cairo.Context method), 103  
restrict\_to\_level() (cairo.PSSurface method), 138  
restrict\_to\_version() (cairo.PDFSurface method), 136  
restrict\_to\_version() (cairo.SVGSurface method), 140  
RGB (cairo.SubpixelOrder attribute), 90  
RGB16\_565 (cairo.Format attribute), 86  
RGB24 (cairo.Format attribute), 85  
RGB30 (cairo.Format attribute), 86  
rotate() (cairo.Context method), 103  
rotate() (cairo.Matrix method), 114  
ROUND (cairo.LineCap attribute), 86  
ROUND (cairo.LineJoin attribute), 87

## S

SATURATE (cairo.Operator attribute), 88  
save() (cairo.Context method), 103  
scale() (cairo.Context method), 103  
scale() (cairo.Matrix method), 114  
ScaledFont (class in cairo), 146  
SCREEN (cairo.Operator attribute), 88  
ScriptDevice (class in cairo), 151  
ScriptMode (class in cairo), 91  
ScriptSurface (class in cairo), 143  
select\_font\_face() (cairo.Context method), 104  
set\_acquire() (cairo.RasterSourcePattern method), 125  
set\_antialias() (cairo.Context method), 104  
set\_antialias() (cairo.FontOptions method), 149  
set\_control\_point() (cairo.MeshPattern method), 124  
set\_corner\_color\_rgb() (cairo.MeshPattern method), 124  
set\_corner\_color\_rgba() (cairo.MeshPattern method), 124  
set\_dash() (cairo.Context method), 104  
set\_device\_offset() (cairo.Surface method), 131  
set\_device\_scale() (cairo.Surface method), 133  
set\_eps() (cairo.PSSurface method), 139  
set\_extend() (cairo.Pattern method), 116  
set\_fallback\_resolution() (cairo.Surface method), 131  
set\_fill\_rule() (cairo.Context method), 105  
set\_filter() (cairo.Pattern method), 116  
set\_font\_face() (cairo.Context method), 105  
set\_font\_matrix() (cairo.Context method), 105  
set\_font\_options() (cairo.Context method), 105  
set\_font\_size() (cairo.Context method), 105  
set\_hint\_metrics() (cairo.FontOptions method), 149  
set\_hint\_style() (cairo.FontOptions method), 149  
set\_line\_cap() (cairo.Context method), 105  
set\_line\_join() (cairo.Context method), 106  
set\_line\_width() (cairo.Context method), 106  
set\_matrix() (cairo.Context method), 106  
set\_matrix() (cairo.Pattern method), 116  
set\_mime\_data() (cairo.Surface method), 130  
set\_miter\_limit() (cairo.Context method), 106

- set\_mode() (cairo.ScriptDevice method), 151  
 set\_operator() (cairo.Context method), 107  
 set\_scaled\_font() (cairo.Context method), 107  
 set\_size() (cairo.PDFSurface method), 135  
 set\_size() (cairo.PSSurface method), 139  
 set\_size() (cairo.XCBSurface method), 142  
 set\_source() (cairo.Context method), 107  
 set\_source\_rgb() (cairo.Context method), 107  
 set\_source\_rgba() (cairo.Context method), 107  
 set\_source\_surface() (cairo.Context method), 108  
 set\_subpixel\_order() (cairo.FontOptions method), 149  
 set\_tolerance() (cairo.Context method), 108  
 show\_glyphs() (cairo.Context method), 108  
 show\_page() (cairo.Context method), 108  
 show\_page() (cairo.Surface method), 131  
 show\_text() (cairo.Context method), 108  
 show\_text\_glyphs() (cairo.Context method), 111  
 SLIGHT (cairo.HintStyle attribute), 86  
 SOFT\_LIGHT (cairo.Operator attribute), 88  
 SolidPattern (class in cairo), 117  
 SOURCE (cairo.Operator attribute), 87  
 SQUARE (cairo.LineCap attribute), 87  
 status (cairo.Error attribute), 112  
 Status (class in cairo), 90  
 STATUS\_CLIP\_NOT\_REPRESENTABLE (in module cairo), 159  
 STATUS\_DEVICE\_ERROR (in module cairo), 159  
 STATUS\_DEVICE\_FINISHED (in module cairo), 159  
 STATUS\_DEVICE\_TYPE\_MISMATCH (in module cairo), 159  
 STATUS\_FILE\_NOT\_FOUND (in module cairo), 159  
 STATUS\_FONT\_TYPE\_MISMATCH (in module cairo), 159  
 STATUS\_INVALID\_CLUSTERS (in module cairo), 159  
 STATUS\_INVALID\_CONTENT (in module cairo), 159  
 STATUS\_INVALID\_DASH (in module cairo), 159  
 STATUS\_INVALID\_DSC\_COMMENT (in module cairo), 159  
 STATUS\_INVALID\_FORMAT (in module cairo), 159  
 STATUS\_INVALID\_INDEX (in module cairo), 159  
 STATUS\_INVALID\_MATRIX (in module cairo), 159  
 STATUS\_INVALID\_MESH\_CONSTRUCTION (in module cairo), 159  
 STATUS\_INVALID\_PATH\_DATA (in module cairo), 159  
 STATUS\_INVALID\_POP\_GROUP (in module cairo), 159  
 STATUS\_INVALID\_RESTORE (in module cairo), 159  
 STATUS\_INVALID\_SIZE (in module cairo), 159  
 STATUS\_INVALID\_SLANT (in module cairo), 159  
 STATUS\_INVALID\_STATUS (in module cairo), 159  
 STATUS\_INVALID\_STRIDE (in module cairo), 159  
 STATUS\_INVALID\_STRING (in module cairo), 159  
 STATUS\_INVALID\_VISUAL (in module cairo), 159  
 STATUS\_INVALID\_WEIGHT (in module cairo), 159  
 STATUS\_LAST\_STATUS (in module cairo), 159  
 STATUS\_NEGATIVE\_COUNT (in module cairo), 159  
 STATUS\_NO\_CURRENT\_POINT (in module cairo), 159  
 STATUS\_NO\_MEMORY (in module cairo), 159  
 STATUS\_NULL\_POINTER (in module cairo), 159  
 STATUS\_PATTERN\_TYPE\_MISMATCH (in module cairo), 159  
 STATUS\_READ\_ERROR (in module cairo), 159  
 STATUS\_SUCCESS (in module cairo), 159  
 STATUS\_SURFACE\_FINISHED (in module cairo), 159  
 STATUS\_SURFACE\_TYPE\_MISMATCH (in module cairo), 159  
 STATUS\_TEMP\_FILE\_ERROR (in module cairo), 159  
 STATUS\_USER\_FONT\_ERROR (in module cairo), 159  
 STATUS\_USER\_FONT\_IMMUTABLE (in module cairo), 159  
 STATUS\_USER\_FONT\_NOT\_IMPLEMENTED (in module cairo), 159  
 STATUS\_WRITE\_ERROR (in module cairo), 159  
 stride\_for\_width() (cairo.Format method), 85  
 stroke() (cairo.Context method), 109  
 stroke\_extents() (cairo.Context method), 109  
 stroke\_preserve() (cairo.Context method), 109  
 stroke\_to\_path() (cairo.Context method), 112  
 SUBPIXEL (cairo.Antialias attribute), 83  
 SUBPIXEL\_ORDER\_BGR (in module cairo), 159  
 SUBPIXEL\_ORDER\_DEFAULT (in module cairo), 159  
 SUBPIXEL\_ORDER\_RGB (in module cairo), 159  
 SUBPIXEL\_ORDER\_VBGR (in module cairo), 159  
 SUBPIXEL\_ORDER\_VRGB (in module cairo), 159  
 SubpixelOrder (class in cairo), 89  
 subtract() (cairo.Region method), 127  
 SUCCESS (cairo.Status attribute), 90  
 supports\_mime\_type() (cairo.Surface method), 130  
 Surface (class in cairo), 128  
 SURFACE\_FINISHED (cairo.Status attribute), 90  
 SURFACE\_TYPE\_MISMATCH (cairo.Status attribute), 90  
 SurfaceObserverMode (class in cairo), 91  
 SurfacePattern (class in cairo), 117  
 SVG\_VERSION\_1\_1 (in module cairo), 158  
 SVG\_VERSION\_1\_2 (in module cairo), 159  
 SVGSurface (class in cairo), 140  
 SVGVersion (class in cairo), 89
- ## T
- TeeSurface (class in cairo), 143  
 TEMP\_FILE\_ERROR (cairo.Status attribute), 90  
 text (class in cairo), 82  
 text\_extents() (cairo.Context method), 110  
 text\_extents() (cairo.ScaledFont method), 147  
 text\_path() (cairo.Context method), 110

text\_to\_glyphs() (cairo.ScaledFont method), 148  
TextCluster (class in cairo), 152  
TextClusterFlags (class in cairo), 91  
TextExtents (class in cairo), 153  
ToyFontFace (class in cairo), 145  
transform() (cairo.Context method), 110  
transform\_distance() (cairo.Matrix method), 114  
transform\_point() (cairo.Matrix method), 114  
translate() (cairo.Context method), 110  
translate() (cairo.Matrix method), 115  
translate() (cairo.Region method), 127

## U

union() (cairo.Region method), 127  
unmap\_image() (cairo.Surface method), 133  
USER\_FONT\_ERROR (cairo.Status attribute), 90  
USER\_FONT\_IMMUTABLE (cairo.Status attribute), 90  
USER\_FONT\_NOT\_IMPLEMENTED (cairo.Status attribute), 90  
user\_to\_device() (cairo.Context method), 110  
user\_to\_device\_distance() (cairo.Context method), 111

## V

VBGR (cairo.SubpixelOrder attribute), 90  
version (in module cairo), 81  
VERSION\_1\_1 (cairo.SVGVersion attribute), 89  
VERSION\_1\_2 (cairo.SVGVersion attribute), 89  
VERSION\_1\_4 (cairo.PDFVersion attribute), 89  
VERSION\_1\_5 (cairo.PDFVersion attribute), 89  
version\_info (in module cairo), 81  
version\_to\_string() (cairo.PDFSurface static method), 136  
version\_to\_string() (cairo.SVGSurface static method), 141  
VRGB (cairo.SubpixelOrder attribute), 90

## W

width (cairo.Rectangle attribute), 152  
width (cairo.RectangleInt attribute), 128  
width (cairo.TextExtents attribute), 154  
Win32PrintingSurface (class in cairo), 141  
Win32Surface (class in cairo), 141  
WINDING (cairo.FillRule attribute), 84  
write\_comment() (cairo.ScriptDevice method), 151  
WRITE\_ERROR (cairo.Status attribute), 90  
write\_to\_png() (cairo.Surface method), 131

## X

x (cairo.Glyph attribute), 152  
x (cairo.Rectangle attribute), 152  
x (cairo.RectangleInt attribute), 128  
x\_advance (cairo.TextExtents attribute), 154  
x\_bearing (cairo.TextExtents attribute), 153

XCBSurface (class in cairo), 142  
XlibSurface (class in cairo), 142  
XOR (cairo.Operator attribute), 88  
xor() (cairo.Region method), 128

## Y

y (cairo.Glyph attribute), 152  
y (cairo.Rectangle attribute), 152  
y (cairo.RectangleInt attribute), 128  
y\_advance (cairo.TextExtents attribute), 154  
y\_bearing (cairo.TextExtents attribute), 153