python-sane Documentation

Release 2.8.2

Andrew Kuchling, Ralph Heinkel, Sandro Mani

August 07, 2015

Contents

The sane module is an Python interface to the SANE (Scanning is Now Easy) library, which provides access to various raster scanning devices such as flatbed scanners and digital cameras. For more information about SANE, consult the SANE website at www.sane-project.org. Note that this documentation doesn't duplicate all the information in the SANE documentation, which you must also consult to get a complete understanding.

This module has been originally developed by A.M. Kuchling, it is currently maintained by Sandro Mani.

CHAPTER 1

Indices

- genindex
- search

CHAPTER 2

Reference

sane.**init**()
Initialize sane.

Returns A tuple `(sane_ver, ver_maj, ver_min, ver_patch)`.

Raises _sane.error If an error occurs.

sane.**get_devices**(*localOnly=False*)
Return a list of 4-tuples containing the available scanning devices. If *localOnly* is *True*, only local devices will be returned. Each tuple is of the format `(device_name, vendor, model, type)`, with:

- *device_name* – The device name, suitable for passing to *sane.open()*.
- *vendor* – The device vendor.
- *mode* – The device model vendor.
- *type* – the device type, such as `"virtual device"` or `"video camera"`.

Returns A list of scanning devices.

Raises _sane.error If an error occurs.

sane.**open**(*devname*)
Open a device for scanning. Suitable values for devname are returned in the first item of the tuples returned by *sane.get_devices()*.

Returns A *SaneDev* object on success.

Raises _sane.error If an error occurs.

sane.**exit**()
Exit sane.

class sane.**SaneDev**(*devname*)
Class representing a SANE device. Besides the functions documented below, the class has some special attributes which can be read:

- ***devname* – The scanner device name (as passed to *sane.open()*).**
- *sane_signature* – The tuple `(devname, brand, name, type)`.
- *scanner_model* – The tuple `(brand, name)`.
- *opt* – Dictionary of options.
- *optlist* – List of option names.

- *area* – Scan area.

Furthermore, the scanner options are also exposed as attributes, which can be read and set:

```
print scanner.mode
scanner.mode = 'Color'
```

An *Option* object for a scanner option can be retrieved via `__getitem__()`, i.e.:

```
option = scanner['mode']
```

arr_scan()
Convenience method which calls *SaneDev.start()* followed by *SaneDev.arr_snap()*.

arr_snap()
Read image data and return a 3d numpy array of the shape `(nbands, width, heigth)`.

Returns A `numpy.array` object.

Raises

- **_sane.error** – If an error occurs.
- **RuntimeError** – If *numpy* cannot be imported.

cancel()
Cancel an in-progress scanning operation.

close()
Close the scanning device.

fileno()

Returns The file descriptor for the scanning device, if any.

Raises _sane.error If an error occurs.

get_options()

Returns A list of tuples describing all the available options.

get_parameters()
Returns a 5-tuple holding all the current device settings: `(format, last_frame, (pixels_per_line, lines), depth, bytes_per_line)`

- ***format* – One of "grey", "color", "red",** "green", "blue" or "unknown format".
- ***last_frame* – Whether this is the last frame of a multi frame** image.
- *pixels_per_line* – Width of the scanned image.
- *lines* – Height of the scanned image.
- *depth* – The number of bits per sample.
- *bytes_per_line* – The number of bytes per line.

Returns A tuple containing the device settings.

Raises _sane.error If an error occurs.

Note: Some backends may return different parameters depending on whether *SaneDev.start()* was called or not.

multi_scan()

Returns A *_SaneIterator* for ADF scans.

scan()
Convenience method which calls *SaneDev.start()* followed by *SaneDev.snap()*.

snap(*no_cancel=False*)
Read image data and return a PIL.Image object. An RGB image is returned for multi-band images, a L image for single-band images. no_cancel is used for ADF scans by *_SaneIterator*.

Returns A PIL.Image object.

Raises

- **_sane.error** – If an error occurs.
- **RuntimeError** – If *PIL.Image* cannot be imported.

start()
Initiate a scanning operation.

Throws _sane.error If an error occurs, for instance if a option is set to an invalid value.

class sane.**Option**(*args*, *scanDev*)
Class representing a SANE option. These are returned by a SaneDev.__getitem__() lookup of an option on the device, i.e.:

```
option = scanner["mode"]
```

The *Option* class has the following attributes:

- *index* – Number from 0 to n, giving the option number.
- *name* – A string uniquely identifying the option.
- *title* – Single-line string containing a title for the option.
- ***desc* – A long string describing the option, useful as a help** message.
- ***type* – Type of this option: TYPE_BOOL, TYPE_INT,** TYPE_STRING, etc.
- *unit* – Units of this option. UNIT_NONE, UNIT_PIXEL, etc.
- *size* – Size of the value in bytes.
- ***cap* – Capabilities available: CAP_EMULATED, CAP_SOFT_SELECT,** etc.
- *constraint* – Constraint on values. Possible values:
  - None : No constraint
  - (min,max,step) : Range
  - list of integers or strings: listed of permitted values

is_active()

Returns Whether the option is active.

is_settable()

Returns Whether the option is settable.

class sane.**_SaneIterator**(*device*)
Iterator for ADF scans.

CHAPTER 3

Example

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

from __future__ import print_function
import sane
import numpy
from PIL import Image

#
# Change these for 16bit / grayscale scans
#
depth = 8
mode = 'color'

#
# Initialize sane
#
ver = sane.init()
print('SANE version:', ver)

#
# Get devices
#
devices = sane.get_devices()
print('Available devices:', devices)

#
# Open first device
#
dev = sane.open(devices[0][0])

#
# Set some options
#
params = dev.get_parameters()
try:
    dev.depth = depth
except:
    print('Cannot set depth, defaulting to %d' % params[3])

try:
    dev.mode = mode
except:
    print('Cannot set mode, defaulting to %s' % params[0])
```

```
44
45  try:
46      dev.br_x = 320.
47      dev.br_y = 240.
48  except:
49      print('Cannot set scan area, using default')
50
51  params = dev.get_parameters()
52  print('Device parameters:', params)
53
54  #
55  # Start a scan and get and PIL.Image object
56  #
57  dev.start()
58  im = dev.snap()
59  im.save('test_pil.png')
60
61
62  #
63  # Start another scan and get a numpy array object
64  #
65  # Initiate the scan and get and numpy array
66  dev.start()
67  arr = dev.arr_snap()
68  print("Array shape: %s, size: %d, type: %s, range: %d-%d, mean: %.1f, stddev: "
69        "%.1f" % (repr(arr.shape), arr.size, arr.dtype, arr.min(), arr.max(),
70                  arr.mean(), arr.std()))
71
72  if arr.dtype == numpy.uint16:
73      arr = (arr / 255).astype(numpy.uint8)
74
75  if params[0] == 'color':
76      im = Image.frombytes('RGB', arr.shape[1:], arr.tostring(), 'raw', 'RGB', 0,
77                           1)
78  else:
79      im = Image.frombytes('L', arr.shape[1:], arr.tostring(), 'raw', 'L', 0, 1)
80
81  im.save('test_numpy.png')
82
83  #
84  # Close the device
85  #
86  dev.close()
```

S

Symbols

A

C

E

F

G

I

M

O

S