
ScopeSim Documentation

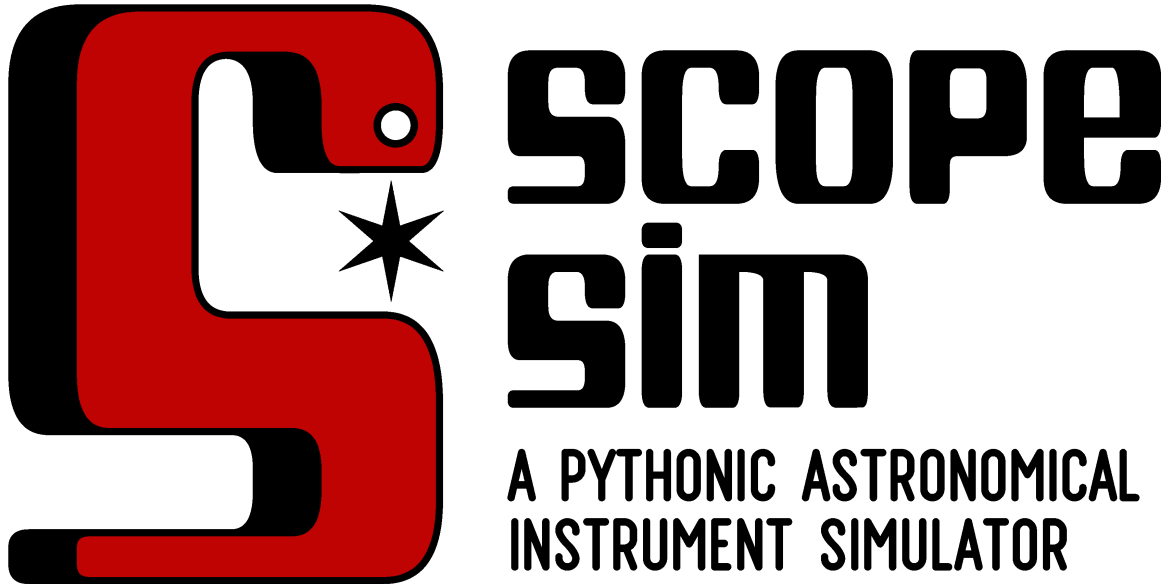
Kieran Leschinski

Feb 15, 2024

CONTENTS:

1	Code breakdown	3
2	Tips and tricks	5
2.1	Focal plane images	5
2.2	Turning optical effects on or off	5
2.3	Listing available modes and filters	5
2.4	Setting observation sequence	6
2.5	Listing and changing simulation parameters	6
3	More information	7
4	Contact	9
5	Examples	11
5.1	1: A quick use case for MICADO at the ELT	11
5.2	2: Observing the same object with multiple telescopes	14
5.3	3: Writing and including custom Effects	19
6	Hints and Tricks	31
6.1	Using !-string and #-string commands	31
6.2	Turning Effect objects on or off	32
6.3	Downloading packages	32
6.4	Science target templates	33
6.5	Global rc simulation parameters	34
6.6	Source : from FITS images	35
6.7	Source : Point sources from arrays	36
7	FAQs	39
7.1	Binding ScopeSim to a local copy of the IRDB	39
8	Effect Descriptions	41
8.1	apertures	41
8.2	detector_list	44
8.3	effects	46
8.4	electronic	46
8.5	fits_headers	49
8.6	metis_lms_trace_list	54
8.7	obs_strategies	55
8.8	psfs	56
8.9	rotation	58
8.10	shifts	58

8.11	shutter	59
8.12	spectral_efficiency	59
8.13	spectral_trace_list	60
8.14	surface_list	63
8.15	ter_curves	63
9	scopesim	69
9.1	scopesim package	69
10	The ScopeSim python ecosystem	169
11	Contact	171
	Python Module Index	173
	Index	175



An attempt at creating a common pythonic framework for telescope instrument data simulators.

ScopeSim is on pip:

```
pip install scopesim
```

ScopeSim [templates](#) provides templates for creating on-sky sources:

```
pip install scopesim_templates
```

Warning: July 2022: The downloadable content server was retired and the data migrated to a new server.

ScopeSim v0.5.1 and above have been redirected to a new server URL.

For older versions, please either upgrade to the latest version (`pip install --upgrade scopesim`), or follow these [instructions to update the server URL](#) in the config file.

Note: For instrument specific guides, please see the [IRDB](#)

A basic simulation would look something like this:

```
[ ]: from matplotlib import pyplot as plt
    from matplotlib.colors import LogNorm

    import scopesim as sim
    from scopesim.source import source_templates as st

    src = st.star_field(n=100,
                        mmax=15,      # [mag]
                        mmin=20,
                        width=200)    # [arcsec]
```

(continues on next page)

(continued from previous page)

```
opt = sim.load_example_optical_train()
opt.cmds["!OBS.dit"] = 60          # [s]
opt.cmds["!OBS.ndit"] = 10

opt.observe(src)
hdulist = opt.readout()[0]

plt.figure(figsize=(10,8))
plt.imshow(hdulist[1].data, norm=LogNorm(vmin=1))
plt.colorbar()
```

CODE BREAKDOWN

Let's break this down a bit.

There are three major components of any simulation workflow:

1. the target description,
2. the telescope/instrument model, and
3. the observation.

For the target description we are using the ScopeSim internal template functions from `scopesim.source.source_templates`, however many more dedicated science related templates are available in the external python package [ScopeSim-Templates](#)

Here we create a field of 100 A0V stars with Vega magnitudes between $V=15$ and $V=20$ within a box of 200 arcsec:

```
[ ]: src = st.star_field(n=100,
                        mmax=15,      # [mag]
                        mmin=20,
                        width=200)    # [arcsec]
```

Next we load the sample optical train object from ScopeSim.

Normally we will want to use an actual instrument. Dedicated documentation for real telescope+instrument systems can be found in the documentation sections of the individual instruments in the [Instrument Reference Database \(IRDB\) documentation](#)

For real instruments loading the optical system generally follows a different pattern:

```
cmd = sim.UserCommands(use_instrument="instrument_name", set_modes=["mode_1", "mode_2"])
opt = sim.OpticalTrain(cmds)
```

Once we have loaded the instrument, we can set the observation parameters by accessing the internal commands dictionary:

```
[ ]: opt = sim.load_example_optical_train(set_modes=["imaging"])
opt.cmds["!OBS.dit"] = 60      # [s]
opt.cmds["!OBS.ndit"] = 10
```

Finally we observe the target source and readout the detectors.

What is returned (`hdulist`) is an `astropy.fits.HDUList` object which can be saved to disk in the standard way, or manipulated in a python session.

```
[ ]: opt.observe(src)
hdulist = opt.readout()[0]
```


TIPS AND TRICKS

2.1 Focal plane images

Intermediate frames of the focal plane image without the noise properties can be accessed by looking inside the optical train object and accessing the first image plane:

```
[ ]: noiseless_image = opt.image_planes[0].data
```

2.2 Turning optical effects on or off

All effects modelled by the optical train can be listed with the `.effects` attribute:

```
[ ]: opt.effects
```

These can be turned on or off by using their name and the `.include` attribute:

```
[ ]: opt["detector_linearity"].include = False
```

2.3 Listing available modes and filters

The list of observing modes can be found by using the `.modes` attribute of the commands objects:

```
[ ]: opt.cmds.modes
```

The names of included filters can be found in the filter effect. Use the name of the filter object from the table above to list these:

```
[ ]: opt["filter_wheel"].filters
```

2.4 Setting observation sequence

Although this could be different for some instruments, most instruments use the `exptime = ndit * dit` format. `ndit` and `dit` are generally accessible in the top level `!OBS` dictionary of the command object in the optical train.

```
[ ]: opt.cmds["!OBS.dit"] = 60          # [s]
     opt.cmds["!OBS.ndit"] = 10
```

2.5 Listing and changing simulation parameters

The command dictionary inside the optical system contains all the necessary parameters.

```
[ ]: opt.cmds
```

The command object is a series of nested dictionaries that can be accessed using the `!-string` format:

```
opt.cmds["!<alias>.<param>"]
opt.cmds["!<alias>.<sub_dict>.<param>"]
```

For example, setting the atmospheric background level is achieved thusly:

```
[ ]: opt.cmds["!ATMO.background.filter_name"] = "K"
     opt.cmds["!ATMO.background.value"] = 13.6
```

MORE INFORMATION

For more information on how to use ScopeSim be see:

- *Use Examples*
- Instrument Specific Documentation
- Effect data formats
- **`Setting up a custom instrument <>`** [__](#)

CONTACT

- For bugs, please add an [issue to the github repo](#)
- For enquiries on implementing your own instrument package, please drop us a line at astar.astro@univie.ac.at or kieran.leschinski@univie.ac.at

EXAMPLES

5.1 1: A quick use case for MICADO at the ELT

5.1.1 A brief introduction into using ScopeSim to observe a cluster in the LMC

This is a step-by-step guide. The complete script can be found at the bottom of this page/notebook.

First set up all relevant imports:

```
[1]: import matplotlib.pyplot as plt
    from matplotlib.colors import LogNorm
    %matplotlib inline

    import scopesim as sim
    import scopesim_templates as sim_tp

    updating/loading 'index.yml'
    Downloading https://scopesim.univie.ac.at/spextra/database/index.yml [Done]
    updating/loading 'default_filters.yml'
    Downloading https://scopesim.univie.ac.at/spextra/database/default_filters.yml [Done]
    updating/loading 'default_spectra.yml'
    Downloading https://scopesim.univie.ac.at/spextra/database/default_spectra.yml [Done]
    updating/loading 'default_curves.yml'
    Downloading https://scopesim.univie.ac.at/spextra/database/default_curves.yml [Done]
```

Scopesim works by using so-called instrument packages, which have to be downloaded separately. For normal use, you would set the package directory (a local folder path, `local_package_folder` in this example), download the required packages *once*, and then **remove the download command**.

```
[2]: local_package_folder = "./inst_pkgs"
```

However, to be able to run this example on the *Readthedocs* page, we need to include a temporary directory.

Do not copy and run this code locally, it is **only** needed to set things up for *Readthedocs*!

```
[3]: from tempfile import TemporaryDirectory
    local_package_folder = TemporaryDirectory().name
```

Download the required instrument packages for an observation with MICADO at the ELT.

Again, you would only need to do this **once**, not every time you run the rest of the script, assuming you set a (permanent) instrument package folder.

```
[4]: sim.rc.__config__["!SIM.file.local_packages_path"] = local_package_folder
sim.download_packages(["Armazones", "ELT", "MORFEO", "MICADO"])

astar.scopesim.server.database - Gathering information from server ...
astar.scopesim.server.database - Connection successful, starting download ...
```

```
Downloading Armazones: 100%| 74.3k/74.3k [00:00<00:00, 704kB/s]
Extracting Armazones: 100%| 10/10 [00:00<00:00, 3191.53it/s]
Downloading ELT : 100%| 53.2k/53.2k [00:00<00:00, 33.7MB/s]
Extracting ELT : 100%| 16/16 [00:00<00:00, 5408.95it/s]
Downloading MORFEO : 100%| 1.38M/1.38M [00:02<00:00, 599kB/s]
Extracting MORFEO : 100%| 12/12 [00:00<00:00, 959.85it/s]
Downloading MICADO : 100%| 14.3M/14.3M [00:25<00:00, 586kB/s]
Extracting MICADO : 100%| 120/120 [00:00<00:00, 873.99it/s]
```

```
[4]: [PosixPath('/tmp/tmpzkthwspw/Armazones.zip'),
PosixPath('/tmp/tmpzkthwspw/ELT.zip'),
PosixPath('/tmp/tmpzkthwspw/MORFEO.zip'),
PosixPath('/tmp/tmpzkthwspw/MICADO.zip')]
```

Now, create a star cluster using the `scopesim_templates` package. You can ignore the output that is sometimes printed. The `seed` argument is used to control the random number generation that creates the stars in the cluster. If this number is kept the same, the output will be consistent with each run, otherwise the position and brightness of the stars is randomised every time.

```
[5]: cluster = sim_tp.stellar.clusters.cluster(mass=1000,      # Msun
                                              distance=50000,  # parsec
                                              core_radius=0.3,  # parsec
                                              seed=9002)

imf - sample_imf: Setting maximum allowed mass to 1000
imf - sample_imf: Loop 0 added 1.26e+03 Msun to previous total of 0.00e+00 Msun
updating/loading 'filter_systems/etc/index.yml'
Downloading https://scopesim.univie.ac.at/spextra/database/filter_systems/etc/index.yml_
↪ [Done]
updating/loading 'filter_systems/etc/V.dat'
Downloading https://scopesim.univie.ac.at/spextra/database/filter_systems/etc/V.dat_
↪ [Done]
updating/loading 'libraries/ref/index.yml'
Downloading https://scopesim.univie.ac.at/spextra/database/libraries/ref/index.yml [Done]
updating/loading 'libraries/ref/vega.fits'
Downloading https://scopesim.univie.ac.at/spextra/database/libraries/ref/vega.fits [Done]
```

Next, make the MICADO optical system model with `OpticalTrain`. Observe the cluster `Source` object with the `.observe()` method and read out the MICADO detectors with `.readout()`. This may take a few moments on slower machines.

The resulting FITS file can either be returned as an `astropy.fits.HDUList` object, or saved to disk using the optional `filename` parameter

```
[6]: micado = sim.OpticalTrain("MICADO")
micado.observe(cluster)
hdus = micado.readout()
# micado.readout(filename="TEST.fits")
```



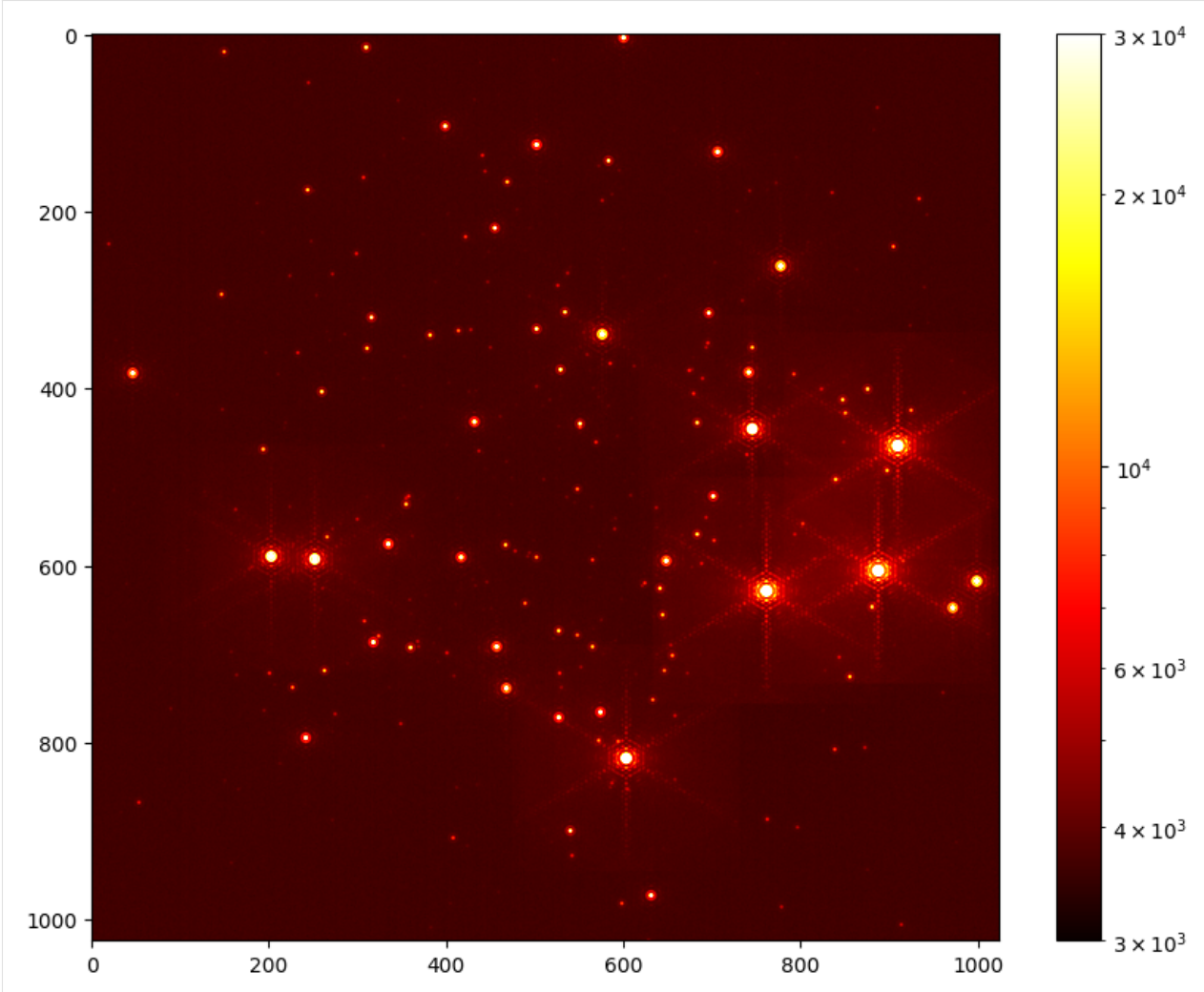
```
FOVs: 0%|          | 0/1 [00:00<?, ?it/s]
FOV effects: 0%|          | 0/2 [00:00<?, ?it/s]
FOV effects: 50%|          | 1/2 [00:00<00:00, 6.64it/s]
FOV effects: 100%|| 2/2 [00:00<00:00, 7.28it/s]
FOVs: 100%|| 1/1 [00:00<00:00, 2.00it/s]
Image Plane effects: 100%|| 1/1 [00:00<00:00, 4951.95it/s]
```

```
astar.scopesim.detector.detector_array - Extracting from 1 detectors...
```

Display the contents the first HDU

```
[7]: plt.figure(figsize=(10,8))
plt.imshow(hdus[0][1].data, norm=LogNorm(vmax=3E4, vmin=3E3), cmap="hot")
plt.colorbar()
```

```
[7]: <matplotlib.colorbar.Colorbar at 0x7f5f75d91050>
```



5.1.2 Complete script

Included below is the complete script for convenience, including the downloads, but not including the plotting.

```
[8]: import scopesim as sim
import scopesim_templates as sim_tp

#sim.download_packages(["Armazones", "ELT", "MORFEO", "MICADO"])

cluster = sim_tp.stellar.clusters.cluster(mass=1000,      # Msun
                                         distance=50000, # parsec
                                         core_radius=0.3, # parsec
                                         seed=9002)

micado = sim.OpticalTrain("MICADO")
micado.observe(cluster)

hdus = micado.readout()
# micado.readout(filename="TEST.fits")

imf - sample_imf: Setting maximum allowed mass to 1000
imf - sample_imf: Loop 0 added 1.26e+03 Msun to previous total of 0.00e+00 Msun

FOVs: 0%|          | 0/1 [00:00<?, ?it/s]
FOV effects: 0%|          | 0/2 [00:00<?, ?it/s]
FOV effects: 50%|      | 1/2 [00:00<00:00, 7.54it/s]
FOV effects: 100%|| 2/2 [00:00<00:00, 8.36it/s]
FOVs: 100%|| 1/1 [00:00<00:00, 2.19it/s]
Image Plane effects: 100%|| 1/1 [00:00<00:00, 5660.33it/s]

astar.scopesim.detector.detector_array - Extracting from 1 detectors...
```

```
[ ]:
```

5.2 2: Observing the same object with multiple telescopes

A brief introduction into using ScopeSim to observe a cluster in the LMC using the 39m ELT and the 1.5m LFOA

This is a step-by-step guide. The complete script can be found at the bottom of this page/notebook.

First set up all relevant imports:

```
[1]: import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
%matplotlib inline

import scopesim as sim
import scopesim_templates as sim_tp
```

Scopesim works by using so-called instrument packages, which have to be downloaded separately. For normal use, you would set the package directory (a local folder path, `local_package_folder` in this example), download the required packages *once*, and then **remove the download command**.

```
[2]: local_package_folder = "./inst_pkgs"
```

However, to be able to run this example on the *Readthedocs* page, we need to include a temporary directory.

Do not copy and run this code locally, it is **only** needed to set things up for *Readthedocs*!

```
[3]: from tempfile import TemporaryDirectory
local_package_folder = TemporaryDirectory().name
```

Download the packages for MICADO at the ELT and the viennese 1.5m telescope at the LFOA

Again, you would only need to do this **once**, not every time you run the rest of the script, assuming you set a (permanent) instrument package folder.

```
[4]: sim.rc.__config__["!SIM.file.local_packages_path"] = local_package_folder
sim.download_packages(["Armazones", "ELT", "MICADO", "MORFEO", "LFOA"])

astar.scopesim.server.database - Gathering information from server ...
astar.scopesim.server.database - Connection successful, starting download ...
```

```
Downloading Armazones: 100%| 74.3k/74.3k [00:00<00:00, 678kB/s]
Extracting Armazones: 100%| 10/10 [00:00<00:00, 3012.50it/s]
Downloading ELT      : 100%| 53.2k/53.2k [00:00<00:00, 29.6MB/s]
Extracting ELT      : 100%| 16/16 [00:00<00:00, 5510.66it/s]
Downloading MICADO   : 100%| 14.3M/14.3M [00:25<00:00, 585kB/s]
Extracting MICADO   : 100%| 120/120 [00:00<00:00, 874.85it/s]
Downloading MORFEO   : 100%| 1.38M/1.38M [00:02<00:00, 599kB/s]
Extracting MORFEO   : 100%| 12/12 [00:00<00:00, 970.40it/s]
Downloading LFOA     : 100%| 409k/409k [00:00<00:00, 634kB/s]
Extracting LFOA     : 100%| 45/45 [00:00<00:00, 3657.04it/s]
```

```
[4]: [PosixPath('/tmp/tmpi0a6c02j/Armazones.zip'),
PosixPath('/tmp/tmpi0a6c02j/ELT.zip'),
PosixPath('/tmp/tmpi0a6c02j/MICADO.zip'),
PosixPath('/tmp/tmpi0a6c02j/MORFEO.zip'),
PosixPath('/tmp/tmpi0a6c02j/LFOA.zip')]
```

5.2.1 Create a star cluster Source object

Now, create a star cluster using the `scopesim_templates` package. You can ignore the output that is sometimes printed. The seed argument is used to control the random number generation that creates the stars in the cluster. If this number is kept the same, the output will be consistent with each run, otherwise the position and brightness of the stars is randomised every time.

```
[5]: cluster = sim_tp.stellar.clusters.cluster(mass=10000,      # Msun
                                              distance=50000,   # parsec
                                              core_radius=2,    # parsec
                                              seed=9001)        # random seed

imf - sample_imf: Setting maximum allowed mass to 10000
imf - sample_imf: Loop 0 added 1.01e+04 Msun to previous total of 0.00e+00 Msun
```

5.2.2 Observe with the 1.5m telescope at the LFOA

`[2f1d2da07a734f6f861ea2524faa1c7e]`

```
[6]: lfoa = sim.OpticalTrain("LFOA")
lfoa.observe(cluster,
              properties={"!OBS.ndit": 10, "!OBS.ndit": 360},
              update=True)
hdus_lfoa = lfoa.readout()
```

```
FOVs: 0%|          | 0/1 [00:00<?, ?it/s]
FOV effects: 0%|          | 0/1 [00:00<?, ?it/s]
FOV effects: 100%|| 1/1 [00:00<00:00, 1.83it/s]
FOVs: 100%|| 1/1 [00:00<00:00, 1.14it/s]
Image Plane effects: 0it [00:00, ?it/s]
```

`astar.scopesim.detector.detector_array - Extracting from 1 detectors...`

`astar.scopesim.optics.optical_train - ERROR: Header update failed, data will be saved_`
`↳with incomplete header. See stack trace for details.`

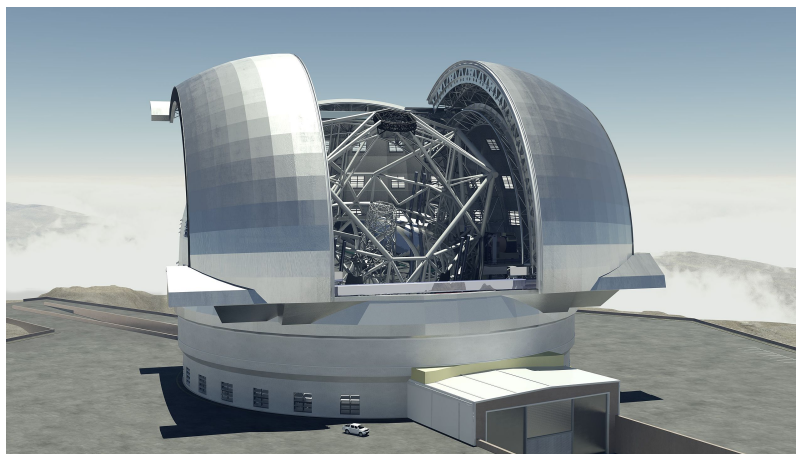
Traceback (most recent call last):

File `"/home/docs/checkouts/readthedocs.org/user_builds/scopesim/checkouts/hb-rtddownload/scopesim/optics/optical_train.py"`, line 383, in `readout`
`↳hdul = self.write_header(hdul)`
`AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`

File `"/home/docs/checkouts/readthedocs.org/user_builds/scopesim/checkouts/hb-rtddownload/scopesim/optics/optical_train.py"`, line 405, in `write_header`
`↳pheader["INSTRUME"] = from_currsys("!OBS.instrument", self.cmds)`
`AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`

File `"/home/docs/checkouts/readthedocs.org/user_builds/scopesim/checkouts/hb-rtddownload/scopesim/utls.py"`, line 882, in `from_currsys`
`↳raise ValueError(f"{item} was not found in rc.__currsys__")`
`ValueError: !OBS.instrument was not found in rc.__currsys__`

5.2.3 Observe the same Source with MICADO at the ELT



```
[7]: micado = sim.OpticalTrain("MICADO")
micado.cmds["!OBS.dit"] = 10
micado.cmds["!OBS.ndit"] = 360
micado.update()

micado.observe(cluster)
hdus_micado = micado.readout()

FOVs: 0%|      | 0/1 [00:00<?, ?it/s]
FOV effects: 0%|      | 0/2 [00:00<?, ?it/s]
FOV effects: 50%|      | 1/2 [00:00<00:00, 8.11it/s]
FOV effects: 100%|| 2/2 [00:00<00:00, 8.20it/s]
FOVs: 100%|| 1/1 [00:00<00:00, 2.19it/s]
Image Plane effects: 100%|| 1/1 [00:00<00:00, 4728.64it/s]

astar.scopesim.detector.detector_array - Extracting from 1 detectors...
```

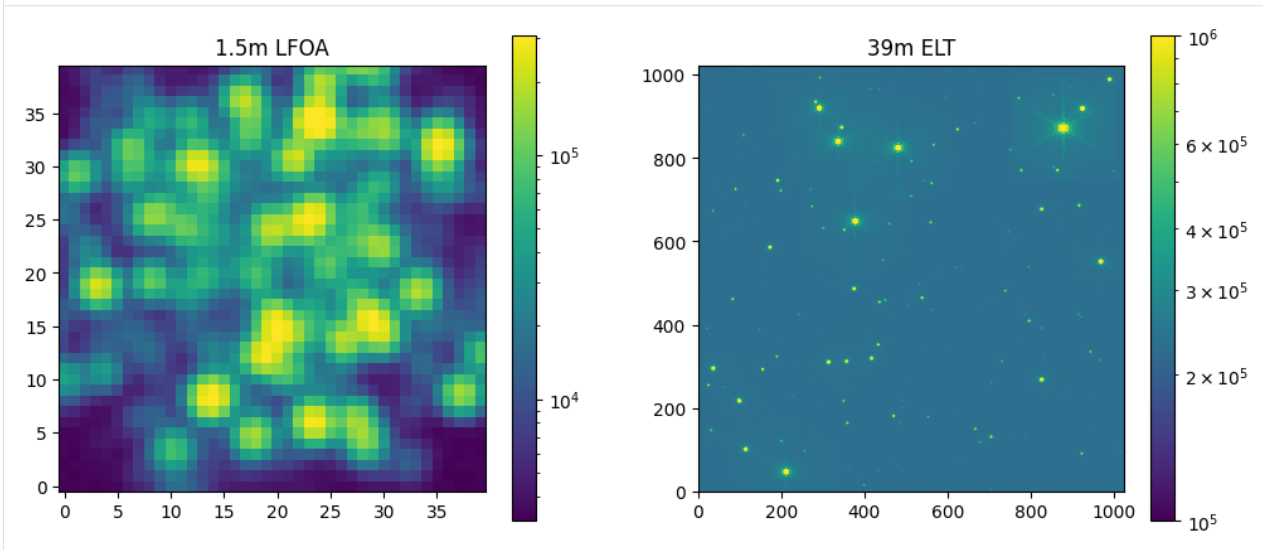
Plot up the results

```
[8]: plt.figure(figsize=(12,5))

plt.subplot(121)
plt.imshow(hdus_lfoa[0][1].data[345:385, 525:565], norm=LogNorm(), origin="lower")
plt.colorbar()
plt.title("1.5m LFOA")

plt.subplot(122)
plt.imshow(hdus_micado[0][1].data, norm=LogNorm(vmax=1E6, vmin=1e5), origin="lower")
plt.colorbar()
plt.title("39m ELT")

[8]: Text(0.5, 1.0, '39m ELT')
```



5.2.4 Complete script

Included below is the complete script for convenience, including the downloads, but not including the plotting.

```
[9]: import scopesim as sim
import scopesim_templates as sim_tp

# sim.download_packages(["Armazones", "ELT", "MICADO", "MORFEO", "LFOA"])

cluster = sim_tp.stellar.clusters.cluster(mass=10000,      # Msun
                                          distance=50000,  # parsec
                                          core_radius=2,   # parsec
                                          seed=9001)       # random seed

lfoa = sim.OpticalTrain("LFOA")
lfoa.observe(cluster,
               properties={"!OBS.ndit": 10, "!OBS.ndit": 360},
               update=True)
hdus_lfoa = lfoa.readout()

micado = sim.OpticalTrain("MICADO")
micado.cmds["!OBS.dit"] = 10
micado.cmds["!OBS.ndit"] = 360
micado.update()

micado.observe(cluster)
hdus_micado = micado.readout()
```

```
imf - sample_imf: Setting maximum allowed mass to 10000
imf - sample_imf: Loop 0 added 1.01e+04 Msun to previous total of 0.00e+00 Msun
```

```
FOVs:  0%|          | 0/1 [00:00<?, ?it/s]
FOV effects:  0%|          | 0/1 [00:00<?, ?it/s]
FOV effects: 100%|| 1/1 [00:00<00:00, 2.02it/s]
FOVs: 100%|| 1/1 [00:00<00:00, 1.24it/s]
Image Plane effects: 0it [00:00, ?it/s]
```

```
astar.scopesim.detector.detector_array - Extracting from 1 detectors...
```

```
astar.scopesim.optics.optical_train - ERROR: Header update failed, data will be saved_
↳with incomplete header. See stack trace for details.
Traceback (most recent call last):
  File "/home/docs/checkouts/readthedocs.org/user_builds/scopesim/checkouts/hb-
↳rtddownload/scopesim/optics/optical_train.py", line 383, in readout
    hdul = self.write_header(hdul)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/docs/checkouts/readthedocs.org/user_builds/scopesim/checkouts/hb-
↳rtddownload/scopesim/optics/optical_train.py", line 405, in write_header
    pheader["INSTRUME"] = from_currsys("!OBS.instrument", self.cmds)
                            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/docs/checkouts/readthedocs.org/user_builds/scopesim/checkouts/hb-
↳rtddownload/scopesim/utils.py", line 882, in from_currsys
    raise ValueError(f"{item} was not found in rc.__currsys__")
```

(continues on next page)

(continued from previous page)

```
ValueError: !OBS.instrument was not found in rc.__currsys__
```

```
FOVs: 0%|          | 0/1 [00:00<?, ?it/s]
FOV effects: 0%|          | 0/2 [00:00<?, ?it/s]
FOV effects: 50%|        | 1/2 [00:00<00:00, 8.14it/s]
FOV effects: 100%|| 2/2 [00:00<00:00, 8.98it/s]
FOVs: 100%|| 1/1 [00:00<00:00, 2.28it/s]
Image Plane effects: 100%|| 1/1 [00:00<00:00, 5691.05it/s]
```

```
astar.scopesim.detector.detector_array - Extracting from 1 detectors...
```

```
[ ]:
```

5.3 3: Writing and including custom Effects

In this tutorial, we will load the model of MICADO (including Armazones, ELT, MORFEO) and then turn off all effect that modify the spatial extent of the stars. The purpose here is to see in detail what happens to the **distribution of the stars flux on a sub-pixel level** when we add a plug-in astrometric Effect to the optical system.

For real simulation, we will obviously leave all normal MICADO effects turned on, while still adding the plug-in Effect. Hopefully this tutorial will serve as a reference for those who want to see **how to create Plug-ins** and how to manipulate the effects in the MICADO optical train model.

5.3.1 Create and optical model for MICADO and the ELT

```
[1]: from tempfile import TemporaryDirectory

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

import scopesim as sim
from scopesim_templates.stellar import stars, star_grid
```

Scopesim works by using so-called instrument packages, which have to be downloaded separately. For normal use, you would set the package directory (a local folder path, `local_package_folder` in this example), download the required packages *once*, and then **remove the download command**.

```
[2]: local_package_folder = "./inst_pkgs"
```

However, to be able to run this example on the *Readthedocs* page, we need to include a temporary directory.

Do not copy and run this code locally, it is **only** needed to set things up for *Readthedocs*!

```
[3]: from tempfile import TemporaryDirectory
local_package_folder = TemporaryDirectory().name
```

Download the required instrument packages for an observation with MICADO at the ELT.

Again, you would only need to do this **once**, not every time you run the rest of the script, assuming you set a (permanent) instrument package folder.


```
[4]: sim.download_packages(["Armazones", "ELT", "MICADO", "MORFEO"])

astar.scopesim.server.database - Gathering information from server ...
astar.scopesim.server.database - Connection successful, starting download ...

Downloading Armazones: 100%| 74.3k/74.3k [00:00<00:00, 672kB/s]
Extracting Armazones: 100%| 10/10 [00:00<00:00, 3343.94it/s]
Downloading ELT      : 100%| 53.2k/53.2k [00:00<00:00, 29.9MB/s]
Extracting ELT      : 100%| 16/16 [00:00<00:00, 5915.28it/s]
Downloading MICADO   : 100%| 14.3M/14.3M [00:25<00:00, 586kB/s]
Extracting MICADO   : 100%| 120/120 [00:00<00:00, 884.15it/s]
Downloading MORFEO   : 100%| 1.38M/1.38M [00:02<00:00, 598kB/s]
Extracting MORFEO   : 100%| 12/12 [00:00<00:00, 984.35it/s]

[4]: [PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/scopesim/checkouts/hb-
↳rtddownload/docs/source/examples/inst_pkgs/Armazones.zip'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/scopesim/checkouts/hb-
↳rtddownload/docs/source/examples/inst_pkgs/ELT.zip'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/scopesim/checkouts/hb-
↳rtddownload/docs/source/examples/inst_pkgs/MICADO.zip'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/scopesim/checkouts/hb-
↳rtddownload/docs/source/examples/inst_pkgs/MORFEO.zip')]
```

We can see which Effects are already included by calling `micado.effects`:

```
[5]: cmd = sim.UserCommands(use_instrument="MICADO", set_modes=["SCAO", "IMG_1.5mas"])
micado = sim.OpticalTrain(cmd)
```

```
micado.effects
```

```
[5]: <Table length=23>
```

element str13	name str23	class str31	included bool
armazones	skycalc_atmosphere	SkycalcTERCurve	True
ELT	telescope_reflection	SurfaceList	True
MICADO	micado_static_surfaces	SurfaceList	True
MICADO	micado_ncpas_psf	NonCommonPathAberration	True
MICADO	filter_wheel_1 : [open]	FilterWheel	True
MICADO	filter_wheel_2 : [Ks]	FilterWheel	True
MICADO	pupil_wheel : [open]	FilterWheel	True
MICADO_DET	full_detector_array	DetectorList	False
MICADO_DET	detector_window	DetectorWindow	True
MICADO_DET	qe_curve	QuantumEfficiencyCurve	True
...
MICADO_DET	detector_linearity	LinearityCurve	True
MICADO_DET	border_reference_pixels	ReferencePixelBorder	True
MICADO_DET	readout_noise	PoorMansHxRGReadoutNoise	True
MICADO_DET	source_fits_keywords	SourceDescriptionFitsKeywords	True
MICADO_DET	extra_fits_keywords	ExtraFitsKeywords	True
default_ro	relay_psf	FieldConstantPSF	True
default_ro	relay_surface_list	SurfaceList	True
default_ro	extra_fits_keywords_ro	ExtraFitsKeywords	True
MICADO_IMG_HR	zoom_mirror_list	SurfaceList	True
MICADO_IMG_HR	micado_adc_3D_shift	AtmosphericDispersionCorrection	False

Now we turn off all Effects that cause spatial aberrations:

```
[6]: for effect_name in ["full_detector_array", "micado_adc_3D_shift",
                        "micado_ncpas_psf", "relay_psf"]:
    micado[effect_name].include = False
    print(micado[effect_name])
```

```
DetectorList: "full_detector_array"
AtmosphericDispersionCorrection: "micado_adc_3D_shift"
NonCommonPathAberration: "micado_ncpas_psf"
FieldConstantPSF: "relay_psf"
```

The normal detector window is set to 1024 pixels square. Let's reduce the size of the detector readout window:

```
[7]: micado["detector_window"].data["x_cen"] = 0          # [mm] distance from optical axis,
    ↪ on the focal plane
micado["detector_window"].data["y_cen"] = 0
micado["detector_window"].data["x_size"] = 64           # [pixel] width of detector
micado["detector_window"].data["y_size"] = 64
```

By default ScopeSim works on the whole pixel level for saving computation time. However it is capable of integrating sub pixel shift. For this we need to turn on the sub-pixel mode:

```
[8]: micado.cmds["!SIM.sub_pixel.flag"] = True
```

We can test what's happening by making a grid of stars and observing them:

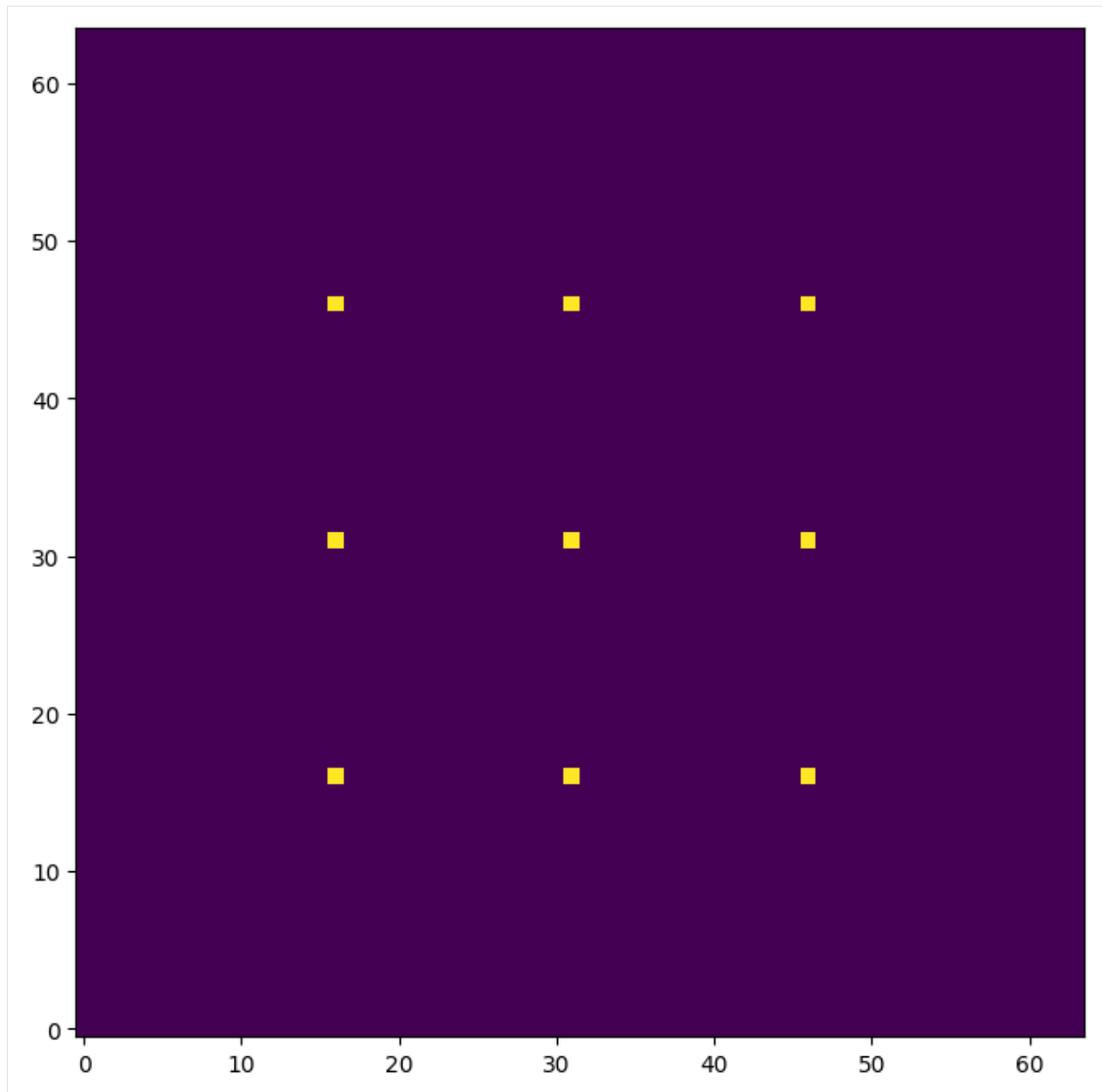
```
[9]: src = star_grid(n=9, mmin=20, mmax=20.0001, separation=0.0015 * 15)
src.fields[0]["x"] -= 0.00075
src.fields[0]["y"] -= 0.00075

micado.observe(src, update=True)

plt.figure(figsize=(8,8))
plt.imshow(micado.image_planes[0].data, origin="lower")

FOVs:   0%|          | 0/1 [00:00<?, ?it/s]
FOV effects: 0it [00:00, ?it/s]
FOVs: 100%|| 1/1 [00:00<00:00, 15.85it/s]
Image Plane effects: 100%|| 1/1 [00:00<00:00, 5115.00it/s]
```

```
[9]: <matplotlib.image.AxesImage at 0x7f2ac7271e10>
```



```
[10]: micado["detector_window"].data
```

```
[10]: <Table length=1>
      id  x_cen y_cen x_size y_size angle  gain pixel_size
int64  str6  str6 str10  str11  int64 int64  float64
-----
      0    0    0   64   64    0    1   0.015
```

5.3.2 Writing a custom Effect object

The following code snippet creates a new Effect class.

```
[11]: import numpy as np
from astropy.table import Table

from scopesim.effects import Effect
from scopesim.base_classes import SourceBase

class PointSourceJitter(Effect):
    def __init__(self, **kwargs):
        super(PointSourceJitter, self).__init__(**kwargs)  # initialise the underlying
        # Effect class object
        self.meta["z_order"] = [500]  # z_order number for knowing
        # when and how to apply the Effect
        self.meta["max_jitter"] = 0.001  # [arcsec] - a parameter
        # needed by the effect
        self.meta.update(kwargs)  # add any extra parameters
        # passed when initialising

    def apply_to(self, obj):  # the function that does the
        # work
        if isinstance(obj, SourceBase):
            for field in obj.fields:
                if isinstance(field, Table):
                    dx, dy = 2 * (np.random.random(size=(2, len(field))) - 0.5)
                    field["x"] += dx * self.meta["max_jitter"]
                    field["y"] += dy * self.meta["max_jitter"]

        return obj
```

Lets break it down a bit (THIS IS JUST A STEP-BY-STEP EXPLANATION OF THE CODE ABOVE, NOT SOMETHING NEW!):

```
class PointSourceJitter(Effect):
    ...
```

Here we are subclassing the Effect object from ScopeSim. This has the basic functionality for reading in ASCII and FITS files, and for communicating with the OpticsManager class in ScopeSim.

The initialisation function looks like this:

```
def __init__(self, **kwargs):
    super(PointSourceJitter, self).__init__(**kwargs)  # initialise the underlying
    # Effect class object
    self.meta["z_order"] = [500]
```

Here we make sure to activate the underlying Effect object. The `z_order` keyword in the meta dictionary is used by ScopeSim to determine when and where this Effect should be applied during a simulations run. The exact z-order numbers are described in [\[insert link here\]](#).

The main function of any Effect is the `apply_to` method:

```
def apply_to(self, obj):
    if isinstance(obj, SourceBase):
        ...

    return obj
```

It should be noted that what is passed in via (obj) must be returned in the same format. The contents of the obj can change, but the obj object must be returned.

All the code which enacts the results of the physical effect are contained in this method. For example, if we are writing a redshifting Effect, we could write the code to shift the wavelength array of a Source object by $z+1$ here.

There are 4 main classes that are cycled through during an observation run: * SourceBase: contains the original 2+1D distribution of light, * FieldOfViewBase: contains a (quasi-)monochromatic cutout from the Source object, * ImagePlaneBase: contains the expectation flux image on the detector plane * DetectorBase: contains the electronic readout image

An Effect object can be applied to any number of objects based on one or more of these base classes. Just remember to segregate the base-class-specific code with if statements.

One further method should be mentioned: def fov_grid(). This method is used by FOVManager to estimate how many FieldOfView objects to generate in order to best simulation the observation. If your Effect object might alter this estimate, then you should include this method in your class. See the code base for further details.

Note: The fov_grid method will be depreciated in a future release of ScopeSim. It will most likely be replaced by a FOVSetupBase class that will be cycled through the apply_to function. However this is not yet 100% certain, so please bear with us.

5.3.3 Including a custom Effect

First we need to initialise an instance of the Effect object:

```
[12]: jitter_effect = PointSourceJitter(max_jitter=0.001, name="random_jitter")
```

Then we can add it to the optical model:

```
[13]: micado.optics_manager.add_effect(jitter_effect)
```

```
micado.effects
```

```
[13]: <Table length=24>
```

element str13	name str23	class str31	included bool
armazones	skycalc_atmosphere	SkycalcTERCurve	True
armazones	random_jitter	PointSourceJitter	True
ELT	telescope_reflection	SurfaceList	True
MICADO	micado_static_surfaces	SurfaceList	True
MICADO	micado_ncpas_psf	NonCommonPathAberration	False
MICADO	filter_wheel_1 : [open]	FilterWheel	True
MICADO	filter_wheel_2 : [Ks]	FilterWheel	True
MICADO	pupil_wheel : [open]	FilterWheel	True
MICADO_DET	full_detector_array	DetectorList	False
MICADO_DET	detector_window	DetectorWindow	True
...

(continues on next page)

(continued from previous page)

MICADO_DET	detector_linearity	LinearityCurve	True
MICADO_DET	border_reference_pixels	ReferencePixelBorder	True
MICADO_DET	readout_noise	PoorMansHxRGReadoutNoise	True
MICADO_DET	source_fits_keywords	SourceDescriptionFitsKeywords	True
MICADO_DET	extra_fits_keywords	ExtraFitsKeywords	True
default_ro	relay_psf	FieldConstantPSF	False
default_ro	relay_surface_list	SurfaceList	True
default_ro	extra_fits_keywords_ro	ExtraFitsKeywords	True
MICADO_IMG_HR	zoom_mirror_list	SurfaceList	True
MICADO_IMG_HR	micado_adc_3D_shift	AtmosphericDispersionCorrection	False

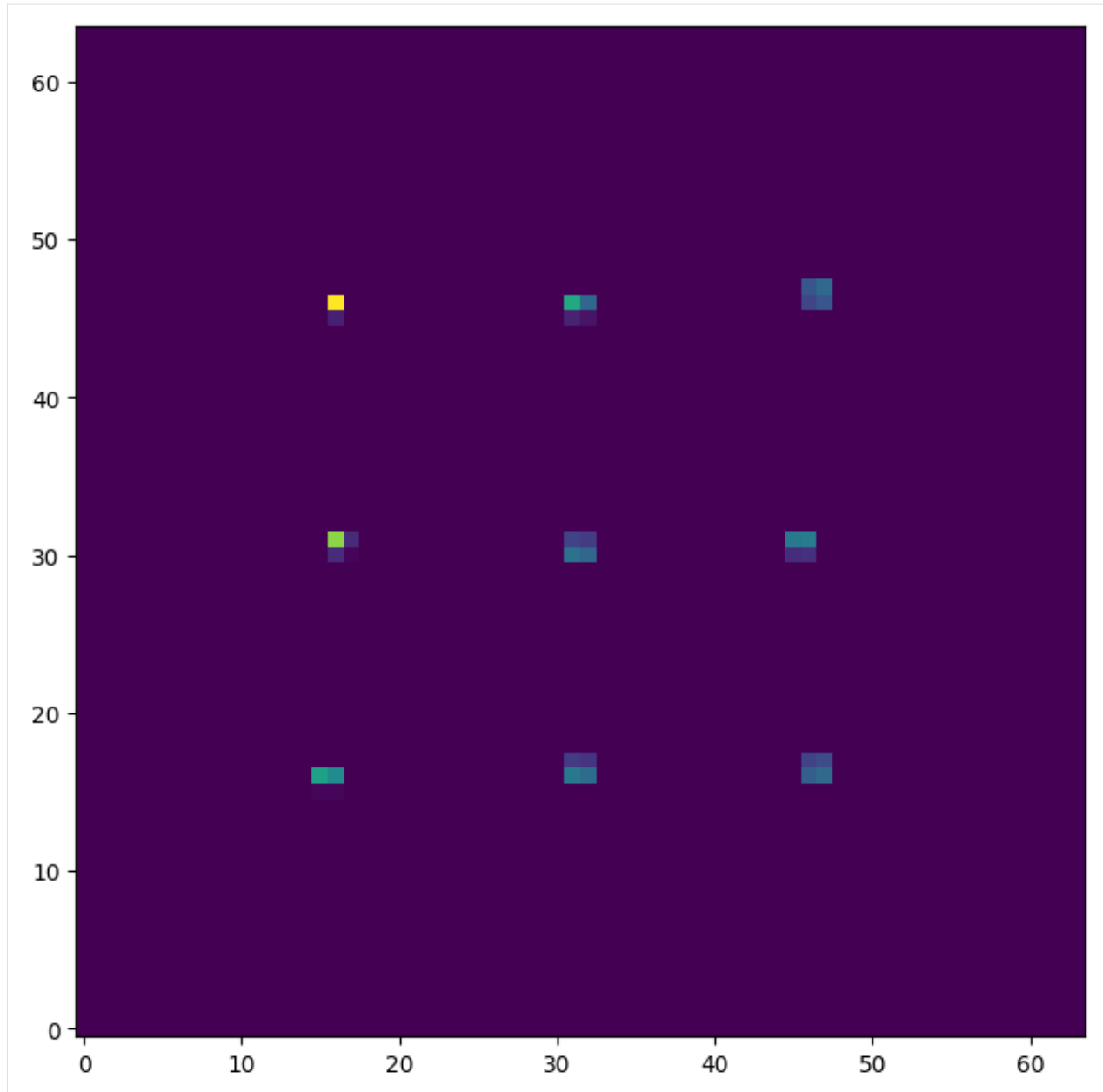
When we want to observe, we need to include the `update=True` flag so that the optical model is updated to include the instance of our new Effect:

```
[14]: micado.observe(src, update=True)
```

```
plt.figure(figsize=(8,8))
plt.imshow(micado.image_planes[0].data, origin="lower")
```

```
FOVs: 0%|          | 0/1 [00:00<?, ?it/s]
FOV effects: 0it [00:00, ?it/s]
FOVs: 100%|| 1/1 [00:00<00:00, 16.45it/s]
Image Plane effects: 100%|| 1/1 [00:00<00:00, 5691.05it/s]
```

```
[14]: <matplotlib.image.AxesImage at 0x7f2ac6fc2490>
```



We can update the parameters of the object on-the-fly by accessing the meta dictionary:

```
[15]: micado["random_jitter"].meta["max_jitter"] = 0.005
```

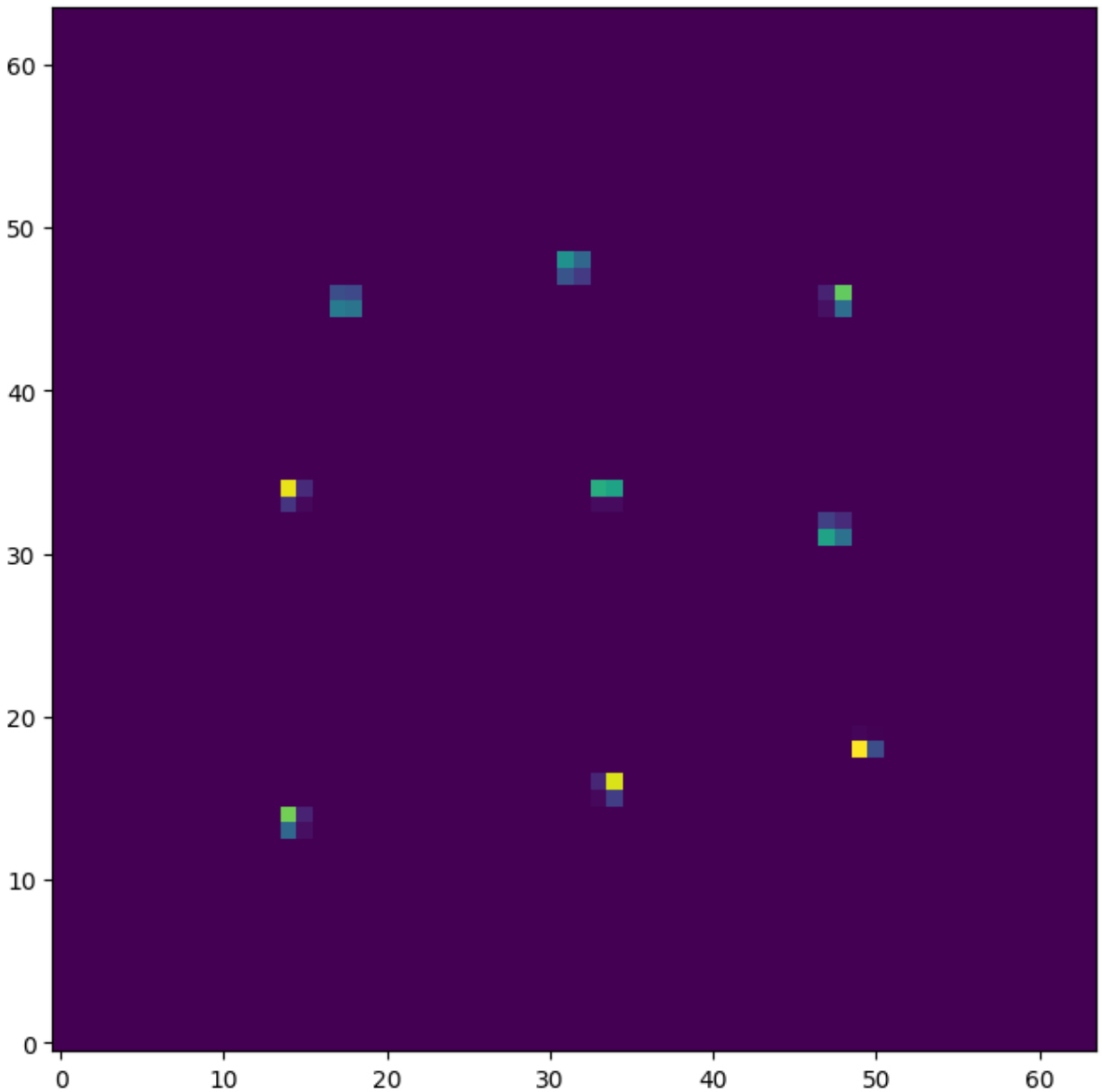
```
micado.observe(src, update=True)
```

```
plt.figure(figsize=(8,8))
```

```
plt.imshow(micado.image_planes[0].data, origin="lower")
```

```
FOVs: 0%|          | 0/1 [00:00<?, ?it/s]
FOV effects: 0it [00:00, ?it/s]
FOVs: 100%|| 1/1 [00:00<00:00, 16.03it/s]
Image Plane effects: 100%|| 1/1 [00:00<00:00, 5555.37it/s]
```

```
[15]: <matplotlib.image.AxesImage at 0x7f2ac6fbf510>
```



Here we can see that there is a certain amount of sub-pixel jitter being introduced into each observation. However this bare-bones approach is not very realistic. We should therefore turn the PSF back on to get a more realistic observation:

```
[16]: micado["relay_psf"].include = True

micado.observe(src, update=True)
hdus = micado.readout()

plt.figure(figsize=(8,8))
plt.imshow(hdus[0][1].data, origin="lower", norm=LogNorm())

FOVs:   0%|          | 0/1 [00:00<?, ?it/s]
```

(continues on next page)

(continued from previous page)

```
FOV effects: 0%|          | 0/1 [00:00<?, ?it/s]
```

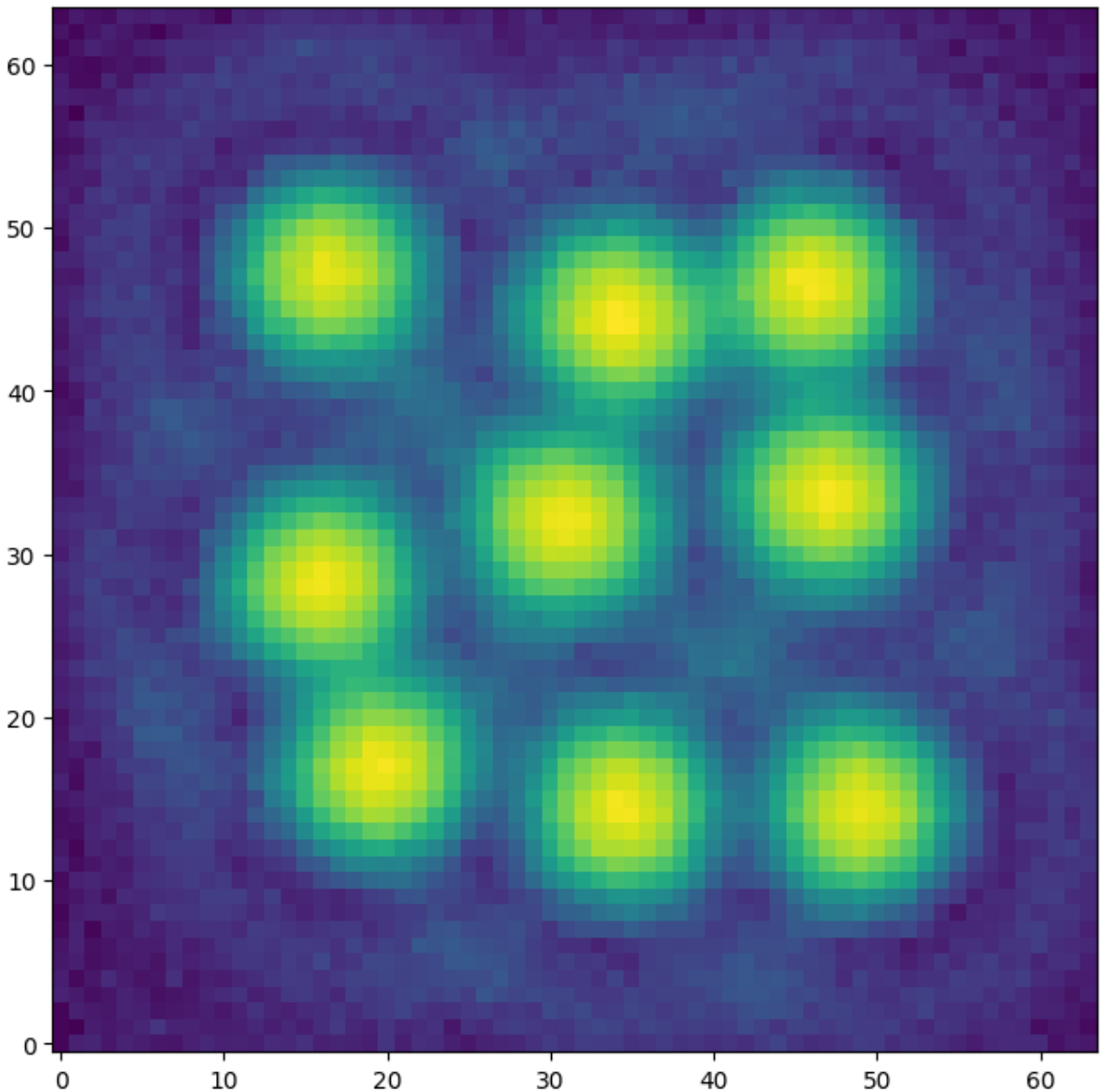
```
astar.scopesim.effects.psf_utils - WARNING: PSF center off
```

```
FOVs: 100%|| 1/1 [00:00<00:00, 11.95it/s]
```

```
Image Plane effects: 100%|| 1/1 [00:00<00:00, 5482.75it/s]
```

```
astar.scopesim.detector.detector_array - Extracting from 1 detectors...
```

```
[16]: <matplotlib.image.AxesImage at 0x7f2ac700e150>
```



[]:

HINTS AND TRICKS

6.1 Using !-string and #-string commands

6.1.1 !-strings are for setting simulation parameters

!-strings are a convenient way of accessing multiple layers of a nested dictionary structure with a single string using the format:

```
"!<ALIAS>.<sub-dict>...<sub-dict>.<param>"
```

Any level of the nested dictionary can be reached by truncating the keyword.

Note: !-strings only work on ``UserCommands`` objects

Below is an example of how to use !-strings, using the example optical train.

```
[ ]: import scopesim as sim
     opt = sim.load_example_optical_train()
```

```
[ ]: opt.cmds["!ATMO"]
```

```
[ ]: opt.cmds["!ATMO.background"]
```

```
[ ]: opt.cmds["!ATMO.background.filter_name"]
```

6.1.2 #-strings are for accessing Effect object parameters

Similar to !-strings, #-strings allow us to get at the preset values inside the Effect-objects of the optical system. #-strings allow us to pring the contents of an effect's meta dictionary.

Note: !-strings only work on ``OpticalTrain`` objects

Here, we're again using the example optical train defined above. First let's list the effects:

```
[ ]: opt.effects
```

We list the meta dictionary contents by using the string format

```
"#<effect-name>."
```

Note: The ``.`` at the end is important, otherwise the optical train will look for a non-existant effect named ``#<effect-name>``

```
[ ]: opt["#exposure_action."]
```

We print a specific meta parameter by adding it after the .

```
[ ]: opt["#exposure_action.ndit"]
```

Notice that the value of this dictionary entry is itself a !-string. We can resolve this by adding a ! to the end of the string, to force it to get the actual value from `opt.cmds`:

```
[ ]: opt["#exposure_action.ndit!"]
```

6.2 Turning Effect objects on or off

To list all the effects in an optical train, we do use the `effects` attribute.

Alternatively, we can call `opt.optics_manager.all_effects()`

```
[ ]: import scopesim as sim

opt = sim.load_example_optical_train()
opt.effects
```

Turning an effect on or off is as simple as using setting the `.include` attribute to `true` or `False`:

```
[ ]: opt["slit_wheel"].include = True
opt["slit_wheel"].include = False
```

```
[ ]:
```

6.3 Downloading packages

Note: Instrument packages are kept in a separate repository: the [Instrument Reference Database \(IRDB\)](#)

Before simulating anything we need to get the relevant instrument packages. Packages are split into the following categories

- Locations (e.g. Armazones, LaPalma)
- Telescopes (e.g. ELT, GTC)
- Instruments (e.g. MICADO, METIS, MORFEO, OSIRIS, MAAT)

We need to make sure we have all the packages required to build the optical system. E.g. observing with MICADO is useless without including the ELT.

```
[ ]: import scopesim as sim

from tempfile import TemporaryDirectory
tmpdir = TemporaryDirectory()
sim.rc.__config__["!SIM.file.local_packages_path"] = tmpdir.name
```

6.3.1 scopesim.download_packages()

Stable packages

The simplest way is to simply get the latest stable versions of the packages by calling their names.

Call `list_packages()` or see the [IRDB](#) for names.

```
[ ]: sim.list_packages()
```

```
[ ]: sim.download_packages(["Armazones", "ELT", "MICADO"])
```

Development version

Use the `release="latest"` parameter to get the latest stable development versions of the packages

```
[ ]: sim.download_packages("test_package", release="latest")
```

Bleeding-edge versions from GitHub

We can also pull the latest versions of a package from GitHub. This is useful if you are the one writing the package and want to test how it works on a different machine.

The following strings will work:

- Latest from a branch: `release="github:<branch-name>"`
- From a specific tag: `release="github:<tag>"`
- From a specific commit: `release="github:<super-long-commit-hash>"`

```
[ ]: sim.download_packages("test_package", release="github:dev_master")
```

```
[ ]: sim.download_packages("LFOA", release="github:3c136cd59ceeca551c01c6fa79f87377997f33f9")
```

6.4 Science target templates

The companion python package [ScopeSim-Templates](#) contains a library of helper functions for generating ScopeSim-friendly Source objects for various common astronomical sources.

For more information, please see the [ScopeSim-Templates documentation](#)

```
[ ]: import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

import scopesim_templates as sim_tp
```

6.4.1 A basic star cluster

```
[ ]: my_cluster = sim_tp.stellar.clusters.cluster(mass=1000.0,      # [Msun]
                                                distance=8000,     # [pc]
                                                core_radius=1)      # [pc]

my_cluster.plot()
```

6.4.2 A basic elliptical galaxy

```
[ ]: # See the docstring of `elliptical` for more keywords
my_elliptical = sim_tp.extragalactic.galaxies.elliptical(half_light_radius=30,  #_
↳[arcsec]
                                                    pixel_scale=0.1,      #_
↳[arcsec]
                                                    filter_name="Ks",
                                                    amplitude=10,
                                                    normalization="total",  #_
↳[Ks=10 for integrated flux]
                                                    n=4,                  #_
↳Sersic index
                                                    ellipticity=0.5,
                                                    angle=30)

plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.imshow(my_elliptical.fields[0].data, norm=LogNorm(), extent=[-25.6, 25.6, -25.6, 25.
↳6])
plt.xlabel("[arcsec]")
plt.subplot(122)
wave = np.arange(5000, 25000) # [angstrom]
plt.plot(wave, my_elliptical.spectra[0](wave))
plt.xlabel("Wavelength [Angstrom]")
```

6.5 Global rc simulation parameters

Default global simulation parameters used as a base layer for all instruments.

Also accessible via an empty `scopesim.UserCommands()` or via an empty `scopesim.OpticalTrain()` object

```
[ ]: import scopesim

scopesim.rc.__currsys__["!SIM.random.seed"] = 9001
scopesim.rc.__currsys__["!SIM.file.local_packages_path"] = "./"

scopesim.rc.__currsys__["!SIM"]
```

6.6 Source : from FITS images

We can use a FITS image as the Source object for a ScopeSim Simulation

Warning: The simulation output is only as good as the input

If the pixel scale of the input (CDELTn) is bigger than the pixel scale of the instrument, ScopeSim will simply interpolate the image.

Please don't expect wonders if the input image WCS information is not appropriate for the instrument you are using.

ScopeSim Source objects can be generated from fits.ImageHDU object in the following ways:

1. Just an Image and scaling flux value
2. An Image and the associated synphot Spectrum
3. An Image and arrays for wavelength and flux
 - `image_hdu=<fits.ImageHDU> + flux=<astropy.Quantity>`
 - `image_hdu=<fits.ImageHDU> + spectra=<list of synphot.SourceSpectrum>`
 - `image_hdu=<fits.ImageHDU> + lam=<array> + spectra=<list of arrays>`

```
[ ]: import numpy as np
import scipy
import astropy.io.fits as fits
from astropy import units as u
import matplotlib.pyplot as plt
%matplotlib inline

import scopesim

# Make an ImageHDU with some pixel data
hdu = fits.ImageHDU(data=scipy.misc.face(gray=True))

# Give the header some proper WCS info
hdu.header.update({"CDELT1": 1, "CUNIT1": "arcsec", "CRPIX1": 0, "CRVAL1": 0,
                  "CDELT2": 1, "CUNIT2": "arcsec", "CRPIX2": 0, "CRVAL2": 0,})

# plotting function for later
def plot(src):
    plt.figure(figsize=(12, 5))
    plt.subplot(121)
    wave = range(3000, 25000)
    plt.plot(wave, image_source.spectra[0](wave))
    plt.xlabel("Wavelength [Angstrom]")
    plt.ylabel("Flux [ph/s/cm2/Angstrom]")
    plt.subplot(122)
    plt.imshow(image_source.fields[0].data)
```

6.6.1 1. Just an Image and scaling flux value

```
image_hdu=<fits.ImageHDU> + flux=<astropy.Quantity>
```

It is assumed that the flux defined here is **integrated** flux and is the total flux contained in the image.

Note: In future version, header keywords like ``BUNIT`` etc will also be accepted. This functionality is not yet implemented though (April 2022).

```
[ ]: image_source = scopesim.Source(image_hdu=hdu, flux=10*u.ABmag)

plot(image_source)
```

6.6.2 2. An Image and the associated synphot Spectrum

```
image_hdu=<fits.ImageHDU> + spectra=<list of synphot.SourceSpectrum>
```

In this case, the image pixel values are seen as multipliers for the spectrum at a given coordinate.

Note: It is the users responsibility to make sure the total flux of the “cube” (image * spectrum) is scaled appropriately.

```
[ ]: # Alternatively, see the SpeXtra and Pyckles libraries for more spectra
vega_spec = scopesim.source.source_templates.vega_spectrum(mag=20)

image_source = scopesim.Source(image_hdu=hdu, spectra=[vega_spec])

plot(image_source)
```

6.6.3 3. An Image and arrays for wavelength and flux

```
image_hdu=<fits.ImageHDU> + lam=<array> + spectra=<array of arrays>
```

```
[ ]: n = 100
wavelengths = np.geomspace(0.3, 2.5, n) * u.um
flux = np.ones(n)

image_source = scopesim.Source(image_hdu=hdu, lam=wavelengths, spectra=np.array([flux]))

plot(image_source)
```

6.7 Source : Point sources from arrays

Collections of point sources can be initialised through either a collection of arrays, or an astropy Table

```
[ ]: import numpy as np
import astropy.table as table
from astropy import units as u

import scopesim
```

(continues on next page)

(continued from previous page)

```

# how many stars
n = 200

# random coordinated in a 100 arcsec box
x, y = 100 * np.random.random(size=(2, n)) - 50

# All stars reference the Vega spectrum
ref = np.zeros(n)
# Each star is given a different weighting, i.e. magnitude
weight = 10**(-0.4*np.linspace(10, 20, n))

# Note: The Pyckles and SpeXtra libraries contain many more stellar and galactic spectra
vega = scopesim.source.source_templates.vega_spectrum(mag=20)

```

6.7.1 astropy.Table

```

[ ]: tbl = table.Table(names=["x", "y", "ref", "weight"],
                        data= [x,    y,    ref,    weight],
                        units=[u.arcsec, u.arcsec, None, None])

table_source = scopesim.Source(table=tbl, spectra=[vega])

table_source.plot()

```

6.7.2 From loose arrays

```

[ ]: point_source = scopesim.Source(spectra=[vega], x=x, y=y, ref=ref, weight=weight)

point_source.plot()

```


7.1 Binding ScopeSim to a local copy of the IRDB

There may be cases where we would like to have a local copy of all the files and instruments available in the Instrument Reference Database (IRDB). In these cases it is helpful to bind our local version of ScopeSim to the IRDB. This is quite easy and involves only two steps:

1. *Make a local copy of the IRDB*
2. *Tell ScopeSim where to find the IRDB*
 - *By using runtime ScopeSim commands*
 - *By editing the ScopeSim config file*

7.1.1 Make a local copy of the IRDB

First we must clone the current version of the IRDB from GitHub:

```
$ git clone https://github.com/AstarVienna/irdb.git
```

Note: By default this will clone the master branch of the IRDB.

To clone a specific branch, either checkout the branch you want from origin after cloning master or specify the branch at clone time with:

```
$ git clone -b dev_master https://github.com/AstarVienna/irdb.git
```

7.1.2 Tell ScopeSim where to find the IRDB

By using runtime ScopeSim commands

At the beginning of any script or jupyter notebook we can run the following command to set the default location of the instrument packages: This is by far the simplest method, but it requires this lone of code in every script:

```
import scopesim
scopesim.rc.__config__["!SIM.file.local_packages_path"] = "<path/to/IRDB>"
```

By editing the ScopeSim config file

Find the `defaults.yaml` file in the top-level of the ScopeSim install directory. This should be here:

```
<path-to-python>/site-packages/scopesim/defaults.yaml
```

Note: Depending on where ScopeSim is installed, we may need administrator permissions to edit this file.

On line 36 (or thereabouts), we must replace the default path string (`/inst_pkgs/`) with the path to the top level of our cloned IRDB directory (wherever we chose to put it):

```
file :  
  local_packages_path : "./inst_pkgs/" # --> change this line to point to the top level  
↳ of the local IRDB folder
```

This path can be relative or absolute. If a relative path is used, ScopeSim will look for instrument packages at this location relative to the directory where ScopeSim was executed (e.g. where the notebook session was started).

EFFECT DESCRIPTIONS

8.1 apertures

8.1.1 ApertureList

A list of apertures, useful for IFU or MOS instruments.

Parameters

Examples

File format

Much like an ApertureMask, an ApertureList can be initialised by either of the three standard DataContainer methods. The easiest is however to make an ASCII file with the following columns:

id	left	right	top	bottom	angle	conserve_image	shape
	arcsec	arcsec	arcsec	arcsec	deg		
int	float	float	float	float	float	bool	str/int

Acceptable shape entries are: round, rect, hex, oct, or an integer describing the number of corners the polygon should have.

A polygonal mask is generated for a given shape and will be scaled to fit inside the edges of each aperture list row. The corners of each aperture defined by shape are found by finding equidistant positions around an ellipse constrained by the edges (left, ..., etc). An additional optional column offset may be added. This column describes the offset from 0 deg to the angle where the first corner is set.

Additionally, the filename of an ApertureMask polygon file can be given. The geometry of the polygon defined in the file will be scaled to fit inside the edges of the row.

Note: shape values "rect" and 4 do not produce equal results

Both use 4 equidistant points around an ellipse constrained by [left, ..., etc]. However "rect" aims to fill the constraining area, while 4 simply uses 4 points on the ellipse. Consequently, 4 results in a diamond shaped mask covering only half of the constraining area filled by "rect".

8.1.2 ApertureMask

Only provides the on-sky window coords of the Aperture.

- **Case: Imaging**
 - Covers the whole FOV of the detector
 - Round (with mask), square (without mask)
- **Case**
[LS Spec]
 - Covers the slit FOV
 - Polygonal (with mask), square (without mask)
- **Case**
[IFU Spec]
 - Covers the on-sky FOV of one slice of the IFU
 - Square (without mask)
- **Case**
[MOS Spec]
 - Covers a single MOS fibre FOV
 - Round, Polygonal (with mask), square (without mask)

The geometry of an `ApertureMask` can be initialised with the standard `DataContainer` methods (see Parameters below). Regardless of which method is used, the following columns must be present:

```
x      y
arcsec arcsec
float  float
```

Certain keywords need to also be included in the ascii header:

```
# id: <int>
# conserve_image: <bool>
# x_unit: <str>
# y_unit: <str>
```

If `conserve_image` is `False`, the flux from all sources in the aperture is summed and distributed uniformly over the aperture area.

Parameters

filename

[str] Path to ASCII file containing the columns listed above

table

[astropy.Table] An astropy Table containing the columns listed above

array_dict

[dict] A dictionary containing the columns listed above: {x: [...], y: [...], id: <int>, conserve_image: <bool>}

Other Parameters

pixel_scale

[float] [arcsec] Defaults to "`!INST.pixel_scale`" from the config

id

[int] An integer to identify the `ApertureMask` in a list of apertures

8.1.3 RectangularApertureMask

8.1.4 SlitWheel

Selection of predefined spectroscopic slits and possibly other field masks.

It should contain an open position. A user can define a non-standard slit by directly using the `Aperture` effect.

Parameters

`slit_names` : list of str

filename_format

[str] A f-string for the path to the slit files

current_slit

[str] Default name

Examples

This Effect assumes a folder full of ASCII files containing the edges of each slit. Each file should be names the same except for the slit's name or identifier.

This example assumes a folder `masks` containing the slit ASCII files with the naming convention: `slit_A.dat`, `slit_B.dat`, etc.

```
name: slit_wheel
class: SlitWheel
kwargs:
  slit_names:
    - A
    - B
    - C
  filename_format: "masks/slit_{}.dat"
  current_slit: "C"
```

8.2 detector_list

8.2.1 DetectorList

A description of detector positions and properties.

The list of detectors must have the following table columns

id	x_cen	y_cen	x_size	y_size	pixel_size	angle	gain
----	-------	-------	--------	--------	------------	-------	------

where:

- “id” is a reference id for the chip (fits header EXTNAME)
- “x_cen” and “y_cen” [mm] are the physical coordinates of centre of the chip on the detector plane
- “x_size”, “y_size” [mm, pixel] are the width/height of the chip
- “pixel_size” [mm] is the physical size of pixels in the detector
- “angle” [deg] is the rotation of the detector relative to the x-axis
- “gain” [e-/ADU] is the conversion factor for electrons (photons) to ADUs

The units for each column (except id) must be given in the meta data using the format <colname>_unit. E.g. x_size_unit. See examples below.

Note: Currently only the units specified below are accepted.

For x(y)_size_unit, acceptable units are mm, pixel

Parameters

filename

[str, optional] Filename of the ASCII file with the detector description. See examples

array_dict

[dict] Dict containing the detector description. See examples

image_plane_id

[int] Which image plane the detector will look at (generally 0)

Examples

With the array_dict feature

```
- name: single_detector
  class: DetectorList
  kwargs:
    image_plane_id : 0
    array_dict:
      id: [1]
      x_cen: [0.]
      y_cen: [0.]
      x_size: [5.12]
```

(continues on next page)

(continued from previous page)

```

        y_size: [5.12]
        pixel_size: [0.01]
        angle: [0.]
        gain: [1.0]
    x_cen_unit: mm
    y_cen_unit: mm
    x_size_unit: mm
    y_size_unit: mm
    pixel_size_unit: mm
    angle_unit: deg
    gain_unit: electron/adu

```

Or referring to a table contained in a separate ASCII file

```

- name : full_detector_array
  class : DetectorList
  kwargs :
    filename : "detecotr_list.dat"
    active_detectors : [1, 3]
    image_plane_id : 0

```

where the file detector_list.dat contains the following information

```

# x_cen_unit : mm
# y_cen_unit : mm
# x_size_unit : pix
# y_size_unit : pix
# pixel_size_unit : mm
# angle_unit : deg
# gain_unit : electron/adu
#
id   x_cen   y_cen   x_size   y_size   pixel_size   angle   gain
1   -63.94   0.00   4096   4096       0.015     0.0     1.0
2    0.00   0.00   4096   4096       0.015    90.0     1.0
3    63.94   0.00   4096   4096       0.015   180.0     1.0

```

8.2.2 DetectorWindow

For when a full DetectorList is too cumbersome.

Parameters

pixel_size

[float] [mm pixel-1] Physical pixel size

x, y

[float] [mm] Position of window centre relative to optical axis

width, height=None

[float] [mm] Dimensions of window. If height is None, height=width

angle

[float, optional] [deg] Rotation of window

gain

[float, optional] [ADU/e-]

units

[str, optional] [mm, pixel] Default “mm”. Sets the input parameter units. If “pixel”, (x, y, width, height) are multiplied by pixel_size

8.3 effects

8.3.1 Effect

Base class for representing the effects (artifacts) in an optical system.

The Effect class is conceived to independently apply the changes that an optical component (or series thereof) has on an incoming 3D description of an on-sky object. In other words, **an Effect object should receive a derivative of a ``Source`` object, alter it somehow, and return it.**

The interface for the Effect base-class has been kept very general so that it can easily be sub-classed as data for new effects becomes available. Essentially, a sub-classed Effects object must only contain the following attributes:

- `self.meta` - a dictionary to contain meta data.
- `self.apply_to(obj, **kwargs)` - a method which accepts a Source-derivative and returns an instance of the same class as obj
- `self.fov_grid(which="", **kwargs)`

Parameters

See `DataContainer` for input parameters

8.4 electronic

8.4.1 AutoExposure

Determine DIT and NDIT automatically from ImagePlane.

DIT is determined such that the maximum value in the incident photon flux (including astronomical source, sky and thermal backgrounds) fills the full well of the detector (`!DET.full_well`) to a given fraction (`!OBS.autoexposure.fill_frac`). NDIT is determined such that `DIT * NDIT` results in the requested exposure time.

The requested exposure time is taken from `!OBS.exptime`.

The effects sets the parameters `!OBS.dit` and `!OBS.ndit`.

Examples

The parameters *!OBS.exptime*, *!DET.full_well* and *!DET.mindit* should be defined as properties in the respective sub-sections.

```
name: auto_exposure
description: automatic determination of DIT and NDIT
class: AutoExposure
include: True
kwargs:
    fill_frac: "!OBS.auto_exposure.fill_frac"
```

8.4.2 BasicReadoutNoise

Readout noise computed as: $\text{ron} * \sqrt{\text{NDIT}}$.

8.4.3 Bias

Adds a constant bias level to readout.

8.4.4 BinnedImage

8.4.5 DarkCurrent

required: dit, ndit, value

8.4.6 DetectorModePropertiesSetter

Set mode specific curr_sys properties for different detector readout modes.

A little class (*DetectorModePropertiesSetter*) that allows different "*!DET*" properties to be set on the fly.

Parameters

mode_properties

[dict] A dictionary containing the DET parameters to be changed for each mode. See below for an example yaml entry.

Examples

Add the values for the different detector readout modes to all the relevant detector yaml files. In this case the METIS HAWAII (L, M band) and GeoSnap (N band) detectors: *METIS_DET_IMG_LM.yaml* , *METIS_DET_IMG_N.yaml*

```
- name: lm_detector_readout_parameters
  class: DetectorModePropertiesSetter
  kwargs:
    mode_properties:
      fast:
```

(continues on next page)

(continued from previous page)

```
mindit: 0.04
full_well: !!float 1e5
ron: 70
slow:
mindit: 1.3
full_well: !!float 1e5
ron: 14
```

Add the OBS dict entry !OBS.detector_readout_mode to the properties section of the mode_yamls descriptions in the default.yaml files.

```
mode_yamls:
- object: observation
  alias: OBS
  name: lss_1
  yamls:
    ...
  properties:
    ...
    detector_readout_mode: slow
```

8.4.7 LinearityCurve

Detector linearity effect.

The detector linearity curve is set in terms of *incident* flux (e/s) and *measured* detector values (ADU).

Examples

The effect can be instantiated in various ways.:

```
- name: detector_linearity
  class: LinearityCurve
  kwargs:
    filename: FPA_linearity.dat

- name: detector_linearity
  class: LinearityCurve
  kwargs:
    array_dict: {incident: [0, 77000, 99999999999],
                    measured: [0, 77000, 77000]}

- name: detector_linearity
  class: LinearityCurve
  kwargs:
    incident: [0, 77000, 99999999]
    measured: [0, 77000, 77000]
```

8.4.8 PoorMansHxRGReadoutNoise

8.4.9 Quantization

Converts raw data to whole photons.

8.4.10 ReferencePixelBorder

8.4.11 ShotNoise

8.4.12 SummedExposure

Simulates a summed stack of `ndit` exposures.

8.4.13 UnequalBinnedImage

8.5 fits_headers

8.5.1 EffectsMetaKeywords

Adds meta dictionary info from all Effects to the FITS headers.

Parameters

ext_number

[int, list of ints, optional] Default 0. The numbers of the extensions to which the header keywords should be added

add_excluded_effects

[bool, optional] Default False. Add meta dict for effects with `<effect>.include=False`

keyword_prefix

[str, optional] Default "HIERARCH SIM". Custom FITS header keyword prefix. Effect meta dict entries will appear in the header as: `<keyword_prefix> EFFn <key> : <value>`

Examples

Yaml file entry:

```
name: effect_dumper
class: EffectsMetaKeywords
description: adds all effects meta dict entries to the FITS header
kwargs:
  ext_number: [0, 1]
  add_excluded_effects: False
  keyword_prefix: HIERARCH SIM
```

8.5.2 ExtraFitsKeywords

Extra FITS header keywords to be added to the pipeline FITS files.

These keywords are ONLY for keywords that should be MANUALLY ADDED to the headers after a simulation is read-out by the detector.

Simulation parameters (Effect kwargs values, etc) will be added automatically by ScopeSim in a different function, but following this format.

The dictionaries should be split into different HIERARCH lists, e.g.:

- HIERARCH ESO For ESO specific keywords
- HIERARCH SIM For ScopeSim specific keywords, like simulation parameters
- HIERARCH MIC For MICADO specific keywords, (unsure what these would be yet)

More HIERARCH style keywords can also be added as needed for other use-cases.

Parameters

filename

[str, optional] Name of a .yaml nested dictionary file. See below for examples

yaml_string

[str, optional] A triple-” string containing the contents of a yaml file

header_dict

[nested dicts, optional] A series of nested python dictionaries following the format of the examples below. This keyword allows these dicts to be defined directly in the Effect yaml file, rather than in a separate header keywords file.

Examples

Specifying the extra FITS keywords directly in the .yaml file where the Effect objects are described.

```
name: extra_fits_header_entries
class: ExtraFitsKeywords
kwargs:
  header_dict:
    - ext_type: PrimaryHDU
      keywords:
        HIERARCH:
          ESO:
            ATM:
              TEMPERAT: -5
```

The contents of header_dict can also be abstracted away into a separate file, e.g. extra_FITS_keys.yaml. The file format is described below in detail below.

```
name: extra_fits_header_entries
class: ExtraFitsKeywords
kwargs:
  filename: extra_FITS_keys.yaml
```

The Effect can be added directly in an iPython session.

```
>>> hdr_dic = {"ext_type": "PrimaryHDU",
               "keywords":
                 {"HIERARCH":
                  {"SIM":
                   {"hello": world}
                  }
                 }
               }
>>> extra_keys = ExtraFitsKeywords(header_dict=hdr_dic)
>>> optical_train.optics_manager.add_effect(extra_keys)
```

Yaml file format

This document is a yaml document. Hence all new keywords should be specified in the form of yaml nested dictionaries. As each `astropy.HDUList` contains one or more extensions, the initial level is reserved for a list of keyword groups. For example:

```
- ext_type: PrimaryHDU
  keywords:
    HIERARCH:
      ESO:
        ATM:
          TEMPERAT: -5

- ext_number: [1, 2]
  keywords:
    HIERARCH:
      ESO:
        DET:
          DIT: [5, '[s] exposure length'] # example of adding a comment
          EXTNAME: "DET$.DATA"           # example of extension specific qualifier
```

The keywords can be added to one or more extensions, based on one of the following `ext_` qualifiers: `ext_name`, `ext_number`, `ext_type`

Each of these `ext_` qualifiers can be a `str` or a `list`. For a list, ScopeSim will add the keywords to all extensions matching the specified type/name/number

The number of the extension can be used in a value by using the “\$” string. That is, keyword values with “\$” will have the extension number inserted where the “\$” is.

The above example (`EXTNAME: "DET$.DATA"`) will result in the following keyword added only to extensions 1 and 2:

- PrimaryHDU (ext 0):

```
header['HIERARCH ESO ATM TEMPERAT'] = -5
```

- Extension 1 (regardless of type):

```
header['HIERARCH ESO DET DIT'] = (5, '[s] exposure length')
header['EXTNAME'] = "DET1.DATA"
```

- Extension 2 (regardless of type):

```
header['HIERARCH ESO DET DIT'] = (5, '[s] exposure length')
header['EXTNAME'] = "DET2.DATA"
```

Resolved and un-resolved keywords

ScopeSim uses bang-strings to resolve global parameters. E.g: `from_currsys('!ATMO.temperature')` will resolve to a float. These bang-strings will be resolved automatically in the `keywords` dictionary section.

If the keywords bang-string should instead remain unresolved and the string added verbatim to the header, we use the `unresolved_keywords` dictionary section.

Additionally, new functionality will be added to ScopeSim to resolve the `kwargs/meta` parameters of Effect objects. The format for this will be to use a new type: the hash-string. This will have this format:

```
#<optical_element_name>.<effect_name>.<kwarg_name>
```

For example, the temperature of the MICADO detector array can be accessed by:

```
'#MICADO_DET.full_detector_array.temperature'
```

In the context of the `yaml` file this would look like:

```
- ext_type: PrimaryHDU
  keywords:
    HIERARCH:
      ESO:
        DET
          TEMPERAT: '#MICADO_DET.full_detector_array.temperature'
```

Obviously some thought needs to be put into how exactly we list the simulation parameters in a coherent manner. But this is ‘Zukunftsmusik’. For now we really just want an interface that can add the ESO header keywords, which can also be expanded in the future for our own purposes.

Below is an example of some extra keywords for MICADO headers:

```
- ext_type: PrimaryHDU
  keywords:
    HIERARCH:
      ESO:
        ATM:
          TEMPERAT: '!ATMO.temperature'  # will be resolved via from_currsys
          PWV: '!ATMO.pwv'
          SEEING: 1.2
        DAR:
          VALUE: '#<effect_name>.<meta_name>'  # will be resolved via effects
        DPR:
          TYPE: 'some_type'
      SIM:
        random_simulation_keyword: some_value
      MIC:
        micado_specific: ['keyword', 'keyword comment']

  unresolved_keywords:
    HIERARCH:
```

(continues on next page)

(continued from previous page)

```

    ESO:
      ATM:
        TEMPERAT: '!ATMO.temperature'  # will be left as a string
- ext_type: ImageHDU
  keywords:
    HIERARCH:
      SIM:
        hello: world
        hallo: welt
        grias_di: woed
        zdrasviute: mir
        salud: el mundo

```

8.5.3 SimulationConfigFitsKeywords

Adds parameters from all config dictionaries to the FITS headers.

Parameters

ext_number

[int, list of ints, optional] Default 0. The numbers of the extensions to which the header keywords should be added

resolve

[bool] Default True. If True, all !-strings and #-strings are resolved via `from_currsys` before being added to the header. If False, the unaltered !-strings or #-strings are added to the header.

keyword_prefix

[str, optional] Default "HIERARCH SIM". Custom FITS header keyword prefix. Effect meta dict entries will appear in the header as: `<keyword_prefix> SRCn <key> : <value>`

Examples

Yaml file entry:

```

name: source_descriptor
class: SimulationConfigFitsKeywords
description: adds info from all config dicts to the FITS header
kwargs:
  ext_number: [0]
  resolve: False
  keyword_prefix: HIERARCH SIM

```

8.5.4 SourceDescriptionFitsKeywords

Adds parameters from all Source fields to the FITS headers.

Parameters

ext_number

[int, list of ints, optional] Default 0. The numbers of the extensions to which the header keywords should be added

keyword_prefix

[str, optional] Default “HIERARCH SIM”. Custom FITS header keyword prefix. Effect meta dict entries will appear in the header as: <keyword_prefix> SRCn <key> : <value>

Examples

Yaml file entry:

```
name: source_descriptor
class: SourceDescriptionFitsKeywords
description: adds info from all Source fields to the FITS header
kwargs:
  ext_number: [0]
  keyword_prefix: HIERARCH SIM
```

8.6 metis_lms_trace_list

8.6.1 MetisLMSEfficiency

Computes the grating efficiency (blaze function) for the METIS LMS.

The procedure is described in E-REP-ATC-MET-1016_1.0. For a given order (determined by the central wavelength) the grating efficiency is modelled as a squared sinc function of wavelength via the grating angle.

8.6.2 MetisLMSImageSlicer

Treats the METIS LMS image slicer as an aperture mask effect.

This helps in building a `FieldOfView` object that combines the spatial field of the slicer with the spectral range covered by the LMS setting.

The effect differs from its parent class `ApertureMask` in the initialisation from the *Aperture List* extension of the trace file `!OBS.trace_file`.

8.6.3 MetisLMSSpectralTraceList

SpectralTraceList for the METIS LM spectrograph.

8.7 obs_strategies

8.7.1 ChopNodCombiner

Creates and combines 4 images for each of the chop/nod positions.

- AA : original position (dx, dy) = (0, 0)
- AB : chop position (dx, dy) = chop_offsets
- BA : nod position (dx, dy) = nod_offsets
- BB : chop-nod position (dx, dy) = nod_offsets + chop_offsets

Images are combined using:

```
im_combined = (AA - AB) - (BA - BB)
```

If no nod_offset is given, it is set to the inverse of chop_offset.

ChopNodCombiner is a detector effect and should be placed last in the detector yaml (after the noise effects).

Keyword Arguments

chop_offsets

[tuple, optional] [arcsec] (dx, dy) offset of chop position relative to AA

nod_offsets

[tuple, optional] [arcsec] (dx, dy) offset of nod position relative to AA

Example yaml entry

```
name: perpendicular_chop_nod_slanted_pattern
description: chop throw to (+5, 0) and nod throw to (-7, +10) arcsec
class: ChopNodCombiner
include: True
kwargs:
  pixel_scale : "!INST.pixel_scale"
  chop_offsets : (5, 0)
  nod_offsets : (-7, 10)
```

8.8 psfs

8.8.1 AnalyticalPSF

8.8.2 AnisocadoConstPSF

Makes a SCAO on-axis PSF with a desired Strehl ratio at a given wavelength.

To make the PSFs a map connecting Strehl, Wavelength, and residual wavefront error is required.

Parameters

filename

[str] Path to Strehl map with axes (x, y) = (wavelength, wavefront error).

strehl

[float] Desired Strehl ratio. Either percentage [1, 100] or fractional [1e-3, 1].

wavelength

[float] [um] The given strehl is valid for this wavelength.

psf_side_length

[int] [pixel] Default is 512. Side length of the kernel images.

offset

[tuple] [arcsec] SCAO guide star offset from centre (dx, dy).

rounded_edges

[bool] Default is True. Sets all halo values below a threshold to zero. The threshold is determined from the max values of the edge rows of the kernel image.

Other Parameters

convolve_mode

[str] ["same", "full"] convolution keywords from `scipy.signal.convolve`

Examples

Add an AnisocadoConstPSF with code:

```
from scopesim.effects import AnisocadoConstPSF
psf = AnisocadoConstPSF(filename="test_AnisoCADO_rms_map.fits",
                        strehl=0.5,
                        wavelength=2.15,
                        convolve_mode="same",
                        psf_side_length=512)
```

Add an AnisocadoConstPSF to a yaml file:

```
effects:
- name: Ks_Stehl_40_PSF
  description: A 40% Strehl PSF over the field of view
  class: AnisocadoConstPSF
```

(continues on next page)

(continued from previous page)

```
kwargs:
  filename: "test_AnisoCADO_rms_map.fits"
  strehl: 0.5
  wavelength: 2.15
  convolve_mode: full
  psf_side_length: 512
```

8.8.3 DiscretePSF

8.8.4 FieldConstantPSF

A PSF that is constant across the field.

For spectroscopy, a wavelength-dependent PSF cube is built, where for each wavelength the reference PSF is scaled proportional to wavelength.

8.8.5 FieldVaryingPSF

TBA.

Parameters

sub_pixel_flag : bool, optional flux_accuracy : float, optional

Default 1e-3. Level of flux conservation during rescaling of kernel

8.8.6 GaussianDiffractionPSF

8.8.7 NonCommonPathAberration

TBA.

Needed: pixel_scale Accepted: kernel_width, strehl_drift

8.8.8 PSF

8.8.9 SeeingPSF

Currently only returns gaussian kernel with a `fwhm` [arcsec].

Parameters

fwhm

[float] [arcsec]

8.8.10 SemiAnalyticalPSF

8.8.11 Vibration

Creates a wavelength independent kernel image.

8.9 rotation

8.9.1 Rotate90CCD

Rotates CCD by integer multiples of 90 degrees.

rotations kwarg is number of counter-clockwise rotations

Author: Dave jones

8.10 shifts

8.10.1 AtmosphericDispersion

Used to generate the wavelength bins based on shifts due to the atmosphere.

Doesn't contain an `apply_to` function, but provides information through the `fov_grid` function.

Required Parameters

airmass

[float] Recommended to use “!OBS.airmass” in the OBS properties

temperature

[float] [degC] Recommended to use “!ATMO.temperature” in the ATMO properties

humidity

[float] [0..1] Recommended to use “!ATMO.humidity” in the ATMO properties

pressure

[float] [bar] Recommended to use “!ATMO.pressure” in the ATMO properties

latitude

[float] [deg] Recommended to use “!ATMO.latitude” in the ATMO properties

altitude

[m] Recommended to use “!ATMO.altitude” in the ATMO properties

pixel_scale

[arcsec] Recommended to use “!INST.pixel_scale” in the INST properties

Optional Parameters

wave_min

[float] [um] Defaults to “!SIM.spectral.wave_min”

wave_mid

[float] [um] Defaults to “!SIM.spectral.wave_mid”

wave_max

[float] [um] Defaults to “!SIM.spectral.wave_max”

sub_pixel_fraction

[float] [0..1] Defaults to “!SIM.sub_pixel.fraction”

num_steps

[int] Default: 1000. Number of wavelength steps to use when interpolating the atmospheric dispersion curve

8.10.2 AtmosphericDispersionCorrection

Alters the position on the detector for a FOV object (WCS_prefix=”D”).

Only acts on FOVs during the main effects loop in OpticalTrain. For the sake of computational efficiency, the ADC can be instructed to counteract the atmospheric diffraction during the OpticalTrain setup phase, by passing the kwarg: quick_adc=True

Parameters

kwargs

8.10.3 Shift3D

8.11 shutter

8.11.1 Shutter

Simulate a closed shutter, useful for dark exposures.

8.12 spectral_efficiency

8.12.1 SpectralEfficiency

Applies the grating efficiency (blaze function) for a SpectralTraceList.

Input Data Format

The efficiency curves are taken from a fits file *filename* with a structure similar to the trace definition file (see *SpectralTraceList*). The required extensions are: - 0 : PrimaryHDU [header] - 1 : BinTableHDU or TableHDU[header, data] : Overview table of all traces - 2..N : BinTableHDU or TableHDU : Efficiency curves, one per trace. The

tables must have the two columns *wavelength* and *efficiency*

Note that there must be one extension for each trace defined in the *SpectralTraceList*. Extensions for other traces are ignored.

8.12.2 EXT 0 : PrimaryHDU

Required header keywords:

- ECAT : int : Extension number of overview table, normally 1
- EDATA : int : Extension number of first Trace table, normally 2

No data is required in this extension

8.12.3 EXT 1 : (Bin)TableHDU : Overview of traces

No special header keywords are required in this extension.

Required Table columns: - description : str : identifier for each trace - extension_id : int : which extension is each trace in

8.12.4 EXT 2 : (Bin)TableHDU : Efficiencies for individual traces

Required header keywords: - EXTNAME : must be identical to the *description* in EXT 1

Required Table columns: - wavelength : float : [um] - efficiency : float : number [0..1]

8.13 spectral_trace_list

8.13.1 SpectralTraceList

List of spectral trace geometries for the detector plane.

Should work in concert with an ApertureList (or ApertureMask) object and a DetectorList object

Spectral trace patterns are to be kept in a `fits.HDUList` with one or more `fits.BinTableHDU` extensions, each one describing the geometry of a single trace. The first extension should be a `BinTableHDU` connecting the traces to the correct `Aperture` and `ImagePlane` objects.

The `fits.HDUList` objects can be loaded using one of these two keywords:

- `filename`: for on disk FITS files, or
- `hdulist`: for in-memory `fits.HDUList` objects

The format and contents of the extensions in the `HDUList` (FITS file) object is listed below

Input Data Format

A trace list FITS file needs the following extensions:

- 0 : PrimaryHDU [header]
- 1 : BinTableHDU [header, data] : Overview table of all traces
- 2..N : BinTableHDU [header, data] : Trace tables. One per spectral trace

8.13.2 EXT 0 : PrimaryHDU

Required Header Keywords:

- ECAT : int : Extension number of overview table. Normally 1
- EDATA : int : Extension number of first Trace table. Normally 2

No data is required in this extension

8.13.3 EXT 1 : BinTableHDU : Overview of traces

No special header keywords are required in this extension

Required Table columns:

- description : str : identifier of each trace
- extension_id : int : which extension is each trace in
- aperture_id : int : which aperture matches this trace (e.g. MOS / IFU)
- image_plane_id : int : on which image plane is this trace projected

8.13.4 EXT 2 : BinTableHDU : Individual traces

Required header keywords: - EXTNAME : must be identical to the *description* in EXT 1

Recommended header keywords: - DISPDIR : “x” or “y” : dispersion axis. If not present, Scopesim tries to determine this automatically; this may be unreliable in some cases.

Required Table columns: - wavelength : float : [um] : wavelength of monochromatic aperture image - s : float : [arcsec] : position along aperture perpendicular to trace - x : float : [mm] : x position of aperture image on focal plane - y : float : [mm] : y position of aperture image on focal plane

8.13.5 SpectralTraceListWheel

A Wheel-Effect object for selecting between multiple gratings/grisms.

See `SpectralTraceList` for the trace file format description.

Parameters

trace_list_names

[list] The list of unique identifiers in the trace filenames

filename_format

[str] f-string that directs scopesim to the folder containing the trace files. This can be a !-string if the trace names are shared with other *Wheel effect objects (e.g. a FilterWheel). See examples.

current_trace_list

[str] default trace file to use

kwargs

[key-value pairs] Addition keywords that are passed to the SpectralTraceList objects See SpectralTraceList docstring

Examples

A simplified YAML file example taken from the OSIRIS instrument package:

```
alias: INST
name: OSIRIS_LSS

properties:
  decouple_detector_from_sky_headers: True
  grism_names:
    - R300B
    - R500B
    - R1000B
    - R2500V

effects:
  - name: spectral_trace_wheel
    description: grism wheel containing spectral trace geometries
    class: SpectralTraceListWheel
    kwargs:
      current_trace_list: "!OBS.grating_name"
      filename_format: "traces/LSS_{}_TRACE.fits"
      trace_list_names: "!INST.grism_names"

  - name: grating_efficiency
    description: OSIRIS grating efficiency curves, piggybacking on FilterWheel
    class: FilterWheel
    kwargs:
      minimum_throughput: !!float 0.
      filename_format: "gratings/{}.txt"
      current_filter: "!OBS.grating_name"
      filter_names: "!INST.grism_names"
```

8.14 surface_list

8.14.1 SurfaceList

8.15 ter_curves

8.15.1 ADCWheel

Wheel holding a selection of predefined atmospheric dispersion correctors.

Example

```
name : adc_wheel
class: ADCWheel
kwargs:
    adc_names: []
    filename_format: "TER_ADC_{}.dat"
    current_adc: "const_90"
```

8.15.2 AtmosphericTERCurve

8.15.3 DownloadableFilterCurve

8.15.4 FilterCurve

Description TBA.

Parameters

position : int, optional filter_name : str, optional

Ks - corresponding to the filter name in the filename pattern

filename_format

[str, optional] TC_filter_{}.dat

Can either be created using the standard 3 options: - filename: direct filename of the filter curve - table: an astropy.Table - array_dict: a dictionary version of a table: {col_name1: values, }

or by passing the combination of filter_name and filename_format as kwargs. Here all filter file names follow a pattern (e.g. see above) and the {} are replaced by filter_name at run time. filter_name can also be a !bang string for a __currsys__ entry: "!INST.filter_name"

8.15.5 FilterWheel

Wheel holding a selection of predefined filters.

Examples

```
name: filter_wheel
class: FilterWheel
kwargs:
  filter_names: []
  filename_format: "filters/{}".
  current_filter: "Ks"
```

8.15.6 FilterWheelBase

Base class for Filter Wheels.

8.15.7 PupilTransmission

Wavelength-independent transmission curve.

Use this class to describe a cold stop or pupil mask that is characterised by “grey” transmissivity. The emissivity is set to zero, assuming that the mask is cold.

8.15.8 QuantumEfficiencyCurve

8.15.9 SkycalcTERCurve

Retrieve an atmospheric spectrum from ESO’s skycalc server.

kwarg parameters

skycalc parameters can be found by calling:

```
>>> import skycalc_ipy
>>> skycalc_ipy.SkyCalc().keys
```

Note: Different to skycalc_ipy, *wmin* and *wmax* must be given in units of μm

Examples

```
- name : skycalc_background
  class : SkycalcTERCurve
  kwargs :
    wunit : "!SIM.spectral.wave_unit"
    wmin : "!SIM.spectral.wave_min"
    wmax : "!SIM.spectral.wave_max"
    wdelta : 0.0001      # 0.1nm bin width
    outer : 1
    outer_unit : "m"
```

8.15.10 SpanishVOFilterCurve

Pulls a filter transmission curve down from the Spanish VO filter service.

Parameters

observatory : str instrument : str filter_name : str

Examples

```
name: HAWKI-Ks
class: SpanishVOFilterCurve
kwargs:
  observatory : Paranal
  instrument : HAWKI
  filter_name : Ks
```

8.15.11 SpanishVOFilterWheel

A FilterWheel that loads all the filters from the Spanish VO service.

Warning: This use `astropy.download_file(..., cache=True)`.

The filter transmission curves probably won't change, but if you notice discrepancies, try clearing the astropy cache:

```
>> from astropy.utils.data import clear_download_cache
>> clear_download_cache()
```

Parameters

observatory : str

instrument : str

current_filter

[str] Default filter name

include_str, exclude_str

[str] String sequences that can be used to include or exclude filter names which contain a certain string. E.g. GTC/OSIRIS has curves for `sdss_g` and `sdss_g_filter`. We can force the inclusion of only the filter curves by setting `list_include_str: "_filter"`.

Examples

```
name: svo_filter_wheel
class: SpanishVOFilterWheel
kwargs:
  observatory: "GTC"
  instrument: "OSIRIS"
  current_filter: "sdss_r_filter"
  include_str: "_filter"
```

8.15.12 TERCurve

Transmission, Emissivity, Reflection Curve.

note:: This is basically an Effect wrapper for the
SpectralSurface object

Must contain a wavelength column, and one or more of the following: `transmission`, `emissivity`, `reflection`. Additionally, in the header there should be the following keywords: `wavelength_unit`

kwargs that can be passed:

```
"rescale_emission" : { "filter_name": str, "value": float, "unit": str}
```

Examples

Directly inside a YAML file description:

```
name: bogus_surface
class: TERCurve
kwargs:
  array_dict:
    wavelength: [0.3, 3.0]
    transmission: [0.9, 0.9]
    emission: [1, 1]
  wavelength_unit: um
  emission_unit: ph s-1 m-2 um-1
  rescale_emission:
    filter_name: "Paranal/HAWK.Ks"
```

(continues on next page)

(continued from previous page)

```
value: 15.5
unit: ABmag
```

Indirectly inside a YAML file:

```
name: some_curve
class: TERCurve
kwargs:
  filename: bogus_surface.dat
```

which references this ASCII file:

```
# name: bogus_surface
# wavelength_unit: um
wavelength transmission emissivity
0.3          0.9         0.1
3.0          0.9         0.1
```

8.15.13 TopHatFilterCurve

A simple Top-Hat filter profile.

Parameters

transmission

[float] [0..1] Peak transmission of filter

blue_cutoff, red_cutoff

[float] [um] Blue and Red cutoff wavelengths

wing_transmission

[float, optional] [0..1] Default 0. Wing transmission of filter outside the cutoff range

Examples

```
name: J_band_tophat
class: TopHatFilterCurve
kwargs:
  transmission : 0.9
  wing_transmission : 0.001
  blue_cutoff : 1.15
  red_cutoff : 1.35
```

8.15.14 TopHatFilterWheel

A selection of top-hat filter curves as defined in the input lists.

Parameters

filter_names: list of string

transmissions: list of floats

[0..1] Peak transmissions inside the cutoff limits

wing_transmissions: list of floats

[0..1] Wing transmissions outside the cutoff limits

blue_cutoffs: list of floats

[um]

red_cutoffs: list of floats

[um]

current_filter: str, optional

Name of current filter at initialisation. If no name is given, the first entry in *filter_names* is used by default.

Examples

```
name: top_hat_filter_wheel
class: TopHatFilterWheel
kwargs:
  filter_names: ["J", "H", "K"]
  transmissions: [0.9, 0.95, 0.85]
  wing_transmissions: [0., 0., 0.001]
  blue_cutoffs: [1.15, 1.45, 1.9]
  red_cutoffs: [1.35, 1.8, 2.4]
  current_filter: "K"
```


SCOPESIM

9.1 scopesim package

9.1.1 Subpackages

scopesim.commands package

Submodules

scopesim.commands.user_commands module

class scopesim.commands.user_commands.UserCommands(**kwargs)

Bases: `object`

Contains all the setting that a user may wish to alter for an optical train

Most of the important settings are kept in the `.cmds` system dictionary Setting can be accessed by using the alias names. Currently these are:

- ATMO: atmospheric and observatory location settings
- TEL: telescope related settings
- RO: relay optics settings, i.e. between telescope and instrument
- INST: instrument optics settings
- DET: detector settings
- OBS: observation settings, and
- SIM: simulation settings

All of the settings are contained in a special `SystemDict` dictionary that allows the user to access all the settings via a bang-string (!). E.g:

```
cmds = UserCommands()  
cmds["!SIM.file.local_packages_path"]
```

Note: To use this format for accessing hierarchically-stored values, the bang string must always begin with a “!”

Alternatively the same value can be accessed via the normal dictionary format. E.g:

```
cmds["SIM"]["file"]["local_packages_path"]
```

Parameters

use_instrument

[str, optional] The name of the main instrument to use

packages

[list, optional] list of package names needed for the optical system, so that ScopeSim can find the relevant files. E.g. ["Armazones", "ELT", "MICADO"]

yamls

[list, optional] list of yaml filenames that are needed for the combined optical system E.g. ["MICADO_Standalone_RO.yaml", "MICADO_H4RG.yaml", "MICADO_.yaml"]

mode_yamls

[list of yamls, optional] list of yaml docs ("OBS" docs) that are applicable only to specific operational modes of the instrument. Further yaml files can be specified in the recursive doc entry: "yamls"

set_modes

[list of strings, optional] A list of default mode yamls to load. E.g. ["SCAO", "IMG_4mas"]

properties

[dict, optional] Any extra "OBS" properties that should be added

ignore_effects

[list] Not yet implemented

add_effects

[list] Not yet implemented

override_effect_values

[dict] Not yet implemented

Notes

Attention: We track your IP address when ScopeSim checks for updates

When initialising a UserCommands object via `use_instrument=`, ScopeSim checks on the database whether there are updates to the instrument package. Our server records the IP address of each query for out own statistics only.

WE DO NOT STORE OR TRACK PERSONAL DATA. THESE STATISTICS ARE NEEDED FOR GETTING MORE FUNDING TO CONTINUE DEVELOPING THIS PROJECT.

We are doing this solely as a way of showing the austrian funding agency that people are indeed using this software (or not). Your participation in this effort greatly helps our chances of securing the next grant.

However, if you would still like to avoid your IP address being stored, you can run `scopesim` 100% anonymously by setting:

```
>>> scopesim.rc.__config__["!SIM.reports.ip_tracking"] = True
```

at the beginning of each session. Alternatively you can also pass the same bang keyword when generating a UserCommand object:

```
>>> from scopesim import UserCommands
>>> UserCommands(use_instrument="MICADO",
```

```
... properties={"!SIM.reports.ip_tracking": False})
```

If you use a custom yaml configuration file, you can also add this keyword to the `properties` section of the yaml file.

Examples

Here we use a combination of the main parameters: `packages`, `yamls`, and `properties`. When not using the `use_instrument` key, `packages` and `yamls` must be specified, otherwise `scopesim` will not know where to look for yaml files (only relevant if reading in yaml files):

```
>>> from scopesim.server.database import download_package
>>> from scopesim.commands import UserCommands
>>>
>>> download_package("test_package")
>>> cmd = UserCommands(packages=["test_package"],
```

```
... yamls=["test_telescope.yaml", ... {"alias": "ATMO", ... "properties": {"pwv": 9001}}], ... proper-
ties={"!ATMO.pwv": 8999})
```

Attributes

`cmds`

[NestedMapping] Built from the `properties` dictionary of a yaml dictionary. All values here are accessible globally by all `Effects` objects in an `OpticalTrain` once the `UserCommands` has been passed to the `OpticalTrain`.

`yaml_dicts`

[list of dicts] Where all the effects dictionaries are stored

`list_modes()`

property modes

`set_modes(modes=None)`

`update(**kwargs)`

Update the current parameters with a yaml dictionary.

See the `UserCommands` main docstring for acceptable kwargs

Module contents

scopesim.detector package

Submodules

scopesim.detector.detector module

`class scopesim.detector.detector.Detector(header, cmds=None, **kwargs)`

Bases: `DetectorBase`

property `data`

```
extract_from(image_plane, spline_order=1, reset=True)

property hdu

property header

property image

reset()
```

scopesim.detector.detector_array module

Contains DetectorArray and aux functions.

```
class scopesim.detector.detector_array.DetectorArray(detector_list=None, cmds=None, **kwargs)
```

Bases: `object`

Manages the individual Detectors, mostly used for readout.

```
readout(image_planes, array_effects=None, dtr_effects=None, **kwargs) → HDUList
```

Read out the detector array into a FITS HDU List.

1. Select the relevant image plane to extract images from.
2. Apply detector array effects (apply to the entire image plane)
3. Make a series of Detectors for each row in a DetectorList object.
4. Iterate through all Detectors, extract image from *image_plane*.
5. Apply all effects (to all Detectors).
6. Add necessary header keywords (not implemented).
7. Generate a HDUList with the ImageHDUs and any extras:
 - add PrimaryHDU with meta data regarding observation in header
 - add ImageHDU objects
 - add ASCII TableHDU with Effects meta data in final table extension (not implemented)

Parameters

image_planes

[list of ImagePlane objects] The correct image plane is automatically chosen from the list

array_effects

[list of Effect objects] A list of effects related to the detector array

dtr_effects

[list of Effect objects] A list of effects related to the detectors

Returns

latest_exposure

[fits.HDUList] Output FITS HDU List.

```
scopesim.detector.detector_array.make_effects_hdu(effects)
```

```
scopesim.detector.detector_array.make_primary_hdu(meta)
```

Create the primary header from meta data.

scopesim.detector.ngxrg module

NGXRG by Bernard Rauscher see the paper: <http://arxiv.org/abs/1509.06264> downloaded from: <http://jwst.nasa.gov/publications.html>

Module contents

scopesim.effects package

Submodules

scopesim.effects.apertures module

Effects related to field masks, including spectroscopic slits.

class scopesim.effects.apertures.**ApertureList**(**kwargs)

Bases: *Effect*

A list of apertures, useful for IFU or MOS instruments.

Parameters

property apertures

apply_to(obj, **kwargs)

See parent docstring.

get_apertures(row_ids)

plot()

plot_masks()

class scopesim.effects.apertures.**ApertureMask**(**kwargs)

Bases: *Effect*

Only provides the on-sky window coords of the Aperture.

- **Case: Imaging**
 - Covers the whole FOV of the detector
 - Round (with mask), square (without mask)
- **Case**
[LS Spec]
 - Covers the slit FOV
 - Polygonal (with mask), square (without mask)
- **Case**
[IFU Spec]
 - Covers the on-sky FOV of one slice of the IFU
 - Square (without mask)

- **Case**
[MOS Spec]
 - Covers a single MOS fibre FOV
 - Round, Polygonal (with mask), square (without mask)

The geometry of an `ApertureMask` can be initialised with the standard `DataContainer` methods (see `Parameters` below). Regardless of which method is used, the following columns must be present:

```
x      y
arcsec arcsec
float  float
```

Certain keywords need to also be included in the ascii header:

```
# id: <int>
# conserve_image: <bool>
# x_unit: <str>
# y_unit: <str>
```

If `conserve_image` is `False`, the flux from all sources in the aperture is summed and distributed uniformly over the aperture area.

Parameters

filename

[str] Path to ASCII file containing the columns listed above

table

[astropy.Table] An astropy Table containing the columns listed above

array_dict

[dict] A dictionary containing the columns listed above: {x: [...], y: [...], id: <int>, conserve_image: <bool>}

Other Parameters

pixel_scale

[float] [arcsec] Defaults to "`!INST.pixel_scale`" from the config

id

[int] An integer to identify the `ApertureMask` in a list of apertures

apply_to(*obj*, ***kwargs*)

See parent docstring.

fov_grid(*which='edges'*, ***kwargs*)

Return a header with the sky coordinates.

get_header()

get_mask()

For placing over FOVs if the Aperture is rotated w.r.t. the field.

property `hdu`

property `header`

property `mask`

plot(*axes=None*)

class scopesim.effects.apertures.**RectangularApertureMask**(***kwargs*)

Bases: [ApertureMask](#)

get_table(***kwargs*)

class scopesim.effects.apertures.**SlitWheel**(***kwargs*)

Bases: [Effect](#)

Selection of predefined spectroscopic slits and possibly other field masks.

It should contain an open position. A user can define a non-standard slit by directly using the Aperture effect.

Parameters

slit_names

[list of str]

filename_format

[str] A f-string for the path to the slit files

current_slit

[str] Default name

Examples

This Effect assumes a folder full of ASCII files containing the edges of each slit. Each file should be names the same except for the slit's name or identifier.

This example assumes a folder masks containing the slit ASCII files with the naming convention: slit_A.dat, slit_B.dat, etc.

```
name: slit_wheel
class: SlitWheel
kwargs:
  slit_names:
    - A
    - B
    - C
  filename_format: "masks/slit_{}.dat"
  current_slit: "C"
```

add_slit(*newslit, name=None*)

Add a slit to the SlitWheel.

Parameters

newslit

[Slit]

name

[string] Name to be used for the new slit. If None, a name from the newslit object is used.

apply_to(*obj, **kwargs*)

Use apply_to of current_slit.

change_slit(*slitname=None*)

Change the current slit.

property current_slit

Return the currently used slit.

fov_grid(*which='edges', **kwargs*)

See parent docstring.

get_table()

Create a table of slits with centre position, width and length.

Width is defined as the extension in the y-direction, length in the x-direction. All values are in milliarcsec.

required_keys = {'current_slit', 'filename_format', 'slit_names'}

scopesim.effects.apertures.**make_aperture_polygon**(*left, right, top, bottom, angle, shape, **kwargs*)

scopesim.effects.apertures.**mask_from_coords**(*x, y, pixel_scale*)

scopesim.effects.apertures.**points_on_a_circle**(*n, x0=0, y0=0, dx=1, dy=1, offset=0*)

scopesim.effects.apertures.**rotate**(*x, y, x0, y0, angle*)

Rotate a line by *angle* [deg] around the point (*x0, y0*).

scopesim.effects.data_container module

class scopesim.effects.data_container.**DataContainer**(*filename=None, table=None, array_dict=None, cmds=None, **kwargs*)

Bases: `object`

A class to hold data files needed by all Effects objects.

Parameters**filename**

[str] Path to file containing data. Accepted formats: ASCII table, FITS table, FITS image

table

[astropy.Table] An astropy Table containing data

array_dict

[dict] A dictionary out of which an astropy.Table object can be constructed.

kwargs

addition meta data

Notes

If a table is to be generated from an `array_dict` parameter, column units can be passed as keyword arguments (`kwargs`) using the following format:

```
Datacontainer(... , <column name>_unit="<unit string>")
```

where unit string is a string recognised by `astropy.units`. Any additional table meta-data can also be passed using this format.

Attributes**data**

[astropy.Table, fits.HDUList] A generic property method which returns the data from the file. Any function calling this should be prepared to handle both data formats

meta

[dict] Contains all meta data read in from the file’s header, and/or passed via kwargs.

table

[astropy.Table] If the file has a table format (ASCII or FITS) it is read in immediately and stored in `.table`

_file

[HDUList pointer] If the file is a FITS image or cube, the data is only read in when needed in order to save on memory usage. `._file` contains a pointer to the data open FITS file.

property data

get_data(*ext=0, layer=None*)

Return either a table or a ImageHDU object.

Note: Use this call for reading in individual FITS extensions.

The `.data` handle will read in **all** extensions and return an HDUList object

Parameters**ext**

[int]

layer

[int] If the FITS extension is a data cube, layer corresponds to a slice from this cube of `<ImageHDU>.data[layer, :, :]`

Returns**data_set**

[astropy.Table, fits.ImageHDU]

property is_fits

validate(*etype*)

scopesim.effects.detector_list module

TBA.

class scopesim.effects.detector_list.**DetectorList**(**kwargs)

Bases: [Effect](#)

A description of detector positions and properties.

The list of detectors must have the following table columns

id	x_cen	y_cen	x_size	y_size	pixel_size	angle	gain
----	-------	-------	--------	--------	------------	-------	------

where:

- “id” is a reference id for the chip (fits header EXTNAME)
- “x_cen” and “y_cen” [mm] are the physical coordinates of centre of the chip on the detector plane
- “x_size”, “y_size” [mm, pixel] are the width/height of the chip

- “pixel_size” [mm] is the physical size of pixels in the detector
- “angle” [deg] is the rotation of the detector relative to the x-axis
- “gain” [e-/ADU] is the conversion factor for electrons (photons) to ADUs

The units for each column (except id) must be given in the meta data using the format <colname>_unit. E.g. x_size_unit. See examples below.

Note: Currently only the units specified below are accepted.

For x(y)_size_unit, acceptable units are mm, pixel

Parameters

filename

[str, optional] Filename of the ASCII file with the detector description. See examples

array_dict

[dict] Dict containing the detector description. See examples

image_plane_id

[int] Which image plane the detector will look at (generally 0)

Examples

With the array_dict feature

```
- name: single_detector
  class: DetectorList
  kwargs:
    image_plane_id : 0
    array_dict:
      id: [1]
      x_cen: [0.]
      y_cen: [0.]
      x_size: [5.12]
      y_size: [5.12]
      pixel_size: [0.01]
      angle: [0.]
      gain: [1.0]
    x_cen_unit: mm
    y_cen_unit: mm
    x_size_unit: mm
    y_size_unit: mm
    pixel_size_unit: mm
    angle_unit: deg
    gain_unit: electron/adu
```

Or referring to a table contained in a separate ASCII file

```
- name : full_detector_array
  class : DetectorList
  kwargs :
    filename : "detecotr_list.dat"
```

(continues on next page)

(continued from previous page)

```
active_detectors : [1, 3]
image_plane_id : 0
```

where the file detector_list.dat contains the following information

```
# x_cen_unit : mm
# y_cen_unit : mm
# x_size_unit : pix
# y_size_unit : pix
# pixel_size_unit : mm
# angle_unit : deg
# gain_unit : electron/adu
#
id  x_cen  y_cen  x_size  y_size  pixel_size  angle  gain
1   -63.94  0.00   4096   4096      0.015    0.0    1.0
2    0.00   0.00   4096   4096      0.015   90.0    1.0
3    63.94   0.00   4096   4096      0.015  180.0    1.0
```

property active_table

apply_to(obj, **kwargs)

TBA.

detector_headers(ids=None)

fov_grid(which='edges', **kwargs)

Return an ApertureMask object. kwargs are “pixel_scale” [arcsec].

property image_plane_header

property image_plane_id: int

Get ID of the corresponding image plane.

plot(axes=None)

class scopesim.effects.detector_list.DetectorWindow(pixel_size, x, y, width, height=None, angle=0, gain=1, units='mm', **kwargs)

Bases: [DetectorList](#)

For when a full DetectorList is too cumbersome.

Parameters

pixel_size

[float] [mm pixel-1] Physical pixel size

x, y

[float] [mm] Position of window centre relative to optical axis

width, height=None

[float] [mm] Dimensions of window. If height is None, height=width

angle

[float, optional] [deg] Rotation of window

gain

[float, optional] [ADU/e-]

units

[str, optional] [mm, pixel] Default “mm”. Sets the input parameter units. If “pixel”, (x, y, width, height) are multiplied by pixel_size

scopesim.effects.effects module

Contains base class for effects.

class scopesim.effects.effects.**Effect**(cmds=None, **kwargs)

Bases: *DataContainer*

Base class for representing the effects (artifacts) in an optical system.

The **Effect** class is conceived to independently apply the changes that an optical component (or series thereof) has on an incoming 3D description of an on-sky object. In other words, **an Effect object should receive a derivative of a ``Source`` object, alter it somehow, and return it.**

The interface for the Effect base-class has been kept very general so that it can easily be sub-classed as data for new effects becomes available. Essentially, a sub-classed Effects object must only contain the following attributes:

- self.meta - a dictionary to contain meta data.
- self.apply_to(obj, **kwargs) - a method which accepts a Source-derivative and returns an instance of the same class as obj
- self.fov_grid(which="", **kwargs)

Parameters

See :class:`DataContainer` for input parameters

apply_to(obj, **kwargs)

TBA.

property display_name

fov_grid(which="", **kwargs)

Return the edges needed to generate FieldOfViews for an observation.

Parameters**which**

[str] [“waveset”, “edges”, “shifts”] where: * waveset - wavelength bin extremes * edges - on sky coordinate edges for each FOV box * shifts - wavelength dependent FOV position offsets

Returns**waveset**

[list] [um] N+1 wavelengths that set edges of N spectral bins

edges

[list of lists] [arcsec] Contains a list of footprint lists

shifts

[list of 3 lists] [wave, dx, dy] Contains lists corresponding to the (dx, dy) offset from the optical axis (0, 0) induced for each wavelength in (wave) [um, arcsec, arcsec]

property include

info()

Print basic information on the effect, notably the description.

property meta_string

report(*filename=None, output='rst', rst_title_chars='*+', **kwargs*)

For Effect objects, generates a report based on the data and meta-data.

This is to aid in the automation of the documentation process of the instrument packages in the IRDB.

Note: If the Effect can generate a plot, this will be saved to disc

Parameters**filename**

[str, optional] Where to save the RST file

output

[str, optional] ["rst", "latex"] Output file format

rst_title_chars

[2-str, optional] Two unique characters used to denote rst subsection headings. Options: =
- ` : ' " ~ ^ _ * + # < >

Returns**rst_str**

[str] The full reStructureText string

Notes

The format of the RST output is as follows:

```
<ClassType>: <effect name>
*****
File Description: <description for file meta data>
Class Description: <description from class docstring>
Changes: <list of changes from file meta data>

Data
++++
.. figure:: <Figure_name>.png
    If the <Effect> object contains a ``.plot()`` function, add
    plot and write it to disc
Figure caption

Table caption
Table
    If the <Effect> object contains a ``.table()`` function, add
    a pprint version of the table

Meta-data
+++++++
::
    A code block print out of the ``.meta`` dictionary
```

`update(**kwargs)`

scopesim.effects.effects_utils module

TBA.

`scopesim.effects.effects_utils.combine_surface_effects(surface_effects)`

`scopesim.effects.effects_utils.empty_surface_list(**kwargs)`

`scopesim.effects.effects_utils.get_all_effects(effects, effect_class)`

`scopesim.effects.effects_utils.is_spectroscope(effects)`

`scopesim.effects.effects_utils.make_effect(effect_dict, cmds=None, **properties)`

`scopesim.effects.effects_utils.scopesim_effect_classes(base_effect=<class
'scopesim.effects.effects.Effect'>)`

`scopesim.effects.effects_utils.z_order_in_range(z_eff, z_range: range) → bool`

Return True if any of the `z_orders` in `z_eff` is in the given range.

The `z_range` parameter can be constructed as `range(z_min, z_max)`.

Parameters

z_eff

[int or list of ints] `z_order(s)` of the effect.

z_range

[range] range object of allowed `z_order` values.

Returns

bool

True if at least one `z_order` is in range, False otherwise.

scopesim.effects.electronic module

Electronic detector effects - related to detector readout.

Classes: - `DetectorModePropertiesSetter` - set parameters for readout mode - `AutoExposure` - determine DIT and NDIT automatically - `SummedExposure` - simulates a summed stack of `ndit` exposures - `PoorMansHxRGReadoutNoise` - simple readout noise for HAWAII detectors - `BasicReadoutNoise` - readout noise - `ShotNoise` - realisation of Poissonian photon noise - `DarkCurrent` - add dark current - `LinearityCurve` - apply detector (non-)linearity and saturation - `ReferencePixelBorder` - `BinnedImage` - `UnequalBinnedImage` - `Bias` - adds constant bias level to readout

Functions: - `make_ron_frame` - `pseudo_random_field`

class `scopesim.effects.electronic.AutoExposure(**kwargs)`

Bases: [*Effect*](#)

Determine DIT and NDIT automatically from `ImagePlane`.

DIT is determined such that the maximum value in the incident photon flux (including astronomical source, sky and thermal backgrounds) fills the full well of the detector (`!DET.full_well`) to a given fraction (`!OBS.autoexposure.fill_frac`). NDIT is determined such that `DIT * NDIT` results in the requested exposure time.

The requested exposure time is taken from `!OBS.exptime`.

The effects sets the parameters *!OBS.dit* and *!OBS.ndit*.

Examples

The parameters *!OBS.exptime*, *!DET.full_well* and *!DET.mindit* should be defined as properties in the respective subsections.

```
name: auto_exposure
description: automatic determination of DIT and NDIT
class: AutoExposure
include: True
kwargs:
    fill_frac: "!OBS.auto_exposure.fill_frac"
```

apply_to(*obj*, ****kwargs**)

TBA.

class scopesim.effects.electronic.**BasicReadoutNoise**(****kwargs**)

Bases: *Effect*

Readout noise computed as: $\text{ron} * \sqrt{\text{NDIT}}$.

apply_to(*det*, ****kwargs**)

TBA.

plot(*det*)

plot_hist(*det*, ****kwargs**)

class scopesim.effects.electronic.**Bias**(****kwargs**)

Bases: *Effect*

Adds a constant bias level to readout.

apply_to(*obj*, ****kwargs**)

TBA.

class scopesim.effects.electronic.**BinnedImage**(****kwargs**)

Bases: *Effect*

apply_to(*det*, ****kwargs**)

TBA.

class scopesim.effects.electronic.**DarkCurrent**(****kwargs**)

Bases: *Effect*

required: dit, ndit, value

apply_to(*obj*, ****kwargs**)

TBA.

plot(*det*, ****kwargs**)

class scopesim.effects.electronic.**DetectorModePropertiesSetter**(****kwargs**)

Bases: *Effect*

Set mode specific curr_sys properties for different detector readout modes.

A little class (DetectorModePropertiesSetter) that allows different "!DET" properties to be set on the fly.

Parameters

mode_properties

[dict] A dictionary containing the DET parameters to be changed for each mode. See below for an example yaml entry.

Examples

Add the values for the different detector readout modes to all the relevant detector yaml files. In this case the METIS HAWAII (L, M band) and GeoSnap (N band) detectors: METIS_DET_IMG_LM.yaml , METIS_DET_IMG_N.yaml

```
- name: lm_detector_readout_parameters
  class: DetectorModePropertiesSetter
  kwargs:
    mode_properties:
      fast:
        mindit: 0.04
        full_well: !!float 1e5
        ron: 70
      slow:
        mindit: 1.3
        full_well: !!float 1e5
        ron: 14
```

Add the OBS dict entry !OBS.detector_readout_mode to the properties section of the mode_yamls descriptions in the default.yaml files.

```
mode_yamls:
- object: observation
  alias: OBS
  name: lss_l
  yamls:
    ...
  properties:
    ...
    detector_readout_mode: slow
```

apply_to(obj, **kwargs)

TBA.

list_modes()

Return list of available detector modes.

select_mode(obj, **kwargs)

Automatically select detector mode based on image plane peak value.

Select the mode with lowest readnoise that does not saturate the detector. When all modes saturate, select the mode with the lowest saturation level (peak to full_well).

class scopesim.effects.electronic.**LinearityCurve**(**kwargs)

Bases: *Effect*

Detector linearity effect.

The detector linearity curve is set in terms of *incident* flux (e/s) and *measured* detector values (ADU).

Examples

The effect can be instantiated in various ways.:

```
- name: detector_linearity
  class: LinearityCurve
  kwargs:
    filename: FPA_linearity.dat

- name: detector_linearity
  class: LinearityCurve
  kwargs:
    array_dict: {incident: [0, 77000, 99999999999],
                  measured: [0, 77000, 77000]}

- name: detector_linearity
  class: LinearityCurve
  kwargs:
    incident: [0, 77000, 99999999]
    measured: [0, 77000, 77000]
```

apply_to(obj, **kwargs)

TBA.

plot(**kwargs)

class scopesim.effects.electronic.PoorMansHxRGReadoutNoise(**kwargs)

Bases: [Effect](#)

apply_to(det, **kwargs)

TBA.

plot(det, **kwargs)

plot_hist(det, **kwargs)

class scopesim.effects.electronic.Quantization(**kwargs)

Bases: [Effect](#)

Converts raw data to whole photons.

apply_to(obj, **kwargs)

TBA.

class scopesim.effects.electronic.ReferencePixelBorder(**kwargs)

Bases: [Effect](#)

apply_to(implane, **kwargs)

TBA.

plot(implane, **kwargs)

class scopesim.effects.electronic.ShotNoise(**kwargs)

Bases: [Effect](#)

apply_to(det, **kwargs)

TBA.

`plot(det)`

`plot_hist(det, **kwargs)`

class scopesim.effects.electronic.SummedExposure(**kwargs)

Bases: [Effect](#)

Simulates a summed stack of ndit exposures.

`apply_to(obj, **kwargs)`

TBA.

class scopesim.effects.electronic.UnequalBinnedImage(**kwargs)

Bases: [Effect](#)

`apply_to(det, **kwargs)`

TBA.

scopesim.effects.electronic.make_ron_frame(image_shape, noise_std, n_channels, channel_fraction,
line_fraction, pedestal_fraction, read_fraction)

scopesim.effects.electronic.pseudo_random_field(scale=1, size=(1024, 1024))

scopesim.effects.fits_headers module

class scopesim.effects.fits_headers.EffectsMetaKeywords(cmds=None, **kwargs)

Bases: [ExtraFitsKeywords](#)

Adds meta dictionary info from all Effects to the FITS headers.

Parameters

ext_number

[int, list of ints, optional] Default 0. The numbers of the extensions to which the header keywords should be added

add_excluded_effects

[bool, optional] Default False. Add meta dict for effects with <effect>.include=False

keyword_prefix

[str, optional] Default “HIERARCH SIM”. Custom FITS header keyword prefix. Effect meta dict entries will appear in the header as: <keyword_prefix> EFFn <key> : <value>

Examples

Yaml file entry:

```
name: effect_dumper
class: EffectsMetaKeywords
description: adds all effects meta dict entries to the FITS header
kwargs:
  ext_number: [0, 1]
  add_excluded_effects: False
  keyword_prefix: HIERARCH SIM
```

apply_to(*hdul*, ***kwargs*)

See parent docstring.

class scopesim.effects.fits_headers.**ExtraFitsKeywords**(*cmds=None*, ***kwargs*)

Bases: [Effect](#)

Extra FITS header keywords to be added to the pipeline FITS files.

These keywords are ONLY for keywords that should be MANUALLY ADDED to the headers after a simulation is read-out by the detector.

Simulation parameters (Effect kwargs values, etc) will be added automatically by ScopeSim in a different function, but following this format.

The dictionaries should be split into different HIERARCH lists, e.g.:

- HIERARCH ESO For ESO specific keywords
- HIERARCH SIM For ScopeSim specific keywords, like simulation parameters
- HIERARCH MIC For MICADO specific keywords, (unsure what these would be yet)

More HIERARCH style keywords can also be added as needed for other use-cases.

Parameters

filename

[str, optional] Name of a .yaml nested dictionary file. See below for examples

yaml_string

[str, optional] A triple-” string containing the contents of a yaml file

header_dict

[nested dicts, optional] A series of nested python dictionaries following the format of the examples below. This keyword allows these dicts to be defined directly in the Effect yaml file, rather than in a seperate header keywords file.

Examples

Specifying the extra FITS keywords directly in the .yaml file where the Effect objects are described.

```
name: extra_fits_header_entries
class: ExtraFitsKeywords
kwargs:
  header_dict:
    - ext_type: PrimaryHDU
      keywords:
        HIERARCH:
          ESO:
            ATM:
              TEMPERAT: -5
```

The contents of header_dict can also be abstracted away into a seperate file, e.g. `extra_FITS_keys.yaml`. The file format is described below in detail below.

```
name: extra_fits_header_entries
class: ExtraFitsKeywords
kwargs:
  filename: extra_FITS_keys.yaml
```

The Effect can be added directly in an iPython session.

```
>>> hdr_dic = {"ext_type": "PrimaryHDU",
               "keywords":
                 {"HIERARCH":
                  {"SIM":
                   {"hello": world}
                  }
                 }
               }
>>> extra_keys = ExtraFitsKeywords(header_dict=hdr_dic)
>>> optical_train.optics_manager.add_effect(extra_keys)
```

apply_to(*hdul*, ****kwargs**)

Add extra fits keywords from a yaml file including !, #-strings.

Parameters

optical_train

[scopesim.OpticalTrain, optional] Used to resolve #-strings

class scopesim.effects.fits_headers.**SimulationConfigFitsKeywords**(*cmds=None*, ****kwargs**)

Bases: [ExtraFitsKeywords](#)

Adds parameters from all config dictionaries to the FITS headers.

Parameters

ext_number

[int, list of ints, optional] Default 0. The numbers of the extensions to which the header keywords should be added

resolve

[bool] Default True. If True, all !-strings and #-strings are resolved via `from_currsys` before being add to the header. If False, the unaltered !-strings or #-strings are added to the header.

keyword_prefix

[str, optional] Default "HIERARCH SIM". Custom FITS header keyword prefix. Effect meta dict entries will appear in the header as: <keyword_prefix> SRCn <key> : <value>

Examples

Yaml file entry:

```
name: source_descriptor
class: SimulationConfigFitsKeywords
description: adds info from all config dicts to the FITS header
kwargs:
  ext_number: [0]
  resolve: False
  keyword_prefix: HIERARCH SIM
```

apply_to(*hdul*, ****kwargs**)

See parent docstring.

class scopesim.effects.fits_headers.SourceDescriptionFitsKeywords(*cmds=None, **kwargs*)

Bases: [ExtraFitsKeywords](#)

Adds parameters from all Source fields to the FITS headers.

Parameters

ext_number

[int, list of ints, optional] Default 0. The numbers of the extensions to which the header keywords should be added

keyword_prefix

[str, optional] Default “HIERARCH SIM”. Custom FITS header keyword prefix. Effect meta dict entries will appear in the header as: <keyword_prefix> SRCn <key> : <value>

Examples

Yaml file entry:

```
name: source_descriptor
class: SourceDescriptionFitsKeywords
description: adds info from all Source fields to the FITS header
kwargs:
  ext_number: [0]
  keyword_prefix: HIERARCH SIM
```

apply_to(*hdul, **kwargs*)

See parent docstring.

scopesim.effects.fits_headers.**flatten_dict**(*dic, base_key="", flat_dict=None, resolve=False, optics_manager=None*)

Flattens nested yaml dictionaries into a single level dictionary.

Parameters

dic

[dict]

base_key

[str]

flat_dict

[dict, optional] Top-level dictionary for recursive calls

resolve

[bool] If True, resolves #-str via from_currsys and #-str via optics_manager

optics_manager

[scopesim.OpticsManager] Required for resolving #-strings

Returns

flat_dict

[dict]

scopesim.effects.fits_headers.**get_relevant_extensions**(*dic, hdul*)

scopesim.effects.metis_lms_trace_list module

SpectralTraceList and SpectralTrace for the METIS LM spectrograph.

class scopesim.effects.metis_lms_trace_list.**MetisLMSEfficiency**(**kwargs)

Bases: *TERCurve*

Computes the grating efficiency (blaze function) for the METIS LMS.

The procedure is described in E-REP-ATC-MET-1016_1.0. For a given order (determined by the central wavelength) the grating efficiency is modelled as a squared sinc function of wavelength via the grating angle.

make_ter_curve(wcal, wavelen=None)

Compute the blaze function for the selected order.

class scopesim.effects.metis_lms_trace_list.**MetisLMSImageSlicer**(filename, ext_id='Aperture List', **kwargs)

Bases: *ApertureMask*

Treats the METIS LMS image slicer as an aperture mask effect.

This helps in building a *FieldOfView* object that combines the spatial field of the slicer with the spectral range covered by the LMS setting.

The effect differs from its parent class *ApertureMask* in the initialisation from the *Aperture List* extension of the trace file *!OBS.trace_file*.

class scopesim.effects.metis_lms_trace_list.**MetisLMSSpectralTrace**(hdulist, spslice, params, **kwargs)

Bases: *SpectralTrace*

SpectralTrace for the METIS LM spectrograph.

compute_interpolation_functions()

Define the transforms between (xi, lam) and (x, y).

The LMS transforms actually operate on phase rather than wavelength, hence the necessity of defining pre- and posttransforms on the lam variable.

fov_grid()

Provide information on the source space volume required by the effect.

Returns

A dictionary with entries *wave_min* and *wave_max*, *x_min*, *y_min*, *x_max*, *y_max*. Spatial limits refer to the sky and are given in arcsec.

fp2sky(fp_x)

Convert position in FP2 to position on sky.

get_matrices()

Extract matrix from lms_dist_poly.txt.

Evaluate polynomial to obtain matrices A, B, AI and BI at grism angle given echelle order and slice number

Parameters**order**

[int] Echelle order

spslice

[int] Slice number

angle

[float] Grism angle in degrees

Returns**dict of four np.arrays of shape (4, 4) each****get_waverange**(*det_mm_lims*)

Determine wavelength range covered by spec. trace on image plane.

lam2phase(*lam*)

Convert wavelength to phase.

Phase is $\text{lam} * \text{order} / (2 * \text{grat_spacing})$.**Parameters****lam**

[ndarray (float)] wavelength (um)

Returns**Phase**

[ndarray]

phase2lam(*phase*)

Convert phase to wavelength.

Wavelength is $\text{phase} * 2 * \text{grat_spacing} / \text{order}$ **Parameters****phase**

[ndarray (float)] phase (dimensionless)

Returns**wavelength**

[ndarray (um)]

sky2fp(*xi*)

Convert position in arcsec to position in FP2.

class scopesim.effects.metis_lms_trace_list.**MetisLMSSpectralTraceList**(***kwargs*)Bases: [*SpectralTraceList*](#)

SpectralTraceList for the METIS LM spectrograph.

apply_to(*obj, **kwargs*)

See parent docstring.

make_spectral_traces()

Compute the transformations by interpolation.

rectify_cube(*hdulist, xi_min=None, xi_max=None, interps=None, **kwargs*)

Rectify an IFU observation into a data cube

The HDU list (or fits file) must have been created with the present OpticalTrain (or an identically configured one).

Parameters

hdulist

[str or fits.HDUList] an ifu observation created with the present OpticalTrain

xi_min, xi_max

[float [arcsec]] Spatial limits of the image slicer on the sky. For METIS LMS, these values need not be provided by the user.

interps

[list of interpolation functions] If provided, there must be one for each image extension in *hdulist*. The functions go from pixels to the images and can be created with, e.g., RectBi-variateSpline.

`scopesim.effects.metis_lms_trace_list.echelle_setting(wavelength, grat_spacing, wcal_def)`

Determine optimal echelle rotation angle for wavelength.

Parameters**lambda**

[float] central wavelength in microns

grat_spacing

[float] grating rule spacing in microns

wcal_def: fits.TableHDU, fits.BinTableHDU, Table, str

definition of the wavelength calibration parameters If str, interpreted as name of a fits file, with a table extension 'WCAL'.

Returns

a *dict* with entries

- **Ord:** echelle order
- **Angle:** grism angle
- **Phase:** phase

scopesim.effects.obs_strategies module

Effects describing observing strategies.

- ChopNodCombiner: simulate chop-nod cycle

class `scopesim.effects.obs_strategies.ChopNodCombiner(**kwargs)`

Bases: *Effect*

Creates and combines 4 images for each of the chop/nod positions.

- AA : original position (dx, dy) = (0, 0)
- AB : chop position (dx, dy) = chop_offsets
- BA : nod position (dx, dy) = nod_offsets
- BB : chop-nod position (dx, dy) = nod_offsets + chop_offsets

Images are combined using:

```
im_combined = (AA - AB) - (BA - BB)
```

If no `nod_offset` is given, it is set to the inverse of `chop_offset`.

ChopNodCombiner is a detector effect and should be placed last in the detector yaml (after the noise effects).

apply_to(*obj*, ***kwargs*)

TBA.

`scopesim.effects.obs_strategies.chop_nod_image(img, chop_offsets, nod_offsets=None)`

Create four copies and combine in chop-nod pattern.

scopesim.effects.psf_utils module

`scopesim.effects.psf_utils.cutout_kernel(image, fov_header, kernel_header=None)`

`scopesim.effects.psf_utils.get_bkg_level(obj, bg_w)`

Determine the background level of image or cube slices.

Returns a scalar if obj is a 2d image or a vector if obj is a 3D cube (one value for each plane). The method for background determination is decided by self.meta["bkg_width"]: If 0, the background is returned as zero (implying no background subtraction). If -1, the background is estimated as the median of the entire image (or cube plane). If positive, the background is estimated as the median of a frame of width *bkg_width* around the edges.

`scopesim.effects.psf_utils.get_psf_wave_exts(hdu_list, wave_key='WAVE0')`

Return a dict of {extension : wavelength}.

Parameters

hdu_list

Returns

wave_set, wave_ext

`scopesim.effects.psf_utils.get_strehl_cutout(fov_header, strehl_imagehdu)`

`scopesim.effects.psf_utils.get_total_wfe_from_table(tbl)`

`scopesim.effects.psf_utils.make_strehl_map_from_table(tbl, pixel_scale=<Quantity 1. arcsec>)`

`scopesim.effects.psf_utils.nearest_index(x, x_array)`

`scopesim.effects.psf_utils.nmrms_from_strehl_and_wavelength(strehl: float, wavelength, strehl_hdu, plot=False) → float`

Return the wavefront error needed to make a PSF with desired strehl ratio.

Parameters

strehl

[float] [0.001, 1] Desired strehl ratio. Values 1<sr<100 will be scale to <1

wavelength

[float] [um]

strehl_hdu

[np.ndarray] 2D map of strehl ratio as a function of wavelength [um] and residual wavefront error [nm RMS]

plot

[bool]

Returns

nm

[float] [nm] residual wavefront error for generating an on-axis AnisoCADO PSF with the desired strehl ratio at a given wavelength

`scopesim.effects.psf_utils.rescale_kernel(image, scale_factor, spline_order=None)`

`scopesim.effects.psf_utils.rotational_blur(image, angle)`

Rotate and coadd an image over a given angle to imitate a blur.

Parameters

image

[array] Image to blur

angle

[float] [deg] Angle over which the image should be rotationally blurred

Returns

image_rot

[array] Blurred image

`scopesim.effects.psf_utils.round_kernel_edges(kernel)`

`scopesim.effects.psf_utils.sigma2gauss(sigma, x_size=15, y_size=15)`

`scopesim.effects.psf_utils.strehl2sigma(strehl)`

`scopesim.effects.psf_utils.wfe2gauss(wfe, wave, width=None)`

`scopesim.effects.psf_utils.wfe2strehl(wfe, wave)`

scopesim.effects.psf module

class `scopesim.effects.psf.AnalyticalPSF(**kwargs)`

Bases: [*PSF*](#)

class `scopesim.effects.psf.AnisocadoConstPSF(**kwargs)`

Bases: [*SemiAnalyticalPSF*](#)

Makes a SCAO on-axis PSF with a desired Strehl ratio at a given wavelength.

To make the PSFs a map connecting Strehl, Wavelength, and residual wavefront error is required.

Parameters

filename

[str] Path to Strehl map with axes (x, y) = (wavelength, wavefront error).

strehl

[float] Desired Strehl ratio. Either percentage [1, 100] or fractional [1e-3, 1].

wavelength

[float] [um] The given strehl is valid for this wavelength.

psf_side_length

[int] [pixel] Default is 512. Side length of the kernel images.

offset

[tuple] [arcsec] SCAO guide star offset from centre (dx, dy).

rounded_edges

[bool] Default is True. Sets all halo values below a threshold to zero. The threshold is determined from the max values of the edge rows of the kernel image.

Other Parameters**convolve_mode**

[str] ["same", "full"] convolution keywords from `scipy.signal.convolve`

Examples

Add an AnisocadoConstPSF with code:

```
from scopesim.effects import AnisocadoConstPSF
psf = AnisocadoConstPSF(filename="test_AnisoCADO_rms_map.fits",
                        strehl=0.5,
                        wavelength=2.15,
                        convolve_mode="same",
                        psf_side_length=512)
```

Add an AnisocadoConstPSF to a yaml file:

```
effects:
-   name: Ks_Strehl_40_PSF
    description: A 40% Strehl PSF over the field of view
    class: AnisocadoConstPSF
    kwargs:
      filename: "test_AnisoCADO_rms_map.fits"
      strehl: 0.5
      wavelength: 2.15
      convolve_mode: full
      psf_side_length: 512
```

get_kernel(fov)

property nmRms

plot(obj=None, **kwargs)

remake_kernel(x)

Remake the kernel based on either a pixel_scale of FieldOfView.

Parameters

x: float, FieldOfView
[um] if float

property strehl_ratio

property wavelength

class scopesim.effects.psfs.DiscretePSF(**kwargs)

Bases: *PSF*

class scopesim.effects.psfs.FieldConstantPSF(**kwargs)

Bases: *DiscretePSF*

A PSF that is constant across the field.

For spectroscopy, a wavelength-dependent PSF cube is built, where for each wavelength the reference PSF is scaled proportional to wavelength.

get_kernel(fov)

Find nearest wavelength and build PSF kernel from file

make_psf_cube(fov)

Create a wavelength-dependent psf cube

plot()

class scopesim.effects.psfs.**FieldVaryingPSF**(**kwargs)

Bases: [*DiscretePSF*](#)

TBA.

Parameters

sub_pixel_flag

[bool, optional]

flux_accuracy

[float, optional] Default 1e-3. Level of flux conservation during rescaling of kernel

apply_to(fov, **kwargs)

See parent docstring.

get_kernel(fov)

plot()

property **strehl_imagehdu**

The HDU containing the positional info for kernel layers.

class scopesim.effects.psfs.**GaussianDiffractionPSF**(diameter, **kwargs)

Bases: [*AnalyticalPSF*](#)

fov_grid(which='waveset', **kwargs)

See parent docstring.

get_kernel(fov)

plot()

update(**kwargs)

class scopesim.effects.psfs.**NonCommonPathAberration**(**kwargs)

Bases: [*AnalyticalPSF*](#)

TBA.

Needed: pixel_scale Accepted: kernel_width, strehl_drift

fov_grid(which='waveset', **kwargs)

See parent docstring.

get_kernel(obj)

plot()

property **total_wfe**

class scopesim.effects.psfs.PSF(**kwargs)

Bases: [Effect](#)

apply_to(obj, **kwargs)

Apply the PSF.

fov_grid(which='waveset', **kwargs)

See parent docstring.

get_kernel(obj)

plot(obj=None, **kwargs)

class scopesim.effects.psfs.PoorMansFOV(pixel_scale, spec_dict, recursion_call=False)

Bases: [object](#)

class scopesim.effects.psfs.SeeingPSF(fwhm=1.5, **kwargs)

Bases: [AnalyticalPSF](#)

Currently only returns gaussian kernel with a fwhm [arcsec].

Parameters

fwhm

[float] [arcsec]

get_kernel(fov)

plot()

class scopesim.effects.psfs.SemiAnalyticalPSF(**kwargs)

Bases: [PSF](#)

class scopesim.effects.psfs.Vibration(**kwargs)

Bases: [AnalyticalPSF](#)

Creates a wavelength independent kernel image.

get_kernel(obj)

scopesim.effects.rotation module

Effects related to rotation of the field/CCD.

Classes: - RotateCCD - Rotates CCD by integer multiples of 90 degrees

class scopesim.effects.rotation.Rotate90CCD(**kwargs)

Bases: [Effect](#)

Rotates CCD by integer multiples of 90 degrees.

rotations kwarg is number of counter-clockwise rotations

Author: Dave jones

apply_to(obj, **kwargs)

See parent docstring.

scopesim.effects.shifts module**class** scopesim.effects.shifts.**AtmosphericDispersion**(**kwargs)Bases: [*Shift3D*](#)

Used to generate the wavelength bins based on shifts due to the atmosphere.

Doesn't contain an `apply_to` function, but provides information through the `fov_grid` function.**get_table**(**kwargs)Called by the `fov_grid` method of `Shift3D`.**Returns****tbl**

[astropy.Table] A table with the columns - waves : [um] - dx, dy : [arcsec]

NotesSuccess! Returns the same values as: <http://gtc-phase2.gtc.iac.es/science/astroweb/atmosRefraction.php>**class** scopesim.effects.shifts.**AtmosphericDispersionCorrection**(**kwargs)Bases: [*Shift3D*](#)

Alters the position on the detector for a FOV object (WCS_prefix="D").

Only acts on FOVs during the main effects loop in `OpticalTrain`. For the sake of computational efficiency, the ADC can be instructed to counteract the atmospheric diffraction during the `OpticalTrain` setup phase, by passing the kwarg: `quick_adc=True`**Parameters****kwargs****apply_to**(fov, **kwargs)

See parent docstring.

fov_grid(which='shifts', **kwargs)

See parent docstring.

plot()**class** scopesim.effects.shifts.**Shift3D**(**kwargs)Bases: [*Effect*](#)**apply_to**(obj, **kwargs)

See parent docstring.

fov_grid(which='shifts', **kwargs)

See parent docstring.

get_table(**kwargs)**plot**()**scopesim.effects.shifts.atmospheric_refraction**(lam, z0=60, temp=0, rel_hum=60, pres=750,
lat=-24.5, h=3064)

Compute atmospheric refraction.

The function computes the angular difference between the apparent position of a star seen from the ground and its true position.

Parameters**lam**

[float, np.ndarray] [um] wavelength bin centres

z0

[float, optional] [deg] zenith distance. Default is 60 deg from zenith

temp

[float, optional] [deg C] ground temperature. Default is 0 deg C

rel_hum

[float, optional] [%] relative humidity. Default is 60%

pres

[float, optional] [millibar] air pressure. Default is 750 mbar

lat

[float, optional] [deg] latitude. Default set for Cerro Armazones: 24.5 deg South

h

[float, optional] [m] height above sea level. Default is 3064 m

Returns**ang**

[float, np.ndarray] [arcsec] angle between real position and refracted position

References

See Stone 1996 and the review by S. Pedraz - <http://www.caha.es/newsletter/news03b/pedraz/newslet.html>

`scopesim.effects.shifts.get_pixel_border_waves_from_atmo_disp(**kwargs)`

TBA.

Parameters****kwargs**

[TYPE] DESCRIPTION.

Returns**wave_pixel_edges**

[TYPE] DESCRIPTION.

shifts_angle_edges

[TYPE] DESCRIPTION.

scopesim.effects.shutter module

Contains the Shutter effect.

class `scopesim.effects.shutter.Shutter(**kwargs)`

Bases: *Effect*

Simulate a closed shutter, useful for dark exposures.

apply_to(*obj*, ***kwargs*)

Set all pixels of image plane to zero.

scopesim.effects.spectral_efficiency module

Spectral grating efficiencies.

class scopesim.effects.spectral_efficiency.**SpectralEfficiency**(**kwargs)

Bases: *Effect*

Applies the grating efficiency (blaze function) for a SpectralTraceList.

apply_to(obj, **kwargs)

Interface between FieldOfView and SpectralEfficiency.

get_efficiencies()

Read efficiencies from file, returns a dictionary.

plot()

Plot the grating efficiencies.

scopesim.effects.spectral_trace_list module

Effect for mapping spectral cubes to the detector plane.

The Effect is called *SpectralTraceList*, it applies a list of *spectral_trace_list_utils.SpectralTrace* objects to a *FieldOfView*.

class scopesim.effects.spectral_trace_list.**SpectralTraceList**(**kwargs)

Bases: *Effect*

List of spectral trace geometries for the detector plane.

Should work in concert with an ApertureList (or ApertureMask) object and a DetectorList object

Spectral trace patterns are to be kept in a *fits.HDUList* with one or more *fits.BinTableHDU* extensions, each one describing the geometry of a single trace. The first extension should be a *BinTableHDU* connecting the traces to the correct *Aperture* and *ImagePlane* objects.

The *fits.HDUList* objects can be loaded using one of these two keywords:

- **filename**: for on disk FITS files, or
- **hdulist**: for in-memory *fits.HDUList* objects

The format and contents of the extensions in the *HDUList* (FITS file) object is listed below

apply_to(obj, **kwargs)

Interface between *FieldOfView* and *SpectralTraceList*.

This is called twice: 1. During setup of the required *FieldOfView* objects, the *SpectralTraceList* is asked for the source space volumes that it requires (spatial limits and wavelength limits). 2. During “observation” the method is passed a single *FieldOfView* object and applies the mapping to the image plane to it. The *FieldOfView* object is associated to one *SpectralTrace* from the list, identified by *meta[“trace_id”]*.

property footprint

Return the footprint of the entire *SpectralTraceList*.

property image_plane_header

Create and return header for the *ImagePlane*.

make_spectral_traces()

Return a dictionary of spectral traces read in from a file.

plot(*wave_min=None, wave_max=None, axes=None, **kwargs*)

Plot every spectral trace in the spectral trace list.

Parameters

wave_min

[float, optional] Minimum wavelength, if any. If None, value from_currsys is used.

wave_max

[float, optional] Maximum wavelength, if any. If None, value from_currsys is used.

axes

[matplotlib axes, optional] The axes object to use for the plot. If None (default), a new figure with one axes will be created.

****kwargs**

[dict] Any other parameters passed along to the plot method of the individual spectral traces.

Returns

fig

[matplotlib figure] DESCRIPTION.

rectify_cube(*hdulist*)

Rectify traces and combine into a cube.

rectify_traces(*hdulist, xi_min=None, xi_max=None, interps=None, **kwargs*)

Create rectified 2D spectra for all traces in the list.

This method creates an HDU list with one extension per spectral trace, i.e. it essentially treats all traces independently. For the case of an IFU where the traces correspond to spatial slices for the same wavelength range, use method *rectify_cube* (not yet implemented).

Parameters

hdulist

[str or fits.HDUList] The result of scopesim readout()

xi_min, xi_max

[float [arcsec]] Spatial limits of the slit on the sky. This should be taken from the header of the hdulist, but this is not yet provided by scopesim. For the time being, these limits *must* be provided by the user.

interps

[list of interpolation functions] If provided, there must be one for each image extension in *hdulist*. The functions go from pixels to the images and can be created with, e.g. RectBivariateSpline.

update_meta()

Update fov related meta values.

The values describe the full extent of the spectral trace volume in wavelength and space

class scopesim.effects.spectral_trace_list.**SpectralTraceListWheel**(*cmds=None, **kwargs*)

Bases: [Effect](#)

A Wheel-Effect object for selecting between multiple gratings/grisms.

See SpectralTraceList for the trace file format description.

Parameters

trace_list_names

[list] The list of unique identifiers in the trace filenames

filename_format

[str] f-string that directs scopesim to the folder containing the trace files. This can be a !-string if the trace names are shared with other *Wheel effect objects (e.g. a FilterWheel). See examples.

current_trace_list

[str] default trace file to use

kwargs

[key-value pairs] Addition keywords that are passed to the SpectralTraceList objects See SpectralTraceList docstring

Examples

A simplified YAML file example taken from the OSIRIS instrument package:

```
alias: INST
name: OSIRIS_LSS

properties:
  decouple_detector_from_sky_headers: True
  grism_names:
    - R300B
    - R500B
    - R1000B
    - R2500V

effects:
  - name: spectral_trace_wheel
    description: grism wheel contining spectral trace geometries
    class: SpectralTraceListWheel
    kwargs:
      current_trace_list: "!OBS.grating_name"
      filename_format: "traces/LSS_{}_TRACE.fits"
      trace_list_names: "!INST.grism_names"

  - name: grating_efficiency
    description: OSIRIS grating efficiency curves, piggybacking on FilterWheel
    class: FilterWheel
    kwargs:
      minimum_throughput: !!float 0.
      filename_format: "gratings/{}.txt"
      current_filter: "!OBS.grating_name"
      filter_names: "!INST.grism_names"
```

apply_to(obj, **kwargs)

Use apply_to of current trace list.

property current_trace_list

```
required_keys = {'current_trace_list', 'filename_format', 'trace_list_names'}
```

scopesim.effects.spectral_trace_list_utils module

Utility classes and functions for SpectralTraceList.

This module contains

- the definition of the *SpectralTrace* class. The visible effect should always be a *SpectralTraceList*, even if that contains only one *SpectralTrace*.
- the definition of the *XiLamImage* class
- utility functions for use with spectral traces

class scopesim.effects.spectral_trace_list_utils.**SpectralTrace**(*trace_tbl*, *cmds=None*, ***kwargs*)

Bases: `object`

Definition of one spectral trace.

A SpectralTrace describes the mapping of spectral slit coordinates to the focal plane. The class reads an order layout and fits several functions to describe the geometry of the trace.

Slit coordinates are: - *xi* : spatial position along the slit [arcsec] - *lam* : Wavelength [um] Focal plane coordinates are: - *x*, *y* : [mm]

compute_interpolation_functions()

Compute various interpolation functions between slit and focal plane.

Focal plane coordinates are *x* and *y*, in mm. Slit coordinates are *xi* (spatial coordinate along the slit, in arcsec) and *lam* (wavelength, in um).

footprint(*wave_min=None*, *wave_max=None*, *xi_min=None*, *xi_max=None*)

Return corners of rectangle enclosing spectral trace.

Parameters

wave_min, wave_max

[float [um], Quantity] Minimum and maximum wavelength to compute the footprint on. If *None*, use the full range that spectral trace is defined on. Float values are interpreted as microns.

xi_min, xi_max

[float [arcsec], Quantity] Minimum and maximum slit position on the sky. If *None*, use the full range that spectral trace is defined on. Float values are interpreted as arcsec.

fov_grid()

Provide information on the source space volume required by the effect.

Returns

**A dictionary with entries *wave_min* and *wave_max*.
Spatial limits are determined by the *ApertureMask* effect
and are not returned here.**

map_spectra_to_focal_plane(fov)

Apply the spectral trace mapping to a spectral cube.

The cube is contained in a FieldOfView object, which also has world coordinate systems for the Source (sky coordinates and wavelengths) and for the focal plane. The method returns a section of the fov image along with info on where this image lies in the focal plane.

plot(*wave_min=None, wave_max=None, xi_min=None, xi_max=None, *, c='r', axes=None, plot_footprint=True, plot_wave=True, plot_ctrlpnts=True, plot_outline=False, plot_trace_id=False*)

Plot control points (and/or footprint) of the SpectralTrace.

Parameters

wave_min

[float, optional] Minimum wavelength, if any.

wave_max

[float, optional] Maximum wavelength, if any.

xi_min

[float, optional] Minimum slit, if any.

xi_max

[float, optional] Maximum slit, if any.

c

[str, optional] Colour, any valid matplotlib colour string. The default is “r”.

axes

[matplotlib axes, optional] The axes object to use for the plot. If None (default), a new figure with one axes will be created.

Returns

axes

[matplotlib axes] The axes object containing the plot.

Other Parameters

plot_footprint

[bool, optional] Plot a rectangle encompassing all control points, which may be larger than the area actually covered by the trace, if the trace is not exactly perpendicular to the detector. The default is True.

plot_wave

[bool, optional] Annotate the wavelength points. The default is True.

plot_ctrlpnts

[bool, optional] Plot the individual control points as makers. The default is True.

plot_outline

[bool, optional] Plot the smallest tetragon encompassing all control points. The default is False.

plot_trace_id

[bool, optional] Write the trace ID in the middle of the trace. The default is False.

rectify(*hdulist, interps=None, wcs=None, **kwargs*)

Create 2D spectrum for a trace.

Parameters

hdulist

[HDUList] The result of scopesim readout

interps

[list of interpolation functions] If provided, there must be one for each image extension in *hdulist*. The functions go from pixels to the images and can be created with, e.g., RectBi-variateSpline.

wcs

[The WCS describing the rectified XiLamImage. This can be created] in a simple way from the fov included in the *OpticalTrain* used in the simulation run producing *hdulist*.

The WCS can also be set up via the following keywords:

bin_width

[float [um]] The spectral bin width. This is best computed automatically from the spectral dispersion of the trace.

wave_min, wave_max

[float [um]] Limits of the wavelength range to extract. The default is the the full range on which the *SpectralTrace* is defined. This may extend significantly beyond the filter window.

xi_min, xi_max

[float [arcsec]] Spatial limits of the slit on the sky. This should be taken from the header of the *hdulist*, but this is not yet provided by scopesim

property trace_id

Return the name of the trace.

```
class scopesim.effects.spectral_trace_list_utils.Transform2D(matrix, pretransform_x=None,
                                                            pretransform_y=None,
                                                            posttransform=None)
```

Bases: `object`

2-dimensional polynomial transform.

The class is instantiated from a $m \times n$ matrix A that contains the coefficients of the polynomial. Along rows, the power of x increases; along columns, the power of y increases, such that $A[j, i]$ is the coefficient of $x^i y^j$.

The functions *pretransform_x* and *pretransform_y* can be used to transform the input variables before the matrix is applied. The function *posttransform* can be applied to the output after application of the matrix.

In Scopesim, a usecase for the pre- and post-transform functions is the METIS LMS, where the matrices are applied to phases while Scopesim operates on wavelengths. The functions to pass are *lam2phase* and *phase2lam*.

Parameters**matrix**

[np.array] matrix of polynomial coefficients

pretransform_x

[function, tuple]

pretransform_y

[function, tuple] If not None, the function is applied to the input variable x or y before the actual 2D transform is computed

posttransform

[function, tuple] If not None, the function is applied to the output variable after the 2D transform is computed

When passed as a tuple, the first element is the function itself, the second element is a dictionary of arguments to the function.

Example:

```
...
def rescale(x, scale=1.):
    return x * scale

pretransform_x = (rescale, {"scale": 0.5})
...
```

classmethod `fit(xin, yin, xout, degree=4)`

Determine polynomial fits.

gradient()

Compute the gradient of a 2d polynomial transformation.

class `scopesim.effects.spectral_trace_list_utils.XiLamImage(fov, dlam_per_pix)`

Bases: `object`

Class to compute a rectified 2D spectrum.

The class produces and holds an image of xi (relative position along the spatial slit direction) and wavelength lambda.

Parameters

fov

[FieldOfView]

dlam_per_pix

[a 1-D interpolation function from wavelength (in um) to] dispersion (in um/pixel); alternatively a number giving an average dispersion

`scopesim.effects.spectral_trace_list_utils.fill_zeros(x)`

Fill in zeros in a sequence with the previous non-zero number.

`scopesim.effects.spectral_trace_list_utils.fit2matrix(fit)`

Return coefficients from a polynomial fit as a matrix.

The Polynomial2D fits of degree n have coefficients for all i, j with i + j <= n. How would one rearrange those?

`scopesim.effects.spectral_trace_list_utils.get_affine_parameters(coords)`

Return rotation and shear for each MTC point along a SpectralTrace.

Parameters

coords

[dict of 2D arrays] Each dict entry ["x", "y", "s"] contains a [N, M] 2D array of coordinates, where:

- N is the number of points along the slit (e.g. ~5), and
- M is the number of positions along the trace (e.g. >100)

Returns

rotations

[array] [deg] Rotation angles for M positions along the Trace

shears

[array] [deg] Shear angles for M positions along the Trace

`scopesim.effects.spectral_trace_list_utils.make_image_interpolations(hdulist, **kwargs)`

Create 2D interpolation functions for images.

`scopesim.effects.spectral_trace_list_utils.rolling_median(x, n)`

Calculate the rolling median of a sequence for +/- n entries.

`scopesim.effects.spectral_trace_list_utils.xilam2xy_fit(layout, params)`

Determine polynomial fits of FPA position.

Fits are of degree 4 as a function of slit position and wavelength.

`scopesim.effects.spectral_trace_list_utils.xy2xilam_fit(layout, params)`

Determine polynomial fits of wavelength/slit position.

Fits are of degree 4 as a function of focal plane position

scopesim.effects.surface_list module

TBA.

class `scopesim.effects.surface_list.SurfaceList(**kwargs)`

Bases: [TERCurve](#)

add_surface(*surface*, *name=None*, *position=-1*, *add_to_table=True*)

add_surface_list(*surface_list*, *prepend=False*)

property `area`

property `emission`

fov_grid(*which='waveset'*, ***kwargs*)

Return the edges needed to generate FieldOfViews for an observation.

Parameters

which

[str] ["waveset", "edges", "shifts"] where: * waveset - wavelength bin extremes * edges - on sky coordinate edges for each FOV box * shifts - wavelength dependent FOV position offsets

Returns

waveset

[list] [um] N+1 wavelengths that set edges of N spectral bins

edges

[list of lists] [arcsec] Contains a list of footprint lists

shifts

[list of 3 lists] [wave, dx, dy] Contains lists corresponding to the (dx, dy) offset from the optical axis (0, 0) induced for each wavelength in (wave) [um, arcsec, arcsec]

get_emission(*etendue*, *start=0*, *end=None*, *rows=None*, *use_area=False*)

get_throughput(*start=0*, *end=None*, *rows=None*)

Copied directly from radiometry_table.

property `is_empty`

plot(*which='x'*, *wavelength=None*, ***, *axes=None*, ***kwargs*)

Plot TER curves.

Parameters

which

[{"x", "t", "e", "r"}, optional] "x" plots throughput. "t","e","r" plot trans/emission/refl. Can be a combination, e.g. "tr" or "tex" to plot each.

wavelength

[array_like, optional] Passed to `TERCurve.plot()` for each surface. The default is None.

axes

[matplotlib axes, optional] If given, plot into existing axes. The default is None.

Returns

fig

[matplotlib figure] Figure containing plots.

property surface

property throughput

scopesim.effects.ter_curves module

Transmission, emissivity, reflection curves.

class scopesim.effects.ter_curves.**ADCWheel**(**kwargs)

Bases: *Effect*

Wheel holding a selection of predefined atmospheric dispersion correctors.

apply_to(obj, **kwargs)

Use apply_to of current ADC.

change_adc(adcname=None)

Change the current ADC.

property current_adc

Return the currently used ADC.

get_table()

Create a table of ADCs with maximum throughput.

required_keys = {'adc_names', 'current_adc', 'filename_format'}

class scopesim.effects.ter_curves.**AtmosphericTERCurve**(**kwargs)

Bases: *TERCurve*

class scopesim.effects.ter_curves.**DownloadableFilterCurve**(**kwargs)

Bases: *FilterCurve*

class scopesim.effects.ter_curves.**FilterCurve**(cmds=None, **kwargs)

Bases: *TERCurve*

Description TBA.

Parameters

position

[int, optional]

filter_name

[str, optional] Ks - corresponding to the filter name in the filename pattern

filename_format

[str, optional] TC_filter_{}.dat

Can either be created using the standard 3 options:

- `filename`: direct filename of the filter curve
 - `table`: an `astropy.Table`
 - `array_dict`: a dictionary version of a table: `{col_name1: values, }`
- or by passing the combination of `filter_name` and `filename_format` as kwargs. Here all filter file names follow a pattern (e.g. see above) and the `{}` are replaced by `filter_name` at run time. `filter_name` can also be a `!bang` string for a `__currsys__` entry: `""!INST.filter_name""`

property center

property centre

fov_grid(*which='waveset', **kwargs*)

Return the edges needed to generate FieldOfViews for an observation.

Parameters

which

[str] ["waveset", "edges", "shifts"] where: * waveset - wavelength bin extremes * edges - on sky coordinate edges for each FOV box * shifts - wavelength dependent FOV position offsets

Returns

waveset

[list] [um] N+1 wavelengths that set edges of N spectral bins

edges

[list of lists] [arcsec] Contains a list of footprint lists

shifts

[list of 3 lists] [wave, dx, dy] Contains lists corresponding to the (dx, dy) offset from the optical axis (0, 0) induced for each wavelength in (wave) [um, arcsec, arcsec]

property fwhm

class scopesim.effects.ter_curves.**FilterWheel**(***kwargs*)

Bases: [FilterWheelBase](#)

Wheel holding a selection of predefined filters.

Examples

```
name: filter_wheel
class: FilterWheel
kwargs:
  filter_names: []
  filename_format: "filters/{}".
  current_filter: "Ks"
```

required_keys = {'current_filter', 'filename_format', 'filter_names'}

class scopesim.effects.ter_curves.**FilterWheelBase**(***kwargs*)

Bases: [Effect](#)

Base class for Filter Wheels.

add_filter(*newfilter*, *name=None*)

Add a filter to the FilterWheel.

Parameters

newfilter

[FilterCurve]

name

[string] Name to be used for the new filter. If *None* a name from the newfilter object is used.

apply_to(*obj*, ***kwargs*)

Use apply_to of current filter.

change_filter(*filtername=None*)

Change the current filter.

property current_filter

fov_grid(*which='waveset'*, ***kwargs*)

Return the edges needed to generate FieldOfViews for an observation.

Parameters

which

[str] ["waveset", "edges", "shifts"] where: * waveset - wavelength bin extremes * edges - on sky coordinate edges for each FOV box * shifts - wavelength dependent FOV position offsets

Returns

waveset

[list] [um] N+1 wavelengths that set edges of N spectral bins

edges

[list of lists] [arcsec] Contains a list of footprint lists

shifts

[list of 3 lists] [wave, dx, dy] Contains lists corresponding to the (dx, dy) offset from the optical axis (0, 0) induced for each wavelength in (wave) [um, arcsec, arcsec]

get_table()

plot(*which='x'*, *wavelength=None*, ***, *axes=None*, ***kwargs*)

Plot TER curves.

Parameters

which

[{"x", "t", "e", "r"}, optional] "x" plots throughput. "t","e","r" plot trans/emission/refl. Can be a combination, e.g. "tr" or "tex" to plot each.

wavelength

[array_like, optional] DESCRIPTION. The default is None.

axes

[matplotlib axes, optional] If given, plot into existing axes. The default is None.

Returns

fig

[matplotlib figure] Figure containing plots.

required_keys = {}

property surface

property throughput

class scopesim.effects.ter_curves.**PupilTransmission**(*transmission*, *cmds=None*, ***kwargs*)

Bases: [TERCurve](#)

Wavelength-independent transmission curve.

Use this class to describe a cold stop or pupil mask that is characterised by “grey” transmissivity. The emissivity is set to zero, assuming that the mask is cold.

update_transmission(*transmission*, ***kwargs*)

class scopesim.effects.ter_curves.**QuantumEfficiencyCurve**(***kwargs*)

Bases: [TERCurve](#)

class scopesim.effects.ter_curves.**SkycalcTERCurve**(***kwargs*)

Bases: [AtmosphericTERCurve](#)

Retrieve an atmospheric spectrum from ESO’s skycalc server.

Examples

```
- name : skycalc_background
  class : SkycalcTERCurve
  kwargs :
    wunit : "!SIM.spectral.wave_unit"
    wmin : "!SIM.spectral.wave_min"
    wmax : "!SIM.spectral.wave_max"
    wdelta : 0.0001      # 0.1nm bin width
    outer : 1
    outer_unit : "m"
```

property include

load_skycalc_table()

query_server(***kwargs*)

class scopesim.effects.ter_curves.**SpanishVOFilterCurve**(***kwargs*)

Bases: [FilterCurve](#)

Pulls a filter transmission curve down from the Spanish VO filter service.

Parameters

observatory
[str]

instrument
[str]

filter_name
[str]

Examples

```
name: HAWKI-Ks
class: SpanishVOFilterCurve
kwargs:
  observatory : Paranal
  instrument : HAWKI
  filter_name : Ks
```

```
class scopesim.effects.ter_curves.SpanishVOFilterWheel(**kwargs)
```

Bases: [FilterWheelBase](#)

A FilterWheel that loads all the filters from the Spanish VO service.

Warning: This use `astropy.download_file(..., cache=True)`.

The filter transmission curves probably won't change, but if you notice discrepancies, try clearing the astropy cache:

```
>> from astropy.utils.data import clear_download_cache
>> clear_download_cache()
```

Parameters

observatory

[str]

instrument

[str]

current_filter

[str] Default filter name

include_str, exclude_str

[str] String sequences that can be used to include or exclude filter names which contain a certain string. E.g. GTC/OSIRIS has curves for `sdss_g` and `sdss_g_filter`. We can force the inclusion of only the filter curves by setting `list_include_str: "_filter"`.

Examples

```
name: svo_filter_wheel
class: SpanishVOFilterWheel
kwargs:
  observatory: "GTC"
  instrument: "OSIRIS"
  current_filter: "sdss_r_filter"
  include_str: "_filter"
```

```
required_keys = {'current_filter', 'instrument', 'observatory'}
```

```
class scopesim.effects.ter_curves.TERCurve(**kwargs)
```

Bases: [Effect](#)

Transmission, Emissivity, Reflection Curve.

note:: This is basically an **Effect** wrapper for the `SpectralSurface` object

Must contain a wavelength column, and one or more of the following: `transmission`, `emissivity`, `reflection`. Additionally, in the header there should be the following keywords: `wavelength_unit`

kwargs that can be passed:

```
"rescale_emission" : { "filter_name": str, "value": float, "unit": str}
```

Examples

Directly inside a YAML file description:

```
name: bogus_surface
class: TERCurve
kwargs:
  array_dict:
    wavelength: [0.3, 3.0]
    transmission: [0.9, 0.9]
    emission: [1, 1]
  wavelength_unit: um
  emission_unit: ph s-1 m-2 um-1
  rescale_emission:
    filter_name: "Paranal/HAWK.Ks"
    value: 15.5
    unit: ABmag
```

Indirectly inside a YAML file:

```
name: some_curve
class: TERCurve
kwargs:
  filename: bogus_surface.dat
```

which references this ASCII file:

```
# name: bogus_surface
# wavelength_unit: um
wavelength transmission emissivity
0.3          0.9         0.1
3.0          0.9         0.1
```

apply_to(*obj*, ***kwargs*)

TBA.

property background_source

property emission

plot(*which='x'*, *wavelength=None*, ***, *axes=None*, ***kwargs*)

Plot TER curves.

Parameters

which

[{"x", "t", "e", "r"}, optional] "x" plots throughput. "t","e","r" plot trans/emission/refl. Can be a combination, e.g. "tr" or "tex" to plot each.

wavelength

[array_like, optional] Wavelength on x-axis, taken from currsys if None (default).

axes

[matplotlib axes, optional] If given, plot into existing axes. The default is None.

Returns**fig**

[matplotlib figure] Figure containing plots.

property throughput

class scopesim.effects.ter_curves.**TopHatFilterCurve**(cmds=None, **kwargs)

Bases: [FilterCurve](#)

A simple Top-Hat filter profile.

Parameters**transmission**

[float] [0..1] Peak transmission of filter

blue_cutoff, red_cutoff

[float] [um] Blue and Red cutoff wavelengths

wing_transmission

[float, optional] [0..1] Default 0. Wing transmission of filter outside the cutoff range

Examples

```
name: J_band_tophat
class: TopHatFilterCurve
kwargs:
  transmission : 0.9
  wing_transmission : 0.001
  blue_cutoff : 1.15
  red_cutoff : 1.35
```

class scopesim.effects.ter_curves.**TopHatFilterWheel**(**kwargs)

Bases: [FilterWheelBase](#)

A selection of top-hat filter curves as defined in the input lists.

Parameters**filter_names: list of string****transmissions: list of floats**

[0..1] Peak transmissions inside the cutoff limits

wing_transmissions: list of floats

[0..1] Wing transmissions outside the cutoff limits

blue_cutoffs: list of floats

[um]

red_cutoffs: list of floats

[um]

current_filter: str, optional

Name of current filter at initialisation. If no name is given, the first entry in *filter_names* is used by default.

Examples

```
name: top_hat_filter_wheel
class: TopHatFilterWheel
kwargs:
  filter_names: ["J", "H", "K"]
  transmissions: [0.9, 0.95, 0.85]
  wing_transmissions: [0., 0., 0.001]
  blue_cutoffs: [1.15, 1.45, 1.9]
  red_cutoffs: [1.35, 1.8, 2.4]
  current_filter: "K"
```

```
required_keys = {'blue_cutoffs', 'filter_names', 'red_cutoffs', 'transmissions',
                 'wing_transmissions'}
```

scopesim.effects.ter_curves_utils module

TBA.

scopesim.effects.ter_curves_utils.add_edge_zeros(tbl, wave_colname)

scopesim.effects.ter_curves_utils.apply_throughput_to_cube(cube, thru)

Apply throughput curve to a spectroscopic cube.

Parameters

cube

[ImageHDU] three-dimensional image, dimension 0 (in python convention) is the spectral dimension. WCS is required.

thru

[synphot.SpectralElement, synphot.SourceSpectrum]

Returns

cube

[ImageHDU, header unchanged, data multiplied with] wavelength-dependent throughput.

scopesim.effects.ter_curves_utils.combine_two_spectra(spec_a, spec_b, action, wave_min, wave_max)

Combine transmission and/or emission spectrum with a common waverange.

Spec_A is the source spectrum Spec_B is either the transmission or emission that should be applied

Parameters

spec_a

[synphot.SourceSpectrum]

spec_b

[synphot.SpectralElement, synphot.SourceSpectrum]

action: str
[“multiply”, “add”]
wave_min, wave_max
[quantity] [Angstrom]

Returns

new_source
[synphot.SourceSpectrum]

`scopesim.effects.ter_curves_utils.download_svo_filter(filter_name, return_style='synphot')`

Query the SVO service for the true transmittance for a given filter.

Copied 1 to 1 from tynt by Brett Morris

Parameters

filter_name
[str] Name of the filter as available on the spanish VO filter service e.g: Paranal/HAWKI.Ks

return_style
[str, optional] Defines the format the data is returned - synphot: synphot.SpectralElement -
table: astropy.table.Table - quantity: astropy.unit.Quantity [wave, trans] - array: np.ndarray
[wave, trans], where wave is in Angstrom - vo_table : astropy.table.Table - original output
from SVO service

Returns

filt_curve
[See return_style] Astronomical filter object.

`scopesim.effects.ter_curves_utils.download_svo_filter_list(observatory, instrument,
short_names=False, include=None,
exclude=None)`

Query the SVO service for a list of filter names for an instrument.

Parameters

observatory
[str] Name of the observatory as available on the spanish VO filter service e.g: Paranal/
HAWKI.Ks → Paranal

instrument
[str] Name of the instrument. Be careful of hyphens etc. E.g. “HAWK-I”

short_names
[bool] Default False. If True, the full SVO names (obs/inst.filt) are split to only return the
(filt) part of the name

include, exclude: str
Each a string sequence for excluding or including specific filters E.g. GTC/OSIRIS has
curves for `sdss_g` and `sdss_g_filter`. We can force the inclusion of only the filter curves
by setting `include="_filter"`.

Returns

names
[list] A list of filter names

`scopesim.effects.ter_curves_utils.get_filter(filter_name)`

`scopesim.effects.ter_curves_utils.get_filter_effective_wavelength(filter_name)`

`scopesim.effects.ter_curves_utils.get_zero_mag_spectrum(system_name='AB')`

`scopesim.effects.ter_curves_utils.scale_spectrum(spectrum, filter_name, amplitude)`

Scale a SourceSpectrum to a value in a filter.

Parameters

spectrum

[synphot.SourceSpectrum]

filter_name

[str] Name of a filter from - a local instrument package (available in `rc.__search_path__`)
- a generic filter name (see `ter_curves_utils.FILTER_DEFAULTS`) - a spanish-vo filter service reference (e.g. "Paranal/HAWKI.Ks")

amplitude

[astropy.Quantity, float] The value that the spectrum should have in the given filter. Acceptable astropy quantities are: - `u.mag` : Vega magnitudes - `u.ABmag` : AB magnitudes - `u.STmag` : HST magnitudes - `u.Jy` : Jansky per filter bandpass Additionally the FLAM and FNU units from `synphot.units` can be used when passing the quantity for amplitude:

Returns

spectrum

[synphot.SourceSpectrum] Input spectrum scaled to the given amplitude in the given filter

Examples

```
>>> from scopesim.source.source_templates import vega_spectrum
>>> from scopesim.effects.ter_curves_utils as ter_utils
>>>
>>> spec = vega_spectrum()
>>> vega_185 = ter_utils.scale_spectrum(spec, "Ks", -1.85 * u.mag)
>>> ab_0 = ter_utils.scale_spectrum(spec, "Ks", 0 * u.ABmag)
>>> jy_3630 = ter_utils.scale_spectrum(spec, "Ks", 3630 * u.Jy)
```

`scopesim.effects.ter_curves_utils.zero_mag_flux(filter_name, photometric_system, return_filter=False)`

Return the zero magnitude photon flux for a filter.

Acceptable filter names are those given in `scopesim.effects.ter_curves_utils.FILTER_DEFAULTS` or a string with an appropriate name for a filter in the Spanish-VO filter-service. Such strings must use the naming convention: `observatory/instrument.filter`. E.g: `paranal/HAWKI.Ks`, or `Gemini/GMOS-N.CaT`.

Parameters

filter_name

[str] Name of the filter - see above

photometric_system

[str] ["vega", "AB", "ST"] Name of the photometric system

return_filter

[bool, optional] If True, also returns the filter curve object

Returns

flux

[float] [PHOTLAM]

filt

[synphot.SpectralElement] If return_filter is True

Module contents

scopesim.optics package

Submodules

scopesim.optics.fov module

Defines FieldOfView class.

class scopesim.optics.fov.**FieldOfView**(header, waverange, detector_header=None, cmds=None, **kwargs)

Bases: *FieldOfViewBase*

A FOV is spectro-spatial volume cut out of a Source object.

Flux units after extracting the fields from the Source are in ph/s/pixel

The initial header should contain an on-sky WCS description: - CDELTA-, CUNIT-, NAXIS- : for pixel scale and size (assumed CUNIT in deg) - CRVAL-, CRPIX- : for positioning the final image - CTYPE- : is assumed to be "RA—TAN", "DEC—TAN"

and an image-plane WCS description - CDELTA-D, CUNIT-D, NAXISn : for pixel scale and size (assumed CUNIT in mm) - CRVAL-D, CRPIX-D : for positioning the final image - CTYPE-D : is assumed to be "LINEAR", "LINEAR"

The wavelength range is given by waverange

property background_fields

Return list of BG_SRC ImageHDU fields.

property corners

Return sky footprint, image plane footprint.

property cube_fields

Return list of non-BG_SRC ImageHDU fields with NAXIS=3.

property data

Return either hdu.data, image, cube, spectrum or None.

extract_from(src)

..assumption: Bandpass has been applied.

Note: Spectra are cut and copied from the original Source object. They are in original units. ph/s/pix comes in the make_**** methods

flatten()

If cube, collapse along first axis.

property image_fields

Return list of non-BG_SRC ImageHDU fields with NAXIS=2.

make_cube_hdu()

TBA.

Used for IFUs, slit spectrographs, and coherent MOSs (e.g. KMOS)

Returned cube units are $\text{ph s}^{-1} \text{ voxel}^{-1}$

Note: `self.make_cube()` does NOT store anything in `self.cube`

`self.cube` and `self.make_cube()` are deliberately kept separately so that `self.cube` will not be accidentally overwritten by a rogue call from an Effect object.

All Effect objects should specifically test whether `self.cube` is `None` before assigning a new cube it

The cube is made with these steps:

1. Make waveset and canvas cube:

```
if at least one cube:
    set waveset to equal largest cube waveset
else:
    make waveset from self.meta values
    make canvas cube based on waveset of largest cube and NAXIS1,2 from fov.
    ↳ header
```

2. Find Cube fields (see `FieldOfView._make_cube_cubefields()`).
3. Find Image fields (see `FieldOfView._make_cube_imagefields()`).
4. Find Table fields (see `FieldOfView._make_cube_tablefields()`).

PHOTLAM = $\text{ph} / (\text{s} * \text{m}^2 * \text{um})$. Original source fields are in units of:

- tables: (PHOTLAM in spectrum)
- images: arcsec^{-2} (PHOTLAM in spectrum)
- cubes: PHOTLAM arcsec^{-2}

Warning: Input Images and Cubes should have units of PHOTLAM arcsec^{-2}

Returns

canvas_cube_hdu

[fits.ImageHDU] [ph s⁻¹ AA⁻¹ arcsec^{-2}] # as needed by SpectralTrace

make_image_hdu(*use_photlam=False*)

TBA.

Used for imaging.

Output image units are $\text{ph s}^{-1} \text{ pixel}^{-1}$

Note: `self.make_image()` does NOT store anything in `self.image`

See `make_cube` for an explanation

Make canvas image from NAXIS1,2 from fov.header

Parameters**use_photlam**

[bool] Default False. Defines the flux units of the image pixels

Returns**image_hdu**

[fits.ImageHDU] [ph s-1 pixel-1] or PHOTLAM (if use_photlam=True)

make_spectrum()

TBA.

This is needed for when we do incoherent MOS instruments. Each fibre doesn't care about the spatial information.

Returns**spec**

[SourceSpectrum] [PHOTLAM]

property pixel_area**property table_fields**

Return list of Table fields.

property trace_id

Return the name of the trace.

view(hdu_type='image', sub_pixel=None, use_photlam=None)

Force the self.fields to be viewed as a single object.

Parameters**sub_pixel**

[bool]

hdu_type

[str] ["cube", "image", "spectrum"]

Returns**self.hdu**

[fits.ImageHDU, synphot.SourceSpectrum]

volume(wcs_prefix="")**property wavelength**

Return central wavelength in um.

property waverange

Return wavelength range in um [wave_min, wave_max].

property waveset

Return a wavelength vector in um.

scopesim.optics.fov_manager module

class scopesim.optics.fov_manager.**FOVManager**(*effects=None, cmds=None, **kwargs*)

Bases: `object`

A class to manage the (monochromatic) image windows covering the target.

Parameters**effects**

[list of Effect objects] Passed from optics_manager.fov_setup_effects

property fov_footprints

property fovs

generate_fovs_list()

Generate a series of FieldOfViews objects based self.effects.

Returns**fovs**

[list of FieldOfView objects]

class scopesim.optics.fov_manager.**FovVolumeList**(*initial_volume=None*)

Bases: `FOVSetupBase`, `MutableSequence`

List of FOV volumes for FOVManager.

extract(*axes, edges, aperture_id=None*)

Return new volumes from within all existing volumes.

This method DOES NOT alter the existing self.volumes list To include the returned volumes, add them to the self.volumes list

Parameters**axes**

[list of either {"wave", "x", "y"}] Which axis (list of single str) or axes (list[str]) to use. Must be list in either case.

edges

[list, tuple of lists] Edge points for each axes listed

aperture_id

[int, optional] Default None. If None, extract from all volumes. If int, only extract from volumes with this *aperture_id* in the meta dict

Returns**new_vols**

[list of dicts] A list of all new volumes extracted from existing volumes

Examples

::

```
>>> fvl = FovVolumeList()
>>> fvl.split("x", 0)
>>> new_vols = fvl.extract(axes=["wave"], edges=([0.5, 0.6], ))
>>> new_vols = fvl.extract(axes=["x", "y"], edges=([-1, 1], [0, 5]))
>>> new_vols = fvl.extract(axes=["x", "y", "wave"],
>>>                        edges=([-1, 1], [0, 5], [0.5, 0.6]))
>>> new_vols = fvl.extract(axes=["x", "y"], edges=([-1, 1], [0, 5]),
>>>                        aperture_id=1)
>>>
>>> fvl += [new_vols]
```

insert(*index*, *value*)

S.insert(*index*, *value*) – insert value before index

shrink(*axis*, *values*, *aperture_id*=None) → None

- Loop through all volume dict
- Replace any entries where min < values.min
- Replace any entries where max > values.max

Parameters

axis

[{"wave", "x", "y"} or list thereof] Which axis (str) or axes (list[str]) to use.

values

[list of 2 floats] [min, max], [min, None], [None, max]

aperture_id

[int, optional] Default None. If None, shrink all volumes. If int, only shrink volumes with this aperture_id in the meta dict

Examples

::

```
>>> fvl = FovVolumeList()
>>> fvl.shrink(axis="wave", values=[3.0, 3.1])
>>> fvl.shrink(axis="wave", values=[2.9, 3.1], aperture_id=1)
>>> fvl.shrink(axis=["x", "y"], values=([-1, 1], [0, 5]))
```

split(*axis*, *value*, *aperture_id*=None) → None

Split the all volumes that include axis=value into two.

- Loop through all volume dict
- Find any entries where min < value < max
- Add two new entries with [min, value], [value, max]

Parameters

axis

[{"wave", "x", "y"}, or list thereof] Which axis (`str`) or axes (`list[str]`) to use.

value

[float, list of floats]

aperture_id

[int, optional] Default None. If None, split all volumes. If int, only split volumes with this `aperture_id` in the meta dict

Examples

::

```
>>> fvl = FovVolumeList()
>>> fvl.split(axis="wave", value=1.5)
>>> fvl.split(axis="wave", value=[3.0, 3.1])
>>> fvl.split(axis=["x", "y"], value=[0, 0])
>>> fvl.split(axis=["x", "y"], value=[-1, 1], 0))
>>> fvl.split(axis=["x", "y"], value=[-1, 1], [0, 5]))
>>> fvl.split(axis="wave", value=3.0, aperture_id=1)
>>> fvl.split(axis="wave", value=3.0, aperture_id=None)
```

write_string(*stream: TextIO*) → None

Write formatted string representation to I/O stream.

scopesim.optics.fov_manager_utils module

`scopesim.optics.fov_manager_utils.combine_wavesets(*wavesets)`

Join and sorts several sets of wavelengths into a single 1D array.

Parameters**wavesets**

[one or more iterables] A group of wavelength arrays or lists

Returns**wave_set**

[np.ndarray] Combined set of wavelengths

`scopesim.optics.fov_manager_utils.get_3d_shifts(effects, **kwargs)`

Returns the total 3D shifts (x,y,lam) from a series of Shift3D objects

Parameters**effects**

[list of Shift3D effects]

Returns**shift_dict**

[dict] returns the x, y shifts for each wavelength in the `fov_grid`, where `fov_grid` contains the edge wavelengths for each spectral layer

Notes

Units returned by `fov_grid()`: - wavelength: [um] - x_shift, y_shift: [deg]

`scopesim.optics.fov_manager_utils.get_imaging_fovs(headers, waveset, shifts, **kwargs)`

Return a generator of `FieldOfView` objects.

Parameters

headers

[list of `fits.Header` objects] Headers giving spatial extent of each FOV region

waveset

[list of floats] [um] N+1 wavelengths for N spectral layers

shifts

[list of tuples (or actually arrays?)] [deg] x,y shifts w.r.t to the optical axis plane. N shifts for N spectral layers

Returns

fovs

[generator of `FieldOfView` objects]

`scopesim.optics.fov_manager_utils.get_imaging_headers(effects, **kwargs)`

Return a generator of `Header` objects for each of the `FieldOfView` objects.

Parameters

effects

[list of `Effect` objects] Should contain all effects which return a header defining the extent of their spatial coverage

Returns

hdrs

[generator of `Header` objects]

Notes

FOV headers use the return values from the `<Effect>.fov_grid()` method. The `fov_grid` dict must contain the entry `edges`

This may change in future versions of ScopeSim

`scopesim.optics.fov_manager_utils.get_imaging_waveset(effects_list, **kwargs)`

Returns the edge wavelengths for the spectral layers needed for simulation

Parameters

effects_list

[list of `Effect` objects]

Returns

wave_bin_edges

[list] [um] list of wavelengths

`scopesim.optics.fov_manager_utils.get_spectroscopy_fovs(headers, shifts, effects=None, **kwargs)`

Return a generator of `FieldOfView` objects.

`scopesim.optics.fov_manager_utils.get_spectroscopy_headers(effects, **kwargs)`

Return generator of Header objects.

scopesim.optics.fov_utils module

`scopesim.optics.fov_utils.combine_imagehdu_fields(fov_header, src, fields_indexes, wave_min,
wave_max, area, wcs_suffix="")`

Combine list of ImageHDUs into a single one bounded by the Header WCS.

Parameters

fov_header

[fits.Header] Header from the FieldOfView

src

[Source object]

fields_indexes

[list of ints] Which indexes from <Source>.fields to use

wave_min

[float] [deg] Blue spectral border

wave_max

[float] [deg] Red spectral border

area

[float] [m2] Area of the primary aperture

wcs_suffix

[str] Which coordinate system to use - "" for the on-sky coordinate system - "D" for the image-plane coordinate system

Returns

canvas_hdu

[fits.ImageHDU]

`scopesim.optics.fov_utils.combine_table_fields(fov_header, src, field_indexes)`

Combine list of Table objects into a single one bounded by the Header WCS.

Parameters

fov_header

[fits.Header] Header from a FieldOfView objects

src

[Source object]

field_indexes

[list of int]

Returns

tbl

[Table]

`scopesim.optics.fov_utils.extract_area_from_imagehdu(imagehdu, fov_volume)`

Extract the part of a ImageHDU that fits inside the *fov_volume*.

Parameters

imagehdu

[fits.ImageHDU] The field ImageHDU, either an image or a cube with wavelength [um]

fov_volume

[dict]

Contains {"xs": [xmin, xmax], "ys": [ymin, ymax],

"waves": [wave_min, wave_max], "xy_unit": "deg" or "mm", "wave_unit": "um"}

Returns**new_imagehdu**

[fits.ImageHDU]

scopesim.optics.fov_utils.**extract_area_from_table**(table, fov_volume)

Extract the entries of a Table that fits inside the fov_volume.

Parameters**table**

[fits.ImageHDU] The field ImageHDU, either an image of a wavelength [um] cube

fov_volume

[dict]

Contains {"xs": [xmin, xmax], "ys": [ymin, ymax],

"waves": [wave_min, wave_max], "xy_unit": "deg" or "mm", "wave_unit": "um"}

Returns**new_imagehdu**

[fits.ImageHDU]

scopesim.optics.fov_utils.**extract_common_field**(field, fov_volume)

Extract the overlapping parts of a field within a FOV volume.

Parameters**field**

[Table or ImageHDU]

fov_volume

[dict]

Contains {"xs": [xmin, xmax], "ys": [ymin, ymax],

"waves": [wave_min, wave_max], "xy_unit": "deg" or "mm", "wave_unit": "um"}

Returns**field_new**

[Table or ImageHDU]

scopesim.optics.fov_utils.**extract_range_from_spectrum**(spectrum, waverange)

scopesim.optics.fov_utils.**get_cube_waveset**(hdr, return_quantity=False)

scopesim.optics.fov_utils.**is_field_in_fov**(fov_header, field, wcs_suffix="")

Return True if Source.field footprint is inside the FieldOfView footprint.

Parameters**fov_header**

[fits.Header] Header from a FieldOfView object

field[[`astropy.Table`, `astropy.ImageHDU`]] Field object from a `Source` object**wcs_suffix**

[str] ["S", "D"] Coordinate system: Sky or Detector

Returns**is_inside_fov**

[bool]

`scopesim.optics.fov_utils.make_cube_from_table(table, spectra, waveset, fov_header, sub_pixel=False)`**Parameters****table:** `astropy.Table`**spectra:** dict**waveset:** `np.ndarray`**fov_header:** `fits.Header`**sub_pixel:** bool, optional**Returns****cube:** `fits.ImageHDU`Units of $\text{ph/s/m}^2/\text{bin}$ → should this be $\text{ph} / (\text{s} * \text{m}^2 * \text{um})$?`scopesim.optics.fov_utils.make_flux_table(source_tbl, src, wave_min, wave_max, area)``scopesim.optics.fov_utils.sky2fp(header, xsky, ysky)`

Convert sky coordinates to image plane coordinated.

Parameters**header**[Header] Header of a `FieldOfView` object which contains two sets of WCS keywords**xsky, ysky**

[float, array] [deg] The on-sky coordinated

Returns**xdet, ydet**

[float, array] [mm] The coordinated on the image plane

scopesim.optics.image_plane module`class scopesim.optics.image_plane.ImagePlane(header, **kwargs)`Bases: [`ImagePlaneBase`](#)A class to act as a canvas onto which to project *Source* images or tables.**Parameters****header**[`fits.Header`] Must contain a valid WCS**.. todo:** Write the code to deal with a canvas larger than `max_segment_size`

Examples

```
from astropy.table import Table
from scopesim.optics import image_plane as imp

my_point_source_table = Table(names=["x", "y", "flux"],
                              data=[(0, 1, 2)*u.mm,
                                     (0, -5, 10)*u.mm,
                                     (100, 50, 25)*u.ph/u.s])

hdr = imp.make_image_plane_header([my_point_source_table],
                                  pixel_size=0.015*u.mm)

img_plane = imp.ImagePlane(hdr)
img_plane.add(my_point_source_table)

print(img_plane.image)
```

add(*hdus_or_tables*, *sub_pixel*=None, *spline_order*=None, *wcs_suffix*="")

Add a projection of an image or table files to the canvas.

Note: If a Table is provided, it must include the following columns: *x_mm*, *y_mm*, and *flux*.

Units for the columns should be provided in the <Table>.unit attribute or as an entry in the table's meta dictionary using this syntax: <Table>.meta["<colname>_unit"] = <unit>.

For example:

```
tbl["x"].unit = u.arcsec # or
tbl.meta[x_unit] = "deg"
```

If no units are given, default units will be assumed. These are:

- *x, y*: *arcsec*
 - *flux*: *ph / s / pix*
-

Parameters

hdus_or_tables

[*fits.ImageHDU* or *astropy.Table*] The input to be projected onto the image plane. See above.

sub_pixel

[bool, optional] Default is False. Dictates if point files should be projected with sub-pixel shifts or not. Accounting for sub-pixel shifts is approx. 5x slower.

spline_order

[int, optional] Order of spline interpolations used in *scipy.ndimage* functions *zoom* and *rotate*.

wcs_suffix

[str, optional] Default "". For sky coords - "" or "S", Detector coords - "D"

property data

property header

property image

view(*sub_pixel*)

scopesim.optics.image_plane_utils module

`scopesim.optics.image_plane_utils.add_imagehdu_to_imagehdu(image_hdu: ImageHDU, canvas_hdu: ImageHDU, spline_order: int = 1, wcs_suffix: str = "", conserve_flux: bool = True)` → *ImageHDU*

Re-project one fits.ImageHDU onto another fits.ImageHDU.

..assumption:: of equal grid coordinate lengths

Parameters

image_hdu

[fits.ImageHDU] The ImageHDU which will be reprojected onto *canvas_hdu*

canvas_hdu

[fits.ImageHDU] The ImageHDU onto which the image_hdu should be projected. This must include a valid WCS

spline_order

[int, optional] Default is 1. The order of the spline interpolator used by the `scipy.ndimage` functions

wcs_suffix

[str or WCS] To determine which WCS to use. "" for sky HDUs and "D" for ImagePlane HDUs. Can also be `astropy.wcs.WCS` object.

conserve_flux

[bool] Default is True. Used when zooming and rotating to keep flux constant.

Returns

canvas_hdu

[fits.ImageHDU]

`scopesim.optics.image_plane_utils.add_table_to_imagehdu(table: Table, canvas_hdu: ImageHDU, sub_pixel: bool = True, wcs_suffix: str = "")` → *ImageHDU*

Add files from an `astropy.Table` to the image of an fits.ImageHDU.

Parameters

table

[`astropy.Table`] Must contain the columns "x_mm", "y_mm", "flux" with the units in the column attribute `.unit`, or in the table.meta dictionary as "<colname>_unit". Default units are mm and ph / s / pix

canvas_hdu

[fits.ImageHDU] The ImageHDU onto which the table files should be projected. This must include a valid WCS

sub_pixel

[bool, optional] Default is True. If True, sub-pixel shifts of files will be taken into account

when projecting onto the canvas pixel grid. This takes about 5x longer than ignoring the sub-pixel shifts

wcs_suffix
[str, optional]

Returns

canvas_hdu
[fits.ImageHDU]

`scopesim.optics.image_plane_utils.affine_map(input, matrix=None, rotation_angle: float = 0.0, shear_angle: float = 0.0, scale_factor=None, reshape: bool = True, spline_order: int = 3)`

Apply an affine transformation matrix to an image around its centre.

Similar functionality to `scipy.ndimage.rotate` but for the `affine_transformation` function

Either a 2x2 affine transformation matrix can be supplied, or the rotation, shear, and scaling values which are the basis of an affine transformation.

Parameters

input
[array_like] The input array

matrix
[ndarray, optional] A 2x2 affine transformation matrix

rotation_angle
[float, optional] [deg] If matrix==None, a rotation matrix is built from this angle

shear_angle
[float, optional] [deg] If matrix==None, a y-axis shear matrix is built from this angle

scale_factor
[list, array] [mx, my] If matrix==None, a scaling matrix is built from this list

reshape
[bool, optional] If True, the array is re-sized to contain the whole transformed image

spline_order
[int, optional] Default is 3. Spline interpolation order

Returns

output
[array-like] The new mapping of the image

`scopesim.optics.image_plane_utils.calc_footprint(header, wcs_suffix="", new_unit: str = None)`

Return the sky/detector positions [deg/mm] of the corners of a header WCS.

TODO: The rest of this docstring is outdated, please update!

The positions returned correspond to the corners of the header's image array, in this order:

```
(ra, dec) = (0,0), (w, 0), (w, h), (0, h)
(x, y) = (0,0), (w, 0), (w, h), (0, h)
```

where w, h are equal to NAXIS1 and NAXIS2 from the header.

Parameters

header

[fits.Header]

wcs_suffix

[str] Letter suffix for the WCS keywords, e.g. CDELT1D for image-plane coords

Returns**x, y**

[arrays of floats] [deg or mm] x are the coordinates for pixels [0, w, w, 0] [deg or mm] y are the coordinates for pixels [0, 0, h, h]

`scopesim.optics.image_plane_utils.calc_table_footprint`(*table*: *Table*, *x_name*: *str*, *y_name*: *str*,
tbl_unit: *str*, *new_unit*: *str*, *padding*=None)
→ ndarray

Equivalent to `calc_footprint()`, but for tables instead of images.

Parameters**table**

[astropy.table.Table] Table containing data.

x_name[str] Name of the column in *table* to use as x-coordinates.**y_name**[str] Name of the column in *table* to use as y-coordinates.**tbl_unit**[str] Default unit to use for x and y if no units are found in *table*.**new_unit**[str] Unit to convert x and y to, can be identical to *tbl_unit*.**padding**

[astropy.units.Quantity, optional] Constant value to subtract from minima and add to maxima. If used, must be Quantity with same physical type as x and y. If None (default), no padding is added.

Returns**extent**[(4, 2) array] Array containing corner points (clockwise from bottom left). Format and order are equivalent to the output of `astropy.wcs.WCS.calc_footprint()`.

`scopesim.optics.image_plane_utils.create_wcs_from_points`(*points*: ndarray, *pixel_scale*: float,
wcs_suffix: str = "") →
tuple[astropy.wcs.wcs.WCS,
numpy.ndarray]

Create *astropy.wcs.WCS* instance that fits all points inside.

Parameters**corners**

[(N, 2) array] 2D array of N >= 2 points in the form of [x, y].

pixel_scale

[float] DESCRIPTION.

wcs_suffix

[str, optional] DESCRIPTION. The default is "".

Returns

new_wcs
[TYPE] Newly created WCS instance.

naxis
[TYPE] Array of NAXIS needed to fit all points.

`scopesim.optics.image_plane_utils.det_wcs_from_sky_wcs(sky_wcs: WCS, pixel_scale: float, plate_scale: float, naxis=None) → tuple[astropy.wcs.wcs.WCS, numpy.ndarray]`

Create detector WCS from celestial WCS using pixel and plate scales.

Parameters

sky_wcs
[astropy.wcs.WCS] Celestial WCS.

pixel_scale
[float] Quantity or float (assumed to be arcsec / pixel).

plate_scale
[float] Quantity or float (assumed to be arcsec / mm).

naxis
[(int, int), optional] Shape of the image, usually NAXIS1 and NAXIS2. If the input WCS holds this information, the default None will use that. Otherwise not providing *naxis* will raise an error.

Returns

det_wcs
[astropy.wcs.WCS] Detector WCS.

det_naxis
[(int, int)] Shape of the image (NAXIS1, NAXIS2).

`scopesim.optics.image_plane_utils.get_canvas_header(hdu_or_table_list, pixel_scale=<Quantity 1. arcsec>)`

Generate a fits.Header with a WCS that covers everything in the FOV.

Parameters

hdu_or_table_list
[list] A list of Tables and/or ImageHDU py_objects

pixel_scale
[astropy.Quantity] [arcsec] The pixel scale of the projection. Default in 1 arcsec

Returns

header
[fits.Header] A Header containing a WCS and NAXISn values to build an ImageHDU

`scopesim.optics.image_plane_utils.header_from_list_of_xy(x, y, pixel_scale, wcs_suffix="", arcsec=False)`

Make a header large enough to contain all x,y on-sky coordinates.

Parameters

x, y
[list of floats] [deg, mm] List of sky coordinates to be bounded by the NAXISn keys

pixel_scale
[float] [deg, mm]

Returns**hdr**

[fits.Header]

`scopesim.optics.image_plane_utils.overlay_image(small_im, big_im, coords, mask=None, sub_pixel=False)`

Overlay *small_im* on top of *big_im* at the position specified by *coords*.

small_im will be centred at *coords*

Adapted from: <https://stackoverflow.com/questions/14063070/overlay-a-smaller-image-on-a-larger-image->

`scopesim.optics.image_plane_utils.pix2val(header, x, y, wcs_suffix="")`

Return the real coordinates [deg, mm] for coordinates from a Header WCS.

Parameters**header**

[fits.Header]

x, y

[float, list, array]

wcs_suffix

[str]

Returns**a, b**

[float, array] [deg, mm] Real coordinates as given by the Header WCS

`scopesim.optics.image_plane_utils.reorient_imagehdu(imagehdu: ImageHDU, wcs_suffix: str = "", conserve_flux: bool = True, spline_order: int = 1) → ImageHDU`

Apply an affine transformation to the image, as given in its header.

Parameters**imagehdu**

[fits.ImageHDU]

wcs_suffix

[str]

conserve_flux

[bool]

spline_order[int] [1..5] Order of the spline interpolation used by `scipy.ndimage.rotate`**Returns****imagehdu**

[fits.ImageHDU]

`scopesim.optics.image_plane_utils.rescale_imagehdu(imagehdu: ImageHDU, pixel_scale: float, wcs_suffix: str = "", conserve_flux: bool = True, spline_order: int = 1) → ImageHDU`

Scale the .data array by the ratio of *pixel_scale* [deg] and CDELTn.

pixel_scale should NOT be passed as a Quantity!

Parameters

imagehdu

[fits.ImageHDU]

pixel_scale

[float] [deg] NOT to be passed as a Quantity

wcs_suffix

[str]

conserve_flux

[bool]

spline_order

[int] [1..5] Order of the spline interpolation used by `scipy.ndimage.rotate`

Returns

imagehdu

[fits.ImageHDU]

`scopesim.optics.image_plane_utils.sky_wcs_from_det_wcs`(*det_wcs*: *WCS*, *pixel_scale*: *float*,
plate_scale: *float*, *naxis*=None) →
tuple[astropy.wcs.wcs.WCS, numpy.ndarray]

Create celestial WCS from detector WCS using pixel and plate scales.

Parameters

det_wcs

[astropy.wcs.WCS] Detector WCS.

pixel_scale

[float] Quantity or float (assumed to be arcsec / pixel).

plate_scale

[float] Quantity or float (assumed to be arcsec / mm).

naxis

[(int, int), optional] Shape of the image, usually NAXIS1 and NAXIS2. If the input WCS holds this information, the default None will use that. Otherwise not providing *naxis* will raise an error.

Returns

sky_wcs

[astropy.wcs.WCS] Celestial WCS.

sky_naxis

[(int, int)] Shape of the image (NAXIS1, NAXIS2).

`scopesim.optics.image_plane_utils.split_header`(*hdr*, *chunk_size*, *wcs_suffix*="")

Split a header into many smaller parts of the *chunk_size*.

Parameters

hdr

chunk_size

wcs_suffix

Returns

hdr_list

`scopesim.optics.image_plane_utils.sub_pixel_fractions(x, y)`

Make a list of pixel coordinates and weights to reflect sub-pixel shifts.

A point source which isn't centred on a pixel can be modelled by a centred PSF convolved with a shifted delta function. A fraction of the delta function is moved into each of the adjoining pixels. For example, a star at $(x, y) = (0.2, 0.2)$ would be represented by a following pixel weights:

```
-----
| 0.16 | 0.04 |
-----
| 0.64 | 0.16 |
-----
```

where (0,0) is the centre of the bottom-left pixel

Given (x,y) pixel coordinates, this function returns the fractions of flux that should go into the surrounding pixels, as well as the coordinates of those neighbouring pixels.

Parameters

x, y
[float]

Returns

x_pix, y_pix, fracs
[list of (int, int, float)] The x and y pixel coordinates and their corresponding flux fraction

`scopesim.optics.image_plane_utils.val2pix(header, a, b, wcs_suffix="")`

Return the pixel coordinates for real coordinates [deg, mm] from a WCS.

Parameters

header
[fits.Header]

a, b
[float, list, array] [deg, mm]

Returns

x, y
[float, array] [pixel] Pixel coordinates as given by the Header WCS

scopesim.optics.monochromatic_trace_curve module

`class scopesim.optics.monochromatic_trace_curve.MonochromeTraceCurve(x, y, s, wave_min, wave_max, **kwargs)`

Bases: `object`

Contains coordinates along a monochromatic section of a spectral trace.

Used to generate detector plane position WCS info for FOVs in spectral mode

get_header(*pixel_size*)

property header

scopesim.optics.optical_element module

class scopesim.optics.optical_element.**OpticalElement**(yaml_dict=None, cmds=None, **kwargs)

Bases: `object`

Contains all information to describe a section of an optical system.

There are 5 major section: location, telescope, relay optics instrument, detector.

An OpticalElement describes how a certain section of the optical train changes the incoming photon distribution by specifying a list of Effects along with a set of local properties e.g. temperature, etc which are common to more than one Effect

Parameters**yaml_dict**

[dict] Description of optical section properties, effects, and meta-data

kwargs

[dict] Optical Element specific information which has no connection to the effects that are passed. Any global values, e.g. airmass (i.e. bang strings) are passed on to the individual effect which can extract the relevant bang_string from the UserCommands object held in self.cmds

Attributes**meta**

[dict] Contains meta data from the yaml, and is updated with the OBS_DICT key-value pairs

properties

[dict] Contains any properties that is “global” to the optical element, e.g. instrument temperature, atmospheric pressure. Any OBS_DICT keywords are cleaned with from the meta dict during initialisation

effects

[list of dicts] Contains the a list of dict descriptions of the effects that the optical element generates. Any OBS_DICT keywords are cleaned with from the meta dict during initialisation.

add_effect(effect)

property display_name

get_all(effect_class)

get_z_order_effects(z_level: int, z_max: int = None)

Yield all effects in the given 100-range of z_level.

E.g., z_level=200 will yield all effect with a z_order between 200 and 299. Optionally, the upper limit can be set manually with the optional argument z_max.

Parameters**z_level**

[int] 100-range of z_orders.

z_max

[int, optional] Optional upper bound. This is currently not used anywhere in ScopeSim, but the functionality is tested. If None (default), this will be set to z_level + 99.

Yields

eff[Iterator of effects] Iterator containing all effect objects in the given `z_order` range.**Raises****TypeError**Raised if either `z_level` or `z_max` is not of int type.**ValueError**Raised if `z_max` (if given) is less than `z_level`.**list_effects()****property masks_list****pretty_str()** → `str`

Return formatted string representation as str.

property properties_str**report**(*filename=None, output='rst', rst_title_chars='^#+', **kwargs*)**property surfaces_list****write_string**(*stream: TextIO, list_effects: bool = True*) → `None`

Write formatted string representation to I/O stream.

scopesim.optics.optical_train module

class scopesim.optics.optical_train.**OpticalTrain**(*cmds=None*)Bases: `object`

The main class for controlling a simulation.

Parameters**cmds**

[UserCommands, str] If the name of an instrument is passed, OpticalTrain tries to find the instrument package, and internally creates the UserCommands object

Examples

Create an optical train:

```
>>> import scopesim as im
>>> cmd = sim.UserCommands("MICADO")
>>> opt = sim.OpticalTrain(cmd)
```

Observe a Source object:

```
>>> src = sim.source.source_templates.empty_sky()
>>> opt.observe(src)
>>> hdus = opt.readout()
```

List the effects modelled in an OpticalTrain:

```
>>> print(opt.effects)
```

Effects can be accessed by using the name of the effect:

```
>>> print(opt["dark_current"])
```

To include or exclude an effect during a simulation run, use the `.include` attribute of the effect:

```
>>> opt["dark_current"].include = False
```

Data used by an Effect object is contained in the `.data` attribute, while other information is contained in the `.meta` attribute:

```
>>> opt["dark_current"].data
>>> opt["dark_current"].meta
```

Meta data values can be set by either using the `.meta` attribute directly:

```
>>> opt["dark_current"].meta["value"] = 0.5
```

or by passing a dictionary (with one or multiple entries) to the OpticalTrain object:

```
>>> opt["dark_current"] = {"value": 0.75, "dit": 30}
```

property effects

load(*user_commands*)

(Re)Load an OpticalTrain with a new set of UserCommands.

Parameters

user_commands

[UserCommands or str]

observe(*orig_source*, *update=True*, ***kwargs*)

Main controlling method for observing Source objects.

Parameters

orig_source

[Source]

update

[bool] Reload optical system

kwargs

[expanded dict] Any keyword-value pairs from a config file

Notes

How the list of Effects is split between the 5 main tasks:

- Make a FOV list - `z_order = 0..99`
- Make a image plane - `z_order = 100..199`
- Apply Source altering effects - `z_order = 200..299`
- Apply FOV specific (3D) effects - `z_order = 300..399`
- Apply FOV-independent (2D) effects - `z_order = 400..499`
- [Apply detector plane (0D, 2D) effects - `z_order = 500..599`]

Todo: List is out of date - update

prepare_source(*source*)

Prepare source for observation.

The method is currently applied to cube fields only. The source data are converted to internally used units (PHOTLAM). The source data are interpolated to the waveset used by the FieldOfView This is necessary when the source data are sampled on a coarser grid than used internally, or if the source data are sampled on irregular wavelengths. For cube fields, the method assumes that the wavelengths at which the cube is sampled is provided explicitly as attribute *wave* if the cube ImageHDU.

readout(*filename=None, **kwargs*)

Produce detector readouts for the observed image.

Parameters

filename

[str, optional] Where to save the FITS file

kwargs

Returns

hdu

[fits.HDUList]

Notes

- Apply detector plane (0D, 2D) effects - `z_order = 500..599`

report()

shutdown()

Shut down the instrument.

This method closes all open file handles and should be called when the optical train is no longer needed.

update(***kwargs*)

Update the user-defined parameters and remake main internal classes.

Parameters

kwargs

[expanded dict] Any keyword-value pairs from a config file

write_header(*hdulist*)

Write meaningful header to simulation product.

`scopesim.optics.optical_train.apply_fov_effects(fov, fov_effects)`

`scopesim.optics.optical_train.extract_source(fov, source)`

`scopesim.optics.optical_train.view_fov(fov, hdu_type)`

scopesim.optics.optics_manager module

class `scopesim.optics.optics_manager.OpticsManager`(*yaml_dicts=None, cmds=None, **kwargs*)

Bases: `object`

The workhorse class for dealing with all externally defined Effect objects.

Parameters

yaml_dicts

[list of dict] The nested dicts describing the Effects from the relevant YAML files, which include `effects` and `properties` sub-dictionaries

kwargs

[expanded dict] Any extra information not directly related to the optical elements

add_effect(*effect, ext=0*)

Add an Effect object to an OpticalElement at index `ext`.

Parameters

effect

[Effect] Effect object to be added

ext

[int] Index number of the desired OpticalElement, contained in the list `self.optical_elements`

property `all_effects`

Get all effects in all optical elements.

property `area`

property `detector_array_effects`

Get effects with `z_order = 900...999`.

property `detector_effects`

Get effects with `z_order = 800...899`.

property `detector_setup_effects`

Get effects with `z_order = 400...499` (DetectorLists only!).

property `display_name`

property `fov_effects`

Get effects with `z_order = 600...699`.

property `fov_setup_effects`

Get effects with `z_order = 200...299`.

get_all(*class_type*)

Return list of all effects from all optical elements with *class_type*.

Parameters**class_type**

[class object] The class to be searched for. Must be an class object with base-class `Effect`

Returns**effects**

[list of Effect objects]

get_z_order_effects(*z_level: int*)

Return a list of all effects with a *z_order* keywords within *z_level*.

Effect *z_order* values are classified according to the following:

- Make a FOV list - *z_order* = 0..99
- Make a image plane - *z_order* = 100..199
- Apply Source altering effects - *z_order* = 200..299
- Apply FOV specific (3D) effects - *z_order* = 300..399
- Apply FOV-independent (2D) effects - *z_order* = 400..499
- Apply XXX effects - *z_order* = 500..599
- Apply XXX effects - *z_order* = 600..699
- Apply lambda-independent 2D image plane effects - *z_order* = 700..799
- Apply detector effects - *z_order* = 800..899
- Apply detector array effects - *z_order* = 900..999

Parameters**z_level**

[{0, 100, 200, 300, 400, 500, 600, 700, 800, 900}] 100-range of *z_orders*.

Returns**effects**

[list of Effect objects]

property image_plane_effects

Get effects with *z_order* = 700...799.

property image_plane_headers

Get headers from detector setup effects.

property image_plane_setup_effects

Get effects with *z_order* = 300...399.

property is_spectroscope

Return True if any of the effects is a spectroscope.

list_effects()

load_effects(*yaml_dicts*, ***kwargs*)

Generate an OpticalElement for each section of the Optical System.

Make an OpticalElement for each YAML document in the system. For example there should be a YAML document for each of the following:

- Atmosphere
- Telescope
- Relay optics
- Instrument
- Detector

The YAML files can each be separate .yaml files, or be contained in a single .yaml file separated by a yaml-document-separator: ``

— ``.

Parameters

yaml_dicts

[list of dicts] Each YAML dict should contain the descriptions of the Effects needed by each OpticalElement.

pretty_str() → *str*

Return formatted string representation as str.

report(*filename=None*, *output='rst'*, *rst_title_chars='_^#*+', **kwargs*)

set_derived_parameters()

property source_effects

Get effects with z_order = 500...599.

property surfaces_table

Get combined surface table from effects with z_order = 100...199.

property system_transmission

update(***obs_dict*)

Update the meta dictionary with keyword-value pairs.

Parameters

obs_dict

[expanded dict] Keyword-Value pairs to be added to self.meta

write_string(*stream: TextIO*) → *None*

Write formatted string representation to I/O stream

scopesim.optics.radiometry_utils module

scopesim.optics.radiometry_utils.add_surface_to_table(tbl, surf, name, position, silent=True)

scopesim.optics.radiometry_utils.combine_emissions(tbl, surfaces, row_indexes, etendue, use_area=False)

Combine thermal emission from a series of surfaces.

The function traces thermal emission through an optical system, taking into account the finite reflectivities/transmissivities and emissivities of the surfaces. The function assumes that etendue is conserved through the system, i.e. surfaces are neither over- nor undersized.

Parameters

tbl

[astropy Table] Required columns are *name* and *action* (reflection or transmission)

surfaces: OrderedDict of SpectralSurface

Keys are the names from tbl, values are of type *SpectralSurface*

row_indexes

[list of int] Rows of tbl (i.e. surfaces) to combine

etendue, use_area

[not needed (TODO: remove)]

Returns

SourceSpectrum

scopesim.optics.radiometry_utils.combine_tables(new_tables, old_table=None, prepend=False)

scopesim.optics.radiometry_utils.combine_throughputs(tbl, surfaces, rows_indexes)

scopesim.optics.radiometry_utils.string_to_table(tbl)

scopesim.optics.spectrograph module

scopesim.optics.surface module

class scopesim.optics.surface.PoorMansSurface(emission: Any, throughput: Any, meta: Any)

Bases: `object`

Solely used by SurfaceList.

emission: Any

meta: Any

throughput: Any

class scopesim.optics.surface.SpectralSurface(filename=None, cmds=None, **kwargs)

Bases: `object`

Initialised by a file containing one or more of the following columns: transmission, emissivity, reflection. The column wavelength must be given. Alternatively kwargs for the above mentioned quantities can be passed as arrays. If they are not Quantities, then a unit should also be passed with the <array_name>_unit syntax (i.e. emission_unit or wavelength_unit)

If temperature is not given as a Quantity, it defaults to degrees Celsius.

property area

property emission

Look for an emission array in self.meta.

If it doesn't find this, it defaults to creating a blackbody and multiplies this by the emissivity. Assumption is that self.meta["temperature"] is in deg_C, unless it is a u.Quantity with temperature unit attached. Return units are in PHOTLAM arcsec⁻², even though arcsec⁻² is not given.

property emissivity

from_meta(key, default_unit=None)

Convert a specific value in the meta dict to a Quantity.

Parameters

key

[str] Which key to pull from self.meta

default_unit

[str, Unit] In case self.meta doesn't contain a unit for the desired key

Returns

meta_quantity

[Quantity]

property mirror_angle

property reflection

property throughput

property transmission

property wavelength

scopesim.optics.surface_utils module

scopesim.optics.surface_utils.is_flux_binned(unit)

Check if the (flux) unit is a binned unit.

Parameters

unit

[Unit]

Returns

flag

[bool]

scopesim.optics.surface_utils.make_emission_from_array(flux, wave, meta)

Create an emission SourceSpectrum using an array.

Takes care of bins and solid angles. The solid_angle is kept in the returned SourceSpectrum meta dictionary under self.meta["solid_angle"].

Parameters

flux

[array-like, Quantity] if flux is not an array, the emission_unit must be in meta dict

wave
[array-like, Quantity] if flux is not an array, the `wavelength_unit` must be in meta dict

meta
[dict]

Returns

flux
[synphot.SourceSpectrum]

`scopesim.optics.surface_utils.make_emission_from_emissivity(temp, emiss_src_spec)`

Create an emission SourceSpectrum using blackbody and emissivity curves.

Parameters

temp
[float, Quantity] [Kelvin] If float, then must be in Kelvin

emiss_src_spec
[synphot.SpectralElement] An emissivity response curve in the range [0..1]

Returns

flux
[synphot.SourceSpectrum]

`scopesim.optics.surface_utils.normalise_binned_flux(flux, wave)`

Convert a binned flux Quantity array back into flux density.

The flux density normalising unit is taken from the wavelength Quantity unit.

Parameters

flux
[array-like Quantity] flux unit must include bin, e.g. `ph s-1 m-2 bin-1`

wave
[array-like Quantity]

Returns

flux
[array-like Quantity]

Module contents

scopesim.reports package

Submodules

scopesim.reports.report_generator module

`class scopesim.reports.report_generator.ReportGenerator(pkg_name)`

Bases: `object`

`rst_default_yaml()`

scopesim.reports.rst_utils module

```
scopesim.reports.rst_utils.latexify_rst_text(rst_text, filename=None, path=None, title_char='=',
                                              float_figures=True, use_code_box=True)
```

Converts an RST string (block of text) into a LaTeX string

NOTE: plots will NOT be generated with this command. For that we must invoke the `plotfiy` command.

Parameters

rst_text
[str]

filename
[str, optional] What to name the latex file. If None, the filename is derived from the text title

path
[str, optional] Where to save the latex file

title_char
[str, optional] The character used to underline the rst text title. Usually “=”.

float_figures
[bool, optional] Set to False if figures should not be placed by LaTeX. Replaces all `egin{figure}` with `egin{figure}[H]`

use_code_box
[bool, optional] Adds a box around quote blocks

Returns

tex_str
[str] The same string or block of text in LaTeX format

Examples

```
::
rst_text = “” Meaning of life =====
Apparently it’s 42. Lets plot
```

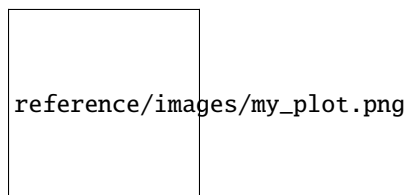


Fig. 1: This is an included figure caption

```
“”
plotfiy_rst_text(rst_text) latexify_rst_text(rst_text, filename="my_latex_file", path="/")
```

```
scopesim.reports.rst_utils.plotfiy_rst_text(rst_text)
```

Generates and saves plots from code blocks in an RST string

The save directory for the plots defaults to `scopesim.rc.__config__["!SIM.reports.image_path"]`. This can be overridden ONLY inside a COMMENT block using the path keyword.

Parameters**rst_text**

[str] Any RST text string

Notes

- Possible actions are: [reset, clear-figure, plot]
- Code is retained between code blocks in the same string, so we do not need to re-write large sections of code
- By default `import numpy as np` and `import matplotlib.pyplot as plt` are loaded automatically, so these do not need to be explicitly specified in each code block.
- THE EXCEPTION is when the action `reset` is specified. This clears the `code_context` variable.

Examples

The following rst text will generate a plot and save it in three formats:

```
..
```

```
!! processed by numpydoc !!
```

`scopesim.reports.rst_utils.process_code(context_code, code, options)`

Extracts and adds code from the node text to the `context_code` string

Code can be passed in either a `literal_block` or `comment` docutils node. The options regarding what to do with the code are included in either the `:class:` and `:name:` tag of a `literal_block` node, or in a yaml header section in a `comment` node.

See below for examples of code blocks.

Options for controlling what happens to the code block are as follows:

- **name:** The filename for the plot
- **format:** Any matplotlib-accepted file format, e.g: **png, pdf, svg, etc.**
Multiple file formats can be
- **action:** [reset, clear-figure, plot]
 - `reset` clears the `context_code` string. By default code is saved from previous code blocks.
 - `clear-figure` adds `plt.clf()` to the context string before the current code block is added
 - `plot` adds `plt.savefig({name}.{format})` to the context string. If multiple formats are given, these will be iterated over.

For `comment` blocks, options should be given in a yaml style header, separated from the code block by exactly three (3) hyphens ('—')

For `literal_block` blocks, we hijack the `:class:` and `:name:` attributes. See the examples below. All action keywords are passed to `:class:..` Format keys are passed as `format-<key>`.

Parameters**context_code**

[str] code from previous Nodes

code

[str] code from the current Node

options

[dict] options dictionary derived from Node attributes or RST header blocks

Returns**context_code**

[str]

Examples

Example of `literal_block` code block:

```
.. code::
   :name: my_fug
   :class: reset, clear-figure, plot, format-png

   plt.plot([0,1], [1,1])

.. figure:: my_fug.png
   :name: fig:my_fug
```

Example of a comment code block:

```
..
   name: my_fug2
   format: [jpg, svg]
   action: [reset, clear-figure, plot]
   ---
   plt.plot([0,1], [1,0])

.. figure:: my_fug2.jpg
   :name: fig:my_fug2
```

`scopesim.reports.rst_utils.process_comment_code(node, context_code)`

Add code from a `comment` node to the `context_code` string

`scopesim.reports.rst_utils.process_literal_code(node, context_code)`

Add code from a `literal_block` node to the `context_code` string

`scopesim.reports.rst_utils.rstify_rst_text(rst_text, filename=None, path=None, title_char='=')`

The same as `latexify_rst_text``, but the output is in RST format

`scopesim.reports.rst_utils.table_to_rst(tbl, indent=0, rounding=None)`

`scopesim.reports.rst_utils.walk(node, context_code=None)`

Recursively walk through a docutils doctree and run/plot code blocks

Parameters**node**

[docutils.node.Node]

context_code

[str, optional] A code string inherited from previous code/comment nodes

Returns**context_code**

[str] Code to be inherited by subsequent code/comment nodes

Module contents**scopesim.server package****Submodules****scopesim.server.database module**

Functions to download instrument packages and example data.

exception `scopesim.server.database.PkgNotFoundError`

Bases: `Exception`

Unable to find given package or given release of that package.

`scopesim.server.database.crawl_server_dirs(client=None) → Iterator[tuple[str, set[str]]]`

Search all folders on server for .zip files.

`scopesim.server.database.download_package(pkg_path, save_dir=None, url=None, from_cache=None)`

DEPRECATED – only kept for backwards compatibility

Downloads a package to the local disk

Parameters**pkg_path**

[str, list] A .zip package path as given by `list_packages()`

save_dir

[str] The place on the local disk where the .zip package is to be saved. If left as None, defaults to the value in `scopesim.rc.__config__["!SIM.file.local_packages_path"]`

url

[str] The URL of the IRDB HTTP server. If left as None, defaults to the value in `scopesim.rc.__config__["!SIM.file.server_base_url"]`

from_cache

[bool] Use the cached versions of the packages. If None, defaults to the RC value: `!SIM.file.use_cached_downloads`

Returns**save_path**

[str] The absolute path to the saved .zip package

`scopesim.server.database.download_packages(pkg_names: Union[Iterable[str], str], release: str = 'stable', save_dir: Optional[str] = None) → list[pathlib.Path]`

Download one or more packages to the local disk.

1. Download stable, dev
2. Download specific version
3. Download from github via url

Parameters**pkg_names**

[str, list] A list of package names, see `list_packages()`

release

[str, optional] By default, the most recent stable version of a package is downloaded. Other options are: - “stable” : the most recent stable version - “latest” : the latest development version to be published - a specific package filename as given by `list_packages` (see examples) - a github url for the specific branch and package (see examples)

save_dir

[str, optional] The place on the local disk where the .zip package is to be saved. If left as None, defaults to the value in `scopesim.rc.__config__` [“!SIM.file.local_packages_path”]

from_cache

[bool, optional] Use the cached versions of the packages. If None, defaults to the RC value: `!SIM.file.use_cached_downloads`

Returns**save_path**

[str] The absolute path to the saved .zip package

Examples

```
::
```

```
from scopesim import download_packages, list_packages

# Stable release of a list of packages download_packages(["test_package", "test_package"])

# Development release of a single package download_packages("test_package", release="latest")

# Specific version of the package list_packages("test_package") download_packages("test_package",
release="2022-04-09.dev")

# Specific package from a Github commit hash or branch/tag name (use "@" or ":") down-
load_packages("ELT", release="github:728761fc76adb548696205139e4e9a4260401dfc") down-
load_packages("ELT", release="github@728761fc76adb548696205139e4e9a4260401dfc") down-
load_packages("ELT", release="github@dev_master")
```

```
scopesim.server.database.get_all_latest(version_groups: Mapping[str, Iterable[str]]) →
Iterator[tuple[str, str]]
```

Yield the most recent stable (not “dev”) version of each package.

Parameters**version_groups**

[Mapping[str, Iterable[str]]] DESCRIPTION.

Yields

Iterator[tuple[str, str]]

Iterator of package name - latest version pairs.

```
scopesim.server.database.get_all_package_versions(client=None) → dict[str, list[str]]
```

Gather all versions for all packages present in any folder on server.

`scopesim.server.database.get_all_packages_on_server()` → `Iterator[tuple[str, set]]`

Retrieve all unique package names present on server in known folders.

Currently hardcoded to look in folders “locations”, “telescopes” and “instruments”. Any packages not in these folders are not returned.

This generator function yields key-value pairs, containing the folder name as the key and the set of unique package names in value. Recommended useage is to turn the generator into a dictionary, i.e.:

```
::
package_dict = dict(get_all_packages_on_server())
```

Yields

Iterator[tuple[str, set]]

Key-value pairs of folder and corresponding package names.

`scopesim.server.database.get_all_stable(version_groups: Mapping[str, Iterable[str]])` → `Iterator[tuple[str, str]]`

Yield the most recent version (stable or dev) of each package.

Parameters

version_groups

[Mapping[str, Iterable[str]]] DESCRIPTION.

Yields

Iterator[tuple[str, str]]

Iterator of package name - latest stable version pairs.

`scopesim.server.database.get_base_url()`

Get instrument package server URL from `rc.__config__`.

`scopesim.server.database.get_latest(versions: Iterable[str])` → `str`

Return the most recent version (stable or dev).

`scopesim.server.database.get_package_folders(client)` → `dict[str, str]`

Map package names to server locations.

`scopesim.server.database.get_server_folder_package_names(client, dir_name: str)` → `set[str]`

Retrieve all unique package names present on server in `dir_name` folder.

Parameters

client

[`httpx.Client`] Pre-existing `httpx` Client context manager.

dir_name

[`str`] Name of the folder on the server.

Returns

package_names

[set of `str`] Set of unique package names in `dir_name` folder.

Raises

ValueError

Raised if no valid packages are found in the given folder.

`scopesim.server.database.get_server_package_list()`

`scopesim.server.database.get_stable(versions: Iterable[str]) → str`

Return the most recent stable (not “dev”) version.

`scopesim.server.database.group_package_versions(all_packages: Iterable[tuple[str, str]]) → Iterator[tuple[str, list[str]]]`

Group different versions of packages by package name.

`scopesim.server.database.list_packages(pkg_name: Optional[str] = None) → list[str]`

List all packages, or all variants of a single package.

Parameters

pkg_name

[str, optional]

- None: lists all stable packages on the server
- <PackageName>: lists all variants of <PackageName> on the server

Returns

pkg_names

[list]

Examples

```
::
    from scopesim import list_packages
    # list all stable packages on the server list_packages()
    # list all variants of a specific package list_packages("Armazones")
```

scopesim.server.download_utils module

Used only by the *database* and *github_utils* submodules.

exception `scopesim.server.download_utils.ServerError`

Bases: `Exception`

Some error with the server or connection to the server.

`scopesim.server.download_utils.create_client(base_url, cached: bool = False, cache_name: str = "")`

Create httpx Client instance, should support cache at some point.

`scopesim.server.download_utils.get_server_folder_contents(client, dir_name: str, unique_str: str = '.zip$') → Iterator[str]`

Find all zip files in a given server folder.

`scopesim.server.download_utils.handle_download(client, pkg_url: str, save_path: Path, pkg_name: str, padlen: int, chunk_size: int = 128, disable_bar=False) → None`

Perform a streamed download and write the content to disk.

`scopesim.server.download_utils.handle_unzipping(save_path: Path, save_dir: Path, pkg_name: str, padlen: int) → None`

Unpack a zipped folder, usually called right after downloading.

`scopesim.server.download_utils.send_get(client, sub_url, stream: bool = False)`

Send a GET request (streamed or not) using an existing client.

The point of this function is mostly elaborate exception handling.

scopesim.server.example_data_utils module

Store the example data functions here instead of polluting database.py.

`scopesim.server.example_data_utils.download_example_data(file_path: Union[Iterable[str], str], save_dir: Optional[Union[Path, str]] = None, url: Optional[str] = None, from_cache: Optional[bool] = None) → list[pathlib.Path]`

Download example fits files to the local disk.

Parameters

file_path

[str, list] Name(s) of FITS file(s) as given by `list_example_data()`

save_dir

[str] The place on the local disk where the downloaded files are to be saved. If left as None, defaults to the current working directory.

url

[str] The URL of the database HTTP server. If left as None, defaults to the value in `scopesim.rc.__config__["!SIM.file.server_base_url"]`

from_cache

[bool] Use the cached versions of the files. If None, defaults to the RC value: `!SIM.file.use_cached_downloads`

Returns

save_path

[Path or list of Paths] The absolute path(s) to the saved files

`scopesim.server.example_data_utils.get_server_elements(url: str, unique_str: str = '/') → list[str]`

Return a list of file and/or directory paths on the HTTP server `url`.

Parameters

url

[str] The URL of the IRDB HTTP server.

unique_str

[str, list] A unique string to look for in the beautiful HTML soup: “/” for directories this, “.zip” for packages

Returns

paths

[list] List of paths containing in `url` which contain `unique_str`

`scopesim.server.example_data_utils.list_example_data(url: Optional[str] = None, return_files: bool = False, silent: bool = False) → list[str]`

List all example files found under url.

Parameters

url

[str] The URL of the database HTTP server. If left as None, defaults to the value in `scopesim.rc.__config__["!SIM.file.server_base_url"]`

return_files

[bool] If True, returns a list of file names

silent

[bool] If True, does not print the list of file names

Returns

all_files

[list of str] A list of paths to the example files relative to url. The full string should be passed to `download_example_data`.

scopesim.server.github_utils module

Used only by the *database* submodule.

Original comment for these functions:

2022-04-10 (KL) Code taken directly from <https://github.com/sdushantha/gitdir> Adapted for ScopeSim usage.
Many thanks to the authors!

`scopesim.server.github_utils.create_github_url(url: str) → None`

From the given url, produce a URL compatible with Github's REST API.

Can handle blob or tree paths.

`scopesim.server.github_utils.download_github_folder(repo_url: str, output_dir: Union[Path, str] = '/') → None`

Download the files and directories in repo_url.

Re-written based on the on the download function [here](#)

Module contents

scopesim.source package

Submodules

scopesim.source.source module

old functionality to implement: # - provide x, y, lam, spectra, weight, ref # - overridden + : number, Source, SourceSpectrum # - overridden * : number, SpectralElement # - write to and read from file # - shift all fields # - rotate around the centre # - photons_in_range returns the photons per spectrum in a wavelength range # - image_in_range returns an image of the source for a wavelength range # # old functionality which will be removed: # - project_onto_chip # - apply_optical_train # # old structure → new structure: # - all data held in 6 arrays # → new dicts for fields, spectrum # field can be a Table or an ImageHDU # spectrum is a SourceSpectrum # # Use cases: # image + spectrum # images + spectra # table + spectrum # table + spectra # # table columns = x, y, spec_id, weight # table meta keywords = x_unit,

y_unit # # image header keywords = WCS, SPEC_ID, WEIGHT # [WCS = CRPIXn, CRVALn = (0,0), CTYPEn, CDn_m, NAXISn, CUNITn

```
class scopesim.source.source.Source(filename=None, cube=None, ext=0, lam=None, spectra=None,
                                     x=None, y=None, ref=None, weight=None, table=None,
                                     image_hdu=None, flux=None, **kwargs)
```

Bases: [SourceBase](#)

Create a source object from a file or from arrays

A Source object must consist of a spatial and a spectral description of the on-sky source. Many sources can be added together and kept in memory as a single Source object.

The spatial descriptions are kept in the <Source>.fields list, while the spectral descriptions are in the <Source>.spectra list.

The spatial description can be built from any combination of:

- a list of arrays (like in SimCADO >v0.5)
- astropy Table objects
- astropy ImageHDU objects
- on disk FITS files
- on disk ASCII tables

The spectral descriptions can be passed as either synphot.SourceSpectrum objects, or a set of two equal length arrays for wavelength and flux.

Hint: Initialisation parameter combinations include:

New ScopeSim-style input - table=<astropy.Table>, spectra=<list of synphot.SourceSpectrum>
 - table=<astropy.Table>, lam=<array>, spectra=<list of array> - image_hdu=<fits.
 ImageHDU>, spectra=<list of synphot.SourceSpectrum> - image_hdu=<fits.ImageHDU>,
 lam=<array>, spectra=<list of array> - image_hdu=<fits.ImageHDU>, flux=<astropy.
 Quantity>

Old SimCADO-style input - x=<array>, y=<array>, ref=<array>, spectra=<list of synphot.
 SourceSpectrum> - x=<array>, y=<array>, ref=<array>, spectra=<list of array>,
 lam=<array> - x=<array>, y=<array>, ref=<array>, weight=<array>, spectra=<list of
 array>, lam=<array>

More details on the content of these combinations can be found in the use-case documentation.

Parameters

filename

[str]

lam

[np.array] [um] Wavelength bins of length (m)

spectra

[list of synphot.SourceSpectra] [ph/s/cm2/AA]

x, y

[np.array] [arcsec] coordinates of where the emitting files are relative to the centre of the field of view

ref

[np.array] the index for .spectra which connects a position (x, y) to a spectrum `flux(x[i], y[i]) = spectra[ref[i]] * weight[i]`

weight

[np.array] A weighting to scale the relevant spectrum for each position

table

[astropy.Table]

image_hdu

[fits.ImageHDU] [arcsec-2] The .data array is simply a map of weights for the associated spectrum referenced by .header["SPEC_REF"]. Surface brightness values are assumed to be per arcsec2

flux

[astropy.Quantity] [u.mag, u.ABmag, u.Jy] Flux values are converted to a reference spectrum that is referenced by image_hdu.header["SPEC_REF"]. flux can only be used in conjunction with image_hdu

See also:

synphot

<https://synphot.readthedocs.io/en/latest/>

Attributes**fields**

[list] The spatial distribution of the on-sky source, either as fits.ImageHDU or astropy.Table objects

spectra

[list of synphot.SourceSpectrum objects] List of spectra associated with the fields

meta

[dict] Dictionary of extra information about the source

add_bandpass(*bandpass*)

append(*source_to_add*)

property cube_fields

List of fields that are defined through three-dimensional cubes

dump(*filename*)

Save to filename as a pickle

fluxes(*wave_min*, *wave_max*, ***kwargs*)

image(*wave_min*, *wave_max*, ***kwargs*)

property image_fields

List of fields that are defined through two-dimensional images

image_in_range(*wave_min*, *wave_max*, *pixel_scale*=<Quantity 1. arcsec>, *layers*=None, *area*=None, *spline_order*=1, *sub_pixel*=False)

classmethod load(*filename*)

Load :class:'.Source' object from filename

make_copy()

photons_in_range(*wave_min*, *wave_max*, *area=None*, *indexes=None*)

Parameters

wave_min

[float, u.Quantity] [um]

wave_max

[float, u.Quantity] [um]

area

[float, u.Quantity, optional] [m2]

indexes

[list of integers, optional]

Returns

counts

[u.Quantity list] [ph / s / m2] if area is None [ph / s] if area is passed

plot()

Plot the location of source components

Source components instantiated from 2d or 3d ImageHDUs are represented by their spatial footprint. Source components instantiated from tables are shown as points.

rotate(*angle*, *offset=None*, *layers=None*)

shift(*dx=0*, *dy=0*, *layers=None*)

Shifts the position of one or more fields w.r.t. the optical axis

Parameters

dx, dy

[float] [arcsec]

layers

[list of ints] which .fields entries to shift

property table_fields

List of fields that are defined through tables

scopesim.source.source_templates module

scopesim.source.source_templates.**ab_spectrum**(*mag=0*)

scopesim.source.source_templates.**empty_sky**(*flux=0*)

Returns an empty source so that instrumental fluxes can be simulated

Returns

sky

[Source]

scopesim.source.source_templates.**st_spectrum**(*mag=0*)

`scopesim.source.source_templates.star(x=0, y=0, flux=0)`

Source object for a single star in either vega, AB magnitudes, or Jansky

The star is associated with the reference spectrum for each photometric system, therefore a reference wavelength or filter does not need to be given

Parameters

x, y
[float] [arcsec] position from centre of field of view

flux
[float] [vega mag, AB mag, Jy] Stellar brightness

Returns

src
[Source] A source object with a single entry table field and a reference spectrum

`scopesim.source.source_templates.star_field(n, mmin, mmax, width, height=None, use_grid=False)`

Creates a super basic field of stars with random positions and brightnesses

Parameters

n
[int] number of stars

mmin, mmax
[float, astropy.Quantity] [mag, ABmag, Jy] min and max magnitudes/fluxes of the population stars. If floats, then assumed Quantity is vega magnitudes

width
[float] [arcsec] width of region to put stars in

height
[float, optional] [arcsec] if None, then height=width

use_grid
[bool, optional] Place stars randomly or on a grid

Returns

stars
[scopesim.Source object] A Source object with a field of stars that can be fed into the method: `OpticalTrain.observe()`

See also:

OpticalTrain.observe

OpticalTrain.readout

`scopesim.source.source_templates.uniform_illumination(xs, ys, pixel_scale, flux=None, spectrum=None)`

Return a Source for a uniformly illuminated area

Parameters

xs, ys
[list of float] [arcsec] min and max extent of each dimension relative to FOV centre E.g. `xs=[-1, 1], ys=[5, 5.5]`

pixel_scale
[float] [arcsec]

flux

[astropy.Quantity] [mag, ABMag, Jy] Flux per arcsecond of the Source

Returns**src**

[scopesim.Source]

Examples

A 200x200 uniform illumination Source at 1 Jy/arcsec²

```
src = uniform_illumination(xs=[-1,1], ys=[-1, 1],
                           pixel_scale=0.01, flux=1*u.Jy)
```

A source that extends just past the MICADO 15" slit dimensions with a flux of 10 mag/arcsec²

```
src = uniform_illumination(xs=[-8, 8], ys=[-0.03, 0.03],
                           pixel_scale=0.004, flux=10*u.mag)
```

Using a self made frequency-comb spectrum with 1 Jy lines ever 0.1μm

```
import numpy as np
from astropy import units as u
from synphot import SourceSpectrum, Empirical1D

wave = np.arange(0.7, 2.5, 0.001) * u.um
flux = np.zeros(len(wave))
flux[::100] = 1 * u.Jy
spec = SourceSpectrum(Empirical1D, points=wave, lookup_table=flux)

src = uniform_illumination(xs=[-8, 8], ys=[-0.03, 0.03],
                           pixel_scale=0.004, spectrum=spec)
```

scopesim.source.source_templates.**uniform_source**(*sp=None, extent=60*)

Simplified form of scopesim_templates.misc.uniform_source, mostly intended for testing

This function creates an image with extend² pixels with pixel size of 1 arcsec² so provided amplitudes are in flux or magnitudes per arcsec²

It accepts any synphot.SourceSpectrum compatible object

sp

[synphot.SourceSpectrum] defaults to vega_spectrum() with magnitude 0 mag/arcsec²

extent

[int, default 60] extension of the field in arcsec, will always produce a square field. Default value produces a field of 60x60 arcsec

scopesim.source.source_templates.**vega_spectrum**(*mag=0*)

scopesim.source.source_utils module

scopesim.source.source_utils.**convert_to_list_of_spectra**(spectra, lam)

scopesim.source.source_utils.**make_imagehdu_from_table**(x, y, flux, pix_scale=<Quantity 1. arcsec>)

scopesim.source.source_utils.**make_img_wcs_header**(pixel_scale, image_size)

Create a WCS header for an image

pixel_scale

[float] arcsecs

image_size

[tuple] x, y where x, y are integers

scopesim.source.source_utils.**photons_in_range**(spectra, wave_min, wave_max, area=None, bandpass=None)

Parameters

spectra

wave_min

[um]

wave_max

[um]

area

[Quantity] [m2]

bandpass

[SpectralElement]

Returns

counts

[u.Quantity array]

scopesim.source.source_utils.**scale_imagehdu**(imagehdu, waverange, area=None)

scopesim.source.source_utils.**validate_source_input**(**kwargs)

Module contents

9.1.2 Submodules

scopesim.base_classes module

class scopesim.base_classes.DetectorBase

Bases: `object`

class scopesim.base_classes.FOVSetupBase

Bases: `object`

class scopesim.base_classes.FieldOfViewBase

Bases: `object`

```
class scopesim.base_classes.ImagePlaneBase
    Bases: object

class scopesim.base_classes.PoorMansHeader(dic=None)
    Bases: object
    as_header()
    items()
    keys()
    update(obj)
    values()
        Like dict.values().

class scopesim.base_classes.SourceBase
    Bases: object
```

scopesim.rc module

Global configurations for ScopeSim.

scopesim.utils module

Helper functions for ScopeSim.

`scopesim.utils.add_keyword(filename, keyword, value, comment="", ext=0)`

Add a keyword, value pair to an extension header in a FITS file.

Parameters

filename

[str] Name of the FITS file to add the keyword to

keyword

[str]

value

[str, float, int]

comment

[str]

ext

[int, optional] The fits extension index where the keyword should be added. Default is 0

`scopesim.utils.add_mags(mags)`

Return a combined magnitude for a group of py_objects with *mags*.

`scopesim.utils.airmass2zendist(airmass)`

Convert airmass to zenith distance.

Parameters

airmass

[float (≥ 1)]

Returns**zenith distance in degrees**`scopesim.utils.airmass_to_zenith_dist(airmass)`

Return zenith distance in degrees.

 $Z = \arccos(1/X)$ `scopesim.utils.angle_in_arcseconds(distance, width)`

Return the angular distance of an object in arcseconds.

Units must be consistent!

`scopesim.utils.bug_report()` → `None`

Print versions of dependencies for inclusion in bug report.

`scopesim.utils.bug_report_to_file(filename)` → `None`Like `bug_report`, but writes to file instead of printing.`scopesim.utils.change_table_entry(tbl, col_name, new_val, old_val=None, position=None)``scopesim.utils.check_keys(input_dict, required_keys, action='error', all_any='all')`

Check to see if all/any of the required keys are present in a dict.

`scopesim.utils.clean_dict(orig_dict, new_entries)`

Replace OBS_DICT keywords with actual values.

Parameters**orig_dict**

[dict]

new_entries

[dict] OBS dict

Returns**orig_dict**

[dict] Updated dict

`scopesim.utils.close_loop(*close_loop(zip(x, y)))``scopesim.utils.convert_table_comments_to_dict(tbl)``scopesim.utils.deg2rad(theta)``scopesim.utils.deriv_polynomial2d(poly)`

Derive (gradient) of a Polynomial2D model.

Parameters**poly**

[astropy.modeling.models.Polynomial2D]

Returns**gradient**

[tuple of Polynomial2d]

`scopesim.utils.dist_mod_from_distance(d)`Use $\mu = 5 * \log_{10}(d) - 5$ formula.

`scopesim.utils.distance_from_dist_mod(mu)`

Use $d = 10^{**}(1 + \mu / 5)$ formula.

`scopesim.utils.empty_type(x)`

`scopesim.utils.extract_base_from_unit(unit, base_unit)`

Extract astropy base unit from a compound unit.

Parameters

unit

[astropy.Unit]

base_unit

[Unit, str]

Returns

new_unit

[Unit] The input unit minus any base units corresponding to *base_unit*.

extracted_units

[Unit] Any base units corresponding to *base_unit*.

`scopesim.utils.extract_type_from_unit(unit, unit_type)`

Extract astropy physical type from a compound unit.

Parameters

unit

[astropy.Unit]

unit_type

[str] The physical type of the unit as given by astropy

Returns

new_unit

[Unit] The input unit minus any base units corresponding to *unit_type*.

extracted_units

[Unit] Any base units corresponding to *unit_type*.

`scopesim.utils.figure_factory(nrows=1, ncols=1, **kwargs)`

Default way to init fig and ax, to easily modify later.

`scopesim.utils.figure_grid_factory(nrows=1, ncols=1, **kwargs)`

Gridspec variant.

`scopesim.utils.find_file(filename, path=None, silent=False)`

Find a file in search path.

Parameters

filename

[str] name of a file to look for

path

[list] list of directories to search (default: ['.'])

silent

[bool] if True, remain silent when file is not found

Returns**Absolute path of the file**

`scopesim.utils.from_currsys(item, cmds=None)`

Return the current value of a bang-string from `rc.__currsys__`.

`scopesim.utils.from_rc_config(item)`

`scopesim.utils.get_fits_type(filename)`

`scopesim.utils.get_meta_quantity(meta_dict, name, fallback_unit="")`

Extract a Quantity from a dictionary.

Parameters

meta_dict

[dict]

name

[str]

fallback_unit

[Quantity]

Returns

quant

[Quantity]

`scopesim.utils.has_needed_keywords(header, suffix="")`

Check to see if the WCS keywords are in the header.

`scopesim.utils.insert_into_orderreddict(dic, new_entry, pos)`

`scopesim.utils.interp2(x_new, x_orig, y_orig)`

Check and correct for decreasing x_orig values.

`scopesim.utils.is_fits(filename) → bool`

`scopesim.utils.log_bug_report(level=10) → None`

Emit bug report as logging message.

`scopesim.utils.log_to_file(enable=True)`

Enable or disable logging to file (convenience function).

`scopesim.utils.moffat(r, alpha, beta)`

!!Unfinished!! Return a Moffat function.

Parameters

r

alpha

beta

Returns

eta

`scopesim.utils.nearest(arr, val)`

Return the index of the value from *arr* which is closest to *val*.

Parameters

arr
[np.ndarray, list, tuple] Array to be searched

val
[float, int] Value to find in *arr*

Returns

i
[int] index of array where the nearest value to *val* is

`scopesim.utils.parallactic_angle(ha, de, lat=-24.589167)`

Compute the parallactic angle.

Parameters

ha
[float] [hours] hour angle of target point

de
[float] [deg] declination of target point

lat
[float] [deg] latitude of observatory, defaults to Armazones

Returns

parang
[float] The parallactic angle

Notes

The parallactic angle is defined as the angle PTZ, where P is the .. $\tan \eta = \frac{\cos \phi \sin H}{\sin \phi \cos \delta - \cos \phi \sin \delta \cos H}$ It is negative (positive) if target point is east (west) of the meridian.

References

R. Ball: "A Treatise on Spherical Astronomy", Cambridge 1908

`scopesim.utils.poissonify(arr)`

Add a realisation of the poisson process to the array *arr*.

Parameters

arr
[np.ndarray] The input array which needs a Poisson distribution applied to items

Returns

arr
[np.ndarray] The input array, but with every pixel altered according to a poisson distribution

`scopesim.utils.power_vector(val, degree)`

Return the vector of powers of *val* up to a degree.

`scopesim.utils.pretty_print_dict(dic, indent=0)`

`scopesim.utils.quantify(item, unit, cmds=None)`

Ensure an item is a Quantity.

Parameters

item

[int, float, array, list, Quantity]

unit

[str, Unit]

Returns

quant

[Quantity]

`scopesim.utils.quantity_from_table(colname: str, table: Table, default_unit: str = "")` → Quantity

`scopesim.utils.rad2deg(theta)`

`scopesim.utils.real_colname(name, colnames, silent=True)`

`scopesim.utils.return_latest_github_actions_jobs_status(owner_name='AstarVienna',
repo_name='ScopeSim',
branch='dev_master',
actions_yaml_name='tests.yml')`

Get the status of the latest test run.

`scopesim.utils.seq(start, stop, step=1)`

Replacement for numpy.arange modelled after R's seq function.

Returns an evenly spaced sequence from start to stop. stop is included if the difference between start and stop is an integer multiple of step.

From the documentation of numpy.range: “When using a non-integer step, such as 0.1, the results will often not be consistent.” This replacement aims to avoid these inconsistencies.

Parameters

start, stop: [int, float]

the starting and (maximal) end values of the sequence.

step

[[int, float]] increment of the sequence, defaults to 1

`scopesim.utils.set_console_log_level(level='INFO')`

Set the level for the console handler (convenience function).

This controls what is actually printed to the console by ScopeSim. Accepted values are: DEBUG, INFO (default), WARNING, ERROR and CRITICAL.

`scopesim.utils.stringify_dict(dic, ignore_types=(<class 'str'>, <class 'int'>, <class 'float'>))`

Turn a dict entries into strings for addition to FITS headers.

`scopesim.utils.telescope_diffraction_limit(aperture_size, wavelength, distance=None)`

Return the diffraction limit of a telescope.

Parameters

aperture_size

[float] [m] The diameter of the primary mirror

wavelength

[float] [um] The wavelength for diffraction

distance

[float, optional] Default is None. If **distance** is given, the transverse distance for the diffraction limit is returned in the same units as **distance**

Returns**diff_limit**

[float] [arcsec] The angular diffraction limit. If **distance** is not None, **diff_limit** is in the same units as **distance**

`scopesim.utils.top_level_catch(func)`

Catch any unhandled exceptions, log it including bug report.

`scopesim.utils.transverse_distance(angle, distance)`

Turn an angular distance into a proper transverse distance.

Parameters**angle**

[float] [arcsec] The on-sky angle

distance

[float] The distance to the object. Units are arbitrary

Returns**trans_distance**

[float] proper transverse distance. Has the same Units as **distance**

`scopesim.utils.unify(x, unit, length=1)`

Convert all types of input to an astropy array/unit pair.

Parameters**x**

[int, float, np.ndarray, astropy.Quantity] The array to be turned into an `astropy.Quantity`

unit

[astropy.Quantity] The units to attach to the array

length

[int, optional] If *x* is a scalar, and the desired output is an array with *length*.

Returns**y**

[astropy.Quantity]

`scopesim.utils.unit_from_table(colname: str, table: Table, default_unit: str = "")` → Unit

Look for the unit for a column based on the meta dict keyword “<col>_unit”.

`scopesim.utils.update_logging(capture_warnings=True)`

Reload logging configuration from `rc.__config__`.

`scopesim.utils.write_report(text, filename=None, output=None)`

Write a report string to file in latex or rst format.

`scopesim.utils.zendist2airmass(zendist)`

Convert zenith distance to airmass.

Parameters

zenith distance

[[deg]] Zenith distance angle

Returns

airmass in sec(z) approximation

`scopesim.utils.zenith_dist_to_airmass(zenith_dist)`

zenith_dist is in degrees.

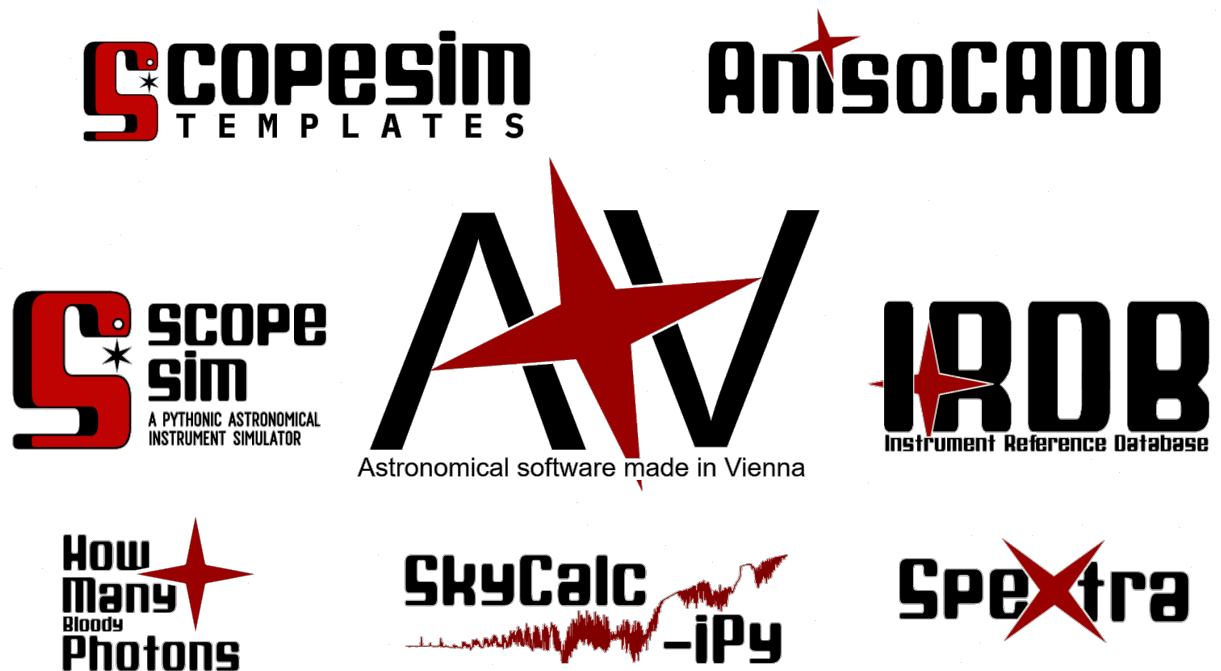
$X = \sec(Z)$

9.1.3 Module contents

Generalised telescope observation simulator.

THE SCOPESIM PYTHON ECOSYSTEM

There are several packages in the [ScopeSim](#) ecosystem to be aware of:



- [ScopeSim](#): The engine behind the whole simulator
- [ScopeSim Templates](#): A series of helper function to generate on-sky targets
- [SpeXtra](#): A pythonic interface to many common astronomical spectra libraries
- [Pyckles](#): Pythonic access to the Pickles (1998) spectral library and Brown (2014) spectral library
- [IRDB](#): The Instrument Reference Database, where the instrument packages are stored
- [AnisoCADO](#): For making SCAO PSF cubes that readable by ScopeSim
- [skycalc_ipy](#): Connects to ESOs SkyCalc server to get atmospheric spectra
- [How Many Photons](#): A simple package for quickly calculating the number of photons within a given astronomical filter

Note: Much more information on these packages will be coming very soon!

CONTACT

- For bugs, please add an [issue to the github repo](#)
- For enquiries on implementing your own instrument package, please drop us a line at
 - astar.astro@univie.ac.at or
 - kieran.leschinski@univie.ac.at
- For friendly chat, join the slack at https://join.slack.com/t/scopesim/shared_invite/zt-143s42izo-LnyqoG7gH5j~aGn5lZ~4IA

PYTHON MODULE INDEX

S

scopesim, 168
scopesim.base_classes, 160
scopesim.commands, 71
scopesim.commands.user_commands, 69
scopesim.detector, 73
scopesim.detector.detector, 71
scopesim.detector.detector_array, 72
scopesim.detector.nghxrg, 73
scopesim.effects, 118
scopesim.effects.apertures, 73
scopesim.effects.data_container, 76
scopesim.effects.detector_list, 77
scopesim.effects.effects, 80
scopesim.effects.effects_utils, 82
scopesim.effects.electronic, 82
scopesim.effects.fits_headers, 86
scopesim.effects.metis_lms_trace_list, 90
scopesim.effects.obs_strategies, 92
scopesim.effects.psf_utils, 93
scopesim.effects.psfs, 94
scopesim.effects.rotation, 97
scopesim.effects.shifts, 98
scopesim.effects.shutter, 99
scopesim.effects.spectral_efficiency, 100
scopesim.effects.spectral_trace_list, 100
scopesim.effects.spectral_trace_list_utils, 103
scopesim.effects.surface_list, 107
scopesim.effects.ter_curves, 108
scopesim.effects.ter_curves_utils, 115
scopesim.optics, 145
scopesim.optics.fov, 118
scopesim.optics.fov_manager, 121
scopesim.optics.fov_manager_utils, 123
scopesim.optics.fov_utils, 125
scopesim.optics.image_plane, 127
scopesim.optics.image_plane_utils, 129
scopesim.optics.monochromatic_trace_curve, 135
scopesim.optics.optical_element, 136
scopesim.optics.optical_train, 137
scopesim.optics.optics_manager, 140
scopesim.optics.radiometry_utils, 143
scopesim.optics.spectrograph, 143
scopesim.optics.surface, 143
scopesim.optics.surface_utils, 144
scopesim.rc, 161
scopesim.reports, 149
scopesim.reports.report_generator, 145
scopesim.reports.rst_utils, 146
scopesim.server, 154
scopesim.server.database, 149
scopesim.server.download_utils, 152
scopesim.server.example_data_utils, 153
scopesim.server.github_utils, 154
scopesim.source, 160
scopesim.source.source, 154
scopesim.source.source_templates, 157
scopesim.source.source_utils, 160
scopesim.utils, 161

A

- `ab_spectrum()` (in module `scopesim.source.source_templates`), 157
- `active_table` (`scopesim.effects.detector_list.DetectorList` property), 79
- `ADCWheel` (class in `scopesim.effects.ter_curves`), 108
- `add()` (`scopesim.optics.image_plane.ImagePlane` method), 128
- `add_bandpass()` (`scopesim.source.source.Source` method), 156
- `add_edge_zeros()` (in module `scopesim.effects.ter_curves_utils`), 115
- `add_effect()` (`scopesim.optics.optical_element.OpticalElement` method), 136
- `add_effect()` (`scopesim.optics.optics_manager.OpticsManager` method), 140
- `add_filter()` (`scopesim.effects.ter_curves.FilterWheelBase` method), 109
- `add_imagehdu_to_imagehdu()` (in module `scopesim.optics.image_plane_utils`), 129
- `add_keyword()` (in module `scopesim.utils`), 161
- `add_mags()` (in module `scopesim.utils`), 161
- `add_slit()` (`scopesim.effects.apertures.SlitWheel` method), 75
- `add_surface()` (`scopesim.effects.surface_list.SurfaceList` method), 107
- `add_surface_list()` (`scopesim.effects.surface_list.SurfaceList` method), 107
- `add_surface_to_table()` (in module `scopesim.optics.radiometry_utils`), 143
- `add_table_to_imagehdu()` (in module `scopesim.optics.image_plane_utils`), 129
- `affine_map()` (in module `scopesim.optics.image_plane_utils`), 130
- `airmass2zendist()` (in module `scopesim.utils`), 161
- `airmass_to_zenith_dist()` (in module `scopesim.utils`), 162
- `all_effects` (`scopesim.optics.optics_manager.OpticsManager` property), 140
- `AnalyticalPSF` (class in `scopesim.effects.psf`), 94
- `angle_in_arcseconds()` (in module `scopesim.utils`), 162
- `AnisocastroConstPSF` (class in `scopesim.effects.psf`), 94
- `ApertureList` (class in `scopesim.effects.apertures`), 73
- `ApertureMask` (class in `scopesim.effects.apertures`), 73
- `apertures` (`scopesim.effects.apertures.ApertureList` property), 73
- `append()` (`scopesim.source.source.Source` method), 156
- `apply_fov_effects()` (in module `scopesim.optics.optical_train`), 140
- `apply_throughput_to_cube()` (in module `scopesim.effects.ter_curves_utils`), 115
- `apply_to()` (`scopesim.effects.apertures.ApertureList` method), 73
- `apply_to()` (`scopesim.effects.apertures.ApertureMask` method), 74
- `apply_to()` (`scopesim.effects.apertures.SlitWheel` method), 75
- `apply_to()` (`scopesim.effects.detector_list.DetectorList` method), 79
- `apply_to()` (`scopesim.effects.effects.Effect` method), 80
- `apply_to()` (`scopesim.effects.electronic.AutoExposure` method), 83
- `apply_to()` (`scopesim.effects.electronic.BasicReadoutNoise` method), 83
- `apply_to()` (`scopesim.effects.electronic.Bias` method), 83
- `apply_to()` (`scopesim.effects.electronic.BinnedImage` method), 83
- `apply_to()` (`scopesim.effects.electronic.DarkCurrent` method), 83
- `apply_to()` (`scopesim.effects.electronic.DetectorModePropertiesSetter` method), 84
- `apply_to()` (`scopesim.effects.electronic.LinearityCurve` method), 85
- `apply_to()` (`scopesim.effects.electronic.PoorMansHxRGReadoutNoise` method), 85
- `apply_to()` (`scopesim.effects.electronic.Quantization` method), 85
- `apply_to()` (`scopesim.effects.electronic.ReferencePixelBorder` method), 85
- `apply_to()` (`scopesim.effects.electronic.ShotNoise` method), 85
- `apply_to()` (`scopesim.effects.electronic.SummedExposure`

- method), 86
- `apply_to()` (*scopesim.effects.electronic.UnequalBinnedImage* method), 86
- `apply_to()` (*scopesim.effects.fits_headers.EffectsMetaKeywords* method), 86
- `apply_to()` (*scopesim.effects.fits_headers.ExtraFitsKeywords* method), 88
- `apply_to()` (*scopesim.effects.fits_headers.SimulationConfigFitsKeywords* method), 88
- `apply_to()` (*scopesim.effects.fits_headers.SourceDescriptionFitsKeywords* method), 89
- `apply_to()` (*scopesim.effects.metis_lms_trace_list.MetisLMSSpectralTraceList* method), 91
- `apply_to()` (*scopesim.effects.obs_strategies.ChopNodCombiner* method), 93
- `apply_to()` (*scopesim.effects.psf.FieldVaryingPSF* method), 96
- `apply_to()` (*scopesim.effects.psf.PSF* method), 97
- `apply_to()` (*scopesim.effects.rotation.Rotate90CCD* method), 97
- `apply_to()` (*scopesim.effects.shifts.AtmosphericDispersionCorrection* method), 98
- `apply_to()` (*scopesim.effects.shifts.Shift3D* method), 98
- `apply_to()` (*scopesim.effects.shutter.Shutter* method), 99
- `apply_to()` (*scopesim.effects.spectral_efficiency.SpectralEfficiency* method), 100
- `apply_to()` (*scopesim.effects.spectral_trace_list.SpectralTraceList* method), 100
- `apply_to()` (*scopesim.effects.spectral_trace_list.SpectralTraceListWheel* method), 102
- `apply_to()` (*scopesim.effects.ter_curves.ADCWheel* method), 108
- `apply_to()` (*scopesim.effects.ter_curves.FilterWheelBase* method), 110
- `apply_to()` (*scopesim.effects.ter_curves.TERCurve* method), 113
- `area` (*scopesim.effects.surface_list.SurfaceList* property), 107
- `area` (*scopesim.optics.optics_manager.OpticsManager* property), 140
- `area` (*scopesim.optics.surface.SpectralSurface* property), 143
- `as_header()` (*scopesim.base_classes.PoorMansHeader* method), 161
- `atmospheric_refraction()` (in module *scopesim.effects.shifts*), 98
- `AtmosphericDispersion` (class in *scopesim.effects.shifts*), 98
- `AtmosphericDispersionCorrection` (class in *scopesim.effects.shifts*), 98
- `AtmosphericTERCurve` (class in *scopesim.effects.ter_curves*), 108
- `AutoExposure` (class in *scopesim.effects.electronic*), 82
- ## B
- `background_fields` (*scopesim.optics.fov.FieldOfView* property), 118
- `background_source` (*scopesim.effects.ter_curves.TERCurve* property), 113
- `BasicReadoutNoise` (class in *scopesim.effects.electronic*), 83
- `bias` (class in *scopesim.effects.electronic*), 83
- `BinnedImage` (class in *scopesim.effects.electronic*), 83
- `bug_report()` (in module *scopesim.utils*), 162
- `bug_report_to_file()` (in module *scopesim.utils*), 162
- ## C
- `calc_footprint()` (in module *scopesim.optics.image_plane_utils*), 130
- `calc_table_footprint()` (in module *scopesim.optics.image_plane_utils*), 131
- `center` (*scopesim.effects.ter_curves.FilterCurve* property), 109
- `centre` (*scopesim.effects.ter_curves.FilterCurve* property), 109
- `change_adc()` (*scopesim.effects.ter_curves.ADCWheel* method), 108
- `change_filter()` (*scopesim.effects.ter_curves.FilterWheelBase* method), 110
- `change_slit()` (*scopesim.effects.apertures.SlitWheel* method), 75
- `change_table_entry()` (in module *scopesim.utils*), 162
- `check_keys()` (in module *scopesim.utils*), 162
- `chop_nod_image()` (in module *scopesim.effects.obs_strategies*), 93
- `ChopNodCombiner` (class in *scopesim.effects.obs_strategies*), 92
- `clean_dict()` (in module *scopesim.utils*), 162
- `close_loop()` (in module *scopesim.utils*), 162
- `combine_emissions()` (in module *scopesim.optics.radiometry_utils*), 143
- `combine_imagehdu_fields()` (in module *scopesim.optics.fov_utils*), 125
- `combine_surface_effects()` (in module *scopesim.effects.effects_utils*), 82
- `combine_table_fields()` (in module *scopesim.optics.fov_utils*), 125
- `combine_tables()` (in module *scopesim.optics.radiometry_utils*), 143
- `combine_throughputs()` (in module *scopesim.optics.radiometry_utils*), 143
- `combine_two_spectra()` (in module *scopesim.effects.ter_curves_utils*), 115
- `combine_wavesets()` (in module *scopesim.optics.fov_manager_utils*), 123
- `compute_interpolation_functions()` (*scopesim.effects.metis_lms_trace_list.MetisLMSSpectralTrace* method), 90

`compute_interpolation_functions()` (*scopesim.effects.spectral_trace_list_utils.SpectralTrace* method), 103
`convert_table_comments_to_dict()` (in module *scopesim.utils*), 162
`convert_to_list_of_spectra()` (in module *scopesim.source.source_utils*), 160
`corners` (*scopesim.optics.fov.FieldOfView* property), 118
`crawl_server_dirs()` (in module *scopesim.server.database*), 149
`create_client()` (in module *scopesim.server.download_utils*), 152
`create_github_url()` (in module *scopesim.server.github_utils*), 154
`create_wcs_from_points()` (in module *scopesim.optics.image_plane_utils*), 131
`cube_fields` (*scopesim.optics.fov.FieldOfView* property), 118
`cube_fields` (*scopesim.source.source.Source* property), 156
`current_adc` (*scopesim.effects.ter_curves.ADCWheel* property), 108
`current_filter` (*scopesim.effects.ter_curves.FilterWheel* property), 110
`current_slit` (*scopesim.effects.apertures.SlitWheel* property), 75
`current_trace_list` (*scopesim.effects.spectral_trace_list_utils.SpectralTraceList* property), 102
`cutout_kernel()` (in module *scopesim.effects.psf_utils*), 93
D
`DarkCurrent` (class in *scopesim.effects.electronic*), 83
`data` (*scopesim.detector.detector.Detector* property), 71
`data` (*scopesim.effects.data_container.DataContainer* property), 77
`data` (*scopesim.optics.fov.FieldOfView* property), 118
`data` (*scopesim.optics.image_plane.ImagePlane* property), 128
`DataContainer` (class in *scopesim.effects.data_container*), 76
`deg2rad()` (in module *scopesim.utils*), 162
`deriv_polynomial2d()` (in module *scopesim.utils*), 162
`det_wcs_from_sky_wcs()` (in module *scopesim.optics.image_plane_utils*), 132
`Detector` (class in *scopesim.detector.detector*), 71
`detector_array_effects` (*scopesim.optics.optics_manager.OpticsManager* property), 140
`detector_effects` (*scopesim.optics.optics_manager.OpticsManager* property), 140
`detector_headers()` (*scopesim.effects.detector_list.DetectorList* method), 79
`detector_setup_effects` (*scopesim.optics.optics_manager.OpticsManager* property), 140
`DetectorArray` (class in *scopesim.detector.detector_array*), 72
`DetectorBase` (class in *scopesim.base_classes*), 160
`DetectorList` (class in *scopesim.effects.detector_list*), 77
`DetectorModePropertiesSetter` (class in *scopesim.effects.electronic*), 83
`DetectorWindow` (class in *scopesim.effects.detector_list*), 79
`DiscretePSF` (class in *scopesim.effects.psf*), 95
`display_name` (*scopesim.effects.effects.Effect* property), 80
`display_name` (*scopesim.optics.optical_element.OpticalElement* property), 136
`display_name` (*scopesim.optics.optics_manager.OpticsManager* property), 140
`dist_mod_from_distance()` (in module *scopesim.utils*), 162
`distance_from_dist_mod()` (in module *scopesim.utils*), 162
`download_example_data()` (in module *scopesim.server.example_data_utils*), 153
`download_github_folder()` (in module *scopesim.server.github_utils*), 154
`download_packages()` (in module *scopesim.server.database*), 149
`download_packages()` (in module *scopesim.server.database*), 149
`download_svo_filter()` (in module *scopesim.effects.ter_curves_utils*), 116
`download_svo_filter_list()` (in module *scopesim.effects.ter_curves_utils*), 116
`DownloadableFilterCurve` (class in *scopesim.effects.ter_curves*), 108
`dump()` (*scopesim.source.source.Source* method), 156
E
`echelle_setting()` (in module *scopesim.effects.metis_lms_trace_list*), 92
`Effect` (class in *scopesim.effects.effects*), 80
`effects` (*scopesim.optics.optical_train.OpticalTrain* property), 138
`EffectsMetaKeywords` (class in *scopesim.effects.fits_headers*), 86
`emission` (*scopesim.effects.surface_list.SurfaceList* property), 107
`emission` (*scopesim.effects.ter_curves.TERCurve* property), 113
`emission` (*scopesim.optics.surface.PoorMansSurface* attribute), 143

- emission (*scopesim.optics.surface.SpectralSurface* property), 144
 emissivity (*scopesim.optics.surface.SpectralSurface* property), 144
 empty_sky() (in module *scopesim.source.source_templates*), 157
 empty_surface_list() (in module *scopesim.effects.effects_utils*), 82
 empty_type() (in module *scopesim.utils*), 163
 extract() (*scopesim.optics.fov_manager.FovVolumeList* method), 121
 extract_area_from_imagehdu() (in module *scopesim.optics.fov_utils*), 125
 extract_area_from_table() (in module *scopesim.optics.fov_utils*), 126
 extract_base_from_unit() (in module *scopesim.utils*), 163
 extract_common_field() (in module *scopesim.optics.fov_utils*), 126
 extract_from() (*scopesim.detector.detector.Detector* method), 71
 extract_from() (*scopesim.optics.fov.FieldOfView* method), 118
 extract_range_from_spectrum() (in module *scopesim.optics.fov_utils*), 126
 extract_source() (in module *scopesim.optics.optical_train*), 140
 extract_type_from_unit() (in module *scopesim.utils*), 163
 ExtraFitsKeywords (class in *scopesim.effects.fits_headers*), 87
- ## F
- FieldConstantPSF (class in *scopesim.effects.psfs*), 95
 FieldOfView (class in *scopesim.optics.fov*), 118
 FieldOfViewBase (class in *scopesim.base_classes*), 160
 FieldVaryingPSF (class in *scopesim.effects.psfs*), 96
 figure_factory() (in module *scopesim.utils*), 163
 figure_grid_factory() (in module *scopesim.utils*), 163
 fill_zeros() (in module *scopesim.effects.spectral_trace_list_utils*), 106
 FilterCurve (class in *scopesim.effects.ter_curves*), 108
 FilterWheel (class in *scopesim.effects.ter_curves*), 109
 FilterWheelBase (class in *scopesim.effects.ter_curves*), 109
 find_file() (in module *scopesim.utils*), 163
 fit() (*scopesim.effects.spectral_trace_list_utils.Transform2D* class method), 105
 fit2matrix() (in module *scopesim.effects.spectral_trace_list_utils*), 106
 flatten() (*scopesim.optics.fov.FieldOfView* method), 118
 flatten_dict() (in module *scopesim.effects.fits_headers*), 89
 fluxes() (*scopesim.source.source.Source* method), 156
 footprint (*scopesim.effects.spectral_trace_list.SpectralTraceList* property), 100
 footprint() (*scopesim.effects.spectral_trace_list_utils.SpectralTrace* method), 103
 fov_effects (*scopesim.optics.optics_manager.OpticsManager* property), 140
 fov_footprints (*scopesim.optics.fov_manager.FOVManager* property), 121
 fov_grid() (*scopesim.effects.apertures.ApertureMask* method), 74
 fov_grid() (*scopesim.effects.apertures.SlitWheel* method), 76
 fov_grid() (*scopesim.effects.detector_list.DetectorList* method), 79
 fov_grid() (*scopesim.effects.effects.Effect* method), 80
 fov_grid() (*scopesim.effects.metis_lms_trace_list.MetisLMSSpectralTrace* method), 90
 fov_grid() (*scopesim.effects.psfs.GaussianDiffractionPSF* method), 96
 fov_grid() (*scopesim.effects.psfs.NonCommonPathAberration* method), 96
 fov_grid() (*scopesim.effects.psfs.PSF* method), 97
 fov_grid() (*scopesim.effects.shifts.AtmosphericDispersionCorrection* method), 98
 fov_grid() (*scopesim.effects.shifts.Shift3D* method), 98
 fov_grid() (*scopesim.effects.spectral_trace_list_utils.SpectralTrace* method), 103
 fov_grid() (*scopesim.effects.surface_list.SurfaceList* method), 107
 fov_grid() (*scopesim.effects.ter_curves.FilterCurve* method), 109
 fov_grid() (*scopesim.effects.ter_curves.FilterWheelBase* method), 110
 fov_setup_effects (*scopesim.optics.optics_manager.OpticsManager* property), 140
 FOVManager (class in *scopesim.optics.fov_manager*), 121
 fovs (*scopesim.optics.fov_manager.FOVManager* property), 121
 FOVSetupBase (class in *scopesim.base_classes*), 160
 FovVolumeList (class in *scopesim.optics.fov_manager*), 121
 fp2sky() (*scopesim.effects.metis_lms_trace_list.MetisLMSSpectralTrace* method), 90
 from_currsys() (in module *scopesim.utils*), 164
 from_meta() (*scopesim.optics.surface.SpectralSurface* method), 144
 from_rc_config() (in module *scopesim.utils*), 164
 fwhm (*scopesim.effects.ter_curves.FilterCurve* property), 109

G

- GaussianDiffractionPSF (class in *scopesim.effects.psfs*), 96
- generate_fovs_list() (*scopesim.fov_manager.FOVManager* method), 121
- get_3d_shifts() (in module *scopesim.optics.fov_manager_utils*), 123
- get_affine_parameters() (in module *scopesim.effects.spectral_trace_list_utils*), 106
- get_all() (*scopesim.optics.optical_element.OpticalElement* method), 136
- get_all() (*scopesim.optics.optics_manager.OpticsManager* method), 140
- get_all_effects() (in module *scopesim.effects.effects_utils*), 82
- get_all_latest() (in module *scopesim.server.database*), 150
- get_all_package_versions() (in module *scopesim.server.database*), 150
- get_all_packages_on_server() (in module *scopesim.server.database*), 150
- get_all_stable() (in module *scopesim.server.database*), 151
- get_apertures() (*scopesim.effects.apertures.ApertureList* method), 73
- get_base_url() (in module *scopesim.server.database*), 151
- get_bkg_level() (in module *scopesim.effects.psf_utils*), 93
- get_canvas_header() (in module *scopesim.optics.image_plane_utils*), 132
- get_cube_waveset() (in module *scopesim.optics.fov_utils*), 126
- get_data() (*scopesim.effects.data_container.DataContainer* method), 77
- get_efficiencies() (*scopesim.effects.spectral_efficiency.SpectralEfficiency* method), 100
- get_emission() (*scopesim.effects.surface_list.SurfaceList* method), 107
- get_filter() (in module *scopesim.effects.ter_curves_utils*), 116
- get_filter_effective_wavelength() (in module *scopesim.effects.ter_curves_utils*), 116
- get_fits_type() (in module *scopesim.utils*), 164
- get_header() (*scopesim.effects.apertures.ApertureMask* method), 74
- get_header() (*scopesim.optics.monochromatic_trace_curve.MonochromeTraceCurve* method), 135
- get_imaging_fovs() (in module *scopesim.optics.fov_manager_utils*), 124
- get_imaging_headers() (in module *scopesim.optics.fov_manager_utils*), 124
- get_imaging_waveset() (in module *scopesim.optics.fov_manager_utils*), 124
- get_kernel() (*scopesim.effects.psfs.AnisocastroConstPSF* method), 95
- get_kernel() (*scopesim.effects.psfs.FieldConstantPSF* method), 96
- get_kernel() (*scopesim.effects.psfs.FieldVaryingPSF* method), 96
- get_kernel() (*scopesim.effects.psfs.GaussianDiffractionPSF* method), 96
- get_kernel() (*scopesim.effects.psfs.NonCommonPathAberration* method), 96
- get_kernel() (*scopesim.effects.psfs.PSF* method), 97
- get_kernel() (*scopesim.effects.psfs.SeeingPSF* method), 97
- get_kernel() (*scopesim.effects.psfs.Vibration* method), 97
- get_latest() (in module *scopesim.server.database*), 151
- get_mask() (*scopesim.effects.apertures.ApertureMask* method), 74
- get_matrices() (*scopesim.effects.metis_lms_trace_list.MetisLMSSpectral* method), 90
- get_meta_quantity() (in module *scopesim.utils*), 164
- get_package_folders() (in module *scopesim.server.database*), 151
- get_pixel_border_waves_from_atmo_disp() (in module *scopesim.effects.shifts*), 99
- get_psf_wave_exts() (in module *scopesim.effects.psf_utils*), 93
- get_relevant_extensions() (in module *scopesim.effects.fits_headers*), 89
- get_server_elements() (in module *scopesim.server.example_data_utils*), 153
- get_server_folder_contents() (in module *scopesim.server.download_utils*), 152
- get_server_folder_package_names() (in module *scopesim.server.database*), 151
- get_server_package_list() (in module *scopesim.server.database*), 151
- get_spectroscopy_fovs() (in module *scopesim.optics.fov_manager_utils*), 124
- get_spectroscopy_headers() (in module *scopesim.optics.fov_manager_utils*), 124
- get_stable() (in module *scopesim.server.database*), 152
- get_strehl_cutout() (in module *scopesim.effects.psf_utils*), 93
- get_table() (*scopesim.effects.apertures.RectangularApertureMask* method), 75
- get_table() (*scopesim.effects.apertures.SlitWheel* method), 76
- get_table() (*scopesim.effects.shifts.AtmosphericDispersion* method), 98

[get_table\(\)](#) (*scopesim.effects.shifts.Shift3D* method), 98
[get_table\(\)](#) (*scopesim.effects.ter_curves.ADCWheel* method), 108
[get_table\(\)](#) (*scopesim.effects.ter_curves.FilterWheelBase* method), 110
[get_throughput\(\)](#) (*scopesim.effects.surface_list.SurfaceList* method), 107
[get_total_wfe_from_table\(\)](#) (in module *scopesim.effects.psf_utils*), 93
[get_waverange\(\)](#) (*scopesim.effects.metis_lms_trace_list.MetisLMSSpectralTraceList* method), 91
[get_z_order_effects\(\)](#) (*scopesim.optics.optical_element.OpticalElement* method), 136
[get_z_order_effects\(\)](#) (*scopesim.optics.optics_manager.OpticsManager* method), 141
[get_zero_mag_spectrum\(\)](#) (in module *scopesim.effects.ter_curves_utils*), 117
[gradient\(\)](#) (*scopesim.effects.spectral_trace_list_utils.TransformUtils* method), 106
[group_package_versions\(\)](#) (in module *scopesim.server.database*), 152

H

[handle_download\(\)](#) (in module *scopesim.server.download_utils*), 152
[handle_unzipping\(\)](#) (in module *scopesim.server.download_utils*), 152
[has_needed_keywords\(\)](#) (in module *scopesim.utils*), 164
[hdu](#) (*scopesim.detector.detector.Detector* property), 72
[hdu](#) (*scopesim.effects.apertures.ApertureMask* property), 74
[header](#) (*scopesim.detector.detector.Detector* property), 72
[header](#) (*scopesim.effects.apertures.ApertureMask* property), 74
[header](#) (*scopesim.optics.image_plane.ImagePlane* property), 128
[header](#) (*scopesim.optics.monochromatic_trace_curve.MonochromaticTraceCurve* property), 135
[header_from_list_of_xy\(\)](#) (in module *scopesim.optics.image_plane_utils*), 132

I

[image](#) (*scopesim.detector.detector.Detector* property), 72
[image](#) (*scopesim.optics.image_plane.ImagePlane* property), 129
[image\(\)](#) (*scopesim.source.source.Source* method), 156
[image_fields](#) (*scopesim.optics.fov.FieldOfView* property), 118
[image_fields](#) (*scopesim.source.source.Source* property), 156
[image_in_range\(\)](#) (*scopesim.source.source.Source* method), 156
[image_plane_effects](#) (*scopesim.optics.optics_manager.OpticsManager* property), 141
[image_plane_header](#) (*scopesim.effects.detector_list.DetectorList* property), 79
[image_plane_header](#) (*scopesim.effects.spectral_trace_list.SpectralTraceList* property), 91
[image_plane_headers](#) (*scopesim.optics.optics_manager.OpticsManager* property), 141
[image_plane_id](#) (*scopesim.effects.detector_list.DetectorList* property), 79
[image_plane_setup_effects](#) (*scopesim.optics.optics_manager.OpticsManager* property), 141
[ImagePlane](#) (class in *scopesim.optics.image_plane*), 127
[ImagePlaneBase](#) (class in *scopesim.base_classes*), 160
[include](#) (*scopesim.effects.effects.Effect* property), 80
[include](#) (*scopesim.effects.ter_curves.SkycalcTERCurve* property), 111
[info\(\)](#) (*scopesim.effects.effects.Effect* method), 80
[insert\(\)](#) (*scopesim.optics.fov_manager.FovVolumeList* method), 122
[insert_into_orderreddict\(\)](#) (in module *scopesim.utils*), 164
[interp2\(\)](#) (in module *scopesim.utils*), 164
[is_empty](#) (*scopesim.effects.surface_list.SurfaceList* property), 107
[is_field_in_fov\(\)](#) (in module *scopesim.optics.fov_utils*), 126
[is_fits](#) (*scopesim.effects.data_container.DataContainer* property), 77
[is_fits\(\)](#) (in module *scopesim.utils*), 164
[is_flux_binned\(\)](#) (in module *scopesim.optics.surface_utils*), 144
[is_spectroscopy](#) (*scopesim.optics.optics_manager.OpticsManager* property), 141
[is_spectroscopy\(\)](#) (in module *scopesim.effects.effects_utils*), 82
[items\(\)](#) (*scopesim.base_classes.PoorMansHeader* method), 161

K

[keys\(\)](#) (*scopesim.base_classes.PoorMansHeader* method), 161

L

[lam2phase\(\)](#) (*scopesim.effects.metis_lms_trace_list.MetisLMSSpectralTraceList* method), 91

latexify_rst_text() (in module <i>scopesim.reports.rst_utils</i>), 146	make_img_wcs_header() (in module <i>scopesim.source.source_utils</i>), 160
LinearityCurve (class in <i>scopesim.effects.electronic</i>), 84	make_primary_hdu() (in module <i>scopesim.detector.detector_array</i>), 72
list_effects() (<i>scopesim.optics.optical_element.OpticalElement</i> method), 137	make_psf_cube() (<i>scopesim.effects.psf.FieldConstantPSF</i> method), 96
list_effects() (<i>scopesim.optics.optics_manager.OpticsManager</i> method), 141	make_spectral_traces() (in module <i>scopesim.effects.electronic</i>), 86
list_example_data() (in module <i>scopesim.server.example_data_utils</i>), 153	make_spectral_traces() (<i>scopesim.effects.metis_lms_trace_list.MetisLMSSpectralTraceList</i> method), 91
list_modes() (<i>scopesim.commands.user_commands.UserCommands</i> method), 71	make_spectral_traces() (<i>scopesim.effects.spectral_trace_list.SpectralTraceList</i> method), 100
list_modes() (<i>scopesim.effects.electronic.DetectorModeProperties</i> method), 84	make_spectrum() (<i>scopesim.optics.fov.FieldOfView</i> method), 120
list_packages() (in module <i>scopesim.server.database</i>), 152	make_strehl_map_from_table() (in module <i>scopesim.effects.psf_utils</i>), 93
load() (<i>scopesim.optics.optical_train.OpticalTrain</i> method), 138	make_ter_curve() (<i>scopesim.effects.metis_lms_trace_list.MetisLMSEfficiency</i> method), 90
load() (<i>scopesim.source.source.Source</i> class method), 156	map_spectra_to_focal_plane() (<i>scopesim.effects.spectral_trace_list_utils.SpectralTraceListUtils</i> method), 103
load_effects() (<i>scopesim.optics.optics_manager.OpticsManager</i> method), 141	mask (<i>scopesim.effects.apertures.ApertureMask</i> property), 74
load_skycalc_table() (<i>scopesim.effects.ter_curves.SkycalcTERCurve</i> method), 111	mask_from_coords() (in module <i>scopesim.effects.apertures</i>), 76
log_bug_report() (in module <i>scopesim.utils</i>), 164	masks_list (<i>scopesim.optics.optical_element.OpticalElement</i> property), 137
log_to_file() (in module <i>scopesim.utils</i>), 164	meta (<i>scopesim.optics.surface.PoorMansSurface</i> attribute), 143
M	meta_string (<i>scopesim.effects.effects.Effect</i> property), 81
make_aperture_polygon() (in module <i>scopesim.effects.apertures</i>), 76	MetisLMSEfficiency (class in <i>scopesim.effects.metis_lms_trace_list</i>), 90
make_copy() (<i>scopesim.source.source.Source</i> method), 156	MetisLMSImageSlicer (class in <i>scopesim.effects.metis_lms_trace_list</i>), 90
make_cube_from_table() (in module <i>scopesim.optics.fov_utils</i>), 127	MetisLMSSpectralTrace (class in <i>scopesim.effects.metis_lms_trace_list</i>), 90
make_cube_hdu() (<i>scopesim.optics.fov.FieldOfView</i> method), 118	MetisLMSSpectralTraceList (class in <i>scopesim.effects.metis_lms_trace_list</i>), 91
make_effect() (in module <i>scopesim.effects.effects_utils</i>), 82	mirror_angle (<i>scopesim.optics.surface.SpectralSurface</i> property), 144
make_effects_hdu() (in module <i>scopesim.detector.detector_array</i>), 72	modes (<i>scopesim.commands.user_commands.UserCommands</i> property), 71
make_emission_from_array() (in module <i>scopesim.optics.surface_utils</i>), 144	module
make_emission_from_emissivity() (in module <i>scopesim.optics.surface_utils</i>), 145	scopesim, 168
make_flux_table() (in module <i>scopesim.optics.fov_utils</i>), 127	scopesim.base_classes, 160
make_image_hdu() (<i>scopesim.optics.fov.FieldOfView</i> method), 119	scopesim.commands, 71
make_image_interpolations() (in module <i>scopesim.effects.spectral_trace_list_utils</i>), 106	scopesim.commands.user_commands, 69
make_imagehdu_from_table() (in module <i>scopesim.source.source_utils</i>), 160	scopesim.detector, 73
	scopesim.detector.detector, 71
	scopesim.detector.detector_array, 72
	scopesim.detector.nghxrg, 73

- scopesim.effects, 118
- scopesim.effects.apertures, 73
- scopesim.effects.data_container, 76
- scopesim.effects.detector_list, 77
- scopesim.effects.effects, 80
- scopesim.effects.effects_utils, 82
- scopesim.effects.electronic, 82
- scopesim.effects.fits_headers, 86
- scopesim.effects.metis_lms_trace_list, 90
- scopesim.effects.obs_strategies, 92
- scopesim.effects.psf_utils, 93
- scopesim.effects.psfs, 94
- scopesim.effects.rotation, 97
- scopesim.effects.shifts, 98
- scopesim.effects.shutter, 99
- scopesim.effects.spectral_efficiency, 100
- scopesim.effects.spectral_trace_list, 100
- scopesim.effects.spectral_trace_list_utils, 103
- scopesim.effects.surface_list, 107
- scopesim.effects.ter_curves, 108
- scopesim.effects.ter_curves_utils, 115
- scopesim.optics, 145
- scopesim.optics.fov, 118
- scopesim.optics.fov_manager, 121
- scopesim.optics.fov_manager_utils, 123
- scopesim.optics.fov_utils, 125
- scopesim.optics.image_plane, 127
- scopesim.optics.image_plane_utils, 129
- scopesim.optics.monochromatic_trace_curve, 135
- scopesim.optics.optical_element, 136
- scopesim.optics.optical_train, 137
- scopesim.optics.optics_manager, 140
- scopesim.optics.radiometry_utils, 143
- scopesim.optics.spectrograph, 143
- scopesim.optics.surface, 143
- scopesim.optics.surface_utils, 144
- scopesim.rc, 161
- scopesim.reports, 149
- scopesim.reports.report_generator, 145
- scopesim.reports.rst_utils, 146
- scopesim.server, 154
- scopesim.server.database, 149
- scopesim.server.download_utils, 152
- scopesim.server.example_data_utils, 153
- scopesim.server.github_utils, 154
- scopesim.source, 160
- scopesim.source.source, 154
- scopesim.source.source_templates, 157
- scopesim.source.source_utils, 160
- scopesim.utils, 161
- moffat() (in module scopesim.utils), 164
- MonochromeTraceCurve (class in scopesim.optics.monochromatic_trace_curve), 135
- N**
- nearest() (in module scopesim.utils), 164
- nearest_index() (in module scopesim.effects.psf_utils), 93
- nmRms (scopesim.effects.psfs.AnisocadoConstPSF property), 95
- nmrms_from_strehl_and_wavelength() (in module scopesim.effects.psf_utils), 93
- NonCommonPathAberration (class in scopesim.effects.psfs), 96
- normalise_binned_flux() (in module scopesim.optics.surface_utils), 145
- O**
- observe() (scopesim.optics.optical_train.OpticalTrain method), 138
- OpticalElement (class in scopesim.optics.optical_element), 136
- OpticalTrain (class in scopesim.optics.optical_train), 137
- OpticsManager (class in scopesim.optics.optics_manager), 140
- overlay_image() (in module scopesim.optics.image_plane_utils), 133
- P**
- parallactic_angle() (in module scopesim.utils), 165
- phase2lam() (scopesim.effects.metis_lms_trace_list.MetisLMSSpectralTrain method), 91
- photons_in_range() (in module scopesim.source.source_utils), 160
- photons_in_range() (scopesim.source.source.Source method), 157
- pix2val() (in module scopesim.optics.image_plane_utils), 133
- pixel_area (scopesim.optics.fov.FieldOfView property), 120
- PkgNotFoundError, 149
- plot() (scopesim.effects.apertures.ApertureList method), 73
- plot() (scopesim.effects.apertures.ApertureMask method), 74
- plot() (scopesim.effects.detector_list.DetectorList method), 79
- plot() (scopesim.effects.electronic.BasicReadoutNoise method), 83
- plot() (scopesim.effects.electronic.DarkCurrent method), 83
- plot() (scopesim.effects.electronic.LinearityCurve method), 85

`plot()` (*scopesim.effects.electronic.PoorMansHxRGReadoutNoise* method), 85
`plot()` (*scopesim.effects.electronic.ReferencePixelBorder* method), 85
`plot()` (*scopesim.effects.electronic.ShotNoise* method), 85
`plot()` (*scopesim.effects.psfs.AnisocadoConstPSF* method), 95
`plot()` (*scopesim.effects.psfs.FieldConstantPSF* method), 96
`plot()` (*scopesim.effects.psfs.FieldVaryingPSF* method), 96
`plot()` (*scopesim.effects.psfs.GaussianDiffractionPSF* method), 96
`plot()` (*scopesim.effects.psfs.NonCommonPathAberration* method), 96
`plot()` (*scopesim.effects.psfs.PSF* method), 97
`plot()` (*scopesim.effects.psfs.SeeingPSF* method), 97
`plot()` (*scopesim.effects.shifts.AtmosphericDispersionCorrection* method), 98
`plot()` (*scopesim.effects.shifts.Shift3D* method), 98
`plot()` (*scopesim.effects.spectral_efficiency.SpectralEfficiency* method), 100
`plot()` (*scopesim.effects.spectral_trace_list.SpectralTraceList* method), 100
`plot()` (*scopesim.effects.spectral_trace_list_utils.SpectralTraceListUtils* method), 103
`plot()` (*scopesim.effects.surface_list.SurfaceList* method), 107
`plot()` (*scopesim.effects.ter_curves.FilterWheelBase* method), 110
`plot()` (*scopesim.effects.ter_curves.TERCurve* method), 113
`plot()` (*scopesim.source.source.Source* method), 157
`plot_hist()` (*scopesim.effects.electronic.BasicReadoutNoise* method), 83
`plot_hist()` (*scopesim.effects.electronic.PoorMansHxRGReadoutNoise* method), 85
`plot_hist()` (*scopesim.effects.electronic.ShotNoise* method), 86
`plot_masks()` (*scopesim.effects.apertures.ApertureList* method), 73
`plotify_rst_text()` (in module *scopesim.reports.rst_utils*), 146
`points_on_a_circle()` (in module *scopesim.effects.apertures*), 76
`poissonify()` (in module *scopesim.utils*), 165
`PoorMansFOV` (class in *scopesim.effects.psfs*), 97
`PoorMansHeader` (class in *scopesim.base_classes*), 161
`PoorMansHxRGReadoutNoise` (class in *scopesim.effects.electronic*), 85
`PoorMansSurface` (class in *scopesim.optics.surface*), 143
`power_vector()` (in module *scopesim.utils*), 165
`prepare_source()` (*scopesim.optics.optical_train.OpticalTrain* method), 139
`pretty_print_dict()` (in module *scopesim.utils*), 165
`pretty_str()` (*scopesim.optics.optical_element.OpticalElement* method), 137
`pretty_str()` (*scopesim.optics.optics_manager.OpticsManager* method), 142
`process_code()` (in module *scopesim.reports.rst_utils*), 147
`process_comment_code()` (in module *scopesim.reports.rst_utils*), 148
`process_literal_code()` (in module *scopesim.reports.rst_utils*), 148
`properties_str` (*scopesim.optics.optical_element.OpticalElement* property), 137
`pseudo_random_field()` (in module *scopesim.effects.electronic*), 86
`PSF` (class in *scopesim.effects.psfs*), 96
`PupilTransmission` (class in *scopesim.effects.ter_curves*), 111

Q

`quantify()` (in module *scopesim.utils*), 166
`quantity_from_table()` (in module *scopesim.utils*), 166
`Quantization` (class in *scopesim.effects.electronic*), 85
`QuantumEfficiencyCurve` (class in *scopesim.effects.ter_curves*), 111
`query_server()` (*scopesim.effects.ter_curves.SkycalcTERCurve* method), 111

R

`rad2deg()` (in module *scopesim.utils*), 166
`readout()` (*scopesim.detector.detector_array.DetectorArray* method), 72
`readout()` (*scopesim.optics.optical_train.OpticalTrain* method), 139
`real_colname()` (in module *scopesim.utils*), 166
`RectangularApertureMask` (class in *scopesim.effects.apertures*), 75
`rectify()` (*scopesim.effects.spectral_trace_list_utils.SpectralTraceListUtils* method), 104
`rectify_cube()` (*scopesim.effects.metis_lms_trace_list.MetisLMSSpectralTraceList* method), 91
`rectify_cube()` (*scopesim.effects.spectral_trace_list.SpectralTraceList* method), 101
`rectify_traces()` (*scopesim.effects.spectral_trace_list.SpectralTraceList* method), 101
`ReferencePixelBorder` (class in *scopesim.effects.electronic*), 85
`reflection` (*scopesim.optics.surface.SpectralSurface* property), 144
`remake_kernel()` (*scopesim.effects.psfs.AnisocadoConstPSF* method), 95

<code>reorient_imagehdu()</code>	(in module <code>scopesim.optics.image_plane_utils</code>), 133	module, 168
<code>report()</code>	(<code>scopesim.effects.effects.Effect</code> method), 81	<code>scopesim.base_classes</code> module, 160
<code>report()</code>	(<code>scopesim.optics.optical_element.OpticalElement</code> method), 137	<code>scopesim.commands</code> module, 71
<code>report()</code>	(<code>scopesim.optics.optical_train.OpticalTrain</code> method), 139	<code>scopesim.commands.user_commands</code> module, 69
<code>report()</code>	(<code>scopesim.optics.optics_manager.OpticsManager</code> method), 142	<code>scopesim.detector</code> module, 73
<code>ReportGenerator</code>	(class in <code>scopesim.reports.report_generator</code>), 145	<code>scopesim.detector.detector</code> module, 71
<code>required_keys</code>	(<code>scopesim.effects.apertures.SlitWheel</code> attribute), 76	<code>scopesim.detector.detector_array</code> module, 72
<code>required_keys</code>	(<code>scopesim.effects.spectral_trace_list.SpectralTraceList</code> attribute), 102	<code>scopesim.detector.ngxrg</code> module, 73
<code>required_keys</code>	(<code>scopesim.effects.ter_curves.ADCWheel</code> attribute), 108	<code>scopesim.effects</code> module, 118
<code>required_keys</code>	(<code>scopesim.effects.ter_curves.FilterWheel</code> attribute), 109	<code>scopesim.effects.apertures</code> module, 73
<code>required_keys</code>	(<code>scopesim.effects.ter_curves.FilterWheelBase</code> attribute), 110	<code>scopesim.effects.data_container</code> module, 76
<code>required_keys</code>	(<code>scopesim.effects.ter_curves.SpanishVOFilterWheel</code> attribute), 112	<code>scopesim.effects.detector_list</code> module, 77
<code>required_keys</code>	(<code>scopesim.effects.ter_curves.TopHatFilterWheel</code> attribute), 115	<code>scopesim.effects.effects</code> module, 80
<code>rescale_imagehdu()</code>	(in module <code>scopesim.optics.image_plane_utils</code>), 133	<code>scopesim.effects.effects_utils</code> module, 82
<code>rescale_kernel()</code>	(in module <code>scopesim.effects.psf_utils</code>), 94	<code>scopesim.effects.electronic</code> module, 82
<code>reset()</code>	(<code>scopesim.detector.detector.Detector</code> method), 72	<code>scopesim.effects.fits_headers</code> module, 86
<code>return_latest_github_actions_jobs_status()</code>	(in module <code>scopesim.utils</code>), 166	<code>scopesim.effects.metis_lms_trace_list</code> module, 90
<code>rolling_median()</code>	(in module <code>scopesim.effects.spectral_trace_list_utils</code>), 106	<code>scopesim.effects.obs_strategies</code> module, 92
<code>rotate()</code>	(in module <code>scopesim.effects.apertures</code>), 76	<code>scopesim.effects.psf_utils</code> module, 93
<code>rotate()</code>	(<code>scopesim.source.source.Source</code> method), 157	<code>scopesim.effects.psfs</code> module, 94
<code>Rotate90CCD</code>	(class in <code>scopesim.effects.rotation</code>), 97	<code>scopesim.effects.rotation</code> module, 97
<code>rotational_blur()</code>	(in module <code>scopesim.effects.psf_utils</code>), 94	<code>scopesim.effects.shifts</code> module, 98
<code>round_kernel_edges()</code>	(in module <code>scopesim.effects.psf_utils</code>), 94	<code>scopesim.effects.shutter</code> module, 99
<code>rst_default_yaml()</code>	(<code>scopesim.reports.report_generator.ReportGenerator</code> method), 145	<code>scopesim.effects.spectral_efficiency</code> module, 100
<code>rstify_rst_text()</code>	(in module <code>scopesim.reports.rst_utils</code>), 148	<code>scopesim.effects.spectral_trace_list</code> module, 100
S		<code>scopesim.effects.spectral_trace_list_utils</code> module, 103
<code>scale_imagehdu()</code>	(in module <code>scopesim.source.source_utils</code>), 160	<code>scopesim.effects.surface_list</code> module, 107
<code>scale_spectrum()</code>	(in module <code>scopesim.effects.ter_curves_utils</code>), 117	<code>scopesim.effects.ter_curves</code>
<code>scopesim</code>		

- module, 108
- scopesim.effects.ter_curves_utils
 - module, 115
- scopesim.optics
 - module, 145
- scopesim.optics.fov
 - module, 118
- scopesim.optics.fov_manager
 - module, 121
- scopesim.optics.fov_manager_utils
 - module, 123
- scopesim.optics.fov_utils
 - module, 125
- scopesim.optics.image_plane
 - module, 127
- scopesim.optics.image_plane_utils
 - module, 129
- scopesim.optics.monochromatic_trace_curve
 - module, 135
- scopesim.optics.optical_element
 - module, 136
- scopesim.optics.optical_train
 - module, 137
- scopesim.optics.optics_manager
 - module, 140
- scopesim.optics.radiometry_utils
 - module, 143
- scopesim.optics.spectrograph
 - module, 143
- scopesim.optics.surface
 - module, 143
- scopesim.optics.surface_utils
 - module, 144
- scopesim.rc
 - module, 161
- scopesim.reports
 - module, 149
- scopesim.reports.report_generator
 - module, 145
- scopesim.reports.rst_utils
 - module, 146
- scopesim.server
 - module, 154
- scopesim.server.database
 - module, 149
- scopesim.server.download_utils
 - module, 152
- scopesim.server.example_data_utils
 - module, 153
- scopesim.server.github_utils
 - module, 154
- scopesim.source
 - module, 160
- scopesim.source.source
 - module, 154
- scopesim.source.source_templates
 - module, 157
- scopesim.source.source_utils
 - module, 160
- scopesim.utils
 - module, 161
- scopesim_effect_classes() (in module *scopesim.effects.effects_utils*), 82
- SeeingPSF (class in *scopesim.effects.psf*), 97
- select_mode() (*scopesim.effects.electronic.DetectorModePropertiesSetter* method), 84
- SemiAnalyticalPSF (class in *scopesim.effects.psf*), 97
- send_get() (in module *scopesim.server.download_utils*), 153
- seq() (in module *scopesim.utils*), 166
- ServerError, 152
- set_console_log_level() (in module *scopesim.utils*), 166
- set_derived_parameters() (*scopesim.optics.optics_manager.OpticsManager* method), 142
- set_modes() (*scopesim.commands.user_commands.UserCommands* method), 71
- shift() (*scopesim.source.source.Source* method), 157
- Shift3D (class in *scopesim.effects.shifts*), 98
- ShotNoise (class in *scopesim.effects.electronic*), 85
- shrink() (*scopesim.optics.fov_manager.FovVolumeList* method), 122
- shutdown() (*scopesim.optics.optical_train.OpticalTrain* method), 139
- Shutter (class in *scopesim.effects.shutter*), 99
- sigma2gauss() (in module *scopesim.effects.psf_utils*), 94
- SimulationConfigFitsKeywords (class in *scopesim.effects.fits_headers*), 88
- sky2fp() (in module *scopesim.optics.fov_utils*), 127
- sky2fp() (*scopesim.effects.metis_lms_trace_list.MetisLMSSpectralTrace* method), 91
- sky_wcs_from_det_wcs() (in module *scopesim.optics.image_plane_utils*), 134
- SkycalcTERCurve (class in *scopesim.effects.ter_curves*), 111
- SlitWheel (class in *scopesim.effects.apertures*), 75
- Source (class in *scopesim.source.source*), 155
- source_effects (*scopesim.optics.optics_manager.OpticsManager* property), 142
- SourceBase (class in *scopesim.base_classes*), 161
- SourceDescriptionFitsKeywords (class in *scopesim.effects.fits_headers*), 88
- SpanishVOFilterCurve (class in *scopesim.effects.ter_curves*), 111
- SpanishVOFilterWheel (class in *scopesim.effects.ter_curves*), 112

- SpectralEfficiency (class in *scopesim.effects.spectral_efficiency*), 100
- SpectralSurface (class in *scopesim.optics.surface*), 143
- SpectralTrace (class in *scopesim.effects.spectral_trace_list_utils*), 103
- SpectralTraceList (class in *scopesim.effects.spectral_trace_list*), 100
- SpectralTraceListWheel (class in *scopesim.effects.spectral_trace_list*), 101
- split() (*scopesim.optics.fov_manager.FovVolumeList* method), 122
- split_header() (in module *scopesim.optics.image_plane_utils*), 134
- st_spectrum() (in module *scopesim.source.source_templates*), 157
- star() (in module *scopesim.source.source_templates*), 157
- star_field() (in module *scopesim.source.source_templates*), 158
- strehl2sigma() (in module *scopesim.effects.psf_utils*), 94
- strehl_imagehdu (*scopesim.effects.psf.FieldVaryingPSF* property), 96
- strehl_ratio (*scopesim.effects.psf.AnisocadoConstPSF* property), 95
- string_to_table() (in module *scopesim.optics.radiometry_utils*), 143
- stringify_dict() (in module *scopesim.utils*), 166
- sub_pixel_fractions() (in module *scopesim.optics.image_plane_utils*), 134
- SummedExposure (class in *scopesim.effects.electronic*), 86
- surface (*scopesim.effects.surface_list.SurfaceList* property), 108
- surface (*scopesim.effects.ter_curves.FilterWheelBase* property), 111
- SurfaceList (class in *scopesim.effects.surface_list*), 107
- surfaces_list (*scopesim.optics.optical_element.OpticalElement* property), 137
- surfaces_table (*scopesim.optics.optics_manager.OpticsManager* property), 142
- system_transmission (*scopesim.optics.optics_manager.OpticsManager* property), 142
- T**
- table_fields (*scopesim.optics.fov.FieldOfView* property), 120
- table_fields (*scopesim.source.source.Source* property), 157
- table_to_rst() (in module *scopesim.reports.rst_utils*), 148
- telescope_diffraction_limit() (in module *scopesim.utils*), 166
- TERCurve (class in *scopesim.effects.ter_curves*), 112
- throughput (*scopesim.effects.surface_list.SurfaceList* property), 108
- throughput (*scopesim.effects.ter_curves.FilterWheelBase* property), 111
- throughput (*scopesim.effects.ter_curves.TERCurve* property), 114
- throughput (*scopesim.optics.surface.PoorMansSurface* attribute), 143
- throughput (*scopesim.optics.surface.SpectralSurface* property), 144
- top_level_catch() (in module *scopesim.utils*), 167
- TopHatFilterCurve (class in *scopesim.effects.ter_curves*), 114
- TopHatFilterWheel (class in *scopesim.effects.ter_curves*), 114
- total_wfe (*scopesim.effects.psf.NonCommonPathAberration* property), 96
- trace_id (*scopesim.effects.spectral_trace_list_utils.SpectralTrace* property), 105
- trace_id (*scopesim.optics.fov.FieldOfView* property), 120
- Transform2D (class in *scopesim.effects.spectral_trace_list_utils*), 105
- transmission (*scopesim.optics.surface.SpectralSurface* property), 144
- transverse_distance() (in module *scopesim.utils*), 167
- U**
- UnequalBinnedImage (class in *scopesim.effects.electronic*), 86
- uniform_illumination() (in module *scopesim.source.source_templates*), 158
- uniform_source() (in module *scopesim.source.source_templates*), 159
- unify() (in module *scopesim.utils*), 167
- unimagefrom_table() (in module *scopesim.utils*), 167
- update() (*scopesim.base_classes.PoorMansHeader* method), 161
- update() (*scopesim.commands.user_commands.UserCommands* method), 71
- update() (*scopesim.effects.effects.Effect* method), 81
- update() (*scopesim.effects.psf.GaussianDiffractionPSF* method), 96
- update() (*scopesim.optics.optical_train.OpticalTrain* method), 139
- update() (*scopesim.optics.optics_manager.OpticsManager* method), 142

[update_logging\(\)](#) (in module *scopesim.utils*), [167](#)
[update_meta\(\)](#) (*scopesim.effects.spectral_trace_list.SpectralTraceList* method), [101](#)
[update_transmission\(\)](#) (*scopesim.effects.ter_curves.PupilTransmission* method), [111](#)
[UserCommands](#) (class in *scopesim.commands.user_commands*), [69](#)

V

[val2pix\(\)](#) (in module *scopesim.optics.image_plane_utils*), [135](#)
[validate\(\)](#) (*scopesim.effects.data_container.DataContainer* method), [77](#)
[validate_source_input\(\)](#) (in module *scopesim.source.source_utils*), [160](#)
[values\(\)](#) (*scopesim.base_classes.PoorMansHeader* method), [161](#)
[vega_spectrum\(\)](#) (in module *scopesim.source.source_templates*), [159](#)
[Vibration](#) (class in *scopesim.effects.psfs*), [97](#)
[view\(\)](#) (*scopesim.optics.fov.FieldOfView* method), [120](#)
[view\(\)](#) (*scopesim.optics.image_plane.ImagePlane* method), [129](#)
[view_fov\(\)](#) (in module *scopesim.optics.optical_train*), [140](#)
[volume\(\)](#) (*scopesim.optics.fov.FieldOfView* method), [120](#)

W

[walk\(\)](#) (in module *scopesim.reports.rst_utils*), [148](#)
[wavelength](#) (*scopesim.effects.psfs.AnisocadoConstPSF* property), [95](#)
[wavelength](#) (*scopesim.optics.fov.FieldOfView* property), [120](#)
[wavelength](#) (*scopesim.optics.surface.SpectralSurface* property), [144](#)
[waverange](#) (*scopesim.optics.fov.FieldOfView* property), [120](#)
[waveset](#) (*scopesim.optics.fov.FieldOfView* property), [120](#)
[wfe2gauss\(\)](#) (in module *scopesim.effects.psf_utils*), [94](#)
[wfe2strehl\(\)](#) (in module *scopesim.effects.psf_utils*), [94](#)
[write_header\(\)](#) (*scopesim.optics.optical_train.OpticalTrain* method), [139](#)
[write_report\(\)](#) (in module *scopesim.utils*), [167](#)
[write_string\(\)](#) (*scopesim.optics.fov_manager.FovVolumeList* method), [123](#)
[write_string\(\)](#) (*scopesim.optics.optical_element.OpticalElement* method), [137](#)
[write_string\(\)](#) (*scopesim.optics.optics_manager.OpticsManager* method), [142](#)

X

[xilamxy_fit\(\)](#) (in module *scopesim.effects.spectral_trace_list_utils*), [106](#)
[XiLamImage](#) (class in *scopesim.effects.spectral_trace_list_utils*), [106](#)
[xy2xilam_fit\(\)](#) (in module *scopesim.effects.spectral_trace_list_utils*), [106](#)

Z

[z_order_in_range\(\)](#) (in module *scopesim.effects.effects_utils*), [82](#)
[zendist2airmass\(\)](#) (in module *scopesim.utils*), [167](#)
[zenith_dist_to_airmass\(\)](#) (in module *scopesim.utils*), [168](#)
[zero_mag_flux\(\)](#) (in module *scopesim.effects.ter_curves_utils*), [117](#)