desimodel Documentation

Release 0.17.0

DESI

Nov 17, 2022

Contents

1 Introduction 1

2 Contents 3

3 Indices and tables 43

Python Module Index 45

Index 47

CHAPTER 1

Introduction

This is the documentation for desimodel.

CHAPTER 2

Contents

2.1 desimodel Release Notes

2.1.1 0.18.0 (unreleased)

- No changes yet.

2.1.2 0.17.0 (2021-09-19)

- On focal plane sync, update KPNO-wide default DESIMODEL/data checkout. (PR #151).
- Handle swapped fibers 3429, 3402 (PR #152).
- Remove deprecated focal plane generation routine, update associated docs.

2.1.3 0.16.0 (2021-06-25)

- Update `etc/desimodel_sync_kpno_cron.sh` for automatically syncing the focalplane model to the latest DB dump (PR #148).
- Add fiberfrac to PSF seeing conversions (PR #149).
- Move from positioner exclusion yaml format to much faster json format (PR #150).

2.1.4 0.15.0 (2021-04-19)

Code in GitHub:

- Use UTC time everywhere in the focalplane model (PR #147). This is backwards compatible with old files, but new FP models will not be readable by previous code tags.

Data in svn:

- DB sync 2021-04-03T23:53:23 appended to `desi-state-2021-03-17T23:20:01.ecsv`.
- DB sync 2021-04-10T20:00:39 appended to `desi-state_2021-03-17T23:20:01.ecsv`.
- DB sync 2021-04-13T20:00:30 appended to `desi-state_2021-03-17T23:20:01.ecsv`.

2.1.5 0.14.2 (2020-03-31)

Data changes to svn, no code changes:

- Added LYA TSNR2 templates.
- Focalplane model updated 2021-03-17.

2.1.6 0.14.1 (2021-03-18)

- Add fastfiberacceptance code originally in specsim (PR #145).

2.1.7 0.14.0 (2021-02-10)

- Code (in GitHub):
  - travis test fixes for old astropy (PR #141).
  - move command line scripts from svn to git (PR #142).
  - add option to exclude petals from restricted reach (PR #144).
- Data (in svn):
  - added Template Signal-to-Noise (TSNR) ensembles
  - added pre-calculated Noise Equivalent Area (NEA) from PSF model
  - Corrected restricted reach focalplane model (@135002)
  - Focal plan model with full reach for petal locs 0,2,4,5 (@135236)

2.1.8 0.13.1 (2020-08-03)

- New tag of data+code since data had been erroneously pre-tagged 0.13.0

2.1.9 0.13.0 (2020-08-03)

- Fix py3.8 syntax warnings (PR #140).
- Fix corner cases in generating and using focalplane models (PR #139).
- Use DESI-5501 (as built) instead of DESI-334 (design) for spectrograph throughput (PR #137).

2.1.10 0.12.0 (2020-03-13)

- update platescale to as-built DESI-4037v5 (PR #136).
- update desi-focalplane model for limited phi range 20200306 (svn data).
- fix bug in generating focalplane model from old fiberpos files (PR #139).
- use >= not > when comparing runtime to focalplane model #139).

2.1.11 0.11.0 (2020-03-13)

- Updated `data/footprint/desi-tiles.fits` and `desi-healpix-weights.fits` with new dither pattern; see DESI-0717. Layers 0=GRAY, 1-4=DARK instead of 0-3=DARK, 4=GRAY. (PR #135).
- Update documentation for *`desimodel.io`*; use *`desimodel.io.findfile()`* consistently throughout the module (PR #133).
- Update README file and Travis tests (PR #132).
- Include S (curved focal surface arc length) vs. R (CS5 xy radius) table from DESI-0530 (PR #130 and #135).

2.1.12 0.10.3 (2019-12-20)

- Pass multiple sets of exclusion polygons (PR #128).
- Propagate existing focalplane state to new focalplanes (PR #129).

2.1.13 0.10.2 (2019-10-31)

- Improve focalplane creation code (PR #127).

2.1.14 0.10.1 (2019-10-17)

- Workaround upstream bugs in positioner locations (PR #118).
- Added *desimodel.focalplate.fieldrot.field_rotation_angle* with field rotation CS5 vs. ICRS due to precession (PR #119).
- Add focalplane model documentation (PR #125).

2.1.15 0.10.0 (2019-09-25)

- Store petal and gfa keepouts in the focalplane model (PR #112).
- When generating a focalplane, check for device locations assigned to the same slitblock and fiber (PR #113).
- Fix support for `datetime.isoformat()` in Python 3.5 (PR #114).
- Update tests and documentation to be consistent with latest desiutil versions (PR #115).

2.1.16 0.9.12 (2019-08-09)

- Support for time-varying focal plane state (*e.g.* broken fibers) (PR #105).
- Documentation about CI weather *versus* model (PR #107).
- Fix `find_points_radec()` for scipy 1.3 (PR #109).
- Replace deprecated `yaml.load` with `yaml.safe_load` (PR #110).

2.1.17 0.9.11 (2019-05-30)

- Added data/footprint/ci-tiles-v7.fits, data/focalplane/ci-corners.ecsv to svn and docs to GitHub (PR #103).

2.1.18 0.9.10 (2019-02-28)

- `io.load_tiles(tilesfile)` warns if local copy exists, but `DESIMODEL` version wins (PR #98 and #101).
- Update default tile radius (max radius, not typical outer pos radius) (PR #102).

2.1.19 0.9.9 (2018-09-27)

- Change default healpy pixel overlap factor from 4 to 128 (PR #93).

2.1.20 0.9.8 (2018-09-05)

- Implement `dome_close_fractions()` to replay daily Mayall weather history (PR #92).
- Run tests using new svn branch test-0.9.8.
- Bug fix for GFA target selection when no targets overlap a GFA (PR #91).

2.1.21 0.9.7 (2018-07-30)

- Create DESI-3977 in doc/tex/desi3977/ to track ELG SNR with changes to the DESI model.
- Add accompanying notebook doc/nb/ELG_SNR.ipynb.

2.1.22 0.9.6 (2018-07-18)

- Update data and associated code to reflect changes in DESI-347-v13 (PR #89): * `data/throughput/thru-[brz].fits`: new corrector coatings. * `data/throughput/DESI-0347_blur.ecsv`: new achromatic blurs. * `data/desi.yaml`: new read noise and dark currents. * `data/focalplane/gfa.ecsv`: replace `RADIUS_MM` with `S`. * `data/throughput/DESI-0347_static_[123].fits`: replace random offset files (RMS=10.886um) with static offset files (RMS=8.0um).
- Use a new svn branch test-0.9.6 for travis tests (was test-0.9.3).

2.1.23 0.9.5 (2018-06-27)

- Increase test coverage, especially for *desimodel.trim* (PR #82).
- Reorganize `desimodel.focalplane` and add more GFA selection code (PR #85).
- Allow an environment variable in the tilesfile filename (PR #87).

2.1.24 0.9.4 (2018-03-29)

- Download script will create `INSTALL_DIR` if it doesn't exist (PR #80).

2.1.25 0.9.3 (2018-03-14)

- Fix some installation bugs, and update to latest versions on various dependencies (PR #77).
- Ensure that desimodel tests are compatible with Astropy 2 and 3, and with other DESI packages (PR #78).
- Add `footprint/desi-healpix-weights.fits` and `throughput/galsim-fiber-acceptance.fits` to the trimmed test data set (PR #79).

2.1.26 0.9.2 (2018-02-27)

- Update LyA S/N calculation (PR #73).
- Optionally use an input pixel weight map in *load_pixweight()* (PR #74).

2.1.27 0.9.1 (2017-11-10)

- Extracts wavelength coverage from specpsf files into params dictionary (PR #68).
- Added *program2pass()* and *pass2program()* to convert between tiling integer pass number and string program name (PR #67).

2.1.28 0.9.0 (2017-09-19)

- Added desimodel.focalplane.radec2xy, which converts RA, Dec coordinates to x, y coordinates on the focal plane, which accepts vector inputs.
- Added desimodel.focalplane.on_gfa() and its respective helper functions to check if a target is on a GFA of arbitrary telescope pointing
- Added desimodel.focalplane.on_tile_gfa() to check return a list of indices of targets on a specific tile
- Added desimodel.focalplane.get_gfa_targets() to return a table with added columns GFA_LOC and TILEID that consists of all targets on any GFA on any tile satisfying a minimum flux in the r-band.
- Unittests for the desimodel.focalplane functions were updated accordingly.
- Added desimodel.footprint.find_points_in_tel_range() to return a list of indices withnin a radius of an arbitray telescope pointing, unaware of tiles (Added respective unittest)
- Adds desimodel.focalplane.fiber_area_arcsec2()
- Updates tests to work with trimmed data subset

2.1.29 0.8.0 (2017-08-07)

- Add new weather module to specify assumed atmospheric seeing and transparency distributions at KPNO, with accompanying DESI-doc and jupyter notebook.
- Remove seeing module, which is superseded by new weather module.
- Added *desimodel.footprint.pixweight()* in `desimodel.footprint` to create an array of what fraction of every HEALPixel at a given nside overlaps the DESI footprint
- Also added *desimodel.footprint.tiles2fracpix()* to estimate which HEALPixels overlap the footprint edges
- Added *desimodel.io.load_pixweight()* in `desimodel.io` to load the array created by *desimodel.footprint.pixweight()* and resample it to any HEALPix nside
- Modified path to Lya SNR spectra files used in desi_quicklya.py, used in Lya Fisher forecast.
- Added desimodel.inputs.build_gfa_table and its helper functions to write a .ecsv file for GFA data
- Added desimodel.io.load_gfa to return the GFA data table
- Added desimodel.focalplane.xy2radec, which converts x,y coordinates on the focal plane to RA, Dec coordinates
- don't print warnings in desimodel.io if specter isn't installed

2.1.30 0.7.0 (2017-06-15)

- Added desimodel.footprint.tiles2pix and .pix2tiles for mapping healpix to DESI tiles.
- fixed psf-quicksim.fits units to be astropy-friendly
- added *desimodel.io.load_target_info()*

2.1.31 0.6.0 (2017-03-27)

- Add desimodel.seeing module with functions that model the expected DESI zenith seeing at 6355A, with an accompanying jupyter notebook.
- Altered xy offset RMS calculation in focalplane.py to scale the distribution RMS rather than the sample standard deviation.
- Update focal plane to positioner mapping
- z-channel 250 um CCD instead of 500 um CCD
- Update DocDB -> desimodel update method for fiberpos and throughput

2.1.32 0.5.1 (2016-12-01)

- By default, desimodel.io.load_tiles now excludes PROGRAM=EXTRA layers
- Adds desi-tiles.* tests

2.1.33 0.5.0 (2016-11-21)

- Moved test of focalplane code into the actual test suite.
- Preparing for Python 3.
- Changed default svn version to trunk and added error handling to **`install_desimodel_data`**.
- Update template module file to reflect DESI+Anaconda infrastructure.
- Add code to generate random centroid offsets in `desimodel.focalplane`.
- Add jupyter notebook documenting new throughput files of PR#29.
- Use Astropy-recommended method of reading FITS data tables.
- Remove reference to Travis scripts in MANIFEST.in.

2.1.34 0.4.5 (2016-07-15)

- Fixed a minor bug that made the help message for **`install_desimodel_data`** garbled.
- Add additional files to lightweight test data to work with quickgen

2.1.35 0.4.4 (2016-03-15)

- Allow **`desiInstall`** to download and install the data from svn.
- No changes to data in svn.

2.1.36 0.4.3 (2016-03-10)

- "First" post-separation tag.
- Added *`desimodel.trim.trim_data()`* for trimming a data directory into a lightweight version for testing.
- svn data includes targets.dat: preliminary numbers for MWS and BGS densities (Still waiting upon supporting technote).

2.1.37 0.4.2 (2016-02-04)

- Improved svn download instructions in the README file.
- Changes to data on svn side
  - updated desi.yaml with dark vs. bright exptime
  - updated targets.dat to include MWS placeholders
- *`desimodel.io.load_desiparams()`* adds 'exptime' -> 'exptime_dark' key for temporary backwards compatibility
- Removed deprecated fibers module
- Use ci-helpers to handle most of the dirty work of Travis build scripts.
- Make specter import errors more verbose.

2.1.38 0.4.1 (2016-01-25)

- Last tag prior to separating desimodel into code (GitHub) and data (svn) repositories.
- pip install support (BAW).
- Replace fitsio dependency with astropy.io.fits.

2.1.39 0.4 (2015-12-14)

- Added tile file for the bright time survey.

2.1.40 0.3.8 (2015-10-30)

- Adds python io library (SJB).

2.1.41 0.3.7 (2015-04-16)

- Tag to support dogwood production (SJB).

2.1.42 0.3.6 (2015-01-30)

- Adds `desimodel.focalplane.FocalPlane.xy2radec()` from Jaime (SJB).

2.1.43 0.3.5 (2014-12-28)

data/targets/targets.dat added fractions for sky and stdstar fibers (SJB).

py/desimodel/focalplane.py bug fixes for transformations (SJB).

2.1.44 0.3.4 (2014-09-23)

- Fix a simple import error (BAW).

2.1.45 0.3.3 (2014-09-23)

- Fix a simple version error (BAW).

2.1.46 0.3.2 (2014-09-23)

- Change how version is set (BAW).
- Updated target numbers.

2.1.47 0.3.1 (2014-07-23)

- Also updated quicksim sn-spec* file output, using IDL version which is slightly more optimistic than the python version (diff is dark current?) (SJB).

2.1.48 0.3 (2014-07-23)

- Updated throughput files for real.
- Added initial "compare_versions.py" script to make it easier to visualize differences in versions. This script should grow as various parameters change; right now it only makes a thoughput difference plot (SJB).
- Updated throughput files from 0334v3 (spectro) and 0347v5 (system throughput) Correction: thoughput files didn't make it into that change (SJB, 2014-07-08).
- Updated psf-b.fits and psf-quicksim.fits to match new npix_y for blue STA/ITL CCDs (SJB, 2014-07-08).

2.1.49 0.2 (2014-07-08)

2014-07-07 SJB

- Added ELG spectrum with continuum and multiple emission lines

2014-07-07 David Kirkby

Python quicksim

- add readnoise contributions in quadrature during the downsampling
- Refactor for speed, results now named ndarray, updated plots
- Allow different base directories

2014-07-02 DJS

- Put sky back to dimmer UVES sky model

2.1.50 0.1 (2014-07-01)

2014-06-29 SJB

- Extended fiberloss range from 3500-10000 instead of 3600-10000
- Added data/throughput/fiberloss-qso.dat (same as fiberloss-star.dat)

2014-06-27 SJB

- Updated data/focalplane/platescale.txt with latest from DESI-0329v14. This includes a new "theta" column.
- Updated desi.yaml from DESI-0347v4. This removes the FWHM and wavemin/max params which are not derived quantities associated with the PSFs.
- Updated throughput files with new numbers from DESI-0347v4.
- Updated spectrograph throughput files with new numbers from DESI-0334v2.
- Updated py/fiberloss.py -> bin/fiberloss.py . Biggest change is ELG half light radius 0.35" -> 0.45" which drops us below 7-sigma.
- Updated data/throughput/fiberloss-*.dat files with calculation based upon fiberloss.py

- **bin/psf2quicksim.py extracted PSF parameters needed for quicksim.**
  - pro/desi_quicksim.pro updated, but it still treats FWHM as constant rather than wavelength dependent.
  - python quicksim will be broken until it is updated to use new inputs.
- Reorganized data/inputs/throughput/
- spots2psf.py: leftover spot mirroring bug removed, PSFs updated

2014-06-12 SJB

- Updated throughputs to not double count central obscuration.
- Updated PSF files to remove throughputs to avoid possible inconsistency.
- Added wavemin_all, wavemax_all to desi.yaml with min/max wavelength seen by all spectra

2014-06-06 SJB

- Updated CCD pixel dimensions and regenerated PSFs to match.

2.2 The desimodel package/API

2.2.1 desimodel

A package for providing machine-readable data about the DESI focal plane and other hardware designs to simulations.

2.2.2 desimodel.focalplane.geometry

Dimensions and coordinate system transforms for the DESI focal plane.

class desimodel.focalplane.geometry.**FocalPlane**(*ra=0.0*, *dec=0.0*)

A class for modeling the DESI focal plane and converting between focal plane coordinates (in mm) and RA, Dec on the sky (in degrees). Provides utility functions for mapping which positioners cover which (RA, Dec) or (x, y) locations and vice versa.

Parameters

- **dec** (*ra,*) – Initialize DESI focal plane model with the telescope pointing at (*ra*, *dec*) in degrees.
- **NOTE** (*this class is deprecated (or should be further expanded), but*) –
- **not removing it yet in order to not arbitrarily break code that** (*I'm*) –
- **be using it.** (*might*) –

_check_radec(*ra*, *dec*)

Raise ValueError if RA or dec are out of bounds.

radec2pos(*ra*, *dec*)
Identify which positioners cover (*ra*, *dec*).

If *ra*, *dec* are floats, return an array of positioner IDs that cover it. The array could be empty if no positioner covers that location.

If *ra*, *dec* are numpy arrays, return a list of arrays. The ith element is an array of positioner IDs that cover (ra[i], dec[i]).

> **Warning:** This method is not implemented!

radec2xy(*ra*, *dec*)
Convert (RA, Dec) in degrees to (x, y) in mm on the focal plane given the current telescope pointing.

If RA and Dec are floats, returns a tuple (x, y) of floats If RA and Dec are numpy arrays, returns a tuple (x, y) of numpy arrays

Parameters **dec**(*ra,*) – Sky position.

Returns A tuple containing the (x, y) coordinates in mm.

Return type tuple()

set_tele_pointing(*ra*, *dec*)
Set telescope pointing to (RA, Dec) in degrees.

Parameters

- **ra** –
- **dec** (float) – Telescope pointing in degrees.

xy2pos(*x*, *y*)
Identify which positioners cover (*x*, *y*).

> **Warning:** This method is not implemented!

xy2radec(*x*, *y*)
Convert (x, y) in mm on the focal plane to (ra_object, dec_object) in degrees on the sky given the current telescope pointing towards (RA, Dec).

x, *y* must be floats. This function is vectorized in xy2radec(), which doesn't appear to exist.

Parameters **y**(*x,*) – Position on the focal plane in mm.

Returns Coordinates of object.

Return type tuple()

desimodel.focalplane.geometry.**_extrapolate_r_s**(*r*, *s*)
Utility function for xy2qs and qs2xy; returns new r, s with extrapolations to 0 and max(r)+10 mm.

desimodel.focalplane.geometry.**fiber_area_arcsec2**(*x*, *y*)
Returns area of fibers at (*x*, *y*) in arcsec^2.

desimodel.focalplane.geometry.**get_radius_deg**(*x*, *y*)
Returns the radius in degrees given *x*, *y* coordinates using the platescale data.

Parameters

- **x** (float) – The x coordinate in mm of a location on the focal plane

- **y** (float) – The y coordinate in mm of a location on the focal plane

Returns Radius corresponding to *x*, *y*.

Return type float

desimodel.focalplane.geometry.**get_radius_mm**(*theta*)
Returns an array of radii in mm given an array of radii in degrees using the platescale data relative to the center of the focal plane as (0,0). Supports scalar and vector inputs.

Parameters **theta** (float or array-like) – An array that represents the angle from the center of the focal plane.

Returns Radii in mm.

Return type float or array-like

desimodel.focalplane.geometry.**get_tile_radius_deg**()
Returns maximum radius in degrees covered by the outermost positioner.

desimodel.focalplane.geometry.**get_tile_radius_mm**()
Returns maximum radius in mm covered by the outermost positioner.

desimodel.focalplane.geometry.**qs2xy**(*q*, *s*)
angular q,s on curved focal surface -> focal tangent plane x,y

Parameters

- **q** – angle in degrees
- **s** – focal surface radial distance in mm

Returns (x, y) cartesian location on focal tangent plane in mm

Notes: (x,y) are in the "CS5" DESI coordinate system tangent plane to the curved focal surface. q is the radial angle measured counter-clockwise from the x-axis; s is the radial distance along the curved focal surface; it is *not* sqrt(x**2 + y**2). (q,s) are the preferred coordinates for the DESI focal plane hardware engineering team.

desimodel.focalplane.geometry.**radec2xy**(*telra*, *teldec*, *ra*, *dec*)
Returns arrays of the x, y positions of given celestial objects on the focal plane given an arbitrary telescope pointing in RA and Dec and arrays of the *ra* and *dec* of celestial objects in the sky.

Parameters

- **telra** (float) – The telescope's RA pointing in degrees.
- **teldec** (float) – The telescope's Dec pointing in degrees.
- **ra** (*array-like*) – An array of RA values for locations in the sky.
- **dec** (*array-like*) – An array of Dec values for locations in the sky.

Returns The x, y positions corrsponding to *ra*, *dec*.

Return type tuple

Notes

Implements the Haversine formula.

desimodel.focalplane.geometry.**xy2qs**(*x*, *y*)
Focal tangent plane x,y -> angular q,s on curved focal surface

Parameters **y** (*x,*) – cartesian location on focal tangent plane in mm

Returns (q, s) where q=angle in degrees; s=focal surface radial dist [mm]

Notes: (x,y) are in the "CS5" DESI coordinate system tangent plane to the curved focal surface. q is the radial angle measured counter-clockwise from the x-axis; s is the radial distance along the curved focal surface; it is *not* sqrt(x**2 + y**2). (q,s) are the preferred coordinates for the DESI focal plane hardware engineering team.

desimodel.focalplane.geometry.**xy2radec**(*telra*, *teldec*, *x*, *y*)
Returns the new RA and Dec of an *x*, *y* position on the focal plane in the sky given an arbitrary telescope pointing in RA and Dec.

Parameters

- **telra** (float) – The telescope's RA pointing in degrees.
- **teldec** (float) – The telescope's Dec pointing in degrees.
- **x** (float) – The x coordinate in mm of a location on the focal plane
- **y** (float) – The y coordinate in mm of a location on the focal plane

Returns The RA, Dec corresponding to *x*, *y*.

Return type tuple

2.2.3 desimodel.footprint

Utility functions for working with the DESI footprint.

desimodel.footprint.**_embed_sphere**(*ra*, *dec*)
Embed *ra*, *dec* to a uniform sphere in three dimensions.

desimodel.footprint.**find_points_in_tiles**(*tiles*, *ra*, *dec*, *radius=None*)
Return a list of indices of points that are within each provided tile(s).

This function is optimized to query a lot of points with relatively few tiles.

radius is in units of degrees. The return value is an array of lists that contains the index of points that are in each tile. The indices are not sorted in any particular order.

if tiles is a scalar, a single list is returned.

default radius is from desimodel.focalplane.get_tile_radius_deg()

desimodel.footprint.**find_points_radec**(*telra*, *teldec*, *ra*, *dec*, *radius=None*)
Return a list of indices of points that are within a radius of an arbitrary telra, teldec.

This function is optimized to query a lot of points with a single telra and teldec.

radius is in units of degrees. The return value is a list that contains the index of points that are in each tile. The indices are not sorted in any particular order.

if tiles is a scalar, a single list is returned.

default radius is from desimodel.focalplane.get_tile_radius_deg()

Note: This is simply a modified version of find_points_in_tiles, but this function does not know about tiles.

desimodel.footprint.**find_tiles_over_point**(*tiles*, *ra*, *dec*, *radius=None*)
Return a list of indices of tiles that covers the points.

This function is optimized to query a lot of points. radius is in units of degrees. The return value is an array of list objects that are the indices of tiles that cover each point.

The indices are not sorted in any particular order.

if ra, dec are scalars, a single list is returned.

default radius is from desimodel.focalplane.get_tile_radius_deg()

desimodel.footprint.**get_tile_radec**(*tileid*)
Get the coordinates of a tile.

Parameters **tileid** (*int*) – ID of a tile.

Returns (ra, dec) in degrees for the requested *tileid*.

Return type tuple

Raises ValueError – If tileid is not in list of known tiles.

desimodel.footprint.**is_point_in_desi**(*tiles*, *ra*, *dec*, *radius=None*, *return_tile_index=False*)
If a point (*ra*, *dec*) is within *radius* distance from center of any tile, it is in DESI.

Parameters

- **tiles** (*Table-like*) – The output of *desimodel.io.load_tiles()*, or a similar Table.
- **ra** (*scalar or array-like*) – Right Ascension in degrees.
- **dec** (*scalar or array-like*) – Declination in degrees. The size of *dec* must match the size of *ra*.
- **radius** (*float, optional*) – Tile radius in degrees; if *None* use desimodel.focalplane.get_tile_radius_deg().
- **return_tile_index** (*bool, optional*) – If True, return the index of the nearest tile in tiles array.

Returns Return True if points given by *ra*, *dec* lie in the set of *tiles*.

Notes

This function is optimized to query a lot of points.

desimodel.footprint.**pass2program**(*tilepass*)
Converts integer tile pass number to string program name.

Parameters **tilepass** (*int or int array*) – tiling pass number.

Returns Program name for each pass (str or list of str).

desimodel.footprint.**pix2tiles**(*nside*, *pixels*, *tiles=None*, *radius=None*)
Returns subset of tiles that overlap the list of pixels.

Parameters

- **nside** (*int*) – HEALPix *nside*, 2**k where 0 < k < 30.
- **pixels** (*array-like*) – Array of integer pixels using nested numbering scheme.
- **tiles** (*Table-like, optional*) – Table-like with RA,DEC columns; or None to use all DESI tiles from *desimodel.io.load_tiles()*.
- **radius** (*float, optional*) – Tile radius in degrees; if *None* use desimodel.focalplane.get_tile_radius_deg().

Returns Table of tiles that cover these pixels.

TODO: add support for tiles as integers or list/array of integer TILEIDs.

desimodel.footprint.**pixweight**(*nside*, *tiles=None*, *radius=None*, *precision=0.01*, *outfile=None*, *outplot=None*)

Create an array of the fraction of each pixel that overlaps the passed tiles.

Parameters

- **nside** (*int*) – HEALPix *nside*, `2**k` where 0 < k < 30.
- **tiles** (*Table-like, optional*) – Table-like with RA,DEC columns; or `None` to use all DESI tiles from *desimodel.io.load_tiles()*.
- **radius** (*float, optional*) – Tile radius in degrees; if *None* use `desimodel.focalplane.get_tile_radius_deg()`.
- **precision** (*float, optional*) – Approximate precision at which to calculate the area of pixels that partially overlap the footprint in SQUARE DEGREES (*e.g.* 0.01 means precise to 0.01 sq. deg., or 36 sq. arcmin.). Lower numbers mean better precision.
- **outfile** (*str, optional*) – Write the pixel->weight array to the file passed as *outfile* (could be full directory path + file).
- **outplot** (*str, optional*) – Create a plot named *outplot* (pass a *name* for a plot in the current directory, a *full path* for a plot in a different directory). This is passed to matplotlib.pyplot's savefig routine.

Returns pixweight: An array of the weight for each pixel at the passed nside. The weight is the fracion of the pixel that overlaps the passed tiles: *WEIGHT=1* for the pixel is entirely contained in the tiles; *WEIGHT=0* for the pixel is entirely outside of the tiles; *0 < WEIGHT < 1* for a pixel that overlaps the tiles. The index of the array is the HEALPixel integer.

Notes

It is sufficient to create the weights at a suitably high nside, say nside=256 (0.052456 sq. deg. per pixel) as pixel numbers at lower nsides can be obtained by integer division by powers of 4, *e.g.* pix_@_nside_128 = pix@nside_256//4 and fractional weights at lower nsides are the mean of the 4 pixels at the higher nside *desimodel.io.load_pixweight()* can downsample the array to lower nsides.

desimodel.footprint.**program2pass**(*program*)

Convert string program name to tile passes for that program.

Parameters **program** (*str for str array*) – program name, *e.g.* DARK, BRIGHT, or GRAY.

Returns List of integer passes that cover that program, or list of lists if input was array-like.

desimodel.footprint.**radec2pix**(*nside*, *ra*, *dec*)

Convert *ra*, *dec* to nested pixel number.

Parameters

- **nside** (*int*) – HEALPix *nside*, `2**k` where 0 < k < 30.
- **ra** (*float or array*) – Right Accention in degrees.
- **dec** (*float or array*) – Declination in degrees.

Returns Array of integer pixel numbers using nested numbering scheme.

Notes

This is syntactic sugar around:

```
hp.ang2pix(nside, ra, dec, lonlat=True, nest=True)
```

but also works with older versions of healpy that didn't have *lonlat* yet.

desimodel.footprint.**tileids2pix**(*nside*, *tileids*, *radius=None*, *per_tile=False*)
Like *tiles2pix()*, but accept integer tileid or list of tileids instead of table of tiles.

desimodel.footprint.**tiles2fracpix**(*nside*, *step=1*, *tiles=None*, *radius=None*, *fact=128*)
Returns a sorted array of just the *fractional* pixels that overlap the tiles.

Parameters

- **nside** (*int*) – HEALPix *nside*, `2**k` where 0 < k < 30.
- **step** (*int, optional*) – The number of integration steps around the edges of a HEALPix pixel. `step=1` means just the pixel vertices. `step=2` means the vertices and the corners and the points halfway between the vertices. See also the HEALPix boundary document .
- **tiles** (*Table-like, optional*) – Table-like with RA,DEC columns; or `None` to use all DESI tiles from *desimodel.io.load_tiles()*.
- **radius** (*float, optional*) – Tile radius in degrees; if `None` use `desimodel.focalplane.get_tile_radius_deg()`.
- **fact** (*int, optional*) – Factor healpy uses to resolve pixel overlaps. When this is large there are fewer false positives at the expense of run time (although `fact=2**8` seems fast). Must be a power of 2.

Returns Integer array of pixel numbers that cover these tiles, *excluding pixels that fully overlap the tiles* (*i.e.*, just pixels that *partially* overlap the tiles). The integers are sorted.

Notes

There are potentially malicious cases where a pixel just brushes a tile, such that there is a very small area where the pixel overlaps the tile. To guard against these case, call this function with progressively larger step values until it converges.

desimodel.footprint.**tiles2pix**(*nside*, *tiles=None*, *radius=None*, *per_tile=False*, *fact=128*)
Returns sorted array of pixels that overlap the tiles.

Parameters

- **nside** (*int*) – HEALPix *nside*, `2**k` where 0 < k < 30.
- **tiles** (*array-like or Table-like, optional*) – Integer tile IDs, or `None` to use all DESI tiles from *desimodel.io.load_tiles()*.
- **radius** (*float, optional*) – tile radius in degrees; if `None` use `desimodel.focalplane.get_tile_radius_deg()`.
- **per_tile** (*bool, optional*) – If `True`, return a list of arrays of pixels per tile.
- **fact** (*int, optional*) – Factor healpy uses to resolve pixel overlaps. When this is large there are fewer false positives at the expense of run time (although `fact=2**8` seems fast). Must be a power of 2.

Returns Integer array of pixel numbers that cover these tiles; or if per_tile is *True*, returns list of arrays such that `pixels[i]` is an array of pixel numbers covering `tiles[i]`.

2.2.4 desimodel.inputs

Code for working with raw input data to the DESI model data.

2.2.5 desimodel.inputs.docdb

Utility functions for working with DocDB files.

`desimodel.inputs.docdb.`**`_auth`**(*machine='desi.lbl.gov'*)
Get authentication credentials.

`desimodel.inputs.docdb.`**`_xls_col2int`**(*col*)
Convert column string name to index, starting at 0

e.g. A -> 0, B -> 1, ... Z -> 25, AA -> 26, AB -> 27

`desimodel.inputs.docdb.`**`download`**(*docnum*, *docver*, *filename*, *outdir=None*, *overwrite=False*)
Downloads and writes outdir/DESI-{docnum}v{docver}-{filename}

Parameters

- **`docnum`** – integer DocDB number
- **`docver`** – integer version number
- **`filename`** – string filename within that DocDB entry

Options: outdir: output directory; default $DESIMODEL/data/inputs/docdb/ overwrite: overwrite pre-existing file

Returns path to output file written

Notes

- only supports python3
- creates outdir if needed
- prepends DESI-{docnum}v{docver} to {filename} even if filename already starts with that (in DocDB, some do and some don't...)

`desimodel.inputs.docdb.`**`xls_read_col`**(*filename*, *sheetname*, *column*, *firstrow*, *lastrow*, *dtype=None*)
Read Excel file column from firstrow to lastrow

Parameters

- **`filename`** (*str*) – Excel filename
- **`sheetname`** (*str*) – sheet name within the filename
- **`column`** (*str*) – Excel-style column string, e.g. 'A', 'B', or 'AC'
- **`firstrow`** (*int*) – 1-indexed first row to include
- **`lastrow`** (*int*) – 1-indexed last row to include

Options: dtype: convert output to this numpy dtype

Returns numpy array of data

Example

B5:B10 -> column='B', firstrow=5, lastrow=10 -> length 6 array

desimodel.inputs.docdb.**xls_read_row**(*filename*, *sheetname*, *rownum*, *firstcol*, *lastcol*, *dtype=None*)

Read Excel file row from firstcol to lastcol

Parameters

- **filename** (*str*) – Excel filename
- **sheetname** (*str*) – sheet name within the filename
- **rownum** (*int*) – 1-indexed row to read
- **firstcol** (*str*) – Excel-style column name, e.g. 'A', 'B', or 'AC'
- **lastcol** (*str*) – last column to include

Options: dtype: convert output to this numpy dtype

Returns numpy array of data

Example

B5:D5 -> rownum=5, firstcol='B', lastcol='D' -> length 3 array

2.2.6 desimodel.inputs.fiberpos

Utilities for updating positioner to fiber mapping.

desimodel.inputs.fiberpos.**update**(*testdir=None*, *seed=2*)

Update positioner to fiber number mapping from DocDB

Options:

testdir: if not None, write files here instead of $DESIMODEL/data/footprint/fiberpos*

seed: integer random number seed for randomization within a cartridge

Writes testdir/fiberpos* or $DESIMODEL/data/focalplane/fiberpos*

desimodel.inputs.fiberpos.**write_text_fiberpos**(*filename*, *fiberpos*)

Writes a fiberpos table to filename, maintaining backwards compatibility with the original fiberpos.txt format

Parameters

- **filename** – output file name string
- **fiberpos** – astropy Table of fiber positions

2.2.7 desimodel.inputs.focalplane

Utilities for constructing a focalplane model.

desimodel.inputs.focalplane.**create**(*testdir=None*, *posdir=None*, *fibermaps=None*, *petalloc=None*, *startvalid=None*, *fillfake=False*, *fakeoffset=False*, *fakefiberpos=False*, *reset=False*)

Construct DESI focalplane and state files.

This function gathers information from the following sources:

- Petal verification files on DocDB
- Positioner device configuration files (e.g. from svn).
- DESI-0530, to get the mapping from device ID to device type as well as the nominal device X/Y offsets on petal 0 (for fillfake option).
- Exclusion configobj files in $DESIMODEL/data/focalplane.

Parameters

- **testdir** (*str*) – Override the output directory for testing.
- **posdir** (*str*) – Directory containing the many positioner conf files. If None, simulate identical, nominal positioners. A None value will force fillfake=True.
- **fibermaps** (*list*) – Override list of tuples (DocDB number, DocDB version, DocDB csv file) of where to find the petal mapping files.
- **petalloc** (*dict*) – Mapping of petal ID to petal location.
- **startvalid** (*str*) – The first time when this focalplane model is valid. ISO 8601 format string.
- **fillfake** (*bool*) – If True, fill missing device locations with fake positioners with nominal values for use in simulations.
- **fakeoffset** (*bool*) – If True, artificially sets the theta / phi angle offsets to zero. This replicates the behavior of legacy fiberassign and should only be used for testing.
- **fakefiberpos** (*bool*) – If True, ignore the real fibermaps and load the old fiberpos file to get the mapping. Only useful for testing.
- **reset** (*bool*) – If True, ignore all previous focalplane models and start with all positioners "good". Default propagates the state of most recent model, after verifying that the positioners are the same.

Returns None

desimodel.inputs.focalplane.**devices_from_fiberpos**(*fp*)

Populate focalplane properties from a fiberpos file.

This is only used for testing consistency with the previous fiberpos files. It should not be used for work with the real instrument. The focalplane properties are modified in place.

Parameters **fp** (*dict*) – The focalplane dictionary.

Returns None

desimodel.inputs.focalplane.**devices_from_files**(*fp*, *posdir=None*, *fillfake=False*, *fakeoffset=False*, *fibermaps=None*)

Populate focalplane properties from information in files.

This populates the focalplane with device information gathered from the "pos_settings" files in svn and from the "Petal verification" files on DocDB.

The focalplane dictionary is modified in place.

Parameters

- **fp** (*dict*) – The focalplane dictionary.
- **posdir** (*str*) – Directory containing the many positioner conf files.
- **fillfake** (*bool*) – If true, fill missing POS and ETC locations with a fake nominal positioner.
- **fakeoffset** (*bool*) – If true, use theta / phi offsets that matched very old versions of fiberassign.
- **fibermaps** (*list*) – (optional) Override list of tuples (DocDB number, DocDB version, DocDB csv file) of where to find the petal mapping files.

Returns None

2.2.8 desimodel.inputs.throughput

Utilities for updating throughput model.

desimodel.inputs.throughput.**get_waveminmax**(*psffile*)
return wmin, wmax as taken from the header of a PSF file, e.g. $DESIMODEL/data/specpsf/psf-b.fits

desimodel.inputs.throughput.**load_fiberinput**(*filename*)
Load fiberinput as calculated by fiberloss.py

Parameters **filename** – fiberloss input file, e.g. $DESIMODEL/data/throughput/fiberloss-elg.dat

Returns InterpolatedUnivariateSpline instance.

desimodel.inputs.throughput.**load_spec_throughputs**(*filenames*, *columns='ABCD'*, *first_row=2*, *last_row=647*)
Loads spectrograph*CCD throughputs from DESI-5501 excel files.

Parameters **filenames** – list of per-spectrograph filenames.

Returns arrays of wavelength in nm and throughput per spectrograph.

desimodel.inputs.throughput.**load_throughput**(*filename*, *specthru_row=95*, *thru_row=97*)
Load throughputs from DESI-0347, removing the spectrograph contributions which will be loaded separately from higher resolution data.

Parameters **filename** – DESI-0347 Excel file location

Returns (thruspine, xlsdata), where

thruspline: InterpolatedUnivariateSpline of thru vs. wave[Angstroms] xlsdata: tuple of (wave, totalthru, specthru)

Notes

- Alas, DESI-0347 doesn't fill in the final throughput for 3500 and 9950 Angstroms, even though the inputs are there.

desimodel.inputs.throughput.**update**(*testdir=None*, *desi347_version=16*, *desi5501_version=3*, *desi5501_KOSI=True*)

Update thru-[brz].fits from DESI-0347 and DESI-0344

Parameters

- **testdir** – If not None, write files here instead of standard locations under $DESIMODEL/data/
- **desi347_version** – version of DESI-347 to use
- **desi5501_version** – version of DESI-5501 to use
- **[bool]** (*desi5501_KOSI*) – use KOSI throughput measurements in 5501

2.2.9 desimodel.install

Install data files not handled by pip install.

desimodel.install.**assert_svn_exists**()

Assert svn command exists and raise an informative error if not

desimodel.install.**default_install_dir**()

Return the default install directory. Assumes this file lives in a 'site-packages' directory.

Returns The path to the install directory.

Return type str

desimodel.install.**install**(*desimodel=None*, *version=None*)

Primary workhorse function.

Parameters

- **desimodel** (str, optional) – Allows the install directory to be explicitly set.
- **version** (str, optional) – Allows the desimodel version to be explicitly set.

Raises RuntimeError – Standard error output from svn export command when status is non-zero.

desimodel.install.**main**()

Entry point for the **install_desimodel_data** script.

Returns An integer suitable for passing to sys.exit().

Return type int

desimodel.install.**svn_export**(*desimodel_version=None*)

Create a **svn export** command suitable for downloading a particular desimodel version.

Parameters **desimodel_version** (str, optional) – The version X.Y.Z to download, trunk, or something of the form branches/... Defaults to trunk.

Returns A **svn** command in list form, suitable for passing to subprocess.Popen.

Return type list

2.2.10 desimodel.io

I/O utility functions for files in desimodel.

desimodel.io.**datadir**()
Returns location to desimodel data.

If set, DESIMODEL overrides data installed with the package.

desimodel.io.**findfile**(*filename*)
Return full path to data file $DESIMODEL/data/filename.

Parameters **filename** (str) – Name of the file, relative to the desimodel data directory.

Returns The full path.

Return type str

Notes

This is a precursor for a potential future refactor where desimodel data would be installed with the package and DESIMODEL would become an optional override.

desimodel.io.**load_desiparams**()
Returns DESI parameter dictionary loaded from $DESIMODEL/data/desi.yaml.

Returns The parameters read from the YAML file.

Return type dict

desimodel.io.**load_deviceloc**()
Returns a table from $DESIMODEL/data/focalplane/fiberpos-all.fits.

Returns The data from the FITS file, with columns converted to uppercase.

Return type Table

desimodel.io.**load_fiberpos**()
Returns fiberpos table from $DESIMODEL/data/focalplane/fiberpos.fits.

Returns The data from the FITS file, sorted by FIBER.

Return type Table

desimodel.io.**load_focalplane**(*time=None*)
Load the focalplane state that is valid for the given time.

Parameters **time** (datetime) – The time to query with explicit timezone. Default to current time (now()) with timezone UTC.

Returns A tuple of (FP layout, exclusion polygons, state, time string). The FP layout is a Table. The exclusion polygons are a dictionary indexed by names that are referenced in the state. The state is a Table. The time string is the resulting UTC ISO format time string for the creation date of the FP model.

Return type tuple

desimodel.io.**load_gfa**()
Returns GFA table from $DESIMODEL/data/focalplane/gfa.ecsv.

Returns The data from the ECSV file.

Return type Table

desimodel.io.**load_pixweight**(*nside*, *pixmap=None*)
Loads $DESIMODEL/data/footprint/desi-healpix-weights.fits.

Parameters

- **nside** (int) – After loading, the array will be resampled to the passed HEALPix *nside*.
- **pixmap** (FITS_rec, optional) – Input pixel weight map, already read from a weights file.

Returns HEALPix weight map for the DESI footprint at the requested *nside*.

Return type Weight

desimodel.io.**load_platescale**()
Loads platescale.txt.

Returns The data table read from the file.

Return type recarray

Notes

The returned object has these columns:

radius Radius from center of focal plane [mm].

theta Radial angle that has a centroid at this radius [deg].

radial_platescale Meridional (radial) plate scale [um/arcsec].

az_platescale: Sagittal (azimuthal) plate scale [um/arcsec].

arclength: Unknown description.

desimodel.io.**load_psf**(*channel*)
Returns specter PSF object for the given channel 'b', 'r', or 'z'.

Parameters **channel** ({'b', 'r', 'z'}) – Spectrograph channel.

Returns A specter PSF object.

Return type PSF

desimodel.io.**load_target_info**()
Loads data/targets/targets.yaml and returns the nested dictionary.

This is primarily syntactic sugar to avoid end users constructing paths and filenames by hand (which *e.g.* broke when targets.dat was renamed to targets.yaml).

Returns The dictionary read from the YAML file.

Return type dict

desimodel.io.**load_throughput**(*channel*)
Returns specter Throughput object for the given channel 'b', 'r', or 'z'.

Parameters **channel** ({'b', 'r', 'z'}) – Spectrograph channel.

Returns A specter throughput object.

Return type Throughput

desimodel.io.**load_tiles**(*onlydesi=True*, *extra=False*, *tilesfile=None*, *cache=True*)
Return DESI tiles structure from $DESIMODEL/data/footprint/desi-tiles.fits.

Parameters

- **onlydesi** (bool, optional) – If True, trim to just the tiles in the DESI footprint.
- **extra** (bool, optional) – If True, include extra layers with PROGRAM='EXTRA'.

- **tilesfile** (str, optional) – Name of tiles file to load; or None for default. See Notes for details.
- **cache** (bool, optional) – If False, force reload of data from tiles file, instead of using cached values.

Returns The data table portion of the FITS file.

Return type FITS_rec

Raises FileNotFoundError – If the value of *tilesfile* does not exist.

Notes

Keyword-based environment variable expansion is performed on the *tilesfile* value, so *e.g.*:

```
tiles = load_tiles(tilesfile='{HOME}/my-tiles.fits')
```

will be expanded with the value of HOME.

If the parameter *tilesfile* is set, this function uses the following search method:

1. If the value includes an explicit path, even ./, use that file.
2. If the value does *not* include an explicit path, *and* the file name is identical to a file in $DESIMODEL/data/footprint/, use the file in $DESIMODEL/data/footprint/ and issue a warning.
3. If no matching file can be found at all, raise an exception.

desimodel.io.**reset_cache**()
Reset I/O cache.

2.2.11 desimodel.trim

Code for trimming desimodel/data into smaller files.

desimodel.trim.**inout**(*indir*, *outdir*, *filename*)
returns os.path.join(indir, filename) and .join(outdir, filename)

desimodel.trim.**rebin_image**(*image*, *n*)
rebin 2D array pix into bins of size n x n

New binsize must be evenly divisible into original pix image

desimodel.trim.**trim_data**(*indir*, *outdir*, *overwrite=False*)
Trim a $DESIMODEL/data directory into a lightweight version for testing.

Parameters

- **indir** (*str*) – A $DESIMODEL/data directory from svn.
- **outdir** (*str*) – Output data directory location.
- **overwrite** (*bool, optional*) – If True, remove *outdir* if it already exists.

desimodel.trim.**trim_focalplane**(*indir*, *outdir*)
copy everything in focalplane

desimodel.trim.**trim_footprint**(*indir*, *outdir*)
Copies subset of desi-tiles.fits and .ecsv but not .par. Also creates a corresponding version of desi-healpix-weights.fits.

desimodel.trim.**trim_inputs**(*indir*, *outdir*)
Don't copy any inputs

desimodel.trim.**trim_sky**(*indir*, *outdir*)
copy solarspec file as-is

desimodel.trim.**trim_specpsf**(*indir*, *outdir*)
trim specpsf files to be much smaller

desimodel.trim.**trim_spectra**(*indir*, *outdir*)
downsample spectra, and only a few of them

desimodel.trim.**trim_targets**(*indir*, *outdir*)
copy everything in targets/

desimodel.trim.**trim_throughput**(*indir*, *outdir*)
downsample throughput files

desimodel.trim.**trim_weather**(*indir*, *outdir*)
copy everything in weather/

2.2.12 desimodel.weather

Model of the expected weather conditions at KPNO during the DESI survey.

To generate a random time series of expected FWHM seeing in arcsecs and atmospheric transparency, use, for example:

```
n = 10000
dt = 300 # seconds
t = np.arange(n) * dt
gen = np.random.RandomState(seed=123)
seeing = sample_seeing(n, dt_sec=dt, gen=gen)
transp = sample_transp(n, dt_sec=dt, gen=gen)
```

The resulting arrays are randomly sampled from models of the 1D probability density and 2-point power spectral density derived from MzLS observations. See DESI-doc-3087 for details.

Used by `surveysim.weather` for simulations of DESI observing and survey strategy studies.

desimodel.weather.**_seeing_fit_model**(*x*)
Evalute the fit to MzLS seeing described in DESI-doc-3087.

desimodel.weather.**_seeing_psd**(*freq*)
Evaluate the 'chi-by-eye' fit of the seeing PSD described in DESI-doc-3087.

desimodel.weather.**_transp_psd**(*freq*)
Evaluate the 'chi-by-eye' fit of the transparency PSD described in DESI-doc-3087.

desimodel.weather.**dome_closed_fractions**(*start_date*, *stop_date*, *replay='Y2007*, *Y2008*, *Y2009*, *Y2010*, *Y2011*, *Y2012*, *Y2013*, *Y2014'*)
Return dome-closed fractions for each night of the survey.

Years can be replayed in any order. If the number of years to replay is less than the survey duration, they are repeated.

Parameters

- **start_date** (*datetime.date or None*) – Survey starts on the evening of this date. Use the `first_day` config parameter if None (the default).
- **stop_date** (*datetime.date or None*) – Survey stops on the morning of this date. Use the `last_day` config parameter if None (the default).

- **replay** (*str*) – Comma-separated list of years to replay, identified by arbitrary strings that must match column names in the DESIMODEL weather history.

Returns 1D array of N probabilities between 0-1, where N is the number of nights spanned by the start and stop dates.

Return type numpy array

desimodel.weather.**get_seeing_pdf**(*median_seeing=1.1*, *max_seeing=2.5*, *n=250*)

Return PDF of FWHM seeing for specified clipped median value.

Note that this is atmospheric seeing, not delivered image quality. The reference wavelength for seeing values is 6355A, in the r band, and the observed wavelength dependence in Dey & Valdes is closer to `lambda ** (-1/15)` than the `lambda ** (-1/5)` predicted by Kolmogorov theory. See DESI-doc-3087 for details.

Scales the clipped MzLS seeing PDF in order to achieve the requested median value. Note that clipping is applied before scaling, so the output PDF is clipped at scale * max_seeing.

Parameters

- **median_seeing** (*float*) – Target FWHM seeing value in arcsec. Must be in the range [0.95, 1.30].
- **max_seeing** (*float*) – Calculate scaled median using unscaled values below this value.
- **n** (*int*) – Size of grid to use for tabulating the returned arrays.

Returns Tuple (fwhm, pdf) that tabulates pdf[fwhm]. Normalized so that `np.sum(pdf * np.gradient(fwhm)) = 1`.

Return type tuple

desimodel.weather.**get_transp_pdf**(*n=250*)

Return PDF of atmospheric transparency.

Derived from MzLS observations, but corrected for dust accumulation and measurement error. See DESI-doc-3087 for details.

Parameters **n** (*int*) – Size of grid to use for tabulating the returned arrays.

Returns Tuple (transp, pdf) that tabulates pdf[transp]. Normalized so that `np.sum(pdf * np.gradient(transp)) = 1`.

Return type tuple

desimodel.weather.**sample_seeing**(*n_sample*, *dt_sec=180.0*, *median_seeing=1.1*, *max_seeing=2.5*, *gen=None*)

Generate a random time series of FWHM seeing values.

See DESI-doc-3087 for details. Uses *get_seeing_pdf()*, *_seeing_psd()* and *sample_timeseries()*.

Parameters

- **n_sample** (*int*) – Number of equally spaced samples to generate.
- **dt_sec** (*float*) – Time interval between samples in seconds.
- **median_seeing** (*float*) – See *get_seeing_pdf()*.
- **mex_seeing** (*float*) – See *get_seeing_pdf()*.
- **gen** (*np.random.RandomState or None*) – Provide an existing RandomState for full control of reproducible random numbers, or None for non-reproducible random numbers.

Returns 1D array of randomly generated samples.

Return type array

desimodel.weather.**sample_timeseries**(*x_grid*, *pdf_grid*, *psd*, *n_sample*, *dt_sec=180.0*, *gen=None*)

Sample a time series specified by a power spectrum and 1D PDF.

The PSD should describe the temporal correlations of whitened samples. Generated samples will then be unwhitened to recover the input 1D PDF. See DESI-doc-3087 for details.

Uses *whiten_transforms_from_cdf()*.

Parameters

- **x_grid** (*array*) – 1D array of N increasing grid values covering the parameter range to sample from.
- **pdf_grid** (*array*) – 1D array of N increasing PDF values corresponding to each x_grid. Does not need to be normalized.
- **psd** (*callable*) – Function of frequency in 1/days that returns the power-spectral density of whitened temporal fluctations to sample from. Will only be called for positive frequencies. Normalization does not matter.
- **n_sample** (*int*) – Number of equally spaced samples to generate.
- **dt_sec** (*float*) – Time interval between samples in seconds.
- **gen** (*np.random.RandomState or None*) – Provide an existing RandomState for full control of reproducible random numbers, or None for non-reproducible random numbers.

desimodel.weather.**sample_transp**(*n_sample*, *dt_sec=180.0*, *gen=None*)

Generate a random time series of atmospheric transparency values.

See DESI-doc-3087 for details. Uses *get_transp_pdf()*, *_transp_psd()* and *sample_timeseries()*.

Parameters

- **n_sample** (*int*) – Number of equally spaced samples to generate.
- **dt_sec** (*float*) – Time interval between samples in seconds.
- **gen** (*np.random.RandomState or None*) – Provide an existing RandomState for full control of reproducible random numbers, or None for non-reproducible random numbers.

Returns 1D array of randomly generated samples.

Return type array

desimodel.weather.**whiten_transforms**(*data*, *data_min=None*, *data_max=None*)

Calculate a pair of transforms to whiten and unwhiten a distribution.

Uses *desimodel.weather.whiten_transforms_from_cdf()*.

Parameters

- **data** (*array*) – 1D array of samples from the distribution to whiten.
- **data_min** (*float or None*) – Clip the distribution to this minimum value, or at min(data) if None. Must be <= min(data).
- **data_max** (*float or None*) – Clip the distribution to this maximum value, or at max(data) if None. Must be >= max(data).

Returns See *desimodel.weather.whiten_transforms_from_cdf()*.

Return type tuple

desimodel.weather.**whiten_transforms_from_cdf**(*x*, *cdf*)

Calculate a pair of transforms to whiten and unwhiten a distribution.

The whitening transform is monotonic and invertible.

Parameters

- **x** (*array*) – 1D array of non-decreasing values giving bin edges for the distribution to whiten and unwhiten.
- **cdf** (*array*) – 1D array of non-decreasing values giving the cummulative probability density associated with each bin edge. Does not need to be normalized. Must have the same length as x.

Returns Tuple (F,G) of callable objects that whiten y=F(x) and unwhiten x=G(y) samples x of the input distribution, so that y has a Gaussian distribution with zero mean and unit variance.

Return type tuple

2.3 How to update inputs from DocDB

Context: Inputs from DocDB that require reformatting before using are kept in data/inputs/ and then reformatted using scripts in bin/ . Inputs from DocDB that can be used as-is are directly added to their final location under data/.

2.3.1 Basic Setup

Make branches of both desimodel GitHub code and svn data:

```
git checkout -b update_inputs
base=https://desi.lbl.gov/svn/code/desimodel
svn copy $base/trunk $base/branches/update_inputs
svn checkout $base/branches/update_inputs/data
export DESIMODEL=`pwd`
```

Add entries to $HOME/.netrc to enable downloading DocDB files without having to enter a password every time:

```
machine desi.lbl.gov
login StephenBailey
password NotMyRealPassword
```

The code in *desimodel.inputs.docdb* requires requests (for communicating with DocDB) and xlrd (for reading Microsoft Excel spreadsheets). Both of these are available via Anaconda.

2.3.2 Inputs to update

The update functions below belong to *desimodel.inputs* and take an optional argument `testdir` to specify an alternate directory where updated outputs should be written. When `testdir` is not specified, outputs are written to their standard locations under *desimodel.io.datadir()*.

DESI-0530-v13 Excel spreadsheet to .ecsv file for GFA locations

This writes the `gfa.ecsv` file containing the GFA data, which is pulled from the "GFALocation" tab on the DESI-0530-v13 Excel spreadsheet and from rows 16-23 and columns A-I. The function `desimodel.inputs.gfa.build_gfa_table()` writes the file in the current directory.

```
import desimodel.inputs.gfa
desimodel.inputs.gfa.build_gfa_table()
```

Positioner to Fiber Mapping

This updates the mapping of device locations on the focal plane to spectrograph fiber numbers using DESI-0530-v14, DESI-2721-v2 and DESI-329-v15.

```
import desimodel.inputs.fiberpos
desimodel.inputs.fiberpos.update()
```

To update a DESI-doc versions, edit the corresponding `docdb.download(...)` call.

Throughput

This updates the throughput model from DESI-0347 and DESI-0344 and also copies the top-level `desi.yaml` from DESI-0347:

```
import desimodel.inputs.throughput
desimodel.inputs.throughput.update()
```

To update the version of DESI-347 that is used, change the default value of `desi347_version` in the `update()` function. Similiarly for DESI-344.

Only three rows of the throughput spreadsheet from DESI-347 are used, with hard-coded row numbers. There are some simple checks that these are correct, using the `specthru_row` and `thru_row` arguments to `load_throughput()`, but check the outputs carefully if you think the spreadsheet structure might have changed.

Blur and Offsets

Use the notebook `doc/nb/DESI-0347_Throughput.ipynb` to update the following ouputs derivied from DESI-347:

- data/inputs/throughput/raytracing.txt
- data/throughput/DESI-0347_blur.ecsv
- data/throughput/DESI-0347_offset.ecsv
- data/throughput/DESI-0347_static_offset_[123].fits

Refer to the instructions in that notebook for details.

Testing

After changing any outputs that might break a unit test, update the small test dataset following *Testing desimodel* and edit `DESIMODEL_VERSION` in `.travis.yml` to point to the new version.

Commissioning Instrument corners

This updates the CI as-measured corner locations from DESI-4633v11 Corners.txt and reformats them into the GFA-corners format needed by desimodel, writing the results into `$$DESIMODEL/data/focalplane/ci-corners.ecsv`:

```
import desimodel.inputs.ci
desimodel.inputs.ci.update()
```

We don't anticipate needing to update this again, so this section is just documenting the provenance of that file.

Commit Changes to SVN

Once you have finished making updates on the `update_inputs` svn branch, checkout the trunk and merge your changes using:

```
svn checkout $base/trunk/data data.trunk
cd data.trunk
svn merge --dry-run $base/branches/update_inputs/data
# Make sure everything looks good. Then do it for real.
svn merge $base/branches/update_inputs/data
svn commit -m "Merge branch update_inputs into trunk"
```

Finally, remove the `update_inputs` branch:

```
svn remove $base/branches/update_inputs -m "Cleanup after updating inputs"
```

and tag the updated trunk (replace 0.13.0 as needed):

```
version=0.13.0
svn copy $base/trunk $base/tags/$version -m "Tagging desimodel $version"
```

2.3.3 To Do

Update methodology and document how to update the following:

- PSF model from DESI-0334
- PSF spots -> PSF for quicksim
- Fiber input loss calculations
- desimodel/data/focalplane/platescale.txt

2.4 Testing desimodel

2.4.1 Introduction

Tests using desimodel are a bit tricky since most of the code involves reading data files that are not included with the git product. In addition, the data files can be rather large, both individually and as a package. This document describes how to create lightweight test branches of the desimodel *data* that can be used for rapid testing.

2.4.2 When to Create a Test Branch

The word "branch(es)" below refers to svn branches not git branches, unless otherwise noted.

Test branches should track major changes to the code and data, but should *not* change for minor bug fixes in the code. For example, a branch named 'test-1.0' should work with all code that has a tag of the form 1.0.x. Similarly a 'test-1.1' branch should work with all code that has a tag of the form 1.1.x.

Test branches should be created immedately after the trunk has been tagged to produce a new minor version. For example, immedately after a svn tag '1.0.0' is created, the 'test-1.0' branch should be created before there are any additional changes to the trunkk.

2.4.3 How to Create a Test Branch

1. Create a branch in the standard way:

```
base=https://desi.lbl.gov/svn/code/desimodel
svn copy $base/trunk $base/branches/test-1.0 -m "Start new test branch"
```

2. Check out the branch, if you did not create it in your own checkout:

```
svn checkout $base/branches/test-1.0
```

3. Change to the branch directory:

```
cd test-1.0
```

4. Run the trim code to create a separate datalite directory:

```
python -c "from desimodel.trim import trim_data; trim_data('data', 'datalite')"
```

5. Add the datalite directory, remove data and commit:

```
svn add datalite
svn remove data
svn commit -m "Adding lite files"
```

6. Rename the existing datalite directory:

```
svn move datalite data
svn commit -m "Rename datalite/ back to data/"
```

7. Now tests can get this lightweight branch on-the-fly with:

```
svn export $base/branches/test-1.0
```

If you find any problems, just wipe out the branch and start again, e.g.:

```
svn rm $base/branches/test-1.0 -m "Starting over"
```

2.5 Focalplane Hardware Model

2.5.1 Overview

The DESI focalplane consists of petals with individual positioners and each positioner is a unique device connected to a fiber. The fibers travel to the spectrographs and are connected to a position on a slitblock. Each positioner has a range of angular motion along two axes (theta and phi).

The canonical focalplane calibration is stored in the ICS database, but we want to copy it into desimodel for offline analysis and to be able to tag exactly what version was used for fiber assignment and data processing runs.

The focalplane model currently uses the CS5 coordinate system for the X/Y locations of devices on the focalplane.

2.5.2 Tracking Changes

Although the generated focalplane models are checked into svn, we need to be able to get the state of the hardware at any time, across the full history of the DESI project. We enable this feature through the use of our file format (see section below on the details of the format). Since most changes are small (a positioner breaks or gets stuck, etc), we keep a running log of these small changes. A completely new model is only generated for large events (e.g. a petal is swapped out). When a focalplane is loaded, the most recent model for a given time is found and the events in this log are replayed up to the requested time. It is worth emphasizing the previous text again. When a focalplane is loaded for a particular timestamp:

1. Each focalplane model has a starting time, and remains valid until it is superceded by a newer model. The most recent set of 3 focalplane files which comes before the requested timestamp is read from disk. The static focalplane properties are kept as a Table and the exclusion polygons are read into a dictionary.
2. The state log (one of the 3 files) is parsed line by line. If the timestamp for that line comes before the requested time, then the event in that line is applied. All positioners have an initial state specified in the state log with a timestamp that matches the starting time of the focalplane. Subsequent events are appended to the state log with a timestamp.

The file formats used are text-based (ECSV and YAML). **However**, these are intended to be modified by the included scripts, which can ensure that the formatting is correct. The risk of typos and subtle errors if hand-editing these files is large. If you find that you frequently need to edit these files, then please open an issue to document your use case.

2.5.3 Inputs

Focalplane models are generated and updated from ICS database dumps at KPNO in `/data/focalplane/calibration/*.ecsv`.

Additionally, the following DocDB files are downloaded and parsed to get the mapping of fiber focalplane location to slitblock location (i.e. fiber number on the spectrographs).

DocDB Number	Purpose
4042	Petal verification for petal ID 02
4043	Petal verification for petal ID 03
4807	Petal verification for petal ID 04
4808	Petal verification for petal ID 05
4809	Petal verification for petal ID 06
4190	Petal verification for petal ID 07
4806	Petal verification for petal ID 08
4810	Petal verification for petal ID 09
4042	Petal verification for petal ID 10
4042	Petal verification for petal ID 11

2.5.4 Generating a New Model

Under normal circumstances focalplane updates are done daily by a KPNO cronjob running `etc/desimodel_sync_kpno_cron.sh` (update existing model) or `etc/desimodel_sync_kpno_force.sh` (use `--reset` to make a new model). The rest of this section documents what those are doing "under the hood", but it should not be necessary to run by hand.

A new focalplane model is generated with the `desi_sync_focalplane` script:

```
usage: desi_sync_focalplane [-h] --calib_file CALIB_FILE [--test] [--reset]
                            [--simulate_good] [--debug_dir DEBUG_DIR]
                            [--commit]

optional arguments:
  -h, --help            show this help message and exit
  --calib_file CALIB_FILE
                        The ECSV database dump file
  --test                Go through the process of updating the focalplane, but
                        do not actually write new files.
  --reset               Create a new focalplane model from the calib file,
                        ignoring all previous state information
  --simulate_good       Create a focalplane model for simulations. Non-broken
                        fibers set to good
  --debug_dir DEBUG_DIR
                        Override the output directory for debugging.
  --commit              Commit updated focalplane model to svn.
```

Note that the `--reset` option generates a new focalplane model, while without that option it updates the state ledger of the current focalplane model for only the positioners that changed. In addition to the `--calib_file` input with the latest ICS database focalplane dump, this script automatically downloads the necessary DocDB entries listed above, which requires you to have DESI DocDB credentials stored in your `$HOME/.netrc` file.

See desimodel tags 0.16.0 and prior for documentation of an older script `desi_generate_focalplane` which uses lab-measured focalplane metrology from DESI SVN `code/focalplane/fp_settings/pos_settings` to create a new focalplane model. This has been removed from newer versions of desimodel in favor of loading the metrology as measured in-situ at KPNO.

2.5.5 Updating the State of a Model

After a focalplane model is created, the state can be updated by rerunning `desi_sync_focalplane` with a new ICS database dump without using the `--reset` option. If needed, one can override the database to update the state of an individual positioner with the following command line tool:

```
usage: desi_update_focalplane_log [-h] [--location LOCATION] [--petal PETAL]
                                  [--device DEVICE] [--state STATE]
                                  [--exclusion EXCLUSION] [--time TIME]

optional arguments:
  -h, --help            show this help message and exit
  --location LOCATION   The device location (petal * 1000 + device loc)
                        modified by this event.
  --petal PETAL         The petal (focalplane location, not petal ID) modified
                        by this event (--device must also be used)
  --device DEVICE       The device location (--petal must also be given)
                        modified by this event.
  --state STATE         The new state to assign to the device.
  --exclusion EXCLUSION
                        The new exclusion polygon to assign to the device
                        (e.g. 'legacy', 'default', etc)
  --time TIME           Optional date/time (default is current date/time) when
                        this event happens. Format is YYYY-MM-DDTHH:mm:ss in
                        UTC time.
```

2.5.6 Updating the Exclusion Polygons in a Model

After a focalplane model is created, one can update the available exclusion polygons with the following command line tool:

```
usage: desi_update_focalplane_exclusion [-h]
                                        [--exclusion [EXCLUSION [EXCLUSION ...]]]
                                        [--time TIME]

optional arguments:
  -h, --help            show this help message and exit
  --exclusion [EXCLUSION [EXCLUSION ...]]
                        One or more text config files containing some
                        exclusion polygons to use. For example,
                        '$DESIMODEL/exclusions.conf'. This file should contain
                        a parameter 'NAME' with the name to use for this set
                        of exclusions.
  --time TIME           Optional date/time (default is current date/time) to
                        use when selecting the focalplane.Format is YYYY-MM-
                        DDTHH:mm:ss in UTC time.
```

Existing exclusion polygons with the same name as any input files will be replaced. New polygons will be appended.

2.5.7 File Format and Loading

A single focalplane model (with a starting valid datetime) consist of 3 files on disk. These files contain matching date / time stamps that correspond the the first valid time for that focalplane model. For example:

```
desi-focalplane_2019-09-16T00:00:00.ecsv
desi-exclusion_2019-09-16T00:00:00.yaml
desi-state_2019-09-16T00:00:00.ecsv
```

The first is an ECSV text file containing the static information such as the positioner locations and angle range, the mapping of device locations to fibers, etc.

Column	Data Type	Description
PETAL	int32	Petal location [0-9]
DEVICE	int32	Device location on the petal
LOCATION	int32	PETAL * 1000 + DEVICE
PETAL_ID	int32	The physical petal ID
DEVICE_ID	string	The physical device ID string
DEVICE_TYPE	string	The device type (POS, ETC, FIF)
SLITBLOCK	int32	The slit block where this fiber goes
BLOCKFIBER	int32	The fiber index within the slit block
CABLE	int32	The cable ID
CONDUIT	string	The conduit
FIBER	int32	PETAL * 500 + SLITBLOCK * 25 + BLOCKFIBER
FWHM	float64	FWHM at f/3.9
FRD	float64	FRD Throughput
ABS	float64	ABS Throughput
OFFSET_X	float64	X location of positioner center
OFFSET_Y	float64	Y location of positioner center
OFFSET_T	float64	THETA zero point angle
OFFSET_P	float64	PHI zero point angle
LENGTH_R1	float64	Length of THETA arm
LENGTH_R2	float64	Length of PHI arm
MAX_T	float64	Maximum THETA angle relative to OFFSET_T
MIN_T	float64	Minimum THETA angle relative to OFFSET_T
MAX_P	float64	Maximum PHI angle relative to OFFSET_P
MIN_P	float64	Minimum PHI angle relative to OFFSET_P

The second file is a YAML format file which contains one or more exclusion polygons for the positioners. Each named exclusion entry actually has multiple polygons: for the GFA, petal boundary, theta arm and phi arm. These define the shape of the polygon at the origin, which is then translated and rotated differently for every positioner based on the arm length, etc. Exclusion polygons are specified in terms of lists of circles and line segments.

The third file is another ECSV format file that contains the *state log* for the focalplane model. This is the running log of *events* that happen which modify the instantaneous state of the focalplane.

Column	Data Type	Description
TIME	string	The timestamp of the event (UTC, ISO format)
PETAL	int32	Petal location [0-9]
DEVICE	int32	Device location on the petal
LOCATION	int32	PETAL * 1000 + DEVICE
STATE	uint32	State bit field (good == 0)
EXCLUSION	string	The exclusion polygon for this device

The file formats above are documented for completeness, but you should not generally read these manually. Instead, one calls the *load_focalplane()* function:

`desimodel.io.`**`load_focalplane`**`(`*`time=None`*`)`

Load the focalplane state that is valid for the given time.

Parameters **`time`** (`datetime`) – The time to query with explicit timezone. Default to current time (`now()`) with timezone UTC.

Returns A tuple of (FP layout, exclusion polygons, state, time string). The FP layout is a Table. The exclusion polygons are a dictionary indexed by names that are referenced in the state. The

state is a Table. The time string is the resulting UTC ISO format time string for the creation date of the FP model.

Return type `tuple`

The state table returned by this function contains the instantaneous state of the focalplane at the input time (i.e. the log of events has already been replayed and the state at the requested time is returned)

2.5.8 TO-DO

There are several small features needed:

- When marking fibers as broken or stuck, their current X/Y or theta/phi location should be marked. See https://github.com/desihub/desimodel/issues/122
- We should build this focalplane model from the online instrument DB. See https://github.com/desihub/desimodel/issues/124

2.6 Fiber Positioners

2.6.1 Input Files

DESI-0530 defines the location of positioners on the focal plane.

DESI-2721 defines the mapping from cassettes of 50 positioners/fibers on the focal plane to bundles of fibers on the spectrograph slit heads

2.6.2 Output Files

Within a cassette, the fiber order is randomized. desimodel.inputs.fiberpos.update() randomizes the fibers within a cassette and outputs $DESIMODEL/data/focalplane/fiberpos.[fits, txt, png] with the mapping of positioner -> fiber number. fiberpos-all.[fits, ecsv] also includes non-spectrograph fiber positioner locations such as fiducials and sky monitors.

2.6.3 Coordinate Systems

As defined in DESI-0481v1 table 1 (last row "CS5 DESI Focal Plane"), when the telescope is parked at zenith the +x direction is East (+RA), +z is pointed toward the ground, and thus +y points south (-dec).

Note that y and z are sign swapped wrt to the other DESI coordinate systems: most coordinate systems think about +z pointing toward the sky, while the focal plane thinks about +z as pointing away from the focal plane and thus away from the sky.

2.7 Footprint

2.7.1 Files

desi-tiles.fits contains a tiling of the sky in 10 overlapping layers, with column *IN_DESI* indicating whether a particular tile is within the DESI footprint. See DESI-0717 for a description of how *desi-tiles.fits* was generated.

desi-healpix-weights.fits contains a nside=256 nested healpix map of weights for what fraction of those healpix is covered by the footprint. This was generated with:

```
from desimodel.footprint import pixweight
newmap = pixweight(256,outfile="desi-healpix-weights.fits")
```

2.8 PSF Spots

2.8.1 Overview

Spectrograph PSF files are in data/specpsf/desi-psf-*.fits . These are in the Specter "SpotGrid" format.

2.8.2 Converting Zemax spots to Specter spots

DocDB DESI-0334v1 contains spots from zemax + diffraction + CCD effects, sampled on a grid of wavelength and slit position. Here are the commands to download and convert these to the files in etc/data/specpsf/.

YOU NORMALLY SHOULD NOT NEED TO RUN THESE.

They are for reference for when we need to generate new PSF files from a new set of spots in DocDB:

```
cd somewhere
mkdir blue red nir

DOCDB_USER=StephenBailey
DOCDB_PASS=NotMyRealPassword

cd blue
SPOTFILE=DESI-0334-blue-images.zip
wget --user $DOCDB_USER --password $DOCDB_PASS -O $SPOTFILE \
    "https://desi.lbl.gov/DocDB/cgi-bin/private/RetrieveFile?docid=334;filename=
↪$SPOTFILE;version=1"
unzip $SPOTFILE
rm $SPOTFILE

cd ../red
SPOTFILE=DESI-0334-red-images.zip
wget --user $DOCDB_USER --password $DOCDB_PASS -O $SPOTFILE \
    "https://desi.lbl.gov/DocDB/cgi-bin/private/RetrieveFile?docid=334;filename=
↪$SPOTFILE;version=1"
unzip $SPOTFILE
rm $SPOTFILE

cd ../nir
SPOTFILE=DESI-0334-NIR-images-500.zip
wget --user $DOCDB_USER --password $DOCDB_PASS -O $SPOTFILE \
    "https://desi.lbl.gov/DocDB/cgi-bin/private/RetrieveFile?docid=334;filename=
↪$SPOTFILE;version=1"
unzip $SPOTFILE
rm $SPOTFILE

cd ..
python $DESIMODEL/bin/spots2psf.py blue/Blue*.fits --camera b \
    -o $DESIMODEL/data/specpsf/psf-b.fits
```

(continues on next page)

(continued from previous page)

```
python $DESIMODEL/bin/spots2psf.py red/Red*.fits --camera r \
    -o $DESIMODEL/data/specpsf/psf-r.fits
python $DESIMODEL/bin/spots2psf.py nir/NIR*.fits --camera z \
    -o $DESIMODEL/data/specpsf/psf-z.fits

rm -r blue nir red
```

2.9 Components to the throughput calculation

2.9.1 Introduction

The DESI throughput model comes from the systems engineering throughput budget spreadsheet DESI-0347, augmented with higher resolution throughput data for the spectrographs + CCDs from DESI-0334. These are combined with the KPNO extinction model ZenithExtinction-KPNO.fits and pre-calculated fiber input geometric loss in $DESIMODEL/data/throughput/fiberloss*.dat .

The various contributions to the throughput are stored in a binary table following the throughput format used by Specter:

https://github.com/sbailey/specter/blob/dev/doc/datamodel/throughput.md

It is basically a binary table with columns:

- wavelength : in Angstroms
- extinction : atmospheric extinction in magnitudes per airmass
- fiberinput : geometrical loss at fiber input
- throughput : all other throughput terms, e.g. mirrors, spectrograph, CCDs

Different types of sources are affected by different combinations of throughput terms.

Term	OBJECT	SKY	CALIB
EXTINCTION	yes	yes	no
FIBERINPUT	yes	no	no
THROUGHPUT	yes	yes	yes

Source types:

- OBJECT: astronomical objects, affected by all sources of throughput loss
- SKY: sky spectra do not have a geometrical loss term for the fiber input. Positioner misalignments and changes to the atmospheric PSF still get the same amount of sky light down the fiber.
- CALIB: calibration lamps internal to the dome do not see atmospheric extinction or fiber geometric loss terms.

The sections below detail the input data used to generate this table.

2.9.2 Atmospheric Extinction

Affects astronomical objects and sky spectra, but not calibration exposures.

Depends upon airmass; extinction curve from ZenithExtinction-KPNO.fits included in this product.

2.9.3 Fiber Input

Affects astronomical objects, but not sky or calibration spectra.

Includes both PSF/aperture losses and pointing/guiding mis-alignment.

From:

- DESI-0347v2
  - row 16 "PSF and Aperture efficiency"
  - row 27 "Lateral errors"
  - row 52 "Fiber Defocus overall budget"
  - Multiplied then interpolated with a cubic spline

2.9.4 All other telescope, fiber, and instrument throughputs

Affects all object types.

From DESI-0347v11 row 112 (total throughput) divided by the low resolution spectrograph throughtput row 93, then multiplied by the high resolution spectrograph+CCD throughputs in DESI-0334 *-thru*.txt files.

2.9.5 TO DO

The spectrograph throughput numbers in DESI-0334 have been superseded by as-built measurements from a variety of DocDB entries for each component of the spectrographs, as listed in DESI-0347 rows 97-110. These newer numbers are not yet included in desimodel.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

desimodel, 12
desimodel.focalplane.geometry, 12
desimodel.footprint, 15
desimodel.inputs, 19
desimodel.inputs.docdb, 19
desimodel.inputs.fiberpos, 20
desimodel.inputs.focalplane, 20
desimodel.inputs.throughput, 22
desimodel.install, 23
desimodel.io, 23
desimodel.trim, 26
desimodel.weather, 27

Index

Symbols

_auth() (*in module desimodel.inputs.docdb*), 19
_check_radec() (*desimodel.focalplane.geometry.FocalPlane method*), 12
_embed_sphere() (*in module desimodel.footprint*), 15
_extrapolate_r_s() (*in module desimodel.focalplane.geometry*), 13
_seeing_fit_model() (*in module desimodel.weather*), 27
_seeing_psd() (*in module desimodel.weather*), 27
_transp_psd() (*in module desimodel.weather*), 27
_xls_col2int() (*in module desimodel.inputs.docdb*), 19

A

assert_svn_exists() (*in module desimodel.install*), 23

C

create() (*in module desimodel.inputs.focalplane*), 21

D

datadir() (*in module desimodel.io*), 23
default_install_dir() (*in module desimodel.install*), 23
DESIMODEL, 6, 24
desimodel (*module*), 12
desimodel.focalplane.geometry (*module*), 12
desimodel.footprint (*module*), 15
desimodel.inputs (*module*), 19
desimodel.inputs.docdb (*module*), 19
desimodel.inputs.fiberpos (*module*), 20
desimodel.inputs.focalplane (*module*), 20
desimodel.inputs.throughput (*module*), 22
desimodel.install (*module*), 23
desimodel.io (*module*), 23
desimodel.trim (*module*), 26
desimodel.weather (*module*), 27
devices_from_fiberpos() (*in module desimodel.inputs.focalplane*), 21
devices_from_files() (*in module desimodel.inputs.focalplane*), 21
dome_closed_fractions() (*in module desimodel.weather*), 27
download() (*in module desimodel.inputs.docdb*), 19

E

environment variable
    DESIMODEL, 6, 24
    HOME, 26
    INSTALL_DIR, 7

F

fiber_area_arcsec2() (*in module desimodel.focalplane.geometry*), 13
find_points_in_tiles() (*in module desimodel.footprint*), 15
find_points_radec() (*in module desimodel.footprint*), 15
find_tiles_over_point() (*in module desimodel.footprint*), 15
findfile() (*in module desimodel.io*), 24
FocalPlane (*class in desimodel.focalplane.geometry*), 12

G

get_radius_deg() (*in module desimodel.focalplane.geometry*), 13
get_radius_mm() (*in module desimodel.focalplane.geometry*), 14
get_seeing_pdf() (*in module desimodel.weather*), 28
get_tile_radec() (*in module desimodel.footprint*), 16
get_tile_radius_deg() (*in module desimodel.focalplane.geometry*), 14

get_tile_radius_mm() (*in module desimodel.focalplane.geometry*), 14
get_transp_pdf() (*in module desimodel.weather*), 28
get_waveminmax() (*in module desimodel.inputs.throughput*), 22

H

HOME, 26

I

inout() (*in module desimodel.trim*), 26
install() (*in module desimodel.install*), 23
INSTALL_DIR, 7
is_point_in_desi() (*in module desimodel.footprint*), 16

L

load_desiparams() (*in module desimodel.io*), 24
load_deviceloc() (*in module desimodel.io*), 24
load_fiberinput() (*in module desimodel.inputs.throughput*), 22
load_fiberpos() (*in module desimodel.io*), 24
load_focalplane() (*in module desimodel.io*), 24
load_gfa() (*in module desimodel.io*), 24
load_pixweight() (*in module desimodel.io*), 24
load_platescale() (*in module desimodel.io*), 25
load_psf() (*in module desimodel.io*), 25
load_spec_throughputs() (*in module desimodel.inputs.throughput*), 22
load_target_info() (*in module desimodel.io*), 25
load_throughput() (*in module desimodel.inputs.throughput*), 22
load_throughput() (*in module desimodel.io*), 25
load_tiles() (*in module desimodel.io*), 25

M

main() (*in module desimodel.install*), 23

P

pass2program() (*in module desimodel.footprint*), 16
pix2tiles() (*in module desimodel.footprint*), 16
pixweight() (*in module desimodel.footprint*), 16
program2pass() (*in module desimodel.footprint*), 17

Q

qs2xy() (*in module desimodel.focalplane.geometry*), 14

R

radec2pix() (*in module desimodel.footprint*), 17
radec2pos() (*desimodel.focalplane.geometry.FocalPlane method*), 12
radec2xy() (*desimodel.focalplane.geometry.FocalPlane method*), 13
radec2xy() (*in module desimodel.focalplane.geometry*), 14
rebin_image() (*in module desimodel.trim*), 26
reset_cache() (*in module desimodel.io*), 26

S

sample_seeing() (*in module desimodel.weather*), 28
sample_timeseries() (*in module desimodel.weather*), 29
sample_transp() (*in module desimodel.weather*), 29
set_tele_pointing() (*desimodel.focalplane.geometry.FocalPlane method*), 13
svn_export() (*in module desimodel.install*), 23

T

tileids2pix() (*in module desimodel.footprint*), 18
tiles2fracpix() (*in module desimodel.footprint*), 18
tiles2pix() (*in module desimodel.footprint*), 18
trim_data() (*in module desimodel.trim*), 26
trim_focalplane() (*in module desimodel.trim*), 26
trim_footprint() (*in module desimodel.trim*), 26
trim_inputs() (*in module desimodel.trim*), 26
trim_sky() (*in module desimodel.trim*), 27
trim_specpsf() (*in module desimodel.trim*), 27
trim_spectra() (*in module desimodel.trim*), 27
trim_targets() (*in module desimodel.trim*), 27
trim_throughput() (*in module desimodel.trim*), 27
trim_weather() (*in module desimodel.trim*), 27

U

update() (*in module desimodel.inputs.fiberpos*), 20
update() (*in module desimodel.inputs.throughput*), 22

W

whiten_transforms() (*in module desimodel.weather*), 29
whiten_transforms_from_cdf() (*in module desimodel.weather*), 30
write_text_fiberpos() (*in module desimodel.inputs.fiberpos*), 20

X

xls_read_col() (*in module desimodel.inputs.docdb*), 19
xls_read_row() (*in module desimodel.inputs.docdb*), 20
xy2pos() (*desimodel.focalplane.geometry.FocalPlane method*), 13
xy2qs() (*in module desimodel.focalplane.geometry*), 14

xy2radec() (*desimodel.focalplane.geometry.FocalPlane method*), 13
xy2radec() (*in module desimodel.focalplane.geometry*), 15