desisurvey Documentation

Release 0.19.0

DESI

May 13, 2022

Contents

CHAPTER 1

Contents

1.1 desisurvey change log

1.1.1 0.19.1 (unreleased)

- No changes yet.

1.1.2 0.19.0 (2022-05-13)

- Change dec prioritization scheme in rules.yaml to make the dlog(priority)/ddec not depend on the range of dec in the group.
- Adopt normal import numpy as np convention in tileqa.
- Return to 1.62 deg instead of 1.63 deg for tile overlap definition.
- Add some forecast plot trend lines.
- Add retire_tile script to set IN_DESI = False for particular tiles, adding duplicate tiles with new TILEID at those locations. Used when we want to abandon a particular design but still want observations at that location in the future.
- Trust EFFTIME_SPEC from offline pipeline rather than using ELG_EFFTIME_DARK or BGS_EFFTIME_BRIGHT explicitly. This is to make the transition to LRG_EFFTIME_DARK seamless.
- Simplify afternoon planning. Automatically make backup tiles as part of manual plan. Make night plots. (PR #144)
- Minor updates to unit test configuration (PR #145).

1.1.3 0.18.0 (2021-07-06)

- Increase number of starting points in HA optimization.

- Add priority to tiles with observed neighbors.
- Mark a tile as having had observations started even if EFFTIME = 0, as long as the tile has been used in a SCIENCE observation.
- Don't count IN_DESI = False tiles against completeness. Don't plot IN_DESI = False tiles in movies.
- Enable identification of high priority 10k footprint tiles.
- Add slew time to simulations and favor short slews in tile selection.
- Implement "holding pen" where new fiberassign files are found (PR #139).
- Miscellaneous holding pen and tile selection improvements (PR #140, #141).
- Let ICS move tiles into place. Make tile selection only use speed (PR #142).
- Add fiberassign-on-the-fly capability (PR #143)

1.1.4 0.17.0 (2021-04-23)

- Allow program selection according to conditions, rather than ephemerides. Make NTS more robust. Set max dwell times based on not hitting airmass or observing dark tiles in twilight. Allow requiring low pass tiles before overlapping high pass tiles, and preferring high pass tiles otherwise. (PR *#135*)
- Fix ERFA and testing support for astropy 4.2 (PRs #136, #138).
- Switch to github actions instead of travis for testing (PR #138).

1.1.5 0.16.0 (2021-03-31)

Multiple updates for survey operations

- Add sbprof argument to etc.seeing_exposure_factor, accounting for different sensitivities of BRIGHT and DARK programs to seeing.
- Reorganize state to match DailyOps desiderata. Implement multiple survey choices in NTS. Reorganize config file. Reduce verbosity. Make HA observation choices more stringent at high airmass. (PR *#131*)
- Optionally merge dark and gray layers. Implement nopass strategy. Move scheduler state to planner and start to change data model. (PR *#130*)
- Use sky levels in ETC. Use EFFTIME rather than R_DEPTH, and harmonize MW extinction correction with EFFTIME. (PR #129)
- Fix bug in transparency and ETC snr2 accumulation rate calculation (PR #128)

1.1.6 0.15.0 (2021-02-15)

- NTS updates for SV tiles (PRs #123 and *#124*).

1.1.7 0.14.1 (2020-12-11)

- Update desisurvey.plots.plot_sky_passes for compatibility with desiutil >= 3.0.0 (PR #122).

1.1.8 0.14.0 (2020-08-03)

- Fix py3.8 invalid escape sequences (PR #120).
- Update rules.yaml to use new tile file pass ordering (PR #114); requires desimodel>=0.11.0
- Move IERS functions to desiutil (PR #113).

1.1.9 0.13.0 (2020-04-07)

Requires desimodel/0.12.0 or later with new tile file.

- Change fiber_assignment_order to reflect new pass ordering (PR #111).
- Enable using NTS/desisurvey with CMX tile file; add HA limit (PR #107).

1.1.10 0.12.1 (2019-12-20)

- Workaround for missing IERS server (PR #105).

1.1.11 0.12.0 (2019-08-09)

- Minor updates to conform to data model standards (PR #94).
- Improved documentation (PR #94).
- Increase tile radius for coverage check (PR #97).
- Fix RA,DEC vs. DEC,RA bug (PR #99).
- Adds *desisurvey.scheduler.NTS* (Next Tile Selector) interface to ICS (PR #99)

1.1.12 0.11.0 (2018-11-26)

This version is a major refactoring of the code to simplify the logic for easier maintenance and documentation. There is now a clean separation between survey strategy, afternoon planning, next-tile selection, and exposure-time calculations. The refactored code is also significantly faster (PR #91).

- Add new modules: tiles, forecast, scheduler.
- Move modules schedule, progress, surveyplan to old/.
- Add new class ExposureTimeCalculator to etc module.
- Add new class Planner to plan module.
- Decouple ephemerides date range from nominal survey start/stop.
- Rename ephemerides to ephem (to enforce new get_ephem access pattern).
- Use of twilight is now optional and off by default.
- Exposure times include an average correction for the moon: this will be fixed in a future release.

1.1.13 0.10.4 (2018-10-02)

Updates for survey margin estimates (PR #89):

- Implement realistic 18-day monsoon shutdowns instead of fixed 45-day period.
- Replay daily Mayall weather history instead of fixed monthly fractions (needs desimodel >= 0.9.8)
- Update exposure-time model for atmospheric seeing.
- Speed up full-moon, program change and LST calculations in ephemerides module.
- Requires desimodel >= 0.9.8

1.1.14 0.10.3 (2018-09-26)

- Added tiling dithering and QA code (PR #87).
- Allow `PASS` to be as large as 99 (PR #88).

1.1.15 0.10.2 (2018-06-27)

- Do not assume that input tile file includes all of DARK, BRIGHT, and GRAY tiles (PR #83).
- Enforce at least six characters in program name in exposures table (PR #86).

1.1.16 0.10.1 (2017-12-20)

- Set the `EXTNAME` keyword on the Table returned by `Progress.get_exposures()`.

1.1.17 0.10.0 (2017-11-09)

- Progress.get_exposures() updates:
  - includes FLAVOR and PROGRAM columns.
  - uses desimodel.footprint.pass2program if available.
  - standardized on UPPERCASE column names and NIGHT=YEARMMDD not YEAR-MM-DD.

1.1.18 0.9.3 (2017-10-09)

- Fixes #18, #49, #54.
- Improvements to surveymovie script.
- Add progress columns to track fiber assignment and planning.
- Add support for optional depth-first survey strategy.
- Docs now auto-generated at http://desisurvey.readthedocs.io/en/latest/

1.1.19 0.9.2 (2017-09-29)

- Implement fiber assignment policy via –fa-delay option to surveyplan.

1.1.20 0.9.1 (2017-09-20)

- Command line scripts –config-file option to override default config file.
- Fixes for bugs that occur when testing with a small subset of tiles.
- Changes $DESISURVEY -> $DESISURVEY_OUTPUT as output dir envvar name
- Remove astropy units from function signatures (for readthedocs).
- Add travis, coveralls and readthedocs automation.

1.1.21 0.9.0 (2017-09-11)

- Create surveyinit script to calculate initial HA assignments.
- Improve Optimizer algorithms (~10x faster, better initialization).
- Create surveymovie to visualize survey scheduling and progress.
- Rework surveyplan to track fiber assignment availability.
- Validate a set of observing rules consistent with the baseline strategy described in DESI-doc-1767-v3.

1.1.22 0.8.2 (2017-07-12)

- Fix flat vs. flatten for older versions of numpy (PR #52).

1.1.23 0.8.1 (2017-06-19)

- Fix unit tests broken in 0.8.0 (PR #46).

1.1.24 0.8.0 (2017-06-18)

- Implement LST-driven scheduling strategy.
- Create new optimize module for iterative HA optimization.
- Rename module plan -> schedule.
- Create new plan module to manage fiber-assignment groups and priorities.

1.1.25 0.7.0 (2017-06-05)

- Freeze IERS table used by astropy time, coordinates.
- Implement alternate greedy scheduler with optional policy weights.
- Add *plots.plot_scheduler()*
- Partial fix of RA=0/360 planning bug

1.1.26 0.6.0 (2017-05-10)

- Add new config yaml file and python wrapper.
- Convert all code to use new config machinery.
- Add new class Plan for future use in scheduling.
- Unify different output files with overlapping contents into single output managed by desisurvey.progress.
- Cleanup and reorganize the Ephemerides class.
- Add comparisons with independent JPL Horizons run to unit tests for AltAz transforms and ephemerides calculations.
- Add new plot utilities for Progress and Plan objects.
- Document and handle astropy IERS warnings about future times.
- Rename exposurecalc module to etc (exposure-time calculator).
- Update docstrings and imports, and remove unused code.

1.1.27 0.5.0 (2017-04-13)

- Add new plot methods
- Bug fix to Az computation and airmass calculator
- Code reorganization

1.1.28 0.4.0 (2017-04-04)

This version was tagged for the 2% sprint data challenge.

- Add unit tests; fix afternoon planning tile updates and other minor bugs
- Fix off-by-one with YEARMMDD vs. MJD of sunset
- Add new plots module
- Refactor nightcal module into ephmerides

1.1.29 0.3.1 (2016-12-21)

- fixed E(B-V) scaling for exposure time (PR #12)

1.1.30 0.3.0 (2016-11-29)

First release after refactoring.

1.1.31 0.2.0 (2016-11-19)

Last version before repackaging of surveysim.

1.2 Survey Planning Tile Prioritization Rules

Rules for how tiles are prioritized are contained in a YAML file, with a baseline survey example in *py/desisurvey/data/rules.yaml*. Each tile group is separately scheduled and prioritized. All tiles in the footprint must be assigned to exactly one group.

Group names are arbitrary strings. The tiles associated with a group can be optionally restricted using:

- cap = N or S.
- dec_min or dec_max limits.
- covers keyword specifies previously defined subgroups that the tiles in this group cover (for fiber assignment).

Groups are also associated with a list of passes, with each pass defining a subgroup.

Each subgroup has associated rules that specify its relative weight when selecting the next tile in the scheduler.

Rules are assigned separately for each subgroup using the notation:

```
GROUP_NAME(PASS): { ...rules... }
```

By default, each subgroup has an initial weight of zero when the survey starts (so will not be scheduled), unless it has an explicit start rule:

```
START: INITIAL_WEIGHT
```

Additional rules of the form:

```
GROUP_NAME(PASS): NEW_WEIGHT
```

specify the subgroup weight that will be assigned when the specified subgroup has been completely observed. Forward references to subgroups that have not been defined yet are ok.

1.3 Full desisurvey API Reference

1.3.1 Modules

desisurvey.config

Manage survey planning and schedule configuration data.

The normal usage is:

```
>>> config = Configuration()
>>> config.programs.BRIGHT.max_sun_altitude()
<Quantity -13.0 deg>
```

Use dot notation to specify nodes in the configuration hiearchy and function call notation to access terminal node values.

Terminal node values are first converted according to YAML rules. Strings containing a number followed by valid astropy units are subsequently converted to astropy quantities. Strings of the form YYYY-MM-DD are converted to datetime.date objects.

To change a value after the configuration has been loaded into memory use, for example:

```
>>> config.full_moon_nights.set_value(5)
```

Assigned values must have the appropriate converted types, for example:

```
>>> import datetime
>>> config.last_day.set_value(datetime.date(2024, 1, 1))
>>> import astropy.units as u
>>> config.location.temperature.set_value(-5 * u.deg_C)
```

The configuration is implemented as a singleton so the YAML file is only loaded and parsed the first time a Configuration() is built. Subsequent calls to Configuration() always return the same object.

class desisurvey.config.**Configuration**(*file_name='config.yaml'*)

Top-level configuration data node.

_initialize(*file_name=None*)

Initialize a configuration data structure from a YAML file.

get_path(*name*)

Prepend this configuration's output_path to non-absolute paths.

Configured by the `output_path` node and *set_output_path()*.

An absolute path is returned immediately so an environment variable used in output_path only needs to be defined if relative paths are used.

Parameters **name** (*str*) – Absolute or relative path name, which does not need to exist yet.

Returns Path name to use. Relative path names will have our output_path prepended. Absolute path names will be unchanged.

Return type str

static reset()

Forget our singleton instance. Mainly intended for unit tests.

set_output_path(*output_path*)

Set the output directory for relative paths.

The path must exist when this method is called. Called by *get_path()* for a non-absolute path. This method updates the configuration output_path value.

Parameters **output_path** (*str*) – A path possibly including environment variables enclosed in {...} that will be substituted from the current environment.

Raises `ValueError` – Path uses undefined environment variable or does not exist.

class desisurvey.config.**Node**(*value*, *path=[]*)

A single node of a configuration data structure.

The purpose of this class is to allow nested dictionaries to be accessed using attribute dot notation, and to implement automatic conversion of strings of the form "<value> <units>" into corresponding astropy quantities.

keys

Return the list of keys for a non-leaf node or raise a RuntimeError for a terminal node.

path

Return the full path to this node using dot notation.

set_value(*new_value*)

Set a terminal node's value or raise a RuntimeError for a non-terminal node.

desisurvey.ephem

Tabulate sun and moon ephemerides during the survey.

class desisurvey.ephem.**Ephemerides**(*start_date*, *stop_date*, *num_obj_steps=25*, *restore=None*)
Tabulate ephemerides.

get_ephem() should normally be used rather than calling this constructor directly.

Parameters

- **start_date** (*datetime.date*) – Calculated ephemerides start on the evening of this date.
- **stop_date** (*datetime.date*) – Calculated ephemerides stop on the morning of this date.
- **num_obj_steps** (*int*) – Number of steps for tabulating object (ra, dec) during each 24-hour period from local noon to local noon. Ignored when restore is set.
- **restore** (*str or None*) – Name of a file to restore ephemerides from. Construct ephemerides from scratch when None. A restored file must have start and stop dates that match our args.

start
Local noon before the first night for which ephemerides are calculated.

Type astropy.time.Time

stop
Local noon after the last night for which ephemerides are calculated.

Type astropy.time.Time

num_nights
Number of consecutive nights for which ephemerides are calculated.

Type int

get_available_lst(*start_date=None*, *stop_date=None*, *nbins=192*, *origin=-60*, *weather=None*, *include_monsoon=False*, *include_full_moon=False*, *include_twilight=False*)
Calculate histograms of available LST for each program.

Parameters

- **start_date** (*date or None*) – First night to include or use the first date of the survey. Must be convertible to a date using *desisurvey.utils.get_date()*.
- **stop_date** (*date or None*) – First night to include or use the last date of the survey. Must be convertible to a date using *desisurvey.utils.get_date()*.
- **nbins** (*int*) – Number of LST bins to use.
- **origin** (*float*) – Rotate DEC values in plots so that the left edge is at this value in degrees.
- **weather** (*array or None*) – 1D array of nightly weather factors (0-1) to use, or None to calculate available LST assuming perfect weather. Length must equal the number of nights between start and stop. Values are fraction of the night with the dome open (0=never, 1=always). Use 1 - desimodel.weather.dome_closed_fractions() to lookup suitable corrections based on historical weather data.
- **include_monsoon** (*bool*) – Include nights during the annual monsoon shutdowns.

- **include_fullmoon** (*bool*) – Include nights during the monthly full-moon breaks.
- **include_twilight** (*bool*) – Include twilight in the BRIGHT program when True.

Returns Tuple (lst_hist, lst_bins) with lst_hist having shape (3,nbins) and lst_bins having shape (nbins+1,).

Return type tuple

get_moon_illuminated_fraction(*mjd*)

Return the illuminated fraction of the moon.

Uses linear interpolation on the tabulated fractions at midnight and should be accurate to about 0.01. For reference, the fraction changes by up to 0.004 per hour.

Parameters **mjd** (*float or array*) – MJD values during a single night where the program should be tabulated.

Returns Illuminated fraction at each input time.

Return type float or array

get_night(*night*, *as_index=False*)

Return the row of ephemerides for a single night.

Parameters

- **night** (*date*) – Converted to a date using *desisurvey.utils.get_date()*.
- **as_index** (*bool*) – Return the row index of the specified night in our per-night table if True. Otherwise return the row itself.

Returns Row of ephemeris data for the requested night or the index of this row (selected via `as_index`).

Return type astropy.table.Row or int

get_night_program(*night*, *include_twilight=False*, *program_as_int=False*)

Return the program sequence for one night.

The program definitions are taken from *desisurvey.config.Configuration* and depend only on sun and moon ephemerides for the night.

Parameters

- **night** (*date*) – Converted to a date using *desisurvey.utils.get_date()*.
- **include_twilight** (*bool*) – Include twilight time at the start and end of each night in the BRIGHT program.
- **program_as_int** (*bool*) – Return program encoded as a small integer instead of a string when True.

Returns Tuple (programs, changes) where programs is a list of N program names and changes is a 1D numpy array of N+1 MJD values that bracket each program during the night.

Return type tuple

get_program_hours(*start_date=None*, *stop_date=None*, *include_monsoon=False*, *include_full_moon=False*, *include_twilight=True*)

Tabulate hours in each program during each night of the survey.

Use *desisurvey.plots.plot_program()* to visualize program hours.

This method calculates scheduled hours with no correction for weather. Use 1 - `desimodel.weather.dome_closed_fractions()` to lookup nightly corrections based on historical weather data.

Parameters

- **ephem** (*desisurvey.ephem.Ephemerides*) – Tabulated ephemerides data to use for determining the program.
- **start_date** (*date or None*) – First night to include or use the first date of the survey. Must be convertible to a date using *desisurvey.utils.get_date()*.
- **stop_date** (*date or None*) – First night to include or use the last date of the survey. Must be convertible to a date using *desisurvey.utils.get_date()*.
- **include_monsoon** (*bool*) – Include nights during the annual monsoon shutdowns.
- **include_fullmoon** (*bool*) – Include nights during the monthly full-moon breaks.
- **include_twilight** (*bool*) – Include twilight time at the start and end of each night in the BRIGHT program.

Returns Numpy array of shape (3, num_nights) containing the number of hours in each program (0=DARK, 1=GRAY, 2=BRIGHT) during each night.

Return type array

get_row(*row_index*)
Return the specified row of our table.

Parameters

- **row_index** (*int*) – Index starting from zero of the requested row. Negative values are allowed and specify offsets from the end of the table in the usual way.
- **Returns** –
- **or int** (*astropy.table.Row*) – Row of ephemeris data for the requested night.

is_full_moon(*night*, *num_nights=None*)
Test if a night occurs during a full-moon break.

The full moon break is defined as the `num_nights` nights where the moon is most fully illuminated at local midnight. This method should normally be called with `num_nights` equal to None, in which case the value is taken from our `desisurvey.config.Configuration`.

Parameters

- **night** (*date*) – Converted to a date using *desisurvey.utils.get_date()*.
- **num_nights** (*int or None*) – Number of nights to block out around each full-moon.

Returns True if the specified night falls during a full-moon break.

Return type bool

table
Read-only access to our internal table.

tabulate_program(*mjd*, *include_twilight=False*, *as_tuple=True*)
Tabulate the program during one night.

The program definitions are taken from *desisurvey.config.Configuration* and depend only on sun and moon ephemerides for the night.

Parameters

- **mjd** (*float or array*) – MJD values during a single night where the program should be tabulated.

- **include_twilight** (*bool*) – Include twilight time at the start and end of each night in the BRIGHT program.
- **as_tuple** (*bool*) – Return a tuple (dark, gray, bright) or else a vector of int16 values.

Returns Tuple (dark, gray, bright) of boolean arrays that tabulates the program at each input MJD or an array of small integer indices into `desisurvey.tiles.Tiles.CONDITIONS`, with the special value -1 indicating DAYTIME. All output arrays have the same shape as the input `mjd` array.

Return type tuple or array

`desisurvey.ephem.`**get_ephem**(*use_cache=True*, *write_cache=True*)

Return tabulated ephemerides for (START_DATE,STOP_DATE).

The pyephem module must be installed to calculate ephemerides, but is not necessary when a FITS file of precalcuated data is available.

Parameters

- **use_cache** (*bool*) – Use cached ephemerides from memory or disk if possible when True. Otherwise, always calculate from scratch.
- **write_cache** (*bool*) – When True, write a generated table so it is available for future invocations. Writing only takes place when a cached object is not available or `use_cache` is False.

Returns Object with tabulated ephemerides for (START_DATE,STOP_DATE).

Return type *Ephemerides*

`desisurvey.ephem.`**get_grid**(*step_size=1*, *night_start=-6*, *night_stop=7*)

Calculate a grid of equally spaced times covering one night.

In case the requested step size does not evenly divide the requested range, the last grid point will be rounded up.

The default range covers all possible observing times at KPNO.

Parameters

- **step_size** (`astropy.units.Quantity`, optional) – Size of each grid step with time units, default 1 min.
- **night_start** (`astropy.units.Quantity`, optional) – First grid point relative to local midnight with time units, default -6 h.
- **night_stop** (`astropy.units.Quantity`, optional) – Last grid point relative to local midnight with time units, default 7 h.

Returns Numpy array of dimensionless offsets relative to local midnight in units of days.

Return type array

`desisurvey.ephem.`**get_object_interpolator**(*row*, *object_name*, *altaz=False*)

Build an interpolator for object location during one night.

Wrap around in RA is handled correctly and we assume that the object never wraps around in DEC. The interpolated unit vectors should be within 0.3 degrees of the true unit vectors in both (dec,ra) and (alt,az).

Parameters

- **row** (*astropy.table.Row*) – A single row from the ephemerides astropy Table corresponding to the night in question.
- **object_name** (*string*) – Name of the object to build an interpolator for. Must be listed under avoid_objects in *our configuration*.

- **altaz** (*bool*) – Interpolate in (alt,az) if True, else interpolate in (dec,ra).

Returns A callable object that takes a single MJD value or an array of MJD values and returns the corresponding (dec,ra) or (alt,az) values in degrees, with -90 <= dec,alt <= +90 and 0 <= ra,az < 360.

Return type callable

desisurvey.etc

Calculate the nominal exposure time for specified observing conditions.

Use *exposure_time()* to combine all effects into an exposure time in seconds, or call functions to calculate the individual exposure-time factors associated with each effect.

The following effects are included: seeing, transparency, galactic dust extinction, airmass, scattered moonlight. The following effects are not yet implemented: twilight sky brightness, clouds, variable OH sky brightness.

class desisurvey.etc.**ExposureTimeCalculator**(*save_history=False*)

Online Exposure Time Calculator.

Track observing conditions (seeing, transparency, sky background) during an exposure using the *start()*, *update()* and *stop()* methods.

Exposure time tracking is configured by the following parameters:

- nominal_exposure_time
- new_field_setup
- same_field_setup
- cosmic_ray_split
- min_exposures

Note that this version applies an average correction for the moon during the GRAY and BRIGHT programs, rather than idividual corrections based on the moon parameters. This will be fixed in a future version.

Parameters **save_history** (*bool*) – When True, records the history of internal calculations during an exposure, for debugging and plotting.

active

Are we tracking an exposure?

Set True by *start()* and False by *stop()*.

could_complete(*t_remaining*, *program*, *snr2frac*, *exposure_factor*)

Determine which tiles could be completed.

Completion refers to achieving SNR2 = 1, which might require multiple exposures.

Used by *desisurvey.scheduler.Scheduler.next_tile()* and uses *estimate_exposure()*.

Parameters

- **t_remaining** (*float*) – Time remaining in units of days.
- **program** (*str*) – Program that the candidate tiles belong to (must be the same for all tiles).
- **snr2frac** (*float or array*) – Fractional SNR2 integrated so far for each tile to consider.

- **exposure_factor** (*float or array*) – Exposure-time factor for each tile to consider.

Returns 1D array of booleans indicating which tiles (if any) could completed within the remaining time.

Return type array

estimate_exposure(*program*, *snr2frac*, *exposure_factor*, *nexp_completed=0*)
Estimate exposure time(s).

Can be used to estimate exposures for one or many tiles from the same program.

Parameters

- **program** (*str*) – Name of the program to estimate exposure times for. Used to determine the nominal exposure time. All tiles must be from the same program.
- **snr2frac** (*float or array*) – Fractional SNR2 integrated so far for the tile(s) to estimate.
- **exposure_factor** (*float or array*) – Exposure-time factor for the tile(s) to estimate.
- **nexp_completed** (*int or array*) – Number of exposures completed so far for tile(s) to estimate.

Returns Tuple (texp_total, texp_remaining, nexp) of floats or arrays, where texp_total is the total time that would be required under current conditions, texp_remaining is the remaining time under current conditions taking the already accumulated SNR2 into account, and nexp is the estimated number of remaining exposures required.

Return type tuple

exptime
Exposure time in days recorded by last call to *stop()*.

snr2frac
Integrated fractional SNR2 of tile currently being exposed.

Includes signal accumulated in previous exposures. Initialized by *start()*, updated by *update()* and frozen by *stop()*.

start(*mjd_now*, *tileid*, *program*, *snr2frac*, *exposure_factor*, *seeing*, *transp*, *sky*)
Start tracking an exposure.

Must be called before using *update()* to track changing conditions during the exposure.

Parameters

- **mjd_now** (*float*) – MJD timestamp when exposure starts.
- **tileid** (*int*) – ID of the tile being exposed. This is only used to recognize consecutive exposures of the same tile.
- **program** (*str*) – Name of the program the exposed tile belongs to.
- **snr2frac** (*float*) – Previous accumulated fractional SNR2 of the exposed tile.
- **exposure_factor** (*float*) – Exposure factor of the tile when the exposure starts, based on the current conditions specified by the remaining parameters.
- **seeing** (*float*) – Initial atmospheric seeing in arcseconds.
- **transp** (*float*) – Initial atmospheric transparency (0,1).

- **sky** (*float*) – Initial sky background level.

stop(*mjd_now*)
Stop tracking an exposure.

After calling this method, use *exptime* to look up the exposure time.

Parameters **mjd_now** (*float*) – MJD timestamp when the current exposure was stopped.

Returns True if this tile is "done" or False if another exposure of the same tile should be started immediately. Note that "done" normally means the tile has reached its target SNR2, but could also mean that the SNR2 accumulation rate has fallen below some threshold so that it is no longer useful to continue exposing.

Return type bool

update(*mjd_now*, *seeing*, *transp*, *sky*)
Track changing conditions during an exposure.

Must call *start()* first to start tracking an exposure.

Parameters

- **mjd_now** (*float*) – Current MJD timestamp.
- **seeing** (*float*) – Estimate of average atmospheric seeing in arcseconds since last update (or start).
- **transp** (*float*) – Estimate of average atmospheric transparency since last update (or start).
- **sky** (*float*) – Estimate of average sky background level since last update (or start).

Returns True if the exposure should continue integrating.

Return type bool

weather_factor(*seeing*, *transp*, *sky_level*, *sbprof='ELG'*)
Return the relative SNR2 accumulation rate for specified conditions.

This is the inverse of the instantaneous exposure factor due to seeing and transparency.

Parameters

- **seeing** (*float*) – Atmospheric seeing in arcseconds.
- **transp** (*float*) – Atmospheric transparency in the range (0,1).
- **sky_level** (*float*) – sky_level relative to nominal

desisurvey.etc.**airmass_exposure_factor**(*airmass*)
Scaling of exposure time with airmass relative to nominal.

The exponent 1.25 is based on empirical fits to BOSS exposure times. See eqn (6) of Dawson 2012 for details.

Parameters **airmass** (*float or array*) – Airmass value(s)

Returns Multiplicative factor(s) that exposure time should be adjusted based on the actual vs nominal airmass.

Return type float

desisurvey.etc.**dust_exposure_factor**(*EBV*)
Scaling of exposure time with median E(B-V) relative to nominal.

The model uses the SDSS-g extinction coefficient (3.303) from Table 6 of Schlafly & Finkbeiner 2011 by default, or config.ebv_coefficient if specified.

Parameters **EBV** (*float or array*) – Median dust extinction value(s) E(B-V) for the tile area.

Returns Multiplicative factor(s) that exposure time should be adjusted based on the actual vs nominal dust extinction.

Return type float

desisurvey.etc.**exposure_time**(*program*, *seeing*, *transparency*, *airmass*, *EBV*, *moon_frac*, *moon_sep*, *moon_alt*)

Calculate the total exposure time for specified observing conditions.

The exposure time is calculated as the time required under nominal conditions multiplied by factors to correct for actual vs nominal conditions for seeing, transparency, dust extinction, airmass, and scattered moon brightness.

Note that this function returns the total exposure time required to achieve the target SNR**2 at current conditions. The caller is responsible for adjusting this value when some signal has already been acummulated with previous exposures of a tile.

Parameters

- **program** (*'DARK', 'BRIGHT' or 'GRAY'*) – Which program to use when setting the target SNR**2.
- **seeing** (*float or array*) – FWHM seeing value(s) in arcseconds.
- **transparency** (*float or array*) – Dimensionless transparency value(s) in the range [0-1].
- **EBV** (*float or array*) – Median dust extinction value(s) E(B-V) for the tile area.
- **airmass** (*float*) – Airmass used for observing this tile.
- **moon_frac** (*float*) – Illuminated fraction of the moon, between 0-1.
- **moon_sep** (*float*) – Separation angle between field center and moon in degrees.
- **moon_alt** (*float*) – Altitude angle of the moon above the horizon in degrees.

Returns Estimated exposure time(s) with time units.

Return type astropy.unit.Quantity

desisurvey.etc.**moon_exposure_factor**(*moon_frac*, *moon_sep*, *moon_alt*, *airmass*)

Calculate exposure time factor due to scattered moonlight.

The returned factor is relative to dark conditions when the moon is below the local horizon.

This factor is based on a study of SNR for ELG targets and designed to achieve a median SNR of 7 for a typical ELG [OII] doublet at the lower flux limit of 8e-17 erg/(cm2 s A), averaged over the expected ELG target redshift distribution 0.6 < z < 1.7.

TODO: - Check the assumption that exposure time scales with SNR ** -0.5. - Check if this ELG-based analysis is also valid for BGS targets.

For details, see the jupyter notebook doc/nb/ScatteredMoon.ipynb in this package.

Parameters

- **moon_frac** (*float*) – Illuminated fraction of the moon, in the range [0,1].
- **moon_sep** (*float*) – Separation angle between field center and moon in degrees, in the range [0,180].
- **moon_alt** (*float*) – Altitude angle of the moon above the horizon in degrees, in the range [-90,90].
- **airmass** (*float*) – Airmass used for observing this tile, must be >= 1.

Returns Dimensionless factor that exposure time should be increased to account for increased sky brightness due to scattered moonlight. Will be 1 when the moon is below the horizon.

Return type float

desisurvey.etc.**seeing_exposure_factor**(*seeing*, *sbprof='ELG'*)

Scaling of exposure time with seeing, relative to nominal seeing. The model is based on DESI simulations with convolutions of realistic atmospheric and instrument PSFs, for a nominal sample of DESI ELG targets (including redshift evolution of ELG angular size).

The simulations predict SNR for the ELG [OII] doublet during dark-sky conditions at airmass X=1. The exposure factor assumes exposure time scales with SNR ** -0.5.

Parameters

- **seeing** (*float or array*) – FWHM seeing value(s) in arcseconds.
- **sbprof** (*str*) – source profile to use, one of PSF, ELG, BGS

Returns Multiplicative factor(s) that exposure time should be adjusted based on the actual vs nominal seeing.

Return type float

desisurvey.etc.**transparency_exposure_factor**(*transparency*)

Scaling of exposure time with transparency relative to nominal.

The model is that exposure time scales with 1 / transparency**2.

Parameters **transparency** (*float or array*) – Dimensionless transparency value(s) in the range [0-1].

Returns Multiplicative factor(s) that exposure time should be adjusted based on the actual vs nominal transparency.

Return type float

desisurvey.optimize

Optimize future DESI observations.

class desisurvey.optimize.**Optimizer**(*condition, lst_edges, lst_hist, subset=None, start=None, stop=None, init='flat', initial_ha=None, stretch=1.0, smoothing_radius=10, center=None, seed=123, weights=[5, 4, 3, 2, 1], completed=None*)

Initialize the hour angle assignments for specified tiles.

See DESI-3060 for details.

Parameters

- **condition** (*'DARK', 'GRAY' or 'BRIGHT'*) – Which obsconditions to optimize.
- **lst_edges** (*array*) – Array of N+1 LST bin edges.
- **lst_hist** (*array*) – Array of N bins giving the available LST distribution in units of sidereal hours per bin.
- **subset** (*array or None*) – An array of tile ID values to optimize. Optimizes all tiles with the relevant conditions if None.
- **start** (*date or None*) – Only consider available LST starting from this date. Use the nominal survey start date if None.

- **stop** (*date or None*) – Only consider available LST before this end date. Use the nominal survey stop date if None.
- **init** (*'zero', 'flat' or 'array'*) – Method for initializing tile hour angles: 'zero' sets all hour angles to zero, 'flat' matches the CDF of available LST to planned LST (without accounting for exposure time), 'array' initializes from the initial_ha argument.
- **initial_ha** (*array or None*) – Only used when init is 'array'. The subset arg must also be provided to specify which tile each HA applies to.
- **stretch** (*float*) – Amount to stretch exposure times to account for factors other than dust and airmass (i.e., seeing, transparency, moon). This does not have a big effect on the results so can be approximate.
- **smoothing_radius** (astropy.units.Quantity) – Gaussian sigma for calculating smoothing weights with angular units.
- **center** (*float or None*) – Used by the 'flat' initialization method to specify the starting DEC for the CDF balancing algorithm. When None, the 'flat' method scans over a grid of center values and picks the best one, but this is relatively slow. Ignored unless init is 'flat'.
- **seed** (*int or None*) – Random number seed to use for stochastic elements of the optimizer. Do not use None if reproducible results are required.
- **weights** (*array*) – Array of relative weights to use when selecting which LST bin to optimize next. Candidate bins are ordered by an estimated improvement. The length of the weights array determines how many candidates to consider, in decreasing order, and the weight values determines their relative weight. The next bin to optimize is then selected at random.
- **completed** (*array*) – Array of tileid, donefrac_{bright/gray/dark} indicating what fraction of particular tiles has already been observed in a particular condition.

eval_RMSE (*plan_hist*)

Evaluate the mean-squared error metric for the specified plan.

This is the metric that `optimize()` attempts to improve. It measures the similarity of the available and planned LST histogram shapes, but not their normalizations. A separate *eval_scale()* metric measures how efficiently the plan uses the available LST.

The histogram of available LST is rescaled to the same total time (area) before calculating residuals relative to the planned LST usage.

RMSE values are scaled by (10K / ntiles) so the absolute metric value is more consistent when ntiles is varied.

Parameters **plan_hist** (*array*) – Histogram of planned LST usage for all tiles.

Returns Mean squared error value.

Return type float

eval_loss (*plan_hist*)

Evaluate relative loss of current plan relative to HA=0 plan.

Calculated as (T-T0)/T0 where T is the total exposure time of the current plan and T0 is the total exposure time of an HA=0 plan.

Parameters **plan_hist** (*array*) – Histogram of planned LST usage for all tiles.

Returns Loss factor.

Return type float

eval_scale(*plan_hist*)
Evaluate the efficiency of the specified plan.

Calculates the minimum scale factor applied to the available LST histogram so that the planned LST usage is always <= the scaled available LST histogram. This value can be intepreted as the fraction of the available time required to complete all tiles.

This metric is only loosely correlated with the RMSE metric, so provides a useful independent check that the optimization is producing the desired results.

This metric is not well defined if any bin of the available LST histogram is empty, which indicates that some tiles will not be observable during the [start:stop] range being optimized. In this case, only bins with some available LST are included in the scale calculation.

Parameters **plan_hist** (*array*) – Histogram of planned LST usage for all tiles.

Returns Scale factor.

Return type float

eval_score(*plan_hist*)
Evaluate the score that improve() tries to minimize.

Score is calculated as 100 * RMSE + 100 * loss.

Parameters **plan_hist** (*array*) – Histogram of planned LST usage for all tiles.

Returns Score value.

Return type float

get_exptime(*ha*, *subset=None*)
Estimate exposure times for the specified tiles.

Estimates account for airmass and dust extinction only.

Parameters

- **ha** (*array*) – Array of hour angle assignments in degrees.
- **subset** (*array or None*) – Restrict calculation to a subset of tiles specified by the indices in this array, or use all tiles pass to the constructor if None.

Returns Tuple (exptime, subset) where exptime is an array of estimated exposure times in degrees and subset is the input subset or else a slice initialized for all tiles.

Return type tuple

get_plan(*ha*, *subset=None*)
Calculate an LST usage plan for specified hour angle assignments.

Parameters

- **ha** (*array*) – Array of hour angle assignments in degrees.
- **subset** (*array or None*) – Restrict calculation to a subset of tiles specified by the indices in this array, or use all tiles pass to the constructor if None.

Returns Array of shape (ntiles, nbins) giving the exposure time in hours that each tile needs in each LST bin. When a subset is specified, ntiles only indexes tiles in the subset.

Return type array

improve(*frac=1.0*)
Perform one iteration of improving the hour angle assignments.

Each call will adjust the HA of a single tile with a magnitude |dHA| specified by the *frac* parameter.

Parameters **frac** (*float*) – Mean fraction of an LST bin to adjust the selected tile's HA by. Actual HA adjustments are randomly distributed around this mean to smooth out adjustments.

init_smoothing(*radius*)
Calculate and save smoothing weights.

Weights for each pair of tiles [i,j] are calculated as:

```
wgt[i,j] = exp(-0.5 * (sep[i,j]/radius) ** 2)
```

where sep[i,j] is the separation angle between the tile centers.

Parameters **radius** (*astropy.units.Quantity*) – Gaussian sigma for calculating weights with angular units.

next_bin()
Select which LST bin to adjust next.

The algorithm determines which bin of the planned LST usage histogram should be decreased in order to maximize the decrease of the score, assuming that the decrease is moved to one of the neighboring bins.

Since each tile's contribution to the plan can, in general, span several LST bins and can change its area (exposure time) when its HA is adjusted, the assumptions of this algorithm are not valid in detail but it usually does a good job anyway.

This algorithm has a stochastic component controlled by the *weights* parameter passed to our constructor, in order to avoid getting stuck in a local minimum.

Returns Tuple (idx, dha_sign) where idx is the LST bin index that should be decreased (by moving one of the tiles contributing to it) and dha_sign gives the sign +/-1 of the HA adjustment required.

Return type tuple

plot(*save=None*, *relative=True*)
Plot the current optimzation status.

Requires that matplotlib is installed.

Parameters **save** (*str or None*) – Filename where the generated plot will be saved.

smooth(*alpha=0.1*)
Smooth the current HA assignments.

Each HA is replaced with a smoothed value:

```
(1-alpha) * HA + alpha * HA[avg]
```

where HA[avg] is the weighted average of all other tile HA assignments.

use_plan(*save_history=True*)
Use the current plan and update internal arrays.

Calculates the *plan_hist* arrays from the per-tile *plan_tiles* array, and records the current values of the RMSE and scale metrics.

desisurvey.optimize.**wrap**(*angle*, *offset*)
Wrap values in the range [0, 360] to [offset, offset+360].

desisurvey.plan

Plan future DESI observations.

class desisurvey.plan.**Planner**(*rules=None*, *restore=None*, *simulate=False*, *log=None*)
Coordinate afternoon planning activities.

Parameters

- **rules** (*object or None*) – Object with an `apply` method that is used to implement survey strategy by updating tile priorities each afternoon. When None, all tiles have equal priority.
- **restore** (*str or None*) – Restore internal state from the snapshot saved to this filename, or initialize a new planner when None. Use *save()* to save a snapshot to be restored later. Filename is relative to the configured output path unless an absolute path is provided. Raise a RuntimeError if the saved tile IDs do not match the current tiles_file values.
- **simulate** (*bool*) – If True, simulate fiber assignment process.
- **log** (*log object or None*) – logging object to use; None for desiutil default.

add_pending_tile(*tileid*)
Add a newly observed, now-pending tile to the pending tile list.

Updates tile availability so that this tile's neighbors will not be observed until this tile is completed.

Parameters **tileid** (*int*) –

afternoon_plan(*night*)
Update plan for a given night. Update tile availability and priority.

Parameters **night** (*str*) – night string, YYYY-MM-DD, to be planned. This argument has no effect for when Plan.simulate = False; in this case, tile availability and priority is based entirely on what files are currently present in the fiberassign directory and what the planner believes the current tile completions are.

Returns boolean arrays indicating newly observed and newly completed tiles.

Return type new_observed (array), new_completed (array)

fiberassign(*dirnames*)
Update list of tiles available for spectroscopy.

Scans given directory looking for fiberassign file and populates Plan object accordingly.

Parameters **dirnames** (*list*) – list of directory names where fiberassign files are to be found This directory is recursively scanned for all files with names matching tile-(d+).fits. TILEIDs are populated according to the name of the fiberassign file, and any header information is ignored.

fiberassign_simulate(*night*)
Update fiber assignments.

prefer_low_passnum()
Mark only tiles available that are in the lowest pass of any overlapping tiles.

save(*name*)
Save a snapshot of our current state that can be restored.

The output file has a binary table (extname PLAN) with columns TILEID, CENTERID, PASS, RA, DEC, PROGRAM, IN_DESI, EBV_MED, DESIGNHA, PRIORITY, STATUS, and DONEFRAC. Simulations also include COUNTDOWN and header keywords CADENCE, FIRST, LAST.

Parameters **name** (*str*) – Name of FITS file where the snapshot will be saved. The file will be saved under our configuration's output path unless name is already an absolute path. Pass the same name to the constructor's `restore` argument to restore this snapshot.

set_donefrac(*tileid*, *donefrac=None*, *status=None*, *ignore_pending=False*, *nobs=None*)
Update planner with new tile donefrac.

Parameters

- **tileid** (*array*) – 1D array of integer tileIDs to update
- **donefrac** (*array*) – 1D array of completion fractions for tiles, matching tileid, optional
- **status** (*array*) – 1D array of tile status to update; optional
- **ignore_pending** (*bool*) – do not mark newly started files as pending
- **nobs** (*array*) – 1D array of number of observations of each tile A tile with at least 1 observation will never be considered unobserved, regardless of whether it has zero donefrac.

survey_completed()
Test if all tiles have been completed.

desisurvey.plan.**load_design_hourangle**()
Load design hour-angle assignments from disk.

If hour angles are present in the tile file, defaults to those. Otherwise reads column 'DESIGNHA' from file saved by the `surveyinit` script. Contents must row-match the tile file.

Parameters **name** (*str*) – Name of the ecsv file to read. A relative path is assumed to refer to the output path specified in the configuration.

Returns 1D array of design hour angles in degrees, with indexing that matches *desisurvey.tiles.Tiles*.

Return type array

desisurvey.plan.**load_weather**(*start_date=None*, *stop_date=None*, *name='surveyinit.fits'*)
Load dome-open fraction expected during each night of the survey.

Reads Image HDU 'WEATHER'. This is the format saved by the `surveyinit` script, but any FITS file following the same convention can be used.

Parameters

- **name** (*str*) – Name of the FITS file to read. A relative path is assumed to refer to the output path specified in the configuration.
- **start_date** (*date or None*) – First night to include or use the first date of the survey. Must be convertible to a date using *desisurvey.utils.get_date()*.
- **stop_date** (*date or None*) – First night to include or use the last date of the survey. Must be convertible to a date using *desisurvey.utils.get_date()*.

Returns 1D array of length equal to the span between stop_date and start_date. Values are between 0 (dome closed all night) and 1 (dome open all night).

Return type array

desisurvey.plots

Utility functions for plotting DESI survey progress and planning.

desisurvey.plots.**plot_monthly**(*p*, *program='DARK'*, *monsoon=False*, *fullmoon=True*, *cmap='viridis'*, *save=None*)

Plot average nightly visibility by month.

Parameters

- **p** (*desisurvey.old.schedule.Scheduler*) – The scheduler object to use.
- **program** (*'DARK', 'GRAY', 'BRIGHT' or 'ANY'*) – Name of the program to display visibility for.
- **monsoon** (*bool*) – Do not observe during scheduled monsoon shutdowns? Ignored if `when` specifies a time.
- **fullmoon** (*bool*) – Do not observe during scheduled full-moon breaks? Ignored if `when` specifies a time.
- **cmap** (*matplotlib colormap spec*) – Colormap to use to represent observing efficiency. Not used for a time series plot.
- **save** (*string or None*) – Name of file where plot should be saved. Format is inferred from the extension.

Returns

- *tuple* – Tuple (figure, axes) returned by `plt.subplots()`.
- *Visibility does not include dust extinction or monthly weather factors.*
- *The nightly moon is displayed except for the DARK program, with an area*
- *proportional to the illuminated fraction and the nightly program time.*
- *Requires that the matplotlib and basemap packages are installed.*

desisurvey.plots.**plot_next_field**(*date_string*, *obs_num*, *ephem*, *window_size=7.0*, *max_airmass=2.0*, *min_moon_sep=50.0*, *max_bin_area=1.0*, *save=None*)

Plot diagnostics for the next field selector.

The matplotlib and basemap packages must be installed to use this function.

Parameters

- **date_string** (*string*) – Observation date of the form 'YYYYMMDD'.
- **obs_num** (*int*) – Observation number on the specified night, counting from zero.
- **ephem** (*desisurvey.ephem.Ephemerides*) – Ephemerides covering this night.

desisurvey.plots.**plot_observed**(*progress*, *include='observed'*, *start_date=None*, *stop_date=None*, *what='exptime'*, *print_summary=False*, *save=None*)

Plot a summary of observed tiles.

Reports progress tracked by `desisurvey.progress.Progress`. using *plot_sky_passes()* to display a summary of observed tiles in each pass.

Parameters

- **progress** (*desisurvey.progress.Progress*) – Progress tracker to use.
- **include** (*'all', 'observed', or 'completed'*) – Specify which tiles to include in the summary. The 'observed' selection will include tiles that have been observed at least once but have not yet reached their SNR**2 goal.

- **start_date** (*date or None*) – Plot observations starting on the night of this date, or starting with the first observation if None. Must be convertible to a date using *desisurvey.utils.get_date()*.
- **stop_date** (*date or None*) – Plot observations ending on the morning of this date, or ending with the last observation if None. Must be convertible to a date using *desisurvey.utils.get_date()*.
- **what** (*string*) – What quantity to plot for each planned tile. Must be a column name in the summary table returned by `desisurvey.progress.Progress.get_summary()`.
- **print_summary** (*bool*) – Print a summary of observed tiles.
- **save** (*string or None*) – Name of file where plot should be saved.

Returns Tuple (figure, axes) returned by `plt.subplots()`.

Return type tuple

`desisurvey.plots.`**plot_program**(*ephem*, *start_date=None*, *stop_date=None*, *style='localtime'*, *include_monsoon=False*, *include_full_moon=False*, *include_twilight=True*, *night_start=-6.5*, *night_stop=7.5*, *num_points=500*, *bg_color='lightblue'*, *save=None*)

Plot an overview of the DARK/GRAY/BRIGHT program.

Uses `desisurvey.ephem.get_program_hours()` to calculate the hours available for each program during each night.

The matplotlib and basemap packages must be installed to use this function.

Parameters

- **ephem** (*desisurvey.ephem.Ephemerides*) – Tabulated ephemerides data to use for determining the program.
- **start_date** (*date or None*) – First night to include in the plot or use the first date of the survey. Must be convertible to a date using *desisurvey.utils.get_date()*.
- **stop_date** (*date or None*) – First night to include in the plot or use the last date of the survey. Must be convertible to a date using *desisurvey.utils.get_date()*.
- **style** (*string*) – Plot style to use for the vertical axis: "localtime" shows time relative to local midnight, "histogram" shows elapsed time for each program during each night, and "cumulative" shows the cummulative time for each program since `start_date`.
- **include_monsoon** (*bool*) – Include nights during the annual monsoon shutdowns.
- **include_fullmoon** (*bool*) – Include nights during the monthly full-moon breaks.
- **include_twilight** (*bool*) – Include twilight time at the start and end of each night in the BRIGHT program.
- **night_start** (*float*) – Start of night in hours relative to local midnight used to set y-axis minimum for 'localtime' style and tabulate nightly program.
- **night_stop** (*float*) – End of night in hours relative to local midnight used to set y-axis maximum for 'localtime' style and tabulate nightly program.
- **num_points** (*int*) – Number of subdivisions of the vertical axis to use for tabulating the program during each night. The resulting resolution will be `(night_stop - night_start) / num_points` hours.
- **bg_color** (*matplotlib color*) – Axis background color to use. Must be a valid matplotlib color.

- **save** (*string or None*) – Name of file where plot should be saved. Format is inferred from the extension.

Returns Tuple (figure, axes) returned by `plt.subplots()`.

Return type tuple

`desisurvey.plots.`**plot_scheduler**(*s*, *start_date=None*, *stop_date=None*, *where=None*, *when=None*, *night_summary='dark'*, *dust=True*, *monsoon=True*, *fullmoon=True*, *weather=False*, *cmap='magma'*, *save=None*)

Plot a summary of the scheduler observing efficiency forecast.

Requires that the matplotlib and basemap packages are installed.

Parameters

- **s** (*desisurvey.old.schedule.Scheduler*) – The scheduler object to use.
- **start_date** (*date or None*) – First night to include in the plot or use the first scheduler date. Must be convertible to a date using *desisurvey.utils.get_date()*. Ignored if `when` specifies a time.
- **stop_date** (*date or None*) – First night to include in the plot or use the last scheduler date. Must be convertible to a date using *desisurvey.utils.get_date()*. Ignored if `when` specifies a time.
- **where** (*int, 'best', 'random', iterable or None*) – Plot a time series of observing efficiency each night for a specified tile ID, the best location or averaging over randomly chosen locations. An iterable of int, 'best', 'random' is also allowed. Cannot be combined with the `when` option.
- **when** (*astropy.time.Time, int, 'best', 'random' or None*) – Plot an all-sky map of observing efficiency for a specified time, each location's best night or else averaging over randomly chosen nights. A time can be specified with a timestamp or a temporal index. Cannot be combined with the `where` option.
- **night_summary** (*'best', '24hr' or 'dark'*) – Summarize the observing efficiency during each night picking either the best time slot, or else averaging over 24 hours or the actual length of the night. Ignored if `when` specifies a time.
- **dust** (*bool*) – Should dust extinction be included in the observing efficiency?
- **monsoon** (*bool*) – Do not observe during scheduled monsoon shutdowns? Ignored if `when` specifies a time.
- **fullmoon** (*bool*) – Do not observe during scheduled full-moon breaks? Ignored if `when` specifies a time.
- **weather** (*bool*) – Reweight exposure factors by expected dome-open fraction each month. Ignored if `when` specifies a time.
- **cmap** (*matplotlib colormap spec*) – Colormap to use to represent observing efficiency. Not used for a time series plot.
- **save** (*string or None*) – Name of file where plot should be saved. Format is inferred from the extension.

Returns Tuple (figure, axes) returned by `plt.subplots()`.

Return type tuple

desisurvey.plots.**plot_sky_passes**(*ra*, *dec*, *passnum*, *z*, *clip_lo=None*, *clip_hi=None*, *label='label'*, *cmap='viridis'*, *save=None*)

Plot sky maps for each pass of a per-tile scalar quantity.

The matplotlib and basemap packages must be installed to use this function.

Parameters

- **ra** (*array*) – Array of RA values to use in degrees.
- **dec** (*array*) – Array of DEC values to use in degrees.
- **pass** (*array*) – Array of integer pass values to use.
- **z** (*array*) – Array of per-tile values to plot.
- **clip_lo** (*float or string or None*) – See `desiutil.plot.prepare_data()`
- **clip_hi** (*float or string or None*) – See `desiutil.plot.prepare_data()`
- **label** (*string*) – Brief description of per-tile value `z` to use for axis labels.
- **cmap** (*colormap name or object*) – Matplotlib colormap to use for mapping data values to colors.
- **save** (*string or None*) – Name of file where plot should be saved. Format is inferred from the extension.

Returns Tuple (figure, axes) returned by `plt.subplots()`.

Return type tuple

desisurvey.rules

Manage and apply tile observing priorities using rules.

class desisurvey.rules.**Rules**(*file_name='rules.yaml'*)

Load rules from the specified file.

Read tile group definitions and observing rules from the specified YAML file.

Parameters **file_name** (*str*) – Name of YAML file containing the rules to use. A relative path refers to our configured output path.

apply(*donefrac*)

Apply rules to determine tile priorites based on those completed so far.

Parameters **completed** (*array*) – Boolean array of per-tile completion status.

Returns Array of per-tile observing priorities.

Return type array

desisurvey.tiles

Manage static information associated with tiles, programs and passes.

Each tile has an assigned program name. The program names (DARK, BRIGHT) are predefined in terms of conditions on the ephemerides, but not all programs need to be present in a tiles file. Pass numbers are arbitrary integers and do not need to be consecutive or dense.

To ensure consistent and efficient usage of static tile info, all code should use:

```
tiles = desisurvey.tiles.get_tiles()
```

To use a non-standard tiles file, change the configuration before the first call to `get_tiles()` with:

```
config = desisurvey.config.Configuration()
config.tiles_file.set_value(name)
```

The *Tiles* class returned by *get_tiles()* is a wrapper around the FITS table contained in a tiles file, that adds some precomputed derived attributes for consistency and efficiency.

class desisurvey.tiles.**Tiles**(*tiles_file=None*)

Manage static info associated with the tiles file.

Parameters **tile_file** (*str or None*) – Name of the tiles file to use or None for the default specified in our configuration.

_calculate_neighbors()

Initialize attribute _neighbors. A neigbor is defined as a tile in the same pass within 3 * config.tile_radius if config.tiles_lowpass is True. Otherwise, it's all tiles within the program within 3 * config.tile_radius.

This is relatively slow, so only used the first time `overlapping` properties are accessed.

_calculate_overlaps()

Initialize attributes _overlapping.

Uses the config parameters `fiber_assignment_delay` and `tile_diameter` to determine overlap dependencies.

This is relatively slow, so only used the first time `overlapping` properties are accessed.

airmass(*hour_angle*, *mask=None*)

Calculate tile airmass given hour angle.

Parameters

- **hour_angle** (*array*) – Array of hour angles in degrees to use. If mask is None, then should have length `self.ntiles`. Otherwise, should have a value per non-zero entry in the mask.
- **mask** (*array or None*) – Boolean mask of which tiles to perform the calculation for.

Returns Array of airmasses corresponding to each input hour angle.

Return type array

airmass_at_mjd(*mjd*, *mask=None*)

Calculate tile airmass at given MJD.

Parameters

- **mjd** (*array*) – Array of MJD to use. If mask is None, then should have length `self.ntiles`. Otherwise, should have a value per non-zero entry in the mask.
- **mask** (*array or None*) – Boolean mask of which tiles to perform the calculation for.

Returns Array of airmasses corresponding to each input hour angle.

Return type array

airmass_second_derivative(*HA*, *mask=None*)

Calculate second derivative of airmass with HA.

Useful for determining how close to design airmass we have to get for different tiles. When this is large, we really need to observe things right at their design angles. When it's small, we have more flexibility.

fiberassign_delay
Delay between covering a tile and when it can be fiber assigned.

Units are determined by the value of the fiber_assignment_cadence configuration parameter.

index(*tileID*, *return_mask=False*)
Map tile ID to array index.

Parameters

- **tileID** (*int or array*) – Tile ID value(s) to convert.
- **mask** (*bool*) – if mask=True, an additional mask array is returned, indicating which IDs were present in the tile array. Otherwise, an exception is raised if tiles were not found.

Returns Index into internal per-tile arrays corresponding to each input tile ID.

Return type int or array

neighbors
Dictionary of tile neighbor matrices.

neighbors[i] is the list of tile row numbers that neighbor the tile with row number i within a pass.

Neighboring tiles are only computed within a program and pass.

overlapping
Dictionary of tile overlap matrices.

overlapping[i] is the list of tile row numbers that overlap the tile with row number i.

Overlapping tiles are only computed within a program; a tile cannot overlap a tile of a different program. If fiber_assignment_delay is negative, tile do not overlap one another within a program.

read_tiles_table()
Read and trim the tiles table.

Must be called after self.tiles_file and self.nogray member variables have been set.

desisurvey.tiles.**get_nominal_program_times**(*tileprogram*, *config=None*, *return_timetypes=False*)
Return nominal times for given programs in seconds.

desisurvey.tiles.**get_tiles**(*tiles_file=None*, *use_cache=True*, *write_cache=True*)
Return a Tiles object with optional caching.

You should normally always use the default arguments to ensure that tiles are defined consistently and efficiently between different classes.

Parameters

- **tiles_file** (*str or None*) – Use the specified name to override config.tiles_file.
- **use_cache** (*bool*) – Use tiles previously cached in memory when True. Otherwise, (re)load tiles from disk.
- **write_cache** (*bool*) – If tiles need to be loaded from disk with this call, save them in a memory cache for future calls.

desisurvey.scheduler

Schedule observations during an observing night.

This module supercedes desisurvey.old.schedule.

class desisurvey.scheduler.**Scheduler**(*plan*, *log=None*)

Create a new next-tile scheduler.

Design hour angles are read from the output of `surveyinit` using *desisurvey.plan.load_design_hourangle()*, by default.

The only internal state needed by the scheduler is the list of accumulated SNR2 fractions per tile, which can be restored from a file created using `save()`.

A newly created or restored scheduler must be configured with *init_night()* (to precompute data for a night's observing) before tiles can be selected.

Use *next_tile()* to select the next tile to observe during a night. If the tile is observed, the internal state must be updated with a call to *update_snr()*.

Parameters

- **plan** (*desisurvey.plan.Plan instance to use for planning*) –
- **log** (*log object to use*) –

init_night(*night*, *use_twilight=False*)

Initialize scheduling for the specified night.

Must be called before calls to *next_tile()*.

The pool of available tiles during the night consists of those that:

- Have fibers assigned.
- Have non-zero priority (aka "planned").
- Have not already reached their target SNR (aka "completed").
- Are not too close to a planet during this night.

Tile priority is assumed fixed during the night. When the moon is up, tiles are also vetoed if they are too close to the moon. The angles that define "too close" to a planet or the moon are specified in config.avoid_bodies.

Parameters

- **night** (*str*) – Date on the evening this night starts in the format YYYY-MM-DD.
- **use_twilight** (*bool*) – Include twilight when calculating the scheduled program changes during this night when True.
- **verbose** (*bool*) – Generate verbose logging output when True.

next_tile(*mjd_now*, *ETC*, *seeing*, *transp*, *skylevel*, *HA_sigma=15.0*, *greediness=0.0*, *program=None*, *verbose=False*, *current_ra=None*, *current_dec=None*, *speed=None*)

Select the next tile to observe.

The *init_night()* method must be called before calling this method during a night.

The (log) score for each observable tile is calculated as:

$$-(1-g)\,\frac{1}{2}\left(\frac{\mathrm{HA}-\mathrm{HA}_0}{\sigma_{\mathrm{HA}}}\right)^2 - g\log\frac{t_{\mathrm{exp}}}{t_{\mathrm{nom}}} + \log P$$

where HA and HA_0 are the current and design hour angles, respectively, g is the `greediness` parameter below, and P are the tile priorities used to implement survey strategy.

Parameters

- **mjd_now** (*float*) – Time when the decision is being made.

- **ETC** (*desisurvey.etc.ExposureTimeCalculator*) – Object with methods could_complete() and weather_factor(). Normally an instance of *desisurvey.etc.ExposureTimeCalculator*.
- **seeing** (*float*) – Estimate of current atmospherid seeing in arcseconds.
- **transp** (*float*) – Estimate of current atmospheric transparency in the range 0-1.
- **HA_sigma** (*float*) – RMS in degrees for the Gaussian penalty applied to tiles observed away from their design hour angle.
- **greediness** (*float*) – Parameter that controls the balance between observing at the design hour angle and observing tiles with the small exposure-time factor. Set this value to zero to only consider hour angle or to one to only consider isntantaneous efficiency. The meaning of intermediate values will depend on the value of HA_sigma and how exposure factors are calculated. Refer to the equation above for details. Must be between 0 and 1.
- **program** (*string*) – PROGRAM of tile to select. Default of None selects the appropriate PROGRAM given current moon/twilight conditions. Forcing a particular program leads PROGEND to be infinity.
- **current_ra** (*float*) – current ra of telescope; used for computing penalties to long slews
- **current_dec** (*float*) – current dec of telescope; used for computing penalties to long slews
- **speed** (*dict*) – dictionary of DARK, BRIGHT, BACKUP, giving current estimated survey speeds in each program. If present, used instead of transp/skylevel/seeing to estimate exposure times and pick programs.

Returns

Tuple (TILEID,PROGRAM,DONEFRAC,EXPFAC,AIRMASS,PROGRAM,PROGEND) giving the ID and associated properties of the selected tile. When no tile is observable, only the last two tuple fields will be valid, and this method should be called again after some dead-time delay. The tuple fields are:

- TILEID: ID of the tile to observe.
- PROGRAM: program of the tile to observe.
- DONEFRAC: fractional SNR2 already accumulated for the selected tile.
- EXPFAC: initial exposure-time factor for the selected tile.
- AIRMASS: initial airmass of the selected tile.
- SCHEDPROGRAM: scheduled program at mjd_now, which might be different from the program of the selected (TILEID, PASSNUM).
- PROGEND: MJD timestamp when the scheduled program ends.

Return type tuple

select_program(*mjd_now*, *ETC*, *verbose=False*, *seeing=None*, *transparency=None*, *skylevel=None*, *airmass=None*, *speed=None*)

Select program to observe now.

update_snr(*tileID*, *donefrac*)

Update SNR for one tile.

A tile whose update donefrac exceeds the min_snr2frac configuration parameter will be considered completed, and not scheduled for future observing.

Parameters

- **tileID** (*int*) – ID of the tile to update.
- **donefrac** (*float*) – New value of the fractional SNR2 accumulated for this tile, including all previous exposures.

desisurvey.forecast

Simple forecast of survey progress and margin.

class desisurvey.forecast.**Forecast**(*start_date=None*, *stop_date=None*, *use_twilight=False*, *weather=None*, *design_hourangle=None*)

Compute a simple forecast of survey progress and margin.

Based on config, ephemerides, tiles.

Parameters

- **start_date** (*datetime.date*) – Forecast for survey that starts on the evening of this date.
- **stop_date** (*datetime.date*) – Forecast for survey that stops on the morning of this date.
- **use_twilight** (*bool*) – Include twilight time in the forecast scheduled time?
- **weather** (*array or None*) – 1D array of nightly weather factors (0-1) to use, or None to use *desisurvey.plan.load_weather()*. The array length must equal the number of nights in [start,stop). Values are fraction of the night with the dome open (0=never, 1=always). Use 1 - `desimodel.weather.dome_closed_fractions()` to lookup suitable corrections based on historical weather data.
- **design_hourangle** (*array or None*) – 1D array of design hour angles to use in degrees, or None to use *desisurvey.plan.load_design_hourangle()*.

summary(*width=7*, *prec=5*, *separator='-'*)

Print a summary table of the forecast parameters.

desisurvey.utils

Utility functions for survey planning and scheduling.

desisurvey.utils.**cos_zenith**(*ha*, *dec*, *latitude=None*)

Calculate cos(zenith) for specified hour angle, DEC and latitude.

Combine with *cos_zenith_to_airmass()* to calculate airmass.

Parameters

- **ha** (*astropy.units.Quantity*) – Hour angle(s) to use, with units convertible to angle.
- **dec** (*astropy.units.Quantity*) – Declination angle(s) to use, with units convertible to angle.
- **latitude** (*astropy.units.Quantity or None*) – Latitude angle to use, with units convertible to angle. Defaults to the latitude of *get_location()* if None.

Returns cosine of zenith angle(s) corresponding to the inputs.

Return type numpy array

desisurvey.utils.**cos_zenith_to_airmass**(*cosZ*)
Convert a zenith angle to an airmass.

Uses the Rozenberg 1966 interpolation formula, which gives reasonable results for high zenith angles, with a horizon air mass of 40. https://en.wikipedia.org/wiki/Air_mass_(astronomy)#Interpolative_formulas Rozenberg, G. V. 1966. "Twilight: A Study in Atmospheric Optics." New York: Plenum Press, 160.

The value of cosZ is clipped to [0,1], so observations below the horizon return the horizon value (~40).

Parameters **cosZ** (*float or array*) – Cosine of angle(s) to convert.

Returns Airmass value(s) >= 1.

Return type float or array

desisurvey.utils.**day_number**(*date*)
Return the number of elapsed days since the start of the survey.

Does not perform any range check that the date is within the nominal survey schedule.

Parameters **date** (*astropy.time.Time, datetime.date, datetime.datetime, string or number*) – Converted to a date using *get_date()*.

Returns Number of elapsed days since the start of the survey.

Return type int

desisurvey.utils.**get_airmass**(*when*, *ra*, *dec*)
Return the airmass of (ra,dec) at the specified observing time.

Uses *cos_zenith_to_airmass()*.

Parameters

- **when** (*astropy.time.Time*) – Observation time, which specifies the local zenith.
- **ra** (*astropy.units.Quantity*) – Target RA angle(s)
- **dec** (*astropy.units.Quantity*) – Target DEC angle(s)

Returns Value of the airmass for each input (ra,dec).

Return type array or float

desisurvey.utils.**get_average_dome_closed_fractions**(*first*, *last*, *smooth=7*)
Get daily averaged dome-closed fractions between first and last.

Returns the fraction of the time the dome is closed on each night.

Parameters

- **first** (*datetime.date*) – Date of first night.
- **last** (*datetime.date*) – Date of last night. Survey stops the morning of this date.
- **smooth** (*float*) – Number of days to smooth dome closed fraction by.

Returns

Return type np.ndarray giving dome closed fraction on each night.

desisurvey.utils.**get_current_date**()
Give current date following get_date convention (date changes at noon).

Returns

Return type datetime.date object for current night, following get_date convention

desisurvey.utils.**get_date**(*date*)
Convert different date specifications into a datetime.date object.

We use strptime() to convert an input string, so leading zeros are not required for strings in the format YYYY-MM-DD, e.g. 2019-8-3 is considered valid.

Instead of testing the input type, we try different conversion methods: `.datetime.date()` for an astropy time and `datetime.date()` for a datetime.

Date specifications that include a time of day (datetime, astropy time, MJD) are rounded down to the previous local noon before converting to a date. This ensures that all times during a local observing night are mapped to the same date, when the night started. A "naive" (un-localized) datetime is assumed to refer to UTC.

Generates astropy ERFA warnings for future dates.

Parameters **date** (*astropy.time.Time, datetime.date, datetime.datetime, string or number*) – Specification of the date to return. A string must have the format YYYY-MM-DD (but leading zeros on MM and DD are optional). A number will be interpreted as a UTC MJD value.

Returns

Return type datetime.date

desisurvey.utils.**get_location**()
Return the telescope's earth location.

The location object is cached after the first call, so there is no need to cache this function's return value externally.

Returns

Return type astropy.coordinates.EarthLocation

desisurvey.utils.**get_observer**(*when*, *alt=None*, *az=None*)
Return the AltAz frame for the telescope at the specified time(s).

Refraction corrections are not applied (for now).

The returned object is automatically broadcast over input arrays.

Parameters

- **when** (*astropy.time.Time*) – One or more times when the AltAz transformations should be calculated.
- **alt** (*astropy.units.Quantity or None*) – Local altitude angle(s)
- **az** (*astropy.units.Quantity or None*) – Local azimuth angle(s)

Returns AltAz frame object suitable for transforming to/from local horizon (alt, az) coordinates.

Return type astropy.coordinates.AltAz

desisurvey.utils.**is_monsoon**(*night*)
Test if this night's observing falls in the monsoon shutdown.

Uses the monsoon date ranges defined in the *desisurvey.config.Configuration*.

Parameters **night** (*date*) – Converted to a date using *desisurvey.utils.get_date()*.

Returns True if this night's observing falls during the monsoon shutdown.

Return type bool

desisurvey.utils.**local_noon_on_date**(*day*)
Convert a date to an astropy time at local noon.

Local noon is used as the separator between observing nights. The purpose of this function is to standardize the boundary between observing nights and the mapping of dates to times.

Generates astropy ErfaWarnings for times in the future.

Parameters **day** (*datetime.date*) – The day to use for generating a time object.

Returns A Time object with the input date and a time corresponding to local noon at the telescope.

Return type astropy.time.Time

desisurvey.utils.**match**(*a*, *b*)
Find matching elements of b in unique array a by index.

Returns indices ma, mb such that a[ma] == b[mb]

Parameters

- **a** (*unique array*) –
- **b** (*array*) –

Returns **ma, mb**

Return type indices such that a[ma] == b[mb]

desisurvey.utils.**night_to_str**(*date*)
Return DESI string format (YYYYMMDD) of datetime night.

Parameters **date** (*datetime.date object, as from get_date()*) –

Returns YYYMMDD formatted date string

Return type str

desisurvey.utils.**separation_matrix**(*ra1*, *dec1*, *ra2*, *dec2*, *max_separation=None*)
Build a matrix of pair-wise separation between (ra,dec) pointings.

The ra1 and dec1 arrays must have the same shape. The ra2 and dec2 arrays must also have the same shape, but it can be different from the (ra1,dec1) shape, resulting in a non-square return matrix.

Uses the Haversine formula for better accuracy at low separations. See https://en.wikipedia.org/wiki/Haversine_formula for details.

Equivalent to using the separations() method of astropy.coordinates.ICRS, but faster since it bypasses any units.

Parameters

- **ra1** (*array*) – 1D array of n1 RA coordinates in degrees (without units attached).
- **dec1** (*array*) – 1D array of n1 DEC coordinates in degrees (without units attached).
- **ra2** (*array*) – 1D array of n2 RA coordinates in degrees (without units attached).
- **dec2** (*array*) – 1D array of n2 DEC coordinates in degrees (without units attached).
- **max_separation** (*float or None*) – When present, the matrix elements are replaced with booleans given by (value <= max_separation), which saves some computation.

Returns Array with shape (n1,n2) with element [i1,i2] giving the 3D separation angle between (ra1[i1],dec1[i1]) and (ra2[i2],dec2[i2]) in degrees or, if max_separation is not None, booleans (value <= max_separation).

Return type array

desisurvey.utils.**slewtime**(*ra1*, *dec1*, *ra2*, *dec2*, *freeslewtime=10*, *ignore_positive_ra=False*)
Estimate slew times.

Uses slew model from DESI-3687. Assumes that freeslewtime s of slew time is "free"—i.e., it can be overlapped with other overheads.

Parameters

- **ra1** (*float*) – right ascension (deg)
- **dec1** (*float*) – declination (deg)
- **ra2** (*float*) – right ascension (deg)
- **dec2** (*float*) – declination (deg)
- **freeslewtime** (*float*) – amount of time during which one can slew "for free" (s)
- **ignore_positive_ra** (*float*) – if True, slew time in the positive RA direction doesn't count. Intended to provide no penalty for slews that are just keeping up with the sky. In this case, slews in dec don't count if they fit within the slew in RA.

Returns

Return type Estimated slew time in s needed to reach target.

desisurvey.utils.**yesno**(*question*)
Simple Yes/No Function.

1.3.2 Command-Line Scripts

surveyinit

Script wrapper for initializing survey planning and scheduling.

This is normally run once, at the start of the survey, and saves its results to a FITS file surveyinit.fits. With the default parameters, the running time is about 25 minutes.

This script will calculate the ephemerides and expected weather (dome open fraction) for 2019-2025, then calculate design hour angles for the nominal survey dates. The results are saved in a file (normally `surveyinit.fits`) and then do not need to be recalculated ever again.

To run this script from the command line, use the `surveyinit` entry point that is created when this package is installed, and should be in your shell command search path.

desisurvey.scripts.surveyinit.**calculate_initial_plan**(*args*)
Calculate the initial survey plan.

Use *desisurvey.plan.load_weather()* and `desisurvey.plan.load_design_hourangles()` to retrieve these data from the saved plan.

Parameters **args** (*object*) – Object with attributes for parsed command-line arguments.

desisurvey.scripts.surveyinit.**main**(*args*)
Command-line driver for initializing the survey plan.

desisurvey.scripts.surveyinit.**parse**(*options=None*)
Parse command-line options for running survey planning.

surveymovie

Script wrapper for creating a movie of survey progress.

To run this script from the command line, use the `surveymovie` entry point that is created when this package is installed, and should be in your shell command search path.

The optional matplotlib python package must be installed to use this script.

The external program ffmpeg must be installed to use this script. At nersc, try `module add ffmpeg`.

class desisurvey.scripts.surveymovie.**Animator**(*exposures_path*, *start*, *stop*, *label*, *show_scores*)

Manage animation of survey progress.

draw_exposure(*iexp*, *nightly*)

Draw the frame for a single exposure.

Calls *init_date()* if this is the first exposure of the night that we have seen.

Parameters **iexp** (*int*) – Index of the exposure to draw.

Returns True if a new frame was drawn.

Return type bool

init_date(*date*, *ephem*)

Initialize before drawing frames for a new night.

Parameters

- **date** (*datetime.date*) – Date on which this night's observing starts.
- **night** (*astropy.table.Column*) – Ephemerides data for this night.

init_figure(*nightly*, *width=1920*, *height=1080*, *dpi=32*)

Initialize matplot artists for drawing each frame.

desisurvey.scripts.surveymovie.**main**(*args*)

Command-line driver to visualize survey scheduling and progress.

desisurvey.scripts.surveymovie.**parse**(*options=None*)

Parse command-line options for running survey planning.

desisurvey.scripts.surveymovie.**wrap**(*angle*, *offset=-60*)

Wrap values in the range [0, 360] to [offset, offset+360].

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

Symbols

A

C

D

E

F

G

I

K

L

M

N

O

P

R

S

T

U

W

Y