surveysim Documentation

Release 0.12.5

DESI

Jan 13, 2023

Contents

CHAPTER 1

Contents

1.1 surveysim change log

1.1.1 0.12.6 (unreleased)

- No changes yet.

1.1.2 0.12.5 (2023-01-12)

- Add –existing-exposures argument to surveysim. Pointing the surveysim to an existing tiles and exposures file makes a survey simulation starting the day following the night of the last exposure. This enables simulation of the remainder of the survey.
- Note: tag 0.12.4 was corrupted and skipped

1.1.3 0.12.3 (2021-07-06)

- Add current_ra/current_dec to scheduler.next_tile to enable slew optimization.
- Add extra-downtime argument to surveysim randomly mark nights as bad, approximately modeling extra sources of downtime.

1.1.4 0.12.2 (2021-03-31)

- Use updated desisurvey state & config files. (PR *#78*)
- Work in no-pass and no-gray modes, eliminating the notion of pass and merging the dark and gray layers. Update API to match associated desisurvey changes. (PR #77)
- Use variable sky in surveysim, albeit presently only from ephemerides. (PR #76)
- Use consistent conditions in scheduler.next_tile and in ETC.start (PR #75)

1.1.5 0.12.1 (2020-12-11)

- Drop py3.5 for travis testing (PR #71).
- fix test_weather to 2 months instead of one (commit 4d9ceb3).
- fix EXPID int vs. str bug (commit 68c7088).

1.1.6 0.12.0 (2020-08-03)

- Update surveysim to match recent desisurvey updates, particularly regarding fiber assignment (PR #70).

1.1.7 0.11.0 (2019-08-09)

- Travis testing fixes (PR #66)
- Pass dummy sky level to desisurvey scheduler.next_tile; needed to match API change in desisurvey PR #99. (surveysim PR #64). Requires desisurvey 0.12.0 or later.

1.1.8 0.10.1 (2018-12-16)

- Include EXTNAME in output files (PR #63).

1.1.9 0.10.0 (2018-11-26)

This version is a major refactoring of the code to take advantage of the refactoring in desisurvey 0.11.0 and simplify the simulation classes used to track survey statistics and exposure metadata (PR #60).

- Add new modules: exposures, stats.
- Add new simulate_night that supercedes the original nightOps.
- Use new Scheduler and ExposureTimeCalculator from desisurvey.
- Requires desisurvey 0.11.0.
- Refactor desisurvey.ephemerides -> desisurvey.ephem and use get_ephem().
- Update the tutorial.

1.1.10 0.9.2 (2018-10-02)

- Replay historical Mayall daily weather.
- Implement partial-night dome closure.
- Requires desimodel >= 0.9.8 and desisurvey >= 0.10.4.

1.1.11 0.9.1 (2018-06-27)

- Do arc exposures before flat exposures (PR #57).

1.1.12 0.9.0 (2017-11-09)

- Add `surveysim.util.add_calibration_exposures()`, to add simulated calibration exposures to a set of science exposures (PR #55).

1.1.13 0.8.2 (2017-10-09)

- Use new desisurvey config api (requires desisurvey >= 0.9.3)
- Add support for optional depth-first survey strategy.
- Docs now auto-generated at http://surveysim.readthedocs.io/en/latest/

1.1.14 0.8.1 (2017-09-20)

- Adds surveysim –config-file option (PR #49); requires desisurvey/0.9.1.

1.1.15 0.8.0 (2017-09-11)

- Track API changes in desisurvey 0.9.0.
- The surveysim script is now called once per night, alternating with a surveyplan script that lives in the desisurvey package.
- See https://www.youtube.com/watch?v=vO1QZD_aCIo for a visualization of the full 5-year survey simulation that matches DESI-doc-1767-v3.

1.1.16 0.7.1 (2017-08-07)

- Use new desimodel.weather to randomly sample seeing and transparency. Requires desimodel >= 0.8.0.

1.1.17 0.7.0 (2017-06-18)

- First implementation of fiber-assignment groups and priorities.
- Integration with the new desisurvey surveyplan script.
- Create tutorial document and sample automation scripts.

1.1.18 0.6.0 (2017-06-05)

- Add strategy, weights options to surveysim script.
- Add hooks for using greedy scheduler
- Terminate exposures at sunset

1.1.19 0.5.0 (2017-05-10)

- Use desisurvey.config to manage all non-simulation configuration data.
- Unify different output files with overlapping contents into single output managed by desisurvey.progress.
- Overhaul of weather simulator to generate continuous stationary time series that are independent of the observing sequence. Use desimodel.seeing.
- Simulate multiple exposures for cosmics and more realistic overhead.
- Clean up of README, docstrings, imports, unit tests, requirements, unused code.

1.1.20 0.4.1 (2017-04-13)

- Fixed package names to work with desisurvey >= 0.4.0

1.1.21 0.4.0 (2017-04-04)

- Adds unit tests
- removes data/tile-info.fits (not used here; was moved to desisurvey)
- adds nightops.py (from desisurvey, used here but not there)
- create surveysim command-line script
- use new desisurvey config machinery (first steps, in progress)

1.1.22 0.3.1 (2016-12-21)

- Fixed outlier HA tile assignments around RA 200-220 (PR #26)
- Added 7 day shutdown around full moon (PR #25)

1.1.23 0.3.0 (2016-11-29)

- Moved non-simulation specific parts to desisurvey

1.1.24 0.2.0 (2016-11-18)

- Modified some file names
- Moved some functions from one file to another

1.1.25 0.1.1 (2016-11-14)

- fixed crash at end and data/ install (PR #3)
- initial tests for NERSC install

1.1.26 0.1.0 and prior

- No changes.rst yet

1.2 Tutorial Guide to Running DESI Survey Simulations

1.2.1 Introduction

The instructions below simulate the DESI survey as a sequence of tile exposures and record metadata for each exposure and accumulate some summary statistics. Simulations are stochastic since the scheduling algorithms respond to randomly generated weather conditions.

These instructions do not perform fiber assignment or simulate any spectra, but instead provide the necessary scheduling inputs for these tasks.

Please create an issue with any corrections or suggestions for improvement to this tutorial.

1.2.2 Quick Start

Login to cori, then:

```
source /project/projectdirs/desi/software/desi_environment.sh 18.12
mkdir -p $SCRATCH/desi/output
export DESISURVEY_OUTPUT=$SCRATCH/desi/output
surveyinit
surveysim
```

The results are then saved as `stats_surveysim.fits` and `exposures_surveysim.fits` in `$DESISURVEY_OUTPUT`. For a tutorial on interpreting these outputs start here.

For more details and variations on these steps, read on.

1.2.3 Install Software

Requirements

If this is your first exposure to DESI software, start here. We use git for source control and you will need to install the base DESI packages on your laptop or else work at NERSC.

The following DESI packages must be installed to run this tutorial:

- specsim
- desiutil
- desimodel
- desisurvey
- surveysim

In addition, the following non-DESI packages must be installed via pip since they are not included with the anaconda distribution:

- fitsio
- speclite

- ephem
- healpy

Note that these packages are already included in the custom DESI anaconda distribution installed at NERSC, so only need to be installed when running on your laptop or if you need to use non-default versions.

NERSC Installation

Setup the standard DESI conda environment using, for example:

```
source /project/projectdirs/desi/software/desi_environment.sh 18.12
```

Replace `18.12` with `master` for the latest and greatest (which might not work), or leave it out for the current default.

If you want to use new features of `desisurvey` or `surveysim` that are not yet included in a numbered DESI conda environment, swap them in using:

```
module swap desisurvey/master
module swap surveysim/master
```

For even more bleeding-edge features that are only available on a development branch, use, for example:

```
module unload desisurvey
pip install --user git+https://github.com/desihub/desisurvey@refactor
export PATH=$HOME/.local/bin:$PATH
```

where `refactor` is the branch name in this example.

Laptop Installation

The following instructions assume that you have installed the anaconda scientific python distribution and will create a new python environment for your DESI work. Start from the directory you wish to install software into, then:

```
conda create --name desi pip ipython jupyter numpy scipy astropy pyyaml requests h5py␣
↪scikit-learn matplotlib basemap
source activate desi
pip install fitsio speclite ephem healpy
for package in specsim desiutil desimodel desisurvey surveysim; do
    pip install git+https://github.com/desihub/$package
done
export DESIMODEL=$PWD/desimodel
install_desimodel_data -d $DESIMODEL
```

Notes for experts:

- The instructions above assume that you are using the bash shell, and need to be modified slightly for (t)csh.
- The matplotlib and basemap packages are not required to follow the instructions below but are useful for plotting the outputs.

1.2.4 Setup Environment

In General

Create an output directory to hold all survey planning and simulation outputs and create an environment variable pointing to it.

Ensure that your `$DESIMODEL` environment variable points to a valid data directory:

```
ls $DESIMODEL/data/weather
```

Also check that the relevant command-line scripts are in your path:

```
surveyinit --help
surveysim --help
```

Note that all output from these commands goes into `$DESISURVEY_OUTPUT` so they can be run from any directory and will not write anything to your current working directory.

NERSC Environment

Save the output to the `$SCRATCH` volume, for example:

```
mkdir -p $SCRATCH/desi/output
export DESISURVEY_OUTPUT=$SCRATCH/desi/output
```

Note that we use `$SCRATCH` for faster I/O but files are periodically removed. See NERSC best practices for details.

Laptop Environment

Enter the parent directory where you will save outputs, then:

```
mkdir output
export DESISURVEY_OUTPUT=$PWD/output
```

If you followed the installation recipe above then make sure you have activated your `desi` environment with:

```
conda activate desi
```

(Older versions of `conda` might require `source activate desi` instead.)

1.2.5 Configuration

Parameters for planning and scheduling the DESI survey are stored in a configuration file which is well commented and provides a good overview of the assumptions being made. You do not normally need to change these parameters, but are welcome to experiment by copying and editing this file then passing your custom version to the `surveyinit` and `surveysim` scripts described below using their `config-file` option.

1.2.6 Initialize Survey Planning

Before starting the survey, we precompute some tabulated planning data using:

```
surveyinit --verbose
```

This step takes about 50 minutes (on cori) and writes the following files into `$DESISURVEY_OUTPUT`:

- `ephem_2019-01-01_2025-12-31.fits`: tabulated ephemerides during 2019-25.
- `surveyinit.fits`: estimated average weather and optimized initial hour angle (HA) assignments for each tile.

These files take some time to generate, but are cached and not regenerated after the first time you run this command. If you want to force these files to be recalculated, add the `--recalc` option. To avoid generating these files yourself, you can also copy them into your `$DESISURVEY_OUTPUT` from this NERSC directory:

```
$DESI_ROOT/datachallenge/surveysim2018/shared/
```

To ensure they have been copied correctly, you should still run `surveyinit --verbose`, which should now exit immediately.

1.2.7 Simulate Observations

To simulate the nomimal 5-year survey, use:

```
surveysim
```

This should complete in about 2 minutes (on cori) and writes two FITS files to `$DESISURVEY_OUTPUT`:

- `stats.fits`: tables of per-tile and per-night summary statistics.
- `exposures.fits`: table listing all simulated exposures in time order.

For a tutorial on interpreting these outputs start here.

By default, simulations are run entirely in memory for speed. However, during operations the internal states of the afternoon planner and tile scheduler are written to disk daily and then restored the next day. Use the `--save-restore` option to `surveysim` to run in this mode, and write daily files:

- `planner_YYYY-MM-DD.fits`: internal state of the planner after afternoon planning for YYYY-MM-DD.
- `scheduler_YYYY-MM-DD.fits`: internal state of the tile scheduler after observing on the night of YYYY-MM-DD.

This mode gives identical results but is slower (about 3 minutes) and writes many files (about 3.6K files totalling amost 1Gb), so is mainly intended as a technical check of this mode and for developing tools that read these intermediate files.

Variations

There are many options you can experiment with to simulate a different survey weather, strategy, or schedule, for example. For a full list, refer to:

```
surveyinit --help
surveysim --help
```

You can also vary parameters in the survey configuration file.

In order to keep the outputs from different runs separate, use a separate output directory each time a change to the `surveyinit` outputs is required. For example, when changing the tiles file or nominal survey start/stop dates. To run with a different output directory you can either update `$DESISURVEY_OUTPUT` or else use the `--output-path` option of `surveyinit` and `surveysim`.

For different runs with the same `surveyinit` outputs, use the `--name` and `--comment` options to `surveysim` to distinguish each run. For example:

```
surveysim --name twilight --comment 'Include twilight in schedule' --twilight
```

Will run with twilight included in the schedule and save *stats_twilight.fits* and *exposures_twilight.fits* to `$DESISURVEY_OUTPUT`.

To study how survey progress depends on the random weather realization (including seeing and transparency), change the default seed (1), for example:

```
surveysim --name weather1 --comment 'Random weather realization #1' --seed 1
surveysim --name weather2 --comment 'Random weather realization #2' --seed 2
surveysim --name weather3 --comment 'Random weather realization #3' --seed 3
```

To simulate with an estimate of the worst-case weather, replay the historical dome-open fractions from 2015 during each year of the simulation with:

```
surveysim --name worstcase --comment 'Worst-case dome-open fractions' --replay Y2015
```

1.2.8 Custom Simulation

Instead of running `surveysim`, you can incorporate and customize the following top-level simulation driver directly into your own script or jupyter notebook:

```
import datetime

import desisurvey.config
import desisurvey.rules
import desisurvey.plan
import desisurvey.scheduler

import surveysim.weather
import surveysim.stats
import surveysim.exposures
import surveysim.nightops

def simulate_survey(rules, weather, use_twilight=False):

    # Simulate the nominal survey dates.
    config = desisurvey.config.Configuration()
    start, stop = config.first_day(), config.last_day()
    num_nights = (stop - start).days

    # Initialize simulation progress tracking.
    stats = surveysim.stats.SurveyStatistics()
    explist = surveysim.exposures.ExposureList()

    # Initialize afternoon planning.
    planner = desisurvey.plan.Planner(rules, simulate=True)

    # Initialize next tile selection.
    scheduler = desisurvey.scheduler.Scheduler()

    # Loop over nights.
    num_simulated = 0
    for num_simulated in range(num_nights):
        night = start + datetime.timedelta(num_simulated)
```

(continues on next page)

(continued from previous page)

```
        # Perform afternoon planning.
        explist.update_tiles(night, *planner.afternoon_plan(night))

        if not desisurvey.utils.is_monsoon(night) and not scheduler.ephem.is_full_
↪moon(night):
            # Simulate one night of observing.
            surveysim.nightops.simulate_night(
                night, scheduler, stats, explist, weather=weather, use_twilight=use_
↪twilight)
            planner.set_donefrac(scheduler.tiles.tileID, scheduler.snr2frac)

            if scheduler.plan.survey_completed():
                break

    return stats, explist
```

To run a simulation, define the survey strategy rules, e.g.:

```
rules = desisurvey.rules.Rules('rules-depth.yaml')
```

and the random weather realization to use, e.g.:

```
weather = surveysim.weather.Weather(seed=1, replay='random')
```

then call the function defined above:

```
stats, exposures = simulate_survey(rules, weather)
```

1.2.9 Visualization

The `surveymovie` script reads simulation outputs and generates a movie with one frame per exposure to visualize the scheduler algorithm and survey progress:

```
surveymovie --verbose
```

An example is available. A key describing the information displayed in each frame is here. To generate a PNG of a single frame, use:

```
surveymovie --expid 123 --save exposure123
```

to create `exposure123.png`.

To generate a smaller summary movie with one frame per night, use the *–nightly* option, e.g.:

```
surveymovie --nightly --save summary
```

The `surveymovie` script uses the external `ffmpeg` program to generate movies, so this must be installed. At NERSC, use:

```
module add ffmpeg
```

1.3 Full surveysim API Reference

1.3.1 Modules

surveysim.nightops

Simulate one night of observing.

surveysim.nightops.**simulate_night**(*night*, *scheduler*, *stats*, *explist*, *weather*, *use_twilight=False*, *update_interval=10.0*, *plot=False*, *verbose=False*)

Simulate one night of observing.

Uses the online tile scheduler and exposure time calculator.

Parameters

- **night** (*datetime.date*) – Date that the simulated night starts.
- **scheduler** (desisurvey.scheduler.Scheduler) – Next tile scheduler to use.
- **stats** (*surveysim.stats.SurveyStatistics*) – Object for accumulating simulated survey statistics.
- **explist** (*surveysim.exposures.ExposureList*) – Object for recording simulated exposures.
- **weather** (*surveysim.weather.Weather*) – Simulated weather conditions to use.
- **use_twlight** (*bool*) – Observe during twilight when True.
- **update_interval** (*float*) – Interval in seconds for simulated ETC updates.
- **plot** (*bool*) – Generate a plot summarizing the simulated night when True.
- **verbose** (*bool*) – Produce verbose output on simulation progress when True.

surveysim.simulator

Top-level survey simulation manager.

class surveysim.simulator.**Simulator**(*start_date*, *stop_date*, *progress*, *weather*, *stats*, *strategy*, *plan*, *gen=None*)

Initialize a survey simulation.

Parameters

- **start_date** (*datetime.date*) – Survey starts on the evening of this date.
- **stop_date** (*datetime.date*) – Survey stops on the morning of this date.
- **progress** (*desisurvey.progress.Progress*) – Progress of survey at the start of this simulation.
- **weather** (surveysim.weather.Weather) – Simulated weather conditions use use.
- **stats** (*astropy.table.Table or None*) – Table of per-night efficiency statistics to update.
- **strategy** (*str*) – Strategy to use for scheduling tiles during each night.
- **plan** (*str*) – Name of plan file to use.

- **gen** (*numpy.random.RandomState or None*) – Random number generator to use for reproducible samples. Will be initialized (un-reproducibly) if None.

date
The current simulation date as a datetime object.

next_day(*scores=None*)
Simulate the next day of survey operations.

A day runs from local noon to local noon. A survey ends, with this method returning False, when either we reach the last scheduled day or else we run out of tiles to observe.

Returns True if there are more days to simulate.

Return type bool

surveysim.weather

Simulate stochastic observing weather conditions.

The simulated conditions include seeing, transparency and the dome-open fraction.

class surveysim.weather.**Weather**(*seed=1*, *replay='random'*, *time_step=5*, *restore=None*, *extra_downtime=0*)
Simulate weather conditions affecting observations.

The start/stop date range is taken from the survey config.

Seeing and transparency values are stored with 32-bit floats to save some memory.

Parameters

- **seed** (*int*) – Random number seed to use to generate stochastic conditions. The seed determines the same seeing and transparency realization independent of the value of `replay`.
- **replay** (*str*) – Either 'random' or a comma-separated list of years whose historical weather should be replayed, e.g. 'Y2010,Y2012'. Replayed weather will be used cyclically if necessary. Random weather will be a boostrap sampling of all available years with historical weather data. Use 'Y2015' for the worst-case weather scenario.
- **time_step** (float or `astropy.units.Quantity`, optional) – Time step calculating updates. Must evenly divide 24 hours. If unitless float, will be interpreted as minutes.
- **restore** (*filename or None*) – Restore an existing weather simulation from the specified file name. All other parameters are ignored when this is provided. A relative path name refers to the `configuration output path`.
- **extra_downtime** (*float*) – Additionally close the dome completely on some nights. Nights are chosen randomly, with the chance of the night being closed equal to extra_random_close_fraction. This is intended to include margin.

get(*time*)
Get the weather conditions at the specified time(s).

Returns the conditions at the closest tabulated time, rather than using interpolation.

Parameters **time** (*astropy.time.Time*) – Time(s) when the simulated weather is requested.

Returns Slice of precomputed table containing row(s) corresponding to the requested time(s).

Return type table slice

save (*filename*, *overwrite=True*)
Save the generated weather to a file.

The saved file can be restored using the constructor *restore* parameter.

Parameters

- **filename** (*str*) – Name of the file where the weather should be saved. A relative path name refers to the `configuration output path`.
- **overwrite** (*bool*) – Silently overwrite any existing file when this is True.

surveysim.util

Simulation utilities that may be used by other packages.

`surveysim.util.`**add_calibration_exposures** (*exposures*, *flats_per_night=3*, *arcs_per_night=3*, *darks_per_night=0*, *zeroes_per_night=0*, *exptime=None*, *readout=30.0*)
Prepare a list of science exposures for desisim.wrap-newexp.

Insert calibration exposures at the start of each night, and add the following columns for all exposures: EXPID, PROGRAM, NIGHT, FLAVOR.

Parameters

- **exposures** (table like or *surveysim.exposures.ExposureList*) – A table of science exposures including, at a minimum, MJD, EXPTIME and TILEID columns. The exposures must be sorted by increasing MJD. Could be a numpy recarray, an astropy table, or an ExposureList object. Columns other than the required ones are copied to the output.
- **flats_per_night** (`int`, optional) – Add this many arc exposures per night (default 3).
- **arcs_per_night** (`int`, optional) – Add this many arc exposures per night (default 3).
- **darks_per_night** (`int`, optional) – Add this many dark exposures per night (default 0).
- **zeroes_per_night** (`int`, optional) – Add this many zero exposures per night (default 0).
- **exptime** (`dict`, optional) – A dictionary setting calibration exposure times for each calibration flavor.
- **readout** (`float`, optional) – Set readout time for calibration exposures (default 30.0 s).

Returns The output table augmented with calibration exposures and additional columns.

Return type `astropy.table.Table`

Raises `ValueError` – If the input is not sorted by increasing MJD/timestamp.

surveysim.stats

Record simulated nightly statistics by program.

class `surveysim.stats.`**SurveyStatistics** (*start_date=None*, *stop_date=None*, *restore=None*)
Collect nightly statistics by program.

Parameters

- **start_date** (*datetime.date or None*) – Record statistics for a survey that starts on the evening of this date. Uses the configured nominal start date when None.

- **stop_date** (*datetime.date*) – Record statistics for a survey that stops on the morning of this date. Uses the configured nominal stop date when None.
- **restore** (*str or None*) – Restore internal state from the snapshot saved to this filename, or initialize a new object 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.

plot (*forecast=None*)
Plot a summary of the survey statistics.

Requires that matplotlib is installed.

save (*name='stats.fits'*, *comment=''*, *overwrite=True*)
Save a snapshot of these statistics as a binary FITS table.

The saved file size is ~800 Kb.

Parameters

- **name** (*str*) – File name to write. Will be located in the configuration output path unless it is an absolute path. Pass the same name to the constructor's `restore` argument to restore this snapshot.
- **comment** (*str*) – Comment to include in the saved header, for documentation purposes.
- **overwrite** (*bool*) – Silently overwrite any existing file when True.

summarize (*nthday=None*)
Print a tabular summary of the accumulated statistics to stdout.

surveysim.exposures

Record simulated exposures and collect per-tile statistics.

class surveysim.exposures.**ExposureList** (*restore=None*, *max_nexp=60000*, *existing_exposures=None*)
Record simulated exposures and collect per-tile statistics.

Parameters

- **restore** (*str or None*) – Restore internal state from the snapshot saved to this filename, or initialize a new object 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.
- **max_nexp** (*int*) – The maximum expected number of exposures, which determines the memory size of this object.

add (*mjd*, *exptime*, *tileID*, *snr2frac*, *dsnr2frac*, *airmass*, *seeing*, *transp*, *sky*)
Record metadata for a single exposure.

Parameters

- **mjd** (*float*) – MJD timestamp at the start of the exposure.
- **tileID** (*int*) – ID of the observed tile.
- **snr2frac** (*float*) – Fractional SNR2 accumulated on this tile at the end of exposure.
- **dsnr2frac** (*float*) – Fractional SNR2 accumulated on this tile during this exposure.
- **airmass** (*float*) – Average airmass during this exposure.
- **seeing** (*float*) – Average atmospheric seeing in arcseconds during this exposure.
- **transp** (*float*) – Average atmospheric transparency during this exposure.

- **sky** (*float*) – Average sky background level during this exposure.

save (*name='exposures.fits'*, *comment=''*, *overwrite=True*)
Save exposures to a FITS file with two binary tables.

The saved file size scales linearly with the number of exposures added so far, and is independent of the memory size of this object.

Parameters

- **name** (*str*) – File name to write. Will be located in the configuration output path unless it is an absolute path. Pass the same name to the constructor's `restore` argument to restore this snapshot.
- **comment** (*str*) – Comment to include in the saved header, for documentation purposes.
- **overwrite** (*bool*) – Silently overwrite any existing file when True.

update_tiles (*night*, *available*, *planned*)
Update tile availability and planning status.

Parameters

- **night** (*datetime.date*) – Night of initial observing.
- **available** (*array*) – Array of tile indices that are newly available since the last call to this method.
- **planned** (*array*) – Array of tile indices that are newly planned (priority > 0) since the last call to this method.

1.3.2 Command-Line Scripts

surveysim

Script wrapper for running survey simulations.

Simulate a sequence of observations until either the nominal survey end date is reached, or all tiles have been observed. See doc/tutorial.rst for details.

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

surveysim.scripts.surveysim.**main** (*args*)
Command-line driver for running survey simulations.

surveysim.scripts.surveysim.**parse** (*options=None*)
Parse command-line options for running survey simulations.

1.4 surveysim Data Model

1.4.1 Warning

This data model needs to be incorporated into the main DESI data model!

1.4.2 Contents

obslist_all

Summary List of tiles observed, along with tile info and observing conditions.

Naming Convention `obslist_all.fits`

Regex `obslist\_all\.fits`

File Type FITS, 1 MB

Contents

Number	EXTNAME	Type	Contents
HDU0		IMAGE	N/A
HDU1		BINTABLE	Observed tiles

FITS Header Units

HDU0

N/A.

This HDU has no non-standard required keywords.

Empty HDU.

HDU1

Table of observed tiles.

Required Header Keywords

KEY	Example Value	Type	Comment
NAXIS1	152	int	length of dimension 1
NAXIS2	10092	int	length of dimension 2

Required Data Table Columns

Name	Type	Units	Description
TILEID	int32		DESI tile ID
RA	float64	deg	right ascension
DEC	float64	deg	declination
PROGRAM	char[8]		'DARK', 'BRIGHT' or 'GRAY'
EBMV	float64		Extinction
MAXLEN	float64	sec	Maximum exposure length
MOONFRAC	float64		Fraction of the Moon's illuminated surface
MOONDIST	float64	deg	Distance between the Moon and the tile centre.
MOONALT	float64	deg	Elevation of the Moon
SEEING	float64	"	Seeing
LINTRANS	float64		Linear transparency
AIRMASS	float64		Airmass
DESSN2	float64		Design (S/N)^2
STATUS	int32		2 (fully observed)
EXPTIME	float64	sec	Exposure time
OBSSN2	float64		Achieved (S/N)^2
DATE-OBS	char[24]		Date and time of observation
MJD	float64	days	Modified Julian Date of observation

Notes and Examples

Tiles with status==0 (unobserved) or status==1 (partially observed) should not be in this list.

obslistYYYYMMDD

Summary List of tiles observed, along with tile info and observing conditions.

Naming Convention `obslistYYYYMMDD.fits`, where YYYYMMDD is the date of the start of the night of observations.

Regex `obslist20([0-9]{2})(0[1-9]|1[0-2])(0[1-9]|1[0-9]|2[0-9]|3[0-1])\.fits`

File Type FITS, 10-15 KB.

Contents

Number	EXTNAME	Type	Contents
HDU0		IMAGE	N/A
HDU1		BINTABLE	Observed tiles

FITS Header Units

HDU0

N/A.

This HDU has no non-standard required keywords.

Empty HDU.

HDU1

Table of observed tiles.

Required Header Keywords

KEY	Example Value	Type	Comment
NAXIS1	152	int	length of dimension 1
NAXIS2	30	int	length of dimension 2

Required Data Table Columns

Name	Type	Units	Description
TILEID	int32		DESI tile ID
RA	float64	deg	right ascension
DEC	float64	deg	declination
PROGRAM	char[8]		'DARK', 'BRIGHT' or 'GRAY'
EBMV	float64		Extinction
MAXLEN	float64	sec	Maximum exposure length
MOONFRAC	float64		Fraction of the Moon's illuminated surface
MOONDIST	float64	deg	Distance between the Moon and the tile centre.
MOONALT	float64	deg	Elevation of the Moon
SEEING	float64	"	Seeing
LINTRANS	float64		Linear transparency
AIRMASS	float64		Airmass
DESSN2	float64		Design (S/N)^2
STATUS	int32		1 or 2 if partially or fully observed
EXPTIME	float64	sec	Exposure time
OBSSN2	float64		Achieved (S/N)^2
DATE-OBS	char[24]		Date and time of observation
MJD	float64	days	Modified Julian Date of observation

Notes and Examples

The status can also be 0, but tiles with status==0 should not be in this table.

obsplanYYYYMMDD

Summary Contains the afternoon plan.

Naming Convention `obsplanYYYYMMDD.fits`, where YYYYMMDD is the date of the start of the night of observations.

Regex `obsplan20([0-9]{2})(0[1-9]|1[0-2])(0[1-9]|1[0-9]|2[0-9]|3[0-1])\.fits`

File Type FITS, 8-560 KB

Contents

Number	EXTNAME	Type	Contents
HDU0		IMAGE	N/A
HDU1		BINTABLE	List of tiles in plan

FITS Header Units

HDU0

N/A.

Required Header Keywords

KEY	Example Value	Type	Comment
MOONFRAC	0.144870860633956	float	Moon illumination fraction

Empty HDU.

HDU1

List of tiles to observe in order of priority.

Required Header Keywords

KEY	Example Value	Type	Comment
NAXIS1	56	int	length of dimension 1
NAXIS2	9607	int	length of dimension 2

Required Data Table Columns

Name	Type	Units	Description
TILEID	int32		DESI tile ID
RA	float64	deg	Right ascencion
DEC	float64	deg	Declination
EBV_MED	float64		Extinction
LSTMIN	float32	deg	Start of LST observing window
LSTMAX	float32	deg	End of LST observing window
MAXEXPLEN	float32	sec	Maximum exposure lenght
PRIORITY	int32		Between 0(high priority) - 10(low priority)
STATUS	int32		0 (unobserved) or 1 (partially observed)
PROGRAM	char[6]		'DARK', 'BRIGHT' or 'GRAY'
OBSCONDITIONS	int16		0 (DARK), 1 (GRAY), 2 (BRIGHT) or 3 (POOR)

Notes and Examples

Status==2 tiles (observed) should not be in this list.

tiles_observed

Summary Lists the tile ID of every tile observed so far.

Naming Convention `tiles_observed.fits`

Regex `tiles\_observed\.fits`

File Type FITS, 8 KB

Contents

Number	EXTNAME	Type	Contents
HDU0		IMAGE	N/A
HDU1		BINTABLE	List of observed tiles

FITS Header Units

HDU0

N/A.

This HDU has no non-standard required keywords.

Empty HDU.

HDU1

List of observed tiles.

Required Header Keywords

KEY	Example Value	Type	Comment
NAXIS1	12	int	length of dimension 1
NAXIS2	141	int	length of dimension 2
MJDBEGIN	57749.79166666666	float	Start day of the survey

Required Data Table Columns

Name	Type	Units	Description
TILEID	int64		DESI tile ID
STATUS	int32		1 (partially observed) or 2 (completed)

Notes and Examples

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index