desiutil Documentation

Release 3.2.5

DESI

Nov 15, 2022

Contents

CHAPTER 1

Introduction

desiutil is a set of low-level utilities used by all DESI packages.

CHAPTER 2

Dependencies

2.1 Required Dependencies

These packages must be installed for desiutil to work properly:

- pyyaml
- requests
- astropy
  - This implies a dependency on NumPy

2.2 Optional Dependencies

If you want to use the plotting utilities in `desiutil.plots`, you will need:

- matplotlib
- healpy

CHAPTER 3

Contents

3.1 desiutil API

3.1.1 desiutil

This package provides low-level utilities for general use by DESI.

3.1.2 desiutil.bitmask

Mask bits for the spectro pipeline.

Individual packages will define their own mask bits and use this as a utility access wrapper. Typical users will get their bitmasks pre-made from those packages, not from here.

Stephen Bailey, Lawrence Berkeley National Lab Fall 2015

Examples

desispec could create a ccdmask like this:

```
>>> from desiutil.bitmask import BitMask
>>> import yaml
>>> _bitdefs = yaml.safe_load('''
... ccdmask:
...     - [BAD,       0, "Pre-determined bad pixel (any reason)"]
...     - [HOT,       1, "Hot pixel"]
...     - [DEAD,      2, "Dead pixel"]
...     - [SATURATED, 3, "Saturated pixel from object"]
...     - [COSMIC,    4, "Cosmic ray"]
... ''')
...
>>> ccdmask = BitMask('ccdmask', _bitdefs)
```

Users would then access this mask with:

```
>>> from desispec.bitmasks import ccdmask
>>> ccdmask.COSMIC | ccdmask.SATURATED  #- 2**4 + 2**3
24
>>> ccdmask.mask('COSMIC')      # 2**4, same as ccdmask.COSMIC
16
>>> ccdmask.mask(4)             # 2**4, same as ccdmask.COSMIC
16
>>> ccdmask.COSMIC              # 2**4, same as ccdmask.mask('COSMIC')
16
>>> ccdmask.bitnum('COSMIC')
4
>>> ccdmask.bitname(4)
'COSMIC'
>>> ccdmask.names()
['BAD', 'HOT', 'DEAD', 'SATURATED', 'COSMIC']
>>> ccdmask.names(3)
['BAD', 'HOT']
>>> ccdmask.comment(0)
'Pre-determined bad pixel (any reason)'
>>> ccdmask.comment('COSMIC')
'Cosmic ray'
```

class desiutil.bitmask.**BitMask**(*name*, *bitdefs*)

BitMask object to represent bit names, masks, and comments.

Typical users are not expected to create BitMask objects directly; other packages like desispec and desitarget will have used this to pre-create the bitmasks for them using definition files in those packages.

Parameters

- **name** (str) – Name of this mask, must be key in *bitdefs*.
- **bitdefs** (dict) – Dictionary of different mask bit definitions; each value is a list of `[bitname, bitnum, comment]`. A 4th entry is optional, which must be a dictionary.

bitname(*bitnum*)

Return bit name (string) for this *bitnum* (integer).

Parameters **bitnum** (int) – The number of the bit.

Returns The name of the bit.

Return type str

bitnum(*bitname*)

Return bit number (int) for this *bitname* (string).

Parameters **bitname** (str) – The bit name.

Returns The bit value.

Return type int

comment(*bitname_or_num*)

Return comment for this bit name or bit number.

Parameters **bitname_or_num** (int or str) – Name of number of the mask.

Returns The comment string.

Return type str

mask(*name_or_num*)
Return mask value.

Parameters **name_or_num** (int or str) – Name of number of the mask.

Returns The value of the mask.

Return type int

Examples

```
>>> bitmask.mask(3)          #  2**3
8
>>> bitmask.mask('BLAT')
>>> bitmask.mask('BLAT|FOO')
```

names(*mask=None*)
Return list of names of masked bits.

Parameters **mask** (int, optional) – The mask integer to convert to names. If not supplied, return names of all known bits.

Returns The list of names contained in the mask.

Return type list

class desiutil.bitmask.**_MaskBit**
A single mask bit.

Subclasses int to act like an int, but allows the ability to extend with blat.name, blat.comment, blat.mask, blat.bitnum.

name
The name of the bit.

Type str

bitnum
The number of the bit. The value of the bit is 2**bitnum.

Type int

mask
The value of the bit, 2**bitnum.

Type int

comment
A comment explaining the meaning of the bit.

Type str

3.1.3 desiutil.brick

Code for calculating bricks, which are a tiling of the sky with the following properties:

- bricks form rows in dec like a brick wall; edges are constant RA or dec
- they are rectangular with longest edge shorter or equal to bricksize
- circles at the poles with diameter=bricksize

- there are an even number of bricks per row

Use this with caution! In most cases you should be propagating brick info from input targeting, not recalculating brick locations and names.

Note that this code was originally in desispec, so earlier commit information is in the desispec repository.

class desiutil.brick.**Bricks**(*bricksize=0.25*)

The Bricks object describes bricks of a certain size.

Parameters **bricksize** (float, optional) – Brick size in degrees. Default 0.25 degrees.

_array_radec(*ra*, *dec*)

Convert (*ra*, *dec*) to arrays and clean up the data.

_row_col(*ra*, *dec*)

Determine the brick row and column, given *ra*, *dec*.

brick_radec(*ra*, *dec*)

Return center (ra,dec) of brick that contains input (*ra*, *dec*) [deg]

Parameters

- **ra** (float or ndarray) – Right Ascension in degrees.
- **dec** (float or ndarray) – Declination in degrees.

Returns The centers of the bricks at the locations of interest.

Return type ndarray

brick_tan_wcs_size()

Compute required angular size needed for WCS transformation.

Returns the minimum required angular size (pixel scale x number of pixels) for a TAN WCS tiling on these brick centers, so that RA1, RA2, DEC1, DEC2 land within the tile.

Returns The angular size in degrees.

Return type float

brickarea(*ra*, *dec*)

Return the area of the brick for a given location.

Parameters

- **ra** (float or ndarray) – Right Ascension in degrees.
- **dec** (float or ndarray) – Declination in degrees.

Returns The areas of the bricks at the locations of interest.

Return type ndarray

brickid(*ra*, *dec*)

Return the BRICKID for a given location.

Parameters

- **ra** (float or ndarray) – Right Ascension in degrees.
- **dec** (float or ndarray) – Declination in degrees.

Returns The legacysurvey BRICKID at the locations of interest.

Return type ndarray

brickname(*ra*, *dec*)
Return brick name of brick covering (*ra*, *dec*).

Parameters

- **ra** (float or ndarray) – Right Ascension in degrees.
- **dec** (float or ndarray) – Declination in degrees.

Returns An array of strings containing the names.

Return type ndarray

brickq(*ra*, *dec*)
Return the BRICKQ for a given location.

Parameters

- **ra** (float or ndarray) – Right Ascension in degrees.
- **dec** (float or ndarray) – Declination in degrees.

Returns The legacysurvey BRICKQ at the locations of interest.

Return type ndarray

bricksize
Size of a brick in degrees.

brickvertices(*ra*, *dec*)
Return the vertices in RA/Dec of the brick that given locations lie in

Parameters

- **ra** (float or ndarray) – Right Ascension in degrees.
- **dec** (float or ndarray) – Declination in degrees.

Returns The 4 vertices of the bricks at the locations of interest (an array with 4 columns of (RA, Dec) and `len(ra)` rows).

Return type ndarray

Notes

The vertices are ordered counter-clockwise from the minimum (RA, Dec).

to_table()
Convert *Bricks* object into a Table.

Returns A table containing the brick data.

Return type astropy.table.Table

desiutil.brick.**brickname**(*ra*, *dec*, *bricksize=0.25*)
Return brick name of brick covering (*ra*, *dec*).

Parameters

- **ra** (float or ndarray) – Right Ascension in degrees.
- **dec** (float or ndarray) – Declination in degrees.
- **bricksize** (float, optional) – Brick size in degrees. Default 0.25 degrees.

Returns An array of strings containing the names.

Return type ndarray

Notes

This function is a convenience wrapper on *desiutil.brick.Bricks.brickname()*. It will cache the brick computation to speed up repeated calls.

3.1.4 desiutil.census

Determine the number of files and size in DESI data file systems.

Notes

- Directories to check:
  - Imaging raw & reduced.
  - spectro raw & reduced.
  - Work directories.
  - Non-Footprint image data.
- Check group id, readability.
- Count number of files and size.
- Extract year from mtime. Shift to fiscal year. FY starts in October.
- Don't record filenames, just high-level directories.
- Treat projecta as same system, follow symlinks to projecta
- If a symlink is followed to another filesystem, `os.walk()` can't get back to the original filesystem.
- Symlinks to another subdirectory should only count as the symlink. The file itself belongs to the other subdirectory.
- Physical directories count toward inode and size total.

class desiutil.census.**ScannedFile**(*filename*, *size*, *year*)
Simple object to store results of a file scan.

filename
Name of the file.

Type str

size
Size in bytes of the file.

Type int

year
Year the file was modified.

Type int

islink
Is the file a symbolic link?

Type bool

isexternal
If the file is a symbolic link, does it link outside the tree being scanned?

Type bool

linkname
If the file is a symbolic link, it points to this file.

Type str

linksize
If the file is a symbolic link, this is the size of the link *itself*, and the size attribute is the size of the file it *points to*.

Type int

linkyear
Year the link *itself* was modified.

Type int

desiutil.census.**get_options**(*test_args=None*)
Parse command-line options.

Parameters **test_args** (list) – Override command-line arguments for testing purposes.

Returns A simple object containing the parsed options.

Return type argparse.Namespace

desiutil.census.**in_path**(*root*, *path*)
Check if *path* is in the same directory hierarchy as *root*.

Parameters

- **root** (str) – Root directory.
- **path** (str) – Filename, could be a file or a directory.

Returns True if *path* is in *root*.

Return type bool

desiutil.census.**main**()
Entry point for the **desi_data_census** script.

Returns Exit status that will be passed to sys.exit().

Return type int

desiutil.census.**output_csv**(*summary*, *filename*)
Convert data into CSV file.

Parameters

- **summary** (list) – A data structure.
- **filename** (str) – Name of the file to write to.

Returns The data written to the CSV file, as a list of rows.

Return type list

desiutil.census.**scan_directories**(*conf*, *data*)
Scan the directories specified by the configuration file.

Parameters

- **conf** (dict) – The configuration that applies to all directories.
- **data** (list) – The specific directories to scan.

Returns A list containing data structures summarizing data found.

Return type list

desiutil.census.**scan_directory**(*dirpath*, *dirnames*, *filenames*, *gid*)
Count number and size of files in a single directory hierarchy.

Parameters

- **dirpath** (str) – Current directory, returned by os.walk().
- **dirnames** (list) – List of directories in *dirpath*.
- **filenames** (list) – List of files in *dirpath*.
- **gid** (int) – Group ID number that should be associated with this directory.

Returns A tuple containing two dictionaries: the summary results organized by year, and a summary of links to external directories.

Return type tuple()

desiutil.census.**scan_file**(*dirpath*, *filename*, *gid*)
Analyze a single file or directory.

Parameters

- **dirpath** (str) – Current directory, returned by os.walk().
- **filename** (str) – Base name of current file.
- **gid** (int) – Group ID number that should be associated with this directory.

Returns A simple object containing the metadata relating to the file.

Return type *ScannedFile*

desiutil.census.**walk_error**(*e*)
Handle errors reported by os.walk().

Parameters **e** (OSError) – The exception reported.

desiutil.census.**year**(*mtime*, *fy=True*)
Convert a file's modification time into a year.

Parameters

- **mtime** (int or float) – File modification time as reported by os.stat().
- **fy** (bool, optional) – If True use Fiscal Year (FY) instead of calendar year. FY is defined to begin 1 October.

Returns The year to which a file belongs.

Return type int

3.1.5 desiutil.depend

Utilities for working with code dependencies stored in FITS headers.

The code name and the code version are stored in pairs of keywords, similar to how table columns are defined. *e.g.*:

```
DEPNAM00 = 'numpy'
DEPVER00 = '1.11'
DEPNAM01 = 'desiutil'
DEPVER01 = '1.4.1'
```

The functions and Dependencies class provide convenience wrappers to loop over they keywords looking for a particular dependency and adding a new dependency version to next available DEPNAMnn/DEPVERnn.

Examples:

```
>>> import desiutil
>>> from desiutil import depend
>>> import astropy
>>> from astropy.io import fits
>>>
>>> hdr = fits.Header()
>>> depend.setdep(hdr, 'desiutil', desiutil.__version__)
>>> depend.setdep(hdr, 'astropy', astropy.__version__)
>>> depend.getdep(hdr, 'desiutil')
'1.4.1.dev316'
>>> depend.hasdep(hdr, 'astropy')
True
>>> hdr
DEPNAM00= 'desiutil'
DEPVER00= '1.4.1.dev316'
DEPNAM01= 'astropy '
DEPVER01= '1.1.1   '
```

There is also an object wrapper that gives a header dict-like semantics to update or view dependencies. This directly updates the input header object so that it can be used in subsequent I/O

```
>>> codever = depend.Dependencies(hdr)
>>> codever['blat'] = '1.2'
>>> codever['foo'] = '3.4'
>>> for codename, version in codever.items():
...     print(codename, version)
...
('desiutil', '1.4.1.dev316')
('astropy', u'1.1.1')
('blat', '1.2')
('foo', '3.4')
```

class desiutil.depend.**Dependencies**(*header=None*)

Dictionary-like object to track dependencies.

Parameters **header** (*dict-like, optional*) – A dict-like object. If not provided, a OrderedDict will be used.

items()

Returns iterator over (name, version).

desiutil.depend.**add_dependencies**(*header*, *module_names=None*, *long_python=False*, *envvar_names=None*)

Adds DEPNAMnn, DEPVERnn keywords to header for imported modules.

Parameters

- **header** (*dict-like*) – A dict-like object, *e.g.* astropy.io.fits.Header.

- **module_names** (list, optional) – List of of module names to check; if None, checks desiutil.depend.possible_dependencies.
- **long_python** (bool, optional) – If True use the full, verbose sys.version string for the Python version. Otherwise, use a short version, *e.g.*, 3.5.2.
- **envvar_names** (list, optional) – List of of environment variables to check; if None, checks desiutil.depend.possible_envvars.

Notes

Only adds the dependency keywords if the module has already been previously loaded in this python session. Uses module.__version__ if available, otherwise unknown (/path/to/module/).

desiutil.depend.**getdep**(*header*, *name*)
Get dependency version for code *name*.

Parameters

- **header** (*dict-like*) – A dict-like object, *e.g.* astropy.io.fits.Header.
- **name** (str) – Code name string.

Returns The version string for *name*.

Return type str

Raises KeyError – If *name* not tracked in *header*.

desiutil.depend.**hasdep**(*header*, *name*)
Check if *name* is defined in *header*.

Parameters

- **header** (*dict-like*) – A dict-like object, *e.g.* astropy.io.fits.Header.
- **name** (str) – Code name string.

Returns True if version for *name* is tracked in *header*, otherwise False.

Return type bool

desiutil.depend.**iterdep**(*header*)
Returns iterator over (codename, version) tuples.

Parameters **header** (*dict-like*) – A dict-like object, *e.g.* astropy.io.fits.Header.

desiutil.depend.**mergedep**(*srchdr*, *dsthdr*, *conflict='src'*)
Merge dependencies from srchdr into dsthdr

Parameters

- **srchdr** (*dict-like*) – source dict-like object, *e.g.* astropy.io.fits.Header, with dependency keywords DEPNAMnn, DEPVERnn
- **dsthdr** (*dict-like*) – destination dict-like object
- **conflict** (*str, optional*) – 'src' or 'dst' or 'exception'; see notes

Notes

Dependencies in srchdr are added to dsthdr, modifying it in-place, adjusting DEPNAMnn/DEPVERnn numbering as needed. If the same dependency appears in both headers with different versions, conflict controls the behavior:

- if 'src', the srchdr value replaces the dsthdr value
- if 'dst', the dsthdr value is retained unchanged
- if 'exception', raise a ValueError exception

Raises `ValueError` – If `conflict == 'exception'` and the same dependency name appears in both headers with different values; or if *conflict* isn't one of 'src', 'dst', or 'exception'.

`desiutil.depend.`**`setdep`**(*header*, *name*, *version*)
Set dependency *version* for code *name*.

Parameters

- **`header`** (*dict-like*) – A dict-like object, *e.g.* `astropy.io.fits.Header`.
- **`name`** (`str`) – Code name string.
- **`version`** (`str`) – Code version string.

Raises `IndexError` – If there are more than 100 dependencies to track.

3.1.6 desiutil.dust

Get $E(B-V)$ values from the Schlegel, Finkbeiner & Davis (1998; SFD98) dust map.

`class` `desiutil.dust.`**`SFDMap`**(*mapdir=None*, *north='SFD_dust_4096_ngp.fits'*, *south='SFD_dust_4096_sgp.fits'*, *scaling=1.0*)
Map of E(B-V) from Schlegel, Finkbeiner and Davis (1998).

Use this class for repeated retrieval of E(B-V) values when there is no way to retrieve all the values at the same time: It keeps a reference to the FITS data from the maps so that each FITS image is read only once.

Parameters

- **`mapdir`** (`str`, optional, defaults to `DUST_DIR`+`/maps`.) – Directory in which to find dust map FITS images, named `SFD_dust_4096_ngp.fits` and `SFD_dust_4096_sgp.fits`. If not specified, the map directory is derived from the value of the `DUST_DIR` environment variable, otherwise an empty string is used.
- **`south`** (*north,*) – Names of north and south galactic pole FITS files. Defaults are `SFD_dust_4096_ngp.fits` and `SFD_dust_4096_sgp.fits` respectively.
- **`scaling`** (`float`, optional, defaults to 1) – Scale all E(B-V) map values by this multiplicative factor. Pass scaling=0.86 for the recalibration from Schlafly & Finkbeiner (2011).

Notes

Modified from https://github.com/kbarbary/sfdmap/

`ebv`(**args*, ***kwargs*)
Get E(B-V) value(s) at given coordinate(s).

Parameters

- **`coordinates`** (`SkyCoord` or `ndarray`) – If one argument is passed, assumed to be an `SkyCoord` instance, in which case the `frame` and `unit` keyword arguments are ignored. If two arguments are passed, they are treated as `latitute, longitude` (can be scalars or arrays or a tuple), in which case the frame and unit are taken from the passed keywords.

- **frame** (str, optional, defaults to 'icrs') – Coordinate frame, if two arguments are passed. Allowed values are any SkyCoord frame, and 'fk5j2000' and 'j2000'.
- **unit** (str, optional, defaults to 'degree') – Any SkyCoord unit.
- **interpolate** (bool, optional, defaults to True) – Interpolate between the map values using bilinear interpolation.

Returns Specific extinction E(B-V) at the given locations.

Return type ndarray

Notes

Modified from https://github.com/kbarbary/sfdmap/

class desiutil.dust.**_Hemisphere**(*fname*, *scaling*)

Represents one of the hemispheres (in a single file).

Parameters

- **fname** (str) – File name containing one hemisphere of the dust map.
- **scaling** (float) – Multiplicative factor by which to scale the dust map.

data

Pixelated array of dust map values.

Type ndarray

crpix1, crpix2

World Coordinate System: Represent the 1-indexed X and Y pixel numbers of the poles.

Type float

lam_scal

Number of pixels from b=0 to b=90 deg.

Type int

lam_nsgp

+1 for the northern hemisphere, -1 for the south.

Type int

Notes

Taken in full from https://github.com/kbarbary/sfdmap/

ebv(*l*, *b*, *interpolate*)

Project Galactic longitude/latitude to lambert pixels (See SFD98).

Parameters

- **b** (*l,*) – Galactic longitude and latitude.
- **interpolate** (bool) – If True use bilinear interpolation to obtain values.

Returns Reddening values.

Return type ndarray

desiutil.dust.**_bilinear_interpolate**(*data*, *y*, *x*)

Map a two-dimensional integer pixel-array at float coordinates.

Parameters

- **data** (ndarray) – Pixelized array of values.
- **y** (float or ndarray) – y coordinates (each integer y is a row) of location in pixel-space at which to interpolate.
- **x** (float or ndarray) – x coordinates (each integer x is a column) of location in pixel-space at which to interpolate.

Returns Interpolated data values at the passed locations.

Return type float or ndarray

Notes

Taken in full from https://github.com/kbarbary/sfdmap/

desiutil.dust.**_get_ext_coeff**(*temp*, *photsys*, *band*, *ebv_sfd*, *rv=3.1*)
Obtain extinction coeffiecient A_X/E(B-V)_SFD for black body spectrum with a given temperature observed with photsys and band and extinction ebv_sfd

Parameters

- **temp** (*float*) – temperature in Kelvin
- **photsys** (*str*) – N, S, or G (gaia)
- **band** (*str*) – g,r,z (if photsys=N or S); G, BP, or RP if photsys=G
- **ebv_sfd** (*float*) – E(B-V) from SFD dust map
- **Rv** (*float*) – Value of extinction law R_V; default is 3.1

Returns extinction coefficient A_X / E(B-V)_SFD

Note: this is currently a _hidden function due to its speclite dependency, but it could be promoted if needed by external libraries.

desiutil.dust.**dust_transmission**(*wave*, *ebv_sfd*, *Rv=3.1*)
Return the dust transmission [0-1] vs. wavelength.

Parameters

- **wave** – 1D array of vacuum wavelength [Angstroms]
- **ebv_sfd** – E(B-V) as derived from the map of Schlegel, Finkbeiner and Davis (1998)
- **Rv** – total-to-selective extinction ratio, defaults to 3.1

Returns 1D array of dust transmission (between 0 and 1)

The routine does internally multiply ebv_sfd by dust.ebv_sfd_scaling. The Fitzpatrick dust extinction law is used, given R_V (default 3.1).

Also see *mwdust_transmission* which return transmission within a filter

desiutil.dust.**ebv**(**args*, ***kwargs*)
Convenience function, equivalent to SFDMap().ebv(*args).

desiutil.dust.**ext_ccm**(*wave*, *Rv=3.1*)
Return extinction curve from CCM (1989), defined in the wavelength range [1250,33333] Angstroms.

Parameters

- **wave** – 1D array of vacuum wavelength [Angstroms]

- **Rv** – Value of R_V (scalar); default is 3.1

Returns 1D array of A(lambda)/A(V)

desiutil.dust.**ext_fitzpatrick**(*wave*, *R_V=3.1*, *avglmc=False*, *lmc2=False*, *x0=None*, *gamma=None*, *c1=None*, *c2=None*, *c3=None*, *c4=None*)

Return extinction curve from Fitzpatrick (1999).

Parameters **wave** – 1D array of vacuum wavelength [Angstroms]

Returns 1D array of A(lambda)/A(V)

OPTIONAL INPUT KEYWORDS

R_V - scalar specifying the ratio of total to selective extinction R(V) = A(V) / E(B - V). If not specified, then R = 3.1 Extreme values of R(V) range from 2.3 to 5.3

avglmc - if set, then the default fit parameters c1,c2,c3,c4,gamma,x0 are set to the average values determined for reddening in the general Large Magellanic Cloud (LMC) field by Misselt et al. (1999, ApJ, 515, 128)

lmc2 - if set, then the fit parameters are set to the values determined for the LMC2 field (including 30 Dor) by Misselt et al. Note that neither /AVGLMC or /LMC2 will alter the default value of R_V which is poorly known for the LMC.

The following five input keyword parameters allow the user to customize the adopted extinction curve. For example, see Clayton et al. (2003, ApJ, 588, 871) for examples of these parameters in different interstellar environments.

x0 - Centroid of 2200 A bump in microns (default = 4.596)

gamma - Width of 2200 A bump in microns (default = 0.99)

c3 - Strength of the 2200 A bump (default = 3.23)

c4 - FUV curvature (default = 0.41)

c2 - Slope of the linear UV extinction component (default = -0.824 + 4.717 / R)

c1 - Intercept of the linear UV extinction component (default = 2.030 - 3.007 * c2)

Notes

(1) The following comparisons between the FM curve and that of Cardelli, Clayton, & Mathis (1989), (see ccm_unred.pro):

(a) • In the UV, the FM and CCM curves are similar for R < 4.0, but diverge for larger R

(b) • In the optical region, the FM more closely matches the monochromatic extinction, especially near the R band.

(2) **Many sightlines with peculiar ultraviolet interstellar extinction** can be represented with the FM curve, if the proper value of R(V) is supplied.

REQUIRED MODULES: scipy, numpy

REVISION HISTORY: Written W. Landsman Raytheon STX October, 1998 Based on FMRCurve by E. Fitzpatrick (Villanova) Added /LMC2 and /AVGLMC keywords, W. Landsman August 2000 Added ExtCurve keyword, J. Wm. Parker August 2000 Assume since V5.4 use COMPLEMENT to WHERE W. Landsman April 2006 Ported to Python, C. Theissen August 2012

desiutil.dust.**ext_odonnell**(*wave*, *Rv=3.1*)
Return extinction curve from Odonnell (1994), defined in the wavelength range [3030,9091] Angstroms. Outside this range, use CCM (1989).

Parameters

- **wave** – 1D array of vacuum wavelength [Angstroms]
- **Rv** – Value of R_V (scalar); default is 3.1

Returns 1D array of A(lambda)/A(V)

desiutil.dust.**extinction_total_to_selective_ratio**(*band*, *photsys*, *match_legacy_surveys=False*)
Return the linear coefficient R_X = A(X)/E(B-V) where A(X) = -2.5*log10(transmission in X band), for band X in 'G','R' or 'Z' when photsys = 'N' or 'S' specifies the survey (BASS+MZLS or DECALS), or for band X in 'G', 'BP', 'RP' when photsys = 'G' (when gaia dr2) or for band X in 'W1', 'W2', 'W3', 'W4' when photsys is either 'N' or 'S' E(B-V) is interpreted as SFD.

Parameters

- **band** – 'G', 'R', 'Z', 'BP', 'RP', 'W1', 'W2', 'W3', or 'W4'
- **photsys** – 'N' or 'S'

Returns scalar, total extinction A(band) = -2.5*log10(transmission(band))

desiutil.dust.**mwdust_transmission**(*ebv*, *band*, *photsys*, *match_legacy_surveys=False*)
Convert SFD E(B-V) value to dust transmission 0-1 for band and photsys

Parameters

- **ebv** (*float or array-like*) – SFD E(B-V) value(s)
- **band** (*str*) – 'G', 'R', 'Z', 'W1', 'W2', 'W3', or 'W4'
- **photsys** (*str or array of str*) – 'N' or 'S' imaging surveys photo system

Returns scalar or array (same as ebv input), Milky Way dust transmission 0-1

If *photsys* is an array, *ebv* must also be array of same length. However, *ebv* can be an array with a str *photsys*.

Also see *dust_transmission* which returns transmission vs input wavelength

3.1.7 desiutil.funcfits

Module for fitting simple functions to 1D arrays

J. Xavier Prochaska, UC Santa Cruz Fall 2015

desiutil.funcfits.**func_fit**(*x*, *y*, *func*, *deg*, *xmin=None*, *xmax=None*, *w=None*, ***kwargs*)
Simple function fit to 2 arrays.

Modified code originally from Ryan Cooke (PYPIT).

Parameters

- **x** (ndarray) – Independent data values.
- **y** (ndarray) – Dependent data to fit.
- **func** (str) – Name of the fitting function: polynomial, legendre, chebyshev.
- **deg** (int or dict) – Order of the fit.

- **xmin** (float, optional) – Minimum value in the array (or the left limit for a legendre/chebyshev polynomial).
- **xmax** (float, optional) – Maximum value in the array (or the left limit for a legendre/chebyshev polynomial).
- **w** (ndarray, optional) – Weights to be used in the fitting (weights = 1/sigma).

Returns Dictionary describing the Fit including the coefficients.

Return type dict

desiutil.funcfits.**func_val**(*x*, *fit_dict*)

Get values from a fit_dict.

Modified code originally from Ryan Cooke (PYPIT).

Parameters **x** (ndarray) – Evaluate the fit at these coordinates.

Returns Array containing the values.

Return type ndarray

desiutil.funcfits.**iter_fit**(*xarray*, *yarray*, *func*, *order*, *weights=None*, *sigma=None*, *max_rej=None*, *maxone=True*, *sig_rej=3.0*, *initialmask=None*, *forceimask=False*, *xmin=None*, *xmax=None*, *niter=999*, ***kwargs*)

A "robust" fit with iterative rejection is performed to the *xarray*, *yarray* pairs.

Modified code originally from Ryan Cooke (PYPIT).

Parameters

- **xarray** (ndarray) – Independent variable values.
- **yarray** (ndarray) – Dependent variable values.
- **func** (str) – Name of the fitting function: polynomial, legendre, chebyshev.
- **order** (int) – The order of the function to be used in the fitting.
- **sigma** (ndarray, optional) – Error in the yvalues. Used only for rejection.
- **weights** (ndarray, optional) – Weights to be used in the fitting (weights = 1/sigma).
- **maxone** (bool, optional [True]) – If True, only the most deviant point in a given iteration will be removed.
- **sig_rej** (float, optional [3.0]) – Confidence interval for rejection.
- **max_rej** (int, optional [None]) – Maximum number of points to reject.
- **initialmask** (ndarray) – A mask can be supplied as input, these values will be masked for the first iteration. 1 = value masked.
- **forceimask** (bool, optional [False]) – If True, the initialmask will be forced for all iterations.
- **niter** (int, optional [999]) – Maximum number of iterations.
- **xmin** (float) – Minimum value in the array (or the left limit for a legendre/chebyshev polynomial).
- **xmax** (float) – Maximum value in the array (or the right limit for a legendre/chebyshev polynomial).

Returns The tuple contains a dict containing the fit and a mask array containing masked values.

Return type tuple()

desiutil.funcfits.**mk_fit_dict**(*coeff*, *order*, *func*, *xmin=None*, *xmax=None*, ***kwargs*)
Generate a dict that is formatted for using func_val.

Parameters

- **coeff** (*array*) – Coefficients of the fit
- **order** (int) – The order of the function to be used in the fitting.
- **func** (str) – Name of the fitting function: polynomial, legendre, chebyshev.
- **xmin** (float) – Minimum value in the array (or the left limit for a legendre/chebyshev polynomial).
- **xmax** (float) – Maximum value in the array (or the right limit for a legendre/chebyshev polynomial).

Returns The formatted dictionary.

Return type dict

3.1.8 desiutil.git

This package contains code for interacting with DESI git products.

desiutil.git.**last_tag**(*owner*, *repo*)
Scan GitHub tags and return the most recent tag.

Parameters

- **owner** (str) – The owner or group in GitHub
- **repo** (str) – Name of the product.

Returns The most recent tag found on GitHub.

Return type str

desiutil.git.**version**(*git='git'*)
Use `git describe` to generate a version string.

Parameters **git** (str, optional) – Path to the git executable, if not in PATH.

Returns A **PEP 386**-compatible version string.

Return type str

Notes

The version string should be compatible with **PEP 386** and **PEP 440**.

3.1.9 desiutil.iers

Utilities for overriding astropy IERS functionality (astropy.utils.iers), especially for preventing unnecessary downloads of IERS data files in a high performance computing environment.

desiutil.iers.**freeze_iers**(*name='iers_frozen.ecsv'*, *ignore_warnings=True*)
Use a frozen IERS table saved with this package.

This should be called at the beginning of a script that calls astropy time and coordinates functions which refer to the UT1-UTC and polar motions tabulated by IERS. The purpose is to ensure identical results across systems and astropy releases, to avoid a potential network download, and to eliminate some astropy warnings.

After this call, the loaded table will be returned by `astropy.utils.iers.IERS_Auto.open()` and treated like a a normal IERS table by all astropy code. Specifically, this method registers an instance of a custom IERS_Frozen class that inherits from IERS_B and overrides `astropy.utils.iers.IERS._check_interpolate_indices()` to prevent any IERSRangeError being raised.

See http://docs.astropy.org/en/stable/utils/iers.html for details.

This function returns immediately after the first time it is called, so it it safe to insert anywhere that consistent IERS models are required, and subsequent calls with different args will have no effect.

The *desiutil.plots.plot_iers()* function is useful for inspecting IERS tables and how they are extrapolated to DESI survey dates.

Parameters

- **name** (str, optional) – Name of the file to load the frozen IERS table from. Should normally be relative and then refers to this package's data/ directory. Must end with the .ecsv extension.
- **ignore_warnings** (bool, optional) – Ignore ERFA and IERS warnings about future dates generated by astropy time and coordinates functions. Specifically, ERFA warnings containing the string "dubious year" are filtered out, as well as AstropyWarnings related to IERS table extrapolation.

desiutil.iers.**update_iers**(*save_name='iers_frozen.ecsv'*, *num_avg=1000*)

Update the IERS table used by astropy time, coordinates.

Downloads the current IERS-A table, replaces the last entry (which is repeated for future times) with the average of the last num_avg entries, and saves the table in ECSV format.

This should only be called every few months, *e.g.*, with major releases. The saved file should then be copied to this package's data/ directory and committed to the git repository.

Requires a network connection in order to download the current IERS-A table. Prints information about the update process.

The *desiutil.plots.plot_iers()* function is useful for inspecting IERS tables and how they are extrapolated to DESI survey dates.

Parameters

- **save_name** (str, optional) – Name where frozen IERS table should be saved. Must end with the .ecsv extension.
- **num_avg** (int, optional) – Number of rows from the end of the current table to average and use for calculating UT1-UTC offsets and polar motion at times beyond the table.

3.1.10 desiutil.install

This package contains code for installing DESI software products.

class desiutil.install.**DesiInstall**

Code and data that drive the desiInstall script.

baseproduct

The bare name of the product, *e.g.* "desiutil".

Type str

baseversion

The bare version, without any branches/ qualifiers.

Type str

default_nersc_dir_template
The default code and Modules install directory for every NERSC host.

Type str

fullproduct
The path to the product including its URL, *e.g.*, "https://github.com/desihub/desiutil".

Type str

github
True if the selected product lives on GitHub.

Type bool

is_branch
True if a branch has been selected.

Type bool

log
Logging object.

Type logging.Logger

nersc
Holds the value of NERSC_HOST, or None if not defined.

Type str

options
The parsed command-line options.

Type argparse.Namespace

product_url
The URL that will be used to download the code. This differs from *fullproduct* in that it includes the tag or branch name.

Type str

anaconda_version()
Try to determine the exact DESI+Anaconda version from the environment.

Returns The DESI+Anaconda version.

Return type str

build_type
Determine the build type.

Returns A set containing the detected build types.

Return type set

cleanup()
Clean up after the install.

Returns Returns True

Return type bool

default_nersc_dir(*nersc_host=None*)
Set the directory where code will reside.

Parameters **nersc_host** (str, optional) – Specify a NERSC host that might be different from the current host.

Returns Path to the host-specific install directory.

Return type str

get_code()
Actually download the code.

Following the standard order of execution, this is the first method that might actually modify the system (by downloading code).

Raises *DesiInstallException* – If any download errors are detected.

get_extra()
Download any additional data not included in the code repository.

This is done here so that INSTALL_DIR is defined *and* exists.

get_options(*test_args=None*)
Parse command-line arguments passed to the desiInstall script.

Parameters **test_args** (list) – Normally, this method is called without arguments, and sys.argv is parsed. Arguments should only be passed for testing purposes.

Returns A simple object containing the parsed options. Also, the attribute *options* is set.

Return type argparse.Namespace

get_product_version()
Determine the base product and version information.

Returns A tuple containing the base product name and version.

Return type tuple()

Raises *DesiInstallException* – If the product and version inputs didn't make sense.

identify_branch()
If this is not a tag install, determine the branch identity.

Returns The full path to the branch code.

Return type str

install()
Run setup.py, etc.

install_module()
Process the module file.

Returns The text of the processed module file.

Return type str

module_dependencies()
Figure out the dependencies and load them.

Returns The list of dependencies.

Return type list

nersc_module_dir
The directory that contains Module directories at NERSC.

permissions()
Fix possible install permission errors.

Returns Status code returned by fix_permissions.sh script.

Return type int

prepare_environment()
Prepare the environment for the install.

Returns The current working directory. Because we're about to change it.

Return type str

run()
This method wraps all the standard steps of the desiInstall script.

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

Return type int

sanity_check()
Sanity check the options.

Returns True if there were no problems.

Return type bool

Raises *DesiInstallException* – If any options don't make sense.

set_install_dir()
Decide on an install directory.

Returns The directory selected for installation.

Return type str

start_modules()
Set up the modules infrastructure.

Returns True if the modules infrastructure was initialized successfully.

Return type bool

verify_bootstrap()
Make sure that desiutil/desiInstall was installed with an explicit Python executable path.

For anything besides an initial bootstrap install of desiutil, this function does nothing.

Returns Returns True if everything is OK.

Return type bool

verify_url(*svn='svn'*)
Ensure that the download URL is valid.

Parameters **svn** (str, optional) – The path to the subversion command.

Returns True if everything checked out OK.

Return type bool

Raises *DesiInstallException* – If the subversion URL could not be found.

exception desiutil.install.**DesiInstallException**
The methods of *DesiInstall* should raise this exception to indicate that the command-line script should exit immediately.

desiutil.install.**dependencies**(*modulefile*)
Process the dependencies for a software product.

Parameters **modulefile** (str) – Name of the module file containing dependencies.

Returns Returns the list of dependencies. If the module file is not found or there are no dependencies, the list will be empty.

Return type list

Raises ValueError – If *modulefile* can't be found.

desiutil.install.**main**()
Entry point for the desiInstall script.

Returns Exit status that will be passed to sys.exit().

Return type int

3.1.11 desiutil.io

Module for I/O related code.

desiutil.io.**_dtype_size**(*dtype*)
Parse *dtype* to find its size.

For example, <U14 returns 14.

Parameters **dtype** (numpy.dtype) – Dtype object.

Returns The size of the type.

Return type int

Notes

This is different from dtype.itemsize, which is number of bytes.

desiutil.io.**_pick_encoding**(*table*, *encoding*)
Pick which encoding to use; giving warning if options are in conflict.

Parameters

- **table** (astropy.table.Table) – Table object.
- **encoding** (str) – Encoding to use. If None, use table.meta['ENCODING'].

Returns The chosen encoding.

Return type str

Raises UnicodeError – If no enoding could be found at all.

Notes

encoding trumps `table.meta['ENCODING'].

desiutil.io.**combine_dicts**(*dict1*, *dict2*)
Combine two dict objects into one, respecting common keys.

If *dict1* and *dict2* both have key a, then dict1[a] and dict2[a] must both be dictionaries to recursively merge.

Parameters

- **dict1** (dict) – First dictionary.
- **dict2** (dict) – Second dictionary.

Returns The combined dictionary.

Return type dict

Raises ValueError – If the values for a common key are not both dict.

desiutil.io.**decode_table**(*data*, *encoding='ascii'*, *native=True*)

Decode byte strings in a table into unicode strings.

Parameters

- **data** (numpy structured array or Table) – Data for conversion.
- **encoding** (str, optional) – Encoding to use for converting bytes into unicode; default 'ascii'; if None, try ENCODING keyword in *data* instead.
- **native** (bool, optional) – If *True* (default), only decode if native str type is unicode (*i.e.* python3 but not python2)

Returns Decoded data.

Return type Table

Notes

encoding option overides data.meta['ENCODING']; use encoding=None to use data.meta['ENCODING'] instead.

desiutil.io.**encode_table**(*data*, *encoding='ascii'*)

Encode unicode strings in a table into bytes using numpy.char.encode.

Parameters

- **data** (numpy structured array or Table) – Data for conversion.
- **encoding** (str, optional) – Encoding to use for converting unicode to bytes; default 'ascii' (FITS and HDF5 friendly); if None, try ENCODING keyword in *data* instead.

Returns Table with unicode columns converted to bytes.

Return type Table

Raises

- UnicodeEncodeError – If any input strings cannot be encoded using the specified encoding.
- UnicodeError – If no encoding is given as argument or in table metadata.

Notes

encoding option overides data.meta['ENCODING']; use encoding=None to use data.meta['ENCODING'] instead.

desiutil.io.**unlock_file**(**args*, ***kwargs*)

Unlock a read-only file, return a file-like object, and restore the read-only state when done. Arguments are the same as open().

Returns A file-like object, as returned by open().

Return type file-like

Notes

- This assumes that the user of this function is also the owner of the file. `os.chmod()` would not be expected to work in any other circumstance.
- Technically, this restores the *original* permissions of the file, it does not care what the original permissions were.
- If the named file does not exist, this function effectively does not attempt to guess what the final permissions of the file would be. In other words, it just does whatever `open()` would do. In this case it is the user's responsibilty to change permissions as needed after creating the file.

Examples

```
>>> with unlock_file('read-only.txt', 'w') as f:
...     f.write(new_data)
```

`desiutil.io.`**`yamlify`**(*obj*, *debug=False*)

Recursively process an object so it can be serialised for yaml. Based on jsonify in linetools.

Note: All string-like keys in `dict` s are converted to `str`.

Parameters

- **`obj`** (`object`) – Any object.
- **`debug`** (`bool`, optional) – Print extra information if requested.

Returns An object suitable for yaml serialization. For example `numpy.ndarray` is converted to `list`, `numpy.int64` is converted to `int`, etc.

Return type `object`

3.1.12 desiutil.log

DESI-specific utility functions that wrap the standard `logging` module.

This module is intended to support three different logging use patterns:

1. Just get an easy-to-use, pre-configured logging object.
2. Easily change the log level temporarily within a function. This is provided by a context manager.
3. Change the default log level on the command-line. This can actually be accomplished in two ways: the command-line interpreter can call *`get_logger()`* with the appropriate level, or the environment variable `DESI_LOGLEVEL` can be set.

In addition, it is possible to add timestamps and change the delimiter of log messages as needed. See the optional arguments to *`get_logger()`*.

Examples

Simplest possible use:

```
>>> from desiutil.log import log
>>> log.info('This is some information.')
```

This is exactly equivalent to:

```
>>> from desiutil.log import get_logger
>>> log = get_logger()
>>> log.info('This is some information.')
```

Temporarily change the log level with a context manager:

```
>>> from desiutil.log import get_logger, DesiLogContext, DEBUG
>>> log = get_logger()  # defaults to INFO
>>> log.info('This is some information.')
>>> log.debug("This won't be logged.")
>>> with DesiLogContext(log, DEBUG):
...     log.debug("This will be logged.")
>>> log.debug("This won't be logged.")
```

Create the logger with a different log level:

```
>>> from desiutil.log import get_logger, DEBUG
>>> if options.debug:
...     log = get_logger(DEBUG)
>>> else:
...     log = get_logger()
```

class desiutil.log.**DesiLogContext**(*logger*, *level=None*)
Provides a context manager to temporarily change the log level of an existing logging object.

Parameters

- **logger** (logging.Logger) – Logging object.
- **level** (int, optional) – The logging level to set. If it is not set, this whole class does nothing.

exception desiutil.log.**DesiLogWarning**
Warnings related to misconfiguration of the DESI logging object.

desiutil.log.**_configure_root_logger**(*timestamp=False*, *delimiter=':'*)
Configure a root logger.

Parameters

- **timestamp** (bool, optional) – If True, add a timestamp to the log message.
- **delimiter** (str, optional) – Use *delimiter* to separate fields in the log message (default :).

Returns The name of the root logger, suitable for input to logging.getLogger().

Return type str

desiutil.log.**get_logger**(*level=None*, *timestamp=False*, *delimiter=':'*)
Returns a default DESI logger.

Parameters

- **level** (int or str, optional) – Set the logging level (default INFO).
- **timestamp** (bool, optional) – If True, add a timestamp to the log message.
- **delimiter** (str, optional) – Use *delimiter* to separate fields in the log messages (default :).

Returns A logging object configured with the DESI defaults.

Return type logging.Logger

Notes

- If *level* is not None, that sets the log level, overriding anything else.
- If *level* is not set, and if the environment variable DESI_LOGLEVEL exists and has value DEBUG, INFO, WARNING, ERROR or CRITICAL (upper or lower case), that is used to set the log level.
- If DESI_LOGLEVEL is not set and *level* is None, the default level is set to INFO.

3.1.13 desiutil.modules

This package contains code for processing and installing Module files.

desiutil.modules.**_write_module_data**(*filename*, *data*)
Write and permission-lock Module file data. This is intended to consolidate some duplicated code.

desiutil.modules.**configure_module**(*product*, *version*, *product_root*, *working_dir=None*, *dev=False*)
Decide what needs to go in the Module file.

Parameters

- **product** (str) – Name of the product.
- **version** (str) – Version of the product.
- **product_root** (str) – Directory that contains the installed code.
- **working_dir** (str, optional) – The directory to examine. If not set, the current working directory will be used.
- **dev** (bool, optional) – If True, interpret the directory as a 'development' install, *e.g.* a trunk or branch install.

Returns A dictionary containing the module configuration parameters.

Return type dict

desiutil.modules.**default_module**(*module_keywords*, *module_dir*)
Install or update a .version file to set the default Module.

Parameters

- **module_keywords** (dict) – The parameters to use for Module file processing.
- **module_dir** (str) – The directory where the Module file should be installed.

Returns The text of the processed .version file.

Return type str

desiutil.modules.**init_modules**(*moduleshome=None*, *method=False*, *command=False*)
Set up the Modules infrastructure.

Parameters

- **moduleshome** (str, optional) – The path containing the Modules init code. If not provided, MODULESHOME will be used.
- **method** (bool, optional) – If True the function returned will be suitable for converting into an instance method.

- **command** (bool, optional) – If True, return the command used to call Modules, rather than a function.

Returns A function that wraps the module() function, and deals with setting sys.path. Returns None if no Modules infrastructure could be found.

Return type callable

desiutil.modules.**process_module**(*module_file*, *module_keywords*, *module_dir*)
Process a Module file.

Parameters

- **module_file** (str) – A template Module file to process.
- **module_keywords** (dict) – The parameters to use for Module file processing.
- **module_dir** (str) – The directory where the Module file should be installed.

Returns The text of the processed Module file.

Return type str

3.1.14 desiutil.plots

Module for code plots.

class desiutil.plots.**MaskedArrayWithLimits**(**args*, ***kwargs*)
Masked array with additional *vmin*, *vmax* attributes.

This class accepts the same arguments as MaskedArray.

This is not a general-purpose subclass and is only intended to simplify passing *vmin*, *vmax* limits from *prepare_data()* to the plotting utility methods defined in this module.

Parameters

- **vmin** (float, optional) – Minimum value when used for clipping or masking.
- **vmax** (float, optional) – Maximum value when used for clipping or masking.

vmin
Minimum value when used for clipping or masking.

Type float

vmax
Maximum value when used for clipping or masking.

Type float

desiutil.plots.**init_sky**(*projection='mollweide'*, *ra_center=120*, *galactic_plane_color='red'*, *ecliptic_plane_color='red'*, *ax=None*)
Initialize matplotlib axes with a projection of the full sky.

Parameters

- **projection** (str, optional) – Projection to use. Defaults to 'mollweide'. To show the available projections, call matplotlib.projections.get_projection_names().
- **ra_center** (float, optional) – Projection is centered at this RA in degrees. Default is +120°, which avoids splitting the DESI northern and southern regions.

- **galactic_plane_color** (*color name, optional*) – Draw a solid curve representing the galactic plane using the specified color, or do nothing when None.
- **ecliptic_plane_color** (*color name, optional*) – Draw a dotted curve representing the ecliptic plane using the specified color, or do nothing when None.
- **ax** (Axes, optional) – Axes to use for drawing this map, or create new axes if None.

Returns A matplotlib Axes object. Helper methods projection_ra() and projection_dec() are added to the object to facilitate conversion to projection coordinates.

Return type Axes

Notes

If requested, the ecliptic and galactic planes are plotted with zorder set to 20. This keeps them above most other plotted objects, but legends should be set to a zorder higher than this value, for example:

```
leg = ax.legend(ncol=2, loc=1)
leg.set_zorder(25)
```

desiutil.plots.**plot_grid_map**(*data*, *ra_edges*, *dec_edges*, *cmap='viridis'*, *colorbar=True*, *label=None*, *ax=None*, ***kwargs*)

Plot an array of 2D values using an all-sky projection.

Pass the data array through *prepare_data()* to select a subset to plot and clip the color map to specified values or percentiles.

This function is similar to *plot_healpix_map()* but is generally faster and has better handling of RA wrap around artifacts.

Additional keyword parameters will be passed to *init_sky()*.

Parameters

- **data** (*array or masked array*) – 2D array of data associated with each grid cell, with shape $(N_{\rm RA}, N_{\rm Dec})$. Use the output of *prepare_data()* as a convenient way to specify data cuts and color map clipping.
- **ra_edges** (*array*) – 1D array of $N_{\rm RA} + 1$ RA grid edge values in degrees, which must span the full circle, *i.e.*, ra_edges[0] == ra_edges[-1] - 360. The RA grid does not need to match the edges of the projection, in which case any wrap-around cells will be duplicated on both edges.
- **dec_edges** (*array*) – 1D array of $N_{\rm Dec} + 1$ Dec grid edge values in degrees. Values are not required to span the full range [-90, +90].
- **cmap** (*colormap name or object, optional*) – Matplotlib colormap to use for mapping data values to colors.
- **colorbar** (bool, optional) – Draw a colorbar below the map when True.
- **label** (str, optional) – Label to display under the colorbar. Ignored unless colorbar is True.
- **ax** (Axes, optional) – Axes to use for drawing this map, or create default axes using *init_sky()* when None.

Returns The axis object used for the plot.

Return type Axes

desiutil.plots.**plot_healpix_map**(*data*, *nest=False*, *cmap='viridis'*, *colorbar=True*, *label=None*, *ax=None*, ***kwargs*)

Plot a healpix map using an all-sky projection.

Pass the data array through *prepare_data()* to select a subset to plot and clip the color map to specified values or percentiles.

This function is similar to *plot_grid_map()* but is generally slower at high resolution and has less elegant handling of pixels that wrap around in RA, which are not drawn.

Requires that matplotlib and healpy are installed.

Additional keyword parameters will be passed to *init_sky()*.

Parameters

- **data** (*array or masked array*) – 1D array of data associated with each healpix. Must have a size that exactly matches the number of pixels for some NSIDE value. Use the output of *prepare_data()* as a convenient way to specify data cuts and color map clipping.
- **nest** (bool, optional) – If True, assume NESTED pixel ordering. Otheriwse, assume RING pixel ordering.
- **cmap** (*colormap name or object, optional*) – Matplotlib colormap to use for mapping data values to colors.
- **colorbar** (bool, optional) – Draw a colorbar below the map when True.
- **label** (str, optional) – Label to display under the colorbar. Ignored unless colorbar is True.
- **ax** (Axes, optional) – Axes to use for drawing this map, or create default axes using *init_sky()* when None.

Returns The axis object used for the plot.

Return type Axes

desiutil.plots.**plot_iers**(*which='auto'*, *num_points=500*, *save=None*)

Plot IERS data from 2015-2025.

Plots the UT1-UTC time offset and polar x,y motions over a 10-year period that includes the DESI survey, to demonstrate the time ranges when different sources (IERS-A, IERS-B) are used and where values are predicted then fixed.

This function is primarily intended to document and debug the *desiutil.iers.freeze_iers()* function.

Requires that the matplotlib package is installed.

Parameters

- **which** (*{'auto', 'A', 'B', 'frozen'}*) – Select which IERS table source to use. The default 'auto' matches the internal astropy default. Use 'A' or 'B' to force the source to be either the latest IERS-A table (which will be downloaded), or the IERS-B table packaged with the current version of astropy. The 'frozen' option calls *freeze_iers()*.
- **num_points** (int, optional) – The number of times covering 2015-25 to calculate and plot.
- **save** (str, optional) – 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()

desiutil.plots.**plot_sky_binned**(*ra*, *dec*, *weights=None*, *data=None*, *plot_type='grid'*, *max_bin_area=5*, *clip_lo=None*, *clip_hi=None*, *verbose=False*, *cmap='viridis'*, *colorbar=True*, *label=None*, *ax=None*, *return_grid_data=False*, ***kwargs*)

Show objects on the sky using a binned plot.

Bin values either show object counts per unit sky area or, if an array of associated data values is provided, mean data values within each bin. Objects can have associated weights.

Requires that matplotlib is installed. When plot_type is "healpix", healpy must also be installed.

Additional keyword parameters will be passed to *init_sky()*.

Parameters

- **ra** (*array*) – Array of object RA values in degrees. Must have the same shape as dec and will be flattened if necessary.
- **dec** (*array*) – Array of object Dec values in degrees. Must have the same shape as ra and will be flattened if necessary.
- **weights** (*array, optional*) – Optional of weights associated with each object. All objects are assumed to have equal weight when this is None.
- **data** (*array, optional*) – Optional array of scalar values associated with each object. The resulting plot shows the mean data value per bin when data is specified. Otherwise, the plot shows counts per unit sky area.
- **plot_type** (*{'grid', 'healpix'}*) – Must be either 'grid' or 'healpix', and selects whether data in binned in healpix or in (sin(Dec), RA).
- **max_bin_area** (float, optional) – The bin size will be chosen automatically to be as close as possible to this value but not exceeding it.
- **clip_lo** (float or str, optional) – Clipping is applied to the plot data calculated as counts / area or the mean data value per bin. See *prepare_data()* for details.
- **clip_hi** (float or str, optional) – Clipping is applied to the plot data calculated as counts / area or the mean data value per bin. See *prepare_data()* for details.
- **verbose** (bool, optional) – Print information about the automatic bin size calculation.
- **cmap** (*colormap name or object, optional*) – Matplotlib colormap to use for mapping data values to colors.
- **colorbar** (bool, optional) – Draw a colorbar below the map when True.
- **label** (str, optional) – Label to display under the colorbar. Ignored unless colorbar is True.
- **ax** (Axes, optional) – Axes to use for drawing this map, or create default axes using *init_sky()* when None.
- **return_grid_data** (bool, optional) – If True, return (ax, grid_data) instead of just ax.

Returns The axis object used for the plot, and the grid_data if *return_grid_data* is True.

Return type Axes or (ax, grid_data)

desiutil.plots.**plot_sky_circles**(*ra_center*, *dec_center*, *field_of_view=3.2*, *data=None*, *cmap='viridis'*, *facecolors='skyblue'*, *edgecolor='none'*, *colorbar=True*, *colorbar_ticks=None*, *label=None*, *ax=None*, ***kwargs*)

Plot circles on an all-sky projection.

Pass the optional data array through *prepare_data()* to select a subset to plot and clip the color map to specified values or percentiles.

Requires that matplotlib is installed.

Additional keyword parameters will be passed to *init_sky()*.

Parameters

- **ra_center** (*array*) – 1D array of RA in degrees at the centers of each circle to plot.
- **dec_center** (*array*) – 1D array of DEC in degrees at the centers of each circle to plot.
- **field_of_view** (*array*) – Full sky openning angle in degrees of the circles to plot. The default is appropriate for a DESI tile.
- **data** (*array, optional*) – 1D array of data associated with each circle, used to set its facecolor.
- **cmap** (*colormap name, optional*) – Matplotlib colormap to use for mapping data values to colors. Ignored unless data is specified.
- **facecolors** (*matplotlib color or array of colors, optional*) – Ignored when data is specified. An array must have one entry per circle or a single value is used for all circles.
- **edgecolor** (*matplotlib color, optional*) – The edge color used for all circles. Use 'none' to hide edges.
- **colorbar** (bool, optional) – Draw a colorbar below the map when `True` and data is provided.
- **colorbar_ticks** (list, optional) – Use the specified colorbar ticks or determine them automatically when None.
- **label** (str) – Label to display under the colorbar. Ignored unless a colorbar is displayed.
- **ax** (Axes, optional) – Axes to use for drawing this map, or create default axes using *init_sky()* when `None`.

Returns The axis object used for the plot.

Return type Axes

desiutil.plots.**plot_slices**(*x*, *y*, *x_lo*, *x_hi*, *y_cut*, *num_slices=5*, *min_count=100*, *axis=None*, *set_ylim_from_stats=True*, *scatter=True*)

Scatter plot with 68, 95 percentiles superimposed in slices. Modified from code written by D. Kirkby

Requires that the matplotlib package is installed.

Parameters

- **x** (array of float) – X-coordinates to scatter plot. Points outside [x_lo, x_hi] are not displayed.
- **y** (array of float) – Y-coordinates to scatter plot. Y values are assumed to be roughly symmetric about zero.
- **x_lo** (float) – Minimum value of *x* to plot.

- **x_hi** (float) – Maximum value of x to plot.
- **y_cut** (float) – The target maximum value of $|y|$. A dashed line at this value is added to the plot, and the vertical axis is clipped at $|y|$ = 1.25 * *y_cut* (but values outside this range are included in the percentile statistics).
- **num_slices** (int, optional) – Number of equally spaced slices to divide the interval [x_lo, x_hi] into.
- **min_count** (int, optional) – Do not use slices with fewer points for superimposed percentile statistics.
- **axis** (matplotlib.axes.Axes, optional) – Uses the current axis if this is not set.
- **set_ylim_from_stats** (bool, optional) – Set ylim of plot from 95% stat.
- **scatter** (*bool, optional*) – Show the data as a scatter plot. Best to limit to small datasets

Returns The Axes object used in the plot.

Return type matplotlib.axes.Axes

desiutil.plots.**prepare_data**(*data*, *mask=None*, *clip_lo=None*, *clip_hi=None*, *save_limits=False*)

Prepare array data for color mapping.

Data is clipped and masked to be suitable for passing to matplotlib routines that automatically assign colors based on input values.

Parameters

- **data** (*array or masked array*) – Array of data values to assign colors for.
- **mask** (*array of bool or None*) – Array of bools with same shape as data, where True values indicate values that should be ignored when assigning colors. When None, the mask of a masked array will be used or all values of an unmasked array will be used.
- **clip_lo** (*float or str*) – Data values below clip_lo will be clipped to the minimum color. If clip_lo is a string, it should end with "%" and specify a percentile of un-masked data to clip below.
- **clip_hi** (*float or str*) – Data values above clip_hi will be clipped to the maximum color. If clip_hi is a string, it should end with "%" and specify a percentile of un-masked data to clip above.
- **save_limits** (*bool*) – Save the calculated lo/hi clip values as attributes vmin, vmax of the returned masked array. Use this flag to indicate that plotting functions should use these vmin, vmax values when mapping the returned data to colors.

Returns Masked numpy array with the same shape as the input data, with any input mask applied (or copied from an input masked array) and values clipped to [clip_lo, clip_hi].

Return type masked array

Examples

If no optional parameters are specified, the input data is returned with only non-finite values masked:

```
>>> data = np.arange(5.)
>>> prepare_data(data)
masked_array(data = [0.0 1.0 2.0 3.0 4.0],
             mask = [False False False False False],
```

(continues on next page)

(continued from previous page)

```
       fill_value = 1e+20)
<BLANKLINE>
```

Any mask selection is propagated to the output:

```
>>> prepare_data(data, data == 2)
masked_array(data = [0.0 1.0 -- 3.0 4.0],
             mask = [False False  True False False],
       fill_value = 1e+20)
<BLANKLINE>
```

Values can be clipped by specifying any combination of percentiles (specified as strings ending with "%") and numeric values:

```
>>> prepare_data(data, clip_lo='25%', clip_hi=3.5)
masked_array(data = [1.0 1.0 2.0 3.0 3.5],
             mask = [False False False False False],
       fill_value = 1e+20)
<BLANKLINE>
```

Clipped values are also masked when the clip value or percentile is prefixed with "!":

```
>>> prepare_data(data, clip_lo='!25%', clip_hi=3.5)
masked_array(data = [-- 1.0 2.0 3.0 3.5],
             mask = [ True False False False False],
       fill_value = 1e+20)
<BLANKLINE>
```

An input masked array is passed through without any copying unless clipping is requested:

```
>>> masked = numpy.ma.arange(5)
>>> masked is prepare_data(masked)
True
```

Use the save_limits option to store the clipping limits as vmin, vmax attributes of the returned object:

```
>>> d = prepare_data(data, clip_lo=1, clip_hi=10, save_limits=True)
>>> d.vmin, d.vmax
(1.0, 10.0)
```

These attributes can then be used by plotting routines to fix the input range used for colormapping, independently of the actual range of data.

3.1.15 desiutil.redirect

Utilities for redirecting stdout / stderr to files.

`desiutil.redirect._get_libc()`
Helper function to import libc once.

`desiutil.redirect.stdouterr_redirected(to=None, comm=None)`
Redirect stdout and stderr to a file.

The general technique is based on:

http://stackoverflow.com/questions/5081657 http://eli.thegreenplace.net/2015/redirecting-all-kinds-of-stdout-in-python/

If the optional communicator is specified, then each process redirects to a different temporary file. Upon exit from the context the rank zero process concatenates these in order to the final file result.

If the enclosing code raises an exception, the traceback is printed to the log file.

Parameters

- **to** (*str*) – The output file name.
- **comm** (*mpi4py.MPI.Comm*) – The optional MPI communicator.

3.1.16 desiutil.setup

This package contains code that might be useful in DESI setup.py files.

class desiutil.setup.**DesiAPI**(*dist*, ***kw*)
Generate an api.rst file.

finalize_options()
Set final values for all the options that this command supports. This is always called as late as possible, ie. after any option assignments from the command-line or from other commands have been done. Thus, this is the place to code option dependencies: if 'foo' depends on 'bar', then it is safe to set 'foo' from 'bar' as long as 'foo' still has the same value it was assigned in 'initialize_options()'.

This method must be implemented by all command classes.

initialize_options()
Set default values for all the options that this command supports. Note that these defaults may be overridden by other commands, by the setup script, by config files, or by the command-line. Thus, this is not the place to code dependencies between options; generally, 'initialize_options()' implementations are just a bunch of "self.foo = None" assignments.

This method must be implemented by all command classes.

run()
A command's raison d'etre: carry out the action it exists to perform, controlled by the options initialized in 'initialize_options()', customized by other commands, the setup script, the command-line, and config files, and finalized in 'finalize_options()'. All terminal output and filesystem interaction should be done by 'run()'.

This method must be implemented by all command classes.

class desiutil.setup.**DesiModule**(*dist*, ***kw*)
Allow users to install module files with `python setup.py module_file`.

finalize_options()
Set final values for all the options that this command supports. This is always called as late as possible, ie. after any option assignments from the command-line or from other commands have been done. Thus, this is the place to code option dependencies: if 'foo' depends on 'bar', then it is safe to set 'foo' from 'bar' as long as 'foo' still has the same value it was assigned in 'initialize_options()'.

This method must be implemented by all command classes.

initialize_options()
Set default values for all the options that this command supports. Note that these defaults may be overridden by other commands, by the setup script, by config files, or by the command-line. Thus, this is not the place to code dependencies between options; generally, 'initialize_options()' implementations are just a bunch of "self.foo = None" assignments.

This method must be implemented by all command classes.

run()
A command's raison d'etre: carry out the action it exists to perform, controlled by the options initialized in 'initialize_options()', customized by other commands, the setup script, the command-line, and config files, and finalized in 'finalize_options()'. All terminal output and filesystem interaction should be done by 'run()'.

This method must be implemented by all command classes.

class desiutil.setup.**DesiTest**(*dist*, ***kw*)
Add coverage to test commands.

finalize_options()
Set final values for all the options that this command supports. This is always called as late as possible, ie. after any option assignments from the command-line or from other commands have been done. Thus, this is the place to code option dependencies: if 'foo' depends on 'bar', then it is safe to set 'foo' from 'bar' as long as 'foo' still has the same value it was assigned in 'initialize_options()'.

This method must be implemented by all command classes.

initialize_options()
Set default values for all the options that this command supports. Note that these defaults may be overridden by other commands, by the setup script, by config files, or by the command-line. Thus, this is not the place to code dependencies between options; generally, 'initialize_options()' implementations are just a bunch of "self.foo = None" assignments.

This method must be implemented by all command classes.

class desiutil.setup.**DesiVersion**(*dist*, ***kw*)
Allow users to easily update the package version with `python setup.py version`.

finalize_options()
Set final values for all the options that this command supports. This is always called as late as possible, ie. after any option assignments from the command-line or from other commands have been done. Thus, this is the place to code option dependencies: if 'foo' depends on 'bar', then it is safe to set 'foo' from 'bar' as long as 'foo' still has the same value it was assigned in 'initialize_options()'.

This method must be implemented by all command classes.

initialize_options()
Set default values for all the options that this command supports. Note that these defaults may be overridden by other commands, by the setup script, by config files, or by the command-line. Thus, this is not the place to code dependencies between options; generally, 'initialize_options()' implementations are just a bunch of "self.foo = None" assignments.

This method must be implemented by all command classes.

run()
A command's raison d'etre: carry out the action it exists to perform, controlled by the options initialized in 'initialize_options()', customized by other commands, the setup script, the command-line, and config files, and finalized in 'finalize_options()'. All terminal output and filesystem interaction should be done by 'run()'.

This method must be implemented by all command classes.

desiutil.setup.**find_version_directory**(*productname*)
Return the name of a directory containing version information.

Looks for files in the following places:

- py/*productname*/_version.py
- *productname*/_version.py

Parameters **productname** (str) – The name of the package.

Returns Name of a directory that can or does contain version information.

Return type str

Raises IOError – If no valid directory can be found.

desiutil.setup.**get_version**(*productname*)
Get the value of __version__ without having to import the module.

Parameters **productname** (str) – The name of the package.

Returns The value of __version__.

Return type str

desiutil.setup.**update_version**(*productname*, *tag=None*)
Update the _version.py file.

Parameters

- **productname** (str) – The name of the package.
- **tag** (str, optional) – Set the version to this string, unconditionally.

Raises IOError – If the repository type could not be determined.

3.1.17 desiutil.sklearn

Useful functions from the sklearn python package.

class desiutil.sklearn.**GaussianMixtureModel**(*weights*, *means*, *covars*, *covtype='full'*)
Read and sample from a pre-defined Gaussian mixture model.

Parameters

- **weights** (numpy.ndarray) – A 1D array of weights. The length of the array is the number of components
- **means** (numpy.ndarray) – A 2D array of means. The number of rows is the number of components. The number of columns is the number of dimensions.
- **covars** (numpy.ndarray) – A 3D array of covariances. The first dimension is the number of components. Each component has a 2D array with size given by the number of dimensions.
- **covtype** (str, optional) – Type of covariance. Defaults to 'full'.

static load(*filename*)
Load a model from a file.

Parameters **filename** (str) – The name of the file to load from.

Returns The model that was in *filename*.

Return type *desiutil.sklearn.GaussianMixtureModel*

sample(*n_samples=1*, *random_state=None*)
Sample from a model.

Parameters

- **n_samples** (int, optional) – Number of samples to return, default 1.

- **random_state** (numpy.random.RandomState, optional) – A random state object.

Returns An array containing the samples.

Return type numpy.ndarray

Raises ValueError – If the covariance type is unknown.

static save(*model*, *filename*)
Save a model to a file.

Parameters

- **model** (*desiutil.sklearn.GaussianMixtureModel*) – The model to be saved.
- **filename** (str) – The name of the file to save to.

3.1.18 desiutil.stats

Just contains a dead-simple wrapper on numpy.percentile().

desiutil.stats.**perc**(*x*, *per=68.2*)
Calculate the percentile bounds of a distribution, *i.e.* for per=68, the code returns the upper and lower bounds that encompass 68 percent of the distribution.

Uses simple interpolation.

Parameters

- **x** (numpy.ndarray) – numpy array of values
- **per** (float, optional) – Percentile for the calculation [0-100].

Returns Value at lower, value at upper.

Return type numpy.ndarray

3.1.19 desiutil.svn

This package contains code for interacting with DESI svn products.

desiutil.svn.**last_revision**()
Get the svn revision number.

Returns The latest svn revision number. A revision number of 0 indicates an error of some kind.

Return type str

Notes

This assumes that you're running `python setup.py version` in an svn checkout directory.

desiutil.svn.**last_tag**(*tags*, *username=None*)
Scan an SVN tags directory and return the most recent tag.

Parameters

- **tags** (str) – A URL pointing to an SVN tags directory.
- **username** (str, optional) – If set, pass the value to SVN's `--username` option.

Returns The most recent tag found in `tags`.

Return type `str`

`desiutil.svn.`**`version`**(*productname*, *url=None*)

Returns the version of a package.

Parameters

- **`productname`** (`str`) – The name of the package.
- **`url`** (`str`, optional) – If the product is not defined in the known_products file, the URL can be set this way.

Returns A **PEP 386**-compatible version string.

Return type `str`

Notes

The version string should be compatible with **PEP 386** and **PEP 440**.

3.1.20 desiutil.timer

Provides *Timer* class to standardize reporting of algorithm and I/O timing.

The *timer.Timer* class is intended for reporting timing of high-level events that take seconds to minutes, e.g. reporting the timing for the input, calculation, and output steps of a spectroscopic pipeline script. It is *not* intended for detailed profiling of every possible function for the purposes of optimization studies (use python cProfile for that); nor is it intended for timing small functions deep in the call stack that could result in N>>1 messages when run in parallel.

Example:

```
from desiutil.timer import Timer
t = Timer()
t.start('blat')

#- Use context manager for timing simple steps, e.g. one-liners
with t.time('blat.input'):
    stuff = io.read(filename)

#- Or use full start/stop if you prefer
t.start('blat.algorithm')
results = calculate(stuff)
t.stop('blat.algorithm')

with t.time('blat.output'):
    io.write(results)

#- Stop outer timer; Note that named timers can be nested or interleaved
t.stop('blat')

#- Print a json report of the timing
print(t.report())
```

This module has the philosophy that adding timing information should not crash your code, even if the timer is used incorrectly, e.g. starting or stopping a timer multiple times, stopping a timer that was never started, or forgetting to stop a timer before asking for a report. These print additional warnings or error messages, but don't raise exceptions.

class desiutil.timer.**Timer**(*silent=False*)

A basic timer class for standardizing reporting of algorithm and I/O timing

TIMER:<START|STOP>:<filename>:<lineno>:<funcname>: <message>

_prefix(*step*)

Return standard prefix string for timer reporting

Parameters **step** (*str*) – timing step, e.g. "START" or "STOP"

_print(*level*, *message*)

Print message with timing level prefix if not *self.silent*

Parameters

- **level** (*str*) – timing level ('START', 'STOP', 'WARNING', ...)
- **message** (*str*) – message to print after standardized prefix

cancel(*name*)

Cancel timer *name* and remove from timing log

report()

Return a json dump of self.timers, with start/stop as ISO-8601 strings

Implicitly stops any timers that are still running

Use *Timer.timers* for access as a dictionary, where start/stop are seconds elapsed since the epoch (1970-01-01T00:00:00 UTC on Unix).

start(*name*, *starttime=None*)

Start a timer *name* (str); prints TIMER-START message

Parameters **name** (*str*) – name of timer to stop

Options: starttime (str or float): start time to use instead of now

If *name* is started multiple times, the last call to *.start* is used for the timestamp.

Tries to parse *starttime* in this order:

1. (float) Unix time-since epoch
2. (str) ISO-8601
3. (str) Unix *date* cmd, e.g. "Mon Sep 21 20:09:48 PDT 2020"

stop(*name*, *stoptime=None*)

Stop timer *name* (str); prints TIMER-STOP message

Parameters **name** (*str*) – name of timer to stop

Options: stoptime (str or float): stop time to use instead of now

Returns time since start in seconds, or -1.0 if *name* wasn't started

Note: this purposefully does *not* raise an exception if called with a *name* that wasn't started, under the philosophy that adding timing statements shouldn't crash code, even if used incorrectly.

If a timer is stopped multiple times, the final stop and duration are timestamped as the last call to *Timer.stop*.

Tries to parse *stoptime* in this order:

1. (float) Unix time-since epoch
2. (str) ISO-8601

3. (str) Unix *date* cmd, e.g. "Mon Sep 21 20:09:48 PDT 2020"

stopall()
Stop any timers that have not yet been individually stopped

time(*name*)
Context manager for timing a code snippet.

Usage:

```
t = Timer()
with t.time('blat'):
    blat()
```

is equivalent to:

```
t = Timer()
t.start('blat')
blat()
t.stop('blat')
```

timer_seconds2iso8601()
Return copy *self.timers* with start/stop as ISO-8601 strings

Does *not* stop any running timers.

desiutil.timer.**compute_stats**(*timerlist*)
Compute timer min/max/mean/median stats

Parameters **timerlist** – list of Timer objects

Returns: dict[timername][...] with keys for start/stop/duration.min/max/mean/median

Different Timers can have different named subtimers

desiutil.timer.**parsetime**(*t*)
Parse time as int,float,str(int),str(float),ISO-8601, or Unix *date*

If *t* is None, return *time.time()*

Returns *t* as float Unix seconds since epoch timestamp

desiutil.timer.**timestamp2isotime**(*timestamp*)
Return seconds since epoch *timestamp* as ISO-8601 string

3.2 desiInstall

3.2.1 Introduction

This document describes the desiInstall process and the logic behind it.

The primary purpose of **desiInstall** is to install DESI software **at NERSC**. Using it to install software at locations other than NERSC is theoretically possible, but not supported.

3.2.2 Basic Invocation of desiInstall

desiInstall is invoked with the product and version to be installed:

```
desiInstall desiutil 3.0.3
```

The version should correspond to a tag in the repository corresponding to the product.

Branches can also be installed, and the resulting install will be a checkout of the repository, set to the requested branch. This is specified by prepending `branches/` to the version name:

```
desiInstall desiutil branches/branch-compile
```

Finally, for Subversion repositories, `trunk` is a shorthand for `branches/trunk`; for GitHub repositories, `main` is a shorthand for `branches/main`. All other branch names *must* be prepended with `branches/`.

3.2.3 Configuring desiInstall

desiInstall has many options, which are best viewed by typing `desiInstall --help`.

In addition, **desiInstall** both reads and sets several environment variables.

Environment variables that strongly affect the behavior of **desiInstall**.

DESICONDA This variable contains the path to the DESI+Anaconda infrastructure.

DESICONDA_VERSION This variable should contain the version of the DESI+Anaconda infrastructure.

DESIUTIL This variable contains the path to the installed version of desiutil. It is needed to find the `etc/desiutil.module` file.

NERSC_HOST This will automatically be set on NERSC systems. Although it is fine to manipulate this variable during unit tests, for general user and production purposes, it should be considered strictly read-only.

USER This variable is read to determine the username to pass to, *e.g.*, **svn**.

Environment variables that are *set* by **desiInstall** for use by **pip install** or **make**.

INSTALL_DIR This variable is *set* by **desiInstall** to the directory that will contain the final, installed version of the software package.

PRODUCT_VERSION This variable is *set* by **desiInstall**, with `PRODUCT` replaced by the actual name of the software being installed, *e.g.*, `DESISPEC_VERSION`.

WORKING_DIR This variable is *set* by **desiInstall** to the path containing a downloaded, expanded software package.

Environment variables related to the Modules infrastructure that may be manipulated by setting up Modules, or loading Module files.

LOADEDMODULES This variable contains a list of the Module files currently loaded. It may be manipulated by *desiutil.modules*.

MODULE_VERSION This variable is set on some NERSC systems and is needed to determine the full path to **modulecmd**.

MODULE_VERSION_STACK This variable is set on some NERSC systems may be set by *desiutil.modules* for compatibility.

MODULEPATH This variable contains a list of directories containing Module files. It may be manipulated by *desiutil.modules*.

MODULESHOME This variable points to the Modules infrastructure. If it is not set, it typically means that the system has no Modules infrastructure. This is needed to find the executable program that reads Module files.

PYTHONPATH Obviously this is important for any Python package! PYTHONPATH may be manipulated by *desiutil.modules*.

TCLSH May be used to determine the full path to **modulecmd.tcl** on systems with a pure-TCL Modules infrastructure.

3.2.4 Directory Structure Assumed by the Install

desiInstall is primarily intended to run in a production environment that supports Module files, *i.e.* at NERSC.

desiInstall *does not install a Modules infrastructure for you.* You have to do this yourself, if your system does not already have this.

For the purposes of this section, we define $product_root as the directory that **desiInstall** will be writing to. For standard NERSC installs it defaults to a pre-defined value. $product_root may contain the following directories:

code/ This contains the installed code, the result of **pip install .** or **make install**. The code is always placed in a product/version directory. So for example, the full path to **desiInstall** might be $product_root/code/desiutil/1.8.0/bin/desiInstall.

modulefiles/ This contains the the Module files installed by **desiInstall**. A Module file is almost always named product/version. For example, the Module file for desiutil might be $product_root/modulefiles/desiutil/1.8.0.

The --root option can override the built-in default value of $product_root, which is useful for testing:

```
desiInstall --root $SCRATCH/test_install desispec 0.20.0
```

In the example above, desispec would be installed in $SCRATCH/test_install/code/desispec/0.20.0, with a corresponding Module file at $SCRATCH/test_install/modulefiles/desispec/0.20.0

Within a $product_root/code/product/version directory, you might see the following:

bin/ Contains command-line executables, including Python or Shell scripts.

data/ Rarely, packages need data files that cannot be incorporated into the package structure itself, so it will be installed here. desimodel is an example of this.

etc/ Miscellaneous metadata and configuration. In most packages this only contains a template Module file.

lib/pythonX.Y/site-packages/ Contains installed Python code. X.Y would be, *e.g.*, 3.6 or 3.8.

py/ Sometimes we need to install a git checkout rather than an installed package. If so, the Python code will live in *this* directory not the lib/ directory, and the product's Module file will be adjusted accordingly.

3.2.5 Stages of the Install

Input Validation

desiInstall checks the command-line input, verifying that the user has specified a product and a version to install.

Product/Version Parsing

Because of the structures of the DESI code repositories, it is sometimes necessary to specify a directory name along with the product name. **desiInstall** contains a list of known products, but it is not necessarily complete.

desiInstall parses the input to determine the base name and base version to install. At this stage **desiInstall** also determines whether a branch install[1] has been requested.

The internal list of known products can be added to or overridden on the command line:

```
desiInstall -p new_product:https://github.com/me/new_product new_product 1.2.3

desiInstall -p desiutil:https://github.com/alternate_repository/desiutil desiutil 1.9.
↪9
```

The `-p` option can be specified multiple times, though in practice, it only matters to the product actually being installed.

Product Existence

After the product name and version have been determined, **desiInstall** constructs the full URL pointing to the product/version and runs the code necessary to verify that the product/version really exists. Typically, this will be **svn ls**, unless a GitHub install is detected.

Download Code

The code is downloaded, using **svn export** for standard (tag) installs, or **svn checkout** for branch installs. For GitHub installs, **desiInstall** will look for a release tarball, or do a **git clone** for tag or branch installs. **desiInstall** will set the environment variable `WORKING_DIR` to point to the directory containing this downloaded code.

Determine Build Type

The downloaded code is scanned to determine the build type. There are several possible build types that are mutually exclusive. They are derived in this order and the first matching method is used:

py If a setup.py file is detected, **desiInstall** will attempt to execute **pip install .**. This build type can be suppressed with the command line option `--compile-c`.

make If a Makefile is detected, **desiInstall** will attempt to execute **make install**.

src If a Makefile is not present, but a src/ directory is, **desiInstall** will attempt to execute **make -C src all**.

plain If no other build type is detected, the downloaded code is simply copied to the final install directory.

It is the responsibility of the code developer to understand these build types and choose the one appropriate for the package being developed.

Determine Install Directory

The install directory is where the code will live permanently. If the install is taking place at NERSC, the top-level install directory is predetermined based on the value of `NERSC_HOST`:

```
/global/common/software/desi/${NERSC_HOST}/desiconda/${DESICONDA_VERSION}
```

The actual install directory is determined by appending `/code/product/verson` to the combining the top-level directory listed above.

[1] In this document, "branch" refers to anything that is not a tagged version. This could include default branches such as "trunk" in Subversion repositories, or any default branch in a git repository.

If the install directory already exists, **desiInstall** will exit, unless the `--force` parameter is supplied on the command line.

desiInstall will set the environment variable `INSTALL_DIR` to point to the install directory.

Module Infrastructure

desiInstall sets up the Modules infrastructure by running code in *desiutil.modules* that is *based on* the Python init file supplied by the Modules infrastructure.

Find Module File

desiInstall will search for a module file in `$WORKING_DIR/etc`. If that module file is not found, **desiInstall** will use the file that comes with desiutil (*i.e.*, this product's own module file).

Load Dependencies

desiInstall will scan the module file identified in the previous stage, and will module load any dependencies found in the file.

Configure Module File

desiInstall will scan `WORKING_DIR` to determine the details that need to be added to the module file. The final module file will then be written into the DESI module directory at NERSC. If `--default` is specified on the command line, an appropriate .version file will be created.

Load Module

desiInstall will load the module file just created to set up any environment variables needed by the install. At this point it is also safe to assume that the environment variables `WORKING_DIR` and `INSTALL_DIR` exist. It will also set `PRODUCT_VERSION`, where `PRODUCT` will be replaced by the actual name of the package, *e.g.*, `DESIMODEL_VERSION`.

Create site-packages

If the build-type 'py' is detected, a site-packages directory will be created in `INSTALL_DIR`. If necessary, this directory will be added to Python's `sys.path`.

Can We Just Copy the Download?

If the build-type is *only* 'plain', or if a branch install is requested, the downloaded code will be copied to `INSTALL_DIR`. Further Python or C/C++ install steps described below will be skipped.

Run pip

If the build-type 'py' is detected, **pip install .** will be run at this point.

Build C/C++ Code

If the build-type 'make' is detected, **`make install`** will be run in `WORKING_DIR`. If the build-type 'src' is detected, **`make -C src all`** will be run in `INSTALL_DIR`.

Download Extra Data

If **`desiInstall`** detects `etc/product_data.sh`, where `product` should be replaced by the actual name of the package, it will download extra data not bundled with the code. The script should download data *directly* to `INSTALL_DIR`. The script should *only* be used with **`desiInstall`** and unit tests. Note that here are other, better ways to install and manipulate data that is bundled *with* a Python package.

Fix Permissions

The script **`fix_permissions.sh`** will be run on `INSTALL_DIR`.

Clean Up

The original download directory, specified by `WORKING_DIR`, is removed, unless `--keep` is specified on the command line.

3.3 Change Log

3.3.1 3.2.6 (unreleased)

- No changes yet.

3.3.2 3.2.5 (2022-01-20)

- Update *`desiutil.depend.add_dependencies()`* to include key environment variables like `DESI_ROOT` (PR #183).

3.3.3 3.2.4 (2022-01-10)

- Update *`desiutil.plots.plot_grid_map()`* to support matplotlib 3.5 (PR #181).

3.3.4 3.2.3 (2021-10-27)

- Optionally compute the MW dust transmission in the WISE bands (PR #175).
- Do not treat messages printed on STDERR as errors during **`desiInstall`** (PR #176).
- Add *`desiutil.brick.Bricks.brick_tan_wcs_size()`* to compute required size of TAN-projection WCS tiles (PR #177).
- Improve some test coverage; RTD fixes (PR #178).

3.3.5 3.2.2 (2021-06-03)

- Add support to `config_module()` for packages like QuasarNP where the GitHub name is capitalized but the internal Python package isn't (PR #173).

3.3.6 3.2.1 (2021-05-13)

- Changes in *desiutil.dust*: use Fitzpatrick reddening, add *dust_transmission()* function, include GAIA bands (PR #171).
- `desiutil.depend.possible_dependencies()`: add fiberassign, desimeter, and gpu_specter (direct commit).

3.3.7 3.2.0 (2021-03-29)

- Use **`pip install .`** instead of **`python setup.py install`** (PR #168).

3.3.8 3.1.2 (2021-02-15)

- Fixes for Numpy 1.20 (PR #162).
- **`desiInstall`** auto derive build type "py" or "make" or "src" but don't combine them (PR #163).
- **`desiInstall`** only fallback to NERSC default installdir if `--root` isn't specified (PR #163).
- Add *desiutil.depend.mergedep()* to merge DEPNAMnn/DEPVERnn dependencies between different headers (PR #164)

3.3.9 3.1.1 (2020-12-23)

- `astropy.erfa` no longer exists in more recent versions of Astropy (PR #159).
- Add function `dust.extinction_total_to_selective_ratio()` (PR #157).

3.3.10 3.1.0 (2020-12-11)

- Migrate unit tests to GitHub Actions; allow **`desiInstall`** to handle a diversity of possible branch names (PR #156).
- Add *redirect* for utilites related to redirecting STDOUT (PR #153).
- Add *Timer* class to standardize timing reports (PRs #151, #152).

3.3.11 3.0.3 (2020-08-04)

- Improve installation robustness when parsing `DESICONDA` environment variable; fix py3.8 SyntaxWarnings about "is not" usage (PR #150).

3.3.12 3.0.2 (2020-07-31)

- Travis testing with old healpy and old astropy (PR #149).
- Use https to avoid redirect for data downloads (PR #148).

3.3.13 3.0.1 (2020-06-12)

- Start migrating to use **pytest** to run tests instead of **python setup.py test** (PR #145).
- Update package list in *desiutil.install*; enable parallel **make** (PR #143).
- Protect against running **fix_permissions.sh** in HOME (PR #142).

3.3.14 3.0.0 (2020-04-15)

Note: minor *desiutil.plots* API and usage changes due to PR #141 so moving to major version 3.0.0, even though the majority of desiutil remains compatible with 2.x.x

- Remove all dependency on basemap (PR #141).

3.3.15 2.0.3 (2020-04-10)

- Add IERS functions originally in desisurvey (PR #139).

3.3.16 2.0.2 (2020-01-25)

- Update NERSC paths for CFS (PR #137).

3.3.17 2.0.1 (2019-09-24)

- Updated to latest ReadTheDocs configuration; standardized some docstrings for better appearance (PR #136).
- No code changes.

3.3.18 2.0.0 (2019-09-15)

- **This version does not support Python 2.**
- No significant API changes, however.

3.3.19 1.9.16 (2019-08-09)

- Add support for auto-generating API documentation via **python setup.py api** (PR #131).
- Fix basemap plot tests by using unique axes (PR #135).

3.3.20 1.9.15 (2018-12-14)

- Add *desiutil.dust.ext_odonnell()* and *desiutil.dust.ext_ccm()* originally from desispec (PR #128).
- Update permissions set by **fix_permissions.sh** (PR #126).
- Set read-only permissions on all Module files, and unlock them as needed (PR #125).
- Draw ecliptic in all-sky plots (PR #124).

3.3.21 1.9.14 (2018-10-05)

- Restrict write access on software installed with **desiInstall** (PR #122).

3.3.22 1.9.13 (2018-09-06)

- Add `/maps` to the default dust directory (PR #119).

3.3.23 1.9.12 (2018-09-05)

- Port the dust map code from desitarget to desiutil (PR #116).

3.3.24 1.9.11 (2018-05-10)

- Installing extra data happens *after* the main install, to prevent collisions in creating the install directory (Issue #102, PR #109).
- `fix_permissions.sh` ignores the group-write bit (Issue #108, PR #109).
- Remove support for a **desiInstall** configuration file. All options are specified on the command-line (Issue #103, PR #109).
- Update sklearn module to support updates to `sklearn.mixture.GaussianMixture` (PR #111).
- Added scatter option to *desiutil.plots.plot_slices()*; avoid slow PNG generation for large data samples (PR #112).

3.3.25 1.9.10 (2018-03-29)

- Remove support for **desiInstall** in environments other than NERSC (PR #101).
- Try as best as possible that Python executable scripts are installed with an explicit desiconda version (PR #105).

3.3.26 1.9.9 (2017-12-20)

- Enhance *desiutil.log* with a context manager (PR #92), and change the way the log level is set.
- Avoid logging interference with *desiutil.log.get_logger()* is called with different log levels (PR #93).
- Use `unittest.mock` to increase test coverage.

3.3.27 1.9.8 (2017-11-09)

- Adds redrock, surveysim, desisurvey, and healpy to dependency version checks.
- Adds redrock and surveysim to known products for installation.
- Fix team name in license file.
- Support new `/global/common/software` filesystem at NERSC.
- Support `coriknl` versions of desiconda.

3.3.28 1.9.7 (2017-09-29)

- Fixed some test failures that occurred in the NERSC environment and/or in an installed package, as opposed to a git checkout (PR #80).
- Fixed bug in *`desiutil.brick.Bricks.brick_radec()`* handling scalar inputs (PR #81).
- Fixed bugs that could cause bricks to be slightly too big, and that incorrectly special-cased the north pole with brick sizes that don't evenly divide 180 degrees (PR #84).
- Adds `return_grid_data` option to *`desiutil.plots.plot_sky_binned()`* (PR #85).
- Added tests of *`desiutil.sklearn`* (PR #86).

3.3.29 1.9.6 (2017-07-12)

- Changed the location where code is installed so that code is correctly matched to the corresponding DESI+Anaconda (desiconda) version (PR #77).

3.3.30 1.9.5 (2017-06-15)

- Improved correctness and functionality of *`desiutil.brick`* (PR #74).

3.3.31 1.9.4 (2017-06-01)

- Moved `desispec.brick` to *`desiutil.brick`* (PR #70).
- Get .travis.yml file and other components ready for Python 3.6.
- Increase test coverage in a few areas.
- Make basemap an optional dependency (PR #61).
- Fix **`desiInstall`** on cori.
- Add *`desiutil.census`* to calculate DESI disk space use.

3.3.32 1.9.3 (2017-03-01)

- Added new *`desiutil.sklearn`* module and `distutils.sklearn.GaussianMixtureModel` class to save and sample from a Gaussian mixture model.
- Added new functions for creating all-sky maps (PR #52) with an accompanying tutorial notebook in *doc/nb/*.

- Add option to **`fix_permissions.sh`** to remove group-writeability for "official" data. Also, make sure that files and directories are group-readable.
- Moved logging infrastructure from desispec (PR #56).

3.3.33 1.9.2 (2016-11-18)

- Enables desiInstall of desihub packages even if they aren't in the `desiutil.install.known_products` list yet.
- Include *`desiutil.plots`* in documentation.

3.3.34 1.9.1 (2016-10-17)

- Allow top-level `/python` directories to be detected (not just `/py`) to support redmonster.

3.3.35 1.9.0 (2016-10-12)

- Shorten Python version printed in dependency headers.
- `desiutil.test.test_plots` was not cleaning up after itself.
- Support new DESI+Anaconda software stack infrastructure (PR #43).
- Fixes *`names()`* when mask is a `numpy.uint64` (desihub/desitarget#79).
- *`names()`* is much faster.
- Fixed problem opening tar files in Python 3.

3.3.36 1.8.0 (2016-09-10)

- Added *`encode_table()`* and *`decode_table()`* for converting string columns in tables between unicode and bytes (PR #41).
- Set apache permissions by number instead of by name.

3.3.37 1.7.0 (2016-08-18)

- Added *`combine_dicts()`* function.
- Added *`desiutil.plots`* module including *`plot_slices()`*.

3.3.38 1.6.0 (2016-07-01)

- Fixed some import statements so documentation will build on readthedocs.
- *`add_dependencies()`* to add DEPNAM/DEPVER for common DESI dependencies

3.3.39 1.5.0 (2016-06-09)

- Fixed bug affecting people with the C version of Modules installed on laptops.
- Added *`desiutil.depend`* tools for manipulating DEPNAMnn and DEPVERnn keywords in FITS headers.

3.3.40 1.4.1 (2016-06-07)

- Don't consider warning messages about astropy_helpers to be errors.
- Update desiInstall documentation, adding information about environment variables.
- Use `distutils.command.sdist.sdist` to ensure that `MANIFEST.in` is respected.
- Add some test coverage in *desiutil.setup*.
- Cleaned up documentation of *desiutil.io* and several other modules.
- Modified conversion of keys to string in `desituil.io.yamlify`
- Log IP address in Travis Tests.

3.3.41 1.4.0 (2016-04-28)

- Fix module processing problem for non-DESI Python packages.
- Allow NERSC Modules root directory to be overridden in a configuration file.
- *desiutil.stats* module was previously snuck in, but never documented.
- Minor fixes for desiInstall bootstrap mode.
- PR #30: Enable use of weights in *iter_fit()*.
- Add a method for connverting Python objects to yaml-ready format. Includes `unicode` to `str` conversion.

3.3.42 1.3.6 (2016-03-25)

- Include *funcfits* in the documentation; added *mk_fit_dict()*.
- Improve coverage of *funcfits*.
- Try to use a nicer Sphinx theme for documentation.

3.3.43 1.3.5 (2016-03-15)

- Ignore some additional MANIFEST.in warnings.
- Allow known_products and cross-install configuration to be overridden using an optional configuration file.
- Allow products to specify a method to download additional data not bundled with the code.

3.3.44 1.3.4 (2016-02-22)

- Support GitHub tags that start with 'v'.
- Add support for speclite.

3.3.45 1.3.3 (2016-02-03)

- Added *stats* module to compute percentiles on distributions.

3.3.46 1.3.2 (2016-01-25)

- Recent versions of setuptools do not include `setuptools.compat`. A simple workaround was added to fix that.

3.3.47 1.3.1 (2016-01-12)

- Update MANIFEST.in file.
- Ignore additional warnings produced by MANIFEST.in.
- Always run `fix_permissions.sh` after install.
- Remove references to defunct hopper system.

3.3.48 1.3.0 (2015-12-09)

- Updated docstrings for bitmasks.
- Added *`funcfits`* module.

3.3.49 1.2.0 (2015-11-24)

- Added bitmask processing code, *`desiutil.bitmask`*.
- Fixed a minor variable name bug.
- Ignore warnings produced by processing MANIFEST.in.
- Fixed return value in cross_install.
- Fixed a missing run stage.

3.3.50 1.1.1 (2015-11-18)

- Update the list of NERSC hosts, including cori.
- Code is now PEP 8 compliant.

3.3.51 1.1.0 (2015-11-06)

- Don't print scary warning about `DESI_PRODUCT_ROOT` not being set if running at NERSC.
- Support running `python setup.py version` in svn products.
- Move Modules support code into separate sub-package.
- Simplify Travis build system.
- Remove some obsolete files.
- Simplify package structure.

3.3.52 1.0.1 (2015-11-03)

- Fix issue where the Python tarfile package was failing to autodetect gzipped files.

3.3.53 1.0.0 (2015-10-29)

- pip install support.
- Travis build support.
- Read the Docs support.
- Remove unnecessary Sphinx extensions.
- Create setup subpackage for functions that go in setup.py files.
- fix_permissions.sh won't clobber executable bits.

3.3.54 0.6.0 (2015-10-13)

Note: This tag should not be used or installed. It is an intermediate tag intended to fix a subtle issue with how svn tags are translated into git tags.

- Fixed a problem with log handling.
- Use `module switch` instead of `module load` when a module is already loaded.
- Add changes.rst file.
- Add LICENSE.rst file.
- Migration to GitHub - Change case of desiutil. - Add git support functions.

3.3.55 0.5.5 (2015-01-16)

- Fix a corner case when desiInstall tries to install desiUtil.
- Fix an svn version string parsing error.

3.3.56 0.5.4 (2015-01-16)

- Fix a minor syntax error.

3.3.57 0.5.3 (2015-01-16)

- Fix a minor syntax error.

3.3.58 0.5.2 (2015-01-16)

- Update desiInstall documentation.
- Changes to doc compilation.

3.3.59 0.5.1 (2015-01-14)

- Update desiInstall documentation.
- Handle `-hpcp` module names.
- Move build type detection to separate function.
- Move documentation generation to separate function.
- Add cross-install support.

3.3.60 0.5.0 (2015-01-14)

- Adding support for GitHub installs.

3.3.61 0.4.2 (2015-01-12)

- Fix a minor syntax error.

3.3.62 0.4.1 (2015-01-12)

- Fix a minor syntax error.

3.3.63 0.4.0 (2015-01-12)

- Major refactor of install, support 'plain' products.
- Use `svn --non-interactive` where possible.

3.3.64 0.3.9 (2014-09-12)

- Change the way tags are sorted.
- Tweak documentation compilation.

3.3.65 0.3.8 (2014-06-24)

- Change severity of certain log messages.

3.3.66 0.3.7 (2014-06-24)

- Minor fix to logging.

3.3.67 0.3.6 (2014-06-24)

- Don't auto-generate the desiInstall script.

3.3.68 0.3.5 (2014-06-24)

- Use ez_setup.py.

3.3.69 0.3.4 (2014-06-23)

- Reconfigure how the desiInstall script is created.

3.3.70 0.3.3 (2014-06-23)

- Tweak module file detection.

3.3.71 0.3.2 (2014-06-23)

- Fix chmod error.

3.3.72 0.3.1 (2014-06-23)

- Change `version()` to `__version__`.

3.3.73 0.3.0 (2014-06-10)

- Change how version strings are set.
- Auto-detect a variety of build types.

3.3.74 0.2.5 (2014-05-26)

- Fix how the Modules Python init file is detected.

3.3.75 0.2.4 (2014-05-06)

- Fix directory creation for trunk/branch installs.

3.3.76 0.2.3 (2014-05-02)

- Change how dependencies are handled in the module file.
- Move some dependency processing to separate function.
- General restructuring.

3.3.77 0.2.2 (2014-05-01)

- Copy extra files in the etc directory.
- Remove some data files from setup.py.

3.3.78 0.2.1 (2014-05-01)

- Tweak how versions are reported.

3.3.79 0.2.0 (2014-05-01)

- Tweak documentation.
- Add ACL detection to fix_permission script.

3.3.80 0.1 (2014-01-09)

- First tag.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Y