astroobs Documentation

Release

Author

August 05, 2016

Contents

Contents:

CHAPTER 1

astroobs package

1.1 Submodules

1.2 astroobs.Moon module

class `astroobs.Moon.`**Moon** (*obs=None*, *input_epoch='2000'*, ***kwargs*)
Bases: `astroobs.Target.Target`

Initialises the Moon. Optionaly, processes the Moon for the observatory and date given (refer to `Moon.process()`).

Args:

- obs (`Observatory`) [optional]: the observatory for which to process the Moon

Kwargs:

- raiseError (bool): if `True`, errors will be raised; if `False`, they will be printed. Default is `False`

Raises: N/A

dec
The declination of the Moon, displayed as tuple of np.array (+/-dd, mm, ss)

decStr
A pretty printable version of the mean of the declination of the moon

plot (*obs*, *y='alt'*, ***kwargs*)
Plots the y-parameter vs time diagram for the moon at the given observatory and date

Args:

- obs (`Observatory`): the observatory for which to plot the moon

Kwargs:

- See class constructor
- See `Observatory.plot()`
- See `Target.plot()`

Raises: N/A

polar (*obs*, ***kwargs*)
Plots the y-parameter vs time diagram for the moon at the given observatory and date

Args:

- obs (`Observatory`): the observatory for which to plot the moon

Kwargs:

- See class constructor
- See `Observatory.plot()`
- See `Target.plot()`

Raises: N/A

process (*obs*, ***kwargs*)

Processes the moon for the given observatory and date.

Args:

- obs (`Observatory`): the observatory for which to process the moon

Kwargs: See class constructor

Raises: N/A

Creates vector attributes:

- `airmass`: the airmass of the moon
- `ha`: the hour angle of the moon (degrees)
- `alt`: the altitude of the moon (degrees - horizon is 0)
- `az`: the azimuth of the moon (degrees)
- `ra`: the right ascension of the moon, see `Moon.ra()`
- `dec`: the declination of the moon, see `Moon.dec()`

Note:

•All previous attributes are vectors related to the time vector of the observatory used for processing: `obs.dates`

Other attributes:

- `rise_time`, `rise_az`: the time (ephem.Date) and the azimuth (degree) of the rise of the moon
- `set_time`, `set_az`: the time (ephem.Date) and the azimuth (degree) of the setting of the moon
- `transit_time`, `transit_az`: the time (ephem.Date) and the azimuth (degree) of the transit of the moon

Warning:

•it can occur that the moon does not rise or set for an observatory/date combination. In that case, the corresponding attributes will be set to `None`, i.e. `set_time`, `set_az`, `rise_time`, `rise_az`. In that case, an additional parameter is added to the Moon object: `Moon.alwaysUp` which is `True` if the Moon never sets and `False` if it never rises above the horizon.

ra

The right ascension of the Moon, displayed as tuple of np.array (hh, mm, ss)

raStr

A pretty printable version of the mean of the right ascension of the moon

1.3 astroobs.Observation module

class astroobs.Observation.**Observation**(*obs*, *long=None*, *lat=None*, *elevation=None*, *timezone=None*, *temp=None*, *pressure=None*, *moonAvoidRadius=None*, *local_date=None*, *ut_date=None*, *horizon_obs=None*, *dataFile=None*, *epoch='2000'*, ***kwargs*)

Bases: astroobs.Observatory.Observatory

Assembles together an Observatory (including itself the Moon target), and a list of Target.

For use and docs refer to:

- add_target() to add a target to the list
- rem_target() to remove one
- change_obs() to change the observatory
- change_date() to change the date of observation

Kwargs:

- raiseError (bool): if True, errors will be raised; if False, they will be printed. Default is False
- fig: TBD

Raises: See Observatory

Warning:
•it can occur that the Sun, the Moon or a target does not rise or set for an observatory/date combination. In that case, the corresponding attributes will be set to None

```
>>> import astroobs.obs as obs
>>> o = obs.Observation('ohp', local_date=(2015,3,31,23,59,59))
>>> o
Observation at Observatoire de Haute Provence on 2015/6/21-22. 0 targets.
    Moon phase: 89.2%
>>> o.moon
Moon - phase: 89.2%
>>> print o.sunset, '...', o.sunrise, '...', o.len_night
2015/3/31 18:08:40 ... 2015/4/1 05:13:09 ... 11.0746939826
>>> import ephem as E
>>> print(E.Date(o.sunsetastro+o.localTimeOffest), '...', E.Date(
        o.sunriseastro+o.localTimeOffest), '...', o.len_nightastro)
2015/3/31 21:43:28 ... 2015/4/1 05:38:26 ... 7.91603336949
>>> o.add_target('vega')
>>> o.add_target('mystar', dec=19.1824, ra=213.9153)
>>> o.targets
[Target: 'vega', 18h36m56.3s +38°35'8.1", O,
 Target: 'mystar', 14h15m39.7s +19°16'43.8", O]
>>> print("%s mags: 'K': %2.2f, 'R': %2.2f"%(o.targets[0].name,
        o.targets[0].flux['K'], o.targets[0].flux['R']))
vega mags: 'K': 0.13, 'R': 0.07
```

add_target(*tgt*, *ra=None*, *dec=None*, *name=''*, ***kwargs*)

Adds a target to the observation list

Args:

- tgt (see below): the index of the target in the Observation.targets list

- ra ('hh:mm:ss.s' or decimal degree) [optional]: the right ascension of the target to add to the observation list. See below
- dec ('+/-dd:mm:ss.s' or decimal degree) [optional]: the declination of the target to add to the observation list. See below
- name (string) [optional]: the name of the target to add to the observation list. See below

`tgt` arg can be:

- a `Target` instance: all other parameters are ignored
- a target name (string): if `ra` and `dec` are not `None`, the target is added with the provided coordinates; if `None`, a SIMBAD search is performed on `tgt`. `name` is ignored
- a ra-dec string ('hh:mm:ss.s +/-dd:mm:ss.s'): in that case, `ra` and `dec` will be ignored and `name` will be the name of the target

Kwargs: See class constructor

Raises:

- ValueError: if ra-dec formating was not understood

Note:

•Automatically processes the target for the given observatory and date

---

```
>>> import astroobs.obs as obs
>>> o = obs.Observation('ohp', local_date=(2015,3,31,23,59,59))
>>> arc = obs.TargetSIMBAD('arcturus')
>>> o.add_target(arc)
>>> o.add_target('arcturus')
>>> o.add_target('arcturusILoveYou', dec=19.1824, ra=213.9153)
>>> o.add_target('14:15:39.67 +10:10:56.67', name='arcturus')
>>> o.targets
[Target: 'arcturus', 14h15m39.7s +19°16'43.8", O,
 Target: 'arcturus', 14h15m39.7s +19°16'43.8", O,
 Target: 'arcturus', 14h15m39.7s +10°40'43.8", O,
 Target: 'arcturus', 14h15m39.7s +19°16'43.8", O]
```

`change_date`(*ut_date=None*, *local_date=None*, *recalcAll=False*, ***kwargs*)

Changes the date of the observation and optionaly re-processes targets for the same observatory and new date

Args:

- ut_date: Refer to `Observatory.upd_date()`
- local_date: Refer to `Observatory.upd_date()`
- recalcAll (bool or None) [optional]: if `False` (default): only targets selected for observation are re-processed, if `True`: all targets are re-processed, if `None`: no re-process

Kwargs: See class constructor

Raises:

- KeyError: if the twilight keyword is unknown
- Exception: if the observatory object has no date

change_obs(*obs*, *long=None*, *lat=None*, *elevation=None*, *timezone=None*, *temp=None*, *pressure=None*, *moonAvoidRadius=None*, *horizon_obs=None*, *dataFile=None*, *recalcAll=False*, ***kwargs*)

Changes the observatory and optionaly re-processes all target for the new observatory and same date

Args:

- recalcAll (bool or None) [optional]: if `False` (default): only targets selected for observation are re-processed, if `True`: all targets are re-processed, if `None`: no re-process

Kwargs: See class constructor

Note:

•Refer to `ObservatoryList.add()` for details on other input parameters

---

plot(*y='alt'*, ***kwargs*)

Plots the y-parameter vs time diagram for the target at the given observatory and date

Kwargs:

- See class constructor
- See `Observatory.plot()`
- moon (bool): if `True`, adds the moon to the graph, default is `True`
- autocolor (bool): if `True`, sets curves-colors automatically, default is `True`

Raises: N/A

polar(***kwargs*)

Plots the sky-view diagram for the target at the given observatory and date

Kwargs:

- See class constructor
- See `Observatory.plot()`
- moon (bool): if `True`, adds the moon to the graph, default is `True`
- autocolor (bool): if `True`, sets curves-colors automatically, default is `True`

Raises: N/A

rem_target(*tgt*, ***kwargs*)

Removes a target from the observation list

Args:

- tgt (int): the index of the target in the `Observation.targets` list

Kwargs: See class constructor

Raises: N/A

targets

Shows the list of targets recorded into the Observation

tick(*tgt*, *forceTo=None*, ***kwargs*)

Changes the ticked property of a target (whether it is selected for observation)

Args:

- tgt (int): the index of the target in the `Observation.targets` list

- forceTo (bool) [optional]: if `True`, selects the target for observation, if `False`, unselects it, if `None`, the value of the selection is inverted

Kwargs: See class constructor

Raises: N/A

Note:

•Automatically reprocesses the target for the given observatory and date if it is selected for observation

---

```
>>> import astroobs.obs as obs
>>> o = obs.Observation('ohp', local_date=(2015,3,31,23,59,59))
>>> o.add_target('arcturus')
>>> o.targets
[Target: 'arcturus', 14h15m39.7s +19°16'43.8", O]
>>> o.tick(4)
>>> o.targets
[Target: 'arcturus', 14h15m39.7s +19°16'43.8", -]
```

ticked
Shows whether the target was select for observation

1.4 astroobs.Observatory module

class `astroobs.Observatory.`**Observatory**(*obs*, *long=None*, *lat=None*, *elevation=None*, *timezone=None*, *temp=None*, *pressure=None*, *moonAvoidRadius=None*, *local_date=None*, *ut_date=None*, *horizon_obs=None*, *dataFile=None*, *epoch='2000'*, ***kwargs*)

Bases: `ephem.Observer`, `object`

Defines an observatory from which the ephemeris of the twilights or a night-sky target are processed. The *night-time* is base on the given date. It ends at the next sunrise and starts at the sunset preceeding this next sunrise.

Args:

- obs (str): id of the observatory to pick from the observatories database OR the name of the custom observatory (in that case, `long`, `lat`, `elevation`, `timezone` must also be given, `temp`, `pressure`, `moonAvoidRadius` are optional)
- local_date (see below): the date of observation in local time
- ut_date (see below): the date of observation in UT time
- horizon_obs (float - degrees): minimum altitude at which a target can be observed, default is 30 degrees altitude
- epoch (str): the 'YYYY' year in which all ra-dec coordinates are converted

Kwargs:

- raiseError (bool): if `True`, errors will be raised; if `False`, they will be printed. Default is `False`
- fig: TBD

Raises:

- NameError: if a mandatory input parameter is missing

- KeyError: if the observatory ID does not exist
- KeyError: if the twilight keyword is unknown
- Exception: if the observatory object has no date

Note:

•For details on `local_date` and `ut_date`, refer to `Observatory.upd_date()`

•For details on other input parameters, refer to `ObservatoryList.add()`

•The `Observatory` automatically creates and manages a `Moon` target under `moon` attribute

•If `obs` is the id of an observatory to pick in the database, the user can still provide `temp`, `pressure`, `moonAvoidRadius` attributes which will override the database default values

•`horizon` attribute is in radian

Main attributes:

- `localnight`: gives the local midnight time in local time (YYYY, MM, DD, 23, 59, 59)
- `date`: gives the local midnight time in UT time
- `dates`: is a vector of Dublin Julian Dates. Refer to `process_obs()`
- `lst`: the local sidereal time corresponding to each `dates` element
- `localTimeOffest`: gives the shift in days between UT and local time: local=UT+localTimeOffest
- `moon`: points to the `Moon` target processed for the given observatory and date

Twilight attributes:

- For the next three attributes, `XXX` shall be replaced by {'' (blank), 'civil', 'nautical', 'astro'} for, respectively, horizon, -6, -12, and -18 degrees altitude
- `sunriseXXX`: gives the sunrise time for different twilights, in Dublin Julian Dates. e.g.: `observatory.sunrise`
- `sunsetXXX`: gives the sunset time for different twilights, in Dublin Julian Dates. e.g.: `observatory.sunsetcivil`
- `len_nightXXX`: gives the night duration for different twilights (between corresponding sunset and sunrise), in hours. e.g.: `observatory.len_nightnautical`

Warning:

•it can occur that the Sun, the Moon or a target does not rise or set for an observatory/date combination. In that case, the corresponding attributes will be set to `None`

```
>>> import astroobs.obs as obs
>>> o = obs.Observatory('ohp', local_date=(2015,3,31,23,59,59))
>>> o
<ephem.Observer date='2015/3/31 21:59:59' epoch='2000/1/1 12:00:00'
lon=5:42:48.0 lat=43:55:51.0 elevation=650.0m horizon=-0:49:04.8
temp=15.0C pressure=1010.0mBar>
>>> o.moon
Moon - phase: 89.2%
>>> print(o.sunset, '...', o.sunrise, '...', o.len_night)
2015/3/31 18:08:40 ... 2015/4/1 05:13:09 ... 11.0746939826
>>> import ephem as E
```

```
>>> print(E.Date(o.sunsetastro+o.localTimeOffest), '...', E.Date(
        o.sunriseastro+o.localTimeOffest), '...', o.len_nightastro)
2015/3/31 21:43:28 ... 2015/4/1 05:38:26 ... 7.91603336949
```

nowArg
Returns the index of *now* in the `observatory.dates` vector, or None if *now* is out of its bounds (meaning the observation is not taking place now)

```
>>> import astroobs.obs as obs
>>> import ephem as E
>>> o = obs.Observatory('ohp')
>>> plt.plot(o.dates, o.moon.alt, 'k-')
>>> now = o.nowArg
>>> if now is not None:
>>>     plt.plot(o.dates[now], o.moon.alt[now], 'ro')
>>> else:
>>>     plt.plot([E.now(), E.now()], [o.moon.alt.min(),o.moon.alt.max()], 'r--')
```

plot (***kwargs*)
Plots the observatory diagram

Kwargs:

- See class constructor
- dt (float - hour): the spacing of x-axis labels, default is 1 hour (not with polar mode)
- t0 (float - DJD or [0-24]): the date of the first tick-label of x-axis, default is sunsetastro. The time type must correspond to `time` parameter (not with polar mode)
- xlim ([xmin, xmax]): bounds for x-axis, default is full night span (not with polar mode)
- retxdisp (bool): if `True`, bounds of x-axis displayed values are returned (`xdisp` key)
- ylim ([ymin, ymax]): bounds for y-axis, default is [horizon_obs-10, 90] (not with polar mode)
- xlabel (str): label for x-axis, default 'Time (UT)'
- ylabel (str): label for y-axis, default 'Elevation (°)'
- title (str): title of the diagram, default is observatory name or coordinates
- ymin_margin (float): margin between xmin of graph and horizon_obs. Low priority vs ylim, default is 10 (not with polar mode)
- retfignum (bool): if `True`, the figure number will be returned, default is `False`
- fignum (int): figure number on which to plot, default is `False`
- retaxnum (bool): if `True`, the ax index as in `figure.axes[n]` will be returned, default is `False`
- axnum (int): axes index on which to plot, default is `None` (create new ax)
- retfig (bool): if `True`, the figure object will be returned, default is `False`
- fig (figure): figure object on which to plot, default is `None` (use fignum)
- retax (bool): if `True`, the ax will be returned, default is `False`
- ax (axes): ax on which to plot, default is `None`
- now (bool): if `True` and within range, a vertical line as indication of "now" will be shown, default is True

- retnow (bool): returns the line object (`nowline` key) corresponding to the 'now-line', default is `False`
- legend (bool): whether to add a legend or not, default is `True`
- loc: location of the legend, default is 8 (top right), refer to plt.legend
- ncol: number of columns in the legend, default is 3, refer to plt.legend
- columnspacing: spacing between columns in the legend, refer to plt.legend
- lfs: legend font size, default is 11
- textlbl (bool): if `True`, a text label with target name or coordinates will be added near transit, default is `False`
- polar (bool): if `True`, plots the sky view, otherwise plots target attribute versus time
- time (str): the type of the x-axis time, `ut` for UT, `loc` for local time and `lst` [0-24] for local sidereal time, default is `ut` (not with polar mode)

Raises: N/A

process_obs(*pts=200*, *margin=15*, *fullhour=False*, ***kwargs*)

Processes all twilights as well as moon rise, set and position through night for the given observatory and date. Creates the vector `observatory.dates` which is the vector containing all timestamps at which the moon and the targets will be processed.

Args:

- pts (int) [optional]: the size of the `dates` vector, whose elements are linearly spaced in time
- margin (float - minutes) [optional]: the margin between the first element of the vector `dates` and the sunset, and between the sunrise and its last element
- fullhour (bool) [optional]: if `True`, then the vector `dates` will start and finish on the first full hour preceeding sunset and following sunrise

Kwargs: See class constructor

Raises:

- KeyError: if the twilight keyword is unknown
- Exception: if the observatory object has no date

Note: In case the observatory is in polar regions where the sun does not alway set and rise everyday, the first and last elements of the `dates` vector are set to local midday right before and after the local midnight of the observation date. e.g.: 24h night centered on the local midnight.

---

upd_date(*ut_date=None*, *local_date=None*, *force=False*, ***kwargs*)

Updates the date of the observatory, and re-process the observatory parameters if the date is different.

Args:

- ut_date (see below): the date of observation in UT time
- local_date (see below): the date of observation in local time
- force (bool): if `False`, the observatory is re-processed only if the date changed

Kwargs: See class constructor

Raises:

- KeyError: if the twilight keyword is unknown

- Exception: if the observatory object has no date

Returns: `True` if the date was changed, otherwise `False`

Note:

- `local_date` and `ut_date` can be date-tuples `(yyyy, mm, dd, [hh, mm, ss])`, timestamps, datetime structures or ephem.Date instances.
- If both are given, `ut_date` has higher priority
- If neither of those are given, the date is automatically set to *tonight* or *now* (whether the sun has already set or not)

1.5 astroobs.ObservatoryList module

class `astroobs.ObservatoryList.`**`ObservatoryList`**(*dataFile=None*, ***kwargs*)

Bases: `object`

Manages the database of observatories.

Args:

- dataFile (str): path+file to the observatories database. If left to `None`, the standard package database will be used

Kwargs:

- raiseError (bool): if `True`, errors will be raised; if `False`, they will be printed. Default is `False`

Raises:

- Exception: if a mandatory input parameter is missing when loading all observatories

Use `add()`, `rem()`, `mod()` to add, remove or modify an observatory to the database.

```
>>> import astroobs.obs as obs
>>> ol = obs.ObservatoryList()
>>> ol
List of 34 observatories
>>> ol.obsids
['mwo',
 'kpno',
 'ctio',
 'lasilla',
 ...
 'vlt',
 'mgo',
 'ohp']
>>> ol['ohp']
{'elevation': 650.0,
 'lat': 0.7667376848115423,
 'long': 0.09971647793060935,
 'moonAvoidRadius': 0.25,
 'name': 'Observatoire de Haute Provence',
 'pressure': 1010.0,
 'temp': 15.0,
 'timezone': 'Europe/Paris'}
```

add(*obsid*, *name*, *long*, *lat*, *elevation*, *timezone*, *temp=15.0*, *pressure=1010.0*, *moonAvoidRadius=0.25*, ***kwargs*)
Adds an observatory to the current observatories database.

Args:

- obsid (str): id of the observatory to add. Must be unique, without spaces or ;
- name (str): name of the observatory
- long (str - '+/-ddd:mm:ss.s'): longitude of the observatory. West is negative, East is positive
- lat (str - '+/-dd:mm:ss.s'): latitude of the observatory. North is Positive, South is negative
- elevation (float - m): elevation of the observatory
- timezone (str): timezone of the observatory, as in pytz library. See note below
- temp (float - degrees Celcius) [optional]: temperature at the observatory
- pressure (float - hPa) [optional]: pressure at the observatory
- moonAvoidRadius (float - degrees) [optional]: minimum distance at which a target must sit from the moon to be observed

Kwargs: See class constructor

Raises:

- NameError: if the observatory ID already exists
- Exception: if a mandatory input parameter is missing when reloading all observatories

---

Note: To view all available timezones, run: >>> import pytz >>> for tz in pytz.all_timezones: >>> print(tz)

---

mod(*obsid*, *name*, *long*, *lat*, *elevation*, *timezone*, *temp=15.0*, *pressure=1010.0*, *moonAvoidRadius=0.25*, ***kwargs*)
Modifies an observatory in the current observatories database.

Args:

- obsid (str): id of the observatory to modify. All other parameters redefine the observatory

Kwargs: See class constructor

Raises:

- NameError: if the observatory ID does not exist
- Exception: if a mandatory input parameter is missing when reloading all observatories

---

Note: Refer to `add()` for details on input parameters

---

nameList()
Provides a list of tuples (obs id, observatory name) in the alphabetical order of the column 'observatory name'.

rem(*obsid*, ***kwargs*)
Removes an observatory from the current observatories database.

Args:

- obsid (str): id of the observatory to remove

Kwargs: See class constructor

Raises:

- NameError: if the observatory ID does not exist
- Exception: if a mandatory input parameter is missing when reloading all observatories

`astroobs.ObservatoryList.`**`showall`**(*dataFile=None*, ***kwargs*)
A quick function to view all available observatories

1.6 astroobs.Target module

class `astroobs.Target.`**`Target`**(*ra*, *dec*, *name*, *input_epoch='2000'*, *obs=None*, ***kwargs*)
Bases: `object`

Initialises a target object from its right ascension and declination. Optionaly, processes the target for the observatory and date given (refer to `Target.process()`).

Args:

- ra (str 'hh:mm:ss.s' or float - degrees): the right ascension of the target
- dec (str '+/-dd:mm:ss.s' or float - degrees): the declination of the target
- name (str): the name of the target, for display
- obs (`Observatory`) [optional]: the observatory for which to process the target
- input_epoch (str): the 'YYYY' year of epoch in which the ra-dec coordinates are given. These coordinates will corrected with precession if the epoch of observatory is different

Kwargs:

- raiseError (bool): if `True`, errors will be raised; if `False`, they will be printed. Default is `False`

Raises: N/A

`dec`
The declination of the target, displayed as tuple (+/-dd, mm, ss)

`decStr`
A pretty printable version of the declination of the target

`plot`(*obs*, *y='alt'*, ***kwargs*)
Plots the y-parameter vs time diagram for the target at the given observatory and date

Args:

- obs (`Observatory`): the observatory for which to plot the target
- y (object attribute): the y-data to plot

Kwargs:

- See class constructor
- See `Observatory.plot()`
- simpleplt (bool): if `True`, the observatory plot will not be plotted, default is `False`
- color (str or #XXXXXX): the color of the target curve, default is 'k'
- lw (float): the linewidth, default is 1

Raises: N/A

polar(*obs*, ***kwargs*)
Plots the sky-view diagram for the target at the given observatory and date

Args:

- obs (`Observatory`): the observatory for which to plot the target
- y (object attribute): the y-data to plot

Kwargs:

- See class constructor
- See `Observatory.plot()`
- See `Target.plot()`

Raises: N/A

process(*obs*, ***kwargs*)
Processes the target for the given observatory and date.

Args:

- obs (`Observatory`): the observatory for which to process the target

Kwargs: See class constructor

Raises: N/A

Creates vector attributes:

- `airmass`: the airmass of the target
- `ha`: the hour angle of the target (degrees)
- `alt`: the altitude of the target (degrees - horizon is 0)
- `az`: the azimuth of the target (degrees)
- `moondist`: the angular distance between the moon and the target (degrees)

Note:

•All previous attributes are vectors related to the time vector of the observatory used for processing, stored under `dates` attribute

---

Other attributes:

- `rise_time`, `rise_az`: the time (ephem.Date) and the azimuth (degree) of the rise of the target
- `set_time`, `set_az`: the time (ephem.Date) and the azimuth (degree) of the setting of the target
- `transit_time`, `transit_az`: the time (ephem.Date) and the azimuth (degree) of the transit of the target

Warning:

•it can occur that the target does not rise or set for an observatory/date combination. In that case, the corresponding attributes will be set to `None`, i.e. `set_time`, `set_az`, `rise_time`, `rise_az`. In that case, an additional parameter is added to the Target object: `Target.alwaysUp` which is `True` if the target never sets and `False` if it never rises above the horizon.

ra
The right ascension of the target, displayed as tuple (hh, mm, ss)

raStr
A pretty printable version of the right ascension of the target

whenobs(*obs*, *fromDate='now'*, *toDate='now+30day'*, *plot=True*, *ret=False*, *dday=1*, ***kwargs*)
Processes the target for the given observatory and dat.

Args:

- obs (`Observatory`): the observatory for which to process the target
- fromDate (see below): the start date of the range
- toDate (see below): the end date of the range
- plot: whether it plots the diagram
- ret: whether it returns the values
- dday: the

Kwargs: See class constructor * legend (bool): whether to add a legend or not, default is `True` * loc: location of the legend, default is 8 (top right), refer to plt.legend * ncol: number of columns in the legend, default is 3, refer to plt.legend * columnspacing: spacing between columns in the legend, refer to plt.legend * lfs: legend font size, default is 11

Raises: N/A

Note:

•`local_date` and `ut_date` can be date-tuples `(yyyy, mm, dd, [hh, mm, ss])`, timestamps, datetime structures or ephem.Date instances.

1.7 astroobs.TargetSIMBAD module

class `astroobs.TargetSIMBAD.`**TargetSIMBAD**(*name*, *obs=None*, *input_epoch='2000'*, ***kwargs*)
Bases: `astroobs.Target.Target`

Initialises a target object from an online SIMBAD database name-search. Optionaly, processes the target for the observatory and date given (refer to `TargetSIMBAD.process()`).

Args:

- name (str): the name of the target as if performing an online SIMBAD search
- obs (`Observatory`) [optional]: the observatory for which to process the target

Kwargs:

- raiseError (bool): if `True`, errors will be raised; if `False`, they will be printed. Default is `False`

Raises: N/A

Creates attributes:

- `flux`: a dictionary of the magnitudes of the target. Keys are part or all of ['U','B','V','R','I','J','H','K']
- `link`: the link to paste into a web-browser to display the SIMBAD page of the target

- `linkbib`: the link to paste into a web-browser to display the references on the SIMBAD page of the target
- `hd`: if applicable, the HD number of the target
- `hr`: if applicable, the HR number of the target
- `hip`: if applicable, the HIP number of the target

1.8 astroobs.astroobsexception module

exception `astroobs.astroobsexception.`**`AstroobsException`**
Bases: `exceptions.Exception`

Root for astroobs Exceptions, only used to except any astroobs error, never raised

exception `astroobs.astroobsexception.`**`DuplicateObservatory`**(*key=''*, **args*)
Bases: `astroobs.astroobsexception.AstroobsException`

If the observatory key is already existing

exception `astroobs.astroobsexception.`**`InputNotUnderstood`**(*ipt=''*, **args*)
Bases: `astroobs.astroobsexception.AstroobsException`

If the input was not understood

exception `astroobs.astroobsexception.`**`NoObservatoryDate`**(**args*)
Bases: `astroobs.astroobsexception.UncompleteObservatory`

If the observatory is missing a date

exception `astroobs.astroobsexception.`**`NoPlotMode`**(**args*)
Bases: `astroobs.astroobsexception.AstroobsException`

If the user doesn't have matplotlib

exception `astroobs.astroobsexception.`**`NonObservatory`**(*obs=''*, **args*)
Bases: `astroobs.astroobsexception.AstroobsException`

If one or more parameter is missing in the setting up of the Obervatory object

exception `astroobs.astroobsexception.`**`NonObservatoryList`**(**args*)
Bases: `astroobs.astroobsexception.AstroobsException`

If the observatory list is not valid

exception `astroobs.astroobsexception.`**`NonTarget`**(*obj=''*, **args*)
Bases: `astroobs.astroobsexception.AstroobsException`

If the type of the object is not astroobs.Target, or is not valid

exception `astroobs.astroobsexception.`**`ReadOnly`**(*attr=''*, **args*)
Bases: `astroobs.astroobsexception.AstroobsException`

If the parameter is read-only

exception `astroobs.astroobsexception.`**`TargetMissingSIMBAD`**(*target=''*, **args*)
Bases: `astroobs.astroobsexception.AstroobsException`

If the target name given was not found in SIMBAD

exception astroobs.astroobsexception.**UncompleteObservatory**(*param=''*, **args*)
Bases: astroobs.astroobsexception.AstroobsException

If one or more parameter is missing in the setting up of the Obervatory object

exception astroobs.astroobsexception.**UnknownObservatory**(*obs=''*, **args*)
Bases: astroobs.astroobsexception.AstroobsException

If the observatory key is not known

exception astroobs.astroobsexception.**UnknownTwilight**(*twi=''*, **args*)
Bases: astroobs.astroobsexception.AstroobsException

If the twilight key is not known

astroobs.astroobsexception.**raiseIt**(*exc*, *raiseoupas*, **args*)

1.9 astroobs.core module

astroobs.core.**airmass_to_rad**(*arr*)
Transforms airmass to radians

astroobs.core.**cleanTime**(*t*, *format=None*)
Raises an error if t not among (ephem.Date, datetime, timestamp, tuple, time.struct_time) date types, and optionaly returns the date into the format: - 'ts': unix timestamp (float) - 'dt': datetime - 'du': date tuple - 'ed': ephem.Date - 'st': time.struct_time

NB: does not keep the tzinfo of datetime

astroobs.core.**convertTime**(*t*, *tzTo*, *tzFrom='utc'*, *format=None*)
Converts the time 't' from timezone 'tzFrom' (default is UT) to timezone 'tzTo'.

tzFrom and tzTo are like 'America/Los_Angeles'

cf cleanTime method to see possible types for 't' and output.

astroobs.core.**make_num**(*numstr*)
Removes any non-number character from numstr. Keeps also decimal separator "." and signs "-", "+". Returns float

astroobs.core.**rad_to_airmass**(*arr*)
Transforms radians to airmass

astroobs.core.**radecFromStr**(*txt*)
Takes a string that contains ra in decimal degrees or in hh:mm:ss.s and dec in decimal degrees or dd:mm:ss.s returns (ra, dec) in decimal degrees

1.10 astroobs.obs module

1.11 astroobs.version module

1.12 Module contents

Provides astronomy ephemeris to plan telescope observations

Note:

- All altitudes, azimuth, hour angle are in degrees
- `horizon` attribute of `Observatory` or `Observation` is in radian
- All times are in UT, except for `Observatory.localnight`

Warning:
- it can occur that the Sun, the Moon or a target does not rise or set for an observatory/date combination. In that case, the corresponding attributes will be set to `None`

Real-life example use: >>>

class `astroobs.`**`ObservatoryList`**(*dataFile=None*, ***kwargs*)
Bases: `object`

Manages the database of observatories.

Args:

- dataFile (str): path+file to the observatories database. If left to `None`, the standard package database will be used

Kwargs:

- raiseError (bool): if `True`, errors will be raised; if `False`, they will be printed. Default is `False`

Raises:

- Exception: if a mandatory input parameter is missing when loading all observatories

Use `add()`, `rem()`, `mod()` to add, remove or modify an observatory to the database.

```
>>> import astroobs.obs as obs
>>> ol = obs.ObservatoryList()
>>> ol
List of 34 observatories
>>> ol.obsids
['mwo',
 'kpno',
 'ctio',
 'lasilla',
 ...
 'vlt',
 'mgo',
 'ohp']
>>> ol['ohp']
{'elevation': 650.0,
 'lat': 0.7667376848115423,
 'long': 0.09971647793060935,
 'moonAvoidRadius': 0.25,
 'name': 'Observatoire de Haute Provence',
 'pressure': 1010.0,
 'temp': 15.0,
 'timezone': 'Europe/Paris'}
```

add(*obsid*, *name*, *long*, *lat*, *elevation*, *timezone*, *temp=15.0*, *pressure=1010.0*, *moonAvoidRadius=0.25*, ***kwargs*)
Adds an observatory to the current observatories database.

Args:

- obsid (str): id of the observatory to add. Must be unique, without spaces or ;

- name (str): name of the observatory
- long (str - '+/-ddd:mm:ss.s'): longitude of the observatory. West is negative, East is positive
- lat (str - '+/-dd:mm:ss.s'): latitude of the observatory. North is Positive, South is negative
- elevation (float - m): elevation of the observatory
- timezone (str): timezone of the observatory, as in pytz library. See note below
- temp (float - degrees Celcius) [optional]: temperature at the observatory
- pressure (float - hPa) [optional]: pressure at the observatory
- moonAvoidRadius (float - degrees) [optional]: minimum distance at which a target must sit from the moon to be observed

Kwargs: See class constructor

Raises:

- NameError: if the observatory ID already exists
- Exception: if a mandatory input parameter is missing when reloading all observatories

---

Note: To view all available timezones, run: >>> import pytz >>> for tz in pytz.all_timezones: >>> print(tz)

---

mod(*obsid*, *name*, *long*, *lat*, *elevation*, *timezone*, *temp=15.0*, *pressure=1010.0*, *moonAvoidRadius=0.25*, ***kwargs*)
Modifies an observatory in the current observatories database.

Args:

- obsid (str): id of the observatory to modify. All other parameters redefine the observatory

Kwargs: See class constructor

Raises:

- NameError: if the observatory ID does not exist
- Exception: if a mandatory input parameter is missing when reloading all observatories

---

Note: Refer to `add()` for details on input parameters

---

nameList()
Provides a list of tuples (obs id, observatory name) in the alphabetical order of the column 'observatory name'.

rem(*obsid*, ***kwargs*)
Removes an observatory from the current observatories database.

Args:

- obsid (str): id of the observatory to remove

Kwargs: See class constructor

Raises:

- NameError: if the observatory ID does not exist
- Exception: if a mandatory input parameter is missing when reloading all observatories

class astroobs.**Observatory**(*obs*, *long=None*, *lat=None*, *elevation=None*, *timezone=None*, *temp=None*, *pressure=None*, *moonAvoidRadius=None*, *local_date=None*, *ut_date=None*, *horizon_obs=None*, *dataFile=None*, *epoch='2000'*, ***kwargs*)

Bases: ephem.Observer, object

Defines an observatory from which the ephemeris of the twilights or a night-sky target are processed. The *night-time* is base on the given date. It ends at the next sunrise and starts at the sunset preceeding this next sunrise.

Args:

- obs (str): id of the observatory to pick from the observatories database OR the name of the custom observatory (in that case, long, lat, elevation, timezone must also be given, temp, pressure, moonAvoidRadius are optional)
- local_date (see below): the date of observation in local time
- ut_date (see below): the date of observation in UT time
- horizon_obs (float - degrees): minimum altitude at which a target can be observed, default is 30 degrees altitude
- epoch (str): the 'YYYY' year in which all ra-dec coordinates are converted

Kwargs:

- raiseError (bool): if True, errors will be raised; if False, they will be printed. Default is False
- fig: TBD

Raises:

- NameError: if a mandatory input parameter is missing
- KeyError: if the observatory ID does not exist
- KeyError: if the twilight keyword is unknown
- Exception: if the observatory object has no date

Note:

•For details on local_date and ut_date, refer to Observatory.upd_date()

•For details on other input parameters, refer to ObservatoryList.add()

•The Observatory automatically creates and manages a Moon target under moon attribute

•If obs is the id of an observatory to pick in the database, the user can still provide temp, pressure, moonAvoidRadius attributes which will override the database default values

•horizon attribute is in radian

Main attributes:

- localnight: gives the local midnight time in local time (YYYY, MM, DD, 23, 59, 59)
- date: gives the local midnight time in UT time
- dates: is a vector of Dublin Julian Dates. Refer to process_obs()
- lst: the local sidereal time corresponding to each dates element
- localTimeOffest: gives the shift in days between UT and local time: local=UT+localTimeOffest

- `moon`: points to the `Moon` target processed for the given observatory and date

Twilight attributes:

- For the next three attributes, `XXX` shall be replaced by {'' (blank), 'civil', 'nautical', 'astro'} for, respectively, horizon, -6, -12, and -18 degrees altitude
- `sunriseXXX`: gives the sunrise time for different twilights, in Dublin Julian Dates. e.g.: `observatory.sunrise`
- `sunsetXXX`: gives the sunset time for different twilights, in Dublin Julian Dates. e.g.: `observatory.sunsetcivil`
- `len_nightXXX`: gives the night duration for different twilights (between corresponding sunset and sunrise), in hours. e.g.: `observatory.len_nightnautical`

Warning:
•it can occur that the Sun, the Moon or a target does not rise or set for an observatory/date combination. In that case, the corresponding attributes will be set to `None`

```
>>> import astroobs.obs as obs
>>> o = obs.Observatory('ohp', local_date=(2015,3,31,23,59,59))
>>> o
<ephem.Observer date='2015/3/31 21:59:59' epoch='2000/1/1 12:00:00'
lon=5:42:48.0 lat=43:55:51.0 elevation=650.0m horizon=-0:49:04.8
temp=15.0C pressure=1010.0mBar>
>>> o.moon
Moon - phase: 89.2%
>>> print(o.sunset, '...', o.sunrise, '...', o.len_night)
2015/3/31 18:08:40 ... 2015/4/1 05:13:09 ... 11.0746939826
>>> import ephem as E
>>> print(E.Date(o.sunsetastro+o.localTimeOffest), '...', E.Date(
        o.sunriseastro+o.localTimeOffest), '...', o.len_nightastro)
2015/3/31 21:43:28 ... 2015/4/1 05:38:26 ... 7.91603336949
```

nowArg
Returns the index of *now* in the `observatory.dates` vector, or None if *now* is out of its bounds (meaning the observation is not taking place now)

```
>>> import astroobs.obs as obs
>>> import ephem as E
>>> o = obs.Observatory('ohp')
>>> plt.plot(o.dates, o.moon.alt, 'k-')
>>> now = o.nowArg
>>> if now is not None:
>>>     plt.plot(o.dates[now], o.moon.alt[now], 'ro')
>>> else:
>>>     plt.plot([E.now(), E.now()], [o.moon.alt.min(),o.moon.alt.max()], 'r--')
```

plot (***kwargs*)
Plots the observatory diagram

Kwargs:

- See class constructor
- dt (float - hour): the spacing of x-axis labels, default is 1 hour (not with polar mode)
- t0 (float - DJD or [0-24]): the date of the first tick-label of x-axis, default is sunsetastro. The time type must correspond to `time` parameter (not with polar mode)

- xlim ([xmin, xmax]): bounds for x-axis, default is full night span (not with polar mode)
- retxdisp (bool): if `True`, bounds of x-axis displayed values are returned (`xdisp` key)
- ylim ([ymin, ymax]): bounds for y-axis, default is [horizon_obs-10, 90] (not with polar mode)
- xlabel (str): label for x-axis, default 'Time (UT)'
- ylabel (str): label for y-axis, default 'Elevation (°)'
- title (str): title of the diagram, default is observatory name or coordinates
- ymin_margin (float): margin between xmin of graph and horizon_obs. Low priority vs ylim, default is 10 (not with polar mode)
- retfignum (bool): if `True`, the figure number will be returned, default is `False`
- fignum (int): figure number on which to plot, default is `False`
- retaxnum (bool): if `True`, the ax index as in `figure.axes[n]` will be returned, default is `False`
- axnum (int): axes index on which to plot, default is `None` (create new ax)
- retfig (bool): if `True`, the figure object will be returned, default is `False`
- fig (figure): figure object on which to plot, default is `None` (use fignum)
- retax (bool): if `True`, the ax will be returned, default is `False`
- ax (axes): ax on which to plot, default is `None`
- now (bool): if `True` and within range, a vertical line as indication of "now" will be shown, default is True
- retnow (bool): returns the line object (`nowline` key) corresponding to the 'now-line', default is `False`
- legend (bool): whether to add a legend or not, default is `True`
- loc: location of the legend, default is 8 (top right), refer to plt.legend
- ncol: number of columns in the legend, default is 3, refer to plt.legend
- columnspacing: spacing between columns in the legend, refer to plt.legend
- lfs: legend font size, default is 11
- textlbl (bool): if `True`, a text label with target name or coordinates will be added near transit, default is `False`
- polar (bool): if `True`, plots the sky view, otherwise plots target attribute versus time
- time (str): the type of the x-axis time, `ut` for UT, `loc` for local time and `lst` [0-24] for local sidereal time, default is `ut` (not with polar mode)

Raises: N/A

process_obs (*pts=200*, *margin=15*, *fullhour=False*, ***kwargs*)

Processes all twilights as well as moon rise, set and position through night for the given observatory and date. Creates the vector `observatory.dates` which is the vector containing all timestamps at which the moon and the targets will be processed.

Args:

- pts (int) [optional]: the size of the `dates` vector, whose elements are linearly spaced in time

- margin (float - minutes) [optional]: the margin between the first element of the vector `dates` and the sunset, and between the sunrise and its last element
- fullhour (bool) [optional]: if `True`, then the vector `dates` will start and finish on the first full hour preceeding sunset and following sunrise

Kwargs: See class constructor

Raises:

- KeyError: if the twilight keyword is unknown
- Exception: if the observatory object has no date

Note: In case the observatory is in polar regions where the sun does not alway set and rise everyday, the first and last elements of the `dates` vector are set to local midday right before and after the local midnight of the observation date. e.g.: 24h night centered on the local midnight.

---

upd_date(*ut_date=None*, *local_date=None*, *force=False*, ***kwargs*)

Updates the date of the observatory, and re-process the observatory parameters if the date is different.

Args:

- ut_date (see below): the date of observation in UT time
- local_date (see below): the date of observation in local time
- force (bool): if `False`, the observatory is re-processed only if the date changed

Kwargs: See class constructor

Raises:

- KeyError: if the twilight keyword is unknown
- Exception: if the observatory object has no date

Returns: `True` if the date was changed, otherwise `False`

Note:

- `local_date` and `ut_date` can be date-tuples `(yyyy, mm, dd, [hh, mm, ss])`, timestamps, datetime structures or ephem.Date instances.
- If both are given, `ut_date` has higher priority
- If neither of those are given, the date is automatically set to *tonight* or *now* (whether the sun has already set or not)

---

class `astroobs.`**Target**(*ra*, *dec*, *name*, *input_epoch='2000'*, *obs=None*, ***kwargs*)

Bases: `object`

Initialises a target object from its right ascension and declination. Optionaly, processes the target for the observatory and date given (refer to `Target.process()`).

Args:

- ra (str 'hh:mm:ss.s' or float - degrees): the right ascension of the target
- dec (str '+/-dd:mm:ss.s' or float - degrees): the declination of the target
- name (str): the name of the target, for display
- obs (`Observatory`) [optional]: the observatory for which to process the target

- input_epoch (str): the 'YYYY' year of epoch in which the ra-dec coordinates are given. These coordinates will corrected with precession if the epoch of observatory is different

Kwargs:

- raiseError (bool): if `True`, errors will be raised; if `False`, they will be printed. Default is `False`

Raises: N/A

dec
The declination of the target, displayed as tuple (+/-dd, mm, ss)

decStr
A pretty printable version of the declination of the target

plot(*obs*, *y='alt'*, ***kwargs*)
Plots the y-parameter vs time diagram for the target at the given observatory and date

Args:

- obs (`Observatory`): the observatory for which to plot the target
- y (object attribute): the y-data to plot

Kwargs:

- See class constructor
- See `Observatory.plot()`
- simpleplt (bool): if `True`, the observatory plot will not be plotted, default is `False`
- color (str or #XXXXXX): the color of the target curve, default is 'k'
- lw (float): the linewidth, default is 1

Raises: N/A

polar(*obs*, ***kwargs*)
Plots the sky-view diagram for the target at the given observatory and date

Args:

- obs (`Observatory`): the observatory for which to plot the target
- y (object attribute): the y-data to plot

Kwargs:

- See class constructor
- See `Observatory.plot()`
- See `Target.plot()`

Raises: N/A

process(*obs*, ***kwargs*)
Processes the target for the given observatory and date.

Args:

- obs (`Observatory`): the observatory for which to process the target

Kwargs: See class constructor

Raises: N/A

Creates vector attributes:

- `airmass`: the airmass of the target
- `ha`: the hour angle of the target (degrees)
- `alt`: the altitude of the target (degrees - horizon is 0)
- `az`: the azimuth of the target (degrees)
- `moondist`: the angular distance between the moon and the target (degrees)

Note:

•All previous attributes are vectors related to the time vector of the observatory used for processing, stored under `dates` attribute

---

Other attributes:

- `rise_time`, `rise_az`: the time (ephem.Date) and the azimuth (degree) of the rise of the target
- `set_time`, `set_az`: the time (ephem.Date) and the azimuth (degree) of the setting of the target
- `transit_time`, `transit_az`: the time (ephem.Date) and the azimuth (degree) of the transit of the target

Warning:

•it can occur that the target does not rise or set for an observatory/date combination. In that case, the corresponding attributes will be set to `None`, i.e. `set_time`, `set_az`, `rise_time`, `rise_az`. In that case, an additional parameter is added to the Target object: `Target.alwaysUp` which is `True` if the target never sets and `False` if it never rises above the horizon.

ra

The right ascension of the target, displayed as tuple (hh, mm, ss)

raStr

A pretty printable version of the right ascension of the target

whenobs(*obs*, *fromDate='now'*, *toDate='now+30day'*, *plot=True*, *ret=False*, *dday=1*, ***kwargs*)

Processes the target for the given observatory and dat.

Args:

- obs (`Observatory`): the observatory for which to process the target
- fromDate (see below): the start date of the range
- toDate (see below): the end date of the range
- plot: whether it plots the diagram
- ret: whether it returns the values
- dday: the

Kwargs: See class constructor * legend (bool): whether to add a legend or not, default is `True` * loc: location of the legend, default is 8 (top right), refer to plt.legend * ncol: number of columns in the legend, default is 3, refer to plt.legend * columnspacing: spacing between columns in the legend, refer to plt.legend * lfs: legend font size, default is 11

Raises: N/A

Note:

- `local_date` and `ut_date` can be date-tuples `(yyyy, mm, dd, [hh, mm, ss])`, timestamps, datetime structures or ephem.Date instances.

---

class `astroobs.`**`Moon`**(*obs=None*, *input_epoch='2000'*, ***kwargs*)

Bases: `astroobs.Target.Target`

Initialises the Moon. Optionaly, processes the Moon for the observatory and date given (refer to `Moon.process()`).

Args:

- obs (`Observatory`) [optional]: the observatory for which to process the Moon

Kwargs:

- raiseError (bool): if `True`, errors will be raised; if `False`, they will be printed. Default is `False`

Raises: N/A

`dec`

The declination of the Moon, displayed as tuple of np.array (+/-dd, mm, ss)

`decStr`

A pretty printable version of the mean of the declination of the moon

`plot`(*obs*, *y='alt'*, ***kwargs*)

Plots the y-parameter vs time diagram for the moon at the given observatory and date

Args:

- obs (`Observatory`): the observatory for which to plot the moon

Kwargs:

- See class constructor
- See `Observatory.plot()`
- See `Target.plot()`

Raises: N/A

`polar`(*obs*, ***kwargs*)

Plots the y-parameter vs time diagram for the moon at the given observatory and date

Args:

- obs (`Observatory`): the observatory for which to plot the moon

Kwargs:

- See class constructor
- See `Observatory.plot()`
- See `Target.plot()`

Raises: N/A

`process`(*obs*, ***kwargs*)

Processes the moon for the given observatory and date.

Args:

- obs (`Observatory`): the observatory for which to process the moon

Kwargs: See class constructor

Raises: N/A

Creates vector attributes:

- `airmass`: the airmass of the moon
- `ha`: the hour angle of the moon (degrees)
- `alt`: the altitude of the moon (degrees - horizon is 0)
- `az`: the azimuth of the moon (degrees)
- `ra`: the right ascension of the moon, see `Moon.ra()`
- `dec`: the declination of the moon, see `Moon.dec()`

Note:

•All previous attributes are vectors related to the time vector of the observatory used for processing: `obs.dates`

Other attributes:

- `rise_time`, `rise_az`: the time (ephem.Date) and the azimuth (degree) of the rise of the moon
- `set_time`, `set_az`: the time (ephem.Date) and the azimuth (degree) of the setting of the moon
- `transit_time`, `transit_az`: the time (ephem.Date) and the azimuth (degree) of the transit of the moon

Warning:

•it can occur that the moon does not rise or set for an observatory/date combination. In that case, the corresponding attributes will be set to `None`, i.e. `set_time`, `set_az`, `rise_time`, `rise_az`. In that case, an additional parameter is added to the Moon object: `Moon.alwaysUp` which is `True` if the Moon never sets and `False` if it never rises above the horizon.

ra

The right ascension of the Moon, displayed as tuple of np.array (hh, mm, ss)

raStr

A pretty printable version of the mean of the right ascension of the moon

class `astroobs.`**`TargetSIMBAD`**(*name*, *obs=None*, *input_epoch='2000'*, ***kwargs*)

Bases: `astroobs.Target.Target`

Initialises a target object from an online SIMBAD database name-search. Optionaly, processes the target for the observatory and date given (refer to `TargetSIMBAD.process()`).

Args:

- name (str): the name of the target as if performing an online SIMBAD search
- obs (`Observatory`) [optional]: the observatory for which to process the target

Kwargs:

- raiseError (bool): if `True`, errors will be raised; if `False`, they will be printed. Default is `False`

Raises: N/A

Creates attributes:

- `flux`: a dictionary of the magnitudes of the target. Keys are part or all of ['U','B','V','R','I','J','H','K']

- `link`: the link to paste into a web-browser to display the SIMBAD page of the target
- `linkbib`: the link to paste into a web-browser to display the references on the SIMBAD page of the target
- `hd`: if applicable, the HD number of the target
- `hr`: if applicable, the HR number of the target
- `hip`: if applicable, the HIP number of the target

class `astroobs.`**`Observation`**(*obs*, *long=None*, *lat=None*, *elevation=None*, *timezone=None*, *temp=None*, *pressure=None*, *moonAvoidRadius=None*, *local_date=None*, *ut_date=None*, *horizon_obs=None*, *dataFile=None*, *epoch='2000'*, ***kwargs*)

Bases: `astroobs.Observatory.Observatory`

Assembles together an `Observatory` (including itself the `Moon` target), and a list of `Target`.

For use and docs refer to:

- `add_target()` to add a target to the list
- `rem_target()` to remove one
- `change_obs()` to change the observatory
- `change_date()` to change the date of observation

Kwargs:

- raiseError (bool): if `True`, errors will be raised; if `False`, they will be printed. Default is `False`
- fig: TBD

Raises: See `Observatory`

Warning:
•it can occur that the Sun, the Moon or a target does not rise or set for an observatory/date combination. In that case, the corresponding attributes will be set to `None`

```
>>> import astroobs.obs as obs
>>> o = obs.Observation('ohp', local_date=(2015,3,31,23,59,59))
>>> o
Observation at Observatoire de Haute Provence on 2015/6/21-22. 0 targets.
    Moon phase: 89.2%
>>> o.moon
Moon - phase: 89.2%
>>> print o.sunset, '...', o.sunrise, '...', o.len_night
2015/3/31 18:08:40 ... 2015/4/1 05:13:09 ... 11.0746939826
>>> import ephem as E
>>> print(E.Date(o.sunsetastro+o.localTimeOffest), '...', E.Date(
        o.sunriseastro+o.localTimeOffest), '...', o.len_nightastro)
2015/3/31 21:43:28 ... 2015/4/1 05:38:26 ... 7.91603336949
>>> o.add_target('vega')
>>> o.add_target('mystar', dec=19.1824, ra=213.9153)
>>> o.targets
[Target: 'vega', 18h36m56.3s +38°35'8.1", O,
 Target: 'mystar', 14h15m39.7s +19°16'43.8", O]
>>> print("%s mags: 'K': %2.2f, 'R': %2.2f"%(o.targets[0].name,
        o.targets[0].flux['K'], o.targets[0].flux['R']))
vega mags: 'K': 0.13, 'R': 0.07
```

add_target(*tgt*, *ra=None*, *dec=None*, *name=''*, ***kwargs*)
Adds a target to the observation list

Args:

- tgt (see below): the index of the target in the `Observation.targets` list
- ra ('hh:mm:ss.s' or decimal degree) [optional]: the right ascension of the target to add to the observation list. See below
- dec ('+/-dd:mm:ss.s' or decimal degree) [optional]: the declination of the target to add to the observation list. See below
- name (string) [optional]: the name of the target to add to the observation list. See below

tgt arg can be:

- a `Target` instance: all other parameters are ignored
- a target name (string): if `ra` and `dec` are not `None`, the target is added with the provided coordinates; if `None`, a SIMBAD search is performed on `tgt`. `name` is ignored
- a ra-dec string ('hh:mm:ss.s +/-dd:mm:ss.s'): in that case, `ra` and `dec` will be ignored and `name` will be the name of the target

Kwargs: See class constructor

Raises:

- ValueError: if ra-dec formating was not understood

Note:

•Automatically processes the target for the given observatory and date

---

```
>>> import astroobs.obs as obs
>>> o = obs.Observation('ohp', local_date=(2015,3,31,23,59,59))
>>> arc = obs.TargetSIMBAD('arcturus')
>>> o.add_target(arc)
>>> o.add_target('arcturus')
>>> o.add_target('arcturusILoveYou', dec=19.1824, ra=213.9153)
>>> o.add_target('14:15:39.67 +10:10:56.67', name='arcturus')
>>> o.targets
[Target: 'arcturus', 14h15m39.7s +19°16'43.8", 0,
 Target: 'arcturus', 14h15m39.7s +19°16'43.8", 0,
 Target: 'arcturus', 14h15m39.7s +10°40'43.8", 0,
 Target: 'arcturus', 14h15m39.7s +19°16'43.8", 0]
```

change_date(*ut_date=None*, *local_date=None*, *recalcAll=False*, ***kwargs*)
Changes the date of the observation and optionaly re-processes targets for the same observatory and new date

Args:

- ut_date: Refer to `Observatory.upd_date()`
- local_date: Refer to `Observatory.upd_date()`
- recalcAll (bool or None) [optional]: if `False` (default): only targets selected for observation are re-processed, if `True`: all targets are re-processed, if `None`: no re-process

Kwargs: See class constructor

Raises:

- KeyError: if the twilight keyword is unknown
- Exception: if the observatory object has no date

change_obs(*obs*, *long=None*, *lat=None*, *elevation=None*, *timezone=None*, *temp=None*, *pressure=None*, *moonAvoidRadius=None*, *horizon_obs=None*, *dataFile=None*, *recalcAll=False*, ***kwargs*)

Changes the observatory and optionaly re-processes all target for the new observatory and same date

Args:

- recalcAll (bool or None) [optional]: if `False` (default): only targets selected for observation are re-processed, if `True`: all targets are re-processed, if `None`: no re-process

Kwargs: See class constructor

Note:

•Refer to `ObservatoryList.add()` for details on other input parameters

---

plot(*y='alt'*, ***kwargs*)

Plots the y-parameter vs time diagram for the target at the given observatory and date

Kwargs:

- See class constructor
- See `Observatory.plot()`
- moon (bool): if `True`, adds the moon to the graph, default is `True`
- autocolor (bool): if `True`, sets curves-colors automatically, default is `True`

Raises: N/A

polar(***kwargs*)

Plots the sky-view diagram for the target at the given observatory and date

Kwargs:

- See class constructor
- See `Observatory.plot()`
- moon (bool): if `True`, adds the moon to the graph, default is `True`
- autocolor (bool): if `True`, sets curves-colors automatically, default is `True`

Raises: N/A

rem_target(*tgt*, ***kwargs*)

Removes a target from the observation list

Args:

- tgt (int): the index of the target in the `Observation.targets` list

Kwargs: See class constructor

Raises: N/A

targets

Shows the list of targets recorded into the Observation

tick(*tgt*, *forceTo=None*, ***kwargs*)

Changes the ticked property of a target (whether it is selected for observation)

Args:

- tgt (int): the index of the target in the `Observation.targets` list
- forceTo (bool) [optional]: if `True`, selects the target for observation, if `False`, unselects it, if `None`, the value of the selection is inverted

Kwargs: See class constructor

Raises: N/A

Note:

•Automatically reprocesses the target for the given observatory and date if it is selected for observation

---

```
>>> import astroobs.obs as obs
>>> o = obs.Observation('ohp', local_date=(2015,3,31,23,59,59))
>>> o.add_target('arcturus')
>>> o.targets
[Target: 'arcturus', 14h15m39.7s +19°16'43.8", O]
>>> o.tick(4)
>>> o.targets
[Target: 'arcturus', 14h15m39.7s +19°16'43.8", -]
```

ticked

Shows whether the target was select for observation

CHAPTER 2

setup module

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

A

C

D

I

M

N

O

P

R

S

T

U

W