
POUET Documentation

Release 0.4

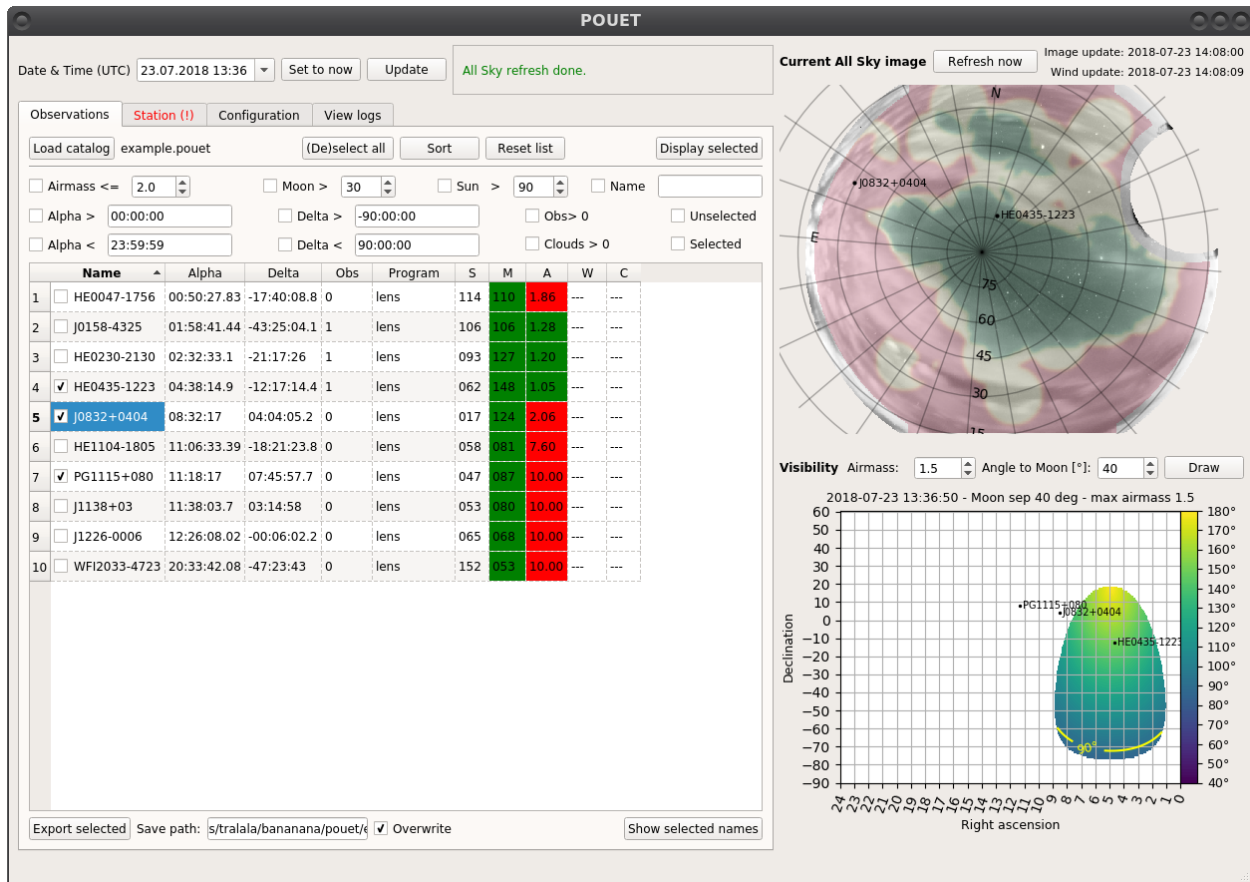
Vivien Bonvin, Thibault Kuntzer

Jan 12, 2020

Contents

| | |
|----------------------------|-----------|
| 1 Quick links | 3 |
| Python Module Index | 37 |
| Index | 39 |

POUET - Programing Observations Usefully at the Euler Telescope



This software helps you planning your observation nights at ANY telescope in the ESO-La Silla Observatory. But POULSO was a crappy software name, and both authors liked the Swiss Leonhard Euler Telescope a lot.

Why using this software and not one of the dozens of alternatives already existing out there? The strongest case is certainly that POUET provides a real-time analysis of the weather situation, including automatic clouds detection from the [Danish telescope AllSky Camera](#).

Designed by observers for observers, with lots of love and hopefully bug-free. Download POUET from [github](#).

1.1 Welcome to POUET's tutorial!

This section covers the basics of POUET usage. It is still under construction, but is progressing every day.

Note: POUET being still under development, some screenshots of this tutorial might not exactly correspond to what you see in your own session. If you feel something is missing or oddly explained, please [let us know](#).

1.1.1 Quick links

Set-up POUET

Installation...

...is currently not required. Simply [clone/download the repository](#) and move to its root.

Make sure you have all the requirements installed

```
pip install docs/requirements.txt
```

Note: Compatibility with older version of the required modules has not been assessed, but might be working. If you find any backward compatibility, please [let us know](#)

Tree structure

At the root of POUET, you will find the following directories

- `archives` contains old pieces of code (will disappear in a future version)

- `cats` contains the catalogs loaded/saved by POUET
- `docs` contains the documentation and requirement files
- `misc` contains stuff (will disappear in a future version)
- `pouet` contains the source code
- `standalone` contains standalone version of the plots (will disappear in a future version)
- `tests` contains a series of tests to ensure smooth continuous integration

We will describe these in due time.

Start-up

From the root directory:

```
python pouet/main.py
```

This will launch POUET in a dedicated window.

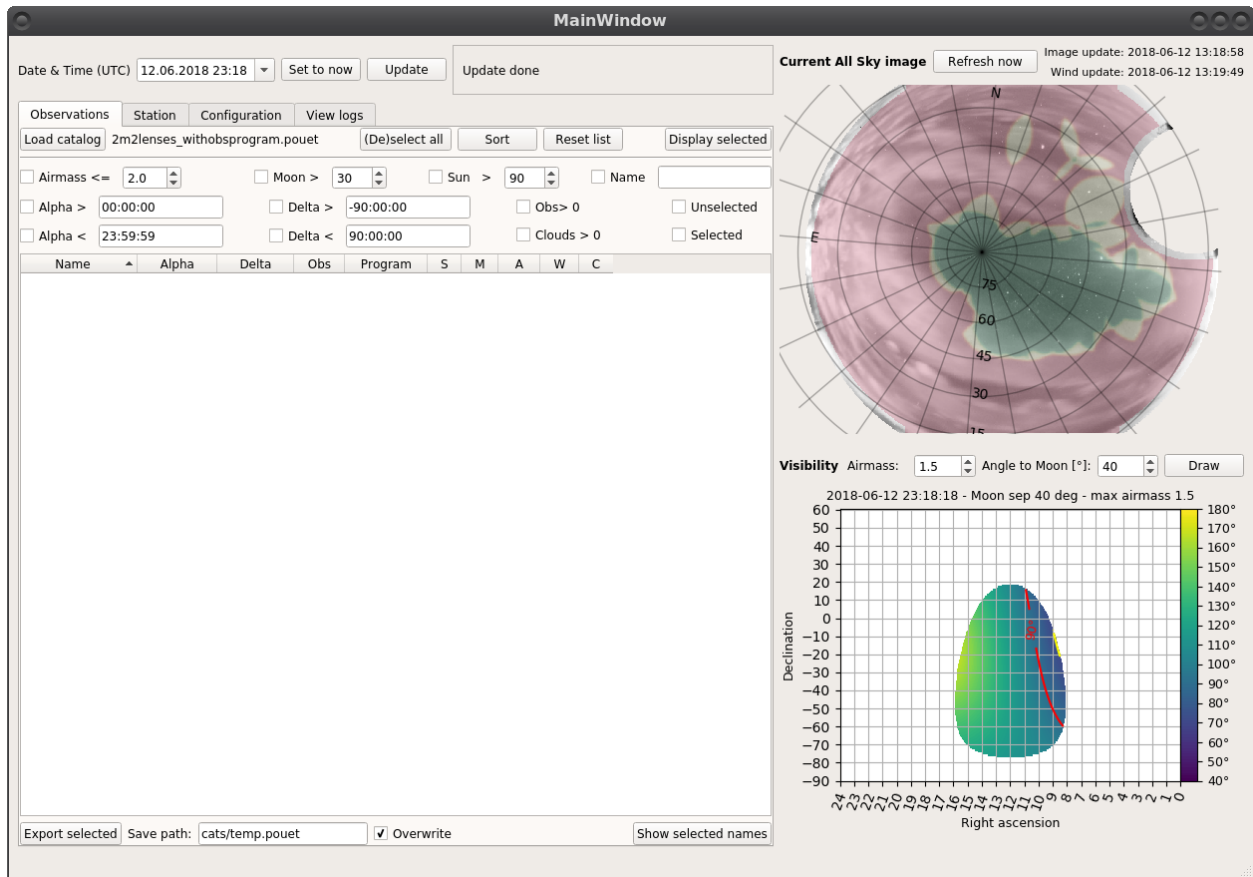


Fig. 1: A fresh POUET session.

Basic usage

Buttons, buttons everywhere

And they all serve a purpose. Let's have a look again at what a freshly opened session of POUET looks like:

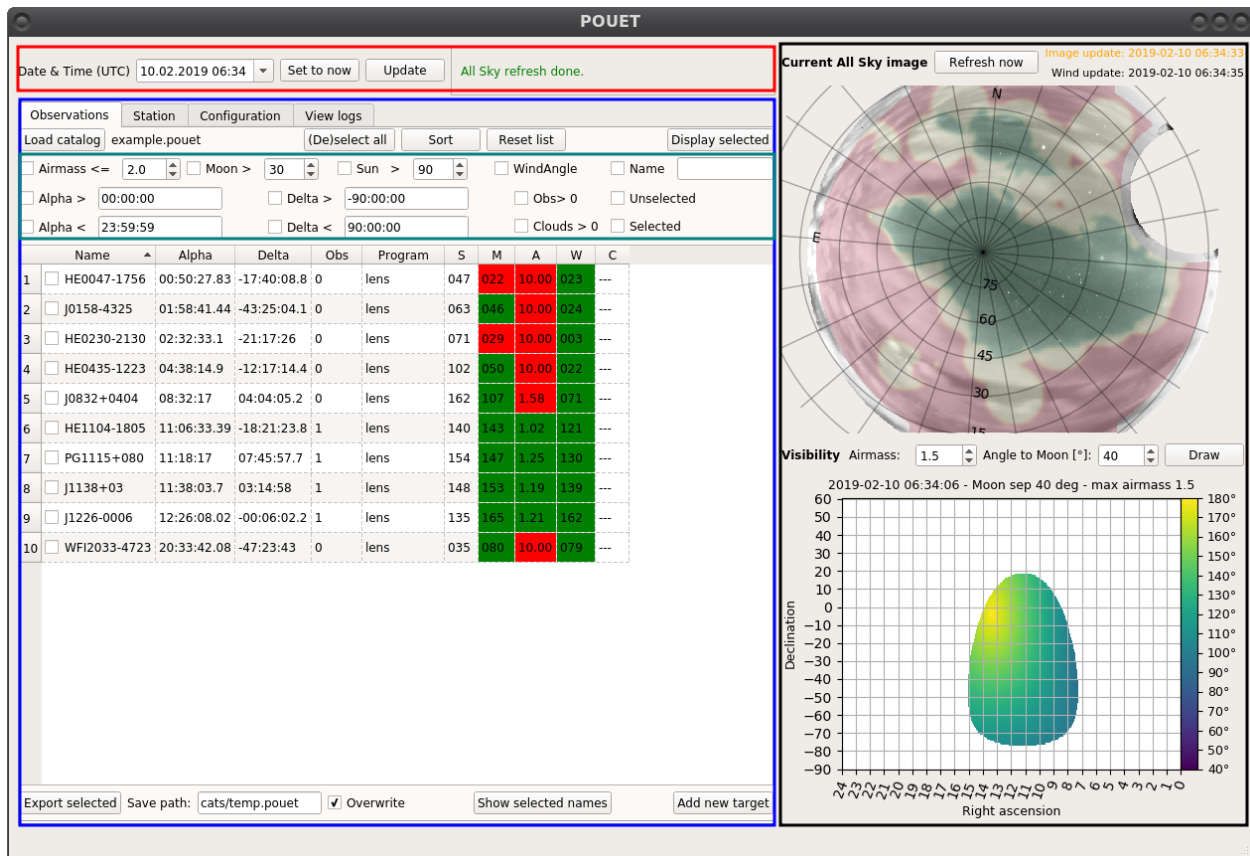


Fig. 2: A fresh POUET session, colored for the occasion

The red box at the top allows you to control the date and time.

The blue box below allows you to display and check the properties of the targets in your catalog - from the Observations tab. The other tabs allow you to display observing site properties (Station tab), POUET parameters (Configuration tab) and see the log of the operations you performed so far (View logs tab).

The inserted turquoise box allows a finer control of which targets are visible in the Observations tab.

The black box on the right contains the all sky display and visibility plots.

We will explore in more details these sections later on. For the moment, let's start by importing a bunch of targets into POUET.

Import a catalog

POUET has been primarily designed to let you browse through a large list of targets (a catalog) and highlight/display only the targets of interest.

Thus, your targets need to be arranged in a catalog. A POUET catalog can be as simple as a tab separated file, where each line is a target and each column a property. The first line is a header and second line is a separator. The minimal required properties are the name, alpha (HH:MM:SS.sss) and delta coordinates (DD:MM:SS.sss). A minimal working catalog should look like this:

| name | alpha | delta |
|-------------|-------------|-------------|
| ---- | ----- | ----- |
| HE0047-1756 | 00:50:27.83 | -17:40:08.8 |
| J0158-4325 | 01:58:41.44 | -43:25:04.1 |
| HE0230-2130 | 02:32:33.1 | -21:17:26 |
| HE0435-1223 | 04:38:14.9 | -12:17:14.4 |

The reading is done by `rdimport()`, which is a simple wrapper around `astropy.table.Table.read()`. Whatever suits astropy should work with POUET as well.

Note: If you do not have a separator line, you should use `data_start=1` in the `astropy.table.Table.read()` called in `rdimport()`.

To load your catalog in POUET, clic on `Load catalog` and chose your file. Then, if the file extension is not `.pouet` (more on this on the [Quality of Life Improvements](#) page), a popup will appear:

The pop-up asks you to associate the headers found in your catalog with the ones POUET needs. The fourth header, called `Obs program`, is optional. It relates to the observing program associated to your targets. The observing program defines a set of properties that some of your targets share. Currently, this is limited to the minimal distance to the Moon and the maximum airmass of your target. If you do not have such a property, simply select `None`. Below, you can chose a default observing program for the targets that have none assigned.

Note: currently, the properties of each observing program are simply used to raise warning flags if your targets are too close to the moon or at too high airmass. It does not prevent your targets to be displayed in POUET, so you can feel safe to use the default observing program (max airmass = 1.5, minimal distance to the moon = 30 deg). Learn how to define your own observing program on the [Quality of Life Improvements](#) page.

The `append` checkbox in the bottom part of the loading pop-up allows you define if you want to append the targets you are loading to the ones already in POUET, or if you want to overwrite them.

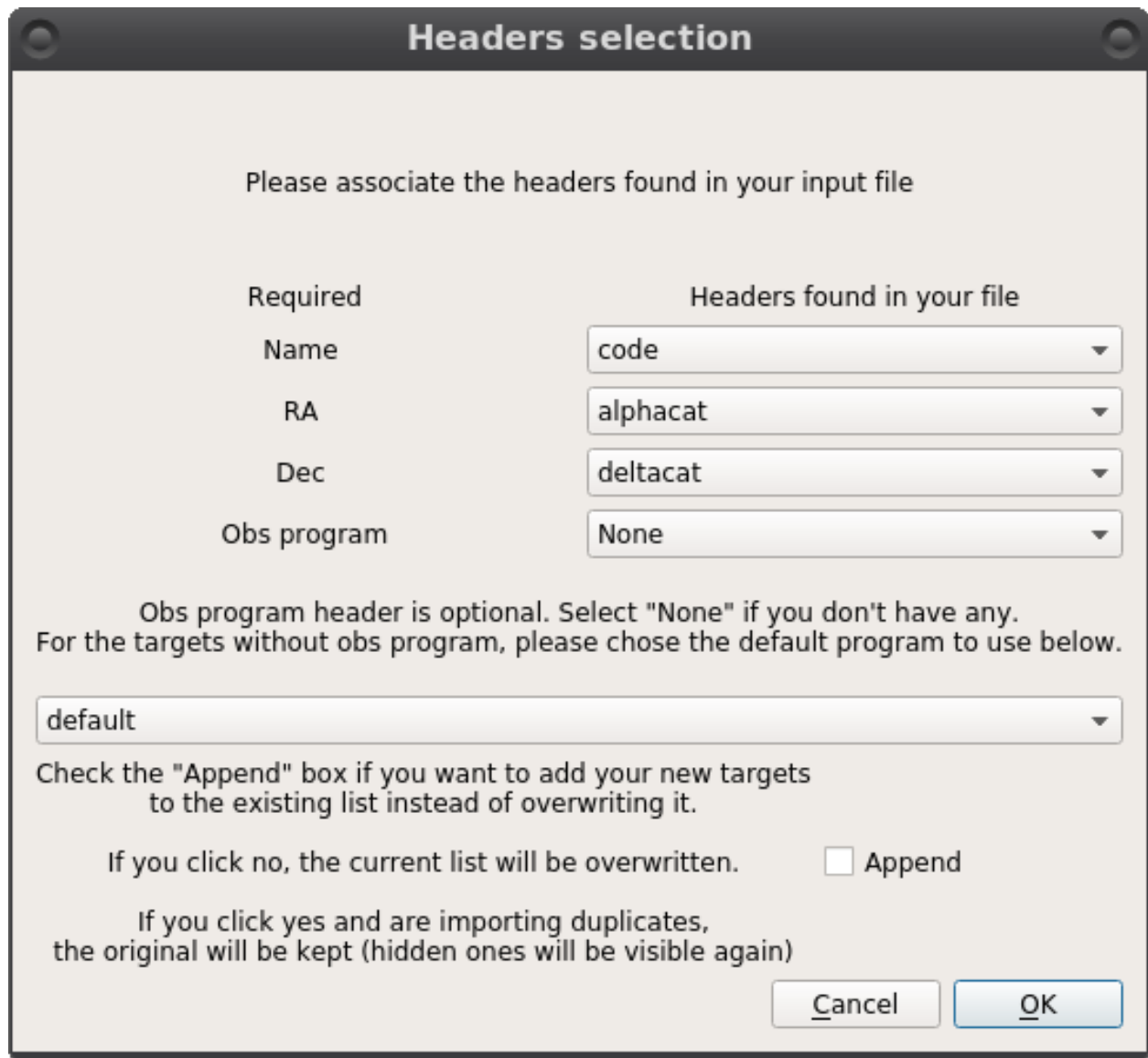
Note: if you append targets that are duplicate of existing ones, they will not be loaded. However, if the original targets were hidden (see more below about that), loading duplicates will make the original corresponding targets visible again.

Once a catalog is loaded, all its targets appear in the list view. Try to load the `example.cat` catalog available on the `cats` folder, chosing the `lens` obsprogram. The output should look as follows:

Each target appear as a line in the list view. Non-straightforwardly understandable header keywords are:

- `Obs` for the observability [0-1]
- `S` for the angular distance to the Sun [degree]
- `M` for the angular distance to the Moon [degree]
- `A` for the airmass [1-10]
- `W` for the angle between the telescope and the wind direction [degree]
- `C` for the cloud index [0-1]

The keyword cells get colored in green or red, depending if the current value matches the obsprogram constraints or not. A description of the wind angle can be found on the ref:[warningmessages](#) page. The observability and cloud index are detailed in the following section.



Headers selection

Please associate the headers found in your input file

| Required | Headers found in your file |
|-------------|----------------------------|
| Name | code |
| RA | alphacat |
| Dec | deltacat |
| Obs program | None |

Obs program header is optional. Select "None" if you don't have any.
For the targets without obs program, please chose the default program to use below.

default

Check the "Append" box if you want to add your new targets to the existing list instead of overwriting it.

If you click no, the current list will be overwritten. ☐ Append

If you click yes and are importing duplicates, the original will be kept (hidden ones will be visible again)

Fig. 3: Headers selection pop-up

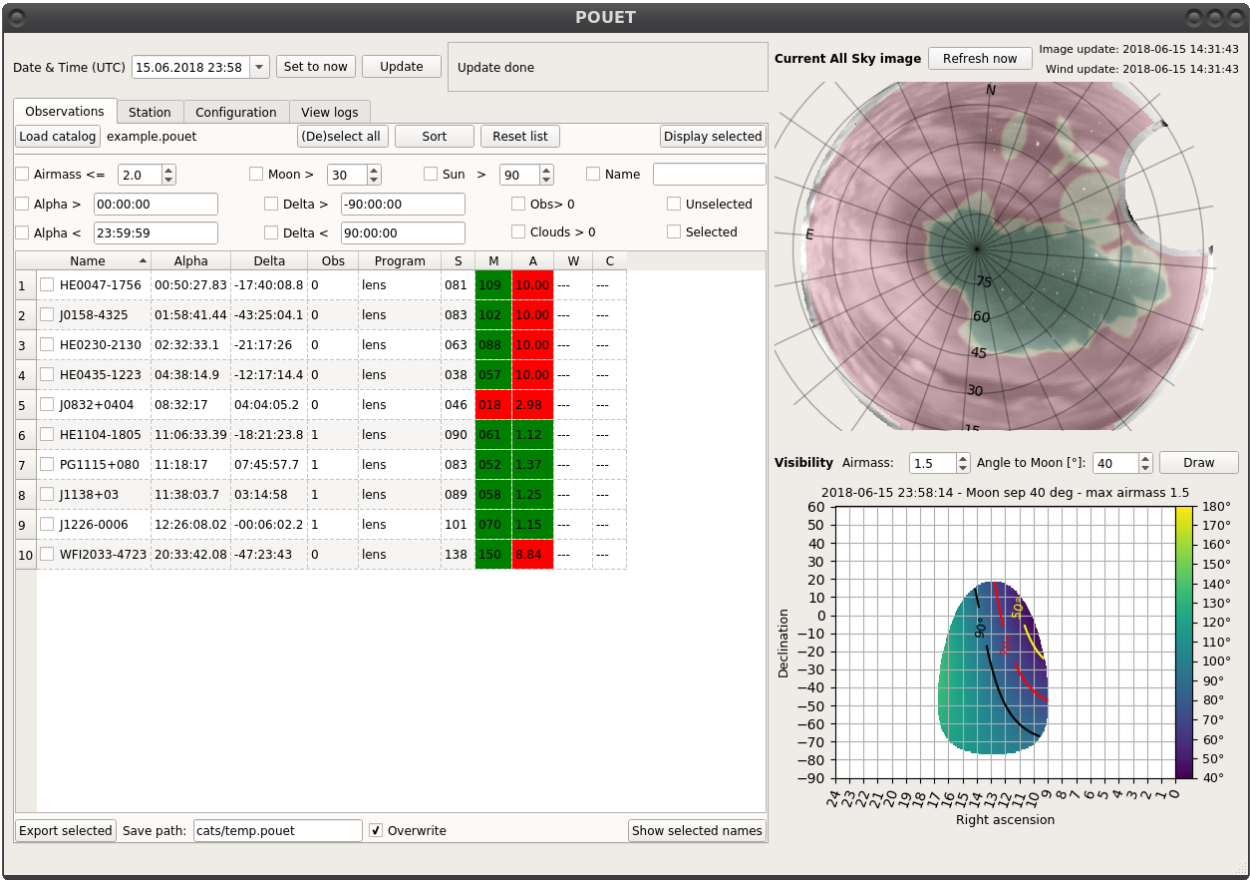


Fig. 4: Loading the example catalog of lensed quasars.

Note: The wind angle and cloud index are displayed only if the observing time is less than 30min off from the actual time.

Finally, you can manually add single targets on the fly to the current list of targets by clicking the `Add new target` button in the bottom right corner of the list view.

Warning: Be aware that POUET does not check that the name of the targets added manually are not duplicates of existing targets.

Sort your targets

You can order the list view by clicking on the corresponding headers. Sorting which targets appear in the list view is done through buttons in the small turquoise box of Fig. `main_colored`. Check the box corresponding on the criteria you want to apply to your list, and enter a value if needed. You can of course mix criteria as it suits you. To apply your sorting criteria, click on the `Sort` button.

The sorting criteria are the following:

- Airmass smaller than
- Sun distance larger than [degree]
- Moon distance larger than [degree]
- Wind Angle larger than 90 [degree]
- Name containing [string pattern]
- Right Ascension (Alpha) earlier/later than [HH:MM:SS], from 00:00:00 to 24:00:00
- Declination (Delta) higher/lower than [DD:MM:SS], from -90:00:00 to +90:00:00
- Clouds index larger than 0. 1 means clear sky, 0 means full cloud coverage.
- Observability larger than 0. The observability is a combination of airmass, moon distance, wind, cloud coverage, etc... that provide a “smart” way of sorting targets.
- Selected/unselected targets. You can check targets in the list directly.

Note: The value of the wind angle below which your targets are hidden (default = 90 degrees) can be changed in the configuration file (see *POUET configuration files*).

Note: In future versions of POUET, users will be able to define their own observability formula per observing program. The default one currently used can be read at `compute_observability()`.

The `Reset list` button make all the targets visible again. The same can be achieved by unchecking the sorting boxes and clicking on the `Sort` button.

If you sort the `example.cat` catalog with a right ascension later than 10h, a declination smaller than 50 degrees, an airmass smaller than 0.3 and keep only the targets that have “J1” in their name, you should have only two targets remaining:

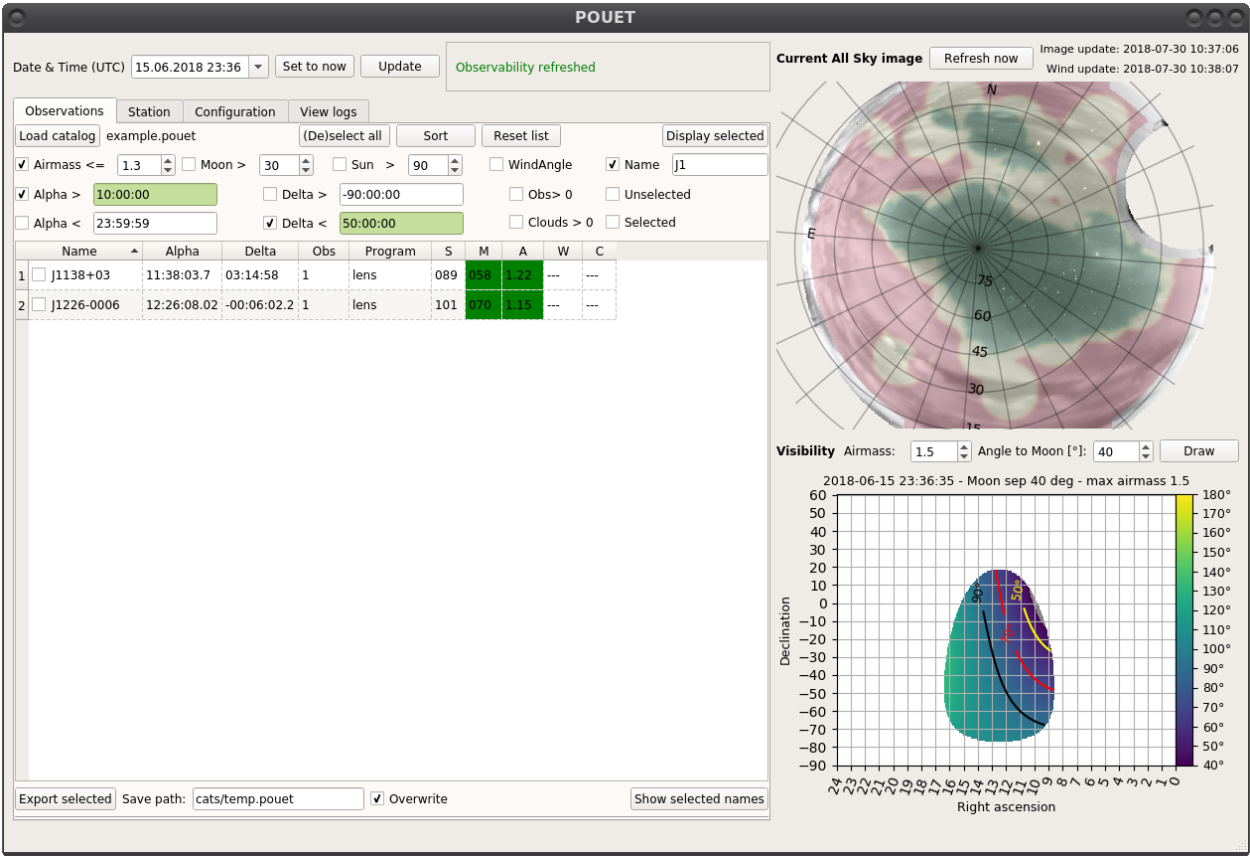


Fig. 5: Sorting the example catalog of lensed quasars.

Display targets on the all-sky and visibility views

The right part of POUET's main window regroups two views. At the top is the all-sky view, and at the bottom is the visibility view.

The all-sky view is, as its name says, an all-sky view from La Silla. The image comes from the [Danish telescope AllSky Camera](#), and is refreshed by default every 2 minutes. A RA/DEC grid is superposed on top of it, as well as a cloud analysis layer that colors the view according to the cloud coverage. At the top of the plot are displayed the date of the last update of the all-sky and the wind (see [Understand the warning messages](#)) as well as a Refresh button, especially useful if you deactivate the automatic updates (see [Extra tabs](#)).

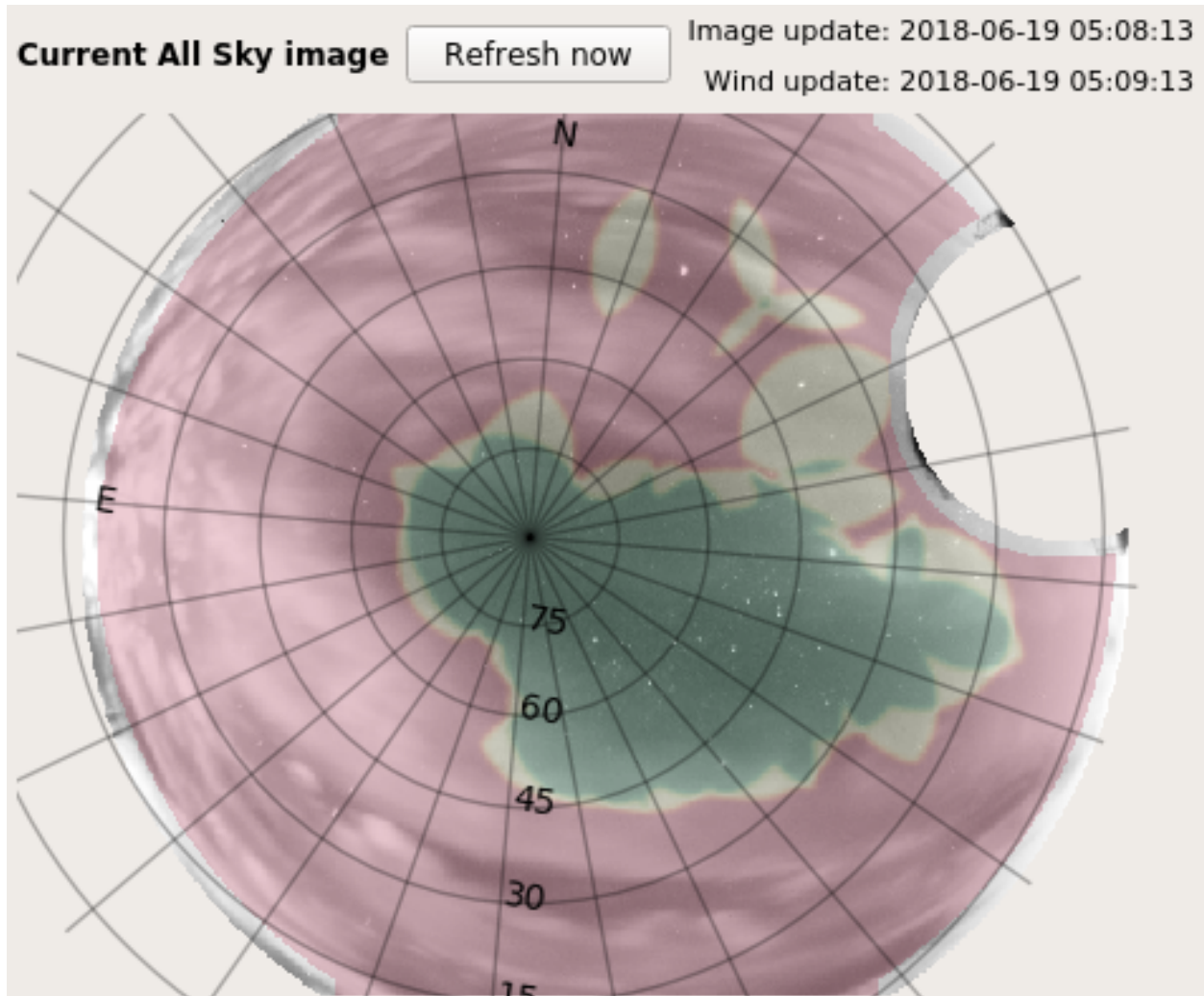


Fig. 6: All-sky view

The visibility view is a RA/DEC plot that indicates which region of the sky can be observed according to two criterias: the angle to the Moon, and the airmass. You can change the values at the top of the view and then click **Draw** to display a new region. The color of the visible region corresponds to the angle to the Moon, indicated in the right colorbar.

Note: If you hover the mouse cursor over the visibility view, a marker appears on the all-sky view at the corresponding

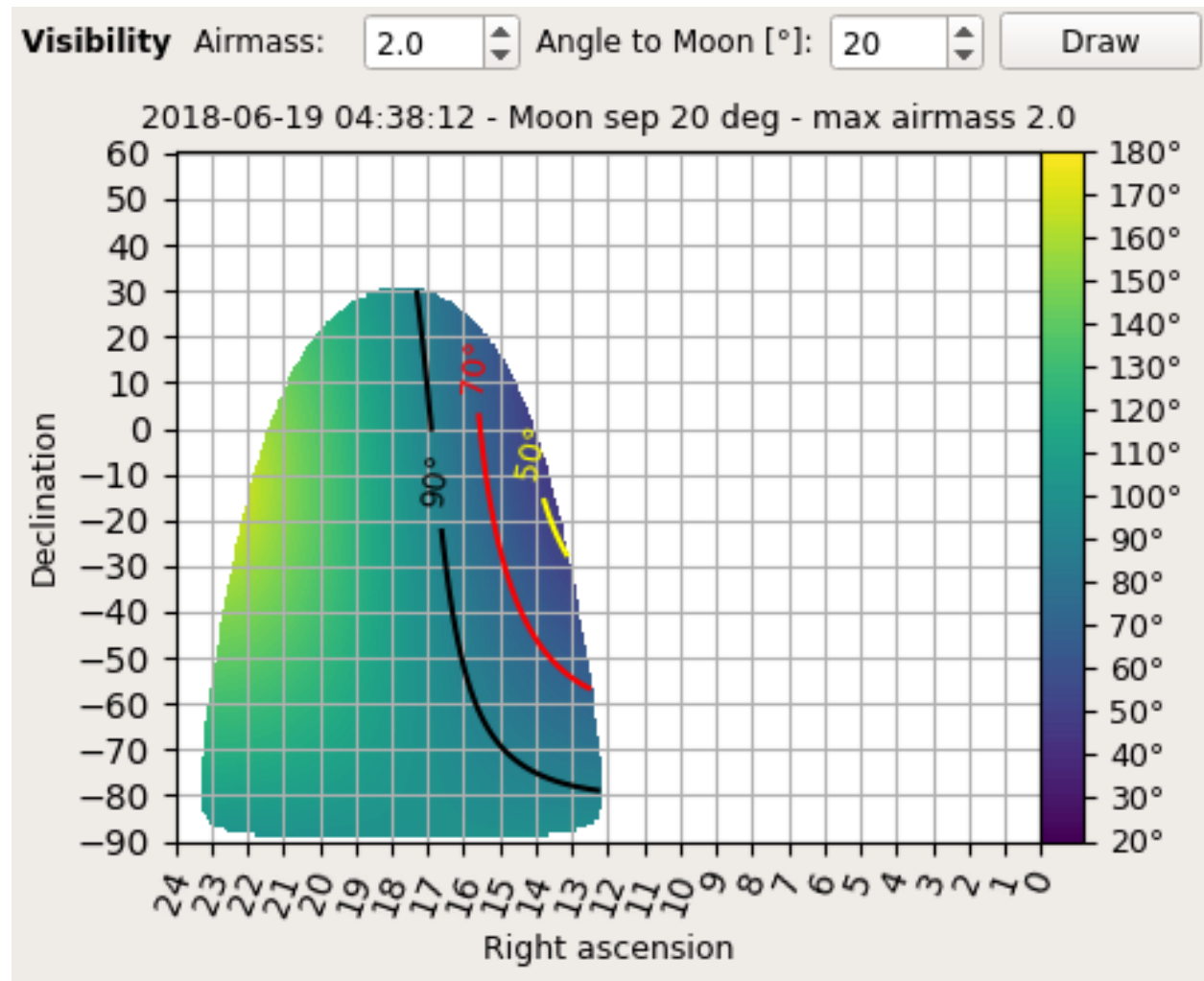


Fig. 7: Visibility view

coordinates.

If you check some targets on the list view and click on `Display selected`, they will appear on the corresponding coordinates on both the all-sky and visibility views.

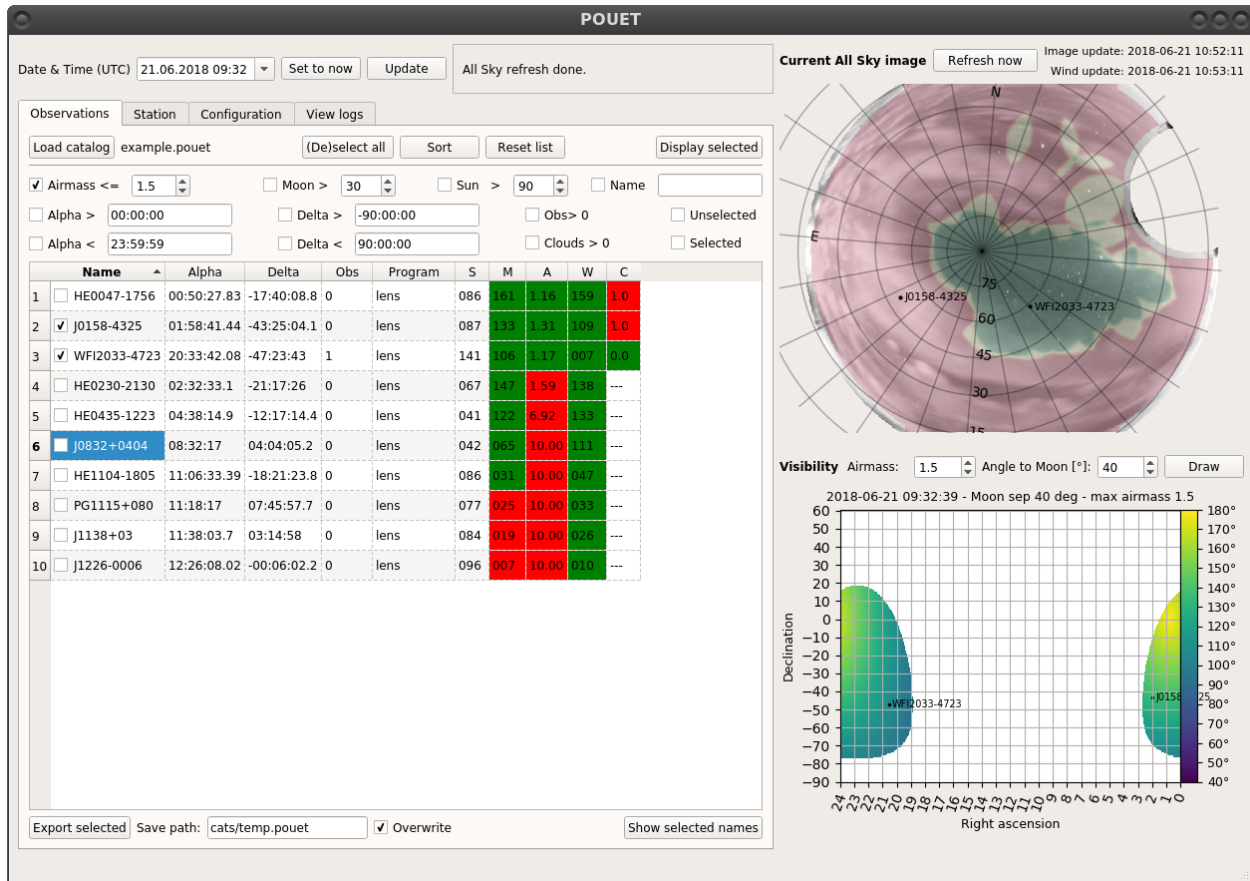


Fig. 8: Targets displayed on the visibility view

Extra plots

Upon right-clicking on a target in the list view, a drop-down menu will appear, giving you access to additional useful plots for the selected target. Currently, two options are available:

- `Show sky chart` opens a pop-up where you can query existing large sky surveys (currently **DSS**, **NEAT** and **2MASS-J**) for a finding chart of your target.
- `Show airmass` opens a pop-up displaying the evolution of the airmass of your target over its visibility range during the current night, in an Altitude-Azimuth radial grid.

Change the current time

If you want to plan ahead (or in the past, who are we to judge), you can use the time box at the top left of the main window to change the date and time of the observations. Once set, click on `Update` to refresh the values displayed in your target list. If you are more than 30 minutes away from the current time, both the clouds and wind values are disabled in the list view.

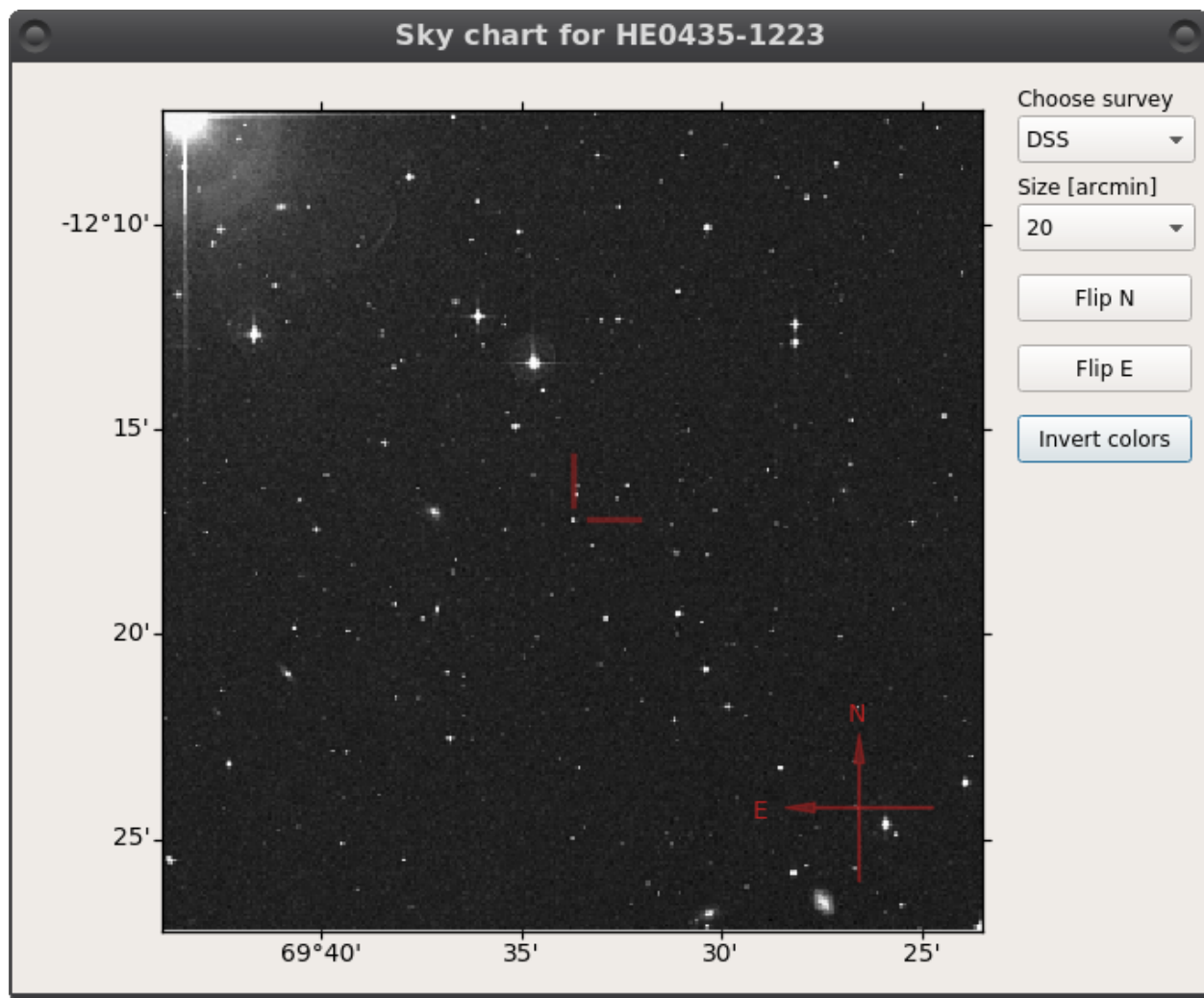


Fig. 9: Sky chart of HE0435-1223

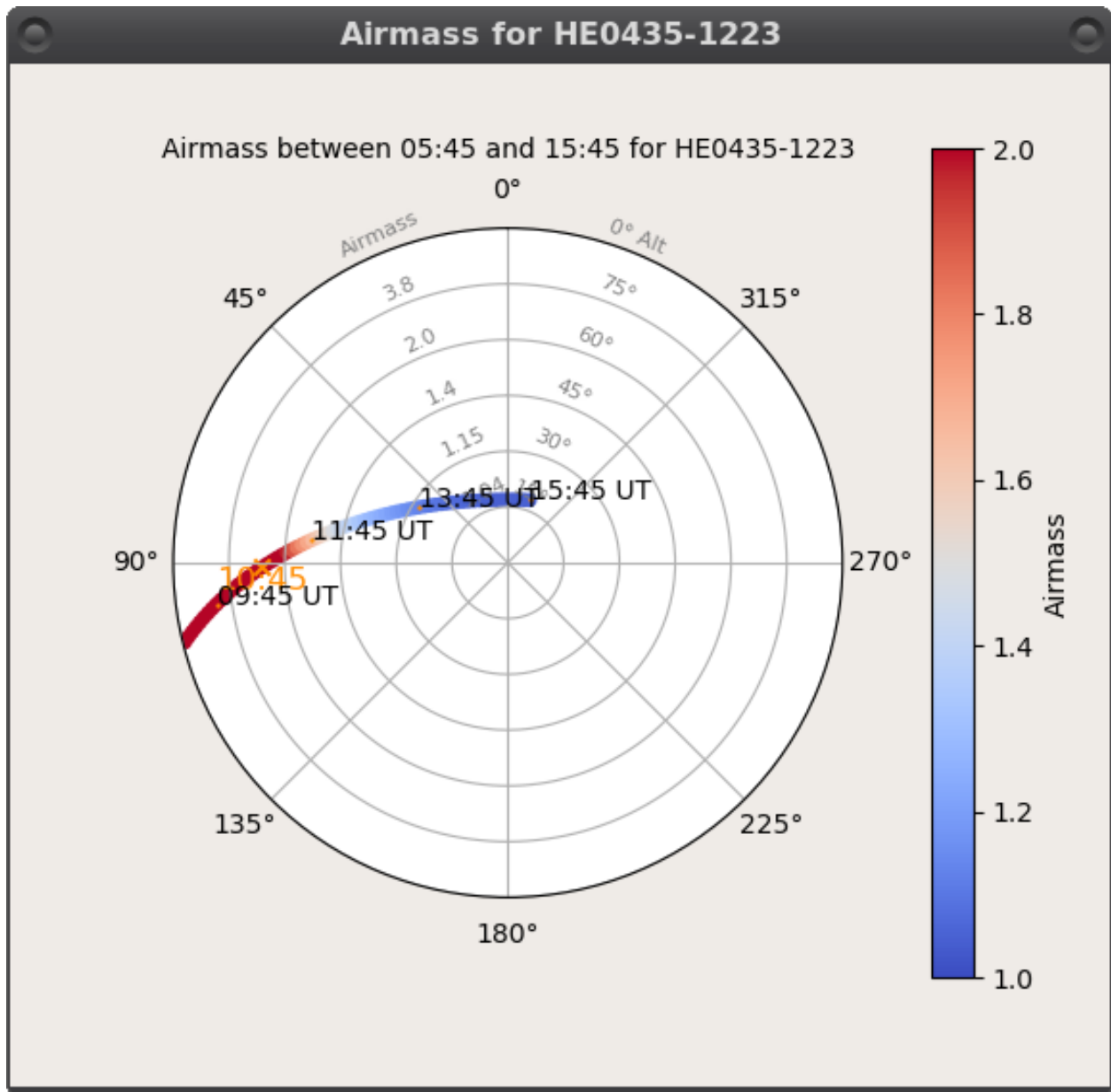


Fig. 10: Airmass evolution of HE0435-1223 over night

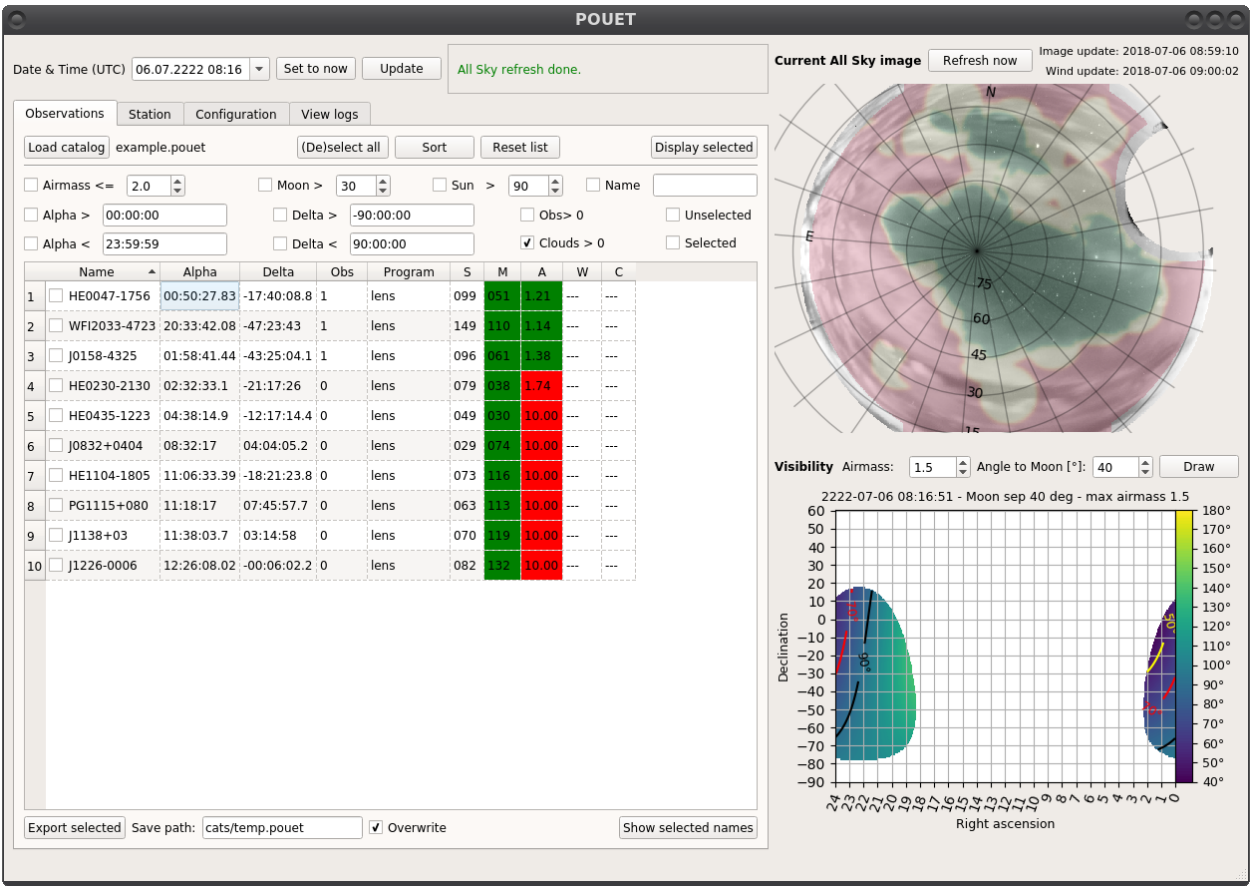


Fig. 11: HE0047 will still be visible in July 2222. No wind or clouds prediction, though.

Note: The all-sky view still displays the up-to-date all-sky image. This is designed to remind you of the current weather situation.

You can come back to the current time by clicking on `Set to now`, then `Update`.

Understand the warning messages

There are two kinds of warning messages. The ones that tell you something is wrong with POUET's code (bad ones, report them [here](#)) and the ones that tell you to pay attention to what's happening. In this page, we focus on the latter.

General status of POUET

At the center top of the main window is a small box. POUET uses this box to print all kind of messages

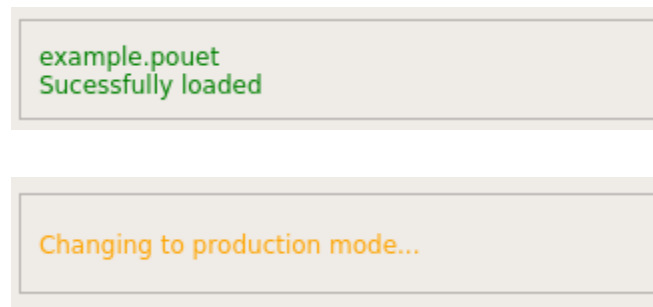


Fig. 12: Examples of status messages

You can see it as a simplified log to let you know the status of the most recent task performed by POUET. A green message tells you that things happened as expected (catalog successfully loaded, all-sky image successfully downloaded), a yellow message tells you that a task is currently being processed by POUET (retrieving a finding chart, refreshing the weather display. . .) and a red message tells you that something went wrong (loading catalog does not comply with POUET's import standards, no connexion to the weather service, etc. . .). To each of these simplified messages corresponds one or more detailed entries in the `Log` tab.

Pointing limit

Usually, when the wind blows above a certain limit, you will want to avoid pointing the telescope in the wind direction (± 90 degree from the wind). The pointing limit is defined as 15m/s, but can be changed by the user before launching POUET (see *POUET configuration files*). When the measured speed exceed this threshold, a hatched grid will appear on the all-sky and visibility views to indicate which regions are not accessible. The targets in the list view will have their respective Wind cells paint in red.

If the wind blows really too hard, then the telescope should be closed for safety measures. Most modern telescopes have automatic closing procedures (either a closing signal sent to the telescope, or a human operator forcing you to close it) but just in case you are all alone and don't pay too much attention to your official weather report, POUET displays a visual reminder in both the visibility and all-sky views.

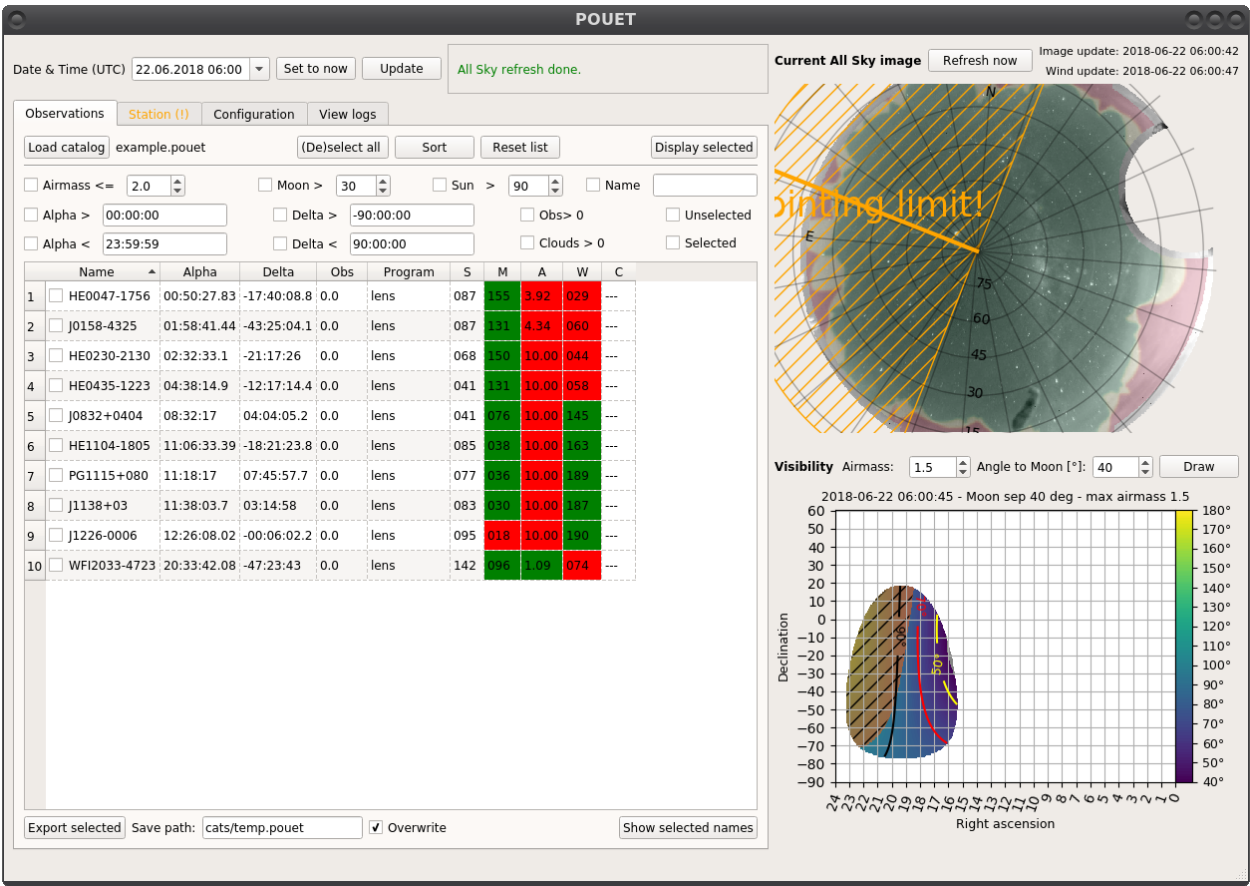


Fig. 13: When the wind blows too hard, a pointing limit grid appears on the displays.

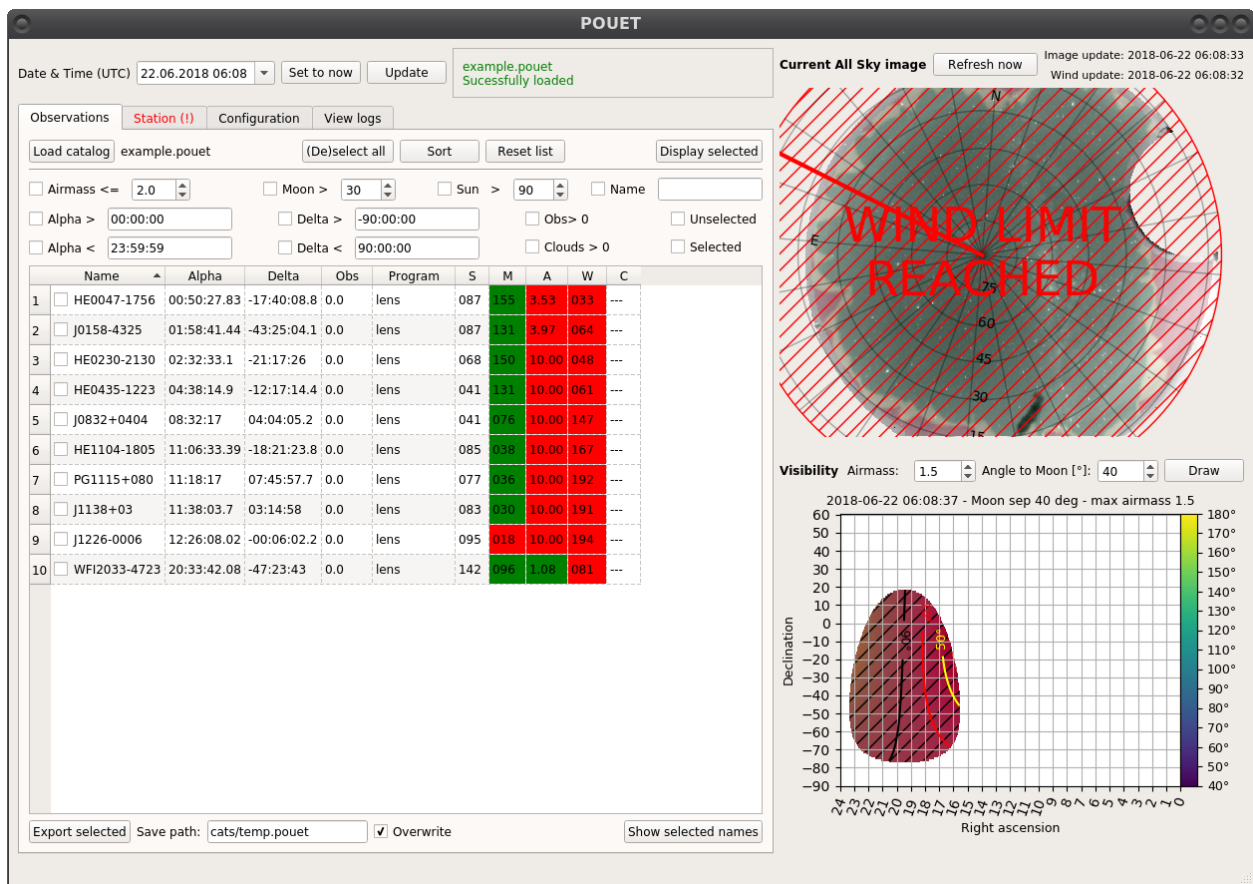


Fig. 14: When the wind blows really too hard, POUET reminds you that no observations are possible.

POUET closing limit is 20m/s, and as the pointing limit can be changed (see [POUET configuration files](#)). As an extra warning, note that in both cases the Station tab changed color (yellow for pointing limit, red for closing limit)

Clouds detection

The approach chosen in POUET to detect clouds is very straightforward: small boxes are drawn on the all sky image and a peak detection algorithm counts how many stars are visible. If that number is large enough, then POUET assumes the sky is clear and paint the all-sky in green. A Gaussian filtering is also applied to reduce the false cloud detection due to the presence of the moon.

The number of stars per box has been optimized to match a visual detection from the [Danish telescope AllSky Camera](#). If you plan to use another all-sky (see [POUET configuration files](#)), the cloud detection algorithm might need some adaptation.

A target behind the clouds will have its cloud flag C in the list view painted in red with a value of 0, and, of course, appears in a correspondingly red region in the all-sky display. The yellow regions corresponds to areas with a cloud density between thin and thick; you can still hope for some flux if you do e.g. spectroscopy of bright objects, but don't even attempt to do photometric observations, unless you are really desperate (The "I'm-a-PhD-student-whose-thesis-can-be-awarded-only-if-I-get-these-observations-done-tonight" kind of desperate).

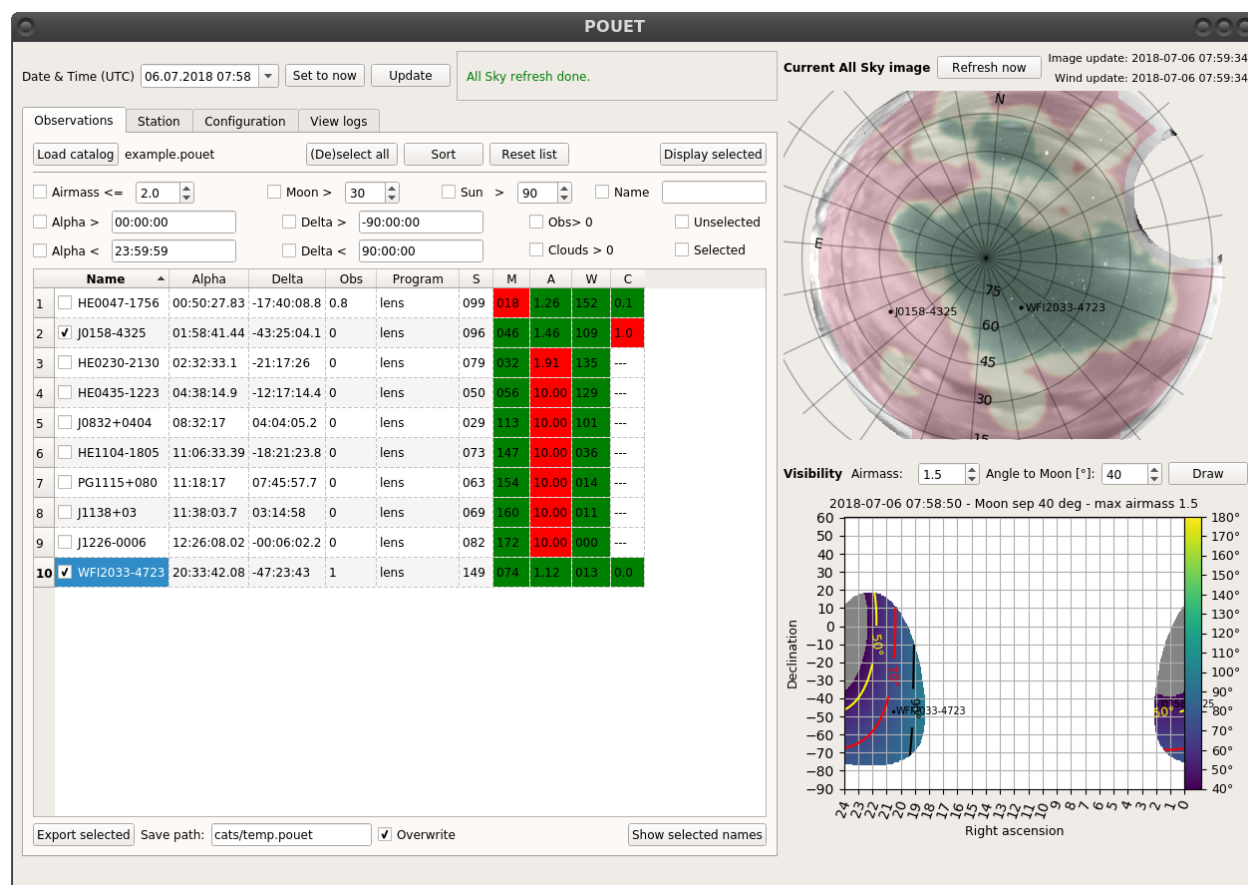


Fig. 15: When everyone else is closed because of bad weather, the smart POUET user knows he can still observe WFI2033.

Moon distance

In a similar fashion than the wind limit or cloud coverage, a target too close to the moon will have its M box in the list view painted in red. The default minimum moon distance is 30 degree, but can of course be changed (see *Quality of Life Improvements*)

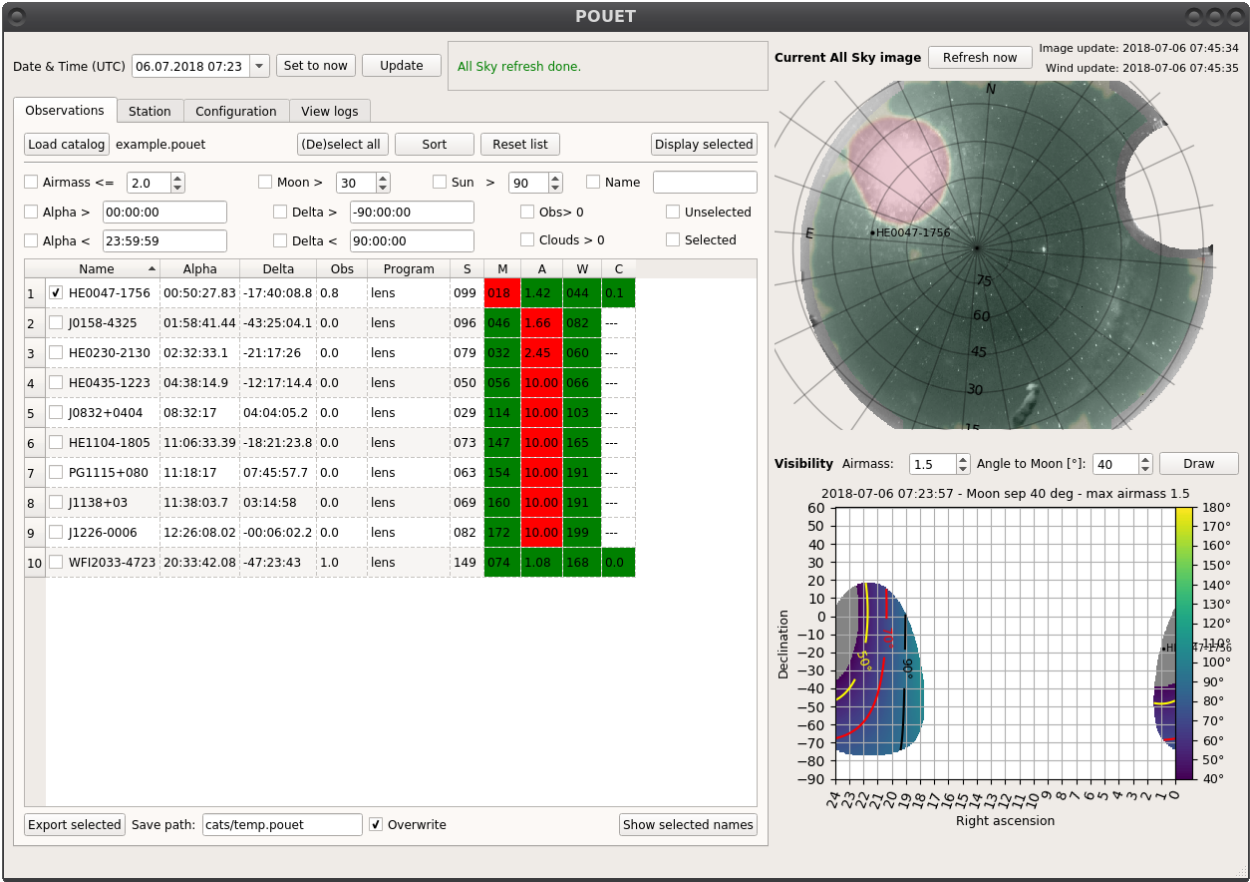


Fig. 16: HE0047-1743 is too close to the moon.

The same warning flag exists for the distance to the Sun, S.

No internet connexion

Observatories are remote places, whose network connections are sometimes hectic. POUET works as best as it could in offline mode. Of course, the all-sky view will be disabled, similarly to the finding charts

Extra tabs

Besides the Observation tab are three other important tabs. In this section, we briefly review the important information you can get from them.

Station tab

The `Station` tab gives you an overview of the Site conditions: weather status, Sun and Moon position, etc... If something goes wrong (Sun is up in the sky, humidity is too high, wind blows too strong, etc...), the `Station` label at the top of the window gets colored, as well as the relevant part in the tab. The example below shows the Site status when the wind blows stronger than the defined pointing limit.

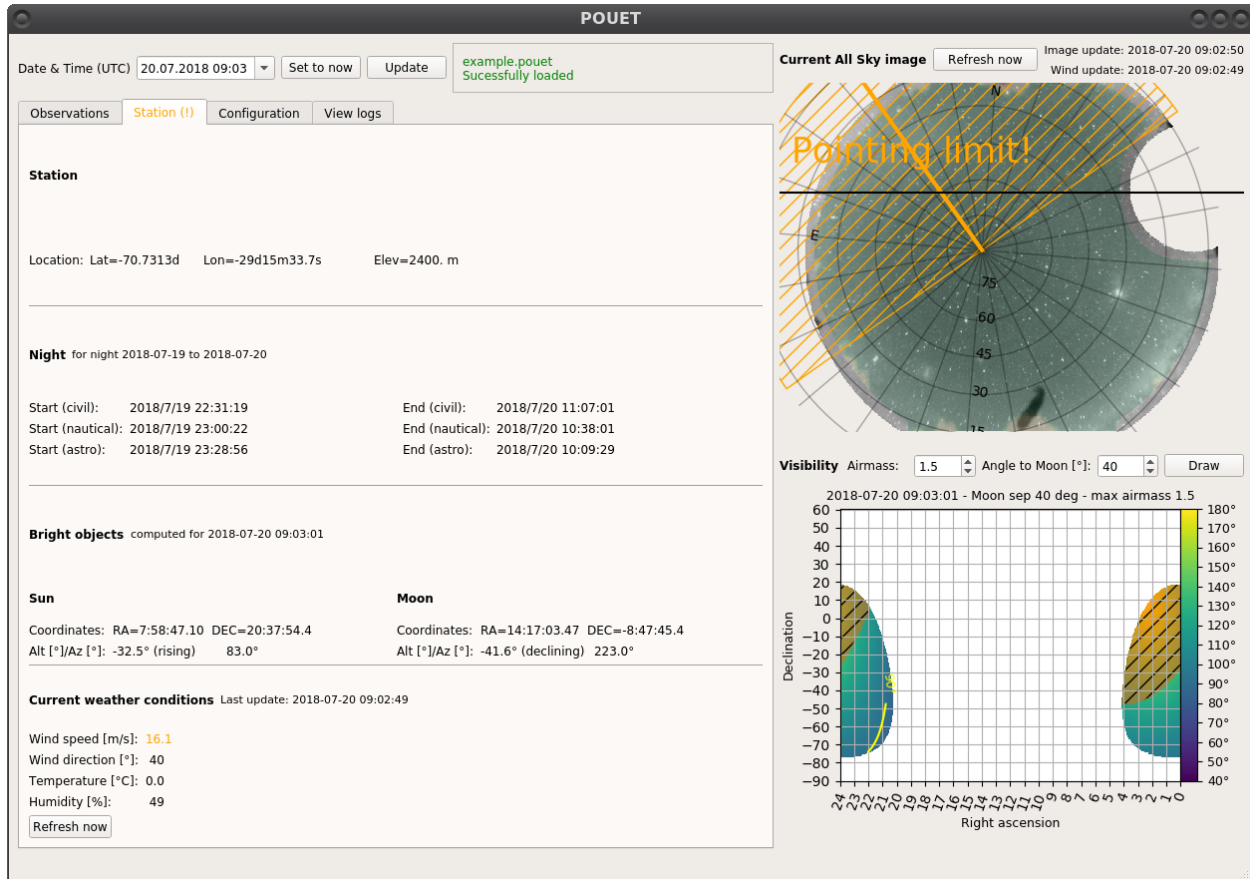


Fig. 17: The `Station` tab of POUET indicates that the wind blows above the pointing limit.

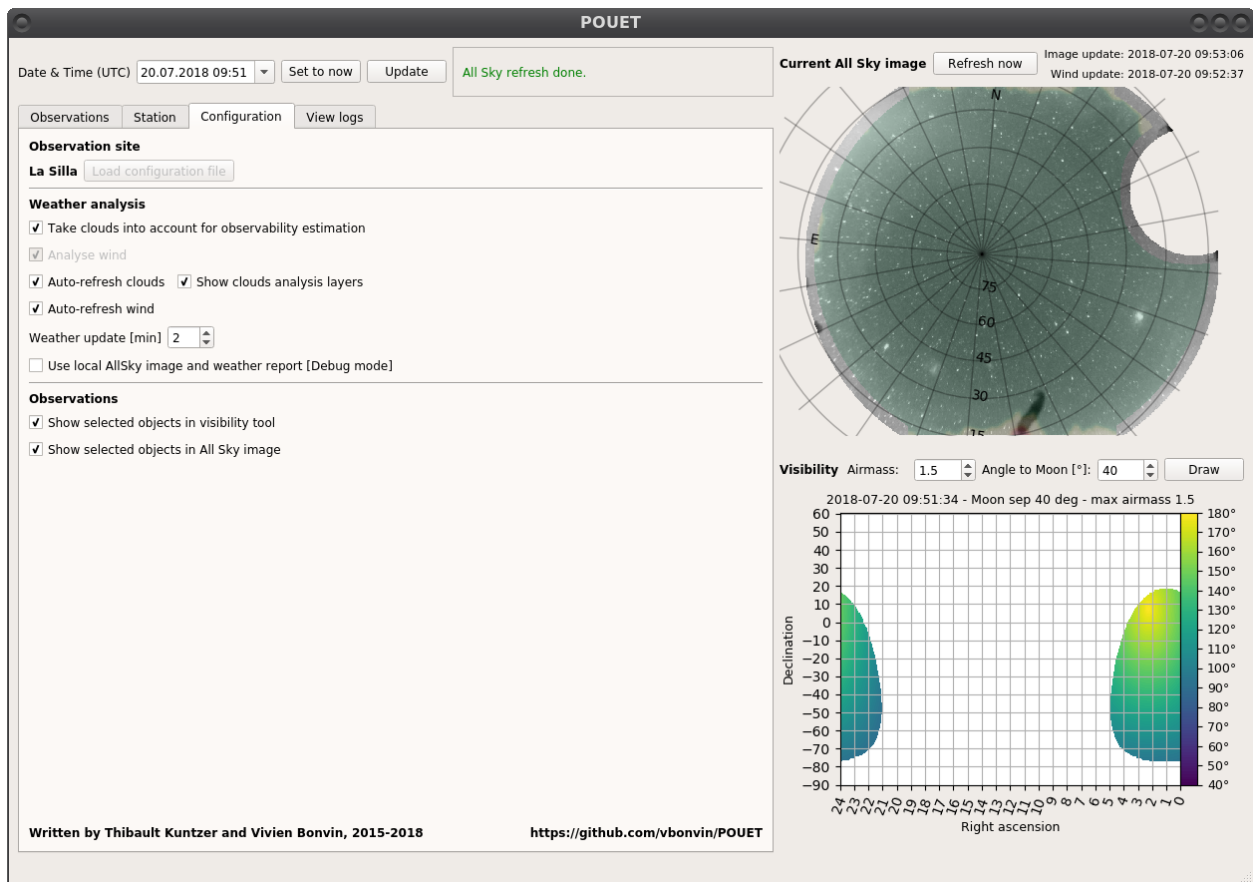
The default values at which the `Station` panel gets colored can of course be redefined (see [POUET configuration files](#)).

Configuration tab

The `Configuration` tab allows you to change some of the running parameters of POUET.

By checking/unchecking the checkboxes, you can control the behaviour of the weather analysis. You can also control the behaviour of the `Display` selected button in the `Observations` panel, if you want the selected targets to be displayed only in the all-sky or visibility view.

Note: an upcoming feature (planned for version 0.4) will allow you to create/load a new configuration file directly from the `Configuration` panel. Until then, the corresponding button will remains gray, a dull reminder of our failures as developer to deliver everything we promise on time.



View logs tab

Particularly useful is something goes nuts and you don't know why, the `View logs` tab keeps track of the majority of what happens in POUET. In case you want to [report a suspicious behaviour](#), copy/pasting the last five minutes of the logs along with your report might be super useful for us.

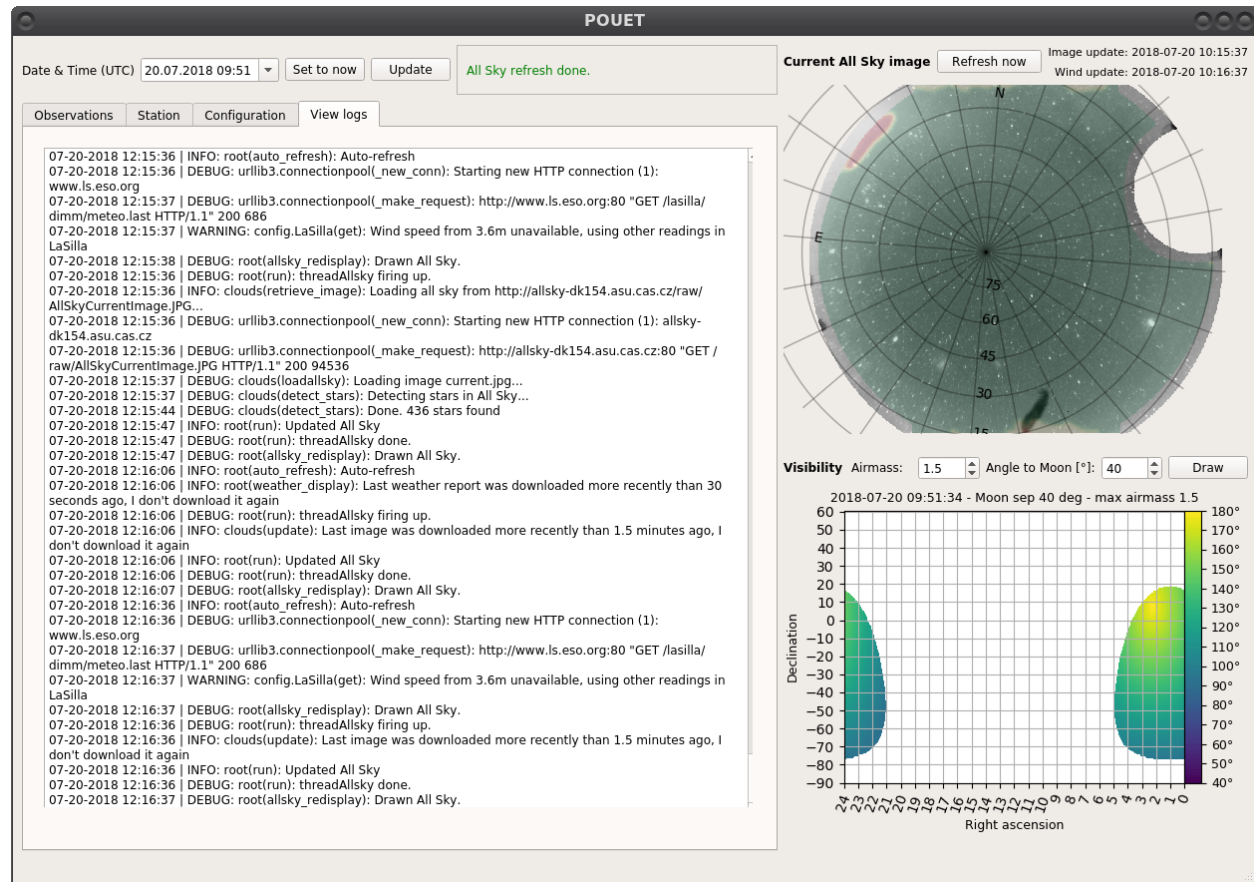


Fig. 18: The `View logs` tab of POUET usually displays a massive wall of text.

Quality of Life Improvements

POUET provides a bunch of interesting features that can make your life much more comfortable. Here is what you should know.

Save your catalogs in .pouet format

Any catalog successfully imported can be saved in a `.pouet` format, using the `Export` selected button at the bottom left of the `Observations` tab.

The advantage of having your catalog in a `.pouet` format is that they can be loaded without prompting any import-related questions. Only the selected targets are written when you click on `Export` selected, which allows you to trim down your list according to whichever criteria you find useful. The path is to be entered in the `Save path` field.



Fig. 19: Export-related widgets, located at the bottom of the Observations tab

The `Overwrite` checkbox, checked by default, ensure that your fully replace any existing catalog when exporting. If unchecked, the selected OBs will be *added* to the existing catalog, thus allowing you to build .pouet catalogs combining targets from various other catalogs.

Note: A safer export function, warning you when you are about to erase an existing catalog as well as a rollback mode will be available in v0.4.

Export the list of selected names

Say you know which targets from your catalog you plan to observe, and you would like to access the list of names quickly to copy/paste it somewhere. This can be done by clicking on the `Show selected names` at the bottom right of the Observations tab. This will prompt a pop-up with the names of all the selected targets from the list view.

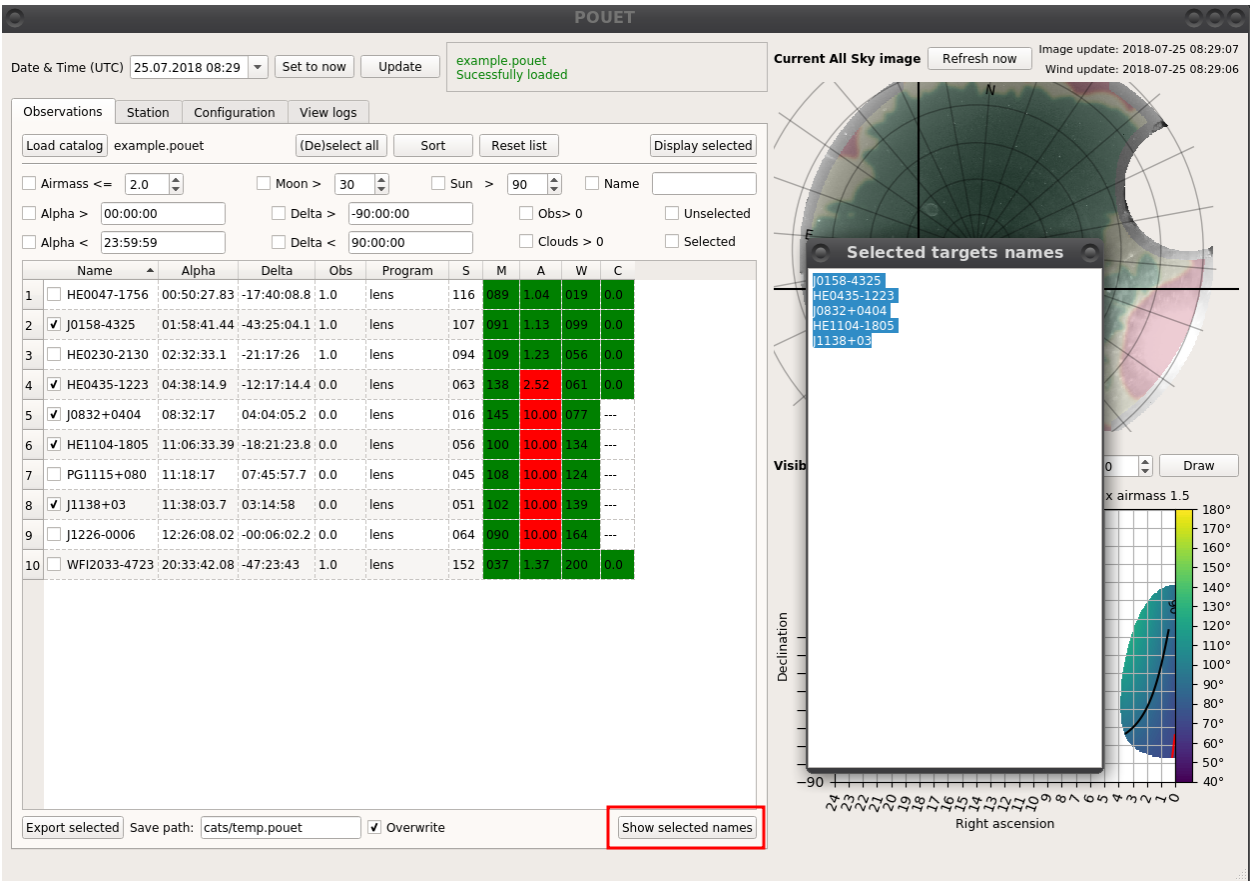


Fig. 20: The pop-up that appears after clicking on the `Show selected names` button.

Change the default obsprogram/create a new obsprogram file

The default obsprogram parameters can be accessed in `pouet/obsprogram/progdefault.py`. It can be accessed and modified with any text editor.

```
1  #=====
2  # Default program
3  #=====
4
5  # Set general constraints
6  # If those numbers are object dependent, set to None and compute in observability function
7  minangletomoon = 30
8  maxairmass = 1.5
9
10 # If there is a common exptime, otherwise define a get_exptime function below
11 exptime = 35*360
12
13 #=====
14 # Now define the exptime function, arguments must be : attributes and obs_time
15 #=====
16 def get_exptime(attributes, obs_time): # Just to have some fun
17     return 600.
18
19 #=====
20 # Now define the observable function, arguments must be : obj and obs_time; should return 1 if
21 # observable, 0 otherwise
22 # Already checked in the observable class: angle to moon, wind, clouds, airmass
23 # So this gives the possibility to have more specific tests, like say the delta v with the moon
24 #=====
25 def observability(attributes, obs_time):
26     msg = '' # This could contain messages that justify the impossibility to observe
27     warnings = '' # This contains warnings
28
29     return 1, msg, warnings
30
```

This file contains a couple of functions we are still playing with (keep in mind that POUET is still under development) and that are currently not used. The only values of interest are the `minangletomoon` and `maxairmass`, that define the observability of your targets once loaded in POUET. To change the default behaviour of POUET, simply edit these values.

If you work with multiple kind of targets that have various angle to moon and airmass requirements, you can also create new obsprogram files. Simply copy/paste the `progdefault.py` file, rename it as `prog%YOUROBSPROGRAMNAME%.py` and edit its value. When loading a new catalog in POUET, your new obsprogram will appear in the obsprogram popup selection.

Note: The obsprogram files are read each time you import a new catalog. If you have a POUET instance launched and edit the obsprogram files on the fly, you should reload your catalogs for the effects to take change, no choice.

POUET configuration files

Change the default configuration (general and Site)

The general POUET configuration file is found under `pouet/config/settings.cfg`. This is a simple text file that is loaded when a POUET instance is launched. Simply edit the variables values to adapt POUET to your liking.

```
1  [color]
2
3  success: green
4  nominal: black
5  nodata: white
6  warn: orange
7  limit: red
8
9  [validity]
10
11  # Minimun time to wait before being able to download a new weather report [in s]
12  weatherreportfrequency: 30
13
14  # Minimun time to wait before being able to download a new all sky image [in min]
15  allskyfrequency: 1.5
16
17  # What is the validity [in min] of the weather report
18  weatherreport: 10
19
20  # What is the validity [in min] of the all sky image
21  allsky: 10
22
23  # After which delta time [in min] do we not consider clouds and all sky in the observability
24  cloudwindanalysis: 30
25
26  # What is the maximum delta time [in min] between observation time and last all sky refresh to
27  # still displaying the coordinates on the all sky when hovering over the visibility plot?
28  showallskycoordinates: 60
29
30  # What is the maximum delta time [in min] between observation time and last all sky refresh to
31  # still displaying selected targets in the observable tab?
32  showallskytargets: 60
33
34
35  [misc]
36
37  # What is the minimum angle [deg] to wind below which you want to be able to hide
38  # the observables from the list view?
39  minangletowinddisplay = 90
```


The default Site configuration file is found under `pouet/config/LaSilla.cfg`. As above, the values of the parameters can be edited prior to launching a POUET instance.

```
1 [location]
2
3 latitude: -29d15m33.7s
4 #[units: degrees] Format: -00d00m00.0s
5
6 longitude: -70.7313d
7 #[units: degrees] Format: -00.0000d
8
9 elevation: 2400.
10 #[units: m]
11
12 [weather]
13
14 url: http://www.ls.eso.org/lasilla/dimm/meteo.last
15
16
17 windWarnLevel: 15
18 windLimitLevel: 20
19 humidityWarnLevel: 80
20 humidityLimitLevel: 90
21
```

All these parameters are fully described in the config files. If you want to adjust the humidity and wind warning levels, you have to edit the `windWarnLevel`, `windLimitLevel`, `humidityWarnLevel` and `humidityLimitLevel` parameters of the `LaSilla.cfg` file.

Unless you work under very specific conditions, there should be no need to tweak the other parameters - we thus recommend you to go with the default settings.

Note: On-the-fly overriding of (most of) the default parameters from POUET's Configuration tab will be available in v0.4.

Adapt POUET to another observing site

This feature, along with its tutorial, will be fully available in a future version of POUET (we target the official release version).

In the meantime, you can still change the default observing Site configuration, but beware that the allsky view will not be adapted anymore. You can deactivate it from the Configuration tab (see [Extra tabs](#)).

1.2 Questions & Answers

1.2.1 Why this name?

Because it sounds funny.

1.2.2 Can I use POUET from every observatory?

Technically, yes. You just need to define a new config file associated to your location, and disable the webcam (see *POUET configuration files*).

1.2.3 How much trust can I put in your weather report?

The weather variable displayed in the code are directly copied from [La Silla weather report](#). Staled weather information turns the `Site` tab red in POUET. However, we offer no guarantee that the displayed information is correct. Always refer to the official weather report of your observatory - POUET is not a replacement for it.

1.2.4 It's full of bugs! What should I do?

Head over to our [github repository](#) and explain to us what is happening. We will do our best to answer within reasonable delays.

1.2.5 How can I contribute?

The code is in open access (GNU GPLv3), feel free to reach us on [github](#) and propose your help.

1.2.6 The code seems... a bit convoluted sometimes, to say the least.

Both authors have learned (py)Qt coding this project. Be understanding.

1.2.7 What is POUET's future?

We aim to make it as moduable as possible, so it can be used from anywhere.

1.3 pouet

1.3.1 pouet package

Submodules

`pouet.clouds` module

`pouet.main` module

`pouet.meteo` module

`pouet.obs` module

Define the `Observable` class, the standard object of pouet, and related functions

```
class obs.Observable (name='emptyobservable', obsprogram=None, attributes=None, alpha=None,
                      delta=None, minangletomoon=None, maxairmass=None, exptime=None)
```

Bases: `object`

Class to hold a specific target from any observational proggamm

Unvariable parameters are defined at initialisation

Variable parameters (distance to moon, azimuth, observability,...) are undefined until associated methods are called

Constructor

An observable is the basic building bloc of POUET. It stores all the values of a target that are needed to compute its observability at a given time from a given position

Parameters

- **name** – string, name of the observable
- **obsprogram** – string, related to one of the existing programs defined in :any:'obsprogram'
- **attributes** – any type of extra information about the target you want to store
- **alpha** – Right Ascension angle, in hours HH:MM:ss
- **delta** – Declination angle, in degree DD:MM:ss
- **minangletomoon** – float, minimum angle in the sky plane to the moon below which the target is not to be observed
- **maxairmass** – float, maximum airmass below which the target is not to be observed
- **exptime** – float, expected exposure time of the target

compute_airmass (*meteo*)

Computes the airmass of the observable.

Parameters **meteo** – a Meteo object, whose time attribute has been actualized beforehand

compute_altaz (*meteo*)

Computes the altitude and azimuth of the observable.

Parameters **meteo** – a Meteo object, whose time attribute has been actualized beforehand

compute_angletomoon (*meteo*)

Computes the distance to the moon

Parameters **meteo** – a Meteo object, whose time attribute has been actualized beforehand

compute_angletosun (*meteo*)

Computes the distance to the Sun

Parameters **meteo** – a Meteo object, whose time attribute has been actualized beforehand

compute_angletowind (*meteo*)

Computes the angle to wind

Parameters **meteo** – a Meteo object, whose time attribute has been actualized beforehand

compute_observability (*meteo*, *cwvalidity=30*, *cloudscheck=True*, *verbose=True*, *displayall=True*, *future=False*)

Update the status using [update\(\)](#). Compute the observability param, a value between 0 and 1 that tells if the target can be observed at a given time. Also define flags for each parameter (moon, wind, etc...)

The closer to 1 the better 0 is impossible to observe

Parameters

- **meteo** – a Meteo object, whose time attribute has been actualized beforehand
- **cwvalidity** – float, current weather validity: time (in minutes) after/before which the allsky cloud coverage and wind are not taken into account in the observability, effectively setting the future variable to True
- **verbose** – boolean, displaying the status of the observable according to the present function
- **displayall** – boolean, if verbose is True, then print also the targets that are not observable.

- **future** – boolean, if set to True then cloud coverage and wind are not taken into account in the observability.

copy ()

Returns an, observable, python deep copy of the current observable

is_cloudfree (*meteo*)

Computes whether the pointing direction is cloudy according to the altaz coordinates in memory

Parameters **meteo** – a Meteo object, whose cloudmap attribute has been actualized beforehand

todo: instead of taking altaz coordinates in memory, shouldn't we use meteo.time to recompute altaz on the fly?

Note: is_cloudfree is actualized with 0: cloudy or 1: no clouds. If unavailable, returns 2: connection error, if error during computation of observability from map: 3

update (*meteo*)

Update the observable parameters according to the meteo object passed: altitude, azimuth, angle to wind, airmass, angle to moon and angle to sun.

Parameters **meteo** – a Meteo object, whose time attribute has been actualized beforehand

obs.**rdlexport** (*filepath, observables, append=False*)

Save a list of observables at a given filepath, respecting the formatting used when default importing with :meth:'obs.rdbimport'.

Parameters

- **filepath** – string. Path of where the .pouet file is written
- **observables** – list of Observables to save
- **append** – boolean. If True, then the current observables are added to the existing list

Note: append=True only works if the file you want to append to has the correct formatting. See headerline and headersubline in the source code.

obs.**rdbimport** (*filepath, namecol=1, alphacol=2, deltacol=3, obsprogramcol=4, obsprogram=None*)

Import an rdb catalog into a list of observables

Must be compatible with astropy Table reader (i.e. a header line, then an empty/blank/comment line, then each obs in a dedicated line, attributes separator by a tab or a space)

Parameters

- **filepath** – path to the file you want to import. Must be a text file, format is not important.
- **namecol** – integer, index of the column containing the names
- **alphacol** – integer, index of the column containing the right ascension
- **deltacol** – integer, index of the column containing the declination
- **obsprogramcol** – integer, index of the column containing the obs program. If not provided, use the provided obsprogram instead.
- **obsprogram** – which :any:'obsprogram.__init__' is to be used as a default if nothing is provided from the imported file.

Note: providing an `obsprogramcol` overloads the given `obsprogram`, as long as there is a valid field in the `rdh obsprogramcol`. You can use both to load a catalogue that has only part of its `programcol` defined.

`obs.showstatus` (*observables, meteo, displayall=True, cloudscheck=True*)
print the observability of a list of observables according to a given `meteo`.

Parameters

- **observables** – list of observables, :meth:`~obs.Observable`
- **displayall** – boolean, if set to True then display the status of all the targets, even those which are not visible.
- **cloudscheck** – boolean, if set to True then use the cloud coverage in the observability computation.

pouet.run module

pouet.util module

Useful functions and definitions

`util.check_value` (*var, flag*)
Check that a value is NaN, replace it with a given flag if True

Parameters

- **var** – value to check against NaN
- **flag** – replacement value

Returns processed value

`util.elev2airmass` (*el, alt, threshold=10.0*)
Converts the elevation to airmass.

Parameters

- **el** – float, elevation in radians
- **alt** – float, altitude of the observer in meters
- **threshold** – maximum allowed airmass, will be returned if actual airmass exceeds the threshold

Returns airmass

Note: This is the code used for the Euler EDP at La Silla.

`util.grid_points` (*res_x=400, res_y=200*)
Maps the whole sky in right ascension and declination

Parameters

- **res_x** – integer, number of points in right ascension
- **res_y** – integer, number of points in declination

Note: the points are equally spaced.

Returns numpy tuples containing right ascension points and declination points

`util.hilite(string, status, bold)`

Helper to add colors and bold in the terminal

Parameters

- **string** – string you want to color or bold
- **status** – boolean, if True then the text is colored in green, otherwise in red.
- **bold** – boolean, if True then the text is bolded

Returns

`util.load_station(name)`

Load the parameters corresponding to an observation station

Parameters **name** – string, name of the station. The corresponding file must be located in `config()`

Returns station parameters

`util.readconfig(configpath)`

Reads in a config file

Parameters **configpath** – path of the configfile

Returns configuration dictionary

`util.readpickle(filepath)`

I read a pickle file and return whatever object it contains. If the filepath ends with .gz, I'll unzip the pickle file.

Parameters **filepath** – string, path of the pickle to load

Returns object contained in the pickle

`util.takeclosest(dico, key, value)`

Warning: I assume that dict[key] is sorted.

Returns the dict value which dict[key] is closest to value. If two dict[key] are equally close to value, return the highest (i.e. latest).

Parameters

- **dico** – python dictionary you want to sort
- **key** – dictionary key used for sorting
- **value** – target value,

Returns index of the element in dico that is the closest to the target value

Note: This is much faster than a simple min loop, although a bit more tedious to use.

`util.time2hhmm (obstime)`

Concatenate a string HH MM SS or HH.MM.SS into an HHMM string

Parameters `obstime` – string, HH MM SS or HH.MM.SS

Returns HHMM string

`util.writepickle (obj, filepath, protocol=-1)`

I write your python object `obj` into a pickle file at `filepath`. If `filepath` ends with `.gz`, I'll use `gzip` to compress the pickle.

Parameters

- `obj` – python container you want to compress
- `filepath` – string, path where the pickle will be written
- `protocol` – Leave `protocol = -1` : I'll use the latest binary protocol of pickle.

pouet.plots module

`plots.plot_airmass_on_sky (target, meteo, ax=None)`

Plots the airmass evolution on the sky of a given target at a given time.

Parameters

- `target` – a `pouet.obs.Observable` class instance
- `meteo` – a `pouet.meteo.Meteo` class instance
- `ax` – the matplotlib axis to plot on. If `None`, then plot on a new figure

`plots.plot_target_on_sky (target, figure=None, northisup=True, eastisright=False, boxsize=None, survey='DSS', cmap='Greys')`

Uses `astroquery` (hopefully soon accessible from `astropy.vo`) to plot an image of the target

`plots.shownightobs (observable, meteo, obs_night=None, savefig=False, dirpath=None, verbose=False)`

Plot the observability of one observable along the night #todo: add the option to be returned in an Axes object instead of plotting

Subpackages

pouet.obsprogram package

Submodules

pouet.obsprogram.prog703 module

`obsprogram.prog703.get_exptime (attributes, obs_time)`

`obsprogram.prog703.observability (attributes, obs_time)`

pouet.obsprogram.prog714 module

`obsprogram.prog714.get_exptime (attributes, obs_time)`

`obsprogram.prog714.observability (attributes, obs_time)`

pouet.obsprogram.progbebop module

obsprogram.progbebop.**get_exptime** (*obj*, *obs_time*)

obsprogram.progbebop.**observability** (*attributes*, *obs_time*)

pouet.obsprogram.proglens module

obsprogram.proglens.**get_exptime** (*attributes*, *obs_time*)

obsprogram.proglens.**observability** (*attributes*, *obs_time*)

Module contents

pouet.LaSilla package

Submodules

pouet.config.LaSilla module

class config.LaSilla.**AllSky**

Bases: object

Station-specific class that handles the all sky image and its transformation of the sky.

Class constructor that saves some important all sky parameters as a class attribute.

get_image_coordinates (*az*, *elev*)

Converts the azimuth and elevation of a target in pixel coordinates

Parameters

- **az** – azimuth (in rad)
- **elev** – elevation (in rad)

Returns x and y position

get_mask (*ar*)

Returns the mask to apply on the AllSky hide unwanted features in the image. In the LaSilla case, to remove the danish and the text in the corners.

Parameters **ar** – original image (or at least an array with the same size). Used to get the image size.

get_radius (*elev*)

Method that computes the radius of a given elevation on the sky, in pixel.

Parameters **elev** – elevation (in radians)

Returns Radius, in px

class config.LaSilla.**WeatherReport** (*name='LaSilla'*)

Bases: object

This class is dedicated to recovering the weather report at the La Silla site and feeding the wind direction, wind speed, temperature and humidity back to pouet. It must contain at least a *get* method that returns the above variable.

Class constructor. Loads the LaSilla.cfg configuration file and saves it as attribute.

Parameters **name** – name of the cfg file, only included for completeness.

get (*debugmode*, *FLAG=-9999*)

Get method that reads the weather reports off the web. In the LaSilla case, it download a *meteo.last* and interprets the data.

Parameters

- **debugmode** – whether or not POUET is in debugmode. If true, it ought to return some static and dummy data
- **FLAG** – what to return in case the weather report cannot be downloaded or treated. Currently, POUET expect -9999 as a placeholder.

Returns Wind direction, speed, temperature and humidity

Warning: Such a method *must* return the following variables in that precise order: wind direction, wind speed, temperature and humidity

Module contents

C

`config`, 36
`config.LaSilla`, 35

O

`obs`, 29
`obsprogram`, 35
`obsprogram.prog703`, 34
`obsprogram.prog714`, 34
`obsprogram.progbebop`, 35
`obsprogram.proglens`, 35

P

`plots`, 34

U

`util`, 32

A

AllSky (*class in config.LaSilla*), 35

C

check_value() (*in module util*), 32

compute_airmass() (*obs.Observable method*), 30

compute_altaz() (*obs.Observable method*), 30

compute_angletomoon() (*obs.Observable method*), 30

compute_angletosun() (*obs.Observable method*), 30

compute_angletowind() (*obs.Observable method*), 30

compute_observability() (*obs.Observable method*), 30

config (*module*), 36

config.LaSilla (*module*), 35

copy() (*obs.Observable method*), 31

E

elev2airmass() (*in module util*), 32

G

get() (*config.LaSilla.WeatherReport method*), 36

get_exptime() (*in module obsprogram.prog703*), 34

get_exptime() (*in module obsprogram.prog714*), 34

get_exptime() (*in module obsprogram.progbebop*), 35

get_exptime() (*in module obsprogram.proglens*), 35

get_image_coordinates() (*config.LaSilla.AllSky method*), 35

get_mask() (*config.LaSilla.AllSky method*), 35

get_radius() (*config.LaSilla.AllSky method*), 35

grid_points() (*in module util*), 32

H

hilite() (*in module util*), 33

I

is_cloudfree() (*obs.Observable method*), 31

L

load_station() (*in module util*), 33

O

obs (*module*), 29

observability() (*in module obsprogram.prog703*), 34

observability() (*in module obsprogram.prog714*), 34

observability() (*in module obsprogram.progbebop*), 35

observability() (*in module obsprogram.proglens*), 35

Observable (*class in obs*), 29

obsprogram (*module*), 35

obsprogram.prog703 (*module*), 34

obsprogram.prog714 (*module*), 34

obsprogram.progbebop (*module*), 35

obsprogram.proglens (*module*), 35

P

plot_airmass_on_sky() (*in module plots*), 34

plot_target_on_sky() (*in module plots*), 34

plots (*module*), 34

R

rdbexport() (*in module obs*), 31

rdbimport() (*in module obs*), 31

readconfig() (*in module util*), 33

readpickle() (*in module util*), 33

S

shownightobs() (*in module plots*), 34

showstatus() (*in module obs*), 32

T

takeclosest() (*in module util*), 33

time2hhmm() (*in module util*), 33

U

`update()` (*obs.Observable method*), [31](#)

`util` (*module*), [32](#)

W

`WeatherReport` (*class in config.LaSilla*), [35](#)

`writepickle()` (*in module util*), [34](#)