OpihiExarata

Release 2022.6.30

Kenji Sparrow Emerson

Jun 30, 2022

USER MANUAL

OpihiExarata is a group of individual software packages, API services, and executables designed to process data from the IRTF Opihi telescope. It serves to be an astrometric solver, photometric calibrator, asteroid finder, ephemeris computer, and astrometric monitor. We accomplish this by integrating services and open source software collections.

This software is specific to the IRTF Opihi telescope.

Note: This documentation comes in two forms, a set of HTML webpages and a LaTeX document. The HTML is given preference when formatting issues arise between the two. We highly suggest that you read the HTML version[1] as it is better formatted, easier to navigate, and more likely to be up to date.

---

[1] https://psmd-iberutaru.github.io/OpihiExarata

CHAPTER

ONE

USER MANUAL

The user manual is primarily for general users, such as telescope operators and observing astronomers. It details how to use OpihiExarata in conjunction with the Opihi telescope, primarily covering the usage of its GUI and command-line interfaces. It can be accessed via the sidebar or by clicking *here*.

1.1 User Manual

This is the user manual portion of the OpihiExarata software package for reducing and analyzing Opihi data.

The Opihi telescope is a smaller telescope mounted on the side of the IRTF. It primarily provides asteroid view-finding and photometric calibration services.

It was conceived to assist the IRTF telescope and its instruments in finding asteroids and other near Earth objects on the sky. This is needed when the positional uncertainty in the ephemeris of these objects are greater than the field of view of the IRTF's current acquisition and tracking instruments (about 1 arcminuite). This can often be the case for newly discovered objects with a small number of observations. With Opihi's 32 arcminuite field of view, these objects can be spotted and the telescope pointing can be corrected, thus greatly reducing the overhead time of finding the target.

However, a minority of time allocated by the IRTF are for these types of targets. Having Opihi only operate a few times a semester when it is needed for acquisition is inefficient. In addition to the asteroid view-finding capabilities, the Opihi telescope also serves as a source for photometric monitoring and calibration. By continuously taking pictures and determining the photometric solution for each, the photometric conditions (as measured by the zero point magnitude) can be monitored over time. Moreover, this photometric data can also be used to assist with photometrically calibrating science targets observed by other IRTF facility instruments (e.g. SpeX, MORIS, SPECTRE, etc).

A more detailed description of the physical and optical specifications of the Opihi telescope can be found in *Opihi Telescope*.

It is typical for the asteroid view-finding to be done manually by either a telescope operator or an observing astronomer; for this reason, we may refer to this view-finding mode as the *Manual Mode* of operation for Opihi and OpihiExarata.

The photometric monitoring mode does not require the constant input from a user, it only requires instructing the Opihi camera to continuously take images and for the software (OpihiExarata) to continuously solve for their respective photometric solutions; for this reason, we may refer to this photometric monitoring mode as the *Automatic Mode* of operation for Opihi and OpihiExarata.

Photometric monitoring may be done manually if the user desires more control over the process. However, asteroid view-finding cannot be done automatically. It is currently beyond the scope of this software to implement automatic asteroid/transient finding.

An astronomer or other user interacts with the Opihi telescope and the OpihiExarata software by connecting to a VNC session; this is common to all IRTF instruments[2]. The Opihi camera controller has an interface and data viewer independent of the interfaces of OpihiExarata.

For the general user, the OpihiExarata presents itself with helpful graphical user interfaces (GUIs). However, a command-line interface is provided to open these GUIs. It is likely that most users do not need to worry about this interface as the GUI may already be open for them in the VNC session, however, the command-line interface and its usage is documented in *Command Line*.

See *Manual Mode* or *Automatic Mode* for usage instructions depending on the desired mode of operation. Both of these have some technical jargon better described in *System Framework*. See *Configuration* for the available configuration options of both modes. If you are having trouble, see *Troubleshooting* for more information and possible solutions. If you believe you found an issue or bug with the software, please report it to the appropriate IRTF staff member(s).

If you used the Opihi telescope or the OpihiExarata software, please acknowledge your usage in any projects or publications; see *Citations* for assistance and for our own references.

[2] http://irtfweb.ifa.hawaii.edu/observing/computer/vnc.php

1.1.1 Opihi Telescope

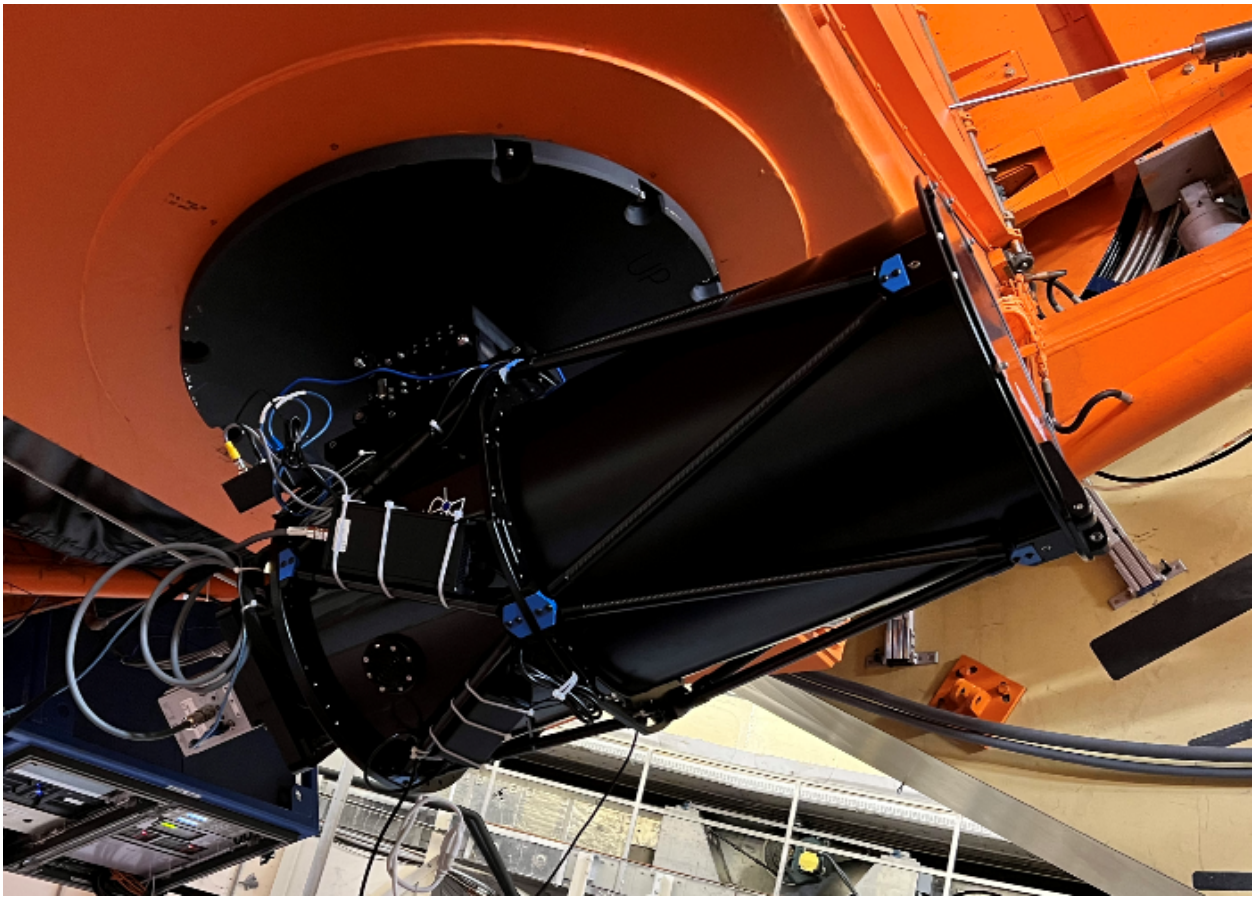

Fig. 1.1: An image of the Opihi telescope during the commissioning process. An aluminum weather proof shield has been since been added which covers the back half of the system to cover the electronics.

We detail the physical specifications of the Opihi telescope itself. As the *Technical Manual* is more for detail about the software, and the actual telescope's physical quantities may be useful for users, we leave this section here in the user manual.

See Fig. 1.1 for an image of the Opihi telescope itself.

The Opihi telescope:

- Is a 0.43 meter corrected Dall-Kirkham telescope from PlaneWave Instruments.
- It has a field of view of 32 arcminuites across.
- The sensitivity is approximately 20 magnitudes in a one minute exposure.
- Has a pixel scale of 0.94 arcseconds per pixel.
- The filter wheel contains four SDSS filters (g', r', i', z'), a clear position, and three (currently unused) slots for other filters.

The camera/detector on the Opihi telescope:

- Is a 2k x 2k Andor iKon-L 936 CCD Camera.
- The operating temperature is -50 Celsius (223.15 K).
- The dark current is about 1 electron/s.
- The read noise is about 10.3 electron/s.
- The well depth is about 150 000 electrons.

Note: Most, if not all, of the information provided on this page/section is taken from the Opihi telescope and OpihiExarata SPIE conference paper (see *Citations*) or the IRTF Opihi website[3]. You may find more information from these resources.

1.1.2 System Framework

We briefly cover a few terms and processes necessary for usage of this software. This does not go into more detail than is needed by a standard user; for more information, see the *Technical Manual*.

Engines

For processing of Opihi data, there are four problems which the OpihiExarata does not solve on its own (or does not have the data to do so). They are:

- Astrometric plate solving of an image.
- Photometeric calibration.
- Preliminary orbit determination.
- Ephemeris computations.

Additionally, there is another problem which is implemented similarly as if OpihiExarata could not solve it on its own:

- On-sky target propagation.

These five problems are solved by sending relevant data to other services, utilizing their APIs to compute a solution. (See our *Citations* for more references.)

Each of these services are made by different organizations. We access the capabilities of theses service by what we call an "engine". Each service has their own custom implemented engine. Selecting a given engine (as you use this program) means that your data will be processed by the service the engine corresponds to.

You can find out about the different available services (and thus engines) in *Services and Engines*.

[3] http://irtfweb.ifa.hawaii.edu/~opihi/

Vehicles and Solutions

There may be multiple engines for solving one of the five problems. Different datasets may be predisposed to work better with one engine over another. As an implementation detail, we also built wrappers around the engines called vehicle functions and solution classes for ease of use. More detail about these wrappers can be found in *Vehicles and Solutions*, but it likely is of little concern to the average user.

Conceptually, you may think of the engine as synonymous with the service; the rest are just unnecessary details.

1.1.3 Command Line

The OpihiExarata software system should already be installed on the computer Opihi operates off of. (If that is not the case, consult *Installation* for more information or contact a staff member.) To test that OpihiExarata is installed on the system, you can run the following in a terminal:

```
opihiexarata
```

This should print a help menu of sorts.

The general overview of the command-line syntax of this command is:

```
opihiexarata [action] [options]
```

Where *[action]* is the specified actions to take. There are many actions which the command-line interface may execute. The currently available actions are detailed in *Available Actions*.

Different command-line options, *[options]*, are detailed in *Available Options*. The options detailed are all optional. The only mandatory input is the required action to take.

Available Actions

[action]

Here we list the available actions for the command-line interface, ordered by (approximately) the order of importance each command is to the average user.

Note that there are aliases for different actions for ease of use, they are all listed here.

Manual

manual, m

This opens up the manual mode GUI. This is used when the user wants to utilize the manual (asteroid view-finding) of Opihi and OpihiExarata. Or, alternatively, the user wants to manually operate the photometric monitoring mode.

Automatic

automatic, auto, a

This opens up the automatic mode GUI. This is used when the user wants to utilize the automatic (photometric monitoring) mode of Opihi and OpihiExarata. Typically, this user would be a telescope operator.

Generate

generate, g

This generates configuration files to allow a user to edit and customize the functionality of OpihiExarata. There are two different configuration files. Which type of configuration file, and where it is saved are determined by the specification of the optional parameters For more information about configuration and configuration files, see *Configuration*.

Help

help, h

This displays the help dialog in the terminal. It is identical to the `--help` option or invoking **opihiexarata** without any specified action.

> **Warning:** If any action is specified that does not match any of the expected actions, the program will raise a Python exception as opposed to ignoring it or raising a shell error.

Available Options

[options]

Here we list the available actions for the command-line interface, ordered by (approximately) the order of importance each command is to the average user.

Note that there are aliases for different actions for ease of use, they are not listed here. Instead, the overall preferred syntax and formatting is provided. To see the aliases, see the help screen for more information: *Help*.

Help

`--help`

This overrides any specified actions and executes the help dialog. See *Help*.

Manual

`--manual`

This opens up the manual mode GUI regardless of the action specified. See *Manual*.

Automatic

`--automatic`

This opens up the automatic mode GUI regardless of the action specified. See *Automatic*.

Configuration

`--config``=<path/to/config.yaml>`

This specifies the path of the configuration file. The configuration file is in a YAML format. If the action specified is `generate`, then this is the path where the generated configuration fill will be saved. Otherwise, the program will read the configuration file at this path and use its values instead of the program's defaults, where they differ.

See *Standard Configuration File* for the specifications of the configuration file.

Secrets

`--secret``=<path/to/secret.yaml>`

This specifies the path of the secrets file. The secrets file is in a YAML format. If the action specified is `generate`, then this is the path where the generated secrets fill will be saved. Otherwise, the program will read the secrets file at this path and use its values instead of the program's defaults, where they differ.

See *Secrets Configuration File* for the specifications of the secrets file.

Overwrite

`--overwrite`

This option allows for the specification of what to do when a provided file at a path already exists. This is typically used when generating new configuration files. When provided, any pre-existing files are overwritten. The default, when this option is not provided, is to raise an error because a file already exists at a given path.

Keep Temporary

`--keep-temporary`

The normal operation of OpihiExarata requires the writing of temporary files. A temporary directory is created (as specified by the configuration file) and is then purged and deleted. Using this flag prevents the cleanup of the temporary directory on the program's exit. However, as the software itself often cleans up after itself periodically during usage, this option is not very useful for the end user. It is more for debugging purposes.

Warning: If any option is specified that does not match any of the expected option, the program will raise a shell error.

1.1.4 Manual Mode

The manual mode of OpihiExarata is its asteroid view-finding mode. Although, photometric monitoring can be done manually, and the manual mode is well equipped for said use case, it is not the primary use case for the manual mode.

The user uses the Opihi instrument to take images of an asteroid. Then they specify the location of the asteroid using a GUI (usually with help from comparing their image to a reference image). An astrometric solution can convert the asteroid's pixel location to an on-sky location. Then by utilizing previous observations, its path across the sky can be determined. This information is used to properly point the IRTF telescope to the desired asteroid target.

We present the procedure for operating OpihiExarata in its manual mode, please also reference the GUI figure provided for reference. We also summarize the procedure and process of the manual mode via a flowchart.

Graphical User Interface

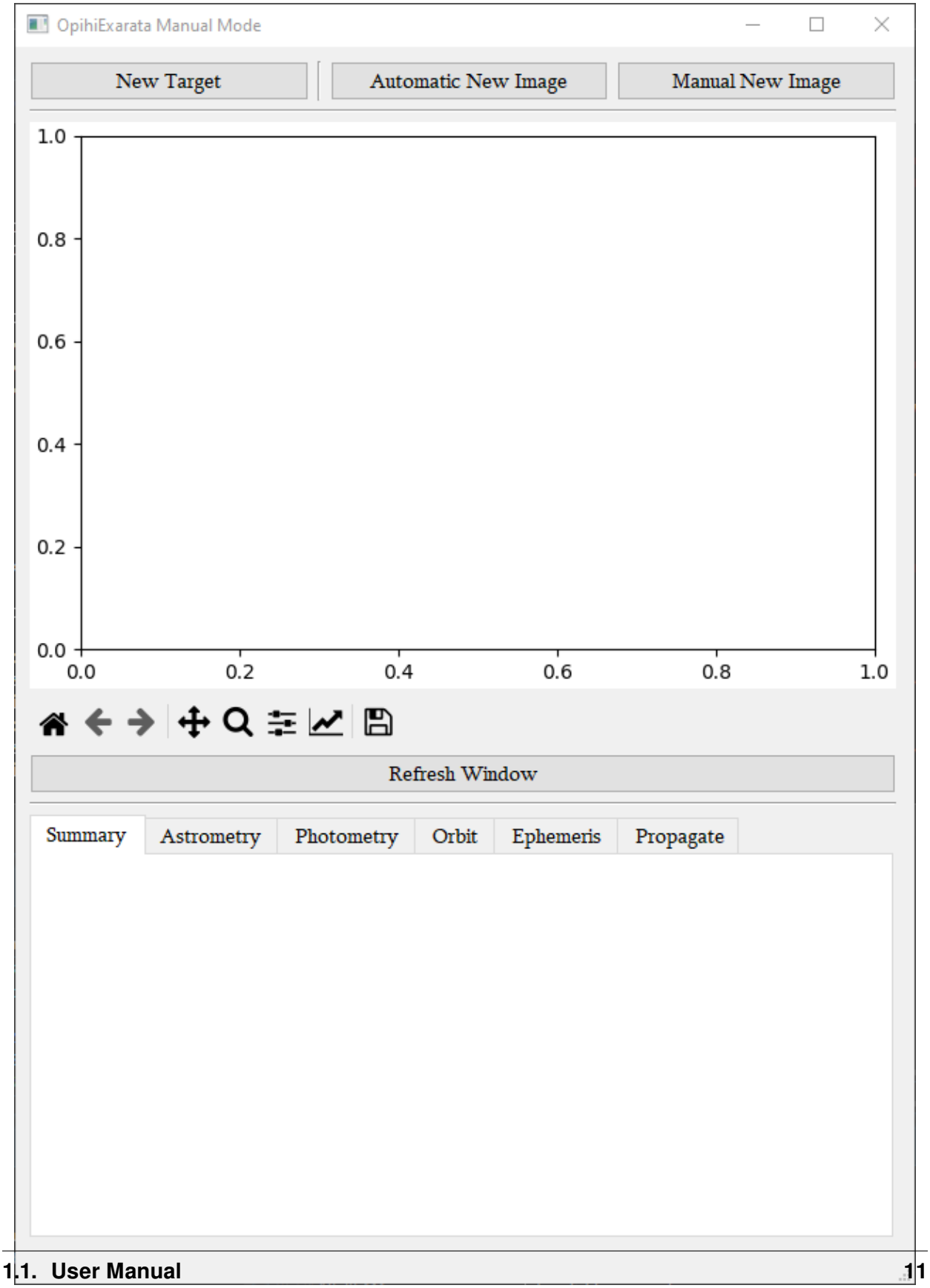

Fig. 1.2: The GUI of the manual mode of OpihiExarata. This is what you see when the interface is freshly loaded. Most of the text is filler text, they will change as the program is executed. Note that there are different

The manual mode GUI contains a main data window and a few tabs compartmentalizing a lot of the functionality of OpihiExarata, see Fig. 1.2.

In the manual mode general GUI, you have three buttons which control file and target acquisition. The *New Target* button specifies that you desire to work on a new target (typically a new asteroid). The *Manual New Image* button opens a file dialog so that you can select a new Opihi image FITS file to load into OpihiExarata. The *Automatic New Image* button pulls the most recent FITS image from the same directory as the last image specified manually.

There is also a data viewer. Relevant information from the solutions will be plotted here along with the image data. There is a navigation bar for manipulating the image, including zooming, panning, saving, and configuring other options. This functionality makes use of matplotlib, see Interactive navigation (outdated)[4] or Interactive figures[5] for more information.

There is also a *Refresh Window* button to redraw the image in the viewer and to also refresh the information in the tabs. This should not be needed too much as the program should automatically refresh itself if there is new information.

[4] https://matplotlib.org/3.2.2/users/navigation_toolbar.html
[5] https://matplotlib.org/stable/users/explain/interactive.html

Procedure

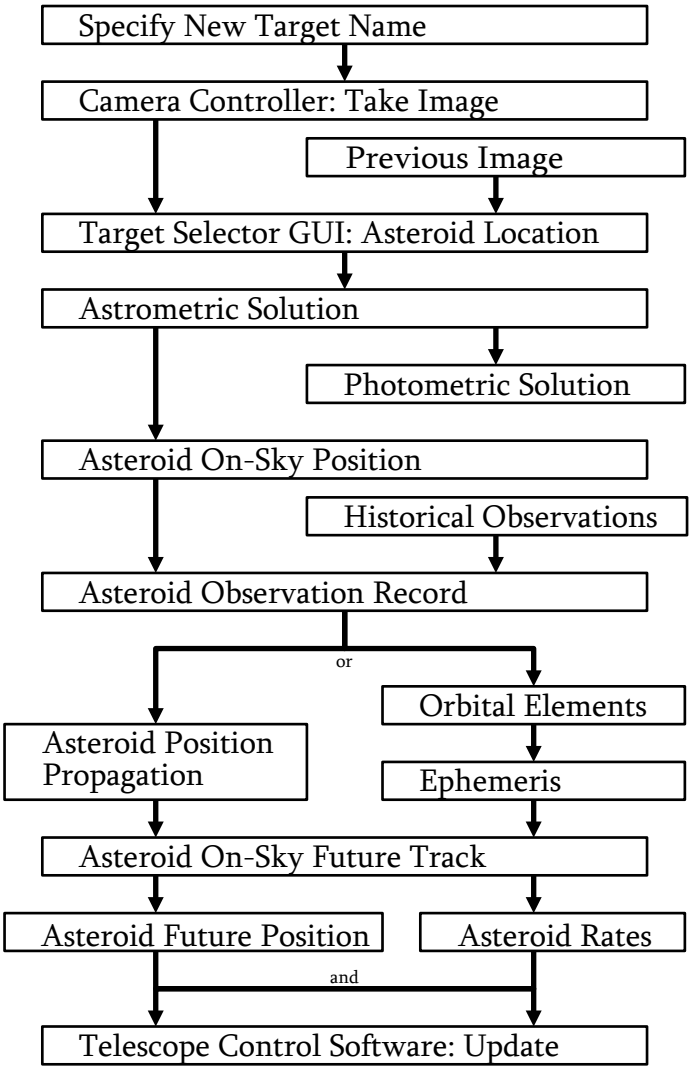

Fig. 1.3: A flowchart summary of the procedure of the manual mode. It includes the actions of the user along with the program's flow afterwards.

We describe the procedure for utilizing the asteroid view-finding (manual) mode of OpihiExarata. See Fig. 1.11 for a flowchart summary of this procedure.

Start and Open GUI

You will want to open the OpihiExarata manual mode GUI, typically via the command-line interface with:

```
opihiexarata manual --config=config.yaml --secret=secrets.yaml
```

Please replace the configuration file names with the correct path to your configuration and secrets file; see *Configuration* for more information.

Specify New Target Name

The OpihiExarata manual mode software groups sequential images as belonging to a set of observations of a single target. In order to tell the software what target/asteroid you are observing you need to provide its name.

Use the *New Target* button, it will open a small prompt for you to enter the name of the asteroid/target you are observing. From this point on, all images loaded (either manually or automatically fetched) will be assumed to be of the target you provided (until you change it by clicking the button again).

Note: If you are observing an asteroid, this target name should also be the name of the asteroid. The software will attempt to use this name to obtain historical observations from the Minor Planet Center.

Take and Image

The first step in processing data is to take an image using the camera controller.

Note: It is highly suggested that you take two images if this is the first image you are taking of a given specified asteroid. This allows you to have a reference image which makes asteroid finding much easier. You process the first image normally (using the second image as the reference image) then process the second image (using the first image as the reference image). It does not matter which is the reference image for more images taken. (See *Find Asteroid Location*.)

Find Asteroid Location

You will need to find and specify the location of the asteroid in the image. It is beyond the scope of this software and procedure to implement this automatically. (If you are not observing asteroids, you may skip this step and just click *Submit* in the *Target Selector GUI*.)

Use the *Target Selector GUI* to find the asteroid pixel location.

Target Selector GUI

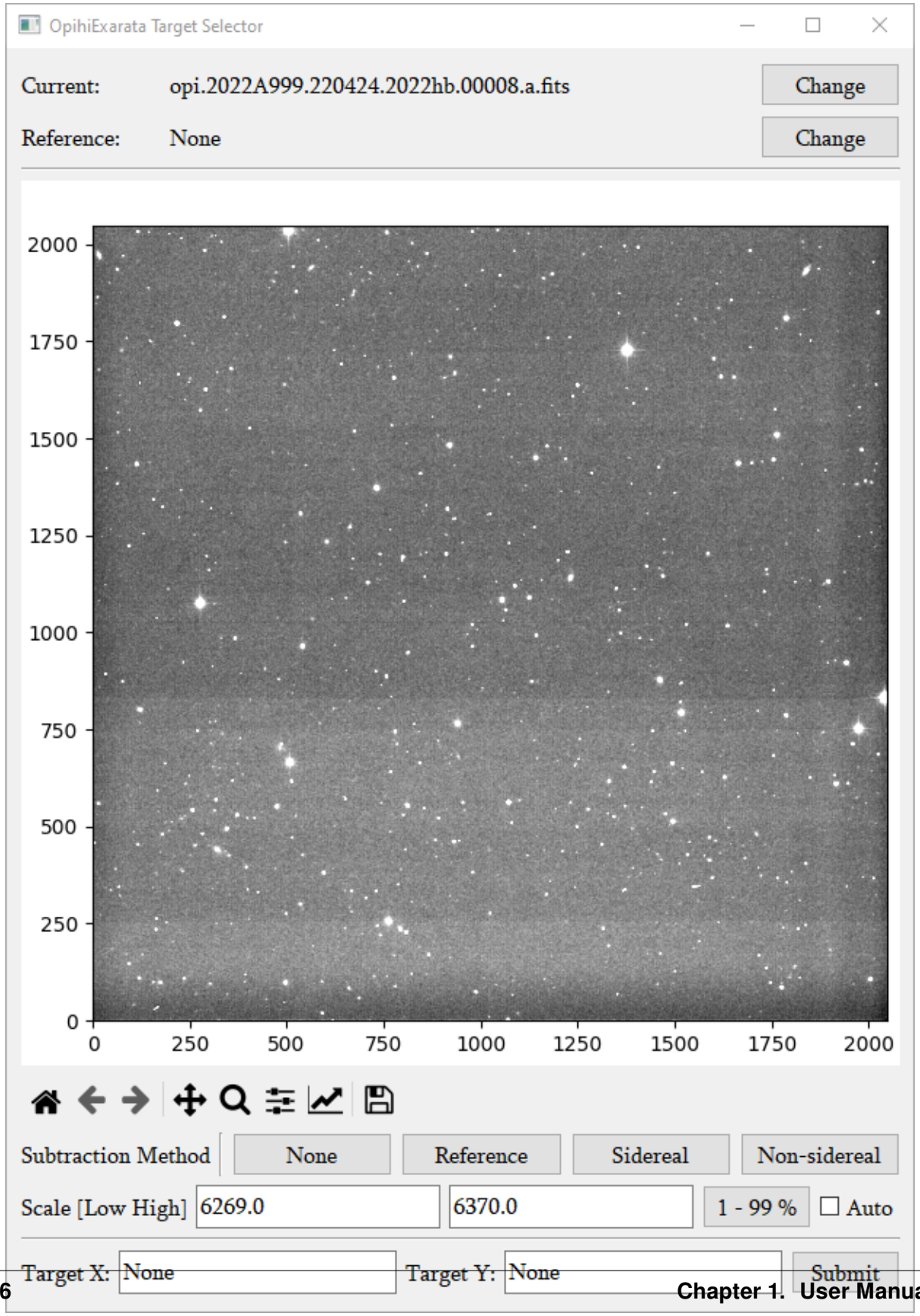

Fig. 1.4: The GUI for finding the pixel location of a target in the image. The targets are typically asteroids.

The target selector GUI allows you to select a specific target (generally an asteroid) location in an image, see Fig. 1.4.

The current file which you are determining the location of a target in is given by *Current:*. The reference image (if provided) used to compare against is given by *Reference:*. Both of these files can be changed using their respective *Change* buttons; a file dialog will be opened so you can specify the new FITS files.

There is a data viewer similar to the one specified in *Graphical User Interface*. However, in addition, if you drag a box (left click and hold, drag, then release) without any tool selected in toolbar, the software will search within the drawn (blue) box and extract the brightest object within the box. It will mark this target with a red triangle. It will assume that this is the desired target and update the *Target X* and *Target Y* fields with its pixel coordinates.

Note: This box drawing method finds the brightest object in the current image. It ignores the subtractive comparison method and its result as such comparisons do not affect the actual current image.

You can compare your current image file C with your reference image R file in two subtractive ways using the two labeled buttons under *Subtraction Method*. (There are also buttons for simply viewing the images.) Therefore, the two (plus two) ways of viewing the data are:

- *None*, $C - 0$: The current image is not compared with the reference image.
- *Reference*, $R - 0$: The reference image is shown rather than the current image.
- *Sidereal*, $C - R$: The two images are subtracted assuming the IRTF is doing sidereal tracking. Because of this assumption, no shifting is done.
- *Non-sidereal*, $C - T_v(R)$: The two images are subtracted assuming the IRTF is doing non-sidereal tracking. Because of this assumption, the images are shifted based on the non-sidereal rates of the current image and the time between the two images.

The displayed image's color bar scale can be modified manually by entering values into the boxes accompanying *Scale [Low High]*, the left and right being the lower and higher bounds of the color bar respectively as indicated. The scale can also be automatically set so that the lower bound is the 1 percentile and the higher bound is the 99 percentile by clicking the *1 - 99 %* button. If the *Auto* checkbox is enabled, this autoscaling is done whenever a new operation is done the image (i.e. using the tools in the toolbar, changing the comparison method, among others).

Select your target, either from the box method or by manually entering the coordinates in the *Target X* and *Target Y* boxes, and click *Submit*. The location of your target will be recorded.

Compute Astrometric Solution

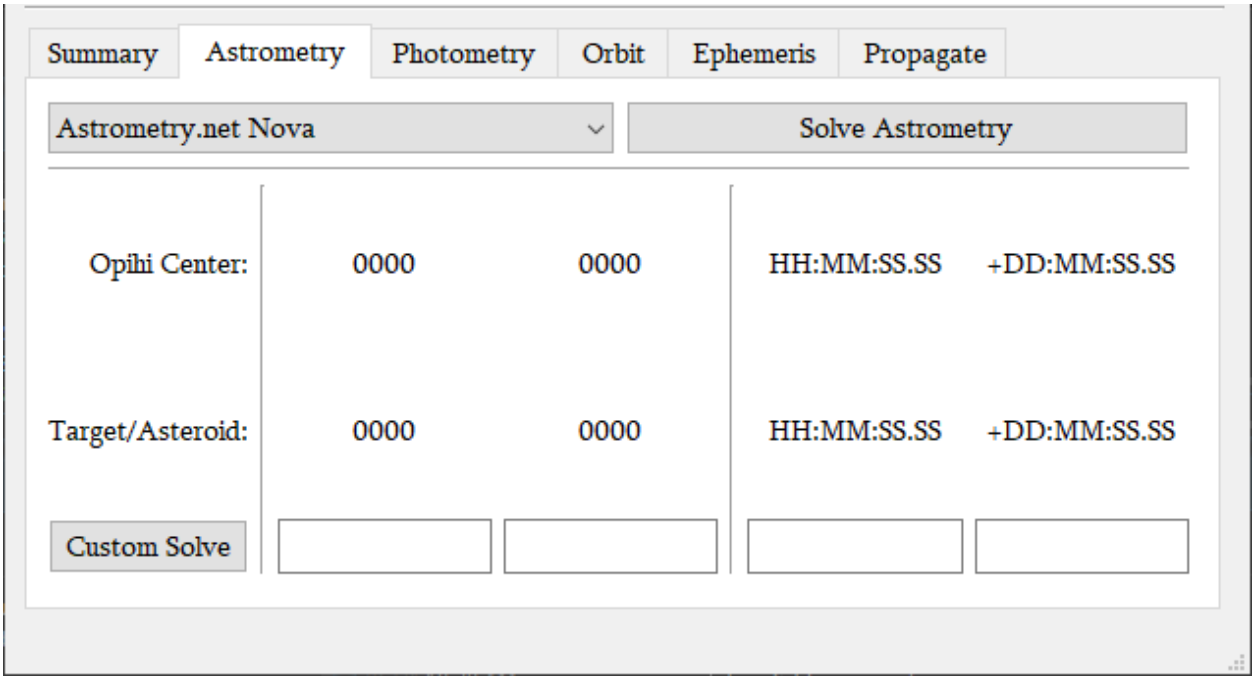

Fig. 1.5: The astrometry GUI tab for customizing and executing astrometric solutions. This is the default view before any values have been calculated.

The astrometric solution of the image is next to be solved. The pattern of stars within the image is compared with known patterns in astrometric star databases to derive the WCS[6] astrometric solution of the image. See Fig. 1.5 for the interface for astrometric solutions.

To solve for the astrometric solution of the image, you will need to select the desired astrometric engine from the drop down menu then click on the *Solve Astrometry* button to solve. (See *Services and Engines* for more information on the available engines.)

The pixel location (X,Y) of the center of the image, given by *Opihi Center*, and the specified target, given by *Target/Asteroid*, is provided with or without an astrometric solution. When the astrometric solution is provided, the right ascension and declination of these will also be provided.

Custom pixel coordinate (X,Y) can be provided in the boxes to be translated to the sky coordinates that they correspond to. Alternatively, if sky coordinates are provided (in sexagesimal form, RA hours and DEC degrees, delimitated by colons), the pixel coordinates of the sky coordinates can also be determined; the pixel coordinate boxes must be empty as the solving gives preference to pixel to on-sky solving. Enter in either pixel or sky coordinates as described and click the *Custom Solve* button to convert it to the other. The button does nothing without a valid astrometric solution.

[6] https://fits.gsfc.nasa.gov/fits_wcs.html

Compute Photometric Solution

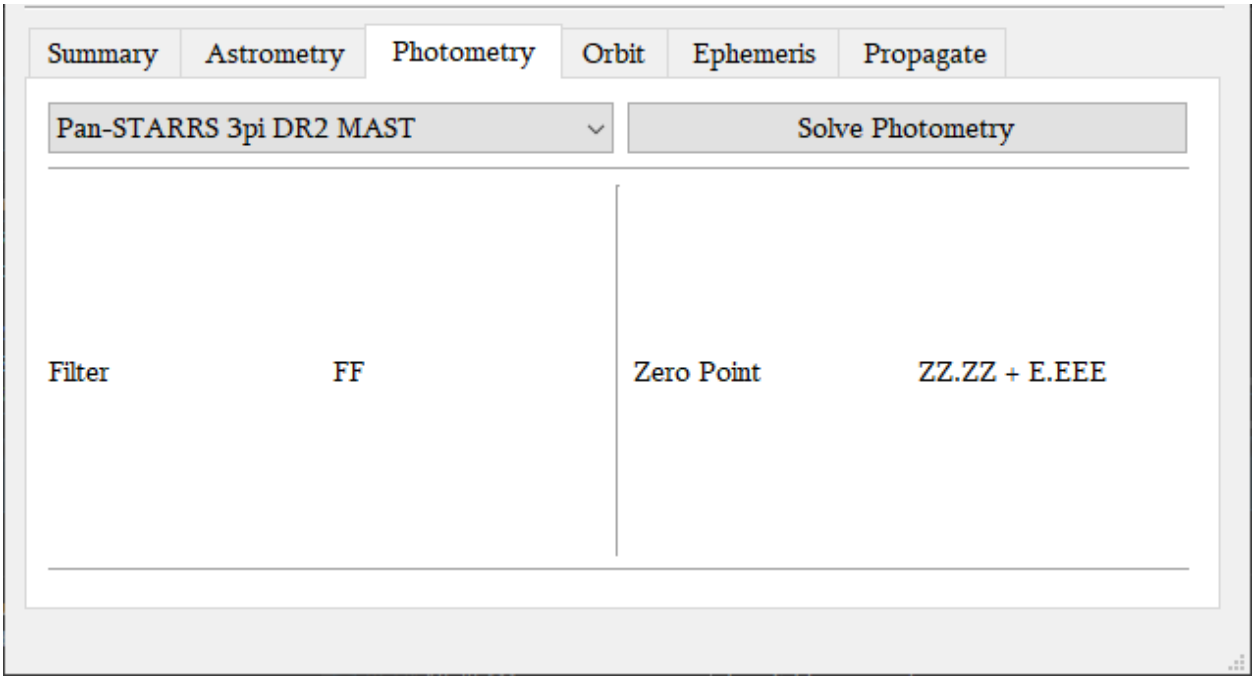

Fig. 1.6: The photometry GUI tab for customizing and executing photometric solutions. This is the default view before any values have been calculated.

The photometric solution of the image is next to be solved. The brightness of the stars in the image is compared to known filter magnitudes from a photometric database to derive a photometric calibration solution. See Fig. 1.6 for the interface for photometric solutions.

This is an optional step and is not related to asteroid finding in of itself. This operation can be skipped entirely if a photometric solution is not necessary.

To solve for the photometric solution of the image, you will need to select the desired photometric engine from the drop down menu then click on the *Solve Photometry* button to solve. (See *Services and Engines* for more information on the available engines.)

The filter that the image was taken in is noted by *Filter*, this is determined by the FITS file header.

Once a photometric solution has been solved, the corresponding filter zero point magnitude (and its error) of the image is provided by *Zero Point*.

Note: Execution of the photometric solution requires a completed astrometric solution from *Compute Astrometric Solution*.

Asteroid On-Sky Position

The asteroid pixel location is derived from the procedure in *Target Selector GUI* and the corresponding on-sky location is derived from the procedure in *Compute Astrometric Solution*.

Historical Observations

The software will attempt to use the target/asteroid name provided in *Specify New Target Name* to obtain the set of historical observations from the Minor Planet Center.

Recently taken images will also be considered part of the set of historical observations.

Asteroid Observation Record

The combination of both *Historical Observations* and *Asteroid On-Sky Position* makes up the sum total of the asteroid observation record. Using this asteroid observation record, the future path of the asteroid on the sky can be determined to eventually allow for the proper acquisition.

There are two different procedures for determining the future track of the asteroid:

- Propagating the on-sky motion of the asteroid into the future.
- Solving for the orbital elements and deriving an ephemeris.

Both options are sufficient but we recommend *Asteroid Position Propagation*.

Asteroid Position Propagation

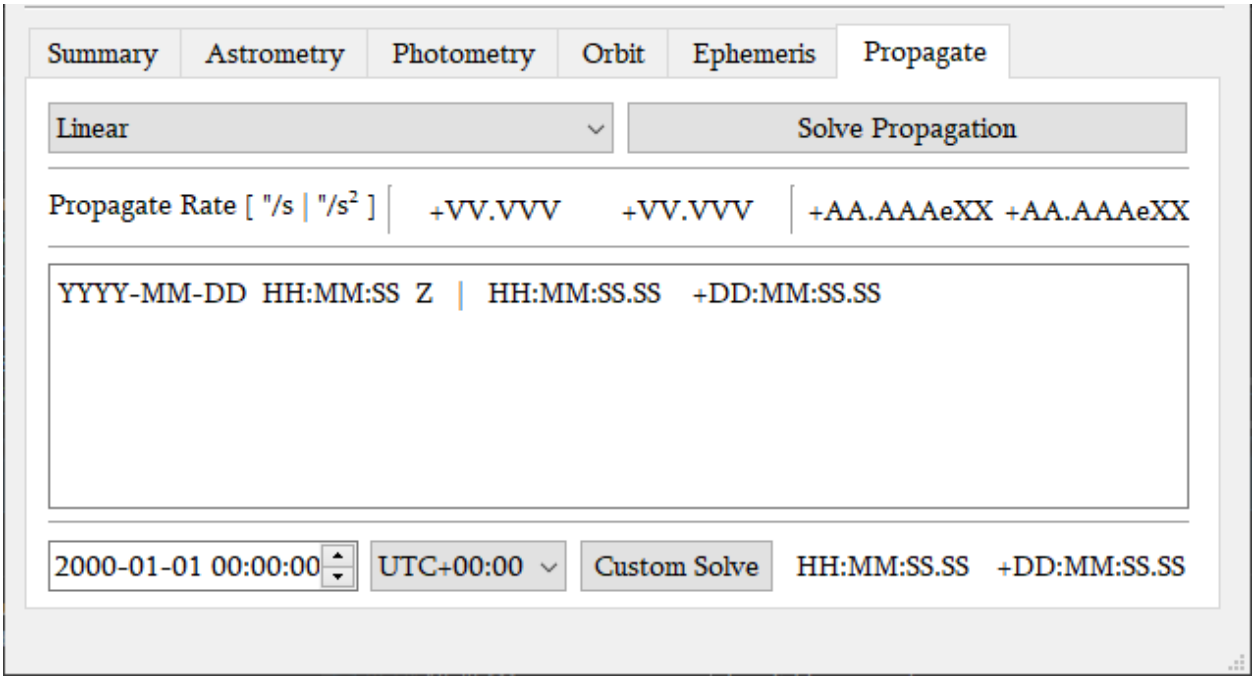

Fig. 1.7: The propagation GUI tab for customizing and executing propagation solutions. This is the default view before any values have been calculated.

Propagating the on-sky motion of the asteroid is done by taking the observational record from *Asteroid Observation Record* and propagating only the most recent observations forward in time. See Fig. 1.7 for the interface for propagation solutions.

To solve for the propagation solution from the observations, you will need to select the desired propagation engine from the drop down menu then click on the *Solve Propagation* button to solve. (See *Services and Engines* for more information on the available engines.)

If a propagation solution is done, the on-sky rates will be provided under *Propagate Rate ["/s | "/s²]*. Both the first order (velocity) and second order (acceleration) on-sky rates in RA and DEC are given in arcseconds per second or arcseconds per second squared. The RA is given on the right and DEC on the left within the first or second order pairs.

You may also provide a custom date and time, in the provided dialog box (using (ISO-8601 like formatting[7]). You can specify the timezone that the provided date and time corresponds to using the dropdown menu. When you click *Custom Solve*, the displayed RA and DEC coordinates are the estimated sky coordinates for the asteroid at the provided input time.

Note: Execution of the propagation solution requires a completed astrometric solution from *Compute Astrometric Solution*.

---

[7] https://www.iso.org/standard/70907.html

Orbital Elements

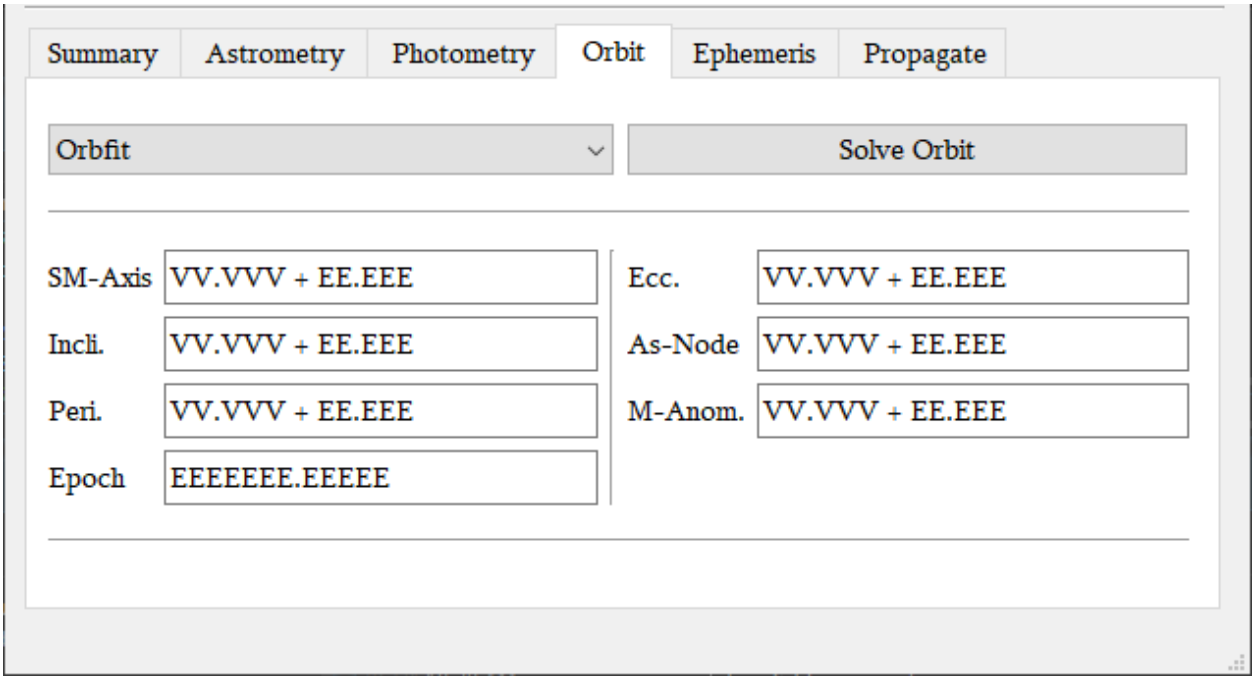

Fig. 1.8: The orbit GUI tab for customizing and executing orbital solutions. This is the default view before any values have been calculated.

Provided a list of historical observations, we can solve for the Keplerian orbital elements using preliminary orbit determination for osculating elements. See Fig. 1.8 for the interface for orbital solutions.

To solve for the orbital solution from the observations, you will need to select the desired orbit engine from the drop down menu then click on the *Solve Orbit* button to solve. (See *Services and Engines* for more information on the available engines.)

The six Keplerian orbital elements (plus the epoch) are provided after the orbital solution is solved. They are:

- *SM-Axis*: The semi-major axis of the orbit, this is in AU.
- *Ecc.*: The eccentricity of the orbit, this is unit-less.
- *Incli.*: The inclination of the orbit, in degrees.
- *As-Node*: The longitude of the ascending node, in degrees.
- *Peri.*: The argument of perihelion, in degrees.
- *M-Anom.*: The mean anomaly, in degrees.
- *Epoch*: The epoch of these of these osculating orbital elements, in Julian days.

If the Engine provided is *Custom*, then you are trying to provide a custom orbit. You provide your Keplerian orbital parameters in the boxes. You may also specify the error in these elements by providing another number delimitated from the first by a letter. (Note, scientific notation is not supported, especially E-notation based entries.) After you provide your orbital parameters, you can click *Solve Orbit* to *solve* for your orbital solution.

Note: Execution of the orbital solution requires a completed astrometric solution from *Compute Astrometric Solution*.

Ephemeris

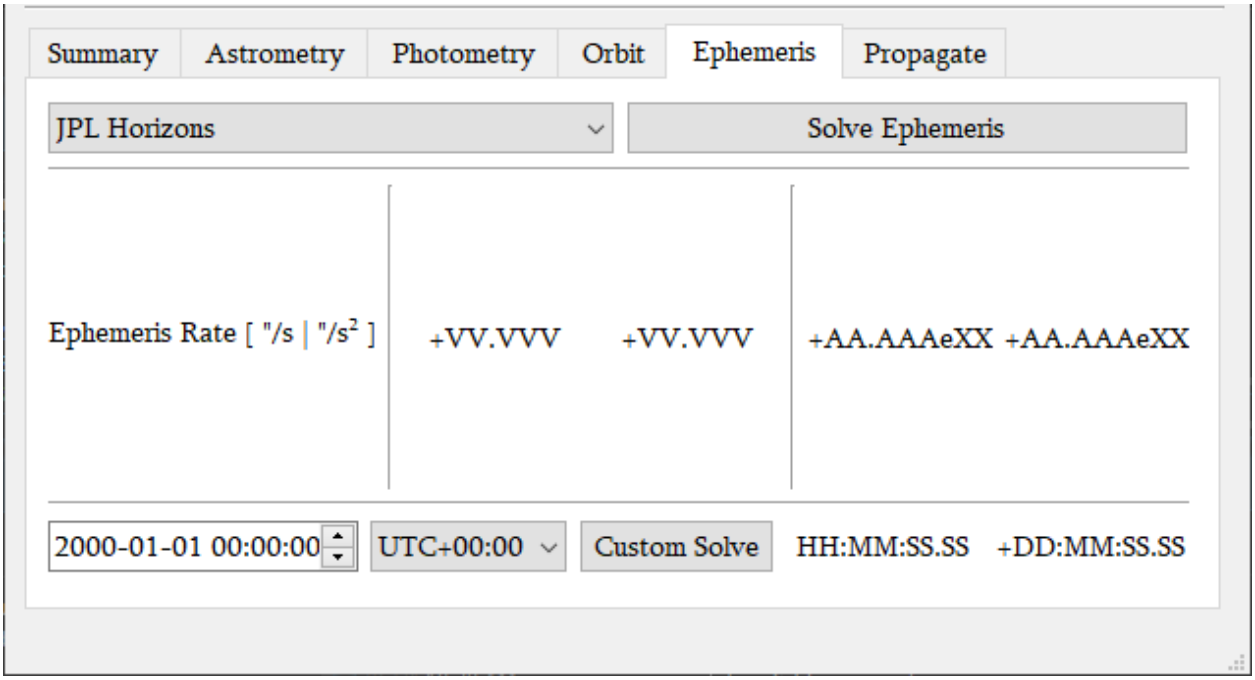

Fig. 1.9: The ephemeris GUI tab for customizing and executing ephemeris solutions. This is the default view before any values have been calculated.

The orbital elements derived in *Orbital Elements* can then be used to derive an ephemeris of an asteroid. See Fig. 1.9 for the interface for ephemeris solutions.

To solve for the ephemeris solution from the orbital elements, you will need to select the desired ephemeris engine from the drop down menu then click on the *Solve Ephemeris* button to solve. (See *Services and Engines* for more information on the available engines.)

If an ephemeris solution is done, the on-sky rates will be provided under *Ephemeris Rate ["/s | "/s²]*. Both the first order (velocity) and second order (acceleration) on-sky rates in RA and DEC are given in arcseconds per second, or arcseconds per second squared. The RA is given on the right and DEC on the left within the first or second order pairs.

You may also provide a custom date and time, in the provided dialog box (using (ISO-8601 like formatting[8]). You can specify the timezone that the provided date and time corresponds to using the dropdown menu. When you click *Custom Solve*, the displayed RA and DEC coordinates are the estimated sky coordinates for the asteroid at the provided input time.

[8] https://www.iso.org/standard/70907.html

Note: Execution of the ephemeris solution requires a completed orbital solution from *Orbital Elements* which itself depends on a completed astrometric solution from *Compute Astrometric Solution*.

Asteroid On-Sky Future Track

Regardless of which method you use to derive the future track of the asteroid (either from *Asteroid Position Propagation* or from *Orbital Elements* and *Ephemeris*), the future position of the asteroid and the on-sky rates are determined (see the respective sections for details).

Telescope Control Software: Update

The new asteroid on-sky future track (position and on-sky rates) derived from *Asteroid On-Sky Future Track* can be sent to the telescope control software to slew the telescope to the correct location of the asteroid.

1.1.5 Automatic Mode

The automatic mode of OpihiExarata is its atmospheric monitoring mode. It is best operated by its dedicated graphical user interface.

Images, continuously taken by the Opihi telescope and camera, are photometrically solved such that their zero points are calculated. The program does this automatically after the initial specifications and trigger.

We present the procedure for operating OpihiExarata in its automatic mode; please also reference the GUI figure provided for reference. We also summarize the procedure and process of the automatic mode via a flowchart.

Graphical User Interface

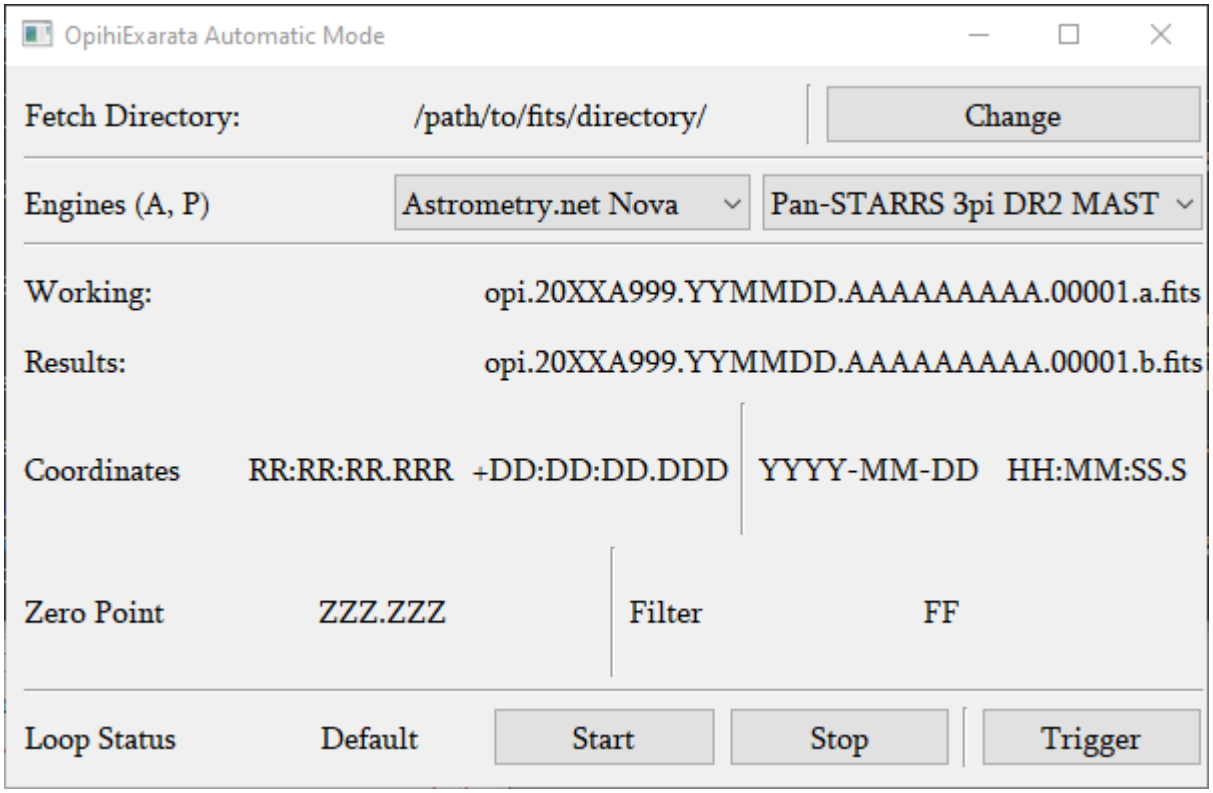

Fig. 1.10: The GUI of the automatic mode of OpihiExarata. This is what you see when the interface is freshly loaded. Most of the text is filler text, they will change as the program is executed.

The graphical user interface of the automatic mode of OpihiExarata, see Fig. 1.10. More information is detailed in the respective subsections of this section.

Procedure

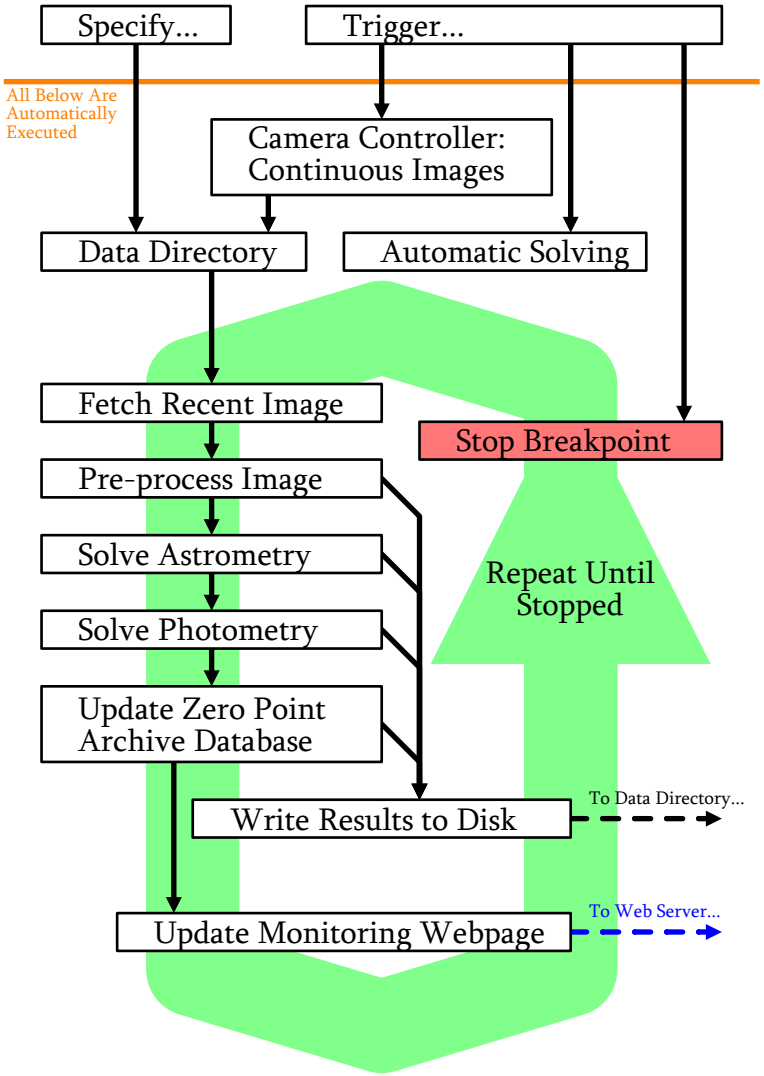

Fig. 1.11: A flowchart summary of the procedure of the automatic mode. It includes the actions of the user along with the program's flow afterwards.

We describe the procedure to configuring and starting and stopping the automatic mode of OpihiExarata. See Fig. 1.11 for a flowchart summary of this procedure.

Start and Open GUI

You will want to open the OpihiExarata automatic mode GUI, typically via the command-line interface with:

```
opihiexarata automatic --config=config.yaml --secret=secrets.yaml
```

Please replace the configuration file names with the correct path to your configuration and secrets file; see *Configuration* for more information.

Specify Data Directory

You will need to specify the data directory that the OpihiExarata program will pull from the most recent images Opihi has taken. This should be the same directory where the Opihi camera is saving the images it is taking constantly.

You can specify this directory by clicking the *Change* button to bring up a directory selection dialog. The exact form of the dialog is operating system dependent as we defer to their implementation. Select the directory using that dialog.

Once submitted, the *Fetch Directory* entry should change to match your selection. This display uses absolute paths; ensure that it is correct.

Specify Solving Engines

You will need to specify the solving engines that OpihiExarata will use to in order to solve the images. For this mode, only the AstrometryEngine(s) and PhotometryEngine(s) are considered as it does not use any other engines. The selected engines will constantly be used until a different one is specified.

Specify the desired AstrometryEngine using the left drop-down combo box. Specify the desired PhotometryEngine using the right drop-down combo box. The *Engines (A, P)* label has also small key to tell you which is which. For more information on the available engines, see *Services and Engines*

Trigger Continuous Images

You will need to trigger the Opihi camera to continuously take images. Please consult the Opihi camera controller documentation for more information.

(A little supporting information about operating the camera controller will be posted here when more information is available.)

Ensure that the directory that these continuous images are being saved to is the same directory you specified for this program to fetch from.

Trigger Automatic Solving

You will need to start automatic solving by triggering it via the *Start* button. Once it starts, it will do the following, in order, automatically, without the need for user intervention:

1. It will fetch the most recent fits image as determined by the modification timestamp of all fits files within the specified directory as provided by the operating system.
2. It will pre-process the fetched image according to the preprocessing algorithm, see *PreprocessSolution*.
3. It will solve for the astrometric solution of the pre-processed image via the specified AstrometryEngine. The results of the solution will be displayed in the GUI.
4. It will solve for the photometric solution of the pre-processed image via the specified PhotometryEngine. The results of the solution will be displayed in the GUI.
5. It will add this result (the filter zero point measurement) to the archive of observations. The monitoring webpage uses this archive to derive its figures.
6. The results from the pre-processing of the raw image and the subsequent engine solves will be saved to disk under a similar name to the original raw file and in the data directory specified.
7. It will repeat this process until stopped.

Trigger Once Manually

You may also trigger the automatic solving procedure manually once via the *Trigger* button. This will do the entire process as elaborated above, except for repeating it.

Displayed Status and Results

As each image is solved, the results of the solve will be displayed. We describe the fields which change as the automatic solving runs.

The *Working* field details the file that is currently being (or was last) worked on by the automatic solving procedure. The *Results* field details the file that has already been worked on and that has been solved without failure. When the working file, undergoing the solving, is solved successfully, it becomes a file with results and the program designates it as such. If the working file failed to solve, it is not transferred over.

The astrometric and photometric results of the *Results* file is displayed as well.

The *Coordinates* felids specify the on-sky right ascension and declination of the center of the image along with the UTC time of when this image was taken.

The *Zero Point* value, calculated via the photometric solver, of the image is provided along with the *Filter* that the image was taken in specified in said field. (The filter term is based on the fits header metadata.)

The status of the automatic solving will be displayed under *Loop Status*. The possible statuses, and their meanings, are:

- *Running*: The automatic loop of fetching and solving images is currently running.

- *Stopped*: The automatic loop of fetching and solving images is stopped.
- *Triggered*: The solving of a single image has been triggered and it is being worked on.
- *Failed*: An image in the automatic loop failed to solve, but this does not stop the loop.
- *Halted*: The loop has been stopped via an alternative method than the *Stop* button.
- *Default*: This is filler text when the GUI is first opened. This should not reappear throughout usage.

Stop Automatic Solving

When you want to stop the automatic solving, you can click the *Stop* button at any time. This will finish the current image it is working on and stop the automatic loop from fetching another image from the data directory. Because of the nature of sending information to external services (i.e. the backends to the engines), the solving of an image cannot be gracefully stopped mid-way and so the process must finish and we only prevent it from repeating.

If you want to stop the solving immediately for whatever reason, it is suggested to cancel or crash the process that OpihiExarata is running on.

The *Stop* button will not prevent a manual trigger from being executed via the *Trigger* button.

If the infinite automatic loop continues fetching images even after the stop button is pressed, this likely means something was changed in the code and the original logic failed. A solution to stop the loop is detailed in *Automatic Mode Stop Button Not Working*.

1.1.6 Configuration

There are many different configuration options available to the user to modify the OpihiExarata software. The configuration options are split into two different configuration files, detailed here.

If these configuration files to not exist on the system, you can generate a new configuration file, which contains the defaults, using the `generate` action from the command-line interface, see *Generate*.

Standard Configuration File

A standard configuration file (simplified to configuration file), typically called something like `configuration.yaml`, is where most of the configuration parameters for all aspects of the OpihiExarata software package exists.

The configuration file is a YAML formatted file with relatively verbose names. A newly generated configuration file is a copy of the default file being used.

To detail all of the configuration parameters here is not particularly efficient. The parameters are documented in the configuration file itself. It can be found at `OpihiExarata/src/opihiexarata/configuration.yaml` in the directory tree. Alternatively, it can also be found here on the OpihiExarata Github, configuration.yaml[9].

[9] https://github.com/psmd-iberutaru/OpihiExarata/blob/master/src/opihiexarata/configuration.yaml

Secrets Configuration File

A secrets configuration file (simplified to secrets file), typically called something like `secrets.yaml`, is where configuration parameters which should not be released to the public (via the open source repository) are kept, hence a secrets file. These are often software and API keys.

The secrets file is a YAML formatted file with relatively verbose names. A newly generated secrets file is a copy of the default file being used. But, there are no defaults and the secrets are blank as that is the whole point of a secrets file. The configuration values need to be filled in with a user provided information.

To detail all of the secrets parameters here is not particularly efficient. The secrets themselves are documented in the secrets file itself. It can be found at `OpihiExarata/src/opihiexarata/secrets.yaml` in the directory tree. Alternatively, it can also be found here on the OpihiExarata Github, secrets.yaml[10].

1.1.7 Troubleshooting

For all of your troubleshooting needs. Here we detail some of the common problems encountered and the solutions for these problems.

Automatic Mode Stop Button Not Working

If the *Stop* button in the automatic mode GUI is not working and fails to stop the loop as designed, then this is likely because of an oversight when the code was changed. The loop will run forever until stopped as per the design of the automatic mode.

You can force the loop to stop gracefully by creating a file named `opihiexarata_automatic.stop` in the same data directory that the automatic loop is fetching the most recent images from. The loop checks for the existence of this file and it will stop in a similar way to if the stop button was pressed. The loop will be considered *Halted* as it stopped via a method other than the *Stop* button.

If this fails, then the suggested remedy is to ungracefully stop or crash the process that OpihiExarata is running in.

In either case, this is an error with the code of OpihiExarata and the maintainers of the software should be contacted.

PySide GUI Does Not Match Documentation or UI Files

This problem can be caused by one of two things.

If the documentation and the GUI differs, then the documentation may be out of date. Otherwise, the Python GUI files built by PySide6 are out of date and need to be built again.

If the documentation is out of date, see *Optional: Documentation* for rebuilding up to date documentation. If information is missing, please contact the maintainers to update the documentation.

If the GUI differs from the Qt Designer UI files, then the Python UI needs to be built from these files. See *Building UI Files* for more information.

[10] https://github.com/psmd-iberutaru/OpihiExarata/blob/master/src/opihiexarata/secrets.yaml

1.1.8 Citations

If you use the Opihi telescope or its accompanying OpihiExarata software, please cite your usage using the following citation information:

(The citation information here will be updated as more publication information is provided.)

References

We detail the other projects that OpihiExarata uses to accomplish its goals, in no particular order.

NASA Infrared Telescope Facility

This work utilizes, is utilized by, is built for, and is funded by the NASA Infrared Telescope Facility is operated by the University of Hawaii under contract 80HQTR19D0030 with the National Aeronautics and Space Administration.

MAST

Some/all of the data presented in this work were obtained from the Mikulski Archive for Space Telescopes (MAST). STScI is operated by the Association of Universities for Research in Astronomy, Inc., under NASA contract NAS5-26555. Support for MAST for non-HST data is provided by the NASA Office of Space Science via grant NNX13AC07G and by other grants and contracts.

Minor Planet Center

This work makes use of the Minor Planet & Comet Ephemeris Service (IAU Minor Planet Center)

Scipy

This work makes use of SciPy (Virtanen et. al. 2020[11]).

Matplotlib

This work makes use of matplotlib, a Python library for publication quality graphics (Hunter 2007[12]).

[11] https://doi.org/10.1038/s41592-019-0686-2
[12] https://doi.ieeecomputersociety.org/10.1109/MCSE.2007.55

Astropy

This work makes use of Astropy, a community-developed core Python package for Astronomy (Astropy Collaboration et. al. 2018[13], Astropy Collaboration et. al. 2013[14]).

Numpy

This work makes use of NumPy (Harris et. al. 2020[15]).

Gaia

This work makes use of data from the European Space Agency (ESA) mission *Gaia* (website[16]), processed by the *Gaia* Data Processing and Analysis Consortium (DPAC[17]). Funding for the DPAC has been provided by national institutions, in particular the institutions participating in the *Gaia* Multilateral Agreement.

Pan-STARRS1

The Pan-STARRS1 Surveys (PS1) have been made possible through contributions of the Institute for Astronomy, the University of Hawaii, the Pan-STARRS Project Office, the Max-Planck Society and its participating institutes, the Max Planck Institute for Astronomy, Heidelberg and the Max Planck Institute for Extraterrestrial Physics, Garching, The Johns Hopkins University, Durham University, the University of Edinburgh, Queen's University Belfast, the Harvard-Smithsonian Center for Astrophysics, the Las Cumbres Observatory Global Telescope Network Incorporated, the National Central University of Taiwan, the Space Telescope Science Institute, the National Aeronautics and Space Administration under Grant No. NNX08AR22G issued through the Planetary Science Division of the NASA Science Mission Directorate, the National Science Foundation under Grant No. AST-1238877, the University of Maryland, and Eotvos Lorand University (ELTE).

JPL Horizons

This work makes use of the JPL Horizons API service (Giorgini, JD and JPL Solar System Dynamics Group, NASA/JPL Horizons On-Line Ephemeris System https://ssd.jpl.nasa.gov/horizons/[18]; Giorgini et. al. 2001[19]; Giorgini et. al. 1999[20]; Giorgini et. al. 1996[21])

[13] https://ui.adsabs.harvard.edu/abs/2018AJ....156..123A
[14] https://ui.adsabs.harvard.edu/abs/2013A&A...558A..33A
[15] https://doi.org/10.1038/s41586-020-2649-2\}
[16] https://www.cosmos.esa.int/gaia
[17] https://www.cosmos.esa.int/web/gaia/dpac/consortium
[18] https://ssd.jpl.nasa.gov/horizons/
[19] https://ui.adsabs.harvard.edu/abs/2001DPS....33.5813G
[20] https://web.archive.org/web/20220620101354/https://www.techbriefs.com/component/content/article/tb/pub/briefs/software/30057
[21] https://ui.adsabs.harvard.edu/abs/1996DPS....28.2504G

Astrometry.net

This work makes use of the Astrometry.net Nova web service and installable programs (Lang et. al 2010[22], Hogg et. al. 2008[23]).

1.1.9 Trivia

Etymology of OpihiExarata

The IRTF Opihi telescope is named after the opihi[24], a limpet which lives on rocky shores by sticking to the rocks. In a similar vein, the Opihi telescope sticks hard onto the side of the main IRTF telescope. One of the three opihi species endemic to the Hawaii islands is the Hawaiian blackfoot opihi[25]. The Opihi telescope is similar to this species and for this reason, this software is named after it. (The similarities: the opihi species and the telescope are both black; the Opihi telescope is rather small compared to the IRTF, the blackfoot is a small opihi; and the blackfoot opihi is a high delicacy, similar to this software, it has a rather small audience.)

The binomial taxonomical name of the Hawaiian blackfoot opihi is Cellana exarata. The genus Cellana describes the group of limpets including the opihi, but *exarata* is the identifying part for the Hawaiian blackfoot, thus the name of this software, we decided, is the hybrid: OpihiExarata. A backronym for it, descriptive of its function: OPIHI Ephemeris with eXtra Atmospheric Response and Asteroid Trajectory Analysis.

[22] https://ui.adsabs.harvard.edu/abs/2010AJ....139.1782L
[23] https://ui.adsabs.harvard.edu/abs/2008ASPC..394...27H
[24] https://www.waikikiaquarium.org/experience/animal-guide/invertebrates/molluscs/opihi/
[25] https://en.wikipedia.org/wiki/Cellana_exarata

CHAPTER

TWO

TECHNICAL MANUAL

The technical manual is primarily for software maintainers and other IRTF staff. (For general use purposes, see the *User Manual*). It details the overall architectural design and thought process of the OpihiExarata software in much greater detail to assist in development. (For function and class documentation of the raw API itself, see the *Technical Manual*.) It can be accessed via the sidebar or by clicking *here*.

2.1 Technical Manual

The technical manual here is made for the those looking for more detail about the inner-workings of the OpihiExarata software. This is typically staff and maintainers for the OpihiExarata package, but it may be useful for some other audiences.

We detail the installation instructions (written primarily for the computer which runs the Opihi telescope in the event that this software needs to be reinstalled).

We also detail some of the architectural considerations and designs of the OpihiExarata software as a whole and some of the algorithms and conventions which underpin the implementation.

The license by which this software is licensed under, and a small discussion on the licenses of its dependencies is also provided.

2.1.1 Installation

The installation instructions are detailed here. For convenience, installation scripts have been written to install the software for choice operating systems. Currently, the supported operating systems are, in order, Rocky Linux, Windows, and Ubuntu. Future compatibility with other operating systems is unlikely to be supported in the future as this software package is purpose built. In order to get the scripts, and the software in general, you still need to download the software, see *Install: Download*.

Automated Scripts

As there are many parts to the OpihiExarata software, for convenience, an installation script has been written for convenience and to help facilitate installation. Please note that the automated script installs everything, if you want to customize your install, please see *Manual Installation*.

The installation script is written in Powershell Core and thus require the installation of Powershell Core. Instructions for the installation and usage of Powershell Core[26] can be found on Microsoft's website. The installation of the OpihiExarata software does not depend on Powershell Core after installation, but the convenience scripts would be unavailable. If this is the case, please follow the other installation instructions as usual.

In addition to the installation script, an auxillary script has been written in Powershell Core. The purpose of the auxillary script is to do tasks not strictly related to installation. These tasks include code formatting, testing, and documentation building. You may use this as well but it is not necessary for installation.

Script Usage

A few elements of the scripts must be tuned to your current operating system. Editing a few lines in the beginning of each script file before running them is all that is needed.

For the installation script `install.ps1`:

- You need to specify the version-specific Python command that Powershell Core uses to enter the correct Python interpreter. It is typically something like `python3.9` for Python 3.9, or something similar for other versions. (Windows operating systems, for example, is often just `python`.) For example, the line in `install.ps1` should be something like:

```
Set-Alias -Name pyox -Value python3.9
```

- If you are using a Linux system, you need to specify the primary flavor as different systems have different ways of installing packages. Set either `$isCentOS` or `$isUbuntu` to True based on what system you have. If the system detects that you have a Linux system and neither is specified, an error will be raised.

For the auxillary script `auxillary.ps1`:

- Like the install script, you need to specify the version-specific Python command that Powershell Core uses. Typically it is typically something like `python3.9` for Python 3.9. It is advised that you use the same entry as in the installation script:

```
Set-Alias -Name pyox -Value python3.9
```

[26] https://docs.microsoft.com/en-us/powershell/

Manual Installation

Instructions are present for those who prefer a manual install, either because the installation script does not work or they want to customize their installation.

The installation instructions must be followed in order. There are optional parts to the installation instructions, typically they are not needed for a feature complete install of OpihiExarata, but they are highly suggested. The installation order is as follows, we note what is optional and why:

1. *Install: Download*: You will download the OpihiExarata package from its Github repository in this step.
2. *Install: Python Part*: You will download a supported Python version and build then install the Python part of OpihiExarata and its dependencies. This part is the main part of installation.
3. *Optional: Documentation*: (Optional) You will build and process the documentation files to build the documentation for OpihiExarata. This is helpful if this set of documentation that you are reading is outdated or if you just want the documentation on your machine to be up to date.
4. *Optional: Windows Compatibility*: (Situational) Some of the services that OpihiExarata depends on only works for Linux. In order to have supported functionality on Windows, the OpihiExarata software leverages the Windows Subsystem for Linux. It is required that Windows users download and install this feature. The instructions are detailed here.
5. *Install: Orbfit*: This installs the Orbfit preliminary orbit determiner program which OpihiExarata uses for orbit determination.

Install: Download

The OpihiExarata software is stored its Github software repository[27]. It needs to be downloaded from there, there are two main methods of doing that. Both are detailed here.

Throughout the installation tutorial we refer to the internal OpihiExarata directory as `OpihiExarata/`. This allows for the instructions to be general. Please adapt any absolute paths as needed.

Via git (Recommended)

The best way to download OpihiExarata is to clone the repository:

```
git clone https://github.com/psmd-iberutaru/OpihiExarata.git
```

The location where you download the OpihiExarata repository is relatively irrelevant.

[27] https://github.com/psmd-iberutaru/OpihiExarata

Via .zip Archive

There are alternative ways to download the software. Although, the only other method that is worthwhile to document is downloading the software as an archive and extracting it to the desired location.

A zip archive of the git repository can be downloaded via:

```
curl -O -L https://github.com/psmd-iberutaru/OpihiExarata/archive/refs/heads/
↪master.zip
unzip master.zip
# Optional; to follow the documentation naming conventions.
mv OpihiExarata-master OpihiExarata
# The zip is no longer needed.
rm master.zip
```

Install: Python Part

The Python part of OpihiExarata is the primary part of OpihiExarata. Luckily, it is also likely the simplest.

You will build the Python wheel from the source and install it.

Download

Download and install a current release of Python if you have not already. You can find the latest releases at: Python Releases[28].

Please note you must install/have Python 3.9+.

Build

You should have have the repository code in `OpihiExarata/`. You can run the build command (while in the directory) based on the operation system you are using:

Windows: **`python -m build`**

Linux: **`python39 -m build`**

The package will build into a distributable wheel, note in the output the version that you installed. The version is likely to be the day you built it, it affects the name of the wheel file for the next step.

[28] https://www.python.org/downloads/

Install

Install the wheel package.

You will need to modify this command to the proper wheel that you generated from before. Because this project uses date-based versioning, your package will likely be that of the date that you build the wheel.

(The `--upgrade` option is to ensure that you have the most up to date version.)

You can install the wheel using while in the `OpihiExarata/` directory with:

```
pip install ./dist/OpihiExarata-YYYY.MM.DD-py3-none-any.whl --upgrade
```

Optional: Documentation

Building the documentation is relatively simple as we leverage Sphinx to build it. This is also done when the `auxiliary.ps1` auxillary script is run.

Prerequisites

Sphinx

To build the Sphinx documentation, Python needs to be installed. (The Python version installed in *Install: Python Part* can be used.)

The following packages must be installed:

```
pip install sphinx sphinx-rtd-theme
```

LaTeX

If a PDF version of the documentation is desired, Sphinx can build it via LaTeX. LaTeX must be installed. We suggest installing the TeX Live distribution[29] and selecting the full install. Instructions on how to install TeX Live is beyond the scope of this documentation.

Directory

Change your directory to the `OpihiExarata/docs` directory, run all of the commands while within this directory.

[29] https://tug.org/texlive/

Build

First, the Python docstrings need to be processed into documentation. This can be done via running:

```
sphinx-apidoc -f -e -o ./code/ ./../src/opihiexarata/
```

Second, it is helpful to removed the cached versions of the documentation files, you can do this via the commands: (This just removes the build directory along with other goodies.)

- Windows: **`.make.bat clean`**
- Linux: **`make clean`**

Third, the documentation can be built using the batch/makefile using the command:

- Windows: **`.make.bat <type>`**
- Linux: **`make <type>`**

The `<type>` should be replaced with the type of output desired, suggestions below:

- `html` A collection of webpages ordered and structured. This is the suggested method.
- `singlehtml` A single HTML page; useful when sending the documentation between devices.
- `latexpdf` A PDF compiled by built LaTeX files. Using this option invokes the entire toolchain for you.

The documentation can be built to other different forms, see the sphinx-build documentation[30].

Optional: Windows Compatibility

Many of the parts of OpihiExarata are built for Linux-only operating systems. However, Windows can use a Linux subsystem to call the compiled Linux parts, allowing OpihiExarata to run on Windows, with a set of assumptions.

When a Linux call is needed, the software calls it via the Windows Subsystem for Linux (WSL) using Powershell. Thus WSL must be installed and Powershell must have script mode enabled.

Powershell Execution Policy

Powershell scripts are seen as the most reliable way to call WSL on Windows. But, the default execution policy of most Windows machines disable scripts. To enable scripts, do the following command in an administrator Powershell shell:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
```

See Microsoft's documentation for more information on setting Powershell execution policies[31] and what policies are available[32].

[30] https://www.sphinx-doc.org/en/master/man/sphinx-build.html
[31] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.security/set-executionpolicy
[32] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_execution_policies

Windows Subsystem for Linux

As the WSL system is slightly new, the steps to install it differ based on what version of Windows is installed.

For Windows 10 version 2004 and higher (Build 19041 and higher) or Windows 11 and higher, see *Modern Install*.

For older versions of Windows 10 (version 1903 or higher, with build 18362 or higher), see *Manual Install*.

OpihiExarata does not support or is designed to run on older versions of Windows.

Modern Install

In newer Windows versions (Windows 10 version 2004 and higher (Build 19041 and higher) or Windows 11), the installation of WSL is relatively simple.

In an administrator Powershell, run the following:

```
wsl --install
```

This installs Ubuntu as the WSL distribution by default. See the WSL install documentation[33] for more information.

Manual Install

These installation instructions was taken from the manual installation documentation[34] from Microsoft.

Enable WSL

In an administrator Powershell, enable the WSL system via:

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-
↪Linux /all /norestart
```

Enable Virtual Machine Feature

In an administrator Powershell, enable the virtual machine platform:

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /
↪norestart
```

Download Linux Kernel Update Package

You can download the latest kernel package from: WSL2 Linux kernel update package for x64 machines[35].

Run the downloaded package; you will likely be prompted for administrative permissions.

Set WSL 2 as Default

[33] https://docs.microsoft.com/en-us/windows/wsl/install
[34] https://docs.microsoft.com/en-us/windows/wsl/install-manual
[35] https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi

In an administrator Powershell, use the following command to set WSL 2 as the default version:

```
wsl --set-default-version 2
```

This the more typical WSL installation, rather than WSL 1 which is a compatibility layer.

Download Linux Distribution

Download your desired Linux distribution from the Windows store, we suggest WSL Ubuntu[36]:

Verify

To verify that that the Powershell execution policy is correct, check that the Local Machine policy is at least RemoteSigned. You can check this by using an administrative Powershell command:

```
Get-ExecutionPolicy -List
```

To verify that you have installed and set up WSL properly, in an administrative Powershell run:

```
wsl
```

If this is the first time you have installed the WSL system onto your computer, you will have to setup a user account similar to a fresh install. Follow and complete the prompt, and then check **`wsl`** again to ensure that it enters a standard Linux shell without any interruptions.

Final

When installing other parts of the OpihiExarata software, follow the instructions using the WSL OS installed where needed. The commands should be run in your usual WSL shell in the proper relevant directory.

Install: Orbfit

Orbfit is one of the available asteroid orbit determiners.

The instructions of installing Orbfit is derived from the Orbfit installation documentation[37].

[36] https://www.microsoft.com/store/apps/9n6svws3rx71
[37] http://adams.dm.unipi.it/~orbmaint/orbfit/OrbFit/doc/help.html#install

Prerequisites

There are a few prerequisites before the software can be installed.

Directory

Enter into a directory which is accessible and usable by programs. In this documentation, we will call this directory `Orbfit/`. You should do the commands of this section of the installation documentation within this directory.

Dependencies

The following dependencies should be installed. The installation of these are dependent on the package manager and operating system used.

For APT-based systems:

```
sudo apt install gcc make curl time
```

For YUM, DNF, or Zypper based systems:

```
sudo yum install gcc make curl time
```

Fortran Compiler

A Fortran compiler is needed to compile the software. Although there are many different compilers available, we document here how to use the Intel Fortran compiler. The method of install is OS dependent, or more specifically, package manager dependent. Other installation method exist, see Intel Fortran compiler install[38] for more information (these instructions were derived from it).

Install

Using package managers to install is the most convenient.

[38] https://www.intel.com/content/www/us/en/develop/documentation/installation-guide-for-intel-oneapi-toolkits-linux/top/installation.html

APT Package Managers

Download the Intel repository to the system keyring:

```
# Downloading key...
wget -O- https://apt.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-
↪PRODUCTS.PUB \
| gpg --dearmor | sudo tee /usr/share/keyrings/oneapi-archive-keyring.gpg > /
↪dev/null
# Adding signed entry...
echo "deb [signed-by=/usr/share/keyrings/oneapi-archive-keyring.gpg] https://
↪apt.repos.intel.com/oneapi all main" | sudo tee /etc/apt/sources.list.d/
↪oneAPI.list
# Update the repository list...
sudo apt update
```

Install the Intel Fortran compiler, the base toolkit is also needed as a requirement:

```
sudo apt install intel-basekit
sudo apt install intel-hpckit
```

It is highly suggested to update or install the following toolchains:

```
sudo apt update
sudo apt install cmake pkg-config build-essential
```

YUM, DNF, Zypper Package Managers

Download the Intel repository file and add it to the configuration directory:

```
# Creating repository file...
tee > /tmp/oneAPI.repo << EOF
[oneAPI]
name=Intel® oneAPI repository
baseurl=https://yum.repos.intel.com/oneapi
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://yum.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-
↪PRODUCTS.PUB
EOF
# Moving the file to the directory...
sudo mv /tmp/oneAPI.repo /etc/yum.repos.d
# Update the repository list...
sudo yum update
```

If you are using Zypper:

```
# Creating repository file...
sudo zypper addrepo https://yum.repos.intel.com/oneapi oneAPI
```

(continues on next page)

(continued from previous page)

```
rpm --import https://yum.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-
↪PRODUCTS.PUB
# Update the repository list...
sudo zypper update
```

Install the Intel Fortran compiler, the base toolkit is also needed as a requirement:

```
sudo yum install intel-basekit
sudo yum install intel-hpckit
```

It is highly suggested to update or install the following toolchains:

```
sudo yum update
sudo yum install cmake pkgconfig
sudo yum groupinstall "Development Tools"
```

For Zypper, both is done via:

```
sudo zypper install intel-basekit
sudo zypper install intel-hpckit
sudo zypper update
sudo zypper install cmake pkg-config
sudo zypper install pattern devel_C_C++
```

Environment Variables

Set the environment variables for command-line development. This command sets the variables for the current session only. (For persistent environment variable set up, consider adding it to your startup script.):

```
. /opt/intel/oneapi/setvars.sh
```

Download OrbFit

The software needs to be downloaded.

You can likely find the software package on the OrbFit website[39]. Otherwise, the download command should work:

```
curl -O http://adams.dm.unipi.it/orbfit/OrbFit5.0.7.tar.gz
```

It can then be extracted using:

```
tar -xvzf OrbFit5.0.7.tar.gz
```

[39] http://adams.dm.unipi.it/orbfit/

Compile

To configure the compilation flags, OrbFit comes with a set of files which describe the flags. Initialize the proper compilation flags via the **`config`** script, (flags for an optimized build using the Intel compiler):

```
./config -O intel
```

The generated compilation flags, however, needs to be changed. The generated compilation flags can be found in the file `Orbfit/src/make.flags`, as generated by the **`config`** script. The options should be (see Intel Fortran compiler options[40] for more information):

```
FFLAGS= -warn nousage -O -mp1 -static-intel -save -assume byterecl -I../
↪include
```

Note: You make use the default compilation flags as well; the change to the flags just allows the program to run with the Intel Fortran libraries statically compiled with the program rather than linked in. This alleviates the need for always needing to set up the environment variables whenever OrbFit is run. The changing memory model does not seem to affect the orbit determination part of the program.

Finally the program can be compiled using the makefile:

```
make
```

The compiled programs should exist in `Orbfit/bin/`.

Download JPL Ephemerides File

The OrbFit package uses the JPL ephemerides file for its calculations, it requires the binary ephemerides files. Current OrbFit documentation suggests using the 405 ephemerides set, but more updated sets (DE 407) also exist. For Linux, precomputed binaries can be found at JPL Ephemerides binary files for Linux[41] and the JPL Ephemerides descriptions[42].

For DE405:

```
curl -O https://ssd.jpl.nasa.gov/ftp/eph/planets/Linux/de405/lnxp1600p2200.405
```

For DE440 (used in this documentation):

```
curl -O https://ssd.jpl.nasa.gov/ftp/eph/planets/Linux/de440/linux_p1550p2650.
↪440
```

The downloaded binary ephemerides file must be linked so that the OrbFit program can properly utilize it; using a symbolic link in the `Orbfit/lib/` directory, and assuming the DE440 file was downloaded (the command below should be changed to fit your file):

[40] https://www.intel.com/content/www/us/en/develop/documentation/fortran-compiler-oneapi-dev-guide-and-reference/top/compiler-reference/compiler-options/alphabetical-list-of-compiler-options.html

[41] https://ssd.jpl.nasa.gov/ftp/eph/planets/Linux/

[42] https://ssd.jpl.nasa.gov/planets/eph_export.html

```
cd ./lib/
ln -s ./../linux_p1550p2650.440 jpleph
# Back to the Orbfit/ directory.
cd ..
```

Testing Suite

The OrbFit software's test suit can be executed from `Orbfit/` via:

```
make tests
```

This ensures that the program has been installed correctly.

Executable Path for Configuration File

For this program's executables to be used, the path that they exist in must be known to OpihiExarata. Copy the output of the working directory command and add it to the configuration file's entries as noted. The path should be an absolute path; these commands should be run in the `Orbfit/` directory:

```
cd .; echo "ORBFIT_DIRECTORY =="; pwd
cd ./bin; echo "ORBFIT_BINARY_EXECUTABLE_DIRECTORY =="; pwd; cd ..
```

If your main operating system is Windows and you are installing this via WSL Ubuntu, use the following instead:

```
cd .; echo "ORBFIT_DIRECTORY =="; echo "\\\\wsl$/Ubuntu"$(pwd)
cd ./bin; echo "ORBFIT_BINARY_EXECUTABLE_DIRECTORY =="; pwd; cd ..
```

OpihiExarata Template Files

In order for OpihiExarata to properly use the OrbFit program, files which are used by OrbFit must be collected for OpihiExarata to leverage.

Create and enter the directory (starting from `Orbfit/`):

```
mkdir exarata; cd exarata
```

Within the `Orbfit/exarata/` directory, copy over the asteroid propagation files. This contains ephemerides data for asteroids to better propagations. Generally, these files can be found in `Orbfit/tests/bineph/testout`. For some reason, the file extensions on these files are not in a form which is liked by Orbfit, so they are also changed. Namely, the commands below should work:

```
# Copying the files...
cp ./../tests/bineph/testout/AST17.* .
cp ./../tests/bineph/testout/CPV.* .
# Extension changes...
```

(continues on next page)

(continued from previous page)

```
mv AST17.bai_431_fcct AST17.bai
mv AST17.bep_431_fcct AST17.bep
mv CPV.bai_431_fcct CPV.bai
mv CPV.bep_431_fcct CPV.bep
mv CPV_iter.bop CPV.bop
```

Also, create the files which are to be used to input data into Orbfit from OpihiExarata. The main file that needs to be created is `exarata.oop`. Create it with your favorite text editor and fill the file and save it with the following:

```
! First object
object1.
    .name = exarata         ! Object name
    .obs_dir = '.'          ! Observations directory

! Elements output
output.
    .epoch = CAL 2022/01/01  00:00:00 UTC  ! The epoch time of the orbit
    .elements = 'KEP'                      ! Kepler output elements

! Operations: preliminary orbits, differential corrections, identification
operations.
    .init_orbdet = 1     ! Initial orbit determination
                         ! (0 = no, 1 = yes)
    .diffcor = 1         ! Differential correction
                         ! (0 = no, 1 = yes)
    .ident = 0           ! Orbit identification
                         ! (0 = no, 1 = yes)
    .ephem = 0           ! Ephemerides
                         ! (0 = no, 1 = yes)

! Error model
error_model.
    .name='fcct14'       ! Error model

! Propagation
propag.

    .iast=17          ! 0=no asteroids with mass, n=no. of massive asteroids␣
↪(def=0)
    .filbe='AST17'    ! name of the asteroid ephemerides file (def='CPV')
    .npoint=600       ! minimum number of data points for a deep close appr␣
↪(def=100)
    .dmea=0.2d0       ! min. distance for ctrl. of close-app. to Earth only␣
↪(def=0.1)
        .dter=0.05d0     ! min. distance for control of close-app.
                         ! to terrestrial planets (MVM)(def=0.1)

! Additional options
IERS.
    .extrapolation=.T.  ! extrapolation of Earth rotation
```

(continues on next page)

(continued from previous page)

```
reject.
    .rejopp=.FALSE.     ! reject entire opposition
```

The other two files you can create with the following commands:

```
echo "exarata" > exarata.inp
touch exarata.obs
```

Note: OpihiExarata will check that these three files exist as a check to see that these steps have been followed correctly. It is also a way to ensure that the configuration paths provided in the configuration are valid and point to the right location.

2.1.2 Architecture

In brief, the OpihiExarata software was created with the following requirements in mind to guide its architectural design:

- It must be able to accommodate and provide for the use cases of the Opihi telescope, (see *User Manual*).
- The requirements for the user for interaction must be minimal in complexity but feature complete.
- The software should be robust and allow for components to be interchangeable with each other.
- The software should be easily maintainable, well documented, and use established technologies over custom implementation.

We thus designed the parts of the OpihiExarata software package to follow these principles. A visual overview is given in Fig. 2.1.

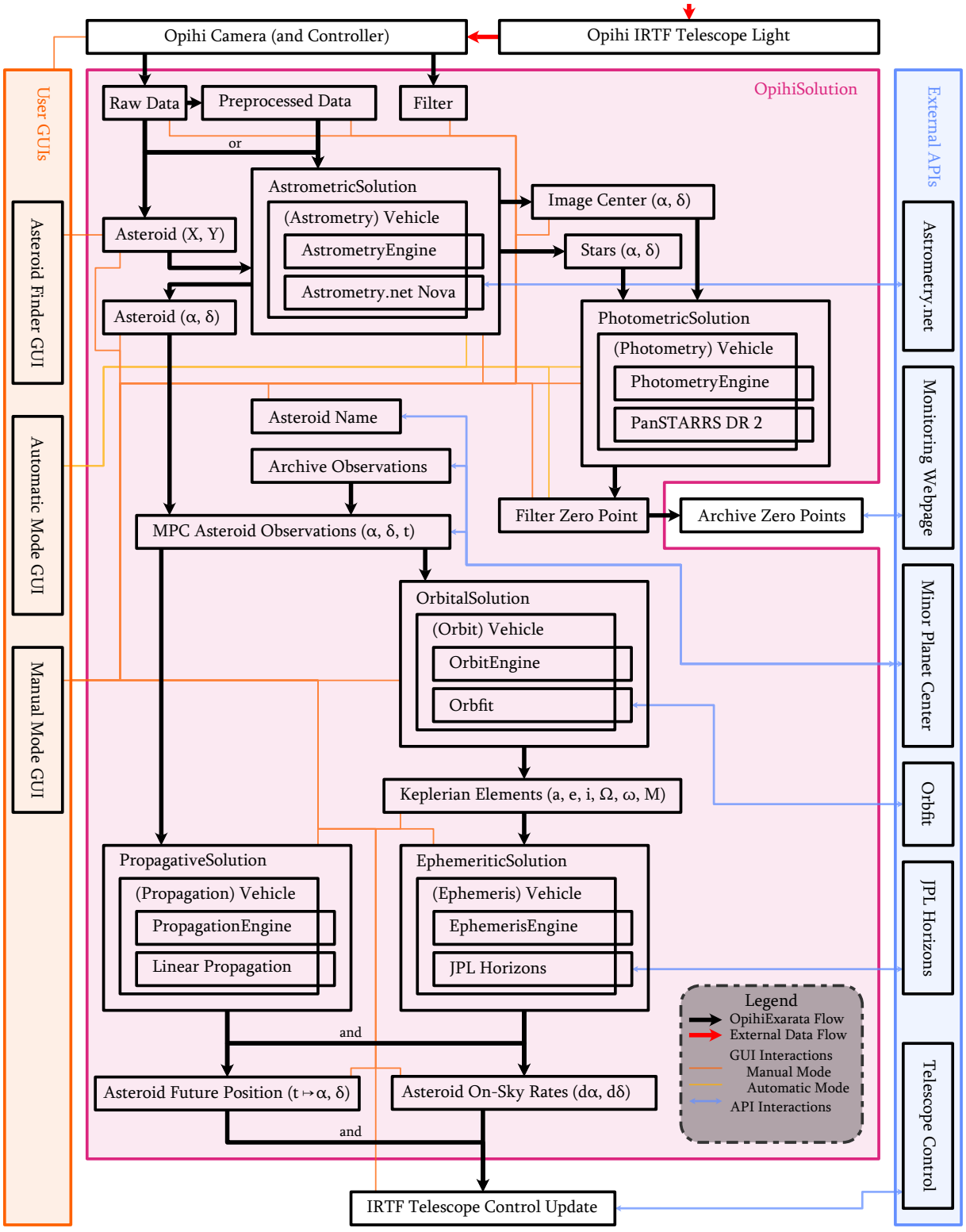

Fig. 2.1: A comprehensive single image overview of the main components of the OpihiExarata software and all of the parts which make it work. Each of the boxes within OpihiSolution represent an attribute of the overall class while the black arrows detail their interactions. The other interaction lines detail external (primarily GUI and internet service) interaction with the attributes.

To best construct the software with interchangeability and maintainability in mind, we often on modularized and abstracted where possible.

The OpihiExarata software handles the analysis of images one at a time. All relevant data and results of a given image are stored in a container class called OpihiSolution. Historical observational information is also provided and used in the OpihiSolution class.

As described in *System Framework*, there are five main problems which this software is to solve for each given image: astrometric plate solving, photometric calibration, orbit determination, ephemeris calculation, and asteroid on-sky propagation. These five problems are distinct and thus we developed five different solution classes to contain the results of each of these problems for every image. These solution classes are unimaginatively named AstrometricSolution, PhotometricSolution, OrbitalSolution, EphemeriticSolution, and PropagativeSolution respectively.

In order for a given solution class to have results to store, we need to actually solve the main five problems. We utilize external services and other established programs to solve them. In order to use these services to obtain the desired results, we use a function to mimic a user using the service. We call these functions "vehicle functions". Each service has their own vehicle function because each service is different.

The results derived from vehicle functions are stored in the solution classes which themselves are ultimately stored in the overarching OpihiSolution class. More information about the vehicle functions and solution classes can be found in *Vehicles and Solutions*.

In order for the software to interact with a given service (through its corresponding vehicle function) we need to have an API to interact with. ALthough many services provide their own API, others do not. Therefore, we built custom APIs for each and every supported service. We call our custom APIs "engines". Each engine is specifically curtailed for handling Opihi data and extracting the needed information from whatever service or program it was specifically made for. A list of available engines and other information is provided in *Services and Engines*.

All implemented engines solve one of the five main problems. Along with their corresponding vehicle functions, there is a standard data input (i.e. to be solved) and standard output of results (to be stored in the solution classes), as detailed in *Vehicles and Solutions*. Because of this standardization, all engines (technically engine-vehicle pairs) which all solve the same problem are interchangeable with each other. The user can select between different implemented engines/services best suited for their particular object. This property is also useful if one of the engines/services break, Opihi would not become useless.

The user should not need to deal with all of this detail as to keep it simple in usage. All that is relevant for the user is picking the appropriate engine-vehicle pairs based on their applicability to the user's data. They should interact with OpihiExarata via the graphical user interface depending on the modes of operation (see *Automatic Mode* and *Manual Mode*). More information on how this GUI was built and related information can be found in *Graphical User Interface*.

For ease of maintainability, specific conventions were adopted to both increase speed and compatibility between different sections of the code. These conventions also assist with making the software easy to maintain. More information about these conventions are provided in *Conventions*. Moreover, functions which are commonly used across the entire software package are collected in a library for reusability and consistency; the available functionality provided by the library is detailed in *Library*.

Overall, we created wrapper code around the available services which solve one of the five problems to have a more maintainable and replaceable codebase. These services (through the "engine" and "vehicle function" wrapper code) take a standardized input and provide a standard set of outputs which the software uses to

generate classes which contain these results and others (as another layer of abstraction). We use the results of the five main problems, as calculated from a specific Opihi image and other historical observations, to determine the needed results (to modify the telescope control or to monitor the atmospheric conditions).

Services and Engines

The OpihiExarata software primarily outsources solving the main five problems for solving an image to other internal or external modules, those being:

- The image astrometric solution, the pointing of the image and its WCS solution;
- The photometric solution, determined from a photometric star catalog;
- The preliminary orbit determination, calculated from a record of observations;
- The ephemeris, calculated from a set of provided orbital elements;
- The asteroid propagation, calculated from a set of asteroid observations.

There are many different types of services and programs which are available that are able to solve the above problems. We decided that it is good to allow the user (or the program in general) to be able to customize/swap which service they use to process their data. This interchangeability also allows for OpihiExarata to be more stable as if a given service fails, a different one can be selected to continue instead of said failure breaking the software and workflow.

Each service has different interfaces and protocols for communication between it and external problems like OpihiExarata; some have nice APIs while others are more complicated.

To deal with the fact that each service has unique interfaces, we implemented a wrapper/abstraction layer for each and every supported service. This abstraction layer allows for easier implementation of these services. These abstraction layers are called Engines and are implemented in as subclasses from `opihiexarata.library.engine`. These engines are classified under *AstrometryEngines*, *PhotometryEngines*, *OrbitEngines*, *EphemerisEngines*, *PropagationEngine* depending on the problem it solves.

We detail all of the available engines (which solve the five problems) here. Note that the names of the engines themselves (when in the code or otherwise) are case insensitive.

A lot of these engines use other external services cited in *Citations* for our references.

AstrometryEngines

These engines solve for the astrometric plate solution of an image.

Astrometry.net Nova

Implementation: *opihiexarata.astrometry.webclient.AstrometryNetWebAPIEngine*

This allows for the leveraging of the Astrometry.net Nova online service[43] for astrometric plate solving. It is a stable system able to solve most images. However, as it is a public system, there is a queue so solving a particular image may take longer from your perspective because of the added wait time. This implementation is based off of the official version[44].

The images taken are uploaded to the service to be solved, OpihiExarata periodically requests the solution, parsing it when the image is astrometrically solved successfully.

PhotometryEngines

These engines solve for the photometric calibration solution of an image.

Pan-STARRS 3pi DR2 MAST

Implementation: *opihiexarata.photometry.panstarrs.PanstarrsMastWebAPIEngine*

This specifies that the photometric database to be used for photometric calibration should be Pan-STARRS 3pi Data Release 2. We access it via a web interface using the Pan-STARRS MAST API[45].

The Pan-STARRS 3pi Data Release 2 survey covers areas north of -30 degrees declination in only the g', r', i', and z' filters. OpihiExarata queries the database around the field of view for photometric stars to use in calculating relevant the photometric quantities of the image.

OrbitEngines

These engines solve for preliminary orbital elements from a list of observations.

OrbFit

Implementation: *opihiexarata.orbit.orbfit.OrbfitOrbitDeterminerEngine*

This utilizes the OrbFit software system[46] in determining the preliminary orbital elements of a sun-orbiting object (typically an asteroid) provided a list of on-sky coordinate observations through time. This particular software system uses least-squares as its method of orbit determination.

[43] https://nova.astrometry.net/
[44] https://github.com/dstndstn/astrometry.net/blob/main/net/client/client.py
[45] https://catalogs.mast.stsci.edu/docs/panstarrs.html
[46] http://adams.dm.unipi.it/orbfit/

Custom Orbit

Implementation: *opihiexarata.orbit.custom.CustomOrbitEngine*

This does not utilize any actual service for orbital determination. The user provides the orbital elements to use along with their respective epoch period. No asteroid observational information is used. Typically orbital elements a user provides through an interface (GUI) should be passed through the vehicle arguments (see *Services and Engines*).

EphemerisEngines

These engines solve for an asteroid's on-sky track from a set of Keplerian orbital elements.

JPL Horizons

Implementation: *opihiexarata.ephemeris.jplhorizons.JPLHorizonsWebAPIEngine*

This utilizes the JPL Horizons System[47] from the JPL Solar Systems Dynamics[48] group for the determination of an ephemeris from a set of Keplerian orbital elements. This software sends the orbital elements (and other observatory information) via the Horizons API[49] service and parses the returned ephemeris.

PropagationEngine

These engines solve for an asteroid's (or other target's) on-sky track from a set of recent observations.

Linear

Implementation: *opihiexarata.propagate.polynomial.LinearPropagationEngine*

This takes the most recent (within a few hours) observations and fits a first order (linear) polynomial function to both the RA and DEC as a function of time. This method assumes a tangent plane projection and so is not suited for propagations on long timescales (greater than a few hours). See *Polynomial Propagation* for more information on the algorithm used.

[47] https://ssd.jpl.nasa.gov/horizons/
[48] https://ssd.jpl.nasa.gov/
[49] https://ssd-api.jpl.nasa.gov/doc/horizons.html

Quadratic

Implementation: `opihiexarata.propagate.polynomial.QuadraticPropagationEngine`

This takes the most recent (within a few hours) observations and fits a second order (quadratic) polynomial function to both the RA and DEC as a function of time. This method assumes a tangent plane projection and so is not suited for propagations on long timescales (greater than a few hours). See *Polynomial Propagation* for more information on the algorithm used.

Vehicles and Solutions

Vehicles

The input and output of each of the engines are curtailed to each service. (See *Services and Engines* for the engines.) However, the solution classes all expect exactly the same input regardless of which engine/service is used. To extract the needed information from an engine for the instantiation of the solution class, we mimic the usage of an engine (by a user) via what we call a vehicle function.

A vehicle function is just a section of code which mimics the usage of the engines that access the service, using it to determine the required results for the solutions. All vehicle functions have the same input (additional arguments may be provided in rare cases for special engines, details are engine specific and may be found in *Services and Engines*) and the exact same output depending on the solution class it services. We detail the inputs and outputs here for vehicle functions corresponding to the solutions, exceptions (like additional arguments).

Astrometry Vehicle Functions

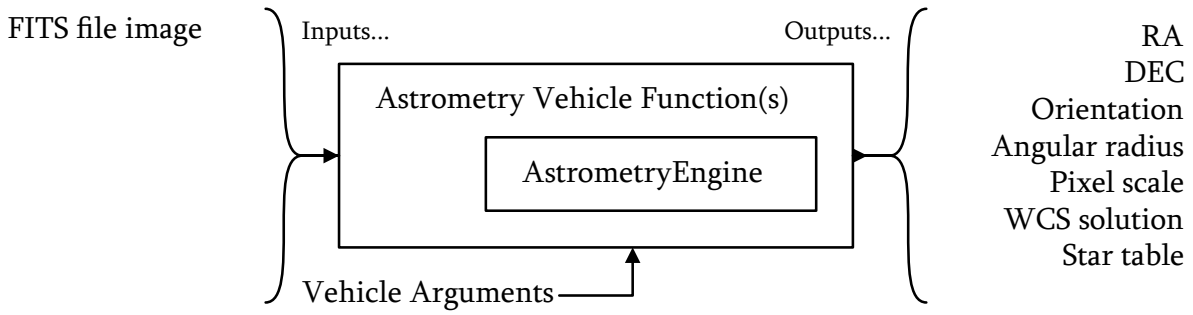

Fig. 2.2: A somewhat visual description of the input arguments used by astrometric vehicle functions and their respective engines to provide the information for astrometric solutions.

As shown in Fig. 2.2, the astrometric vehicle functions and engines generally require the input of the image data. They are required to return the on-sky position of the image, its orientation, and pixel scale. They also must provide a table of all of the stars which are within the field along with a FITS world coordinate system (WCS)[50] solution as implemented via Astropy.

[50] https://fits.gsfc.nasa.gov/fits_wcs.html

Photometry Vehicle Functions

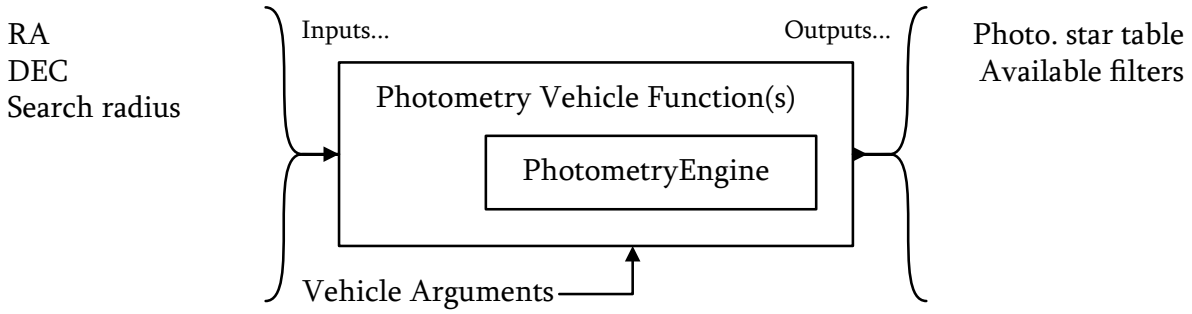

Fig. 2.3: A somewhat visual description of the input arguments used by photometric vehicle functions and their respective engines to provide the information for photometric solutions.

As shown in Fig. 2.3, the photometric vehicle functions and engines generally require the input of parameters (RA, DEC, and search radius) to do a cone search of a photometric catalog. They are required to return a table detailing the location and filter magnitudes of all found stars within the search radius along with a list of filters which the table contains data for.

Orbit Vehicle Functions

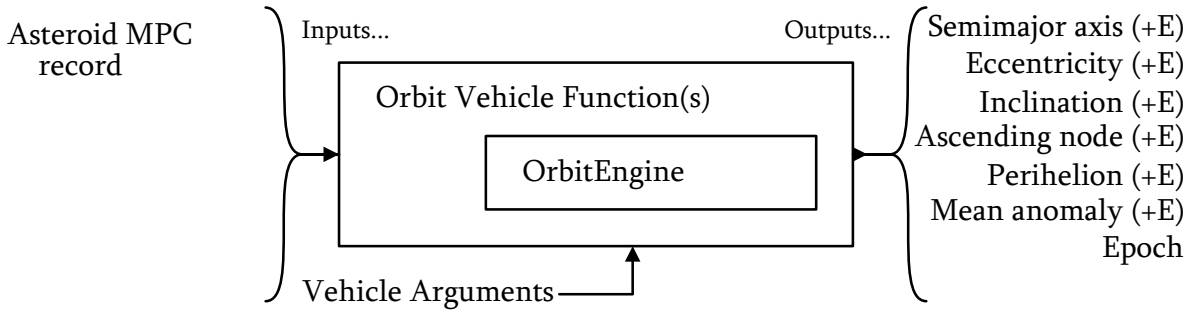

Fig. 2.4: A somewhat visual description of the input arguments used by orbital vehicle functions and their respective engines to provide the information for orbital solutions.

As shown in Fig. 2.4, the orbital vehicle functions and engines generally require the input of asteroid observations in the 80-column Minor Planet Center format. They are required to output (from their calculations) the six Keplerian orbital elements are calculated (along with their errors) as outputs; the epoch of these orbital elements must also be returned.

Ephemeris Vehicle Functions

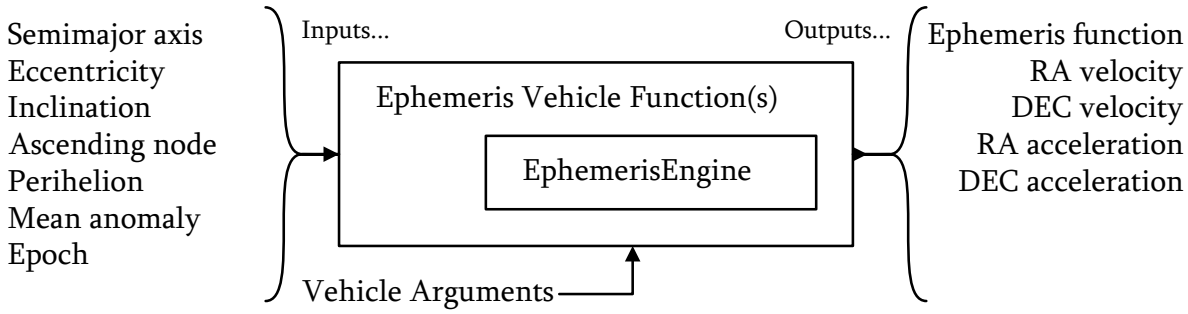

Fig. 2.5: A somewhat visual description of the input arguments used by ephemeris vehicle functions and their respective engines to provide the information for ephemeris solutions.

As shown in Fig. 2.5, the ephemeris vehicle functions and engines generally require the input of the six Keplerian orbital elements and the epoch that they were determined for. They are required to output the on-sky rates (both first order, "velocity", and second order, "acceleration") of an asteroid (or other target) and a function which, when specified a time of observation, gives the predicted on-sky position from the ephemeris.

Propagation Vehicle Functions

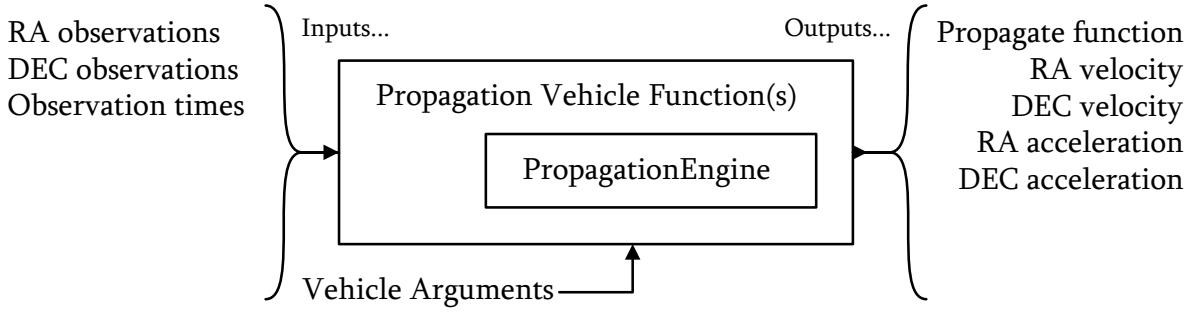

Fig. 2.6: A somewhat visual description of the input arguments used by propagation vehicle functions and their respective engines to provide the information for propagation solutions.

As shown in Fig. 2.6, the propagation vehicle functions and engines generally require the input of on-sky observations (RA, DEC, and observational time) of an asteroid. They are required to output the on-sky rates (both first order, "velocity", and second order, "acceleration") of an asteroid (or other target) and a function which, when specified a time of observation, gives the predicted on-sky position from propagating the on-sky motion.

Solution

Solutions classes incorporate information extracted from the engines and vehicles. These classes also calculates other values which are derived from the results of the engine and vehicle functions.

OpihiSolution

The OpihiSolution is a class which is built to conveniently store and interface with all of the of the other solution classes provided by OpihiExarata for a single image. This grouping is beneficial because a single image and its associated solutions can remain together and saving the newly solved data is a lot easier.

The OpihiSolution class, which contains all of the functionality which OpihiExarata has to offer for a single image from Opihi, is also a helpful abstraction for developing the interactions between the GUI and an image and its data.

AstrometrySolution

Implementation: *opihiexarata.astrometry.solution.AstrometricSolution*

The AstrometrySolution contains the results and other related functions which are derived from the astrometry engines and converted to a standard form from their appropriate vehicle functions.

Primarily, it contains the on-sky location of an image along with its pixel scale. It also contains a table listing the stars which were observed to be in the field and their pixel location. A way to convert between pixel and on-sky coordinates or vice-verse is implemented utilizing the WCS solution.

PhotometrySolution

Implementation: *opihiexarata.photometry.solution.PhotometricSolution*

The PhotometrySolution contains the results and other related functions which are derived from the photometry engines and converted to a standard form from their appropriate vehicle functions.

Primarily, it contains a table listing the stars which are detected within the astrometric solution and it also has their filter magnitudes as provided by the photometric database; aperture DN counts are also given. An average sky value is calculated from sky regions, i.e. not including the stellar regions and other regions on the detector. This average sky value is used to correct the aperture DN counts. From this table and the corrected DN counts, the filter zero point and its error are also calculated.

OrbitalSolution

Implementation: *opihiexarata.orbit.solution.OrbitalSolution*

The OrbitalSolution contains the results and other related functions which are derived from the orbit engines and converted to a standard form from their appropriate vehicle functions.

Primarily, it contains the siz primary Keplerian orbital elements along with the epoch that these orbital elements. The mean anomaly M is the primary anomaly used. However, the eccentric anomaly E is also calculated using Newton's method to solve Kepler's equation: $M = E - e \sin E$. The true anomaly ν is then calculated from the eccentric anomaly and the eccentricity using the geometrically derived formula: (See Broucke, R. Celestial Mechanics p. 388[51].)

$$\tan\left(\frac{\nu - E}{2}\right) = \frac{\beta \sin E}{1 - \beta \cos E}$$

Where β is:

$$\beta \triangleq \frac{e}{1 + \sqrt{1 - e^2}}$$

EphemeriticSolution

Implementation: *opihiexarata.ephemeris.solution.EphemeriticSolution*

The EphemeriticSolution contains the results and other related functions which are derived from the ephemeris engines and converted to a standard form from their appropriate vehicle functions.

Primarily, it contains the ephemeris function. This function provides for the on-sky coordinates of an asteroid at a (provided) future time based on the ephemeris. It also contains the on-sky first-order and second-order rates of an asteroid. The ephemeris itself is derived from the orbital elements as provided by the OrbitalSolution.

PropagativeSolution

Implementation: *opihiexarata.propagate.solution.PropagativeSolution*

The PropagativeSolution contains the results and other related functions which are derived from the propagation engines and converted to a standard form from their appropriate vehicle functions.

Primarily, it contains the propagation function. This function provides for the on-sky coordinates of an asteroid at a (provided) future time based on the propagation of the asteroid's path on the sky. It also contains the on-sky first-order and second-order rates of an asteroid. The propagation of the path is determined by extrapolating the motion of the asteroid on the sky based on a sequence of recent images.

[51] https://ui.adsabs.harvard.edu/abs/1973CeMec...7..388B

PreprocessSolution

The data that comes from the Opihi camera is considered raw data, it has many systematic artifacts like hot pixels, dark current, and bias to name a few. We reduce this data using standard array processing procedures.

We can remove these using preprocessing calibration images. These images are taken before hand. For the use case for Opihi, using archive calibration files are more than satisfactory and avoiding the need for the user to take calibration images on their own reduces the overhead.

An explanation on the procedure and methodology of data preprocessing is assumed by this manual, but a brief summary may be obtained from here[52].

The implementation of image preprocessing is done by the PreprocessSolution class (see *opihiexarata.opihi.preprocess.OpihiPreprocessSolution*). However, because the preprocessing of CCD images is pretty standard and simple and involves only one method, this solution does not require an engine and instead implements it itself.

Moreover, because many of the image calibration files used are generally the same from image to image, this solution is also not image specific unlike the other solution classes. Instead, it contains a function which will take an image (either an array or a fits file) and pre-process it. It is built like this so that the large preprocessing calibration images (which are cached within the class to avoid disk utilization) does not take up a too much memory as opposed if the class was duplicated per image.

Library

Here we provide a brief summary of the available functionality provided by the software library of OpihiExarata. The whole point of the library is to store functions and subroutines which are useful across the entire software package.

When developing and maintaining OpihiExarata itself, please utilize the library before implementing something custom. If the library is incomplete, and the missing functionality would likely be used again at least once, please add it to the library and call it from there. The library does not typically reimplement functionality already implemented by third-party package dependencies; however, there may be some wrapper functions to streamline the capabilities thereof to better fit the use cases for OpihiExarata.

The summaries provided here do not substitute a search through the *Code Manual*, but hopefully they help in searching for library functionality.

Configuration

See *opihiexarata.library.config*.

The implementation of configuration parameters is done via this module.

When a user specifies a configuration file to be applied to this software, the file is loaded and its parameters and values are loaded into the the namespace of this module. Therefore, the software internally can call these configurations as variables in this module; an example, `library.config.LYCANROC` would correspond to the `LYCANROC` configuration parameter in the configuration YAML file.

[52] https://wiki.digiultsparrow.space/en/academic/notes/astronomical-ccd-image-preprocessing

Both normal configuration parameters and secret parameters (detailed in *Configuration*) are taken from their respective files and placed into this same namespace. Therefore, all of the configuration parameters must be uniquely named.

The loading and applying of a configuration file (either secret or not), provided by the user via regular methods, is done via `opihiexarata.library.config.load_then_apply_configuration()`. Note that this will only apply the configuration to the current Python session.

Conversion

See `opihiexarata.library.conversion`.

All types of required conversions are implemented here. The OpihiExarata software has specific conventions (see *Conventions*) for units so that data may be better easily exchanged. However, some of these values needs to be converted for various reasons and so the conversions are implemented here.

Functions for converting between Julian day (convention) to other various formats are implemented.

Functions for formatting RA and DEC from degrees (convention) to sexagesimal string formatting are implemented. Specifically formatted sexagesimal can also be converted back to degrees.

Engines

See `opihiexarata.library.engine`.

Base classes for different engines and solution implementations exist here. They are typically subclassed for the actual implementation of the engines (*Services and Engines*) and solutions (*Vehicles and Solutions*). These are also useful for type checking.

Error

See `opihiexarata.library.error`.

Error exceptions specific to OpihiExarata are created here. All errors that come from OpihiExarata (either directly or indirectly) should be defined here. Using built-in Python errors is not suggested as using an error here helps specify that the issue comes from OpihiExarata.

FITS File Handing

See `opihiexarata.library.fits`.

This implements functions which assist in the reading and writing of image and table FITS files. Astropy has a lot of functionality for this, and these functions wrap around their implementation so that it is more specialized for OpihiExarata and so that the reading and writing of FITS files are uniformly applied across the software.

Type Hinting

See *opihiexarata.library.hint*.

Python is a dynamically typed language. However it implements type hints (see **PEP 483**[53] and **PEP 484**[54]) so that text editors and other development tools and features can be more accurate and detailed. OpihiExarata uses type hints throughout and highly recommends their usage. However, to avoid extremely long object calls and unnecessary importing, object types that would otherwise need to be imported to be used are instead all imported into this one namespace to be used across the codebase.

HTTP Calls

See *opihiexarata.library.http*.

Some of the functionality of OpihiExarata requires the use of HTTP APIs. Although a lot of the HTTP web functionality is implemented outside of this library where specifically needed (because of the unique nature of each process), there are some functions common among them which are implemented here.

Image Array Processing

See *opihiexarata.library.image*.

Opihi is an imaging telescope and images are often represented as arrays. However, there are some functionality that make sense in terms of images but have more involved implementations when using arrays as images. Functions here implement common manipulations of images represented as arrays.

JSON Parsing

See *opihiexarata.library.json*.

Although OpihiExarata prefers YAML formatting for configuration files and other data serializations, JSON is another popular format which is used by some of the services OpihiExarata relies on. Thus some JSON functionality is implemented here as wrapper functions.

Minor Planet Center Records

See *opihiexarata.library.mpcrecord*.

One of the most ubiquitous ways of representing an observation of an asteroid is using the MPC 80-column foarmat record[55]. However, it is not a very connivent format for Python to use and so functions which convert between the 80-column format and an Astropy table (see `astropy.table`[56], or more specifically, `astropy.`

[53] https://peps.python.org/pep-0483/
[54] https://peps.python.org/pep-0484/
[55] https://www.minorplanetcenter.net/iau/info/OpticalObs.html
[56] https://docs.astropy.org/en/stable/table/index.html#module-astropy.table

`table.Table`[57]). In general, the table format is better for internal manipulation while the 80-column format is used primarily to record and send asteroid observations to other services (including, obviously, the Minor Planet Center).

File and Directory Path Manipulations

See *opihiexarata.library.path*.

Path and filename manipulations are common across all aspects of OpihiExarata. For uniform application and convenience, common path manipulations are implemented here. This only has implementations for where the filepaths are strings and not objects.

Photometric and Astrometric Data Handing Table

See *opihiexarata.library.phototable*.

The astrometric solution and the photometric solution (see *Vehicles and Solutions*) both have a lot of similar information in tables. Older versions of this software had two different tables which were very unwieldy as progress continued. As such, this class implements a photometry table which is more coherent and comprehensive to better harmonize the interplay between the astrometric and photometric solutions. Feature expansion in this region is unlikely.

Temporary Directory

Sometimes the OpihiExarata software needs to save temporary files when processing data and reading the results. In order for these files not to mess up anything on the system this software is installed on, a temporary directory is created where the files can be created and utilized. The exact place where this directory is created is given by the configuration parameter `TEMPORARY_DIRECTORY` (see *Configuration*) Functions implemented here help with the management of this temporary directory.

Graphical User Interface

The graphical user interface (GUI) makes use of PySide6 which itself is a Python implementation of the Qt 6 GUI framework.

We use PySide6 as opposed to PyQt 6 because of licensing preferences and because PySide is developed by the Qt Company (the first party) and so it is likely to be supported for longer.

(The PySide6 is LGPL licensed. Qt Designer is GPL licensed but we only use it as a connivent tool and do not bundle or have our program require it. As far at PySide6 is concerned, we are dynamically linking it and are bound only by LGPL. See *License*.)

The exact shape and form of the GUIs are a byproduct of the features required for the *Manual Mode* and *Automatic Mode* operations. The interface is optimized to those use cases and does not otherwise follow a rigid theme.

[57] https://docs.astropy.org/en/stable/api/astropy.table.Table.html#astropy.table.Table

Qt Designer

To design the GUIs, we use Qt Designer[58] as it is an easy interface. This program can be downloaded via the Qt installer[59]. Qt Designer does not need to be installed on the same machine as it just generates the GUI design files.

When installing Pyside6, Qt Designer often is included and can be invoked by:

```
pyside6-designer
```

The GUI design files are saved in the `/OpihiExarata/src/opihiexarata/gui/qtui/` directory typically as something akin to `manual.ui`. These files may be opened by Qt Designer and modified as needed.

Building UI Files

It is part of development (not installation) to develop the Python versions of the GUI files. The UI files created via Qt Designer can be converted to their Python versions using **`pyside6-uic`**. (See Qt's documentation[60] for more information.)

The UI files are stored in `/OpihiExarata/src/opihiexarata/gui/qtui/` and it is expected that the generated Python equivalent files will also be written there as `qtui_*.py`, where the original UI filename replaces the wildcard.

For example, for a file `manual.ui`, it can be converted to the proper Python version using the command:

```
pyside6-uic manual.ui > qtui_manual.py
```

Every UI file should be turned into their respective Python version. A Powershell Core script has been written to automate this process. It can be found in the same directory and is executed by:

```
pwsh build_qtui_window.ps1
```

2.1.3 Conventions

In order for the entire software to be self-consistent, a few conventions have been established to help develop it. We detail these conventions here.

[58] https://doc.qt.io/qt-6/qtdesigner-manual.html
[59] https://doc.qt.io/qtdesignstudio/studio-installation.html
[60] https://doc.qt.io/qtforpython/tutorials/basictutorial/uifiles.html#using-ui-files-from-designer-or-qtcreator-with-quiloader-and-pyside6-uic

Units

Although there are many packages like Astropy which help deal with different units and the conversion between the two, we decided that it is easier to conform to a single set of units for the values within the software and converting only where needed via a module (see *Conversion*). We describe the units used in the following table.

Quantity	Unit
Angles	Degrees
Date-time	Julian days
Time difference	Days or seconds
Orbital elements	(Convention, see below)
Pixel scale	Arcseconds per pixel
On-sky velocity	Degrees per second
On-sky acceleration	Degrees per second squared

- We use degrees for angles because, unlike angular hours, it is much easier to use mathematically. Radians were considered but were ultimately not used because declination already used degrees and it is far less prevalent than degrees in astronomy.
- Date time, the measurement of an absolute time relative to an epoch, uses Julian days (JD). UNIX time is not as prevalent in astronomy compared to JD and MJD for the services encountered. JPL Horizons (see *JPL Horizons*) uses JD and because conversion between JD and MJD is trivial; we stuck with JD.
- Time difference, the measurement of time between two different date times, is either in days (for times on the order of hours or days) or seconds (for times on the order of seconds or minutes). We primarily use days except in cases where the numerical value is too small (for example, with *Polynomial Propagation*). The usage of which unit is documented.
- The Keplerian orbital elements are in the units that they are conventionally in. The semi-major axis a is in AU, the eccentricity e is unit-less, the inclination i is in degrees, the longitude of ascending node Ω is in degrees, the argument of perihelion ω is in degrees, and the mean anomaly M is in degrees. The epoch of these orbital elements is in Julian days, as is the convention for date-times.
- The pixel scale, as is traditionally given in most other contexts, is provided as arcseconds per pixel.
- The on-sky first order motion (i.e. the "velocity") in RA and DEC of an asteroid or other target is given in degrees per second. Degrees are used to keep both the angular units the same as the standard convention and seconds are use because they are a much more reasonable unit for the motion than days (the other time difference unit.)
- The on-sky second order motion (i.e. the "acceleration") in RA and DEC of an asteroid or other target is given in degrees per second squared. The reasoning for using both degrees and seconds is the same as provided for the on-sky first order motion.

Code Style

Formatting Style

For consistency, the Python code style that this project uses is the black[61] code style. This style is enforced by the aforementioned package, the auxillary script from *Automated Scripts* will trigger automated code formatting.

Type Hinting

Type hinting allows for many text and development environments to infer specific properties of code to provide better suggestions and other programming tools. Python supports type hinting (see **PEP 483**[62] and **PEP 484**[63]) and this software adapts type hinting. Although it is not required for the program to run, it both makes the code easier to work with because of the added programming tool functionality along with conveying more information about the code to future maintainers. There is very little downside. There is existing infrastructure for making using type hinting easier to use from *Type Hinting*.

Documentation Files

The documentation is automatically generated using Sphinx. Because of this, there are some peculiarities when writing the documentation, we document them here.

Main Body Files

The main body of documentation uses reStructuredText files as markup. A Sphinx specific primer can be found on their website[64]. A more complete reference for reStructuredText can be found in their documentation[65].

Image Files

Images are important in providing additional information in documentation. Because the documentation can be built into either HTML or LaTeX forms, special care should be exercised for images as both take different file formats.

When calling an image or figure file, the file path in the reStructuredText should have the file extension as a wildcard so that the Sphinx builders can choose the file format best suited for the form the documentation is going to be built into.

When creating images, please adhere to the following:

- If the image is a raster image, save the image as a PNG file.

[61] https://pypi.org/project/black/
[62] https://peps.python.org/pep-0483/
[63] https://peps.python.org/pep-0484/
[64] https://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html
[65] https://docutils.sourceforge.io/rst.html

- If the image is a vector graphics file, save the image as both a SVG and PDF file. (HTML supports only SVG and LaTeX supports only PDF, so both are needed.)

Python Docstrings

For Sphinx to properly load and process the documentation strings from the Python files themselves, they need to be marked up in a specific way. Natively it would be in reStructuredText, but it looks ugly. Instead, we use the Napoleon extension for Sphinx to allow for the usage of NumPy docstring formatting[66]. An example of Numpy formatting is also provided by Sphinx[67].

2.1.4 Algorithms

Here we detail the methods behind a few of the algorithms that we have implemented ourselves. The algorithms and implementations of other systems (especially some of the engines and services of *Services and Engines*) are outside the scope of this documentation. See *Citations* for further references on algorithms implemented by others that we use.

We detail the algorithms here so that its methods (and some particularities thereof) are documented and some of the choice made are not lost in future development and upkeep of this software.

See the sidebar (for the web version of this documentation) for the details on specific algorithms.

Polynomial Propagation

Polynomial propagation is one of the different implemented propagation engines (see *Services and Engines*). Conceptually it is one of the easiest methods. Simply put, it just computes the rate of right ascension and declination change as independent rates from a collection of previous observations.

The use of polynomial propagation assumes the following:

- The set of taken asteroid observations (and desired future positions) are within a relatively short timespan, usually within 24 hours.
- The tangent plane projection of the sky is valid and the effects of curvature because of the spherical sky are negligible.
- The motion of the asteroid across the sky within the provided time span is small enough such that the aforementioned tangent plane projection still holds.

[66] https://numpydoc.readthedocs.io/
[67] https://www.sphinx-doc.org/en/master/usage/extensions/example_numpy.html

Coordinate Conversion

The only required observational data for this method of propagation is the RA and DEC of the observations, α and δ, along with the time of observations t. Because of the conventions of OpihiExarata (see *Conventions*) the units of these are degrees and Julian days. (For notation purposes, the variables stand for all taken observations.)

We found that using these units for fitting the rates were inadequate and suffered from two different problems.

1. The Julian day time difference was often much less than one and so the fitting algorithms became unstable and could properly not fit the observations.
2. Angles are cyclical and the fitting a function to cyclical points are not stable using quick methods.

To solve the first problem, we converted the time of observations to UNIX time. Because it is in seconds, the numerical difference between each data point is much larger and it seems that this change produces more accurate results. Of course the propagation function itself (which has Julian days as input) needs to covert the input date to UNIX time as the fitting constants are now in the UNIX time domain.

To solve the second problem, we convert the angles so that they can be treated linearly. More specifically, as the angular coordinates provided are between $0 \leq \angle \leq 360$ or $-90 \leq \angle \leq 90$, if the maximum difference is (numerically) greater than ≈ 135 degrees in either coordinate, then it guaranteed that a loop around happened for that coordinate. If it was motion that caused this difference, then it must have violated one of the assumptions and so this method is useless for propagation. We use the most recent observation to set the range, the other points are cycled until no wrap around occurs.

Fitting Coordinates

We treat both α and δ as separate and independent coordinates and we also treat their on-sky rates as independent of each other. This is valid as long as the timescales are relatively small and the tangent plane projection of the sky holds.

As such, we have two functions $A(f)$ and $D(f)$ which is the path of the asteroid across the sky over time f in right ascension (A) and declination (D).

We can approximate these functions using polynomials. We fit our n order polynomial P_n for both right ascension and declination to get approximations to these functions:

$$\begin{aligned} \text{fit}[P_n(t) = \alpha] &\implies A(f) \approx P_{n,A}(f) \mapsto (f, \alpha) \\ \text{fit}[P_n(t) = \delta] &\implies D(f) \approx P_{n,D}(f) \mapsto (f, \delta) \end{aligned}$$

Simply put, when we have a fit to the (t, α) and (t, δ) pairs using polynomial functions, we use these functions as approximations to A and D and use these approximations to derive the future location.

Provided the future time f where the user wants to predict the future right ascension α_f and declination δ_f, we can easily compute our estimations as:

$$\begin{aligned} P_{n,A}(f) &= \alpha_f \\ P_{n,D}(f) &= \delta_f \end{aligned}$$

We use low order polynomials as they are simple functions to both fit and interpret. We implement both first order (linear) polynomial (`opihiexarata.propagate.polynomial.LinearPropagationEngine`) and second order (quadratic) polynomial (`opihiexarata.propagate.polynomial.QuadraticPropagationEngine`). Higher order polynomials and

more complicated functions often over-fit (and are otherwise overcomplicated) as typically there are only a few observations and thus few (α, δ, t) sets for fitting.

Wrap Around

In some cases, the approximate angular coordinates given by $P_{n,A}(f)$ and $P_{n,D}(f)$ may exceed the standard angular bounds of $0 \leq \angle \leq 360$ or $-90 \leq \angle \leq 90$ for right ascension and declination respectively. As such, we can convert the angles back into this range R (in degrees) using:

$$\begin{aligned} \alpha_{f,R} &= (\alpha_f + 360) \mod 360 \\ \delta_{f,R} &= |((\delta_f - 90) \mod 360) - 180| - 90 \end{aligned}$$

These values, still in degrees per *Conventions*, are then passed to the systems which need this prediction of the asteroid's location.

Spherical Kinematics

Here we detail the algorithm for computing the propagation of the position of an asteroid given observations. This method uses the laws of kinematic motion in spherical coordinates.

Deprecated since version 2022.2.1: This method has not been implemented nor have the calculations below been proven to be correct and work for the purposes of being a PropagationEngine.

Definitions

We are using spherical coordinates, namely the coordinates (r, θ, ϕ) for the radial distance, polar angle, and azimuthal angle respectively. In relation to astrometric coordinates RA and DEC, (α, δ):

$$\alpha = \phi \qquad \delta = \frac{\pi}{2} - \theta$$

Observations taken by the Opihi telescoped and processed by OpihiExarata are represented as $(\alpha_n, \delta_n, t_n)$. These correspond to the temporal spherical coordinates $(r = 1, \phi_n, \delta_n, t_n)$. For t is the absolute time of the observation; UNIX time or Julian date time works best.

Moreover, in spherical coordinates, the position, velocity, and acceleration vectors are given as: (See Keplerian Ellipses Chapter 2 Reed 2019[68].)

$$\begin{aligned} \mathbf{r} &= r\boldsymbol{\hat{\mathbf{r}}} \\ \mathbf{v} &= \dot{r}\boldsymbol{\hat{\mathbf{r}}} + r\dot{\theta}\hat{\boldsymbol{\theta}} + r\dot{\phi}\sin\theta\hat{\boldsymbol{\phi}} \\ \mathbf{a} &= \left(\ddot{r} - r\dot{\theta}^2 - r\dot{\phi}^2\sin^2\theta\right)\boldsymbol{\hat{\mathbf{r}}} \\ &\quad + \left(r\ddot{\theta} + 2\dot{r}\dot{\theta} - r\dot{\phi}^2\sin\theta\cos\theta\right)\hat{\boldsymbol{\theta}} \\ &\quad + \left(r\ddot{\phi}\sin\theta + 2\dot{r}\dot{\phi}\sin\theta + 2r\dot{\theta}\dot{\phi}\cos\theta\right)\hat{\boldsymbol{\phi}} \end{aligned}$$

[68] http://www.worldcat.org/oclc/1104053368

Where the basis vectors of the spherical coordinates are:

$$\hat{\mathbf{r}} = \sin\theta\cos\phi\hat{\mathbf{x}} + \sin\theta\sin\phi\hat{\mathbf{y}} + \cos\theta\hat{\mathbf{z}}$$
$$\hat{\boldsymbol{\theta}} = \cos\theta\cos\phi\hat{\mathbf{x}} + \cos\theta\sin\phi\hat{\mathbf{y}} - \sin\theta\hat{\mathbf{z}}$$
$$\hat{\boldsymbol{\phi}} = -\sin\phi\hat{\mathbf{x}} + \cos\phi\hat{\mathbf{y}} + 0\hat{\mathbf{z}}$$

However, for the purposes of finding by propagation, we only care about the location of the asteroid on the sky as determined by the celestial coordinates. The distance from the origin of the coordinate system does not change. Thus:

$$r = 1 \qquad \dot{r} = \ddot{r} = 0$$

And thus the kinematic vectors are:

$$\mathbf{r} = \boldsymbol{\imath}$$
$$\mathbf{v} = \dot{\theta}\hat{\boldsymbol{\theta}} + \dot{\phi}\sin\theta\hat{\boldsymbol{\phi}}$$
$$\mathbf{a} = \left(-\dot{\theta}^2 - \dot{\phi}^2\sin^2\theta\right)\boldsymbol{\imath} + \left(\ddot{\theta} - \dot{\phi}^2\sin\theta\cos\theta\right)\hat{\boldsymbol{\theta}} + \left(\ddot{\phi}\sin\theta + 2\dot{\theta}\dot{\phi}\cos\theta\right)\hat{\boldsymbol{\phi}}$$

We can convert these vectors to Cartesian coordinates using the following matrix transformation. The spherical to Cartesian transformation matrix $\mathbf{R}$ can be derived from the defining angles of the spherical coordinate system θ and ϕ. Where $\mathbf{u}_{\text{cart}} = \mathbf{R}\mathbf{u}_{\text{sph}}$.:

$$\mathbf{R} = \begin{bmatrix} \sin\theta\cos\phi & \cos\theta\cos\phi & -\sin\phi \\ \sin\theta\sin\phi & \cos\theta\sin\phi & \cos\phi \\ \cos\theta & -\sin\theta & 0 \end{bmatrix}$$

Thus, in Cartesian coordinates:

$$\mathbf{r} = \mathbf{R}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$
$$\mathbf{v} = \mathbf{R}\begin{bmatrix} 0 \\ \dot{\theta} \\ \dot{\phi}\sin\theta \end{bmatrix}$$
$$\mathbf{a} = \mathbf{R}\begin{bmatrix} -\dot{\theta}^2 - \dot{\phi}^2\sin^2\theta \\ \ddot{\theta} - \dot{\phi}^2\sin\theta\cos\theta \\ \ddot{\phi}\sin\theta + 2\dot{\theta}\dot{\phi}\cos\theta \end{bmatrix}$$

Generally, as kinematics are defined based on the time derivatives, the general equation of motion for constant acceleration can be used to determine the movement of the position vector representing the asteroid:

$$\mathbf{r} \triangleq \mathbf{r} \qquad \mathbf{v} \triangleq \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} \qquad \mathbf{a} \triangleq \frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}t^2} \qquad \mathbf{j} \triangleq \frac{\mathrm{d}^3\mathbf{r}}{\mathrm{d}t^3} = 0$$

$$\mathbf{r} = \mathbf{r}_0 + \mathbf{v}_0\tau + \frac{1}{2}\mathbf{a}\tau^2$$

Where τ is the time interval between the defining time of observation to the current time.

Deriving Rates

Multiple observations from Opihi provides multiple sightings of an asteroid at many different points in the sky, providing multiple RA and DEC coordinates, α_n and δ_n at time t_n. We have a total of N RA-DEC observations. (The propagation calculation will need to be redone for a new observation set $N' = N + 1$.)

We can convert this to spherical coordinates with $\phi_n = \alpha_n$ and $\theta_n = \frac{\pi}{2} - \delta_n$.

These multiple observations allows for the determination of the rates of change of spherical coordinates for the asteroid, namely: (For the time difference $t_\Delta = t_{n+1} - t_n$.)

$$\dot{\theta}_p = \frac{\theta_{n+1} - \theta_n}{t_{n+1} - t_n}$$
$$\dot{\phi}_p = \frac{\phi_{n+1} - \phi_n}{t_{n+1} - t_n}$$
$$t'_p = \frac{1}{2}\left(t_{n+1} + t_n\right)$$

and,

$$\ddot{\theta}_q = \frac{\dot{\theta}_{p+1} - \dot{\theta}_p}{t'_{p+1} - t'_p}$$
$$\ddot{\phi}_q = \frac{\dot{\phi}_{p+1} - \dot{\phi}_p}{t'_{p+1} - t'_p}$$

The first order rates changes over time. As such, it is required that two observations be reserved as special observations which the first order rates are calculated and to established the spherical coordinate system itself. Although it does not need to be the first two observations, it is often connivent to use them. As such, using the first two observations $n = 0$ and $n = 1$, we have:

$$\theta = \theta_0$$
$$\phi = \phi_0$$
$$\dot{\theta} = \dot{\theta}_0 = \frac{\theta_1 - \theta_0}{t_1 - t_0}$$
$$\dot{\phi} = \dot{\phi}_0 = \frac{\phi_1 - \phi_0}{t_1 - t_0}$$

Because we assume constant acceleration ($\mathbf{j} = 0$), the second differential values are assumed to be constant and thus an average is more representational of the value. (A mean or median is valid.)

$$\ddot{\theta} = \frac{1}{Q}\sum_q^Q \ddot{\theta}_q \approx \text{median}(\ddot{\theta}_q)$$
$$\ddot{\phi} = \frac{1}{Q}\sum_q^Q \ddot{\phi}_q \approx \text{median}(\ddot{\phi}_q)$$

In the case for $N = 2$, then the total number of derived angular first order rates is $P = 1$. As such the second order rates cannot be calculated and $Q = 0$ (the cardinality of the arrays are zero). By default, for this special case:

$$\#(\ddot{\theta}_q) = \#(\ddot{\phi}_q) = 0 \implies Q = 0 \longrightarrow \ddot{\theta} = 0 \quad \ddot{\phi} = 0$$

Spherical Motion

With the 0th, 1st, and 2nd order rates calculated from the set of N observations, the kinematic vectors can be calculated. The special observations defining the coordinate system and the velocities also define the initial vectors from which kinematics shall be applied to. The acceleration vector, being constant means $\mathbf{a}_0 = \mathbf{a}$. Namely, these vectors are, in Cartesian coordinates,

$$\mathbf{r_0} = \begin{bmatrix} \sin\theta\cos\phi & \cos\theta\cos\phi & -\sin\phi \\ \sin\theta\sin\phi & \cos\theta\sin\phi & \cos\phi \\ \cos\theta & -\sin\theta & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{v_0} = \begin{bmatrix} \sin\theta\cos\phi & \cos\theta\cos\phi & -\sin\phi \\ \sin\theta\sin\phi & \cos\theta\sin\phi & \cos\phi \\ \cos\theta & -\sin\theta & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ \dot{\phi}\sin\theta \end{bmatrix}$$

$$\mathbf{a} = \begin{bmatrix} \sin\theta\cos\phi & \cos\theta\cos\phi & -\sin\phi \\ \sin\theta\sin\phi & \cos\theta\sin\phi & \cos\phi \\ \cos\theta & -\sin\theta & 0 \end{bmatrix} \begin{bmatrix} -\dot{\theta}^2 - \dot{\phi}^2\sin^2\theta \\ \ddot{\theta} - \dot{\phi}^2\sin\theta\cos\theta \\ \ddot{\phi}\sin\theta + 2\dot{\theta}\dot{\phi}\cos\theta \end{bmatrix}$$

All three of these vectors are constant in future time. The position at a set of future observations at time(s) t_i^+ can be calculated using the kinematic equation; the time intervals τ_i being $\tau_i = t_i^+ - t_0$:

$$\mathbf{r}_i^+ = \mathbf{r}_0 + \mathbf{v}_0\left(t_i^+ - t_0\right) + \frac{1}{2}\mathbf{a}\left(t_i^+ - t_0\right)^2$$

Celestial Sphere

These new future position vectors $\mathbf{r}_i^+$ are in Cartesian coordinates. The calculations should be done in Cartesian, provided the conversion earlier.

Each position vector can be represented as:

$$\mathbf{r}_i^+ = X_i\mathbf{x} + Y_i\mathbf{y} + Z_i\mathbf{z} = \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix}$$

These Cartesian coordinate position vectors, centered on the origin, represents where the asteroid is on the celestial sphere in the future at an observation time of t_i^+. From these Cartesian coordinates, we can extract their location in spherical coordinates,

$$r_i^+ = \sqrt{X_i^2 + Y_i^2 + Z_i^2}$$

$$\theta_i^+ = \arccos\left(\frac{Z_i}{r_i^+}\right) = \arccos\left(\frac{Z_i}{\sqrt{X_i^2 + Y_i^2 + Z_i^2}}\right)$$

$$\phi_i^+ = \text{arctan2}(Y_i, X_i) \simeq \arctan\left(\frac{Y_i}{X_i}\right)$$

Note: In order to properly handle the quadrant issue, the 2-argument arctangent is required. Moreover, if the 2-argument arctangent function returns in a range $-\pi \leq \angle \leq \pi$, it can be converted to the usual range of $0 \leq \phi \leq 2\pi$ with: $\phi = \angle \mod 2\pi$ or $\phi = \angle \mod 360^\circ$

These spherical coordinate locations can then be converted into future RA and DEC temporal coordinates $(\alpha_i^+, \delta_i^+, t_i^+)$:

$$\begin{aligned}\alpha_i^+ &= \phi_i^+ \\ \delta_i^+ &= \frac{\pi}{2} - \theta_i^+ \\ t_i^+ &= t_i^+\end{aligned}$$

Lemmas

Derivation of Vector Equation of Motion

Newton's second law and constant acceleration stipulates:

$$\mathbf{F} = m\mathbf{a} = m\ddot{\mathbf{r}} \qquad \dot{\mathbf{F}} = 0$$

This thus provides the differential equation of motion (For constant $\mathbf{F}$.)

$$\ddot{\mathbf{r}} = \frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}t^2} = \frac{\mathbf{F}}{m}$$

We define based on the laws of integrations (and in essence the fundamental theorem of calculus):

$$\dot{\mathbf{f}} \triangleq \frac{\mathrm{d}\mathbf{f}}{\mathrm{d}t} \iff \int \dot{\mathbf{f}}\mathrm{d}t = \mathbf{f} + \mathbf{C}$$

$$\int \frac{\mathrm{d}\mathbf{f}}{\mathrm{d}t}\mathrm{d}t = \mathbf{f}$$

We can solve the differential equation of motion:

$$\begin{aligned}\ddot{\mathbf{r}} &= \frac{\mathbf{F}}{m} \\ \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t}\right) &= \frac{\mathbf{F}}{m} \\ \int \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t}\right)\mathrm{d}t = \int \frac{\mathbf{F}}{m}\mathrm{d}t &= \frac{\mathbf{F}}{m}\int 1\mathrm{d}t = \frac{\mathbf{F}}{m}t + \mathbf{C_1} \\ \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} &= \frac{\mathbf{F}}{m}t + \mathbf{C_1} \\ \int \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t}\mathrm{d}t = \int \frac{\mathbf{F}}{m}t + \mathbf{C_1}\mathrm{d}t &= \frac{\mathbf{F}}{m}\int t\mathrm{d}t + \int \mathbf{C_1}\mathrm{d}t = \frac{\mathbf{F}}{m}\frac{1}{2}t^2 + \mathbf{C_1}t + \mathbf{C_2} \\ \mathbf{r} &= \frac{\mathbf{F}}{m}\frac{1}{2}t^2 + \mathbf{C_1}t + \mathbf{C_2}\end{aligned}$$

For the initial conditions:

$$\begin{aligned}t = 0 &\implies \mathbf{r} = \mathbf{C_2} = \mathbf{r_0} \\ t = 0 &\implies \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} = \mathbf{C_1} = \mathbf{v_0} \\ \dot{\mathbf{F}} = 0 &\implies \frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}t^2} = \frac{\mathbf{F}}{m} = \mathbf{a_0} = \mathbf{a}\end{aligned}$$

Thus, the total valid solution is:

$$\mathbf{r} = \mathbf{r_0} + \mathbf{v_0}t + \frac{1}{2}\mathbf{a}t^2$$

2.1.5 License

This software is currently licensed under the MIT License[69]. (See the `LICENSE` file in the main `OpihiExarata/` directory.)

For license considerations because of the dependencies of OpihiExarata.

This software makes use of Pyside6[70] which is licensed using the LGPL license. As we use it as a library and do not change its internals, this software is not bound by the LGPL license[71]. (See *Graphical User Interface*.)

This software makes use of Astrometry.net services which are licensed under the 3-clause BSD-style license[72]. We do not redistribute that software but instead require its separate installation. Therefore, we are not bound by the GPLv3[73] requirements which Astrometry.net is bound by for some of its components.

This software makes use of OrbFit software which are licensed under the GPLv3 license[74]. We do not redistribute that software but instead require its separate installation. Therefore, we are not bound by the GPLv3 requirements.

Python, and all of the package dependencies used, utilize BSD 3-clause[75], Apache 2.0[76], HPND licenses[77], or similar MIT-like licenses and are compatible.

[69] https://opensource.org/licenses/MIT
[70] https://wiki.qt.io/Qt_for_Python
[71] https://opensource.org/licenses/LGPL-3.0
[72] https://opensource.org/licenses/BSD-3-Clause
[73] https://opensource.org/licenses/GPL-3.0
[74] https://opensource.org/licenses/GPL-3.0
[75] https://opensource.org/licenses/BSD-3-Clause
[76] https://opensource.org/licenses/Apache-2.0
[77] https://opensource.org/licenses/HPND

CHAPTER

THREE

CODE MANUAL

The code manual is primarily for software maintainers and other IRTF staff. It details the software API documentation of OpihiExarata and its inner workings. Unlike the *Technical Manual*, this does not detail any of the design principles of the software but instead is the function and class documentation of the software (generated by Sphinx).

Code coverage[78] is available in html. The code manual (and overall documentation) is generated via Sphinx and the code coverage is generated by a different tool. As such, it is manually linked rather than integrated.

3.1 opihiexarata

3.1.1 opihiexarata package

Subpackages

opihiexarata.astrometry package

Submodules

opihiexarata.astrometry.solution module

The astrometric solution class.

class opihiexarata.astrometry.solution.**AstrometricSolution**(*fits_filename: str*, *solver_engine:* AstrometryEngine, *vehicle_args: dict = {}*)

Bases: *ExarataSolution*

The primary class describing an astrometric solution, based on an image provided.

This class is the middlewere class between the engines which solve the astrometry, and the rest of the OpihiExarata code.

[78] https://psmd-iberutaru.github.io/OpihiExarata/build/html/code/coverage/index.html

_original_filename

The original filename where the fits file is stored at, or copied to.

Type
string

_original_header

The original header of the fits file that was pulled to solve for this astrometric solution.

Type
Header

_original_data

The original data of the fits file that was pulled to solve for this astrometric solution.

Type
array-like

skycoord

The sky coordinate which describes the current astrometric solution.

Type
SkyCoord

ra

The right ascension of the center of the image, in decimal degrees.

Type
float

dec

The declination of the center of the image, in decimal degrees.

Type
float

orientation

The angle of orientation that the image is at, in degrees.

Type
float

radius

The radius of the image, or more specifically, the approximate radius that the image covers in the sky, in degrees.

Type
float

pixel_scale

The pixel scale of the image, in arcseconds per pixel.

Type
float

wcs

The world coordinate solution unified interface provided by Astropy for interface to the world coordinate which allows conversion between sky and pixel spaces.

Type
Astropy WCS

star_table

A table detailing the correlation of star locations in both pixel and celestial space.

Type
Table

__init__ (*fits_filename: str*, *solver_engine:* AstrometryEngine, *vehicle_args: dict = {}*) → None

Solving the astrometry via the image provided. The engine class must also be provided.

Parameters

- **fits_filename** (*string*) – The path of the fits file that contains the data for the astrometric solution.
- **solver_engine** (AstrometryEngine) – The astrometric solver engine class. This is what will act as the "behind the scenes" and solve the field, using this middleware to translate it into something that is easier.
- **vehicle_args** (*dictionary*) – If the vehicle function for the provided solver engine needs extra parameters not otherwise provided by the standard input, they are given here.

Return type
None

pixel_to_sky_coordinates (*x: Union[float, ndarray[79]]*, *y: Union[float, ndarray[80]]*) → tuple[Union[float, numpy.ndarray[81]], Union[float, numpy.ndarray[82]]]

Compute the RA and DEC sky coordinates of this image provided the pixel coordinates. Floating point pixel values are supported.

This is a wrapper around the WCS-based method.

Parameters

- **x** (*float or array-like*) – The x pixel coordinate in the x-axis direction.
- **y** (*float or array-like*) – The y pixel coordinate in the y-axis direction.

Returns

- **ra** (*float or array-like*) – The right ascension of the pixel coordinate, in degrees.
- **dec** (*float or array-like*) – The declination of the pixel coordinate, in degrees.

sky_to_pixel_coordinates (*ra: Union[float, ndarray[83]]*, *dec: Union[float, ndarray[84]]*) → tuple[Union[float, numpy.ndarray[85]], Union[float, numpy.ndarray[86]]]

Compute the x and y pixel coordinates of this image provided the RA DEC sky coordinates.

This is a wrapper around the WCS-based method.

Parameters

- **ra** (*float or array-like*) – The right ascension of the pixel coordinate, in degrees.
- **dec** (*float or array-like*) – The declination of the pixel coordinate, in degrees.

Returns

- **x** (*float or array-like*) – The x pixel coordinate in the x-axis direction.
- **y** (*float or array-like*) – The y pixel coordinate in the y-axis direction.

opihiexarata.astrometry.solution.**_vehicle_astrometrynet_web_api**(*fits_filename: str*) → dict

A vehicle function for astrometric solutions. Solve the fits file astrometry using the astrometry.net nova web API.

Parameters
fits_filename (*string*) – The path of the fits file that contains the data for the astrometric solution.

Returns
astrometry_results – A dictionary containing the results of the astrometric solution.

Return type
dict

opihiexarata.astrometry.webclient module

class opihiexarata.astrometry.webclient.**AstrometryNetWebAPIEngine**(*url=None*, *apikey: Optional[str] = None*, *silent: bool = True*)

[79] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[80] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[81] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[82] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[83] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[84] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[85] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[86] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

Bases: *AstrometryEngine*

A python-based wrapper around the web API for astrometry.net.

This API does not have the full functionality of the default Python client seen at https://github.com/dstndstn/astrometry.net/blob/master/net/client/client.py. The point of this class is to be simple enough to be understood by others and be specialized for OpihiExarata.

_ASTROMETRY_NET_API_BASE_URL

The base URL for the API which all other service URLs are derived from.

Type
string

_apikey

The API key used to log in.

Type
string

original_upload_filename

The original filename that was used to upload the data.

Type
string

session

The session ID of this API connection to astrometry.net

Type
string

__del_job_id() → None

Remove the current job ID association.

__del_submission_id() → None

Remove the current submission ID association.

__doc_job_id = 'When file upload or table upload is sent to the API, the job ID of the submission is saved here.'

__doc_submission_id = 'When file upload or table upload is sent to the API, the submission ID is saved here.'

__get_job_id() → str

Extract the job ID from the image upload results. It may be the case that there is not job yet associated with this submission.

__get_submission_id() → str

Extract the submission ID from the image upload results.

__init__(*url=None*, *apikey: Optional[str] = None*, *silent: bool = True*) → None

The instantiation, connecting to the web API using the API key.

Parameters

- **url** (*string, default = None*) – The base url which all other API URL links are derived from. This should be used if the API is a self-hosted install or has a different web source than nova.astrometry.net. Defaults to the nova.astrometry.net api service.
- **apikey** (*string*) – The API key of the user.
- **silent** (*bool, default = True*) – Should there be printed messages as the processes are executed. This is helpful for debugging or similar processes.

Return type
None

__job_id = None

__login(*apikey: str*) → str

The method to log into the API system.

Parameters
apikey (*string*) – The API key for the web API service.

Returns
session_key – The session key for this login session.

Return type
string

__set_job_id(*job_id*) → None

Assign the job ID, it should only be done once when the image is obtained.

__set_submission_id(*sub_id*) → None

Assign the submission ID, it should only be done once when the image is obtained.

__submission_id = None

_generate_service_url(*service: str*) → str

Generate the correct URL for the desired service. Because astrometry.net uses a convention, we can follow it to obtain the desired service URL.

Parameters
service (*str*) – The service which the API URL for should be generated from.

Returns
url – The URL for the service.

Return type
str

_generate_upload_args(***kwargs*) → dict

Generate the arguments for sending a request. This constructs the needed arguments, replacing the defaults with user provided arguments where desired.

Parameters
****kwargs** (*dict*) – Arguments which would override the defaults.

Returns
args – The arguments which can be used to send the request.

Return type
dict

_send_web_request(*service: str*, *args: dict = {}*, *file_args: Optional[dict] = None*) → dict

A wrapper function for sending a webrequest to the astrometry.net API service. Returns the results as well.

Parameters

- **service** (*string*) – The service which is being requested. The web URL is constructed from this string.
- **args** (*dictionary, default = {}*) – The arguments being sent over the web request.
- **file_args** (*dictionary, default = None*) – If a file is being uploaded instead, special care must be taken to sure it matches the upload specifications.

Returns
results – The results of the web request if it did not fail.

Return type
dictionary

download_result_file(*filename: str*, *file_type: str*, *job_id: Optional[str] = None*) → None

Downloads fits data table files which correspond to the job id.

Parameters

- **filename** (*str*) – The filename of the file when it is downloaded and saved to disk.
- **file_type** (*str*) – The type of file to be downloaded from astrometry.net. It should one of the following:
  - *wcs*: The world corrdinate data table file.
  - *new_fits*, *new_image*: A new fits file, containing the original image, annotations, and WCS header information.
  - *rdls*: A table of reference stars nearby.
  - *axy*: A table in of the location of stars detected in the provided image.
  - *corr*: A table of the correspondences between reference stars location in the sky and in pixel space.

- **job_id**(*str, default = None*) – The ID of the job that the results should be obtained from. If not provided, the ID determined by the file upload is used.

Return type
None

get_job_results(*job_id: Optional[str] = None*) → dict

Get the results of a job sent to the API service.

Parameters
job_id(*str, default = None*) – The ID of the job that the results should be obtained from. If not provided, the ID determined by the file upload is used.

Returns

results – The results of the astrometry.net job. They are, in general: (If the job has not finished yet, None is returned.)

- Status : The status of the job.
- Calibration : Calibration of the image uploaded.
- Tags : Known tagged objects in the image, people inputted.
- Machine Tags : Ditto for tags, but only via machine inputs.
- Objects in field : Known objects in the image field.
- Annotations : Known objects in the field, with annotations.
- Info : A collection of most everything above.

Return type
dict

get_job_status(*job_id: Optional[str] = None*) → str

Get the status of a job specified by its ID.

Parameters
job_id(*str, default = None*) – The ID of the job that the results should be obtained from. If not provided, the ID determined by the file upload is used.

Returns
status – The status of the submission. If the job has not run yet, None is returned instead.

Return type
string

get_reference_star_pixel_correlation(*job_id: Optional[str] = None*, *temp_filename: Optional[str] = None*, *delete_after: bool = True*) → Table

This obtains the table that correlates the location of reference stars and their pixel locations. It is obtained from the fits corr file that is downloaded into a temporary directory.

Parameters

- **job_id** (*string, default = None*) – The ID of the job that the results should be obtained from. If not provided, the ID determined by the file upload is used.
- **temp_filename** (*string, default = None*) – The filename that the downloaded correlation file will be downloaded as. The path is going to still be in the temporary directory.
- **delete_after** (*bool, default = True*) – Delete the file after downloading it to extract its information.

Returns
correlation_table – The table which details the correlation between the coordinates of the stars and their pixel locations.

Return type
Table

get_submission_results (*submission_id: Optional[str] = None*) → dict

Get the results of a submission specified by its ID.

Parameters
submission_id (*str*) – The ID of the submission. If it is not passed, the ID determined by the file upload is used.

Returns
result – The result of the submission.

Return type
dict

get_submission_status (*submission_id: Optional[str] = None*) → str

Get the status of a submission specified by its ID.

Parameters
submission_id (*str, default = None*) – The ID of the submission. If it is not passed, the ID determined by the file upload is used.

Returns
status – The status of the submission.

Return type
string

get_wcs (*job_id: Optional[str] = None*, *temp_filename: Optional[str] = None*, *delete_after: bool = True*) → WCS

This obtains the wcs header file and then computes World Coordinate System solution from it. Because astrometry.net computes it for us, we just extract it from the header file using Astropy.

Parameters

- **job_id** (*string, default = None*) – The ID of the job that the results should be obtained from. If not provided, the ID determined by the file upload is used.

- **temp_filename** (*string, default = None*) – The filename that the downloaded wcs file will be downloaded as. The path is going to still be in the temporary directory.
- **delete_after** (*bool, default = True*) – Delete the file after downloading it to extract its information.

Returns
wcs – The world coordinate solution class for the image provided.

Return type
Astropy WCS

property job_id: str
When file upload or table upload is sent to the API, the job ID of the submission is saved here.

property submission_id: str
When file upload or table upload is sent to the API, the submission ID is saved here.

upload_file(*pathname: str*, ***kwargs*) → dict
A wrapper to allow for the uploading of files or images to the API.

This also determines the submission ID and the job ID for the uploaded image and saves it.

Parameters
pathname (*str*) – The pathname of the file to open. The filename is extracted and used as well.

Returns
results – The results of the API call to upload the image.

Return type
dict

Module contents

opihiexarata.ephemeris package

Submodules

opihiexarata.ephemeris.jplhorizons module

The different ephemeris engines which use JPL horizons as its backend.

class opihiexarata.ephemeris.jplhorizons.**JPLHorizonsWebAPIEngine**(*semimajor_axis: float*, *eccentricity: float*, *inclination: float*, *longitude_ascending_node: float*, *argument_perihelion: float*, *mean_anomaly: float*, *epoch: float*)

Bases: *EphemerisEngine*

This obtains the ephemeris of an asteroid using JPL horizons provided the Keplerian orbital elements of the asteroid as determined by orbital solutions.

semimajor_axis

The semi-major axis of the orbit, in AU.

Type
float

eccentricity

The eccentricity of the orbit.

Type
float

inclination

The angle of inclination of the orbit, in degrees.

Type
float

longitude_ascending_node

The longitude of the ascending node of the orbit, in degrees.

Type
float

argument_perihelion

The argument of perihelion of the orbit, in degrees.

Type
float

mean_anomaly
The mean anomaly of the orbit, in degrees.

Type
float

epoch
The modified Julian date epoch of the osculating orbital elements.

Type
float

ra_velocity
The right ascension angular velocity of the target, in degrees per second.

Type
float

dec_velocity
The declination angular velocity of the target, in degrees per second.

Type
float

ra_acceleration
The right ascension angular acceleration of the target, in degrees per second squared.

Type
float

dec_acceleration
The declination angular acceleration of the target, in degrees per second squared.

Type
float

__init__(*semimajor_axis: float*, *eccentricity: float*, *inclination: float*, *longitude_ascending_node: float*, *argument_perihelion: float*, *mean_anomaly: float*, *epoch: float*) → None
Creating the engine.

Parameters

- **semimajor_axis** (*float*) – The semi-major axis of the orbit, in AU.
- **eccentricity** (*float*) – The eccentricity of the orbit.
- **inclination** (*float*) – The angle of inclination of the orbit, in degrees.
- **longitude_ascending_node** (*float*) – The longitude of the ascending node of the orbit, in degrees.
- **argument_perihelion** (*float*) – The argument of perihelion of the orbit, in degrees.

- **mean_anomaly** (*float*) – The mean anomaly of the orbit, in degrees.
- **epoch** (*float*) – The full Julian date epoch of these osculating orbital elements.

Return type
None

__parse_jpl_horizons_output() → Table

This function serves to parse the output from the JPL horizons. It is a text output that is human readable but some parsing is needed. We do that here, assuming the quantities in the original request.

Parameters
response_text (*str*) – The raw response from the JPL horizons web API service.

_query_jpl_horizons(*start_time: float*, *stop_time: float*, *time_step: float*) → Table

This function queries JPL horizons. Using the current orbital elements, and provided a minimum time, maximum time, and time step, we can fetch the new table of ephemeris measurements.

Parameters

- **start_time** (*float*) – The time that the ephemeris should start at, in Julian days.
- **stop_time** (*float*) – The time that the ephemeris should stop at, in Julian days.
- **time_step** (*float*) – The time step of the entries of the ephemeris calculation, in seconds. (Note, JPL does not accept anything less than a minute.)

Returns
epidermis_table – The table of computed sky locations over time from JPL horizons.

Return type
Table

_refresh_ephemeris(*start_time: Optional[float] = None*, *stop_time: Optional[float] = None*, *time_step: Optional[float] = None*) → None

This function refreshes the calculations rederiving the solution table and the ephemeris function.

Parameters

- **start_time** (*float, default = None*) – The time that the ephemeris should start at, in Julian days. If not provided, then the old start time will be used instead.
- **stop_time** (*float, default = None*) – The time that the ephemeris should stop at, in Julian days. If not provided, then the old stop time will be used instead.
- **time_step** (*float, default = None*) – The time step of the entries of the ephemeris calculation, in seconds. (Note, JPL does not accept anything less than a minute.) If not provided, then the old time step will be used instead.

Return type
None

forward_ephemeris(*future_time: float*) → tuple[float, float]

This allows the computation of future positions at a future time using the derived ephemeris.

Parameters
future_time (*array-like*) – The set of future times which to derive new RA and DEC coordinates. The time must be in Julian day time.

Returns

- **future_ra** (*ndarray*) – The set of right ascensions that corresponds to the future times, in degrees.
- **future_dec** (*ndarray*) – The set of declinations that corresponds to the future times, in degrees.

opihiexarata.ephemeris.solution module

The ephemeris solution class.

class opihiexarata.ephemeris.solution.**EphemeriticSolution**(*orbitals: OrbitalSolution, solver_engine: EphemerisEngine, vehicle_args: dict = {}*)

Bases: *ExarataSolution*

This obtains the ephemeris of an asteroid using an ephemeris engine provided the Keplerian orbital elements of the asteroid as determined by orbital solutions.

orbitals

The orbital solution from which the orbital elements will be taken from to determine the orbit of the target.

Type
OrbitalSolution

ra_velocity

The right ascension angular velocity of the target, in degrees per second.

Type
float

dec_velocity

The declination angular velocity of the target, in degrees per second.

Type
float

ra_acceleration

The right ascension angular acceleration of the target, in degrees per second squared.

Type
float

dec_acceleration

The declination angular acceleration of the target, in degrees per second squared.

Type
float

__init__ (*orbitals:* OrbitalSolution, *solver_engine:* EphemerisEngine, *vehicle_args: dict = {}*) → None

Instantiating the solution class.

Parameters

- **orbitals** (OrbitalSolution) – The orbital solution from which the orbital elements will be taken from to determine the orbit of the target.
- **solver_engine** (EphemerisEngine) – The ephemeris solver engine class. This is what will act as the "behind the scenes" and solve the orbit, using this middleware to translate it into something that is easier to read.
- **vehicle_args** (*dictionary*) – If the vehicle function for the provided solver engine needs extra parameters not otherwise provided by the standard input, they are given here.

Return type
None

forward_ephemeris (*future_time: ndarray*[87]) → tuple[numpy.ndarray[88], numpy.ndarray[89]]

A wrapper call around the engine's ephemeris function. This allows the computation of future positions at a future time using the ephemeris derived from the orbital elements.

Parameters
future_time (*array-like*) – The set of future times which to derive new RA and DEC coordinates. The time must be in Julian days.

Returns

- **future_ra** (*ndarray*) – The set of right ascensions that corresponds to the future times, in degrees.
- **future_dec** (*ndarray*) – The set of declinations that corresponds to the future times, in degrees.

opihiexarata.ephemeris.solution.**_vehicle_jpl_horizons_web_api** (*orbitals:* OrbitalSolution)

This uses the JPL Horizons web URL API service to derive the ephemeris.

[87] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[88] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[89] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

Parameters
orbitals (OrbitalSolution) – The orbital solution to use to get the orbital elements to send off to the JPL Horizons API.

Returns
ephemeris_results – The results of the ephemeris engine which then gets integrated into the solution.

Return type
dictionary

Module contents

This is the module for all of the ephemeris calculations, provided orbital elements.

opihiexarata.gui package

Subpackages

opihiexarata.gui.qtui package

Submodules

opihiexarata.gui.qtui.qtui_automatic module

class opihiexarata.gui.qtui.qtui_automatic.**Ui_AutomaticWindow**
Bases: object

retranslateUi(*AutomaticWindow*)

setupUi(*AutomaticWindow*)

opihiexarata.gui.qtui.qtui_manual module

class opihiexarata.gui.qtui.qtui_manual.**Ui_ManualWindow**
Bases: object

retranslateUi(*ManualWindow*)

setupUi(*ManualWindow*)

opihiexarata.gui.qtui.qtui_selector module

class opihiexarata.gui.qtui.qtui_selector.**Ui_SelectorWindow**

Bases: object

retranslateUi(*SelectorWindow*)

setupUi(*SelectorWindow*)

Module contents

All of the QtUI frameworks.

Submodules

opihiexarata.gui.automatic module

This is where the automatic mode window is implemented.

class opihiexarata.gui.automatic.**OpihiAutomaticWindow**

Bases: QMainWindow

__connect_push_button_change_directory() → None

The connection for the button to change the automatic fetch directory.

Parameters
None –

Return type
None

__connect_push_button_start() → None

This enables the automatic active mode by changing the flag and starting the process.

Parameters
None –

Return type
None

__connect_push_button_stop() → None

This disables the automatic active mode by changing the flag. The loop itself should detect that the flag has changed. It finishes the current process but does not fetch any more.

Parameters
None –

Return type
None

__connect_push_button_trigger() → None

This does one process, fetching a single image and processing it as normal. However, it does not trigger the automatic mode loop as it is built for a single image only.

Parameters
None –

Return type
None

__init__() → None

The automatic GUI window for OpihiExarata. This interacts with the user with regards to the automatic solving mode of Opihi.

Parameters
None –

Return type
None

__init_gui_connections() → None

Creating the function connections for the GUI interface.

Parameters
None –

Return type
None

__refresh_dynamic_label_text() → None

Refreshes the GUI window's dynamic text.

Parameters
None –

Return type
None

_automatic_triggering_check_stops() → bool

This function checks for the stops to stop the automatic triggering of the next image.

Parameters
None –

Returns
stop_check – This is the flag which signifies if the triggering should stop or not. If True, the triggering should stop.

Return type
bool

_automatic_triggering_infinite_loop() → None

This is where the actual infinite loop is done.

All of the stops are checked before a new trigger is executed.

Parameters
None –

Return type
None

automatic_triggering() → None

This function executes the continuous running automatic mode loop.

All of the stops are checked before a new trigger is executed.

Parameters
None –

Return type
None

fetch_new_filename() → str

This function fetches a new fits filename based on the most recent filename within the automatic fetching directory.

Parameters
None –

Returns
fetched_filename – The filename that was fetched. It is the most recent file added to the automatic fetching directory.

Return type
string

refresh_window() → None

Refreshes the GUI window with new information where available.

Parameters
None –

Return type
None

reset_window() → None

This function resets the window to the default values or parameters

Parameters
None –

Return type
None

solve_astrometry_photometry_single_image(*filename: str*) → *OpihiSolution*

This solves for the astrometric and photometric solutions of a provided file. The engines are provided based on the dropdown menus.

Note this calculation does not affect the *opihi_solution* instance of the class. That is a job for a different function.

Parameters
filename (*string*) – The filename of the fits file to be solved.

Returns
opihi_solution – The solution with the astrometry and photometry engines solved.

Return type
OpihiSolution

staticMetaObject = PySide6.QtCore.QMetaObject("OpihiAutomaticWindow" inherits "QMainWindow":)

trigger_next_image_solve() → None
This function triggers the next iteration of the automatic solving loop.

Parameters
None –

Return type
None

opihiexarata.gui.automatic.**start_automatic_window**() → None
This is the function to create the automatic window for usage.

Parameters
None –

Return type
None

opihiexarata.gui.functions module

Where helpful functions which otherwise do not belong in the library, for the GUIs, exist.

opihiexarata.gui.functions.**pick_engine_class_from_name**(*engine_name: str*, *engine_type: ~opihiexarata.library.engine.ExarataEngine = <class 'opihiexarata.library.engine.ExarataEngine'>*) → *ExarataEngine*
This returns a specific engine class provided its user friendly name. This is a convince function for both development and implementation.

If an engine name provided is not present, this raises.

Parameters

- **engine_name** (*str*) – The engine name, the user friendly version. It is case insensitive.

- **engine_type** (ExarataEngine) – The engine subtype, if not provided, then it searches through all available implemented engines.

Returns
engine_class – The more specific engine class based on the engine name.

Return type
ExarataEngine

opihiexarata.gui.manual module

The manual GUI window.

class opihiexarata.gui.manual.**OpihiManualWindow**

Bases: QMainWindow

__connect_push_button_astrometry_custom_solve() → None

"The button which uses an astrometric solution to solve for a custom pixel location or RA DEC location depending on entry.

This prioritizes solving RA DEC from pixel location.

Parameters
None –

Return type
None

__connect_push_button_astrometry_solve_astrometry() → None

The button to instruct on the solving of the astrometric solution.

Parameters
None –

Return type
None

__connect_push_button_new_image_automatic()

The automatic method relying on earliest fits file available in the expected directory. This function is a connected function action to a button in the GUI.

Parameters
None –

Return type
None

__connect_push_button_new_image_manual()

The manual method relying on earliest fits file available in the expected directory. This function is a connected function action to a button in the GUI.

Parameters
None –

Return type
None

__connect_push_button_new_target() → None

The function serving to set the software to be on a new target. A name is prompted from the user.

Parameters
None –

Return type
None

__connect_push_button_orbit_solve_ephemeris() → None

A routine to use the current observation and historical observations to derive the orbit solution.

Parameters
None –

Return type
None

__connect_push_button_orbit_solve_orbit() → None

A routine to use the current observation and historical observations to derive the orbit solution.

Parameters
None –

Return type
None

__connect_push_button_photometry_solve_photometry() → None

A routine to use the current observation and historical observations to derive the propagation solution.

Parameters
None –

Return type
None

__connect_push_button_propagate_custom_solve() → None

Solving for the location of the target through propagation based on the time and date provided by the user.

Parameters
None –

Return type
None

__connect_push_button_propagate_solve_propagation() → None

A routine to use the current observation and historical observations to derive the propagation solution.

Parameters
None –

Return type
None

__connect_push_button_refresh_window() → None

The function serving to refresh the window and redrawing the plot.

Parameters
None –

Return type
None

__get_mpc_record_filename() → str

This is a function which derives the MPC record filename from naming conventions and the current fits file name.

Parameters
None –

Returns
mpc_record_filename – The filename of the MPC record for this object/image.

Return type
str

__init__() → None

The manual GUI window for OpihiExarata. This interacts directly with the total solution object of Opihi.

Parameters
None –

Return type
None

__init_gui_connections() → None

Assign the action bindings for the buttons which get new file(names).

Parameters
None –

Return type
None

__init_opihi_image() → None

Create the image area which will display what Opihi took from the sky. This takes advantage of a reserved image vertical layout in the design of the window.

Parameters
None –

Return type
None

__init_preprocess_solution()

Initialize the preprocessing solution. The preprocessing files should be specified in the configuration file.

Parameters
None –

Return type
None

__refresh_dynamic_label_text_astrometry() → None

Refresh all of the dynamic label text for astrometry. This fills out the information based on the current solutions available and solved.

An astrometric solution must exist.

Parameters
None –

Return type
None

__refresh_dynamic_label_text_ephemeris() → None

Refresh all of the dynamic label text for ephemerides. This fills out the information based on the current solutions available and solved.

An ephemeritic solution must exist.

Parameters
None –

Return type
None

__refresh_dynamic_label_text_orbit() → None

Refresh all of the dynamic label text for orbit. This fills out the information based on the current solutions available and solved.

An orbital solution must exist.

Parameters
None –

Return type
None

__refresh_dynamic_label_text_photometry() → None

Refresh all of the dynamic label text for photometry. This fills out the information based on the current solutions available and solved.

A photometric solution must exist.

Parameters
None –

Return type
None

__refresh_dynamic_label_text_propagate() → None

Refresh all of the dynamic label text for propagate. This fills out the information based on the current solutions available and solved.

A propagative solution must exist.

Parameters
None –

Return type
None

_load_fits_file(*fits_filename: str*) → None

Load a fits file into the GUI to derive all of the solutions needed.

This loads the fits file into an OpihiSolution class for which this GUI interacts with and derives the information from.

Parameters
fits_filename (*str*) – The fits filename which will be loaded.

Return type
None

_parse_custom_orbital_elements() → dict

This function takes the textual form of the orbital elements as entered and tries to parse it into a set of orbital elements and errors.

Parameters
None –

Returns
orbital_elements – A dictionary of the orbital elements and their errors, if they exist.

Return type
dictionary

_preprocess_fits_file(*raw_filename: str*, *process_filename: Optional[str] = None*) → str

Using the provided filename, preprocess the fits file listed.

This is used primarily during the loading of a new fits file.

Parameters

- **raw_filename** (*string*) – The filename of the raw fits file to be preprocessed.
- **process_filename** (*string, default = None*) – The filename where the processed fits file will be saved to. If None, it defaults to a sensible naming convention to distinguish the two.

Returns
process_filename – The filename of the preprocessed file is returned, this is just in case the value was derived from the naming convention. This is an absolute path.

Return type
string

clear_dynamic_label_text() → None

Clear all of the dynamic label text and other related fields, this is traditionally done just before a new image is going to be introduced.

This resets the text back to their defaults as per the GUI builder.

The selections on the engines do not need to be reset as they are just a selection criteria and do not actually provide any information.

Parameters
None –

Return type
None

redraw_opihi_image() → None

Redraw the Opihi image given that new results may have been added because some solutions were completed. This modifies the GUI in-place.

Parameters
None –

Return type
None

refresh_dynamic_label_text() → None

Refresh all of the dynamic label text, this fills out the information based on the current solutions available and solved.

Parameters
None –

Return type
None

save_results() → None

Save all of the results of the solutions to date. This is especially done upon selecting a new image, the previous image results are saved.

If there is no solution class, then there is no results to save either and this function does nothing.

Parameters
None –

Return type
None

staticMetaObject = PySide6.QtCore.QMetaObject("OpihiManualWindow" inherits "QMainWindow":)

opihiexarata.gui.manual.**start_manual_window**() → None

This is the function to create the manual window for usage.

Parameters
None –

Return type
None

opihiexarata.gui.name module

This window is for allowing the user to fill out the name of the object which they are observing, used when doing new targets.

This is just a simple input dialog.

class opihiexarata.gui.name.**TargetNameWindow**

Bases: QWidget

__init__() → None

Setting up the window for the user prompt.

Parameters
None –

Return type
None

static _sanitize_input(*raw_input: Optional[str] = None*) → str

Sanitization of the raw input to the proper input.

Parameters
raw_input (*string*) – The raw input to be sanitized.

Returns
sanitized_input – The sanitized input.

Return type
string

staticMetaObject = PySide6.QtCore.QMetaObject("TargetNameWindow" inherits "QWidget":)

opihiexarata.gui.name.**ask_user_target_name_window**(*default: Optional[str] = None*) → str

Use the target name window to prompt the user for the name of the object that they are studying.

Parameters
default (*string*) – The default name to provide should the user interact with the dialog box to submit a name. (That is, they cancel or close it.)

Returns
target_name – The target name as specified (or the default if something went amiss.)

Return type
string

opihiexarata.gui.name.**main**() → None

opihiexarata.gui.selector module

class opihiexarata.gui.selector.**TargetSelectorWindow**(*current_fits_filename: str*, *reference_fits_filename: Optional[str] = None*)

Bases: QWidget

__connect_check_box_autoscale_1_99() → None
This check box allows the user to force the autoscaling of images when the subtraction method changes.

Parameters
None –

Return type
None

__connect_line_edit_dynamic_scale_high() → None
A function to operate on the change of the text of the high scale.

Parameters
None –

Return type
None

__connect_line_edit_dynamic_scale_low() → None
A function to operate on the change of the text of the low scale.

Parameters
None –

Return type
None

__connect_matplotlib_mouse_press_event(*event: MouseEvent*[90]) → None
A function to describe what would happen when a mouse press is done on the Matplotlib image.

This function defaults to the toolbar functionality when the toolbar is considered active.

Parameters
event (*MouseEvent*) – The event of the click itself.

Return type
None

__connect_matplotlib_mouse_release_event(*event: MouseEvent*[91]) → None

A function to describe what would happen when a mouse press is released done on the Matplotlib image.

Parameters
event (*MouseEvent*) – The event of the click itself.

Return type
None

__connect_push_button_change_current_filename() → None

This function provides a popup dialog to prompt the user to change the current fits filename.

Parameters
None –

Return type
None

__connect_push_button_change_reference_filename() → None

This function provides a popup dialog to prompt the user to change the reference fits filename.

Parameters
None –

Return type
None

__connect_push_button_mode_non_sidereal() → None

This function sets the subtraction method to non-sidereal, for comparing the current image from the reference image.

This method assumes the approximation that the target itself did not move at all compared to both images, but the stars do as they are moving siderally.

Parameters
None –

Return type
None

__connect_push_button_mode_none() → None

This function sets the subtraction method to None, for comparing the current image from the reference image.

Both None the type and the string is valid as no subtraction. The type just means that it has not been formally specified using the GUI.

This method has no subtraction and thus no comparison to the reference image.

Parameters
None –

Return type
None

__connect_push_button_mode_reference() → None

This function sets the subtraction method to Reference, plotting the reference image instead of the current image.

This method has no subtraction and thus no comparison to the current image.

Parameters
None –

Return type
None

__connect_push_button_mode_sidereal() → None

This function sets the subtraction method to sidereal, for comparing the current image from the reference image.

This method assumes the approximation that both the current and reference images are pointing to the same point in the sky.

Parameters
None –

Return type
None

__connect_push_button_scale_1_99() → None

A function set the scale automatically to 1-99 within the region currently displayed on the screen.

Parameters
None –

Return type
None

__connect_push_button_submit_target() → None

This button submits the current location of the target and closes the window. (The target information is saved within the class instance.)

If the text within the line edits differ than what the box selection has selected, then this prioritizes the values as manually defined. Although this should be rare as any time a box is drawn, the values and text boxes should be updated.

If no entry is properly convertible, we default to center of the image.

Parameters
None –

Return type
None

__init__(*current_fits_filename: str*, *reference_fits_filename: Optional[str] = None*) → None

Create the target selector window. Though often used for asteroids, there is no reason why is should specific to them; so we use a general name.

Parameters

- **current_fits_filename** (*string*) – The current fits filename which will be used to determine where the location of the target is.
- **reference_fits_filename** (*string, default = None*) – The reference fits filename which will be used to compare against the current fits filename to determine where the location of the target is. If None, then no image will be loaded until manually specified.

Return type
None

__init_gui_connections()

A initiation set of functions that attach to the buttons on the GUI.

Parameters
None –

Return type
None

__init_opihi_image() → None

Create the image area which will display what Opihi took from the sky. This takes advantage of a reserved image vertical layout in the design of the window.

Parameters
None –

Return type
None

_recompute_colorbar_autoscale(*lower_percentile: float = 1*, *higher_percentile: float = 99*) → None

This is a function to recompute the autoscaling of the colorbar.

This function needs to be split from the connection buttons otherwise an infinite loop occurs because of their inherent and expected calls to refresh the window.

Parameters

- **lower_percentile** (*float, default = 1*) – The lower percentile value which will be defined at the zero point for the colorbar.
- **higher_percentile** (*float, default = 99*) – The higher (upper) percentile value which will be defined as the one point for the colorbar.

Return type
None

_recompute_subtraction_arrays() → None

This computes the subtracted arrays for both none, sidereal, and non-sidereal subtractions. This is done mostly for speed considerations as the values can be computed and stored during image loading.

Parameters
None –

Return type
None

_refresh_image() → None

Redraw and refresh the image, this is mostly used to allow for the program to update where the user selected.

Parameters
None –

Return type
None

_refresh_text() → None

This function just refreshes the GUI text based on the current actual values.

Parameters
None –

Return type
None

close_window() → None

Closes the window. Generally called when it is all done.

Parameters
None –

Return type
None

find_target_location(*x0: float*, *x1: float*, *y0: float*, *y1: float*) → tuple[float, float]

Find the location of a target by using a guessed location. The bounds of the search is specified by the rectangle.

Parameters

- **x0** (*float*) – The lower x axis bound of the search area. These values are cast into integers upon indexing the search area.
- **x1** (*float*) – The upper x axis bound of the search area. These values are cast into integers upon indexing the search area.
- **y0** (*float*) – The lower y axis bound of the search area. These values are cast into integers upon indexing the search area.
- **y1** (*float*) – The upper y axis bound of the search area. These values are cast into integers upon indexing the search area.

Returns

- **target_x** (*float*) – The location of the target, based on the guess.
- **target_y** (*float*) – The location of the target, based on the guess.

refresh_window() → None

Refresh the text content of the window given new information. This refreshes both the dynamic label text and redraws the image.

Parameters
None –

Return type
None

staticMetaObject = PySide6.QtCore.QMetaObject("TargetSelectorWindow" inherits "QWidget":)

opihiexarata.gui.selector.**ask_user_target_selector_window**(*current_fits_filename*, *reference_fits_filename: Optional[str] = None*) → tuple[float, float]

Use the target selector window to have the user provide the information needed to determine the location of the target.

Parameters

- **current_fits_filename** (*string*) – The current fits filename which will be used to determine where the location of the target is.
- **reference_fits_filename** (*string, default = None*) – The reference fits filename which will be used to compare against the current fits filename to determine where the location of the target is. If None, then no image will be loaded until manually specified.

Returns

- **target_x** (*float*) – The location of the target in the x axis direction.
- **target_y** (*float*) – The location of the target in the y axis direction.

opihiexarata.gui.selector.**main**()

[90] https://matplotlib.org/stable/api/backend_bases_api.html#matplotlib.backend_bases.MouseEvent
[91] https://matplotlib.org/stable/api/backend_bases_api.html#matplotlib.backend_bases.MouseEvent

Module contents

Parts of the Exarata GUI.

opihiexarata.library package

Submodules

opihiexarata.library.config module

Controls the inputting of configuration files. This also serves to bring all of the configuration parameters into a more accessable space which other parts of Exarata can use.

Note these configuration constant parameters are all accessed using capital letters regardless of the configuration file's labels. Moreover, there are constant parameters which are stored here which are not otherwise changeable by the configuration file.

opihiexarata.library.config.**generate_configuration_file_copy**(*filename: str*, *overwrite=False*) → None

This generates a copy of the default configuration file to the given location.

Parameters

- **filename** (*string*) – The pathname or filename where the configuration file should be put to. If it does not have the proper yaml extension, it will be added.
- **overwrite** (*bool, default = False*) – If the file already exists, overwrite it. If False, it would raise an error instead.

Return type
None

opihiexarata.library.config.**generate_secrets_file_copy**(*filename: str*, *overwrite=False*) → None

This generates a copy of the secrets configuration file to the given location.

Parameters

- **filename** (*string*) – The pathname or filename where the configuration file should be put to. If it does not have the proper yaml extension, it will be added.
- **overwrite** (*bool, default = False*) – If the file already exists, overwrite it. If False, it would raise an error instead.

Return type
None

opihiexarata.library.config.**load_configuration_file**(*filename: str*) → dict

Loads a configuration file and outputs a dictionary of parameters.

Note configuration files should be flat, there should be no nested configuration parameters.

Parameters
filename (*string*) – The filename of the configuration file, with the extension. Will raise if the filename is not the correct extension, just as a quick check.

Returns
configuration_dict – The dictionary which contains all of the configuration parameters within it.

Return type
dictionary

opihiexarata.library.config.**load_then_apply_configuration**(*filename: str*) → None

Loads a configuration file, then applies it to the entire Exarata system.

Loads a configuration file and overwrites any overlapping configurations. It writes the configuration to the configuration module for usage throughout the entire program.

Note configuration files should be flat, there should be no nested configuration parameters.

Parameters
filename (*string*) – The filename of the configuration file, with the extension. Will raise if the filename is not the correct extension, just as a quick check.

Return type
None

opihiexarata.library.conversion module

For miscellaneous conversions.

opihiexarata.library.conversion.**current_utc_to_julian_day**() → float

Return the current UTC time when this function is called as a Julian day time.

Parameters
None –

Returns
current_jd – The current time in Julian date format.

Return type
float

opihiexarata.library.conversion.**decimal_day_to_julian_day**(*year: int*, *month: int*, *day: float*)

A function to convert decimal day time formats to the Julian day.

Parameters

- **year** (*int*) – The year of the time stamp to be converted.
- **month** (*int*) – The month of the time stamp to be converted.
- **day** (*float*) – The day of the time stamp to be converted, may be decimal, the values are just passed down.

Returns
julian_day – The Julian day of the date provided.

Return type
float

opihiexarata.library.conversion.**degrees_to_sexagesimal_ra_dec**(*ra_deg: float*, *dec_deg: float*, *precision: int = 2*) → tuple[str, str]

Convert RA and DEC degree measurements to the more familiar HMSDMS sexagesimal format.

Parameters

- **ra_deg** (*float*) – The right ascension in degrees.
- **dec_deg** (*float*) – The declination in degrees.
- **precision** (*int*) – The precision of the conversion's seconds term, i.e. how many numbers are used.

Returns

- **ra_sex** (*str*) – The right ascension in hour:minute:second sexagesimal.
- **dec_sex** (*str*) – The declination in degree:minute:second sexagesimal.

opihiexarata.library.conversion.**full_date_to_julian_day**(*year: int*, *month: int*, *day: int*, *hour: int*, *minute: int*, *second: float*) → float

A function to convert the a whole date format into the Julian day time.

Parameters

- **year** (*int*) – The year of the time stamp to be converted.
- **month** (*int*) – The month of the time stamp to be converted.
- **day** (*int*) – The day of the time stamp to be converted.
- **hour** (*int*) – The hour of the time stamp to be converted.
- **minute** (*int*) – The minute of the time stamp to be converted.
- **second** (*float*) – The second of the time stamp to be converted.

Returns
julian_day – The time input converted into the Julian day.

Return type
float

opihiexarata.library.conversion.**julian_day_to_decimal_day**(*jd: float*) → tuple

A function to convert the Julian day time to the decimal day time.

Parameters
jd (*float*) – The Julian day time that is going to be converted.

Returns

- **year** (*int*) – The year of the Julian day time.
- **month** (*int*) – The month of the Julian day time.
- **day** (*float*) – The day of the the Julian day time, the hours, minute, and seconds are all contained as a decimal.

opihiexarata.library.conversion.**julian_day_to_full_date**(*jd: float*) → tuple

A function to convert the Julian day to a full date time.

Parameters
jd (*float*) – The Julian day time that is going to be converted.

Returns

- **year** (*int*) – The year of the Julian day provided.
- **month** (*int*) – The month of the Julian day provided.
- **day** (*int*) – The day of the Julian day provided.
- **hour** (*int*) – The hour of the Julian day provided.
- **minute** (*int*) – The minute of the Julian day provided.
- **second** (*float*) – The second of the Julian day provided.

opihiexarata.library.conversion.**julian_day_to_modified_julian_day**(*jd: float*) → float

A function to convert Julian days to modified Julian days.

Parameters
jd (*float*) – The Julian day to be converted to a modified Julian day.

Returns
mjd – The modified Julian day value after conversion.

Return type
float

opihiexarata.library.conversion.**julian_day_to_unix_time**(*jd: float*) → float

A function to convert between Julian days to UNIX time.

Parameters
jd (*float*) – The Julian day to be converted.

Returns
unix_time – The time converted to UNIX time.

Return type
float

opihiexarata.library.conversion.**modified_julian_day_to_julian_day**(*mjd: float*) → float

A function to convert modified Julian days to Julian days.

Parameters
mjd (*float*) – The modified Julian day to be converted to a Julian day.

Returns
jd – The Julian day value after conversion.

Return type
float

opihiexarata.library.conversion.**sexagesimal_ra_dec_to_degrees**(*ra_sex: str*, *dec_sex: str*) → tuple[float, float]

Convert RA and DEC measurements from the more familiar HMSDMS sexagesimal format to degrees.

Parameters

- **ra_sex** (*str*) – The right ascension in hour:minute:second sexagesimal.
- **dec_sex** (*str*) – The declination in degree:minute:second sexagesimal.

Returns

- **ra_deg** (*float*) – The right ascension in degrees.
- **dec_deg** (*float*) – The declination in degrees.

opihiexarata.library.conversion.**string_month_to_number**(*month_str: str*) → int

A function to convert from the name of a month to the month number. This is just because it is easy to have here and to add a package import for something like this is silly.

Parameters
month_str (*string*) – The month name.

Returns
month_int – The month number integer.

Return type
int

opihiexarata.library.conversion.**unix_time_to_julian_day**(*unix_time: float*) → float

A function to convert between julian days to Unix time.

Parameters
unix_time (*float*) – The UNIX time to be converted to a Julian day.

Returns
jd – The Julian day value as converted.

Return type
float

opihiexarata.library.engine module

Where the base classes of the solvers and engines lie.

class opihiexarata.library.engine.**AstrometryEngine**

Bases: *ExarataEngine*

The base class where the Astrometry engines are derived from. Should not be used other than for type hinting and subclassing.

class opihiexarata.library.engine.**EphemerisEngine**

Bases: *ExarataEngine*

The base class for all of the Ephemeris determination engines. Should not be used other than for type hinting and subclassing.

class opihiexarata.library.engine.**ExarataEngine**

Bases: object

The base class for all of the engines which power the interfaces for the OpihiExarata system. This should not used for anything other than type hinting and subclassing.

class opihiexarata.library.engine.**ExarataSolution**

Bases: object

The base class for all of the solution classes which use engines to solve particular problems.

class opihiexarata.library.engine.**OrbitEngine**

Bases: *ExarataEngine*

The base class for all of the Orbit determination engines. Should not be used other than for type hinting and subclassing.

class opihiexarata.library.engine.**PhotometryEngine**

Bases: *ExarataEngine*

The base class where the Photometry engines are derived from. Should not be used other than for type hinting and subclassing.

class opihiexarata.library.engine.**PropagationEngine**

Bases: *ExarataEngine*

The base class where the Propagation engines are derived from. Should not be used other than for type hinting and subclassing.

opihiexarata.library.error module

Error, warning, and logging functionality pertinent to the function of Exarata.

exception opihiexarata.library.error.**CommandLineError**(*message: Optional[str] = None*)

Bases: *ExarataException*

An error to be used where the parameters or arguments entered in the command line were not correct.

exception opihiexarata.library.error.**ConfigurationError**(*message: Optional[str] = None*)

Bases: *ExarataException*

An error to be used where the expectation of how configuration files and configuration parameters are structures are violated.

exception opihiexarata.library.error.**DevelopmentError**(*message: Optional[str] = None*)

Bases: *ExarataBaseException*

This is an error where the development of OpihiExarata is not correct and something is not coded based on the expectations of the software itself. This is not the fault of the user.

exception opihiexarata.library.error.**DirectoryError**(*message: Optional[str] = None*)

Bases: *ExarataException*

An error to be used when there are issues specifically with directories and not just files.

exception opihiexarata.library.error.**EngineError**(*message: Optional[str] = None*)

Bases: *ExarataException*

This error is for when the astrometric, photometric, or asteroid-metric solver engines provided are not valid or expected. This error can also be used when an engine fails to solve or otherwise does not work as intended.

exception opihiexarata.library.error.**ExarataBaseException**(*message: Optional[str] = None*)

Bases: BaseException

The base exception class. This is for exceptions that should never be caught and should bring everything to a halt.

__init__(*message: Optional[str] = None*) → None

The initialization of a base exception for OpihiExarata.

Parameters

message (*string*) – The message of the error message.

Return type

None

exception opihiexarata.library.error.**ExarataException**(*message: Optional[str] = None*)

Bases: Exception

The main inheriting class which all exceptions use as their base. This is done for ease of error handling and is something that can and should be managed.

__init__(*message: Optional[str] = None*) → None

The initialization of a normal exception.

Parameters
message (*string*) – The message of the error message.

Return type
None

exception opihiexarata.library.error.**ExarataWarning**

Bases: UserWarning

The base warning class which all of the other OpihiExarata warnings are derived from.

exception opihiexarata.library.error.**FileError**(*message: Optional[str] = None*)

Bases: *ExarataException*

An error to be used when obtaining data files or configuration files and something fails.

exception opihiexarata.library.error.**InputError**(*message: Optional[str] = None*)

Bases: *ExarataException*

An error to be used when the inputs to a function are not valid and do not match the expectations of that function.

exception opihiexarata.library.error.**InstallError**(*message: Optional[str] = None*)

Bases: *ExarataException*

An error to be used when informing the user or the program that the installation was not done properly and lack some of the features and assumptions which are a consequence of it.

exception opihiexarata.library.error.**IntentionalError**(*message: Optional[str] = None*)

Bases: *ExarataException*

An error to be used where error catching is helpful. This error generally should always be caught by the code in context.

exception opihiexarata.library.error.**LogicFlowError**(*message: Optional[str] = None*)

Bases: *ExarataBaseException*

This is an error to ensure that the logic does not flow to a point to a place where it is not supposed to. This is helpful in making sure changes to the code do not screw up the logical flow of the program.

exception opihiexarata.library.error.**PracticalityError**(*message: Optional[str] = None*)

Bases: *ExarataBaseException*

This is an error to be used when what is trying to be done does not seem reasonable. Usually warnings are the better vehicle for this but this error is used when the assumptions for reasonability guided development and what the user is trying to do is not currently supported by the software.

exception opihiexarata.library.error.**ReadOnlyError**(*message: Optional[str] = None*)

Bases: *ExarataException*

An error where variables or files are assumed to be read only, this enforces that notion.

exception opihiexarata.library.error.**SequentialOrderError**(*message: Optional[str] = None*)

Bases: *ExarataException*

An error used when something is happening out of the expected required order. This order being in place for specific publically communicated reasons.

exception opihiexarata.library.error.**UndiscoveredError**(*message: Optional[str] = None*)

Bases: *ExarataBaseException*

This is an error used in cases where the source of the error has not been determined and so a more helpful error message or mitigation strategy cannot be devised.

exception opihiexarata.library.error.**WebRequestError**(*message: Optional[str] = None*)

Bases: *ExarataException*

An error to be used when a web request to some API fails, either because of something from their end, or our end.

opihiexarata.library.error.**warn**(*warn_class: type[opihiexarata.library.error.ExarataWarning] = <class 'opihiexarata.library.error.ExarataWarning'>, message: str = '', stacklevel: int = 2*)

The common method to use to warn for any OpihiExarata based warnings.

This is used because it has better context manager wrappers.

Parameters

- **warn_class** (*type, default = ExarataWarning*) – The warning class, it must be a subtype of a user warning.
- **message** (*string, default = ""*) – The warning message.
- **stacklevel** (*integer, default = 2*) – The location in the stack that the warning call will highlight.

Return type
None

opihiexarata.library.fits module

Fits file based operations. These are kind of like convince functions.

opihiexarata.library.fits.**read_fits_header**(*filename: str*, *extension: Union[int, str] = 0*) → Header

This reads the header of fits files only. This should be used only if there is no data.

Really, this is just a wrapper around Astropy, but it is made for consistency and to avoid the usage of the convince functions.

Parameters

- **filename** (*string*) – The filename that the fits image file is at.
- **extension** (*int or string, default = 0*) – The fits extension that is desired to be opened.

Returns
header – The header of the fits file.

Return type
Astropy Header

opihiexarata.library.fits.**read_fits_image_file**(*filename: str*, *extension: Union[int, str] = 0*) → tuple[astropy.io.fits.header.Header, numpy.ndarray[92]]

This reads fits files, assuming that the fits file is an image. It is a wrapper function around the astropy functions.

Parameters

- **filename** (*string*) – The filename that the fits image file is at.
- **extension** (*int or string, default = 0*) – The fits extension that is desired to be opened.

Returns

- **header** (*Astropy Header*) – The header of the fits file.
- **data** (*array*) – The data image of the fits file.

opihiexarata.library.fits.**read_fits_table_file**(*filename: str*, *extension: Union[int, str] = 0*) → tuple[astropy.io.fits.header.Header, astropy.table.table.Table]

[92] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

This reads fits files, assuming that the fits file is a binary table. It is a wrapper function around the astropy functions.

Parameters

- **filename** (*string*) – The filename that the fits image file is at.
- **extension** (*int or string, default = 0*) – The fits extension that is desired to be opened.

Returns

- **header** (*Astropy Header*) – The header of the fits file.
- **table** (*Astropy Table*) – The data table of the fits file.

opihiexarata.library.fits.**update_fits_header**(*header: Header*, *entries: dict*, *comments: dict = {}*) → Header

This appends entries from a dictionary to an Astropy header.

This function is preferred to adding using standard methods as it performs checks to make sure it only uses header keys reserved for OpihiExarata. This function raises an error upon attempting to add an entry which does not conform to fits standards and is not keyed with the "OX######" template.

Parameters

- **header** (*Astropy Header*) – The header which the entries will be added to.
- **entries** (*dictionary*) – The new entries to the header.
- **comments** (*dictionary, default = {}*) – If comments are to be added to data entries, then they may be provided as a dictionary here with keys exactly the same as the data entries. This is not for comment cards.

opihiexarata.library.fits.**write_fits_image_file**(*filename: str*, *header: Header*, *data: ndarray*[93], *overwrite: bool = False*) → None

This writes fits image files to disk. Acting as a wrapper around the fits functionality of astropy.

Parameters

- **filename** (*string*) – The filename that the fits image file will be written to.
- **header** (*Astropy Header*) – The header of the fits file.
- **data** (*array-like*) – The data image of the fits file.
- **overwrite** (*boolean, default = False*) – Decides if to overwrite the file if it already exists.

Return type
None

[93] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

opihiexarata.library.fits.**write_fits_table_file**(*filename: str*, *header: Header*, *data: Table*, *overwrite: bool = False*) → None

This writes fits table files to disk. Acting as a wrapper around the fits functionality of astropy.

Parameters

- **filename** (*string*) – The filename that the fits image file will be written to.
- **header** (*Astropy Header*) – The header of the fits file.
- **data** (*Astropy Table*) – The data table of the table file.
- **overwrite** (*boolean, default = False*) – Decides if to overwrite the file if it already exists.

Return type
None

opihiexarata.library.hint module

These are redefinitions and wrapping variables for type hints. Its purpose is for just uniform hinting types.

This should only be used for types which are otherwise not native and would require an import, including the typing module. The whole point of this is to be a central collection of types for the purpose of type hinting.

This module should never be used for anything other than hinting. Use proper imports to access these classes. Otherwise, you will likely get circular imports and other nasty things.

opihiexarata.library.http module

Functions and methods which allow for ease of interacting with web based resources. Included here are functions which download files, query web resources and other things. This interacts mostly with HTTP based services.

opihiexarata.library.http.**api_request_sleep**(*seconds: Optional[float] = None*) → None

Sleep for the time, specified in the configuration file, for API requests. This function exists to ensure uniformity in application.

Parameters

- **seconds** (*float, default = None*) – The number of seconds that the program should sleep for. If not provided, then it defaults to the configuration value.
- **Results** –
- **-------** –
- **None** –

opihiexarata.library.http.**download_file_from_url**(*url: str*, *filename: str*, *overwrite: bool = False*) → None

Download a file from a URL to disk.

..warning:: The backend of this function relies on a function which may be depreciated in the future. This function may need to be rewritten.

Parameters

- **url** (*string*) – The url which the file will be downloaded from.
- **filename** (*string*) – The filename where the file will be saved.
- **overwrite** (*bool, default = False*) – If the file already exists, overwrite it. If False, it would raise an error instead.

opihiexarata.library.http.**get_http_status_code**(*url: str*) → int

This gets the http status code of a web resource.

Parameters
url (*string*) – The url which the http status code will try and obtain.

Returns
status_code – The status code.

Return type
int

opihiexarata.library.image module

Functions to help with image and array manipulations.

opihiexarata.library.image.**create_circular_mask**(*array: ndarray*[94], *center_x: int*, *center_y: int*, *radius: float*) → ndarray[95]

Creates an array which is a circular mask of some radius centered at a custom index value location. This process is a little intensive so using smaller subsets of arrays are preferred.

Method inspired by https://stackoverflow.com/a/44874588.

Parameters

- **array** (*array-like*) – The data array which the mask will base itself off of. The data in the array is not actually modified but it is required for the shape definition.
- **center_x** (*integer*) – The x-axis coordinate where the mask will be centered.
- **center_y** (*integer*) – The y-axis coordinate where the mask will be centered.
- **radius** (*float*) – The radius of the circle of the mask in pixels.

Returns
circular_mask – The mask; it is the same dimensions of the input data array. If True, the the mask should be applied.

Return type
array-like

opihiexarata.library.image.**save_array_as_png_grayscale**(*array: ndarray*[96], *filename: str*, *overwrite: bool = False*) → None

This converts an array to a grayscale PNG file.

The PNG specification requires that the data values be integer. Note that if you are saving an array as a PNG, then data may be lost during the conversion between float to integer.

Parameters

- **array** (*array-like*) – The array that will be saved as a png.
- **filename** (*string*) – The filename where the png will be saved. If the filename does not have the appropriate filename extension, it will be appended.
- **overwrite** (*boolean*) – If the file already exists, should it be overwritten?

opihiexarata.library.image.**scale_image_array**(*array: ndarray*[97], *minimum: float*, *maximum: float*, *lower_percent_cut: float = 0*, *upper_percent_cut: float = 0*) → ndarray[98]

This function scales the array to the provided minimum and maximum ranges after the percentile masks are taken.

Parameters

- **array** (*array-like*) – The array to be scaled.
- **minimum** (*float*) – The minimum value of the scaling axis. This will be equal to the minimum value of the scaled array after accounting for the percentile cuts.
- **maximum** (*float*) – The maximum value of the scaling axis. This will be equal to the maximum value of the scaled array after accounting for the percentile cuts.
- **lower_percent_cut** (*float*) – The percent of values that will be masked from the lower end. Must be between 0-100.
- **upper_percent_cut** (*float*) – The percent of values that will be masked from the upper end. Must be between 0-100.

Returns
scaled_array – The array, after the scaling.

Return type
array-like

[94] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[95] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[96] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[97] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[98] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

opihiexarata.library.image.**slice_array_boundary**(*array: ndarray*[99], *x_min: int*, *x_max: int*, *y_min: int*, *y_max: int*) → ndarray[100]

Slice an image array such that it stops at the boundaries and does not exceed past it. This function basically handels runtime slicing, but it returns a copy.

This function does not wrap around slices.

Parameters

- **array** (*array-like*) – The base array which the slice will access.
- **x_min** (*int*) – The lower index bound of the x-axis slice.
- **x_max** (*int*) – The upper index bound of the x-axis slice.
- **y_min** (*int*) – The lower index bound of the y-axis slice.
- **y_max** (*int*) – The upper index bound of the y-axis slice.

Returns

boundary_sliced_array – The array, sliced while adhering to the boundary of the slices.

Return type

array-like

opihiexarata.library.image.**translate_image_array**(*array: ndarray*[101], *shift_x: float = 0*, *shift_y: float = 0*, *pad_value: float = nan*) → ndarray[102]

This function translates an image or array in some direction. The image is treated as value padded so pixels beyond the scope of the image after translation are given by the value specified.

Parameters

- **array** (*array*) – The image array which is going to be translated.
- **shift_x** (*float, default = 0*) – The number of pixels the image will be shifted in the x direction.
- **shift_y** (*float, default = 0*) – The number of pixels the image will be shifted in the y direction.
- **pad_value** (*float, default = np.nan*) – The value to pad around the image.

Returns

shifted_image – The image array after shifting.

Return type

array

[99] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[100] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[101] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[102] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

opihiexarata.library.json module

A collection of functions to deal with JSON input and handling. For the most part, these functions are just wrappers around the built-in JSON handling.

opihiexarata.library.json.**dictionary_to_json**(*dictionary: dict*) → str

Converts a Python dictionary to a JSON string.

Parameters
dictionary (*dict*) – The Python dictionary which will be converted to a JSON string.

Returns
json_string – The JSON string.

Return type
str

opihiexarata.library.json.**json_to_dictionary**(*json_string: str*) → dict

Converts a JSON string to a dictionary.

Parameters
json_string (*str*) – The JSON string.

Returns
dictionary – The Python dictionary which will be converted to a JSON string.

Return type
dict

opihiexarata.library.mpcrecord module

opihiexarata.library.mpcrecord.**blank_minor_planet_table**() → Table

Creates a blank table which contains the columns which are recognized by the MPC standard 80-column record format.

Parameters
None –

Returns
blank_table – The table with only the column headings; no records are in the table.

Return type
Astropy Table

opihiexarata.library.mpcrecord.**minor_planet_record_to_table**(*records: list[str]*) → Table

This converts an 80 column record for minor planets to a table representing the same data.

The documentation for how the columns are assigned is provided by the Minor Planet Center: https://www.minorplanetcenter.net/iau/info/OpticalObs.html

Parameters
records (*list*) – The records in MPC format. Each entry of the list should be an 80 column string representing an observed record, a single line.

Returns
table – A table containing the same information that is in the MPC record format in an easier interface.

Return type
Astropy Table

opihiexarata.library.mpcrecord.**minor_planet_table_to_record**(*table: Table*) → list[str]

This converts an 80 column record for minor planets to a table representing the same data.

This function provides a minimal amount of verification that the input table is correct. If the provided entry of the table is too long, the text is striped of whitespace and then ususally cut.

The documentation for how the columns are assigned is provided by the Minor Planet Center: https://www.minorplanetcenter.net/iau/info/OpticalObs.html

Parameters
table (*Astropy Table*) – A table containing the same information that can be written as the standard 80-column record.

Returns
records – The records in MPC format. Each entry is 80 characters long and are in the standard format. The entries are derived from the provided table with information cut to fit into the format.

Return type
list

opihiexarata.library.path module

This module is just functions to deal with different common pathname manipulations. As Exarata is going to be cross platform, this is a nice abstraction.

opihiexarata.library.path.**get_directory**(*pathname: str*) → str

Get the directory from the pathname without the file or the extension.

Parameters
pathname (*string*) – The pathname which the directory will be extracted.

Returns
directory – The directory which belongs to the pathname.

Return type
string

opihiexarata.library.path.**get_file_extension**(*pathname: str*) → str

Get the file extension only from the pathname.

Parameters
pathname (*string*) – The pathname which the file extension will be extracted.

Returns
extension – The file extension only.

Return type
string

opihiexarata.library.path.**get_filename_with_extension**(*pathname: str*) → str

Get the filename from the pathname with the file extension.

Parameters
pathname (*string*) – The pathname which the filename will be extracted.

Returns
filename – The filename with the file extension.

Return type
string

opihiexarata.library.path.**get_filename_without_extension**(*pathname: str*) → str

Get the filename from the pathname without the file extension.

Parameters
pathname (*string*) – The pathname which the filename will be extracted.

Returns
filename – The filename without the file extension.

Return type
string

opihiexarata.library.path.**get_most_recent_filename_in_directory**(*directory: str*, *extension: Optional[Union[str, list]] = None*) → str

This gets the most recent filename from a directory.

Because of issues with different operating systems having differing issues with storing the creation time of a file, this function sorts based off of modification time. The most recent modified file is thus

Parameters

- **directory** (*string*) – The directory by which the most recent file will be derived from.
- **extension** (*string or list, default = None*) – The extension by which to filter for. It is often the case that some files are created but the most recent

file of some type is desired. Only files which match the included extensions will be considered.

Returns
recent_filename – The filename of the most recent file, by modification time, in the directory.

Return type
string

opihiexarata.library.path.**merge_pathname**(*directory: Optional[Union[str, list]] = None*, *filename: Optional[str] = None*, *extension: Optional[str] = None*) → str

Joins directories, filenames, and file extensions into one pathname.

Parameters

- **directory** (*string or list, default = None*) – The directory(s) which is going to be used. If it is a list, then the paths within it are combined.
- **filename** (*string, default = None*) – The filename that is going to be used for path construction.
- **extension** (*string, default = None*) – The filename extension that is going to be used.

Returns
pathname – The combined pathname.

Return type
string

opihiexarata.library.path.**split_pathname**(*pathname: str*) → tuple[str, str, str]

Splits a path into a directory, filename, and file extension.

This is a wrapper function around the more elementry functions *get_directory*, *get_filename_without_extension*, and *get_file_extension*.

Parameters
pathname (*string*) – The combined pathname which to be split.

Returns

- **directory** (*string*) – The directory which was split from the pathname.
- **filename** (*string*) – The filename which was split from the pathname.
- **extension** (*string*) – The filename extension which was split from the pathname.

opihiexarata.library.phototable module

This is a module for uniform handling of the photometric star table. Many different databases will have their own standards and so functions and classes helpful for unifying all of their different entries into one uniform table which the software can expect are detailed here.

opihiexarata.library.phototable.**blank_photometry_table**() → Table

Creates a blank table which contains the columns which are the unified photometry table.

Parameters
None –

Returns
blank_table – The table with only the column headings; no records are in the table.

Return type
Astropy Table

opihiexarata.library.phototable.**fill_incomplete_photometry_table**(*partial_table: Table*) → Table

This function takes a photometry table which is partially standardized but may be missing a few columns or rows and fills in the missing values with NaNs so that the resulting table is a standardized table for the OpihiExarata software.

Metadata is added to the resulting table to signify that it is a complete photometry table as standardized by OpihiExarata.

Parameters

- **partial_table** (*Table*) – A table which partially covers the photometry table standard but is otherwise missing a few records.
- **complete_table** (*Table*) – A completed form of the partial table which is standardized to the expectations of the photometry table.

opihiexarata.library.temporary module

This is where functions dealing with the temporary files and temporary directory of the OpihiExarata system. Temporary files are helpful because they may also contain information useful to the user. These functions thus serve the same purpose as Python's build-in functions, but it is more restricted to OpihiExarata and it is also more persistant.

opihiexarata.library.temporary.**create_temporary_directory**(*unique: Optional[bool] = None*) → None

Make the temporary directory.

Parameters
unique (*bool, default = None*) – Require a check on the creation of the di-

rectory to require it to be unique. If True, this will raise if the directory already exists otherwise it does not care. Will defer to the configuration file if None.

Return type
None

opihiexarata.library.temporary.**delete_temporary_directory**() → None

Delete the temporary directory. If the directory does not exist, this function will do nothing. If the directory exists, but contains files, then this function will fail.

Parameters
None –

Return type
None

opihiexarata.library.temporary.**make_temporary_directory_path**(*filename: str*) → str

Creates a full filename path to use. This function basically adds the temporary directory path prefix to place the filename path into it.

Parameters
filename (*string*) – The filename of the target to be placed in the temporary directory.

Returns
full_path – The full path of the file, as it would be stored in the temporary directory.

Return type
string

opihiexarata.library.temporary.**purge_temporary_directory**() → None

Delete or purge all files in the temporary directory. This does it recursively.

Parameters
None –

Return type
None

Module contents

Common routines which are important functions of Exarata.

opihiexarata.opihi package

Submodules

opihiexarata.opihi.preprocess module

A data wrapper class which takes in raw Opihi data, flats, and darks and produces a valid reduced image.

class opihiexarata.opihi.preprocess.**OpihiPreprocessSolution**(*mask_c_fits_filename: str*, *mask_g_fits_filename: str*, *mask_r_fits_filename: str*, *mask_i_fits_filename: str*, *mask_z_fits_filename: str*, *mask_1_fits_filename: str*, *mask_2_fits_filename: str*, *mask_3_fits_filename: str*, *flat_c_fits_filename: str*, *flat_g_fits_filename: str*, *flat_r_fits_filename: str*, *flat_i_fits_filename: str*, *flat_z_fits_filename: str*, *flat_1_fits_filename: str*, *flat_2_fits_filename: str*, *flat_3_fits_filename: str*, *bias_fits_filename: str*, *dark_current_fits_filename: str*, *linearity_fits_filename: str*)

Bases: *ExarataSolution*

A class which represents the reduction process of Opihi data, having the raw data corrected using previously and provided derived flats and darks. The required parameters (such as exposure time) must also be provided.

This class does not have an engine as there is only one way to reduce data provided the systematics of the Opihi telescope itself; as such the data is handled straight by this solution class.

_mask_c_fits_filename

The filename for the pixel mask for the clear filter stored in a fits file.

Type
string

_mask_g_fits_filename

The filename for the pixel mask for the g filter stored in a fits file.

Type
string

_mask_r_fits_filename

The filename for the pixel mask for the r filter stored in a fits file.

Type
string

_mask_i_fits_filename

The filename for the pixel mask for the i filter stored in a fits file.

Type
string

_mask_z_fits_filename

The filename for the pixel mask for the z filter stored in a fits file.

Type
string

_mask_1_fits_filename

The filename for the pixel mask for the 1 filter stored in a fits file.

Type
string

_mask_2_fits_filename

The filename for the pixel mask for the 2 filter stored in a fits file.

Type
string

_mask_3_fits_filename

The filename for the pixel mask for the 3 filter stored in a fits file.

Type
string

_flat_c_fits_filename

The filename for the flat field for the clear filter stored in a fits file.

Type
string

_flat_g_fits_filename

The filename for the flat field for the g filter stored in a fits file.

Type
string

_flat_r_fits_filename

The filename for the flat field for the r filter stored in a fits file.

Type
string

_flat_i_fits_filename

The filename for the flat field for the i filter stored in a fits file.

Type
string

_flat_z_fits_filename

The filename for the flat field for the z filter stored in a fits file.

Type
string

_flat_1_fits_filename

The filename for the flat field for the 1 filter stored in a fits file.

Type
string

_flat_2_fits_filename

The filename for the flat field for the 2 filter stored in a fits file.

Type
string

_flat_3_fits_filename

The filename for the flat field for the 3 filter stored in a fits file.

Type
string

_bias_fits_filename

The filename for the per-pixel bias values of the data, stored in a fits file.

Type
string

_dark_current_fits_filename

The filename for the per-pixel rate values of the dark data, stored in a fits file.

Type
string

_linearity_fits_filename

The filename for the linearity responce of the CCD. This should be a 1D fits file detailing counts as a function of time for the saturation curve of the CCD.

Type
string

mask_c
[array] The pixel mask for the clear filter as determined by the provided fits file.

mask_g
[array] The pixel mask for the g filter as determined by the provided fits file.

mask_r
[array] The pixel mask for the r filter as determined by the provided fits file.

mask_i
[array] The pixel mask for the i filter as determined by the provided fits file.

mask_z
[array] The pixel mask for the z filter as determined by the provided fits file.

mask_1
[array] The pixel mask for the 1 filter as determined by the provided fits file.

mask_2
[array] The pixel mask for the 2 filter as determined by the provided fits file.

mask_3
[array] The pixel mask for the 3 filter as determined by the provided fits file.

flat_c
[array] The flat field for the clear filter as determined by the provided fits file.

flat_g
[array] The flat field for the g filter as determined by the provided fits file.

flat_r
[array] The flat field for the r filter as determined by the provided fits file.

flat_i
[array] The flat field for the i filter as determined by the provided fits file.

flat_z
[array] The flat field for the z filter as determined by the provided fits file.

flat_1
[array] The flat field for the 1 filter as determined by the provided fits file.

flat_2
[array] The flat field for the 2 filter as determined by the provided fits file.

flat_3
[array] The flat field for the 3 filter as determined by the provided fits file.

bias
[array] The bias array as determined by the provided fits file.

dark_current
[array] The dark rate, per pixel, as determined by the provided fits file.

linearity_factors
[array] The polynomial factors of the linearity function starting from the 0th order.

linearity_function
[function] The linearity function across the whole CCD. It is an average function across all of the pixels.

__init__(*mask_c_fits_filename: str*, *mask_g_fits_filename: str*, *mask_r_fits_filename: str*, *mask_i_fits_filename: str*, *mask_z_fits_filename: str*, *mask_1_fits_filename: str*, *mask_2_fits_filename: str*, *mask_3_fits_filename: str*, *flat_c_fits_filename: str*, *flat_g_fits_filename: str*, *flat_r_fits_filename: str*, *flat_i_fits_filename: str*, *flat_z_fits_filename: str*, *flat_1_fits_filename: str*, *flat_2_fits_filename: str*, *flat_3_fits_filename: str*, *bias_fits_filename: str*, *dark_current_fits_filename: str*, *linearity_fits_filename: str*) → None

Instantiation of the reduced Opihi data class.

Parameters

- **mask_c_fits_filename** (*string*) – The filename for the pixel mask in the clear filter stored in a fits file.
- **mask_g_fits_filename** (*string*) – The filename for the pixel mask in the g filter stored in a fits file.
- **mask_r_fits_filename** (*string*) – The filename for the pixel mask in the r filter stored in a fits file.
- **mask_i_fits_filename** (*string*) – The filename for the pixel mask in the i filter stored in a fits file.
- **mask_z_fits_filename** (*string*) – The filename for the pixel mask in the z filter stored in a fits file.
- **mask_1_fits_filename** (*string*) – The filename for the pixel mask in the 1 filter stored in a fits file.
- **mask_2_fits_filename** (*string*) – The filename for the pixel mask in the 2 filter stored in a fits file.

- **mask_3_fits_filename** (*string*) – The filename for the pixel mask in the 3 filter stored in a fits file.
- **flat_c_fits_filename** (*string*) – The filename for the flat field in the clear filter stored in a fits file.
- **flat_g_fits_filename** (*string*) – The filename for the flat field in the g filter stored in a fits file.
- **flat_r_fits_filename** (*string*) – The filename for the flat field in the r filter stored in a fits file.
- **flat_i_fits_filename** (*string*) – The filename for the flat field in the i filter stored in a fits file.
- **flat_z_fits_filename** (*string*) – The filename for the flat field in the z filter stored in a fits file.
- **flat_1_fits_filename** (*string*) – The filename for the flat field in the 1 filter stored in a fits file.
- **flat_2_fits_filename** (*string*) – The filename for the flat field in the 2 filter stored in a fits file.
- **flat_3_fits_filename** (*string*) – The filename for the flat field in the 3 filter stored in a fits file.

bias_fits_filename
[string] The filename for the per-pixel bias values of the data, stored in a fits file.

dark_current_fits_filename
[string] The filename for the per-pixel rate values of the dark data, stored in a fits file.

linearity_fits_filename
[string] The filename for the linearity responce of the CCD, stored as a text file.

Return type
None

__init_read_flat_data() → None

This function just reads all of the fits file data for the filter-dependent flat fields and puts it where it belongs per the documentation.

Parameters
None –

Return type
None

__init_read_linearity_data()

This function reads all of the linearity data and creates a function for linearity. First order interpolation is done on this data.

It is expected that the data from the linearity filename is of high enough resolution that first order interpolation is good enough.

Parameters
None –

Return type
None

__init_read_mask_data() → None

This function just reads all of the fits file data for the filter-dependent pixel masks and puts it where it belongs per the documentation.

Parameters
None –

Return type
None

preprocess_data_image(*raw_data: ndarray*[103], *exposure_time: float*, *filter_name: str*) → ndarray[104]

The formal reduction algorithm for data from Opihi. It follows preprocessing instructions for CCDs.

Parameters

- **data** (*array-like*) – The raw image data from the Opihi telescope.
- **exposure_time** (*float*) – The exposure time of the image in seconds.
- **filter_name** (*string*) – The name of the filter which the image was taken in, used to select the correct flat and mask file.

Returns
preprocess_data – The data, after it has been preprocessed.

Return type
array

preprocess_fits_file(*raw_filename: str*, *out_filename: Optional[str] = None*) → tuple[astropy.io.fits.header.Header, numpy.ndarray[105]]

Preprocess an Opihi image, the provided fits filename is read, the needed information extracted from it, and it is processed using historical archive calibration files created per the documentation and specified by the configuration files.

Parameters

- **raw_filename** (*str*) – The filename of the raw fits file image from Opihi.
- **out_filename** (*str, default = None*) – The filename to save the reduced image as a fits file. Some added entries are added to the header. If this is not provided as defaults to None, no file is saved.

Returns

- **preprocess_header** (*Astropy Header*) – The header of the fits file after preprocessing. Some added entries are present to document information from preprocessing.
- **preprocess_data** (*array*) – The data array of the image after the raw image went through the preprocess reduction.

opihiexarata.opihi.solution module

This is the class for a collection of solutions which the GUI interacts with and acts as the complete solver. There is not engine as it just shuffles the solutions.

class opihiexarata.opihi.solution.**OpihiSolution**(*fits_filename: str*, *filter_name: Optional[str] = None*, *exposure_time: Optional[float] = None*, *observing_time: Optional[float] = None*, *asteroid_name: Optional[str] = None*, *asteroid_location: Optional[tuple[float, float]] = None*, *asteroid_history: Optional[list[str]] = None*)

Bases: *ExarataSolution*

This is the main class which acts as a collection container of solution classes. It facilitates the interaction between the solution classes and the GUI.

fits_filename

The fits filename of which is the image which this solution is solving.

Type
str

filter_name

The filter_name which this image is taken in.

Type
str

exposure_time

The exposure time of the image, in seconds.

Type
float

observing_time

The time of observation, this must be a Julian day time.

[103] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[104] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[105] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

Type
float

asteroid_name

The name of the asteroid. This is used to group similar observations and to also retrieve data from the MPC.

Type
str

asteroid_location

The pixel location of the asteroid. (Usually determined by a centroid around a user specified location.) If this is None, then asteroid calculations are disabled as there is no asteroid.

Type
tuple

asteroid_history

The total observational history of the asteroid provided. This includes previous observations done by Opihi and processed by OpihiExarata, but does not include the current one. This is the 80-column text file form of a MPC record. If this is None, then asteroid calculations are disabled as there is no asteroid.

Type
list

asteroid_observations

The total observational history of the asteroid provided. This includes previous observations done by Opihi and processed by OpihiExarata, but does not include the current data. This is the table form of a MPC record. If this is None, then asteroid calculations are disabled as there is no asteroid.

Type
table

header

The header of the fits file.

Type
Astropy Header

data

The image data of the fits file itself.

Type
array

astrometrics
[AstrometricSolution] The astrometric solution; if it has not been solved yet, this is None.

photometrics
[PhotometricSolution] The photometric solution; if it has not been solved yet, this is None.

propagatives
[PropagativeSolution] The propagation solution; if it has not been solved yet, this is None.

orbitals
[OrbitalSolution] The orbit solution; if it has not been solved yet, this is None.

ephemeritics
[EphemeriticSolution] The ephemeris solution; if it has not been solved yet, this is None.

astrometrics_status
[bool, None] The status of the solving of the astrometric solution. It is True or False based on the success of the solve, None if a solve has not been attempted.

photometrics_status
[bool, None] The status of the solving of the photometric solution. It is True or False based on the success of the solve, None if a solve has not been attempted.

propagatives_status
[bool, None] The status of the solving of the propagative solution. It is True or False based on the success of the solve, None if a solve has not been attempted.

orbitals_status
[bool, None] The status of the solving of the orbital solution. It is True or False based on the success of the solve, None if a solve has not been attempted.

ephemeritics_status
[bool, None] The status of the solving of the ephemeris solution. It is True or False based on the success of the solve, None if a solve has not been attempted.

__init__ (*fits_filename: str*, *filter_name: Optional[str] = None*, *exposure_time: Optional[float] = None*, *observing_time: Optional[float] = None*, *asteroid_name: Optional[str] = None*, *asteroid_location: Optional[tuple[float, float]] = None*, *asteroid_history: Optional[list[str]] = None*) → None

Creating the main solution class.

All of the data which is needed to derive the other solutions should be provided. The solutions, however, are only done when called. Overriding parameters can be applied when calling the solutions.

If the asteroid input values are not provided, then this class will prohibit calculations meant for asteroids because of the lack of an asteroid.

Parameters

- **fits_filename** (*str*) – The fits filename of which is the image which this solution is solving.
- **filter_name** (*string, default=None*) – The filter_name of the image which is contained within the data array. If None, we attempt to pull the value from the fits file.
- **exposure_time** (*float, default=None*) – The exposure time of the image, in seconds. If None, we attempt to pull the value from the fits file.
- **observing_time** (*float, default=None*) – The time of observation, this time must in Julian day. If None, we attempt to pull the value from the fits file.

- **asteroid_name** (*str, default = None*) – The name of the asteroid.
- **asteroid_location** (*tuple, default = None*) – The pixel location of the asteroid.
- **asteroid_history** (*list, default = None*) – The history of observations of an asteroid written in a standard 80-column MPC record.

mpc_record_row() → str

Returns an 80-character record describing the observation of this object assuming it is an asteroid. It only uses information that is provided and does not attempt to compute any solutions.

Parameters
None –

Returns
record_row – The 80-character record as determined by the MPC specification.

Return type
str

mpc_table_row() → Table

An MPC table of the current observation with information provided by solved solutions. This routine does not attempt to do any solutions.

Parameters
None –

Returns
table_row – The MPC table of the information. It is a single row table.

Return type
Astropy Table

solve_astrometry(*solver_engine:* AstrometryEngine, *overwrite: bool = True*, *raise_on_error: bool = False*, *vehicle_args: dict = {}*) → *AstrometricSolution*

Solve the image astrometry by using an astrometric engine.

Parameters

- **solver_engine** (AstrometryEngine) – The astrometric engine which the astrometry solver will use.
- **overwrite** (*bool, default = True*) – Overwrite and replace the information of this class with the new values. If False, the returned solution is not also applied.
- **raise_on_error** (*bool, default = False*) – If True, this disables the error handing and allows for errors from the solving engines/solutions to be propagated out.
- **vehicle_args** (*dictionary, default = {}*) – If the vehicle function for the provided solver engine needs extra parameters not otherwise provided by the standard input, they are given here.

Returns

- **astrometric_solution** (*AstrometricSolution*) – The astrometry solution for the image.
- **solve_status** (*bool*) – The status of the solve. If True, the solving was successful.

solve_ephemeris (*solver_engine:* EphemerisEngine, *overwrite: bool = True*, *raise_on_error: bool = False*, *vehicle_args: dict = {}*)

Solve for the ephemeris solution an asteroid using previous observations and derived orbital elements.

Parameters

- **solver_engine** (EphemerisEngine) – The ephemeris engine which the ephemeris solver will use.
- **overwrite** (*bool, default = True*) – Overwrite and replace the information of this class with the new values. If False, the returned solution is not also applied.
- **raise_on_error** (*bool, default = False*) – If True, this disables the error handing and allows for errors from the solving engines/solutions to be propagated out.
- **vehicle_args** (*dictionary, default = {}*) – If the vehicle function for the provided solver engine needs extra parameters not otherwise provided by the standard input, they are given here.

Returns

- **ephemeritics_solution** (*EphemeriticSolution*) – The orbit solution for the asteroid and image.
- **solve_status** (*bool*) – The status of the solve. If True, the solving was successful.
- *Warning* .. – This requires that the orbital solution be computed before-hand. It will not be precomputed automatically; without it being called explicitly, this will instead raise an error.

solve_orbit (*solver_engine:* OrbitEngine, *overwrite: bool = True*, *raise_on_error: bool = False*, *asteroid_location: Optional[tuple] = None*, *vehicle_args: dict = {}*)

Solve for the orbital elements an asteroid using previous observations.

Parameters

- **solver_engine** (OrbitEngine) – The orbital engine which the orbit solver will use.
- **overwrite** (*bool, default = True*) – Overwrite and replace the information of this class with the new values. If False, the returned solution is not also applied.
- **asteroid_location** (*tuple, default = None*) – The pixel location of the asteroid in the image. Defaults to the value provided at instantiation.

- **raise_on_error** (*bool, default = False*) – If True, this disables the error handing and allows for errors from the solving engines/solutions to be propagated out.
- **vehicle_args** (*dictionary, default = {}*) – If the vehicle function for the provided solver engine needs extra parameters not otherwise provided by the standard input, they are given here.

Returns

- **orbital_solution** (*OrbitalSolution*) – The orbit solution for the asteroid and image.
- **solve_status** (*bool*) – The status of the solve. If True, the solving was successful.
- *Warning* .. – This requires that the astrometric solution be computed before-hand. It will not be precomputed automatically; without it being called explicitly, this will instead raise an error.

solve_photometry (*solver_engine:* PhotometryEngine, *overwrite: bool = True*, *raise_on_error: bool = False*, *filter_name: Optional[str] = None*, *exposure_time: Optional[float] = None*, *vehicle_args: dict = {}*) → *PhotometricSolution*

Solve the image photometry by using a photometric engine.

Parameters

- **solver_engine** (PhotometryEngine) – The photometric engine which the photometry solver will use.
- **overwrite** (*bool, default = True*) – Overwrite and replace the information of this class with the new values. If False, the returned solution is not also applied.
- **filter_name** (*string, default = None*) – The filter name of the image, defaults to the value provided at instantiation.
- **exposure_time** (*float, default = None*) – The exposure time of the image, in seconds. Defaults to the value provided at instantiation.
- **raise_on_error** (*bool, default = False*) – If True, this disables the error handing and allows for errors from the solving engines/solutions to be propagated out.
- **vehicle_args** (*dictionary, default = {}*) – If the vehicle function for the provided solver engine needs extra parameters not otherwise provided by the standard input, they are given here.

Returns

- **photometric_solution** (*PhotometrySolution*) – The photometry solution for the image.
- **solve_status** (*bool*) – The status of the solve. If True, the solving was successful.
- *Warning* .. – This requires that the astrometric solution be computed before-hand. It will not be precomputed automatically; without it being called explicitly, this will instead raise an error.

solve_propagate(*solver_engine:* PropagationEngine, *overwrite: bool = True*, *raise_on_error: bool = False*, *asteroid_location: Optional[tuple[float, float]] = None*, *vehicle_args: dict = {}*) → *PropagativeSolution*

Solve for the location of an asteroid using a method of propagation.

Parameters

- **solver_engine** (*PropagationEngine*) – The propagative engine which the propagation solver will use.
- **overwrite** (*bool, default = True*) – Overwrite and replace the information of this class with the new values. If False, the returned solution is not also applied.
- **asteroid_location** (*tuple, default = None*) – The pixel location of the asteroid in the image. Defaults to the value provided at instantiation.
- **raise_on_error** (*bool, default = False*) – If True, this disables the error handing and allows for errors from the solving engines/solutions to be propagated out.
- **vehicle_args** (*dictionary, default = {}*) – If the vehicle function for the provided solver engine needs extra parameters not otherwise provided by the standard input, they are given here.

Returns

- **propagative_solution** (*PropagativeSolution*) – The propagation solution for the asteroid and image.
- **solve_status** (*bool*) – The status of the solve. If True, the solving was successful.
- *Warning ..* – This requires that the astrometric solution be computed before-hand. It will not be precomputed automatically; without it being called explicitly, this will instead raise an error.

Module contents

The class for the collection of Exarata solutions.

opihiexarata.orbit package

Submodules

opihiexarata.orbit.custom module

This is a class which defines a custom orbit. A user supplies the orbital elements to this engine and the vehicle function.

class opihiexarata.orbit.custom.**CustomOrbitEngine**(*semimajor_axis: float*, *eccentricity: float*, *inclination: float*, *longitude_ascending_node: float*, *argument_perihelion: float*, *mean_anomaly: float*, *epoch_julian_day: float*, *semimajor_axis_error: Optional[float] = None*, *eccentricity_error: Optional[float] = None*, *inclination_error: Optional[float] = None*, *longitude_ascending_node_error: Optional[float] = None*, *argument_perihelion_error: Optional[float] = None*, *mean_anomaly_error: Optional[float] = None*)

Bases: *OrbitEngine*

This engine is just a wrapper for when a custom orbit is desired to be specified.

semimajor_axis

The semi-major axis of the orbit provided, in AU.

Type
float

semimajor_axis_error

The error on the semi-major axis of the orbit provided, in AU.

Type
float

eccentricity

The eccentricity of the orbit provided.

Type
float

eccentricity_error

The error on the eccentricity of the orbit provided.

Type
float

inclination

The angle of inclination of the orbit provided, in degrees.

Type
float

inclination_error

The error on the angle of inclination of the orbit provided, in degrees.

Type
float

longitude_ascending_node

The longitude of the ascending node of the orbit provided, in degrees.

Type
float

longitude_ascending_node_error

The error on the longitude of the ascending node of the orbit provided, in degrees.

Type
float

argument_perihelion

The argument of perihelion of the orbit provided, in degrees.

Type
float

argument_perihelion_error

The error on the argument of perihelion of the orbit provided, in degrees.

Type
float

mean_anomaly

The mean anomaly of the orbit provided, in degrees.

Type
float

mean_anomaly_error

The error on the mean anomaly of the orbit provided, in degrees.

Type
float

epoch_julian_day

The epoch where for these osculating orbital elements. This value is in Julian days.

Type
float

__init__(*semimajor_axis: float*, *eccentricity: float*, *inclination: float*, *longitude_ascending_node: float*, *argument_perihelion: float*, *mean_anomaly: float*, *epoch_julian_day: float*, *semimajor_axis_error: Optional[float] = None*, *eccentricity_error: Optional[float] = None*, *inclination_error: Optional[float] = None*, *longitude_ascending_node_error: Optional[float] = None*, *argument_perihelion_error: Optional[float] = None*, *mean_anomaly_error: Optional[float] = None*) → None

The orbital elements are already provided for this custom solution. If errors may optionally be provided.

Parameters

- **semimajor_axis** (*float*) – The semi-major axis of the orbit provided, in AU.
- **eccentricity** (*float*) – The eccentricity of the orbit provided.
- **inclination** (*float*) – The angle of inclination of the orbit provided, in degrees.
- **longitude_ascending_node** (*float*) – The longitude of the ascending node of the orbit provided, in degrees.
- **argument_perihelion** (*float*) – The argument of perihelion of the orbit provided, in degrees.
- **mean_anomaly** (*float*) – The mean anomaly of the orbit provided, in degrees.
- **epoch_julian_day** (*float*) – The epoch where for these osculating orbital elements. This value is in Julian days.
- **semimajor_axis_error** (*float, default = None*) – The error on the semi-major axis of the orbit provided, in AU.
- **eccentricity_error** (*float, default = None*) – The error on the eccentricity of the orbit provided.
- **inclination_error** (*float, default = None*) – The error on the angle of inclination of the orbit provided, in degrees.
- **longitude_ascending_node_error** (*float, default = None*) – The error on the longitude of the ascending node of the orbit provided, in degrees.
- **argument_perihelion_error** (*float, default = None*) – The error on the argument of perihelion of the orbit provided, in degrees.
- **mean_anomaly_error** (*float, default = None*) – The error on the mean anomaly of the orbit provided, in degrees.

Return type
None

opihiexarata.orbit.orbfit module

This contains the Python wrapper class around Orbfit, assuming the installation procedure was followed.

class opihiexarata.orbit.orbfit.**OrbfitOrbitDeterminerEngine**

Bases: *OrbitEngine*

Uses the Orbfit package to determine the orbital elements of an astroid provided observations. This assumes that the installation instructions provided were followed.

orbital_elements

The six Keplarian orbital elements.

Type
dict

orbital_elements_errors

The errors of the orbital elements.

Type
dict

classmethod __check_installation(*no_raise: bool = False*) → bool

Check if the installation was completed according to the documentation provided. This functions only checks for the existence of the template files.

Parameters
no_raise (*bool, default = False*) – By default, an invalid install will raise. Set this if a False return without interrupting the flow of the program is desired.

Returns
valid_install – If the installation is detected to be valid, then this returns True.

Return type
bool

__init__() → None

Instantiation of the orbfit package.

Parameters
None –

Return type
None

_clean_orbfit_files() → None

This function cleans up the operational directory of Orbfit. If there are leftover files, the program may try to use them in a manner which is not desired. It is usually better to start from scratch each time to avoid these issues.

Parameters
None –

Return type
None

_prepare_orbfit_files() → None

This function prepares the operational directory for Orbfit inside of the temporary directory. This allows for files to be generated for useage by the binary orbfit.

Parameters
None –

Return type
None

_solve_single_orbit(*observation_table: Table*) → tuple[dict, dict, float]

This uses the Orbfit program to an orbit provided a record of observations. If it cannot be solved, an error is raised.

This function is not intended to solve an entire set of observations, but instead a subset of them. The rational being that orbfit tends to fail on larger sets; averaging the values of smaller sets is a little more robust against failure. Use the non-hidden function to compute orbital elements for a full range of observations.

Parameters
observation_table (*Astropy Table*) – The table of observations; this will be converted to the required formats for processing.

Returns

- **kepler_elements** (*dict*) – The Keplarian orbital elements.
- **kepler_error** (*dict*) – The error on the Keplarian orbital elements.
- **modified_julian_date** (*float*) – The modified Julian date corresponding to the osculating orbit and the Keplerian orbital parameters provided.

solve_orbit(*observation_table: Table*) → tuple[dict, dict, float]

Attempts to compute Keplarian orbits provided a table of observations.

This function attempts to compute the orbit using the entire observation table. If it is unable to, then the observations are split into subsets based on the year of observations. The derived orbital elements are averaged and errors propagated. If no orbit is found, then an error is raised.

Parameters
observation_record (*Astropy Table*) – The table of observational records; this will be converted to the required MPC 80 column format.

Returns

- **kepler_elements** (*dict*) – The Keplarian orbital elements.
- **kepler_error** (*dict*) – The error on the Keplarian orbital elements.
- **modified_julian_date** (*float*) – The modified Julian date corresponding to the osculating orbit and the Keplerian orbital parameters provided.

solve_orbit_via_record(*observation_record: list[str]*) → tuple[dict, dict, float]

Attempts to compute Keplarian orbits provided a standard 80-column record of observations.

This function calls and depends on *solve_orbfit*.

Parameters

observation_record (*Astropy Table*) – The record of observations in the standard 80-column format.

Returns

- **kepler_elements** (*dict*) – The Keplarian orbital elements.
- **kepler_error** (*dict*) – The error on the Keplarian orbital elements.
- **modified_julian_date** (*float*) – The modified Julian date corresponding to the osculating orbit and the Keplerian orbital parameters provided.

opihiexarata.orbit.solution module

The orbit solution class.

class opihiexarata.orbit.solution.**OrbitalSolution**(*observation_record: list[str]*, *solver_engine:* OrbitEngine, *vehicle_args: dict = {}*)

Bases: *ExarataSolution*

This is the class which solves a record of observations to derive the orbital parameters of asteroids or objects in general. A record of observations must be provided.

semimajor_axis

The semi-major axis of the orbit solved, in AU.

Type
float

semimajor_axis_error

The error on the semi-major axis of the orbit solved, in AU.

Type
float

eccentricity

The eccentricity of the orbit solved.

Type
float

eccentricity_error

The error on the eccentricity of the orbit solved.

Type
float

inclination

The angle of inclination of the orbit solved, in degrees.

Type
float

inclination_error

The error on the angle of inclination of the orbit solved, in degrees.

Type
float

longitude_ascending_node

The longitude of the ascending node of the orbit solved, in degrees.

Type
float

longitude_ascending_node_error

The error on the longitude of the ascending node of the orbit solved, in degrees.

Type
float

argument_perihelion

The argument of perihelion of the orbit solved, in degrees.

Type
float

argument_perihelion_error

The error on the argument of perihelion of the orbit solved, in degrees.

Type
float

mean_anomaly

The mean anomaly of the orbit solved, in degrees.

Type
float

mean_anomaly_error

The error on the mean anomaly of the orbit solved, in degrees.

Type
float

true_anomaly

The true anomaly of the orbit solved, in degrees. This value is calculated from the mean anomaly.

Type
float

true_anomaly_error
The error on the true anomaly of the orbit solved, in degrees. This value is calculated from the error on the mean anomaly.

Type
float

epoch_julian_day
The epoch where for these osculating orbital elements. This value is in Julian days.

Type
float

__calculate_eccentric_anomaly() → tuple[float, float]
Calculating the eccentric anomaly and error from the mean anomaly.

Parameters
None –

Returns

- **eccentric_anomaly** (*float*) – The eccentric anomaly as derived from the mean anomaly.
- **eccentric_anomaly_error** (*float*) – The eccentric anomaly error derived as an average from the upper and lower ranges of the mean anomaly.

__calculate_true_anomaly() → tuple[float, float]
Calculating the true anomaly and error from the eccentric anomaly.

Parameters
None –

Returns

- **true_anomaly** (*float*) – The true anomaly as derived from the mean anomaly.
- **true_anomaly_error** (*float*) – The true anomaly error derived as an average from the upper and lower ranges of the eccentric anomaly.

__init__(*observation_record: list[str]*, *solver_engine:* OrbitEngine, *vehicle_args: dict = {}*) → None
The initialization function. Provided the list of observations, solves the orbit for the Keplarian orbits.

Parameters

- **observation_record** (*list*) – A list of the standard MPC 80-column observation records. Each element of the list should be a string representing the observation.
- **solver_engine** (OrbitEngine) – The engine which will be used to complete the orbital elements from the observation record.

- **vehicle_args** (*dictionary*) – If the vehicle function for the provided solver engine needs extra arguments not otherwise provided by the standard input, they are given here.

Return type
None

opihiexarata.orbit.solution.**_calculate_eccentric_anomaly**(*mean_anomaly: float*, *eccentricity: float*) → float

Calculate the eccentric anomaly from the mean anomaly and eccentricity of an orbit. This is found iteratively using Newton's method.

Parameters
mean_anomaly (*float*) – The mean anomaly of the orbit, in degrees.

Returns
eccentric_anomaly – The eccentric anomaly as derived from the mean anomaly, in degrees.

Return type
float

opihiexarata.orbit.solution.**_calculate_true_anomaly**(*eccentric_anomaly: float*, *eccentricity: float*) → float

Calculate the true anomaly from the mean anomaly and eccentricity of an orbit.

We use the more numerically stable equation from https://ui.adsabs.harvard.edu/abs/1973CeMec…7..388B[106].

Parameters
eccentric_anomaly (*float*) – The eccentric anomaly of the orbit, in degrees.

Returns
true_anomaly – The true anomaly as derived from the eccentric anomaly, in degrees.

Return type
float

opihiexarata.orbit.solution.**_vehicle_custom_orbit**(*observation_record: list[str]*, *vehicle_args: dict*) → dict

This is the vehicle function for the specification of a custom orbit.

A check is done for the extra vehicle arguments to ensure that the orbital elements desired are within the arguments. The observation record is not of concern for this vehicle.

Parameters

- **observation_record** (*list*) – The MPC standard 80-column record for observations of the asteroid by which the orbit shall be computed from.
- **vehicle_args** (*dict*) – The arguments to be passed to the Engine class to help its creation and solving abilities. In this case, it is just the orbital elements as defined.

[106] https://ui.adsabs.harvard.edu/abs/1973CeMec...7..388B

opihiexarata.orbit.solution.**_vehicle_orbfit_orbit_determiner**(*observation_record: list[str]*) → dict

This uses the Orbfit engine to calculate orbital elements from the observation record. The results are then returned to be managed by the main class.

Parameters
observation_record (*list*) – The MPC standard 80-column record for observations of the asteroid by which the orbit shall be computed from.

Returns
orbit_results – The results of the orbit computation using the Orbfit engine. Namely, this returns the 6 classical Kepler elements, using mean anomaly.

Return type
dict

Module contents

opihiexarata.photometry package

Submodules

opihiexarata.photometry.panstarrs module

Photometric database access using PANSTARRS data. There are a few ways and they are implemented here.

class opihiexarata.photometry.panstarrs.**PanstarrsMastWebAPIEngine**(*verify_ssl: bool = True*)

Bases: *PhotometryEngine*

This is a photometric data extractor using PanSTARRS data obtained from their catalogs via the MAST API.

See https://catalogs.mast.stsci.edu/docs/panstarrs.html for more information.

__init__(*verify_ssl: bool = True*) → None

Create the instance of the API.

Parameters
verify_ssl (*boolean, default = True*) – Connecting to the MAST API usually uses SSL verification via HTTPS, set to False to allow bypassing this.

Return type
None

_mask_table_data(*data_table: Table*) → Table

This masks the raw data derived from PanSTARRS, implementing the masking/null value specifics of the PanSTARRS system.

The point of this is to make masked or invalid data more typical to the user by abstracting the idiosyncrasies of PanSTARRS.

Parameters
data_table (*Astropy Table*) – The data table to be cleaned up.

Returns
masked_data_table – The masked table.

Return type
Astropy Table

cone_search(*ra: float*, *dec: float*, *radius: float*, *detections: int = 3*, *color_detections: int = 1*, *columns: Optional[list[str]] = None*, *max_rows: int = 1000*, *data_release: int = 2*) → Table

Search the PanSTARRS database for targets within a cone region specified.

The table data returned from this function is not processed and is raw from the fetching of the results.

Parameters

- **ra** (*float*) – The right ascension of the center point of the cone search, in degrees.
- **dec** (*float*) – The declination of the center point of the cone search, in degrees.
- **radius** (*float*) – The radius from the center point of the cone search in which to search, in degrees.
- **detections** (*int, default = 3*) – The minimum number of detections each object needs to have to be included.
- **color_detections** (*int, default = 1*) – The minimum number of detections for the g, r, i, z filters of the Sloan filters of PanSTARRS. As this is a photometric engine for OpihiExarata, the filters should be the ones pertinent to the telescope.
- **columns** (*list*) – The columns that are desired to be pulled. The purpose of this is to lighten the data download load. If None, then it defaults to all columns.
- **max_rows** (*int, default = 1000*) – The maximum entries that will be pulled from the server.
- **data_release** (*int, default = 2*) – The PanSTARRS data release version from which to take the data from.

Returns
catalog_results – The result of the cone search, pulled from the PanSTARRS catalog.

Return type
Astropy Table

masked_cone_search(**args*, ***kwargs*) → Table

The same as cone_search, but it also masks the data based on the masking idiosyncrasies of PanSTARRS.

Parameters
cone_search) (*(see)* –

Returns
masked_catalog_results – The data from the cone search with the entries masked where appropriate.

Return type
Astropy Table

opihiexarata.photometry.solution module

The general photometric solver.

class opihiexarata.photometry.solution.**PhotometricSolution**(*fits_filename: str*, *solver_engine:* PhotometryEngine, *astrometrics:* AstrometricSolution, *exposure_time: Optional[float] = None*, *filter_name: Optional[str] = None*, *vehicle_args: dict = {}*)

Bases: *ExarataSolution*

The primary class describing an photometric solution, based on an image provided and catalog data provided from the photometric engine.

This class is the middlewere class between the engines which solve the photometry, and the rest of the OpihiExarata code.

_original_filename

The original filename where the fits file is stored at, or copied to.

Type
string

_original_header

The original header of the fits file that was pulled to solve for this astrometric solution.

Type
Header

_original_data
The original data of the fits file that was pulled to solve for this photometric solution.

Type
array-like

astrometrics
The astrometric solution that is required for the photometric solution.

Type
AstrometricSolution

sky_counts
The average sky contribution per pixel.

Type
float

star_table
A table of stars around the image with their RA, DEC, and filter magnitudes. It is not guaranteed that this star table and the astrometric star table is correlated.

Type
Table

intersection_star_table
A table of stars with both the astrometric and photometric RA and DEC coordinates found by the astrometric solution and the photometric engine. The filter magnitudes of these stars are also provided. It is guaranteed that the stars within this table are correlated.

Type
Table

exposure_time
How long, in seconds, the image in question was exposed for.

Type
float

filter_name
A single character string describing the name of the filter band that this image was taken in. Currently, it assumes the MKO/SDSS visual filters.

Type
string

available_filters
The list of filter names which the star table currently covers and has data for.

Type
tuple

zero_point

The zero point of the image.

Type
float

zero_point_error

The standard deviation of the error point mean as calculated using many stars.

Type
float

__calculate_intersection_star_table() → Table

This function determines the intersection star table.

Basically this function matches the entries in the astrometric star table with the photometric star table. This function accomplishes that by simply associating the closest entires as the same star. The distance function assumes a tangent sky projection.

Parameters
None –

Returns
intersection_table – The intersection of the astrometric and photometric star tables giving the correlated star entries between them.

Return type
Table

__calculate_sky_counts_mask() → ndarray[107]

Calculate a mask which blocks out all but the sky for sky counts determination.

The method used is to exclude the regions where stars exist (as determined by the star tables) and also the central region of the image (as it is expected that there is a science object there).

Parameters
None –

Returns
sky_counts_mask – The mask which masks out which is not interesting regarding sky count calculations.

Return type
float

__calculate_sky_counts_value() → float

Calculate the background sky value, in counts, from the image. Obviously needed for photometric calibrations.

The regions outside of the sky mask represent the sky and the sky counts is extracted from that.

Parameters
None –

Returns
sky_counts – The total number of counts, in DN that, on average, the sky contributes per pixel.

Return type
float

__init__ (*fits_filename: str*, *solver_engine:* PhotometryEngine, *astrometrics:* AstrometricSolution, *exposure_time: Optional[float] = None*, *filter_name: Optional[str] = None*, *vehicle_args: dict = {}*) → None

Initialization of the photometric solution.

Parameters

- **fits_filename** (*string*) – The path of the fits file that contains the data for the astrometric solution.
- **solver_engine** (PhotometryEngine) – The photometric solver engine class. This is what will act as the "behind the scenes" and solve the field, using this middlewhere to translate it into something that is easier.
- **astrometrics** (AstrometricSolution, *default = None*) – A precomputed astrometric solution which belongs to this image.
- **exposure_time** (*float, default = None*) – How long, in seconds, the image in question was exposed for. If not provided, calculation of the zero-point is skipped.
- **filter_name** (*string, default = None*) – A single character string describing the name of the filter band that this image was taken in. Currently, it assumes the MKO/SDSS visual filters. If it is None, calculation of the zero-point is skipped.
- **vehicle_args** (*dictionary*) – If the vehicle function for the provided solver engine needs extra parameters not otherwise provided by the standard input, they are given here.

Return type
None

_calculate_star_photon_counts_coordinate (*ra: float*, *dec: float*, *radius: float*) → float

Calculate the total number of photometric counts at an RA DEC. The counts are already corrected for the sky counts.

This function does not check if a star is actually there. This function is a wrapper around its pixel version, converting via the WCS solution.

Parameters

- **ra** (*float*) – The right ascension in degrees.
- **dec** (*float*) – The declination in degrees.

- **radius** (*float*) – The radius of the circular aperture to be considered, in degrees.

Returns
photon_counts – The sum of the sky corrected counts for the region defined.

Return type
float

_calculate_star_photon_counts_pixel(*pixel_x: int*, *pixel_y: int*, *radius: float*) → float

Calculate the total number of photometric counts at a pixel location. The counts are already corrected for the sky counts.

This function does not check if a star is actually there.

Parameters

- **pixel_x** (*int*) – The x coordinate of the center pixel.
- **pixel_y** (*int*) – The y coordinate of the center pixel.
- **radius** (*float*) – The radius of the circular aperture to be considered in pixel counts.

Returns
photon_counts – The sum of the sky corrected counts for the region defined.

Return type
float

_calculate_zero_point(*exposure_time: float*, *filter_name: Optional[str] = None*) → float

This function calculates the photometric zero-point of the image provided the data in the intersection star table.

This function uses the set exposure time and the intersection star table. The band is also assumed from the initial parameters.

Parameters

- **exposure_time** (*float*) – How long, in seconds, the image in question was exposed for.
- **filter_name** (*string, default = None*) – A single character string describing the name of the filter band that this image was taken in. Currently, it assumes the MKO/SDSS visual filters. If it is None, then this function does nothing.

Returns

- **zero_point** (*float*) – The zero point of this image. This is computed as a mean of all of the calculated zero points.
- **zero_point_error** (*float*) – The standard deviation of the zero points calculated.

[107] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

opihiexarata.photometry.solution.**_vehicle_panstarrs_mast_web_api**(*ra: float*, *dec: float*, *radius: float*) → dict

A vehicle function for photometric solutions. Extract photometric data using the PanSTARRS database accessed via the MAST API.

Parameters

- **ra** (*float*) – The right ascension of the center of the area to extract from, in degrees.
- **dec** (*float*) – The declination of the center of the area to extract from, in degrees.
- **radius** (*float*) – The search radius from the center that defines the search area.

Returns
photometry_results – The results of the photometry engine.

Return type
dictionary

Module contents

opihiexarata.propagate package

Submodules

opihiexarata.propagate.polynomial module

For polynomial fitting propagation, using approximations of 1st or 2nd order terms but ignoring some spherical effects.

Although this could be easily implemented in a better method using subclassing rather than having two classes, as having a 3rd order is not really feasible, and for the sake of readability and stability, two separate copy-like classes are written.

class opihiexarata.propagate.polynomial.**LinearPropagationEngine**(*ra: ndarray*[108], *dec: ndarray*[109], *obs_time: ndarray*[110])

Bases: *PropagationEngine*

A simple propagation engine which uses 1st order extrapolation of RA DEC points independently to determine future location.

ra_array

The array of right ascension measurements to extrapolate to.

Type
ndarray

dec_array

The array of declinations measurements to extrapolate to.

Type
ndarray

obs_time_array

The array of observation times which the RA and DEC measurements were taken at. The values are in Julian days.

Type
ndarray

ra_poly_param

The polynomial fit parameters for the RA(time) propagation.

Type
tuple

dec_poly_param

The polynomial fit parameters for the DEC(time) propagation.

Type
tuple

__fit_polynomial_function(*fit_x: ndarray*[111], *fit_y: ndarray*[112]) → tuple[tuple, tuple]

A wrapper class for fitting the defined specific polynomial function.

Parameters

- **fix_x** (*array-like*) – The x values which shall be fit.
- **fix_y** (*array-like*) – The y values which shall be fit.

Returns

- **fit_param** (*tuple*) – The parameters of the polynomial that corresponded to the best fit. Determined by the order of the polynomial function.
- **fit_error** (*tuple*) – The error on the parameters of the fit.

__init__(*ra: ndarray*[113], *dec: ndarray*[114], *obs_time: ndarray*[115]) → None

Instantiation of the propagation engine.

Parameters

- **ra** (*array-like*) – An array of right ascensions to fit and extrapolate to, must be in degrees.

- **dec** (*array-like*) – An array of declinations to fit and extrapolate to, must be in degrees.
- **obs_time** (*array-like*) – An array of observation times which the RA and DEC measurements were taken at. This should be in Julian days.

Return type
None

__linear_function(*c0: float*, *c1: float*) → ndarray[116]

The linear polynomial function that will be used.

This function is hard coded to be a specific order on purpose. The order may be changed between versions if need be, but should not be changed via configuration.

Parameters

- **x** (*array-like*) – The input for computing the polynomial.
- **c0** (*float*) – Coefficient for order 0.
- **c1** (*float*) – Coefficient for order 1.

Returns
y – The output after computing the polynomial with the provided coefficients.

Return type
array-like

forward_propagate(*future_time: ndarray*[117]) → tuple[numpy.ndarray[118], numpy.ndarray[119]]

Determine a new location(s) based on the polynomial propagation, providing new times to locate in the future.

Parameters
future_time (*array-like*) – The set of future times which to derive new RA and DEC coordinates. The time must be in Julian days.

Returns

- **future_ra** (*ndarray*) – The set of right ascensions that corresponds to the future times, in degrees.
- **future_dec** (*ndarray*) – The set of declinations that corresponds to the future times, in degrees.

[108] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[109] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[110] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[111] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[112] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[113] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[114] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[115] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[116] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[117] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[118] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[119] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

class opihiexarata.propagate.polynomial.**QuadraticPropagationEngine**(*ra: ndarray*[120], *dec: ndarray*[121], *obs_time: ndarray*[122])

Bases: *PropagationEngine*

A simple propagation engine which uses 2nd order extrapolation of RA DEC points independently to determine future location.

ra_array

The array of right ascension measurements to extrapolate to.

Type
ndarray

dec_array

The array of declinations measurements to extrapolate to.

Type
ndarray

obs_time_array

The array of observation times which the RA and DEC measurements were taken at. The values are in Julian days.

Type
ndarray

ra_poly_param

The polynomial fit parameters for the RA(time) propagation.

Type
tuple

dec_poly_param

The polynomial fit parameters for the DEC(time) propagation.

Type
tuple

__fit_polynomial_function(*fit_x: ndarray*[123], *fit_y: ndarray*[124]) → tuple[tuple, tuple]

A wrapper class for fitting the defined specific polynomial function.

Parameters

- **fix_x** (*array-like*) – The x values which shall be fit.
- **fix_y** (*array-like*) – The y values which shall be fit.

Returns

- **fit_param** (*tuple*) – The parameters of the polynomial that corresponded to the best fit. Determined by the order of the polynomial function.
- **fit_error** (*tuple*) – The error on the parameters of the fit.

__init__ (*ra: ndarray*[125], *dec: ndarray*[126], *obs_time: ndarray*[127]) → None

Instantiation of the propagation engine.

Parameters

- **ra** (*array-like*) – An array of right ascensions to fit and extrapolate to, must be in degrees.
- **dec** (*array-like*) – An array of declinations to fit and extrapolate to, must be in degrees.
- **obs_time** (*array-like*) – An array of observation times which the RA and DEC measurements were taken at. This should be in Julian days.

Return type
None

__quadratic_function (*c0: float*, *c1: float*, *c2: float*) → ndarray[128]

The polynomial function that will be used.

This function is hard coded to be a specific order on purpose. The order may be changed between versions if need be, but should not be changed via configuration.

Parameters

- **x** (*array-like*) – The input for computing the polynomial.
- **c0** (*float*) – Coefficient for order 0.
- **c1** (*float*) – Coefficient for order 1.
- **c2** (*float*) – Coefficient for order 2.

Returns
y – The output after computing the polynomial with the provided coefficients.

Return type
array-like

forward_propagate (*future_time: ndarray*[129]) → tuple[numpy.ndarray[130], numpy.ndarray[131]]

Determine a new location(s) based on the polynomial propagation, providing new times to locate in the future.

Parameters
future_time (*array-like*) – The set of future times which to derive new RA and DEC coordinates. The time must be in Julian days.

Returns

- **future_ra** (*ndarray*) – The set of right ascensions that corresponds to the future times, in degrees.

- **future_dec** (*ndarray*) – The set of declinations that corresponds to the future times, in degrees.

opihiexarata.propagate.solution module

The main solution class for propagations.

class opihiexarata.propagate.solution.**PropagativeSolution**(*ra: ndarray*[132], *dec: ndarray*[133], *obs_time: list*, *solver_engine: PropagationEngine*, *vehicle_args: dict = {}*)

Bases: *ExarataSolution*

The general solution class for asteroid propagation.

This uses the recent past location of asteroids to determine their future location. For determination based on orbital elements and ephemerids, use the OrbitalSolution and EphemeriticSolution classes respectively.

ra_array

The array of right ascensions used fit and extrapolate to, in degrees.

Type
array-like

dec_array

The array of declinations used fit and extrapolate to, in degrees.

Type
array-like

obs_time_array

An array of observation times which the RA and DEC measurements were taken at. The values are in Julian days.

Type
array-like

[120] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[121] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[122] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[123] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[124] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[125] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[126] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[127] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[128] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[129] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[130] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[131] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

raw_ra_velocity

The right ascension angular velocity of the target, in degrees per second. These values are derived straight from the data and not the propagation engine.

Type
float

raw_dec_velocity

The declination angular velocity of the target, in degrees per second. These values are derived straight from the data and not the propagation engine.

Type
float

raw_ra_acceleration

The right ascension angular acceleration of the target, in degrees per second squared. These values are derived straight from the data and not the propagation engine.

Type
float

raw_dec_acceleration

The declination angular acceleration of the target, in degrees per second squared. These values are derived straight from the data and not the propagation engine.

Type
float

ra_velocity

The right ascension angular velocity of the target, in degrees per second. These values are derived from the engine.

Type
float

dec_velocity

The declination angular velocity of the target, in degrees per second. These values are derived from the engine.

Type
float

ra_acceleration

The right ascension angular acceleration of the target, in degrees per second squared. These values are derived from the engine.

Type
float

dec_acceleration

The declination angular acceleration of the target, in degrees per second squared. These values are derived from the engine.

Type
float

__init__(*ra: ndarray*[134], *dec: ndarray*[135], *obs_time: list*, *solver_engine:* PropagationEngine, *vehicle_args: dict = {}*)

The instantiation of the propagation solution.

Parameters

- **ra** (*array-like*) – An array of right ascensions to fit and extrapolate to, must be in degrees.
- **dec** (*array-like*) – An array of declinations to fit and extrapolate to, must be in degrees.
- **obs_time** (*array-like*) – An array of observation times which the RA and DEC measurements were taken at. Must be Julian days.
- **solver_engine** (PropagationEngine) – The propagation solver engine class that will be used to compute the propagation solution.
- **vehicle_args** (*dictionary*) – If the vehicle function for the provided solver engine needs extra parameters not otherwise provided by the standard input, they are given here.

Return type
None

__init_compute_propagation_motion(*obs_time_array: ndarray*[136]) → tuple[float, float, float, float]

Compute the raw velocities and accelerations of RA and DEC.

This function prioritizes calculating the raw motion using the most recent observations only.

Parameters
obs_time_array (*array-like*) – An array of observation times which the RA and DEC measurements were taken at. The values are in Julian days.

Returns

- **propagate_ra_velocity** (*float*) – The propagative right ascension angular velocity of the target, in degrees per second.
- **propagate_dec_velocity** (*float*) – The propagative declination angular velocity of the target, in degrees per second.
- **propagate_ra_acceleration** (*float*) – The propagative right ascension angular acceleration of the target, in degrees per second squared. propagation engine.
- **propagate_dec_acceleration** (*float*) – The propagative declination angular acceleration of the target, in degrees per second squared.

__init_compute_raw_motion(*ra_array: ndarray*[137], *dec_array: ndarray*[138], *obs_time_array: ndarray*[139]) → tuple[float, float, float, float]

Compute the raw velocities and accelerations of RA and DEC.

This function prioritizes calculating the raw motion using the most recent observations only.

Parameters

- **ra_array** (*array-like*) – The array of right ascensions used fit and extrapolate to, in degrees.
- **dec_array** (*array-like*) – The array of declinations used fit and extrapolate to, in degrees.
- **obs_time_array** (*array-like*) – An array of observation times which the RA and DEC measurements were taken at. The values are in Julian days.

Returns

- **raw_ra_velocity** (*float*) – The raw right ascension angular velocity of the target, in degrees per second.
- **raw_dec_velocity** (*float*) – The raw declination angular velocity of the target, in degrees per second.
- **raw_ra_acceleration** (*float*) – The raw right ascension angular acceleration of the target, in degrees per second squared. propagation engine.
- **raw_dec_acceleration** (*float*) – The raw declination angular acceleration of the target, in degrees per second squared.

forward_propagate (*future_time: ndarray*[140]) → tuple[numpy.ndarray[141], numpy.ndarray[142]]

A wrapper call around the engine's propagation function. This allows the computation of future positions at a future time using propagation.

Parameters

future_time (*array-like*) – The set of future times which to derive new RA and DEC coordinates. The time must be in Julian days.

Returns

- **future_ra** (*ndarray*) – The set of right ascensions that corresponds to the future times, in degrees.
- **future_dec** (*ndarray*) – The set of declinations that corresponds to the future times, in degrees.

[132] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[133] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[134] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[135] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[136] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[137] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[138] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[139] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[140] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[141] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[142] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

opihiexarata.propagate.solution.**_vehicle_linear_propagation**(*ra_array: ndarray*[143], *dec_array: ndarray*[144], *obs_time_array: ndarray*[145]) → dict

Derive the propagation from 1st order polynomial extrapolation methods.

Parameters

- **ra_array** (*array-like*) – The array of right ascensions used fit and extrapolate to, in degrees.
- **dec_array** (*array-like*) – The array of declinations used fit and extrapolate to, in degrees.
- **obs_time_array** (*array-like*) – An array of observation times which the RA and DEC measurements were taken at. The values are in Julian days.

Returns

solution_results – The results of the propagation engine which then gets integrated into the solution.

Return type

dictionary

opihiexarata.propagate.solution.**_vehicle_quadratic_propagation**(*ra_array: ndarray*[146], *dec_array: ndarray*[147], *obs_time_array: ndarray*[148]) → dict

Derive the propagation from 2nd order polynomial extrapolation methods.

Parameters

- **ra_array** (*array-like*) – The array of right ascensions used fit and extrapolate to, in degrees.
- **dec_array** (*array-like*) – The array of declinations used fit and extrapolate to, in degrees.
- **obs_time_array** (*array-like*) – An array of observation times which the RA and DEC measurements were taken at. The values are in Julian days.

[143] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[144] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[145] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

Returns
solution_results – The results of the propagation engine which then gets integrated into the solution.

Return type
dictionary

Module contents

Solution methods and engines for asteroid propagation.

Submodules

opihiexarata.__main__ module

Just a small hook for the main execution. This section parses arguments which is then passed to execution to do exactly as expected by the commands.

The actual execution is done in the command.py file so that this file does not get too large.

opihiexarata.__main__.**__main_execute_arguments**(*parser: ArgumentParser*, *arguments: Namespace*) → None

We actually execute the software using the arguments provided in the # command line call. GUI's are started on separate threads

Parameters
arguments (*dict*) – The parsed arguments from which the interpreted action will use. Note though that these arguments also has the interpreted actions.

Return type
None

opihiexarata.__main__.**__main_parse_arguments**() → tuple[argparse.ArgumentParser, argparse.Namespace]

The main section for argument parsing. We just have it here to better organize things.

Parameters
None –

Returns

- **parser** (*ArgumentParser*) – The parser itself. This may not be needed for a lot of things, but it is still helpful for command processing.
- **parsed_arguments** (*Namespace*) – The arguments as parsed by the parser. Though technically it is a Namespace class, it is practically a dictionary.

[146] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[147] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray
[148] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray

opihiexarata.__main__.**main**() → None

The main command for argument parsing and figuring out what to do based on command-line entries.

Parameters

None –

Return type

None

Module contents

All of the subparts of the OpihiExarata software.

- genindex
- modindex
- search

PYTHON MODULE INDEX

O

INDEX

Non-alphabetical

A

B

C

D

E

F

G

H

I

J

K

L

M

O

P

Q

R

S

T

U

W

Z