Exoplanet Open-Source Imaging Mission Simulator (EXOSIMS) Interface Control Document

Dean Keithly, Christian Delacroix, Daniel Garrett, and Dmitry Savransky
Sibley School of Mechanical and Aerospace Engineering
Cornell University
Ithaca, NY 14853

ABSTRACT

This document describes the extensible, modular, open source software framework EXOSIMS. EXOSIMS creates end-to-end simulations of space-based exoplanet imaging missions using stand-alone software modules. The input/output interfaces of each module and interactions of modules with each other are presented to give guidance on mission specific modifications to the EXOSIMS framework. Last Update: November 14, 2018

CONTENTS

Nomenclature

EXOSIMS	Exoplanet Open-Source Imaging Mission Simulator
ICD	Interface Control Document
MJD	Modified Julian Day

1 Introduction

Building confidence in a mission concept's ability to achieve its science goals is always desirable. Unfortunately, accurately modeling the science yield of an exoplanet imager can be almost as complicated as designing the mission. It is challenging to compare science simulation results and systematically test the effects of changing one aspect of the instrument or mission design.

EXOSIMS (Exoplanet Open-Source Imaging Mission Simulator) addresses this problem by generating ensembles of mission simulations for exoplanet direct imaging missions to estimate science yields. It is designed to allow systematic exploration of exoplanet imaging mission science yields. It consists of stand-alone modules written in Python which may be modified without requiring modifications to other portions of the code. This allows EXOSIMS to be easily used to investigate new designs for instruments, observatories, or overall mission designs independently. This document describes the required input/output interfaces for the stand-alone modules to enable this flexibility.

1.1 Purpose and Scope

This Interface Control Document (ICD) provides an overview of the software framework of EXOSIMS and some details on its component parts. As the software is intended to be highly reconfigurable, operational aspects of the code are emphasized over implementational details. Specific examples are taken from the coronagraphic instrument under development for WFIRST. The data inputs and outputs of each module are described. Following these guidelines will allow the code to be updated to accommodate new mission designs.

This ICD defines the input/output of each module and the interfaces between modules of the code. This document is intended to guide mission planners and instrument designers in the development of specific modules for new mission designs.

2 Overview

The terminology used to describe the software implementation is loosely based upon object-oriented programing (OOP) terminology, as implemented by the Python language, in which EXOSIMS is built. The term module can refer to the object class prototype representing the abstracted functionality of one piece of the software, an implementation of this object class which inherits the attributes and methods of the prototype, or an instance of this class. Input/output definitions of modules refer to the class prototype. Implemented modules refer to the inherited class definition. Passing modules (or their outputs) means the instantiation of the inherited object class being used in a given simulation. Relying on strict inheritance for all implemented module classes provides an automated error and consistency-checking mechanism. The outputs of a given object instance may be compared to the outputs of the prototype. It is trivial to pre-check whether a given module implementation will work within the larger framework, and this approach allows for flexibility and adaptability.

The overall framework of EXOSIMS is depicted in Figure 1, which shows all of the component software modules in the order in which they are instantiated in normal operation. Standard EXOSIMS modules are:

Optical System
Star Catalog
Planet Population
Observatory
Planet Physical Model
Time Keeping
Zodiacal Light
Background Sources
Post-Processing
Target List
Simulated Universe
Survey Simulation
Survey Ensemble

Objects of all module classes can be instantiated independently, although most modules require the instantiation of other modules during their construction (see Figure 1). Different implementations of the modules contain specific mission design parameters and physical descriptions of the universe, and will change according to mission and planet population of interest. The upstream modules (including Target List, Simulated Universe, Survey Simulation, and Survey Ensemble modules) take information contained in the downstream modules and perform mission simulation tasks. The instantiation of an object of any of these modules requires the instantiation of one or more downstream module objects. Any module may perform any number or kind of calculations using any or all of the input parameters provided. The specific implementations are only constrained by their input and output specification contained in this document.

Figures 2 and 3 show schematic representations of the three different aspects of a module, using the Star Catalog and Observatory modules as examples, respectively. Every module has a specific prototype that sets the input/output structure of the module and encodes any common functionality for all module class implementations. The various implementations inherit the prototype and add/overload any attributes and methods required for their particular tasks, limited only by the preset input/output scheme. Finally, in the course of running a simulation, an object is generated for each module class selected for that simulation. The generated objects can be used in exactly the same way in the downstream code, regardless of what implementation they are instances of, due to the strict interface defined in the class prototypes.

For lower level (downstream) modules, the input specification is much more loosely defined than the output specification, as different implementations may draw data from a wide variety of sources. For example, the star catalog may be

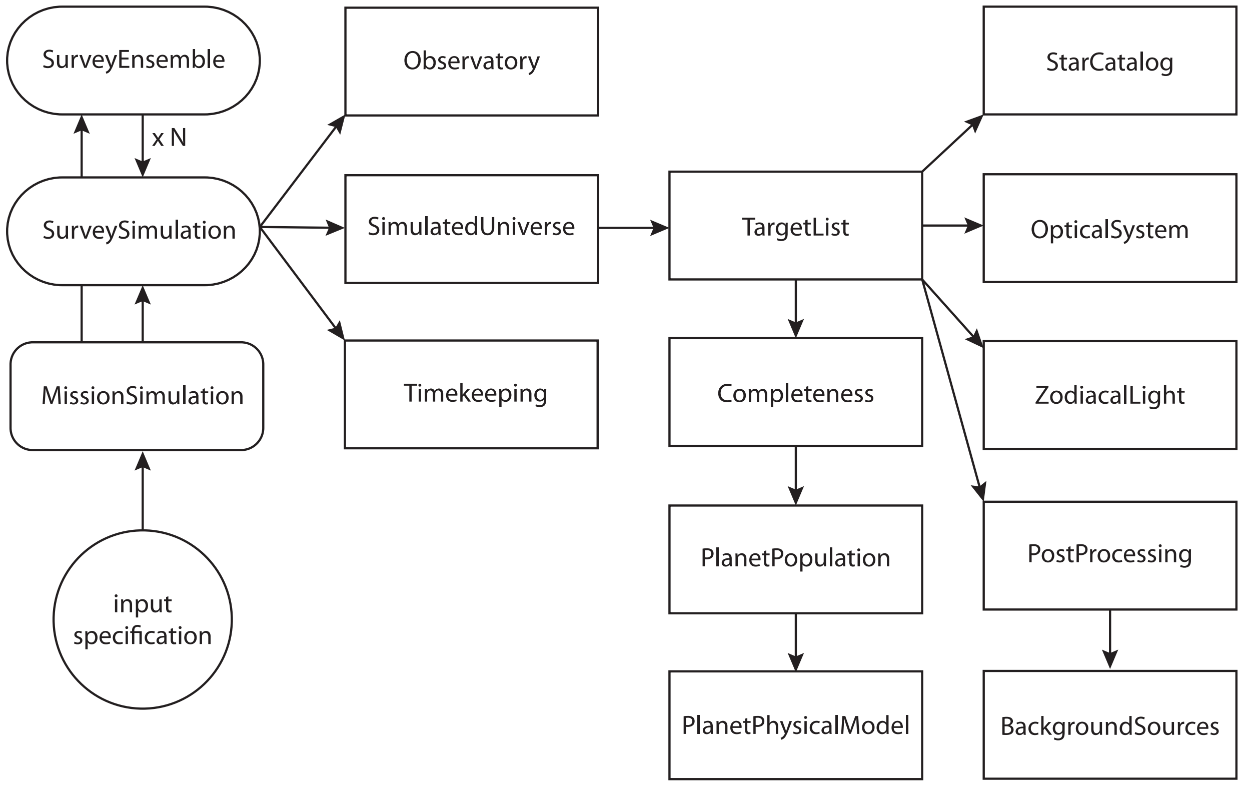

Fig. 1. Schematic depiction of the instantiation path of all EXOSIMS modules. The arrows indicate calls to the object constructor, and object references to each module are always passed up directly to the top calling module, so that a given module has access to any other module connected to it by a direct path of instantiations. For example, the TargetList module has access to both the PostProcessing and BackgroundSources modules, while the Observatory module does not have access to any other modules. The typical entry point to EXOSIMS is the construction of a MissionSim object, which causes the instantiation of the SurveySimulation module, which in turn instnatiates all the other modules. In the case of a parallelize SurveyEnsemble instnatiation, multiple, independent SurveySimulation modules are instantiated at the same time. At the end of construction, the MissionSim and SurveySimulation objects have direct access to all other modules as their attributes.

implemented as reading values from a static file on disk, or may represent an active connection to a local or remote database. The output specification for these modules, however, as well as both the input and output for the upstream modules, is entirely fixed so as to allow for generic use of all module objects in the simulation.

3 Global Specifications

Common references (units, frames of reference, etc.) are required to ensure interoperability between the modules of EXOSIM. All of the references listed below must be followed.

Common Epoch
J2000
Common Reference Frame
Heliocentric Equatorial (HE)

3.1 Python Packages

EXOSIMS is an open source platform. As such, packages and modules may be imported and used for calculations within any of the stand-alone modules. The following commonly used Python packages are used for the WFIRST-specific implementation of EXOSIMS:

```
astropy
```

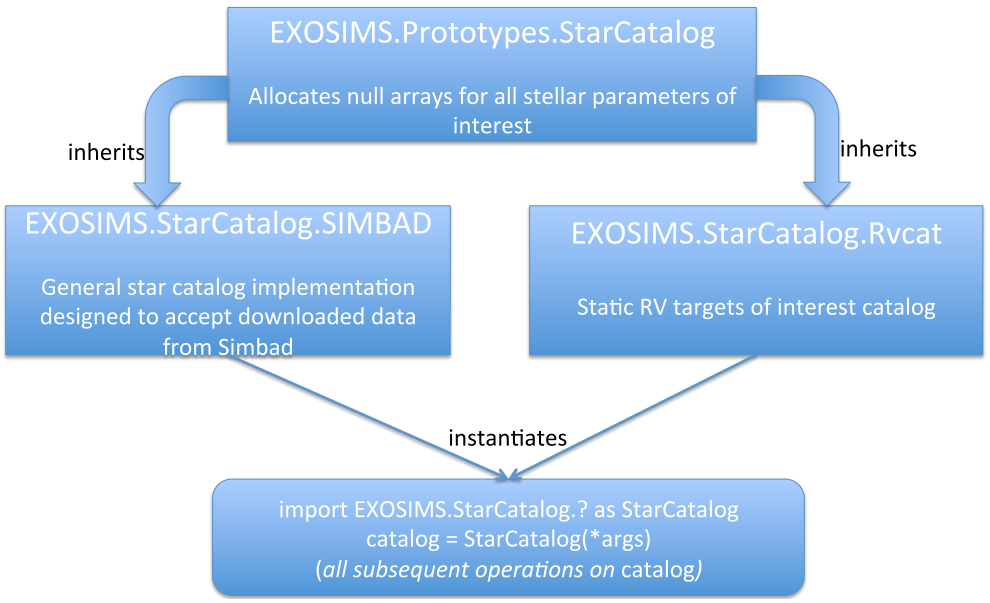
EXOSIMS.Prototypes.StarCatalog
Allocates null arrays for all stellar parameters of interest
inherits
inherits
EXOSIMS.StarCatalog.SIMBAD
General star catalog implementation designed to accept downloaded data from Simbad
EXOSIMS.StarCatalog.Rvcat
Static RV targets of interest catalog
instantiates
import EXOSIMS.StarCatalog.? as StarCatalog
catalog = StarCatalog(*args)
(all subsequent operations on catalog)

Fig. 2. Schematic of a sample implementation for the three module layers for the Star Catalog module. The Star Catalog prototype (top row) is immutable, specifies the input/output structure of the module along with all common functionality, and is inherited by all Star Catalog class implementations (middle row). In this case, two different catalog classes are shown: one that reads in data from a SIMBAD catalog dump, and one which contains only information about a subset of known radial velocity targets. The object used in the simulation (bottom row) is an instance of one of these classes, and can be used in exactly the same way in the rest of the code due to the common input/output scheme.

```
    astropy.constants
    astropy.coordinates
    astropy.time
    astropy.units
copy
hashlib
importlib
numpy
    numpy.linalg
os
    os.path
pickle/cPickle
scipy
    scipy.io
    scipy.special
    scipy.interpolate
sys
logging
time
json
random
re
inspect
subprocess
csv
h5py (optional)
jplephem (optional)
```

Additionally, while not required for running the survey simulation, `matplotlib` is used for visualization of the results.

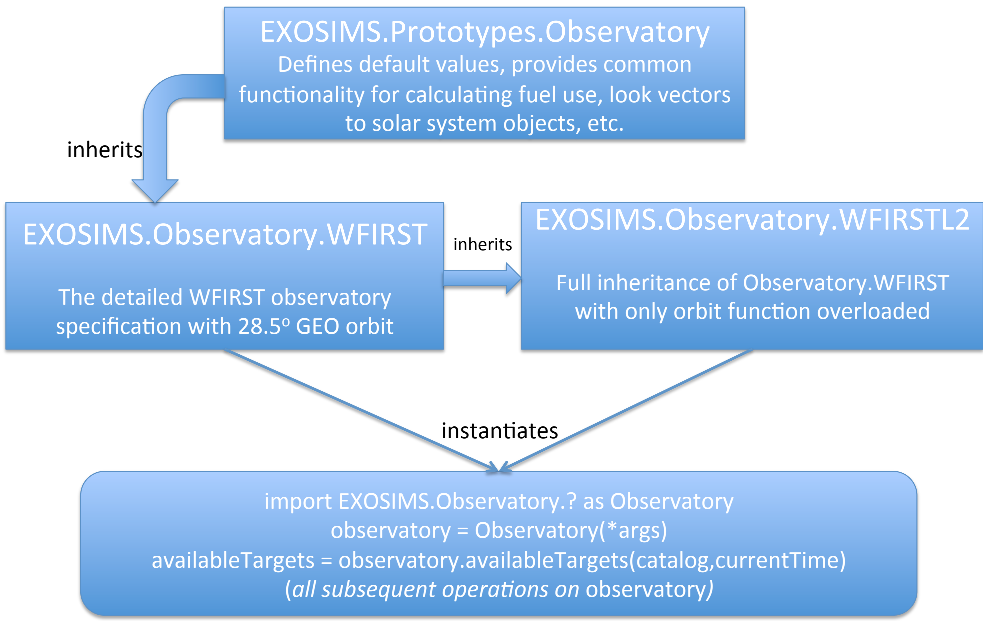
EXOSIMS.Prototypes.Observatory
Defines default values, provides common functionality for calculating fuel use, look vectors to solar system objects, etc.
inherits
EXOSIMS.Observatory.WFIRST
The detailed WFIRST observatory specification with 28.5° GEO orbit
inherits
EXOSIMS.Observatory.WFIRSTL2
Full inheritance of Observatory.WFIRST with only orbit function overloaded
instantiates
import EXOSIMS.Observatory.? as Observatory
observatory = Observatory(*args)
availableTargets = observatory.availableTargets(catalog,currentTime)
(all subsequent operations on observatory)

Fig. 3. Schematic of a sample implementation for the three module layers for the Observatory module. The Observatory prototype (top row) is immutable, specifies the input/output structure of the module along with all common functionality, and is inherited by all Observatory class implementations (middle row). In this case, two different observatory classes are shown that differ only in the definition of the observatory orbit. Therefore, the second implementation inherits the first (rather than directly inheriting the prototype) and overloads only the orbit method. The object used in the simulation (bottom row) is an instance of one of these classes, and can be used in exactly the same way in the rest of the code due to the common input/output scheme.

3.2 Coding Conventions

In order to allow for flexibility in using alternate or user-generated module implementations, the only requirement on any module is that it inherits (either directly or by inheriting another module implementation that inherits the prototype) the appropriate prototype. It is similarly expected that the prototype constructor will be called from the constructor of the newly implemented class. An example of an Optical System module implementation follows:

```
from EXOSIMS.Prototypes.OpticalSystem import OpticalSystem

class ExampleOpticalSystem(OpticalSystem):

    def __init__(self, **specs):

        OpticalSystem.__init__(self, **specs)

        ...
```

Note that the filename must match the class name for all modules.

3.2.1 Module Type

It is always possible to check whether a module is an instance of a given prototype, for example:

```
isinstance(obj,EXOSIMS.Prototypes.Observatory.Observatory)
```

However, it can be tedious to look up all of a given object's base classes so, for convenience, every prototype will provide a private variable `_modtype`, which will always return the name of the prototype and should not be overwritten by any module code. Thus, if the above example evaluates as `True`, `obj._modtype` will return `Observatory`.

3.2.2 Callable Attributes

Certain module attributes must be represented in a way that allows them to be parametrized by other values. For example, the instrument throughput and contrast are functions of both the wavelength and the angular separation, and so must be encodable as such in the optical system module. To accommodate this, as well as simpler descriptions where these parameters may be treated as static values, these and other attributes are defined as 'callable'. This means that they must be set as objects that can be called in the normal Python fashion, i.e., `object(arg1,arg2,...)`.

These objects can be function definitions defined in the code, or imported from other modules. They can be lambda expressions defined inline in the code. Or they can be callable object instances, such as the various scipy interpolants. In cases where the description is just a single value, these attributes can be defined as dummy functions that always return the same value, for example:

```
def throughput(wavelength,angle):
     return 0.5
```

or even more simply:

```
throughput = lambda wavelength,angle: 0.5
```

Note, however, that Python differentiates between how it treats class attributes and methods in inheritance. If a value is originally defined as an attribute (such as a lambda function), then it cannot be overloaded by a method in an inheriting class implementation. So, if a prototype contains a callable value as an attribute, it must be implemented as an attribute in all inheriting implementations that wish to change the value. For this reason, the majority of callable attributes in prototype modules are instead defined as methods to avoid potential overloading issues.

4 Survey Simulation (Backbone)

By default, the simulation execution will be performed via the SurveySimulation module. This will consist of a limited set of functions that will primarily be tasked with parsing the input specification described below, and then creating the specified instances of each of the framework modules, detailed in §5. For convenience, there is a higher-level MissionSim class, whose constructor will take the input script file (§4.1) and generate instances of all module objects, including the SurveySimulation (§5.13) and SurveyEnsemble modules, which will contain the functions to run the survey simulations. However, for mission ensembles, the base object created on all workers is SurveySimulation, so that a MissionSim object will exist only on the controller node of the cluster. Any mission-specific execution variations will be introduced by method overloading in the inherited SurveySimulation implementation. Figure 1 provides a graphical description of the instantiation order of all module objects.

A simulation specification is a single JSON-formatted (`http://json.org/`) file that encodes user-settable parameters and module names. SurveySimulation will contain a reference specification with *all* parameters and modules set via defaults in the constructors of each of the modules. In the initial parsing of the user-supplied specification, it will be merged with the reference specification such that any fields not set by the user will be assigned to their reference (default) values. Each instantiated module object will contain a dictionary called `_outspec`, which, taken together, will form the full specification for the current run (as defined by the loaded modules). This specification will be written out to a json file associated with the output of every run. *Any specification added by a user implementation of any module must also be added to the _outspec dictionary*. The assembly of the full output specification is provided by MissionSim method `genOutSpec`.

For every simulation (or ensemble), an output specification will be written to disk along with the simulation results with all defaults used filled in.

4.1 Specification Format

The JSON specification file will contain a series of objects with members enumerating various user-settable parameters, top-level members for universal settings (such as the mission lifetime) and arrays of objects for multiple related specifications, such as starlight suppression systems and science instruments. The specification file must contain a `modules` dictionary listing the module names (or paths on disk to user-implemented classes) for all modules.

```
{
    "FAP": 3e-07,
    "FAdMag0": 15,
    "IWA": 0.15,
    "Irange": [
        0.0,
        180.0
    ],
    "MDP": 0.001,
    "Mprange": [
        1.0,
        4131.0
    ],
    "OWA": 0.557002,
    "Orange": [
        0.0,
        360.0
    ],
    "Rprange": [
        1.0,
        22.6
    ],
    "WA0": 0.289498,
    "WAint": 0.3,
    "arange": [
        0.1,
        100.0
    ],
    "cachedir": "path/to/desired/cache/directory",
    "charMargin": 0.15,
    "checkKeepoutEnd": true,
    "coMass": 5800.0,
    "constrainOrbits": false,
    "dMag0": 22.5,
    "dMagLim": 22.5,
    "dMagint": 22.5,
    "erange": [
        0.01,
        0.99
    ],
    "esigma": 0.25,
    "forceStaticEphem": false,
    "havejplephem": true,
    "intCutoff": 15.0,
    "keepStarCatalog": false,
    "koAngleMax": 90.0,
    "koAngleMin": 45.0,
    "koAngleMinEarth": 45.0,
    "koAngleMinMoon": 45.0,
    "koAngleSmall": 1.0,
    "magEZ": 22.0,
    "magZ": 23.0,
    "minComp": 0.0,
    "missionLife": 5.0,
    "missionPortion": 0.0493150685,
    "missionStart": 60634.0,
    "OBduration": 14,
    "missionSchedule": sampleOB1.csv,
    "modules": {
        "BackgroundSources": "BackgroundSources",
        "Completeness": "GarrettCompleteness",
        "Observatory": "WFIRSTObservatoryL2",
        "OpticalSystem": "Nemati",
        "PlanetPhysicalModel": "Forecaster",
        "PlanetPopulation": "KeplerLike2",
        "PostProcessing": "PostProcessing",
        "SimulatedUniverse": "KeplerLikeUniverse",
        "StarCatalog": "EXOCAT1",
        "SurveyEnsemble": "SurveyEnsemble",
        "SurveySimulation": "SurveySimulation",
        "TargetList": "TargetList",
        "TimeKeeping": "TimeKeeping",
        "ZodiacalLight": "Stark"
    },
    "nVisitsMax": 5,
    "ntFlux": 1,
    "obscurFac": 0.1724,
    "observingModes": [
        {
            "SNR": 5,
            "detectionMode": true,
            "instName": "imager",
            "radDos": 0.5,
            "systName": "HLC-565"
        },
        {
            "SNR": 10,
            "instName": "spectro",
            "lam": 660,
            "radDos": 1.0,
            "systName": "SPC-660"
        }
    ],
    "occulterSep": 55000.0,
    "ppFact": 0.1,
    "prange": [
        0.083,
        0.882
    ],
    "pupilDiam": 2.37,
    "ref_Time": 0.2,
    "ref_dMag": 3.0,
    "scaleOrbits": false,
    "scienceInstruments": [
        {
            "CIC": 0.01,
            "ENF": 1.0,
            "FoV": 9.5,
            "PCeff": 0.8,
            "QE": "$HOME/Data/QEfile.fits",
            "Rs": 1.0,
            "fnumber": 60.97706197560175,
            "focal": 144.51563688217615,
            "idark": 0.000114,
            "lenslSamp": 1.0,
            "name": "imager",
            "optics": 0.518018590965876,
            "pixelNumber": 1024,
            "pixelScale": 0.0185546875,
            "pixelSize": 1.3e-05,
            "sread": 0.0,
            "texp": 100.0
        },
        {
            "CIC": 0.01,
            "ENF": 1.0,
            "FoV": 1.0,
            "PCeff": 0.8,
            "QE": "$HOME/Data/QEfile.fits",
            "Rs": 50.0,
            "fnumber": 575.4526999602537,
            "focal": 1363.8228989058011,
            "idark": 0.000114,
            "lenslSamp": 2.0,
            "name": "spectro",
            "optics": 0.465846901959329,
            "pixelNumber": 76,
            "pixelScale": 0.02631578947368421,
            "pixelSize": 0.000174,
            "sread": 0.0,
            "texp": 100.0
        }
    ],
    "settlingTime": 0.5,
    "shapeFac": 0.7853981633974483,
    "smaknee": 30.0,
    "starlightSuppressionSystems": [
        {
            "BW": 0.1,
            "IWA": 0.15,
            "OWA": 0.428996,
            "core_area": "$HOME/Data/area.fits",
            "core_contrast": 1e-10,
            "core_mean_intensity": "$HOME/Data/mean_intensity.fits",
            "core_platescale": 0.3,
            "core_thruput": "$HOME/Data/thruput.fits",
            "deltaLam": 56.5,
            "lam": 565.0,
            "name": "HLC-565",
            "occ_trans": "$HOME/Data/occ_trans.fits",
            "occulter": false,
            "ohTime": 0.5,
            "optics": 0.983647,
            "samp": 10.0
        },
        {
            "BW": 0.18,
            "IWA": 0.208876,
            "OWA": 0.557002,
            "core_area": "$HOME/Data/area.fits",
            "core_contrast": 1e-10,
            "core_mean_intensity": "$HOME/Data/mean_intensity.fits",
            "core_platescale": 0.3,
            "core_thruput": "$HOME/Data/thruput.fits",
            "deltaLam": 118.8,
            "lam": 660.0,
            "name": "SPC-660",
            "occ_trans": "$HOME/Data/occ_trans.fits",
            "occulter": false,
            "ohTime": 0.5,
            "optics": 0.9154706,
            "samp": 10.0
        }
    ],
    "staticStars": true,
    "waitMultiple": 2.0,
    "waitTime": 1.0,
    "wrange": [
        0.0,
        360.0
    ]
}
```

4.2 Modules Specification

The final array in the input specification (`modules`) is a list of all the modules that define a particular simulation. This is the only part of the specification that will not be filled in by default if a value is missing - each module must be explicitly specified. The order of the modules in the list is arbitrary, so long as they are all present.

If the module implementations are in the appropriate subfolder in the EXOSIMS tree, then they can be specified by the module name. However, if you wish to use an implemented module outside of the EXOSIMS directory, then you need to specify it via its full path in the input specification.

All modules, regardless of where they are stored on disk must inherit the appropriate prototype.

4.2.1 Different Planet Population and Completeness Distributions

EXOSIMS allows for the calculation of completeness using a different planet population from the one used to populate the SimulatedUniverse. This functionality is intended to simulate the effects of our current lack of knowledge of the 'true' planet population. The functionality is enabled by adding an optional `completeness_specs` dictionary to the JSON script. Below is a simple example where the Completeness is based off KeplerLike2 and Planet Population is based off JupiterTwin.

```
{
  "completeness_specs":{
    "eta":1,
    "modules":{
      "PlanetPopulation": "JupiterTwin",
      "PlanetPhyiscalModel":" "
  },
...
  "modules": {
    "PlanetPopulation": "KeplerLike2",
    "StarCatalog": "EXOCAT1",
    "OpticalSystem": "Nemati",
    "ZodiacalLight": "Stark",
    "BackgroundSources": " ",
    "PlanetPhysicalModel": "Forecaster",
    "Observatory": "WFIRSTObservatoryL2",
    "TimeKeeping": " ",
    "PostProcessing": " ",
    "Completeness": "BrownCompleteness",
    "TargetList": " ",
    "SimulatedUniverse": "KeplerLikeUniverse",
    "SurveySimulation": "SLSQPScheduler",
    "SurveyEnsemble": " "
  }
}
```

In this example, the SimulatedUniverse will be populated using the KeplerLike2 PlanetPopulation with the Forecaster PhysicalModel, while the completeness will be calculated based on the JupiterTwin PlanetPopulation and the Prototype PhyscialModel. Note also that the JupiterTwin PlanetPopulation will be passed a constructor inpu of `eta=1`, whereas the KeplerLike2 constructor would not get this input, unless it was separately defined elsewhere in the JSON script. In the case where the `completeness_specs` dictionary is ommited, all calcluations would be based on the same PlanetPopulation instance, as specified in the primary modules dictionary.

4.3 Universal Parameters

These parameters apply to all simulations. These parameters define the scope of values passable for module initialization. These parameters are described in detail in their specific module definitions.

MissionSim	
`scriptfile`	(string) name of the scriptfile to load json script parameters from.
`nopar`	(boolean) If True, ignore any provided ensemble module in the script or specs and force the prototype ensemble.
`verbose`	(boolean) Boolean used to create the vprint function, equivalent to the python print function with an extra verbose toggle parameter (True by default). The vprint function can be accessed by all modules from EXOSIMS.util.vprint.
PlanetPopulation	
`arange`	(float) 1×2 list of semi-major axis range in units of *AU*.
`erange`	(float) 1×2 list of eccentricity range.
`Irange`	(float) 1×2 list of inclination range in units of *deg*.
`Orange`	(float) 1×2 list of ascension of the ascending node range in units of *deg*.
`wrange`	(float) 1×2 list of argument of perigee range in units of *deg*.
`prange`	(float) 1×2 list of planetary geometric albedo range.
`Rprange`	(float) 1×2 list of planetary radius range in Earth radii.
`Mprange`	(float) 1×2 list of planetary mass range in Earth masses.
`scaleOrbits`	(boolean) True means planetary orbits are scaled by the square root of stellar luminosity.
`constrainOrbits`	(boolean) True means planetary orbits are constrained to never leave the semi-major axis range (arange).
`eta`	(float) The average occurrence rate of planets per star for the entire population.
OpticalSystem	
`obscurFac`	(float) Obscuration factor due to secondary mirror and spiders.
`shapeFac`	(float) Telescope aperture shape factor.
`pupilDiam`	(float) Entrance pupil diameter in units of *m*.
`intCutoff`	(float) Maximum allowed integration time in units of *day*.
`dMag0`	(float) Favorable planet delta magnitude value used to calculate the minimum integration times for inclusion in target list.
`WA0`	(float) Instrument working angle value used to calculate the minimum integration times for inclusion in target list, in units of *arcsec*.
`scienceInstruments`	(list of dicts) Contains specific attributes of all science instruments.
`QE`	(float, callable) Detector quantum efficiency: either a scalar for constant QE, or a two-column array for wavelength-dependent QE, where the first column contains the wavelengths in units of nm. May be data or FITS filename.
`optics`	(float) Attenuation due to optics specific to the science instrument.
`FoV`	(float) Field of view in units of arcsec
`pixelNumber`	(integer) Detector array format, number of pixels per detector lines/columns
`pixelSize`	(float) Detector pixel scale in units of arcsec per pixel
`sread`	(float) Detector effective read noise per frame per pixel
`idark`	(float) Detector dark-current per pixel in units of 1/s
`CIC`	(float) Clock-induced-charge per frame per pixel
`texp`	(float) Exposure time per frame in units of s
`radDos`	(float) Radiation dosage
`PCeff`	(float) Photon counting efficiency
`ENF`	(float) (Specific to EM-CCDs) Excess noise factor
`Rs`	(float) (Specific to spectrometers) Spectral resolving power
`lenslSamp`	(float) (Specific to spectrometers) Lenslet sampling, number of pixel per lenslet rows or cols
`starlightSuppressionSystems`	(list of dicts) All starlight suppression system attributes (variable)
`lam`	(integer) Central wavelength in units of nm
`BW`	(float) Bandwidth fraction
`occ_trans`	(float, callable) Intensity transmission of extended background sources such as zodiacal light. Includes pupil mask, occulter, Lyot stop and polarizer.
`core_thruput`	(float, callable) System throughput in the FWHM region of the planet PSF core.
`core_contrast`	(float, callable) System contrast = mean_intensity / PSF_peak
`core_platescale`	(float) Platescale used for a specific set of coronagraph parameters, in units of lambda/D per pixel

`PSF`	(float, callable) Point spread function - 2D ndarray of values, normalized to 1 at the core. Note: normalization means that all throughput effects must be contained in the throughput attribute.
`ohTime`	(float) Overhead time in units of days
`observingModes`	(list of dicts) Mission observing modes attributes
`SNR`	(integer) Signal-to-noise ratio threshold
`timeMultiplier`	(integer) Integration time multiplier
`IWA`	(float) Fundamental Inner Working Angle in units of *arcsec*. No planets can ever be observed at smaller separations.
`OWA`	(float) Fundamental Outer Working Angle in units of *arcsec*. Set to Inf for no OWA. JSON values of 0 will be interpreted as Inf.
`ref_dMag`	(float) reference star dMag for RDI
`ref_Time`	(float) fraction of time spent on ref star for RDI
`ZodiacalLight`	
`magZ`	(float) 1 zodi brightness magnitude (per arcsec2).
`magEZ`	(float) 1 exo-zodi brightness magnitude (per arcsec2).
`varEZ`	(float) exo-zodiacal light variation (variance of log-normal distribution).
`PostProcessing`	
`FAP`	(float) False Alarm Probability.
`MDP`	(float) Missed Detection Probability.
`ppFact`	(float, callable) Post-processing contrast factor, between 0 and 1.
`FAdMag0`	(float, callable) Minimum delta magnitude that can be obtained by a false alarm.
`Completeness`	
`dMagLim`	(float) Limiting planet-to-star delta magnitude for completeness.
`minComp`	(float) Minimum completeness value for inclusion in target list.
`TargetList`	
`staticStars`	(boolean) Boolean used to force static target positions set at mission start time.
`keepStarCatalog`	(boolean) Boolean representing whether to delete the star catalog after assembling the target list. If true, object reference will be available from TargetList object.
`Observatory`	
`koAngleMin`	(float) Telescope minimum keepout angle in units of *deg*.
`koAngleMinMoon`	(float) Telescope minimum keepout angle in units of *deg*, for the Moon only.
`koAngleMinEarth`	(float) Telescope minimum keepout angle in units of *deg*, for the Earth only.
`koAngleMax`	(float) Telescope maximum keepout angle (for occulter) in units of *deg*.
`koAngleSmall`	(float) Telescope keepout angle for smaller (angular size) bodies in units of *deg*.
`settlingTime`	(float) Amount of time needed for observatory to settle after a repointing in units of *day*.
`thrust`	(float) Occulter slew thrust in units of *mN*.
`slewIsp`	(float) Occulter slew specific impulse in units of *s*.
`scMass`	(float) Occulter (maneuvering spacecraft) initial wet mass in units of *kg*.
`dryMass`	(float) Occulter (maneuvering spacecraft) dry mass in units of *kg*.
`coMass`	(float) Telescope (or non-maneuvering spacecraft) mass in units of *kg*.
`occulterSep`	(float) Occulter-telescope distance in units of *km*.
`skIsp`	(float) Specific impulse for station keeping in units of *s*.
`defburnPortion`	(float) Default burn portion for slewing.
`constTOF`	(float) Constant time of flight for single occulter slew in units of *day*
`maxdVpct`	(float) Maximum percentage of total on board fuel used for single starshade slew
`spkpath`	(string) Full path to SPK kernel file.
`checkKeepoutEnd`	(boolean) Boolean signifying if the keepout method must be called at the end of each observation.
`forceStaticEphem`	(boolean) Force use of static solar system ephemeris if set to True, even if jplephem module is present.
`TimeKeeping`	
`missionStart`	(float) Mission start time in *MJD*.
`missionLife`	(float) The total mission lifetime in units of *year*. When the mission time is equal or greater to this value, the mission simulation stops.
`missionPortion`	(float) The portion of the mission dedicated to exoplanet science, given as a value between 0 and 1. The mission simulation stops when the total integration time plus observation overhead time is equal to the missionLife $\times$ missionPortion.
`OBduration`	(float) Default allocated duration of observing blocks, in units of *day*. If no OBduration was specified, a new observing block is created for each new observation in the SurveySimulation

	module.
`missionSchedule`	(string) filename of csv file containing Observing Block start and end times.
`SurveySimulation`	
`scriptfile`	(string) name of the scriptfile to load json script parameters from
`ntFlux`	(integer) Observation time sampling, to determine the integration time interval.
`nVisitsMax`	(integer) Maximum number of observations (in detection mode) per star.
`charMargin`	(float) Integration time margin for characterization.
`WAint`	(float) Working angle used for integration time calculation in units of *arcsec*.
`dMagint`	(float) Delta magnitude used for integration time calculation.
`dt_max`	(float) The maximum time for a revisit window in units of weeks.
`cachedir`	(string) path to desired cache directory (default is `$HOME/.EXOSIMS/cache`)

5 Prototype Module Specifications

The lower level modules include Planet Population, Star Catalog, Optical System, Zodiacal Light, Background Sources, Planet Physical Model, Observatory, Time Keeping, and Post-Processing. These modules encode and/or generate all of the information necessary to perform mission simulations. The specific mission design determines the functionality of each module, while inputs and outputs of these modules remain the same (in terms of data type and variable representations).

The upstream modules include Completeness, Target List, Simulated Universe, Survey Simulation and Survey Ensemble. These modules perform methods which require inputs from one or more downstream modules as well as calling function implementations in other upstream modules.

This section defines the functionality, major tasks, input, output, and interface of each of these modules. Every module constructor must always accept a keyword dictionary (`**spec`) representing the contents of the specification JSON file organized into a Python dictionary. The descriptions below list out specific keywords that are pulled out by the prototype constructors of each of the modules, but implemented constructors may include additional keywords (so long as they correctly call the prototype constructor). In all cases, if a given `key:value` pair is missing from the dictionary, the appropriate object attributes will be assigned the default values listed.

5.1 Star Catalog

The Star Catalog module includes detailed information about potential target stars drawn from general databases such as SIMBAD, mission catalogs such as Hipparcos, or from existing curated lists specifically designed for exoplanet imaging missions. Information to be stored, or accessed by this module will include target positions and proper motions at the reference epoch, catalog identifiers (for later cross-referencing), bolometric luminosities, stellar masses, and magnitudes in standard observing bands. Where direct measurements of any value are not available, values are synthesized from ancillary data and empirical relationships, such as color relationships and mass-luminosity relations.

This module does not provide any functionality for picking the specific targets to be observed in any one simulation, nor even for culling targets from the input lists where no observations of a planet could take place. This is done in the Target List module as it requires interactions with the Planet Population (to determine the population of interest), Optical System (to define the capabilities of the instrument), and Observatory (to determine if the view of the target is unobstructed) modules.

5.1.1 Star Catalog Object Attribute Initialization

The Star Catalog prototype creates empty 1D NumPy ndarrays for each of the output quantities listed below. Specific Star Catalog modules must populate the values as appropriate. Note that values that are left unpopulated by the implementation will still get all zero array, which may lead to unexpected behavior.

Input

star catalog information

Information from an external star catalog (left deliberately vague as these can be anything).

Attributes

ntargs (integer)

Number of stars

Name (string ndarray)

Star names

Spec (string ndarray)

Spectral types

Umag (float ndarray)
U magnitude

Bmag (float ndarray)
B magnitude

Vmag (float ndarray)
V magnitude

Rmag (float ndarray)
R magnitude

Imag (float ndarray)
I magnitude

Jmag (float ndarray)
J magnitude

Hmag (float ndarray)
H magnitude

Kmag (float ndarray)
K magnitude

BV (float ndarray)
B-V Johnson magnitude

MV (float ndarray)
Absolute V magnitude

BC (float ndarray)
Bolometric correction

L (float ndarray)
Stellar luminosity in Solar luminosities

Binary_Cut (boolean ndarray)
Booleans where True is a star with a companion closer than $10 arcsec$

dist (astropy Quantity array)
Distance to star in units of pc. Defaults to 1.

parx (astropy Quantity array)
Parallax in units of mas. Defaults to 1000.

coords (astropy SkyCoord array)
SkyCoord object containing right ascension, declination, and distance to star in units of deg, deg, and pc.

pmra (astropy Quantity array)
Proper motion in right ascension in units of $mas/year$

pmdec (astropy Quantity array)
Proper motion in declination in units of $mas/year$

rv (astropy Quantity array)
Radial velocity in units of km/s

cachedir (string)
Path to cache directory

5.2 Planet Population

The Planet Population module encodes the probability density functions of all required planetary parameters, both physical and orbital, as well as generating functions to return samples of these parameters. The planet is described by its semi-major axis, eccentricity, orbital orientation, radius, mass, and geometric albedo (see §5.2.2). Certain parameter models may be empirically derived while others may come from analyses of observational surveys. This module also encodes the limits on all parameters to be used for sampling the distributions and determining derived cutoff values such as the maximum target distance for a given instrument's IWA.

The coordinate system of the simulated exosystems is defined as in Figure 4. The observer looks at the target star (r_{targ} in the observatory module) along the $\mathbf{s}_3$ axis. The observer is located $-d\mathbf{s_3}$ from the target star at the time of observation where d is the distance between the observer and the target star. The argument of periapse, inclination, and longitude of the ascending node (ω, I, Ω) are defined as a 3-1-3 rotation about the unit vectors defining the $\mathcal{S}$ reference frame. This rotation defines the standard Equinoctial reference frame ($\hat{\mathbf{e}}, \hat{\mathbf{q}}, \hat{\mathbf{h}}$), with the true anomaly (ν) measured from the eccentricity unit vector ($\hat{\mathbf{e}}$), which points in the direction of periapse. The planet-star orbital radius vector $\mathbf{r}_{P/S}$ (this is the SimulatedUniverse attribute r) is projected into the $\mathbf{s}_1, \mathbf{s}_2$ plane as the projected separation vector $\mathbf{s}$, with magnitude s, and the phase (star-planet-observer) angle (β) is closely approximated by the angle between $\mathbf{r}_{P/S}$ and its projection onto $\mathbf{s}_3$. The parallactic angle of the planet (not drawn) is the angle formed between $\mathbf{s}_1$ and $\mathbf{s}$ in the clockwise direction.

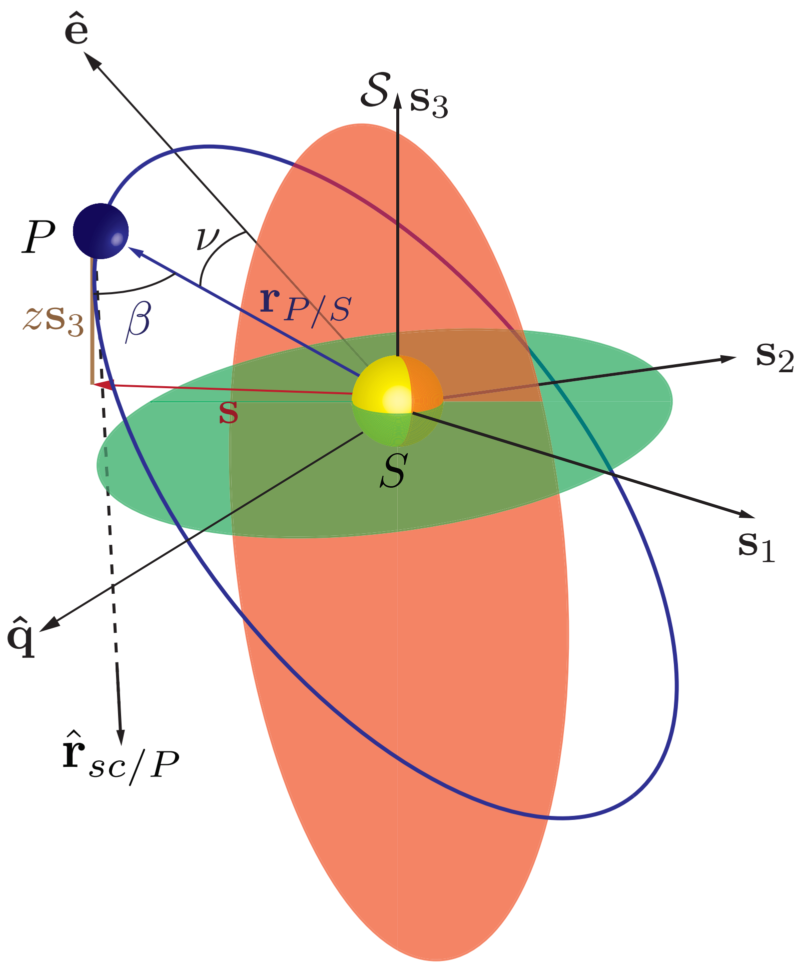

Fig. 4. Definition of reference frames and coordinates of simulated exosystems. The observer lies along the negative $\mathbf{s}_3$ axis so that the observer-star unit vector is $+\mathbf{s}_3$.

Note: The Planet Population module does not model the physics of planetary orbits or the amount of light reflected or emitted by a given planet, but rather encodes the statistics of planetary occurrence and properties.

5.2.1 Planet Population Object Attribute Initialization

Input

The following are all entries in the passed specs dictionary (derived from the JSON script file or another dictionary). Values not specified will be replaced with defaults, as listed. It is important to note that many of these (in particular mass and radius) may be mutually dependent, and so some implementations may choose to only use some for inputs and set the rest via the physical models. The `gen_input_check` method is a helper method to check whether the input is an integer. The `checkranges` method is a helper method checking range validity on 2 element lists of ranges.

arange (float 1×2 array)
Semi-major axis range in units of *AU*. Default value is [0.01, 100]

erange (float 1×2 array)
Eccentricity range. Default value is [0.01,0.99]

Irange (float 1×2 array)
Inclination range in units of *deg*. Default value is [0,180]

Orange (float 1×2 array)
Ascension of the ascending node range in units of *deg*. Default value is [0,360]

wrange (float 1×2 array)
Perigee range in units of *deg*. Default value is [0,360]

prange (float 1×2 array)
Planetary geometric albedo range. Default value is [0.1,0.6]

Rprange (float 1×2 array)
Planetary Radius in Earth radii. Default value is [1, 30]

Mprange (float 1×2 array)
Planetary mass in Earth masses. Default value is [1, 4131]

scaleOrbits (boolean)
Boolean where True means planetary orbits are scaled by the square root of stellar luminosity. Default value is False.

constrainOrbits (boolean)
Boolean where True means planetary orbits are constrained to never leave the semi-major axis range (arange). Default value is False.

eta (float)
The average occurrence rate of planets per star for the entire population. The expected number of planets generated per simulation is equal to the product of eta with the total number of targets. Note that this is the expectation value *only*—the actual number of planets generated in a given simulation may vary depending on the specific method of sampling the population.

Attributes

PlanetPhysicalModel (PlanetPhysicalModel module)
PlanetPhysicalModel class object

arange (astropy Quantity 1×2 array)
Semi-major axis range defined as [a_min, a_max] in units of *AU*

erange (float 1×2 ndarray)
Eccentricity range defined as [e_min, e_max]

Irange (astropy Quantity 1×2 array)
Planetary orbital inclination range defined as [I_min, I_max] in units of *deg*

Orange (astropy Quantity 1×2 array)
Right ascension of the ascending node range defined as [O_min, O_max] in units of *deg*

wrange (astropy Quantity 1×2 array)
Argument of perigee range defined as [w_min, w_max] in units of *deg*

prange (float 1×2 ndarray)
Planetary geometric albedo range defined as [p_min, p_max]

Rprange (astropy Quantity 1×2 array)
Planetary radius range defined as [R_min, R_max] in units of *earthRad*

Mprange (astropy Quantity 1×2 array)
Planetary mass range defined as [Mp_min, Mp_max] in units of *earthMass*

rrange (astropy Quantity 1×2 array)
Planetary orbital radius range defined as [r_min, r_max] derived from PlanetPopulation.arange and PlanetPopulation.erange, in units of *AU*

scaleOrbits (boolean)
Boolean where True means planetary orbits are scaled by the square root of stellar luminosity.

constrainOribts (boolean)
Boolean where True means planetary orbits are constrained to never leave the semi-major axis range (arange). If set to True, an additional method (`gen_eccen_from_sma`) must be provided by the implementation—see below.

eta (float)
The average occurrence rate of planets per star for the entire population.

uniform (float, callable)
Uniform distribution over a given range.

logunif (float, callable)
Log-uniform distribution over a given range.

cachedir (string)
Path to cache directory.

5.2.2 Planet Population Value Generators

For each of the parameters represented by the input attributes, the planet population object will provide a method that returns random values for the attributes, within the ranges specified by each attribute (so that, for example, there will be samples of semi-major axis corresponding to `arange`, etc.). Each of these methods will take a single input of the number of values to generate. These methods will encode the probability density functions representing each parameter, and use either a rejection sampler or other (numpy or scipy) provided sampling method to generate random values. All returned values will have the same type/default units as the attributes.

In cases where values need to be sampled jointly (for example if you have a joint distribution of semi-major axis and planetary radius) then the sampling will be encoded in the `gen_plan_params` function. In cases where there is a deterministic calculation of one parameter from another (as in mass calculated from radius) this will be provided separately in the Planet Physical module. Any non-standard distribution functions being sampled by one of these methods should be created as object attributes in the implementation constructor so that they are available to other modules.

The methods are:

`gen_plan_params`	Returns values of semi-major axis (in units of *AU*), eccentricity, geometric albedo, and planetary radius (in units of *earthRad*)
`gen_angles`	Returns values of orbital inclination, longitude of the ascending node, and argument of perigee, all in units of *deg*
`gen_mass`	Returns planetary mass values in units of *earthMass*
`dist_sma`	Provides the probability density function for the semi-major axis
`dist_eccen`	Provides the probability density function for the eccentricity
`dist_eccen_from_sma`	Provides the probability density function for the eccentricity given a value of semi-major axis. This function is used when `constrainOrbits` is set to `True`.
`dist_albedo`	Provides the probability density function for the albedo
`dist_radius`	Provides the probability density function for the radius
`dist_mass`	Provides the probability density function for the mass

5.3 Planet Physical Model

The Planet Physical Model module contains models of the light emitted or reflected by planets in the wavelength bands under investigation by the current mission simulation. It takes as inputs the physical quantities sampled from the distributions in the Planet Population module and generates synthetic spectra (or band photometry, as appropriate). The specific implementation of this module can vary greatly, and can be based on any of the many available planetary geometric albedo, spectra and phase curve models. This module contains the attribute `cachedir`, the path to the cache directory. As required, this module also provides physical models relating dependent parameters that cannot be sampled independently (for example density models relating plant mass and radius). While the specific methods will depend highly on the physical models being used, the prototype provides four stubs that will be commonly useful:

`calc_albedo_from_sma`	Calculate planetary geometric albedo as a function of the semi-major axis. (see §5.3.1)
`calc_radius_from_mass`	Calculate planetary radii from their masses (see §5.3.2).
`calc_mass_from_radius`	Calculate planetary masses from their radii (see §5.3.3).
`calc_Phi`	Calculate the value of the planet phase function given its phase angle. The prototype implementation uses the Lambert phase function (see §5.3.4).
`calc_Teff`	Calcluate the effective planet temperature given the stellar luminosity, planet albedo and star-planet distance (see §5.3.5).

5.3.1 calc_albedo_from_sma Method

Helper function for calculating albedo given the semi-major axis. The prototype provides only a dummy function that always returns the same value of 0.367.

Input

a (astropy Quantity array)

Semi-major axis values

Output

p (ndarray)

Albedo values

5.3.2 calc_radius_from_mass Method

Helper function for calculating radius given the mass. Prototype provides only a dummy function that assumes a density of water.

Input

Mp (astropy Quantity array)

Planet mass in units of Earth mass

Output

Rp (astropy Quantity arrau)

Planet radius in units of Earth radius

5.3.3 calc_mass_from_radius) Method

Helper function for calculating mass given the radius.

Input

Rp (astropy Quantity array)

Planet radius in units of Earth radius

Output

Mp (astropy Quantity array)

Planet mass in units of Earth mass

5.3.4 calc_Phi Method

Calculate the phase function. Prototype method uses the Lambert phase function from Sobolev 1975.

Input

beta (astropy Quantity array)

Planet phase angles at which the phase function is to be calculated, in units of rad

Output

Phi (ndarray)

Planet phase function

5.3.5 calc_Teff Method

Calcluates the effective planet temperature given the stellar luminosity, planet albedo and star-planet distance. This calculation represents a basic balckbody power balance, and does not take into account the actual emmisivity of the planet, or any non-equilibrium effects or temperature variations over the surface.
Note: The input albedo is taken to be the bond albedo, as required by the equilibrium calculation. For an isotropic scatterer (Lambert phase function) the Bond albedo is 1.5 times the geometric albedo. However, the Bond albedo must be strictly defined between 0 and 1, and an albedo of 1 produces a zero effective temperature.

Input

starL (float ndarray)

Stellar luminosities in units of solar luminosity. Not an astropy quantity.

d (astropy Quantity array)

Star-planet distances

p (float ndarray)

Planet albedos

Output

Teff (astropy Quantity)

Planet effective temperature in degrees K

5.4 Optical System

The Optical System module contains all of the necessary information to describe the planet signal and background noise fluxes at the image plane of all relevant instruments, and calculate the required integration time for a given observation. This also requires encoding the design of the telescope including the attenuation due to all optical elements that are not explicitly part of the starlight suppression system, the area of the entrance pupil, and the fraction of that area that is obscured (by spiders and secondary mirror). A description of the science instruments is also required, including detector details such as read noise, dark current, and readout cycle. The baseline is assumed to be an imager and a spectrograph. Finally, the Optical System must include the performance of all starlight suppression systems, which are broadly grouped as internal coronagraphs and external occulters. The Optical System module also contains a required dictionary of all potential mission observing modes. Each mode is defined by a combination of a science instrument and a starlight suppression system, operating in a set spectral window (bandpass). Note that there is no requirement that the bandpass of the instrument be exactly matched to the bandpass of the science instrument. It is up to the user to ensure that the modes are defined consistently, and scaling of all relevant values to the selected central wavelength will occur automatically.

The starlight suppression system throughput and contrast - or residual intensity - can be encoded with angular separation and wavelength dependant definitions. Some specific Optical System modules may also require encoding the Point Spread Functions (PSF) for on- and off-axis sources. At the opposite level of complexity, the encoded portions of this module may be a description of all of the optical elements between the telescope aperture and the science camera, along with a method of propagating an input wavefront to the final image plane. Intermediate implementations can include partial propagations, or collections of static PSFs representing the contributions of various system elements. The encoding of the optical train will allow for the extraction of specific bulk parameters including the instrument inner working angle (IWA), outer working angle (OWA), and mean and max contrast and throughput.

By definition, the detection mode IWA correspond to the working angle at which integration times are calculated during the detection phase. This IWA must not be confused with the global IWA. There are 3 types of IWA/OWA:
1- Each coronagraph has its own IWA/OWA in arcsec defined at its operation wavelength.
2- Each observing mode has its own IWA/OWA, based on the coronagraph IWA/OWA and rescaled to the modes specific wavelength. For simple cases where no observing modes are specified, the detection IWA will simply correspond to the coronagraph IWA.
3- A global IWA/OWA can be specified for the whole telescope, to filter out targets during the initialization, thus before the mission starts. By defaults, the global IWA = minimum(mode_IWAs) and global OWA = maximum(mode_OWAs). However, the user can specify a global IWA that is very small or even zero to avoid filtering out targets during initialization, without affecting the detection IWA described above.

The input and output of the Optical System methods are depicted in Figure 5. The Optical System module has six methods used in simulation:

`get_coro_param`	Called in OpticalSystem constructor to create a lambda function from a given starlight suppression system fits file
`Cp_Cb_Csp`	Called by `calc_intTime` to calculate the electron count rates for planet signal, background noise, and speckle residuals (see §5.4.3).
`calc_intTime`	Calculates the integration times for specific values of planet zodiacal noise, delta magnitude, and angular separation (see §5.4.4).
`calc_minintTime`	Calculates the minimum integration times for all the stars from the target list, using optimistic input parameters (see §5.4.5).
`calc_dMag_per_intTime`	Calculates achievable planet delta magnitude per integration time (see §5.4.6).
`ddMag_dt`	Calculates derivative of achievable delta mag per integration time (see §5.4.7).

5.4.1 Optical System Object Attribute Initialization

The specific set of inputs to this module will vary based on the simulation approach used. Here we define the specification for the case where static PSF(s), derived from external diffraction modeling, are used to describe the system. Note that some of the inputs are specific to "internal" or "external" (i.e. starshade) systems and will be expected based on the *occulter* flag.

Input

obscurFac (float)

Obscuration factor due to secondary mirror and spiders. Default value is 0.1.

shapeFac (float)

Shape factor of the unobscured pupil area, so that $shapeFac \times pupilDiam^2 \times (1 - obscurFac) = pupilArea$. Default value is $\frac{\pi}{4}$.

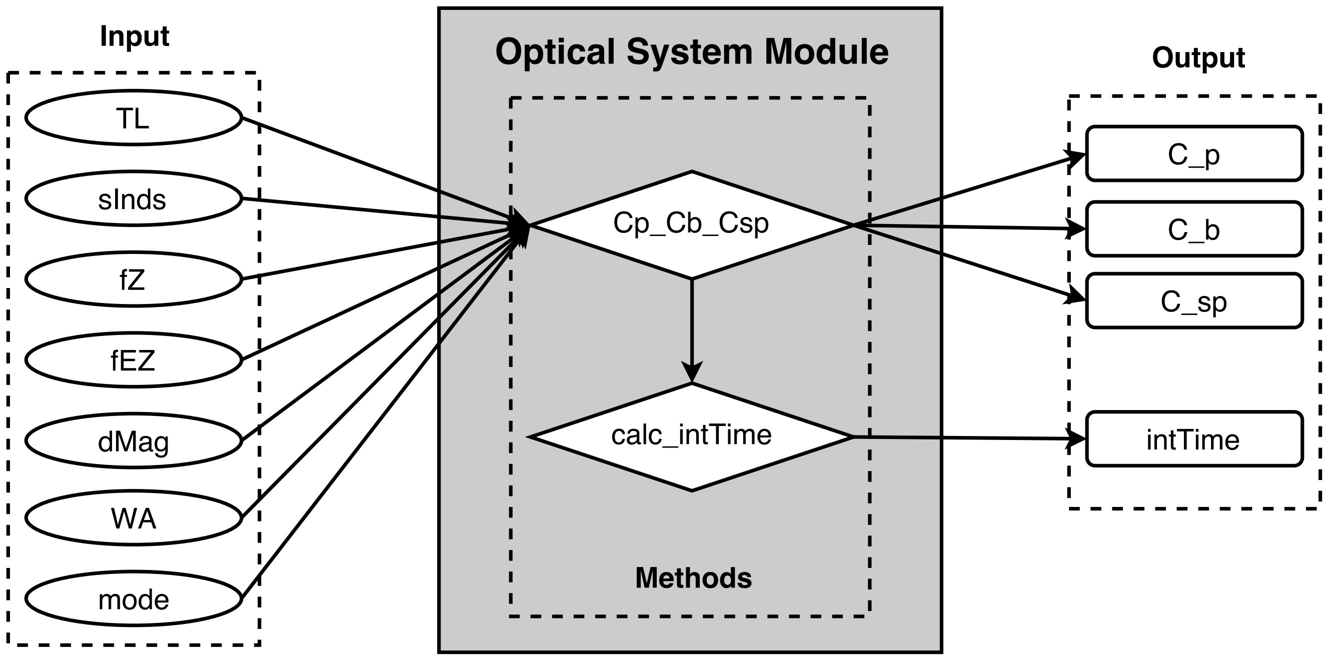

Fig. 5. Depiction of Optical System module methods including input and output (see §5.4.3, §5.4.4).

pupilDiam (float)
Entrance pupil diameter in *m*. Default value is 4.

IWA (float)
Fundamental Inner Working Angle in units of *arcsec*. No planets can ever be observed at smaller separations. If not set, defaults to smallest IWA of all starlightSuppressionSystems.

OWA (float)
Fundamental Outer Working Angle in units of *arcsec*. Set to Inf for no OWA. If not set, defaults to largest OWA of all starlightSuppressionSystems. JSON values of 0 will be interpreted as Inf.

intCutoff (float)
Maximum allowed integration time in units of *day*. No integration will be started that would take longer than this value. Default value is 50.

dMag0 (float)
Favorable planet delta magnitude value used to calculate the minimum integration times for inclusion in target list.

WA0 (float)
Instrument working angle value used to calculate the minimum integration times for inclusion in target list (defaults to detection IWA-OWA midpoint), in units of *arcsec*.

scienceInstruments (list of dicts)
List of dictionaries containing specific attributes of all science instruments. For each instrument, if the below attributes are missing from the dictionary, they will be assigned the default values listed, or any value directly passed as input to the class constructor.

name (string)
(Required) Instrument name (e.g. imager-EMCCD, spectro-CCD), should contain the type of instrument (imager or spectro). Every instrument should have a unique name.

QE (float, callable)
Detector quantum efficiency: either a scalar for constant QE, or a two-column array for wavelength-dependent QE, where the first column contains the wavelengths in units of *nm*. May be data or FITS filename. Default is scalar 0.9.

optics (float)
Attenuation due to optics specific to the science instrument. Default value is 0.5.

FoV (float)
Field of view in units of *arcsec*. Default value is 10.

pixelNumber (integer)
Detector array format, number of pixels per detector lines/columns. Default value is 1000.

pixelSize (float)
Pixel pitch in units of *m*. Default value is 1e-5.

sread (float)
Detector effective read noise per frame per pixel, including any gain (e.g. electron multiplication gain). Default value is 1e-6.

idark (float)
Detector dark-current per pixel in units of $1/s$. Default value is 1e-4.

CIC (float)
(Specific to CCDs) Clock-induced-charge per frame per pixel. Default value is 1e-3.

texp (float)
Exposure time per frame in units of *s*. Default value is 100.

radDos (float)
Radiation dosage. Default value is 0.

PCeff (float)
Photon counting efficiency. Default value is 0.8.

ENF (float)
(Specific to EM-CCDs) Excess noise factor. Default value is 1.

Rs (float)
(Specific to spectrometers) Spectral resolving power defined as $\lambda/d\lambda$. Default value is 50.

lenslSamp (float)
(Specific to spectrometers) Lenslet sampling, number of pixel per lenslet rows or cols. Default value is 2.

starlightSuppressionSystems (list of dicts)
List of dictionaries containing specific attributes of all starlight suppression systems. For each system, if the below attributes are missing from the dictionary, they will be assigned the default values listed, or any value directly passed as input to the class constructor. In case of multiple systems, specified wavelength values (lam, deltaLam, BW) of the first system become the new default values.
The following items can be encoded either as scalar parameters, or as two-column arrays for angular separation-dependent parameters, where the first column contains the separations in units of *arcsec*, or as 2D array for angular separation- and wavelength- dependent parameters, where the first column contains the angular separation values in units of *arcsec* and the first row contains the wavelengths in units of *nm*: *occ_trans*, *core_thruput*, *core_contrast*, *core_mean_intensity*, *core_area*.

name (string)
(Required) System name (e.g. HLC-500, SPC-700), should also contain the central wavelength the system is optimized for. Every system must have a unique Name.

optics (float)
Attenuation due to optics specific to the coronagraph, e.g. polarizer, Lyot stop, extra flat mirror. Default value is 1.

lam (float)
Central wavelength λ in units of *nm*. Default value is 500.

deltaLam (float)
Bandwidth $\Delta\lambda$ in units of *nm*. Defaults to lambda $\times$ BW (defined hereunder).

BW (float)
Bandwidth fraction $(\Delta\lambda/\lambda)$. Only applies when deltaLam is not specified. Default value is 0.2.

IWA (float)
Inner Working Angle of this system in units of *arcsec*. If not set, or if too small for this system contrast/throughput definitions, defaults to smallest WA of contrast/throughput definitions.

OWA (float)
Specific Outer Working Angle of this system in units of *arcsec*. Set to Inf for no OWA. If not set, or if too large for this system contrast/throughput definitions, defaults to largest WA of contrast/throughput definitions. JSON values of 0 will be interpreted as Inf.

occ_trans (float, callable)
Intensity transmission of extended background sources such as zodiacal light. Includes pupil mask, occulter, Lyot stop and polarizer. Default is scalar 0.2.

core_thruput (float, callable)
System throughput in the FWHM region of the planet PSF core. Default is scalar 0.1.

core_contrast (float, callable)
System contrast defined as the starlight residual normalized intensity in the PSF core, divided by the core

throughput. Default is scalar 1e-10.

core_mean_intensity (float, callable)

Mean starlight residual normalized intensity per pixel, required to calculate the total core intensity as $core\_mean\_intensity \times Npix$. If not specified, then the total core intensity is equal to $core\_contrast \times core\_thruput$.

core_area (float, callable)

Area of the FWHM region of the planet PSF, in units of $arcsec^2$. If not specified, the default core area is equal to $\pi\left(\frac{\sqrt{2}}{2}\frac{\lambda}{D}\right)^2$.

core_platescale (float)

Platescale used for a specific set of coronagraph parameters, in units of lambda/D per pixel. Defaults to the instrument pixelScale.

ohTime (float)

Optical system overhead time in units of *day*. Default value is 1. This is the (assumed constant) amount of time required to set up the optical system (i.e., dig the dark hole or do fine alignment with the occulter). It is added to every observation, and is separate from the observatory overhead defined in the observatory module, which represents the observatory's settling time. Both overheads are added to the integration time to determine the full duration of each detection observation.

occulter (boolean)

True if the system has an occulter (external or hybrid system), otherwise False (internal system)

occulterDiameter (float)

Occulter diameter in units of *m*. Measured petal tip-to-tip.

NocculterDistances (integer)

Number of telescope separations the occulter operates over (number of occulter bands). If greater than 1, then the occulter description is an array of dicts.

occulterDistance (float)

Telescope-occulter separation in units of *km*.

occulterBlueEdge (float)

Occulter blue end of wavelength band in units of *nm*.

occulterRedEdge (float)

Occulter red end of wavelength band in units of *nm*.

observingModes (list of dicts)

List of dictionaries containing specific attributes of all mission observing modes. Each observing mode is a combination of an instrument and a system, operating at a given wavelength, which by default is the wavelength defined in the starlight suppression system of the observing mode. If an observing mode is operating at a different wavelength than the system default wavelength, then this new wavelength must be added to the observing mode, and the system performance will be automatically rescaled to the new wavelength. If no observing mode is defined, the default observing mode simply combines the first instrument and the first system.

instName (string)

(Required) Instrument name. Must match with the name of a defined science instrument.

systName (string)

(Required) System name. Must match with the name of a defined starlight suppression system.

inst (dict)

Selected instrument of the observing mode.

syst (dict)

Selected system of the observing mode.

detectionMode (boolean)

True if this observing mode is the detection mode, otherwise False. Only one detection mode can be specified. If not specified, default detection mode is first imager mode.

SNR (float)

Signal-to-noise ratio threshold. Defaults to 5.

timeMultiplier (float)

Integration time multiplier. Equal to the number of discrete integrations needed to cover the full field of view (e.g. shaped pupil), or the full wavelength band and all required polarization states. For example, if the band is split into three sub-bands, and there are two polarization states that must be measured, and each of these must be done sequentially, then this value would equal 6. However, if the three sub-bands could be observed at the same time (e.g., by separate detectors) then the value would be two (for the two polarization states). Defaults to 1.

lam (float)

Central wavelength in units of nm. Defaults to corresponding system value.

deltaLam (float)
Bandwidth in units of nm. Defaults to corresponding system value.

BW (float)
Bandwidth fraction. Defaults to corresponding system value.

For all values that may be either scalars or interpolants, in the case where scalar values are given, the optical system module will automatically wrap them in lambda functions so that they become callable (just like the interpolant) but will always return the same value for all arguments. The inputs for interpolants may be filenames (full absolute paths) with tabulated data, or NumPy ndarrays of argument and data (in that order in rows so that input[0] is the argument and input[1] is the data). When the input is derived from a JSON file, these must either be scalars or filenames.

The starlight suppression system and science instrument dictionaries can contain any other attributes required by a particular optical system implementation. The only significance of the ones enumerated above is that they are explicitly checked for by the prototype constructor, and cast to their expected values.

Attributes

These will always be present in an OpticalSystem object and directly accessible as `OpticalSystem.Attribute`.

obscurFac (float)
Obscuration factor due to secondary mirror and spiders

shapeFac (float)
Shape factor of the unobscured pupil area, so that $shapeFac \times pupilDiam^2 \times (1 - obscurFac) = pupilArea$

pupilDiam (astropy Quantity)
Entrance pupil diameter in units of m

pupilArea (astropy Quantity)
Entrance pupil area in units of m^2

haveOcculter (boolean)
Boolean signifying if the system has an occulter

IWA (astropy Quantity)
Fundamental Inner Working Angle in units of *arcsec*

OWA (astropy Quantity)
Fundamental Outer Working Angle in units of *arcsec*

intCutoff (astropy Quantity)
Maximum allowed integration time in units of *day*

dMag0 (float)
Favorable planet delta magnitude value used to calculate the minimum integration times for inclusion in target list.

WA0 (astropy Quantity)
Instrument working angle value used to calculate the minimum integration times for inclusion in target list.

scienceInstruments (list of dicts)
List of dictionaries containing all supplied science instrument attributes. Typically the first instrument will be the imager, and the second the spectrograph (IFS). Only required attribute is 'name'. See above for other commonly used attributes.

starlightSuppressionSystems (list of dicts)
List of dictionaries containing all supplied starlight suppression system attributes. Typically the first system will be used with the imager, and the second with the IFS. Only required attribute is 'name'. See above for other commonly used attributes.

observingModes (list of dicts)
List of dictionaries containing all mission observing modes. Only required attribute are 'instName' and 'systName'. See above for other commonly used attributes.

cachedir (string)
Path to cache directory.

5.4.2 get_coro_params Method

For a given starlightSuppressionSystem, this method loads an input parameter from a table (fits file) or a scalar value. It then creates a callable lambda function, which depends on the wavelength of the system and the angular separation of the observed planet.

Input

syst (dict)
Dictionary containing the parameters of one starlight suppression system

param_name (string)
Name of the parameter that must be loaded

fill (float)
Fill value for working angles outside of the input array definition

Output

syst (dict)
Updated dictionary of parameters

5.4.3 Cp_Cb_Csp Method

The `Cp_Cb_Csp` method calculates the electron count rates for planet signal, background noise, and speckle residuals.

Input

TL (TargetList module)
TargetList class object, see §5.9 for definition of available attributes

sInds (integer ndarray)
Integer indices of the stars of interest

fZ (astropy Quantity array)
Surface brightness of local zodiacal light in units of $1/arcsec^2$

fEZ (astropy Quantity array)
Surface brightness of exo-zodiacal light in units of $1/arcsec^2$

dMag (float ndarray)
Differences in magnitude between planets and their host star.

WA (astropy Quantity array)
Working angles of the planets of interest in units of *arcsec*

mode (dict)
Selected observing mode

Output

C_p (astropy Quantity array)
Planet signal electron count rate in units of $1/s$

C_b (astropy Quantity array)
Background noise electron count rate in units of $1/s$

C_sp (astropy Quantity array)
Residual speckle spatial structure (systematic error) in units of $1/s$

5.4.4 calc_intTime Method

The `calc_intTime` method calculates the integration time required for specific planets of interest. This method is called from the SurveySimulation module.

Input

TL (TargetList module)
TargetList class object, see §5.9 for definition of available attributes

sInds (integer ndarray)
Integer indices of the stars of interest

fZ (astropy Quantity array)
Surface brightness of local zodiacal light in units of $1/arcsec^2$

fEZ (astropy Quantity array)
Surface brightness of exo-zodiacal light in units of $1/arcsec^2$

dMag (float ndarray)
Differences in magnitude between planets and their host star.

WA (astropy Quantity array)
Working angles of the planets of interest in units of *arcsec*
mode (dict)
Selected observing mode

Output
intTime (astropy Quantity array)
Integration time for each of the planets of interest in units of *day*

5.4.5 calc_minintTime Method

The `calc_minintTime` method calculates the minimum integration time for each star in the target list. This method is called from the TargetList module.

Input
TL (TargetList module)
TargetList class object, see §5.9 for definition of available attributes

Output
minintTime (astropy Quantity array)
Minimum integration time for each target star in units of *day*

5.4.6 calc_dMag_per_intTime Method

The `calc_dMag_per_intTime` method calculates the achievable planet delta magnitude (delta mag) for one integration time per star in the input list at one or more working angles.

Input
intTime (astropy Quantity array)
Integration times in units of *day*
TL (TargetList module)
TargetList class object, see §5.9 for definition of available attributes
sInds (integer ndarray)
Integer indices of the stars of interest
fZ (astropy Quantity array)
Surface brightness of local zodiacal light in units of $1/arcsec^2$
fEZ (astropy Quantity array)
Surface brightness of exo-zodiacal light in units of $1/arcsec^2$
WA (astropy Quantity array)
Working angles of the planets of interest in units of *arcsec*
mode (dict)
Selected observing mode

Output
dMag (float ndarray)
Achievable dMag for given integration time and working angle

5.4.7 ddMag_dt Method

The `ddMag_dt` method calculates the derivative of achievable dMag with respect to integration time.

Input
intTime (astropy Quantity array)
Integration times in units of *day*
TL (TargetList module)
TargetList class object, see §5.9 for definition of available attributes

sInds (integer ndarray)
Integer indices of the stars of interest

fZ (astropy Quantity array)
Surface brightness of local zodiacal light in units of $1/arcsec^2$

fEZ (astropy Quantity array)
Surface brightness of exo-zodiacal light in units of $1/arcsec^2$

WA (astropy Quantity array)
Working angles of the planets of interest in units of $arcsec$

mode (dict)
Selected observing mode

Output

ddMagdt (astropy Quantity array)
Derivative of achievable dMag with respect to integration time in units of $1/s$

5.5 Zodiacal Light

The Zodiacal Light module contains methods to calculate the zodiacal light surface brightness of local Zodi, extrasolar Zodi, star specific minimum Zodi, and star specific maximum Zodi.

`fZ`	Calculates the surface brightness of local zodiacal light (see §5.5.2)
`fEZ`	Calculates the surface brightness of exozodiacal light (see §5.5.3)
`generate_fZ`	Calculates the fZ for each star in TL for 1 year (see §5.5.4)
`calcfZmax`	Calculates the maximum fZ excluding keepout regions (see §5.5.5)
`calcfZmin`	Calculates the minimum fZ (see §5.5.6)

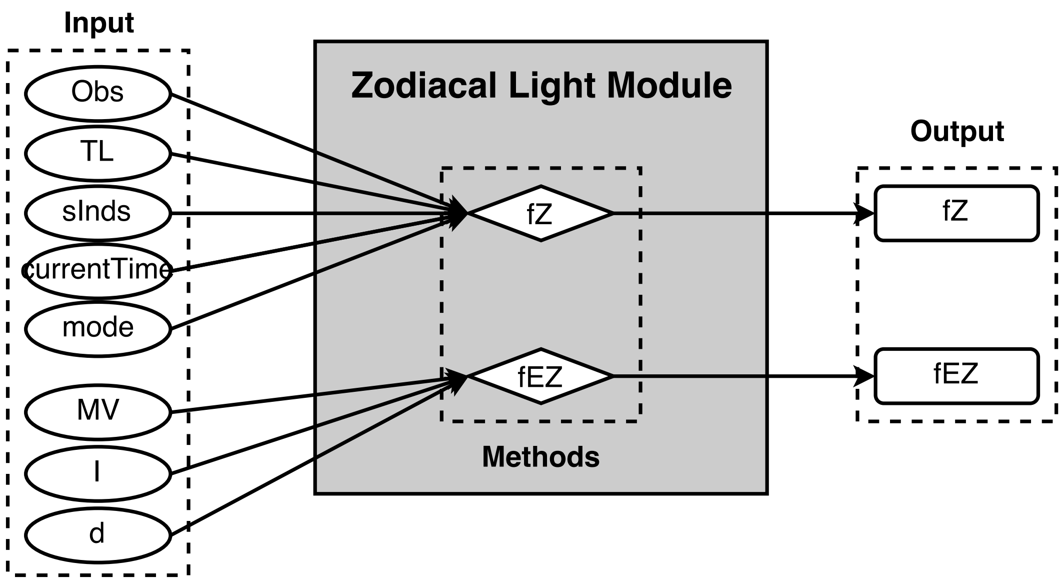

Fig. 6. Depiction of Zodiacal Light module methods including input and output (see §5.5.2 and §5.5.3).

5.5.1 Zodiacal Light Object Attribute Initialization

Input

magZ (float)
Zodiacal light brightness magnitude (per $arcsec^2$). Defaults to 23.

magEZ (float)
Exo-zodiacal light brightness magnitude (per $arcsec^2$). Defaults to 22.

varEZ (float)
Exo-zodiacal light variation (variance of log-normal distribution). Defaults to 0 (constant exo-zodiacal light).

Attributes

magZ (float)
Zodi brightness magnitude (per $arcsec^2$)

magEZ (float)
Exo-zodi brightness magnitude (per $arcsec^2$)

varEZ (float)
Exo-zodiacal light variation (variance of log-normal distribution)

fZ0 (astropy Quantity)
Default surface brightness of zodiacal light in units of $1/arcsec^2$

fEZ0 (astropy Quantity)
Default surface brightness of exo-zodiacal light in units of $1/arcsec^2$

cachedir (string)
Path to cache directory

5.5.2 fZ Method

The `fZ` method returns surface brightness of local zodiacal light for planetary systems. This functionality is used by the Simulated Universe module.

Input

Obs (Observatory module)
Observatory class object, see §5.11 for description of functionality and attributes

TL (TargetList module)
TargetList class object, see §5.9 for description of functionality and attributes

sInds (integer ndarray)
Integer indices of the stars of interest

currentTime (astropy Time array)
Current absolute mission time in MJD

mode (dict)
Selected observing mode

Output

fZ (astropy Quantity array)
Surface brightness of zodiacal light in units of $1/arcsec^2$

5.5.3 fEZ Method

The `fEZ` method returns surface brightness of exo-zodiacal light for planetary systems. This functionality is used by the Simulated Universe module.

Input

MV (integer ndarray)
Apparent magnitude of the star (in the V band)

I (astropy Quantity array)
Inclination of the planets of interest in units of *deg*

d (astropy Quantity n$\times$3 array)
Distance to star of the planets of interest in units of *AU*

Output

fEZ (astropy Quantity array)
Surface brightness of exo-zodiacal light in units of $1/arcsec^2$

5.5.4 generate_fZ Method

Calculates fZ values for each star over an entire earth orbit of the sun. This function is ONLY called by calcfZmax or calcfZmin functions. This method caches data to .starkfZ files. The output is a 2d astropy quantity array containing zodiacal light surface brightness for each star at 1000 points evenly distributed over 1 year. This method is not called by prototype modules.

Input

Obs (Observatory module)
Observatory class object

TL (TargetList module)
TargetList class object

currentTimeAbs (astropy Time)
Current absolute mission time in *MJD*

mode (dict)
Selected observing mode

hashname (string)
String describing the mission specific json file

Output

fZ [resolution, sInds]
where fZ is the zodiacal light for each star and sInds are the indicies to generate fZ for

5.5.5 calcfZmax Method

Finds the value and time of maximum zodiacal light for each star over an entire orbit of the sun not including keeoput angles. The prototype returns a constant fZ and the current absolute time.

Input

sInds [sInds]
the star indices we would like fZmax and fZmaxInds returned for

Obs (Observatory module)
Observatory class object

TL (TargetList module)
TargetList class object

currentTimeAbs (astropy Time)
Current absolute mission time in *MJD*

mode (dict)
Selected observing mode

hashname (string)
String describing the mission specific json file

Output

valfZmax [astropy Quantity array]
maximum value of fZ for each star specified by sInds with units of 1/arcsec^2

absTimefZmax [astropy Time array]
absolute time fZmax occurs for each star specified by sInds

5.5.6 calcfZmin Method

Finds the minimum zodiacal light values for each star over an entire earth orbit of the sun. The prototype implementation returns a constant fZ and the current absolute time.

Input

sInds [sInds]
the star indices we would like fZmin and fZminInds returned for

Obs (Observatory module)
Observatory class object

TL (TargetList module)
TargetList class object

currentTimeAbs (astropy Time)
Current absolute mission time in *MJD*

mode (dict)
Selected observing mode

hashname (string)
String describing the mission specific json file

Output

valfZmin [astropy Quantity array]
minimum value of fZ for each star specified by sInds with units of arcsec^{-2}

absTimefZmin [astropy Time array]
absolute time fZmin occurs for each star specified by sInds

5.6 Background Sources

The Background Sources module provides density of background sources for a given target based on its coordinates and the integration depth. The integration depth is the limiting planet magnitude, that is the magnitude of the faintest planet we can observe. This will be used in the post-processing module to determine false alarms based on confusion. The prototype module has no inputs, the attribute `cachedir` (path to the cache directory), and only a single function: `dNbackground` (see §5.6.1).

5.6.1 dNbackground Method

Returns background source number densities

Input

coords (astropy SkyCoord array)
SkyCoord object containing right ascension, declination, and distance to star of the planets of interest in units of *deg*, *deg* and *pc*.

intDepths (float ndarray)
Integration depths equal to the planet magnitude (Vmag+dMag), i.e. the V magnitude of the dark hole to be produced for each target. Must be of same length as coords.

Output

dN (astropy Quantity array)
Number densities of background sources for given targets in units of $1/arcmin^2$. Same length as inputs.

5.7 Post-Processing

The Post-Processing module encodes the effects of post-processing on the data gathered in a simulated observation, and the effects on the final contrast of the simulation. The Post-Processing module is also responsible for determining whether a planet detection has occurred for a given observation, returning one of four possible states—true positive (real detection), false positive (false alarm), true negative (no detection when no planet is present) and false negative (missed detection). These can be generated based solely on statistical modeling or by processing simulated images.

The Post-Processing module contains the `det_occur` method (see §5.7.2). This method determines if a planet detection occurs for a given observation. The input and output of this method are depicted in Figure 7.

5.7.1 Post-Processing Object Attribute Initialization

Input

FAP (float)
Detection false alarm probability. Default value is 3×10^{-7}.

MDP (float)
Missed detection probability. Default value is 10^{-3}.

ppFact (float, callable)
Post-processing contrast factor, between 0 and 1: either a scalar for constant gain, or a two-column array for

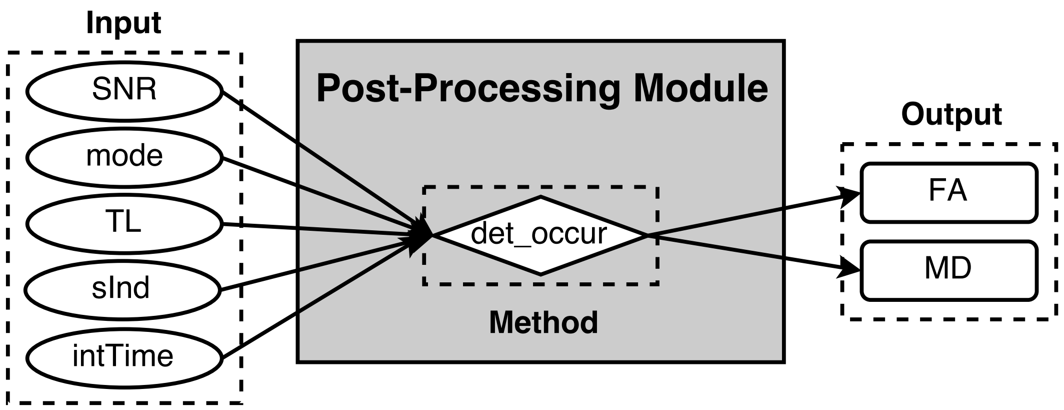

Fig. 7. Depiction of Post-Processing module method including input and output (see §5.7.2).

separation-dependent gain, where the first column contains the angular separation in units of *arcsec*. May be data or FITS filename. Default value is 1.

FAdMag0 (float, callable)
Minimum delta magnitude that can be obtained by a false alarm: either a scalar for constant dMag, or a two-column array for separation-dependent dMag, where the first column contains the angular separation in units of arcsec. May be data or FITS filename. Default value is 15.

Attributes

BackgroundSources (BackgroundSources module)
BackgroundSources class object (see 5.6)

FAP (float)
Detection false alarm probability

MDP (float)
Missed detection probability

ppFact (float, callable)
Post-processing contrast factor, between 0 and 1.

FAdMag0 (float, callable)
Minimum delta magnitude that can be obtained by a false alarm.

cachedir (string)
Path to cache directory

5.7.2 det_occur Method

The `det_occur` method determines if a planet detection has occurred.

Input

SNR (float ndarray)
Signal-to-noise ratio of the planets around the selected target

mode (dict)
Selected observing mode

TL (TargetList module)
TargetList class object

sInd (integer)
Index of the star being observed

intTime (astropy Quantity)
Selected star integration time for detection

Output

FA (boolean)
False alarm (false positive) boolean.

MD (boolean ndarray)
Missed detection (false negative) boolean with the size of number of planets around the target.

5.8 Completeness

The Completeness module takes in information from the Planet Population module to determine initial completeness and update completeness values for target list stars when called upon.

The Completeness module contains the following methods:

`target_completeness`	Generates initial completeness values for each star in the target list (see §5.8.2)
`gen_update`	generates dynamic completeness values for successive observations of each star in the target list (see §5.8.3)
`completeness_update`	Updates the completeness values following an observation (see §5.8.4)
`comp_per_intTime`	Calculates completeness values per integration time (see §5.8.6)
`dcomp_dt`	Calculates derivative of completeness with respect to integration time (see §5.8.7)

5.8.1 Completeness Object Attribute Initialization

Input

dMagLim (float)
Limiting planet-to-star delta magnitude for completeness. Defaults to 25.

minComp (float)
Minimum completeness value for inclusion in target list. Defaults to 0.1.

Monte Carlo methods for calculating completeness will require an input of the number of planet samples called `Nplanets`.

Attributes

PlanetPopulation (PlanetPopulation module)
PlanetPopulation object (see 5.2)

PlanetPhysicalModel (PlanetPhysicalModel module)
PlanetPhysicalModel module object (see 5.3)

dMagLim (float)
Limiting planet-to-star delta magnitude for completeness

minComp (float)
Minimum completeness value for inclusion in target list

cachedir (string)
Path to cache directory

5.8.2 target_completeness Method

The `target_completeness` method generates completeness values for each star in the target list.

Input

TL (TargetList module)
TargetList class object, see §5.9 for definition of functionality and attributes

Output

comp0 (float ndarray)
Contains completeness values for each star in the target list

5.8.3 gen_update Method

The `gen_update` method generates dynamic completeness values for successive observations of each star in the target list.

Input

TL (TargetList module)
TargetList class object, see §5.9 for definition of functionality and attributes

Attributes Updated

updates (float ndarray)
genrates completeness update array necessary for dynamic completeness calculations

5.8.4 completeness_update Method

The `completeness_update` method updates the completeness values for each star in the target list following an observation.

Input

TL (TargetList module)
TargetList class object, see §5.9 for definition of functionality and attributes

sInds (integer array)
Indices of stars to update

visits (integer array)
Number of visits for each star

dt (astropy Quantity array)
Time since previous observation

Output

comp0 (float ndarray)
Updated completeness values for each star in the target list

5.8.5 revise_update

Keeps completeness update values only for targets remaining in target list during filtering (called from TargetList.filter_target_list)

Input

ind (ndarray)
1D numpy ndarray of indices to keep

Attributes Updated

updates (float ndarray)
stores completeness update values only for targets of interest

5.8.6 comp_per_intTime Method

The `comp_per_intTime` method calculates the completeness values per integration time.

Input

intTime (astropy Quantity array)
Integration times in units of *day*

TL (TargetList module)
TargetList class object, see §5.9 for definition of available attributes

sInds (integer ndarray)
Integer indices of the stars of interest

fZ (astropy Quantity array)
Surface brightness of local zodiacal light in units of $1/arcsec^2$

fEZ (astropy Quantity array)
Surface brightness of exo-zodiacal light in units of $1/arcsec^2$

WA (astropy Quantity array)
Working angles of the planets of interest in units of *arcsec*

mode (dict)
Selected observing mode

Output

comp (float ndarray)
: Completeness values

5.8.7 dcomp_dt Method

The `dcomp_dt` method calculates the derivative of completeness with respect to integration time.

Input

intTime (astropy Quantity array)
: Integration times in units of *day*

TL (TargetList module)
: TargetList class object, see §5.9 for definition of available attributes

sInds (integer ndarray)
: Integer indices of the stars of interest

fZ (astropy Quantity array)
: Surface brightness of local zodiacal light in units of $1/arcsec^2$

fEZ (astropy Quantity array)
: Surface brightness of exo-zodiacal light in units of $1/arcsec^2$

WA (astropy Quantity array)
: Working angles of the planets of interest in units of *arcsec*

mode (dict)
: Selected observing mode

Output

dcomp (float ndarray)
: Derivative of completeness with respect to integration time

5.9 Target List

The Target List module takes in information from the Star Catalog, Optical System, Zodiacal Light, Post Processing, Background Sources, Completeness, PlanetPopulation, and Planet Physical Model modules to generate the target list for the simulated survey. This list can either contain all of the targets where a planet with specified parameter ranges could be observed or a list of pre-determined targets such as in the case of a mission which only seeks to observe stars where planets are known to exist from previous surveys. The final target list encodes all of the same information as is provided by the Star Catalog module.

The TargetList module contains the following methods:

`populate_target_list`	Populates values from the star catalog, and updates relevant TargetList attributes (see §5.9.2)
`fillPhotometryVals`	Fills in missing photometric values
`filter_target_list`	Filters the target list by any required metrics (see §5.9.3 and §5.9.4)
`stellar_mass`	Populates target list with 'true' and 'approximate' stellar masses
`starprop`	Finds target star positions vector (see §5.9.6)
`starMag`	Calculates star visual magnitudes with B-V color (see §5.9.7)
`stellarTeff`	Calculates the effective stellar temperature based on B-V color (see §5.9.8)

5.9.1 Target List Object Attribute Initialization

Input

staticStars (boolean)
: Boolean used to force static target positions set at mission start time.

keepStarCatalog (boolean)
: Boolean representing whether to delete the star catalog object after the target list is assembled (defaults to False). If True, object reference will be available from TargetList class object.

Attributes

(StarCatalog values)
Mission specific filtered star catalog values from StarCatalog class object (see 5.1)

StarCatalog (StarCatalog module)
StarCatalog class object (only retained if keepStarCatalog is True, see 5.1)

PlanetPopulation (PlanetPopulation module)
PlanetPopulation class object (see 5.2)

PlanetPhysicalModel (PlanetPhysicalModel module)
PlanetPhysicalModel class object (see 5.3)

OpticalSystem (OpticalSystem module)
OpticalSystem class object (see 5.4)

ZodiacalLight (ZodiacalLight module)
ZodiacalLight class object (see 5.5)

BackgroundSources (BackgroundSources module)
BackgroundSources class object (see 5.6)

PostProcessing (PostProcessing module)
PostProcessing class object (see 5.7)

Completeness (Completeness module)
Completeness class object (see 5.8)

tint0 (astropy Quantity array)
Minimum integration time for each target star. Calculated from `OpticalSystem.calc_minintTime` §5.4.5

comp0 (float ndarray)
Completeness value for each target star. Calculated from `Completeness.target_completeness` §5.8.2

MsEst (float ndarray)
Approximate stellar mass in M_{sun}

MsTrue (float ndarray)
Stellar mass with an error component included in M_{sun}

nStars (int)
Number of target stars

cachedir (string)
Path to cache directory

5.9.2 populate_target_list Method

The `populate_target_list` method is responsible for populating values from the star catalog (or any other source) into the target list attributes. It has not specific inputs and outputs, but is always passed the full specification dictionary, and updates all relevant Target List attributes. This method is called from the prototype constructor, and does not need to be called from the implementation constructor when overloaded in the implementation. The prototype implementation copies values directly from star catalog and removes stars with any NaN attributes. It also calls the `target_completeness` in the Completeness module (§5.8.2) and the `calc_minintTime` in the Optical System module (§5.4.5) to generate the initial completeness and minimum integration time for all targets. It also generates 'true' and 'approximate' star masses using object method `stellar_mass` (see below).

Update Attributes

nStars (integer)
number of target stars

comp0 (float ndarray)
Completeness values for each target star

tint0 (astropy Quantity array)
Minimum integration times for target list stars in units of day

5.9.3 filter_target_list Method

The `filter_target_list` method is responsible for filtering the targetlist to produce the values from the star catalog (or any other source) into the target list attributes. It has not specific inputs and outputs, but is always passed the full specification dictionary, and updates all relevant Target List attributes. This method is called from the prototype constructor, immediately after the `populate_target_list` call, and does not need to be called from the implementation constructor when overloaded in the implementation. The prototype implementation filters out any targets where the widest separation planet in the modeled population would be inside the system IWA, any targets where the minimum integration time for favorable planet delta magnitude and instrument working angle is above the specified integration time cutoff, and all targets where the initial completeness is below the specified threshold.

5.9.4 Target List Filtering Helper Methods

The `filter_target_list` method calls multiple helper functions to perform the actual filtering tasks. Additional filters can be defined in specific implementations and by overloading the `filter_target_list` method. The filter subtasks (with a few exception) take no inputs and operate directly on object attributes. The prototype TargetList module calls the following methods to remove the corresponding stars:

`nan_filter`	Stars with NAN values in their parameters
`binary_filter`	Binary stars
`outside_IWA_filter`	Systems with planets inside the OpticalSystem fundamental IWA
`int_cutoff_filter`	Systems where minimum integration time is longer than OpticalSystem cutoff
`completeness_filter`	Systems not meeting the Completeness threshold
`max_dmag_filter`	Filters out all targets with minimum delta mag above the limiting delta mag (from input spec)
`main_sequence_filter`	Filters any target lists that are not on the Main Sequence (estimated from the MV and BV attributes)
`fgk_filter`	Filters any targets that are not F, G, or K stars
`vis_mag_filter`	Filters out targets with visible magnitudes below input value `Vmagcrit`
`revise_lists`	General helper function for applying filters
`life_expectancy_filter`	Filters stars which have BV ¡ 0.3

5.9.5 stellar_mass Method

This method calculates stellar mass via the formula relating absolute V magnitude and stellar mass. The values are in units of solar mass.

Update Attributes

MsEst (float ndarray)
approximate stellar masses

MsTrue (float ndarray)
stellar mass combined with random error

5.9.6 starprop Method

The `starprop` method finds target star positions vector in heliocentric equatorial (default) or ecliptic frame for current time *MJD*.

Input

sInds (integer ndarray)
Indices of the stars of interest

currentTime (astropy Time array)
Current absolute mission time in MJD

eclip (boolean)
Boolean used to switch to heliocentric ecliptic frame. Defaults to False, corresponding to heliocentric equatorial frame.

Output

r_targ (astropy Quantity $n \times 3$ array)
Target star positions vector in heliocentric equatorial (default) or ecliptic frame in units of *pc*

5.9.7 starMag Method

The `starMag` method calculates star visual magnitudes with B-V color using empirical fit to data from Pecaut and Mamajek (2013, Appendix C). The expression for flux is accurate to about 7%, in the range of validity 400 $nm < \lambda < 1000$ nm (Traub et al. 2016).

Input

sInds (integer ndarray)
Indices of the stars of interest

lam (astropy Quantity)
Wavelength in units of *nm*

Output

mV (float ndarray)
Star visual magnitudes with B-V color

5.9.8 stellarTeff Method

The `stellarTeff` method calculates the effective stellar temperature based on B-V color.

Input

sInds (integer ndarray)
Indices of the stars of interest

Output

mV (float ndarray)
Star visual magnitudes with B-V color

5.10 Simulated Universe

The Simulated Universe module instantiates the Target List module and creates a synthetic universe by populating planetary systems about some or all of the stars in the target list. For each target, a planetary system is generated based on the statistics encoded in the Planet Population module, so that the overall planet occurrence and multiplicity rates are consistent with the provided distribution functions. Physical parameters for each planet are similarly sampled from the input density functions (or calculated via the Planet physical model). All planetary orbital and physical parameters are encoded as arrays of values, with an indexing array (pInds) that maps planets to the stars in the target list.

All planetary parameters are generated in the constructor via calls to the appropriate value generating functions in the planet population module.The Simulated Universe module contains the following methods:

`gen_physical_properties`	Populates the orbital elements and physical characteristics of all planets (see §5.10.2)
`gen_M0`	Finds initial mean anomaly for each planet (see §5.10.3)
`init_systems`	Finds initial time-dependant parameters such as position and velocity vectors, along with exo-zodiacal surface brightness, delta magnitude, and working angle (see §5.10.4)
`propag_system`	Propagates planet time-dependant parameters (position, velocity, distance, separation, exozodiacal brightness, delta magnitude, and working angle) in time (see §5.10.5)
`set_planet_phase`	
`dump_systems`	Create a dictionary of planetary properties for archiving use (see §5.10.7)
`dump_system_params`	Create a dictionary of time-dependant planet properties for a specific target (see §5.10.8)
`revise_planets_list`	Replaces Simulated Universe planet attributes with filtered values, and updates the number of planets (see §5.10.9)
`revise_stars_list`	Revises the TargetList with filtered values, and updates the planets list accordingly (see §5.10.10)

5.10.1 Attributes

StarCatalog (StarCatalog module)
StarCatalog class object (only retained if keepStarCatalog is True, see 5.1)

PlanetPopulation (PlanetPopulation module)
PlanetPopulation class object (see 5.2)

PlanetPhysicalModel (PlanetPhysicalModel module)
PlanetPhysicalModel class object (see 5.3)

OpticalSystem (OpticalSystem module)
OpticalSystem class object (see 5.4)

ZodiacalLight (ZodiacalLight module)
ZodiacalLight class object (see 5.5)

BackgroundSources (BackgroundSources module)
BackgroundSources class object (see 5.6)

PostProcessing (PostProcessing module)
PostProcessing class object (see 5.7)

Completeness (Completeness module)
Completeness class object (see 5.8)

TargetList (TargetList module)
TargetList class object (see 5.9)

nPlans (integer)
Total number of planets

plan2star (integer ndarray)
Indices mapping planets to target stars in TargetList

sInds (integer ndarray)
Unique indices of stars with planets in TargetList

a (astropy Quantity array)
Planet semi-major axis in units of *AU*

e (float ndarray)
Planet eccentricity

I (astropy Quantity array)
Planet inclination in units of *deg*

O (astropy Quantity array)
Planet right ascension of the ascending node in units of *deg*

w (astropy Quantity array)
Planet argument of perigee in units of *deg*

Min (float)
Constant initial mean anomaly for all planets (optional)

M0 (astropy Quantity array)
Initial mean anomaly in units of *deg*

p (float ndarray)
Planet albedo

Rp (astropy Quantity array)
Planet radius in units of *earthRad*

Mp (astropy Quantity array)
Planet mass in units of *earthMass*

r (astropy Quantity n×3 array)
Planet position vector in units of *AU*

v (astropy Quantity n×3 array)
Planet velocity vector in units of AU/day

d (astropy Quantity array)
Planet-star distances in units of *AU*

s (astropy Quantity array)
Planet-star apparent separations in units of *AU*

phi (float ndarray)
Planet phase function, given its phase angle

fEZ (astropy Quantity array)
Surface brightness of exozodiacal light in units of $1/arcsec2$, determined from `ZodiacalLight.fEZ` §5.5.3

dMag (float ndarray)
Differences in magnitude between planets and their host star

WA (astropy Quantity array)
Working angles of the planets of interest in units of *arcsec*

cachedir (string)
Path to cache directory

5.10.2 gen_physical_properties Method

The `gen_physical_properties` method generates the planetary systems for the current simulated universe. This routine populates arrays of the orbital elements and physical characteristics of all planets, and generates indexes that map from planet to parent star. This method does not take any explicit inputs. It uses the inherited TargetList and PlanetPopulation modules.

Generated Module Attributes

nPlans (integer)
Total number of planets

plan2star (integer ndarray)
Indices mapping planets to target stars in TargetList

sInds (integer ndarray)
Unique indices of stars with planets in TargetList

a (astropy Quantity array)
Planet semi-major axis in units of *AU*

e (float ndarray)
Planet eccentricity

I (astropy Quantity array)
Planet inclination in units of *deg*

O (astropy Quantity array)
Planet right ascension of the ascending node in units of *deg*

w (astropy Quantity array)
Planet argument of perigee in units of *deg*

M0 (astropy Quantity array)
Initial mean anomaly in units of *deg*

p (float ndarray)
Planet albedo

Mp (astropy Quantity array)
Planet mass in units of *earthMass*

Rp (astropy Quantity array)
Planet radius in units of *earthRad*

5.10.3 gen_M0 Method

The `gen_M0` method generates the constant or random initial mean anomaly for each simulated planet inside of `gen_physical_properties`. This method does not take any explicit inputs.

Output

M0 (astropy Quantity array)
Initial mean anomalies in units of *deg*

5.10.4 init_systems Method

The `init_systems` method finds initial time-dependant parameters. It assigns each planet an initial position, velocity, planet-star distance, apparent separation, phase function, delta magnitude, working angle, surface brightness of exo-zodiacal light, and initializes the planet current times to zero. This method does not take any explicit inputs. It uses the following attributes assigned before calling this method:

```
SimulatedUniverse.a
SimulatedUniverse.e
SimulatedUniverse.I
SimulatedUniverse.O
SimulatedUniverse.w
SimulatedUniverse.M0
SimulatedUniverse.p
SimulatedUniverse.Mp
SimulatedUniverse.Rp
TargetList.MV
```

`TargetList.dist`

Generated Module Attributes

r (astropy Quantity n$\times$3 array)
: Planet position vector in units of *AU*

v (astropy Quantity n$\times$3 array)
: Planet velocity vector in units of AU/day

d (astropy Quantity array)
: Planet-star distances in units of *AU*

s (astropy Quantity array)
: Planet-star apparent separations in units of *AU*

phi (float ndarray)
: Planet phase function given its phase angle, determined from `PlanetPhysicalModel.calc_Phi` §5.3

fEZ (astropy Quantity array)
: Surface brightness of exozodiacal light in units of $1/arcsec2$, determined from `ZodiacalLight.fEZ` §5.5.3

dMag (float ndarray)
: Differences in magnitude between planets and their host star

WA (astropy Quantity array)
: Working angles of the planets of interest in units of *arcsec*

5.10.5 propag_system Method

The `propag_system` method propagates planet time-dependant parameters: position, velocity, planet-star distance, apparent separation, and surface brightness of exo-zodiacal light.

Input

sInd (integer)
: Index of the target system of interest

dt (astropy Quantity)
: Time increment in units of *day*, for planet position propagation

Updated Module Attributes

r (astropy Quantity n$\times$3 array)
: Planet position vector in units of *AU*

v (astropy Quantity n$\times$3 array)
: Planet velocity vector in units of AU/day

d (astropy Quantity array)
: Planet-star distances in units of *AU*

s (astropy Quantity array)
: Planet-star apparent separations in units of *AU*

phi (float ndarray)
: Planet phase function given its phase angle, determined from `PlanetPhysicalModel.calc_Phi` §5.3

fEZ (astropy Quantity array)
: Surface brightness of exozodiacal light in units of $1/arcsec2$, determined from `ZodiacalLight.fEZ` §5.5.3

dMag (float ndarray)
: Differences in magnitude between planets and their host star

WA (astropy Quantity array)
: Working angles of the planets of interest in units of *arcsec*

5.10.6 set_planet_phase Method

Input

beta (float)
: star-planet-observer phase angle in radians

Updates Attributes

r (astropy Quantity n$\times$3 array)
Planet position vector in units of *AU*

v (astropy Quantity n$\times$3 array)
Planet velocity vector in units of AU/day

d (astropy Quantity array)
Planet-star distances in units of *AU*

s (astropy Quantity array)
Planet-star apparent separations in units of *AU*

phi (float ndarray)
Planet phase function, given its phase angle

fEZ (astropy Quantity array)
Surface brightness of exozodiacal light in units of $1/arcsec2$, determined from `ZodiacalLight.fEZ` §5.5.3

dMag (float ndarray)
Differences in magnitude between planets and their host star

WA (astropy Quantity array)
Working angles of the planets of interest in units of *arcsec*

5.10.7 dump_systems Method

The `dump_systems` method creates a dictionary of planetary properties for archiving use.

Output

systems (dict)
Dictionary of planetary properties

5.10.8 dump_system_params Method

The `dump_system_params` method creates a dictionary of time-dependant planet properties for a specific target.

Input

sInd (integer)
Index of the target system of interest

Output

systems (dict)
Dictionary of time-dependant planet properties

5.10.9 revise_planets_list Method

The `revise_planets_list` method replaces Simulated Universe planet attributes with filtered values, and updates the number of planets.

Input

pInds (integer ndarray)
Planet indices to keep

5.10.10 revise_stars_list Method

The `revise_stars_list` method revises the TargetList with filtered values, and updates the planets list accordingly.

Input

sInds (integer ndarray)
Star indices to keep

5.11 Observatory

The Observatory module contains all of the information specific to the space-based observatory not included in the Optical System module. The module has four main methods: `orbit`, `solarSystem_body_position`, `keepout`, and `distForces`, which are implemented as functions within the module.

The observatory orbit plays a key role in determining which of the target stars may be observed for planet finding at a specific time during the mission lifetime. The Observatory module's `orbit` method takes the current mission time as input and outputs the observatory's position vector. The position vector is standardized throughout the modules to be referenced to a heliocentric equatorial frame at the J2000 epoch. The observatory's position vector is used in the `keepout` method to determine which of the stars are observable at the current mission time.

The position vectors of bright objects such as the sun, the moon, and the solar system planets, are calculated by the `solarSystem_body_position` method.

The `keepout` method determines which target stars are observable at a specific time during the mission simulation and which are unobservable due to bright objects within the field of view. The keepout volume is determined by the specific design of the observatory and, in certain cases, by the starlight suppression system. The `keepout` method takes the Target List module and current mission time as inputs and outputs a list of the target stars which are observable at the current time. It constructs position vectors of the target stars and bright objects which may interfere with observations with respect to the observatory. These position vectors are used to determine if bright objects are in the field of view for each of the potential stars under exoplanet finding observation. If there are no bright objects obstructing the view of the target star, it becomes a candidate for observation in the Survey Simulation module. The solar keepout is typically encoded as allowable angle ranges for the spacecraft-star unit vector as measured from the spacecraft-sun vector.

Finally, the `distForces` method determines the lateral and axial disturbance forces that apply on an external occulter (i.e., starshade).

In addition to these methods, the observatory definition can also encode finite resources used by the observatory throughout the mission. The most important of these is the fuel used for stationkeeping and repointing, especially in the case of occulters which must move significant distances between observations. Other considerations could include the use of other volatiles such as cryogens for cooled instruments, which tend to deplete solely as a function of mission time. This module also allows for detailed investigations of the effects of orbital design on the science yield, e.g., comparing the original baseline geosynchronous 28.5° inclined orbit for WFIRST with an L2 halo orbit, which is the new mission baseline.

5.11.1 Observatory Object Attribute Initialization

Input

koAngleMin (float)
Telescope minimum keepout angle in units of *deg*. Default value is 45.

koAngleMinMoon (float)
Telescope minimum keepout angle in units of *deg*, for the Moon only. Defaults to koAngleMin.

koAngleMinEarth (float)
Telescope minimum keepout angle in units of *deg*, for the Earth only. Defaults to koAngleMin.

koAngleMax (float)
Telescope maximum keepout angle (for occulter) in units of *deg*. Default value is 90.

koAngleSmall (float)
Telescope keepout angle for smaller (angular size) bodies in units of *deg*. Default value is 1.

ko_dtStep (float)
time step for generating koMap of stars in units of *day*

settlingTime (float)
Amount of time needed for observatory to settle after a repointing in units of *day*. Default value is 1.

thrust (float)
Occulter slew thrust in units of *mN*. Default value is 450.

slewIsp (float)
Occulter slew specific impulse in units of *s*. Default value is 4160.

scMass (float)
Occulter (maneuvering spacecraft) initial wet mass in units of *kg*. Default value is 6000.

dryMass (float)
Occulter (maneuvering spacecraft) dry mass in units of *kg*. Default value is 3400.

coMass (float)
Telescope (or non-maneuvering spacecraft) mass in units of *kg*. Default value is 5800.

occulterSep (float)
Occulter-telescope distance in units of *km*. Default value is 55000.

skIsp (float)
Specific impulse for station keeping in units of *s*. Default value is 220.

defburnPortion (float)
Default burn portion for slewing. Default value is 0.05

constTOF (float)
Occulter constant slewtime in *days*

maxdVpct (float)
Occulter maximum dV as percentage of total (must be ¡= 1)

spkpath (string)
String with full path to SPK kernel file (only used if using jplephem for solar system body propagation - see 5.11.10).

checkKeepoutEnd (boolean)
Boolean signifying if the keepout method must be called at the end of each observation.

forceStaticEphem (boolean)
Boolean, forcing use of static solar system ephemeris if set to True, even if jplephem module is present (see 5.11.10). Default value is False.

occ_dtmin (float)
Maximum occulter slew time (days)

occ_dtmax (float)
Minimum occulter slew time (days)

Attributes

koAngleMin (astropy Quantity)
Telescope minimum keepout angle in units of *deg*

koAngleMinMoon (astropy Quantity)
Telescope minimum keepout angle in units of *deg*, for the Moon only

koAngleMinEarth (astropy Quantity)
Telescope minimum keepout angle in units of *deg*, for the Earth only

koAngleMax (astropy Quantity)
Telescope maximum keepout angle (for occulter) in units of *deg*

koAngleSmall (astropy Quantity)
Telescope keepout angle for smaller (angular size) bodies in units of *deg*

ko_dtStep (float)
time step for generating koMap of stars in units of *day*

settlingTime (astropy Quantity)
Amount of time needed for observatory to settle after a repointing in units of *day*

thrust (astropy Quantity)
Occulter slew thrust in units of *mN*

slewIsp (astropy Quantity)
Occulter slew specific impulse in units of *s*

scMass (astropy Quantity)
Occulter (maneuvering spacecraft) initial wet mass in units of *kg*

dryMass (astropy Quantity)
Occulter (maneuvering spacecraft) dry mass in units of *kg*

coMass (astropy Quantity)
Telescope (or non-maneuvering spacecraft) mass in units of *kg*

occulterSep (astropy Quantity)
Occulter-telescope distance in units of *km*

skIsp (astropy Quantity)
Specific impulse for station keeping in units of *s*

defburnPortion (float)
Default burn portion for slewing

checkKeepoutEnd (boolean)
Boolean signifying if the keepout method must be called at the end of each observation

forceStaticEphem (boolean)
Boolean, forcing use of static solar system ephemeris if set to True.

constTOF (float)

Constant occulter slew time in *day*

occ_dtmin (float)
Minimum occulter slew time in *day*

occ_dtmax (float)
Maximum occulter slew time in *day*

maxdVpct (float)
Maximum slew dV as percentage of initial dV

dVtot (float)
Total amount of dV onboard the spacecraft

dVmax (float)
Maximum dV per occulter slew

flowRate (float)
Slew flow rate in kg/day

havejplephem (boolean)
Boolean indicating static solar ephemerides are being used when False

obe (lambda function)
A lambda function for calculating the obliquity of the ecliptic

kernel (jplephem object)
The spk ephemeris, exists only if havejplephem is True

planets (list of dict)
A dictionary containing the planets names

cachedir (string)
Path to cache directory

5.11.2 equat2eclip Method

Rotates heliocentric coordinates from equatorial frame to ecliptic frame

Input

r_equat (astropy Quantity nx3 array)
Positions vector in heliocentric equatorial frame in units of AU

currentTime (astropy Time array)
Current absolute mission time in MJD

rotsign (integer)
Optional flag, default 1, set -1 to reverse the rotation

Output

r_eclip (qstropy Quantity nx3 array)
Positions vector in heliocentric ecliptic frame in units of AU

5.11.3 eclip2equat Method

Rotates heliocentric coordinates from ecliptic to equatorial frame.

Input

r_eclip (astropy Quantity nx3 array)
Positions vector in heliocentric ecliptic frame in units of AU

currentTime (astropy Time array)
Current absolute mission time in MJD

Output

r_equat (astropy Quantity nx3 array)
Positions vector in heliocentric equatorial frame in units of AU

5.11.4 orbit Method

The `orbit` method finds the heliocentric equatorial position vector of the observatory spacecraft.

Input

currentTime (astropy Time array)
Current absolute mission time in MJD

Output

r_sc (astropy Quantity n×3 array)
Observatory orbit position in HE reference frame at current mission time in units of *km*

5.11.5 keepout Method

The `keepout` method determines which stars in the target list are observable at the given input time.

Input

TL (TargetList module)
TargetList class object, see §5.9 for definition of available attributes

sInds (integer ndarray)
Integer indices of the stars of interest

currentTime (astropy Time array)
Current absolute mission time in MJD

mode (dict)
Selected observing mode (from OpticalSystem)

Output

kogood (boolean ndarray)
True is a target unobstructed and observable, and False is a target unobservable due to obstructions in the keepout zone.

5.11.6 generate_koMap Method

This function creates a keepout map for all targets from missionStart until the end of mission.

Input

TL (module)
The Target List module

missionStart (astropy Time)
The absolute start time of mission in MJD

missionFinishAbs
The absolute end time of mission in MJD

Output

koMap (boolean ndarray)
A binary array with TargetList.nStars rows and (missionFinishAbs-missionStart)/dt columns where dt is 1 day by default. A value of 1 indicates the star is in keepout for (and therefore cannot be observed). A value of 0 indicates the star is not in keepout and may be observed.

koTimes (astropy Time ndarray)
An array of astropy Times corresponding to each column in koMap. Ex koMap[0,7] maps to koTimes[7]

5.11.7 calculate_observableTimes Method

For Occulters and No Occulter systems. Calculates the next window of observable times for a given set of stars specified by sInds such that the returned observableTimes has dimensions len(sInds)x2. If sInds has one star such that sInds=[0], then the returned array observableTimes is 1x2 where observableTimes[0,0] is the start time the next star will be observable and observableTimes[0,1] is the end time the next star will be observable.

Input

TL (module)
The Target List module

sInds (numpy array)
The indices of the stars to calculate observable Times for
currentTime (astropy Time)
The current absolute mission time in MJD
koMap
The keepout map generated by generate_koMap method
koTimes
The keepout map Times associated with the keepout map
mode (list dict)
the operating mode of the instrument used for determining whether occulter needs to be accounted for.

Output
observableTimes (astropy Time list)
A binary array with TargetList.nStars rows and (missionFinishAbs-missionStart)/dt columns where dt is 1 day by default. A value of 1 indicates the star is in keepout for (and therefore cannot be observed). A value of 0 indicates the star is not in keepout and may be observed.
koTimes (astropy Time ndarray)
An array of astropy Times corresponding to each column in koMap. Ex koMap[0,7] maps to koTimes[7]

5.11.8 find_nextObsWindow Method

This function should only be called by calculate_observableTimes. *For No Occulter systems.Calculates the next window of observable times for a given set of stars specified by sInds such that the returned observableTimes has dimensions len(sInds)x2. If sInds has one star such that sInds=[0], then the returned array observableTimes is 1x2 where observableTimes[0,0] is the start time the next star will be observable and observableTimes[0,1] is the end time the next star will be observable.

Input
TL (module)
The Target List module
sInds (numpy array)
The indices of the stars to calculate observable Times for
currentTime (astropy Time)
The current absolute mission time in MJD
koMap
The keepout map generated by generate_koMap method
koTimes
The keepout map Times associated with the keepout map

Output
observableTimes (astropy Time list)
A binary array with TargetList.nStars rows and (missionFinishAbs-missionStart)/dt columns where dt is 1 day by default. A value of 1 indicates the star is in keepout for (and therefore cannot be observed). A value of 0 indicates the star is not in keepout and may be observed.
koTimes (astropy Time ndarray)
An array of astropy Times corresponding to each column in koMap. Ex koMap[0,7] maps to koTimes[7]

5.11.9 star_angularSep Method

Finds angular separation from old star to given list of stars.

Input
TL (TargetList module)
TargetList class object
old_sInd (integer)
Integer index of the last star of interest

sInds (integer ndarray)
Integer indices of the stars of interest

currentTime (astropy Time array)
Current absolute mission time in *MJD*

Output

sd (integer)
Angular separation mission time in *MJD*

5.11.10 solarSystem_body_position Method

The `solarSystem_body_position` returns the position of any solar system body (Earth, Sun, Moon, etc.) at a given time in the common Heliocentric Equatorial frame. The observatory prototype will attempt to load the jplephem module, and use a local SPK file for all propagations if available. The SPK file is not packaged with the software but may be downloaded from JPL's website at: `http://naif.jpl.nasa.gov/pub/naif/generic_kernels/spk/planets/a_old_versions/`. The location of the spk file is assumed to be in the Observatory directory but can be set by the `spkpath` input.

If jplephem is not present, the Observatory prototype will load static ephemeris derived from Vallado (2004) and use those for propagation. This behavior can be forced even when jplephem is available by setting the `forceStaticEphem` input to True.

Input

currentTime (astropy Time)
Current absolute mission time in MJD

bodyname (string)
Solar system object name, capitalized by convention

Output

r_body (astropy Quantity n$\times$3 array)
Heliocentric equatorial position vector in units of *km*

5.11.11 spk_body Method

Finds solar system body positions vector in heliocentric equatorial (default) or ecliptic frame for current time (MJD).

Input

currentTime (astropy Time array)
Current absolute mission time in *MJD*

bodyname (string)
Solar system object name

eclip (boolean)
Boolean used to switch to heliocentric ecliptic frame. Defaults to False, corresponding to heliocentric equatorial frame.

Output

r_body (astropy Quantity nx3 array)
Solar system body positions in heliocentric equatorial (default) or ecliptic frame in units of *AU*

5.11.12 keplerplanet Method

Finds solar system body positions vector in heliocentric equatorial (default) or ecliptic frame for current time (MJD).

Input

currentTime (astropy Time array)
Current absolute mission time in *MJD*

bodyname (string)
Solar system object name

eclip (boolean)
Boolean used to switch to heliocentric ecliptic frame. Defaults to False, corresponding to heliocentric equatorial frame.

Output

r_body (astropy Quantity nx3 array)
Solar system body positions in heliocentric equatorial (default) or ecliptic frame in units of AU

5.11.13 moon_earth Method

Finds geocentric equatorial positions vector for Earth's moon using Algorithm 31 from Vallado 2013 to find the geocentric equatorial positions vector for Earth's moon.

Input

currentTime (astropy Time array)
Current absolute mission time in *MJD*

Output

r_moon (astropy Quantity nx3 array)
Geocentric equatorial position vector in units of AU

5.11.14 cent Method

Finds time in Julian centuries since J2000 epoch. This quantity is needed for many algorithms from Vallado 2013.

Input

currentTime (astropy Time array)
Current absolute mission time in *MJD*

Output

TBD (float ndarray)
time in Julian centuries since the J2000 epoch

5.11.15 propeph Method

Propagates an ephemeris from Vallado 2013 to current time.

Input

x (list)
ephemeride list (maximum of 4 elements)

TBD (float)
time in Julian centuries since the J2000 epoch

Output

y (float ndarray)
ephemeride value at current time

5.11.16 rot Method

Finds the rotation matrix of angle th about the axis value

Input

th (float)
Rotation angle in radians

axis (int)
Integer value denoting rotation axis (1,2, or 3)

Output

rot_th (float 3x3 ndarray)
Rotation matrix

5.11.17 distForces Method

The `distForces` method finds lateral and axial disturbance forces on an occulter.

Input

TL (TargetList module)
TargetList class object, see §5.9 for definition of available attributes

sInd (integer)
Integer index of the star of interest

currentTime (astropy Time array)
Current absolute mission time in MJD

Output

dF_lateral (astropy Quantity)
Lateral disturbance force in units of *N*

dF_axial (astropy Quantity)
Axial disturbance force in units of *N*

5.11.18 mass_dec Method

Returns mass used and deltaV, the values returned by this method are used to decrement spacecraft mass for station-keeping.

Input

dF_lateral (astropy Quantity)
Lateral disturbance force in units of *N*

t_int (astropy Quantity)
Integration time in units of *day*

Output

intMdot (astropy Quantity)
Mass flow rate in units of kg/s

mass_used (astropy Quantity)
Mass used in station-keeping units of kg

deltaV (astropy Quantity)
Change in velocity required for station-keeping in units of km/s

5.11.19 mass_dec_sk Method

Returns mass used, deltaV and disturbance forces. This method calculates all values needed to decrement spacecraft mass for station-keeping.

Input

TL (TargetList module)
TargetList class object

sInd (integer)
Integer index of the star of interest

currentTime (astropy Time array)
Current absolute mission time in *MJD*

t_int (astropy Quantity)
Integration time in units of *day*

Output

dF_lateral (astropy Quantity)
Lateral disturbance force in units of *N*

dF_axial (astropy Quantity)
Axial disturbance force in units of *N*

intMdot (astropy Quantity)
Mass flow rate in units of kg/s

mass_used (astropy Quantity)
Mass used in station-keeping units of kg

deltaV (astropy Quantity)
Change in velocity required for station-keeping in units of km/s

5.11.20 calculate_dV Method

Finds the change in velocity needed to transfer to a new star line of sight. This method sums the total delta-V needed to transfer from one star line of sight to another. It determines the change in velocity to move from one station-keeping orbit to a transfer orbit at the current time, then from the transfer orbit to the next station-keeping orbit at currentTime + dt. Station-keeping orbits are modeled as discrete boundary value problems. This method can handle multiple indeces for the next target stars and calculates the dVs of each trajectory from the same starting star.

Input

dt (float 1x1 ndarray)
Number of days corresponding to starshade slew time

TL (float 1x3 ndarray)
TargetList class object

nA (integer)
Integer index of the current star of interest

N (integer)
Integer index of the next star(s) of interest

tA (astropy Time array)
Current absolute mission time in *MJD*

Output

dV (float nx6 ndarray)
State vectors in rotating frame in normalized units

5.11.21 calculate_slewTimes Method

Finds slew times and separation angles between target stars. This method determines the slew times of an occulter spacecraft needed to transfer from one star's line of sight to all others in a given target list.

Input

TL (TargetList module)
TargetList class object

old_sInd (integer)
Integer index of the most recently observed star

sInds (integer ndarray)
Integer indeces of the star of interest

sd (astropy Quantity)
Angular separation between stars in rad

observableTimes (astropy Time list)
A binary array with TargetList.nStars rows and (missionFinishAbs-missionStart)/dt columns where dt is 1 day by default. A value of 1 indicates the star is in keepout for (and therefore cannot be observed). A value of 0 indicates the star is not in keepout and may be observed.

currentTime (astropy Time array)
Current absolute mission time in *MJD*

Output

slewTimes (astropy Quantity)

Time to transfer to new star line of sight in units of *days*

5.11.22 log_occulterResults Method

Updates the given DRM to include occulter values and results

Input

DRM (dict)

Design Reference Mission, contains the results of one complete observation (detection and characterization)

slewTimes (astropy Quantity)

Time to transfer to new star line of sight in units of days

sInd (integer)

Integer index of the star of interest

sd (astropy Quantity)

Angular separation between stars in rad

dV (astropy Quantity)

Delta-V used to transfer to new star line of sight in units of m/s

Output

DRM (dict)

Design Reference Mission, contains the results of one complete observation (detection and characterization)

5.12 Time Keeping

The Time Keeping module is responsible for keeping track of the current mission time, exoplanet observation time, and observing blocks. It encodes the mission start time, mission life, mission portion, mission end time, current time within a simulation, start times of observing blocks, end times of observing blocks, observation block number, and observation time counted towards an instrument. All functions in all modules requiring knowledge of the current time call functions or access parameters implemented within the Time module. Internal encoding of time is implemented as the time from mission start (measured in units of *day*). The Time Keeping module also provides functionality for converting between this time measure and standard measures such as Julian Day Number and UTC time.

The Time Keeping module contains six methods:

`init_OB`	Initializes the mission observing blocks (see §5.12.2)
`mission_is_over`	Evaluates whether the mission is over given the current timekeeping values (see §5.12.6)
`allocate_time`	Allocates a temporal block of width *dt*, updating the mission time during a survey simulation (see §5.12.3)
`advanceToAbsTime`	Advances mission time to absolute time specified accounting for observation blocks (see §5.12.4)
`advancetToStartOfNextOB`	Advances mission time to the start of the next Observing Block (see §5.12.5)
`get_ObsDetectionMaxIntTime`	Returns the maximum integration times that can be passed into the prototype observation_detection (see §5.13.6) as limited by observing blocks, exoplanet observation time, and mission life (see §5.12.7)

5.12.1 Time Keeping Object Attribute Initialization

Input

missionStart (float)

Mission start time in *MJD*. Default value is 60634.

missionLife (float)

Total length of mission in units of *year*. Default value is 0.1.

missionPortion (float)

Portion of mission time devoted to planet-finding. Default value is 1.

OBduration (float)

Default allocated duration of observing blocks in units of *day*. Default value is np.inf.

missionSchedule (string)

Name of CSV file containing Observation Blocks start and end times.

Attributes

missionStart (astropy Time)
Mission start time in *MJD*

missionLife (astropy Quantity)
Mission lifetime in units of *year*

missionPortion (float)
Portion of mission time devoted to planet-finding

missionFinishAbs (astropy Time)
Mission completion date in *MJD*

currentTimeNorm (astropy Quantity)
Current mission time normalized so that start date is 0, in units of *day*

currentTimeAbs (astropy Time)
Current absolute mission time in *MJD*

OBnumber (integer)
Index/number associated with the current observing block (OB). Each observing block has a duration, a start time, an end time, and can host one or multiple observations.

OBduration (astropy Quantity)
Default allocated duration of observing blocks, in units of *day*. If no OBduration was specified, a new observing block is created for each new observation in the SurveySimulation module.

OBstartTimes (astropy Quantity array)
Array containing the normalized start times of each observing block throughout the mission, in units of *day*

OBendTimes (astropy Quantity array)
Array containing the normalized end times of each observing block throughout the mission, in units of *day*

exoplanetObsTime (astropy Quantity)
Tracks instrument time used in units of *day*

cachedir (string)
Path to cache directory

5.12.2 init_OB Method

The init_OB method is ONLY called to instantiate the observing blocks at the beginning of the mission. The function takes in the name of the CSV format file containing the missionSchedule (stored in the Scripts folder) and the OBduration. Both missionSchedule and OBduration are designed to be defined in the JSON script. This method initializes mission observing blocks in priority order:

1. A CSV file specified in the missionSchedule field of the JSON script. The observing blocks are exactly specified by the start and end times in the CSV file.
2. The OBduration, missionPortion, and missionLife specifications in the script (automatically creating observing blocks of length OBduration and constant period repetition)
3. Defaults to single observing block with duration missionLife $\times$ missionPortion

Input

missionSchedule (string)
Filename of a CSV file containing the mission observation block start and end times which are presented in units of day (defaults to None)

OBduration (astropy Quantity)
the duration of a single observation block, used if a mission schedule is not provided. (units of *day*)

Updated Module Attributes

OBstartTimes (astropy Quantity array)
contains the start times of every observation block specified as times since mission start

OBendTimes (astropy Quantity array)
contains the end times of every observation block specified as times since mission start

OBnumber (integer)
the index indicating the current observation block the mission is in ex. OBstartTimes[OBnumber]

5.12.3 allocate_time Method

Allocate a continguous temporal block of width dt. If time advancement would place the mission time outside of an observing block, past the end of mission, or exceed exoplanet observation time, then a false flag will be returned and time will not advanced by dt. allocate_time is designed to be called by observation characterization and observation detection methods. Observation characterization will skip characterization if time allocation fails. Observation detection produces a global error if time allocation fails. The maximum integration time returnable by next_target can be retrieved by calling get_ObsDetectionMaxIntTime.

Input

dt (astropy Quantity)
Temporal block allocated in units of *day*

addExoplanetObsTime (bool)
A True flag indicates dt to allocate is to be counted towards exoplanet observation time otherwise no time is added to exoplanet observation time (default is True)

Updated Module Attributes

currentTimeNorm (astropy Quantity)
Current mission time normalized so start date is 0, in units of *day* is advanced by dt

currentTimeAbs (astropy Time)
Current absolute mission time in *MJD* in advanced by dt

exoplanetObsTime (astropy Quantity)
exoplanet Obs time is advanced by dt if time allocation does not fail and addExoplanetObsTime flag is True

Output

success (boolean)
True if dt could be allocated. False if allocated time exceeds the end of mission time, the end of the observing block, exceeds the total allowed mission time.

5.12.4 advanceToAbsTime Method

Advances mission time to the specified absolute time. If the absolute time to advance to is outside of an observing block, time will advance to the start of the next observing block. The absolute time to be advanced to can be past the mission finish time. The addExoplanetObsTime flag indicates whether time within observing blocks between now and the time to advance to will count towards exoplanet observation time.

Input

tAbs (astropy Time)
Absolute Time to be advanced to in *MJD*

addExoplanetObsTime (bool)
A True flag indicates t to allocate is to be counted towards exoplanet observation time otherwise no time is added to exoplanet observation time (default is True)

Updated Module Attributes

currentTimeNorm (astropy Quantity)
Current mission time normalized so that start date is 0, in units of *day*

currentTimeAbs (astropy Time)
Current absolute mission time in *MJD*

exoplanetObsTime (astropy Quantity)
exoplanet Obs time is advanced by dt if time allocation does not fail and addExoplanetObsTime flag is True

Output

success (bool)
False if advancing tAbs would exceed exoplantObsTime. True under all other conditions

5.12.5 advancetToStartOfNextOB Method

Advances to Start of Next Observation Block. This method is called from run_sim following an allocate_time or advanceToAbsTime function call. Since allocate_time is designed to allocate time up and to OBendTimes[OBnumber], advancetToAbsTime is a mechanism for advancing time to the start of the next OB.

Updated Module Attributes

OBnumber (integer)
Index/number associated with the current observing block (OB). Each observing block has a duration, a start time, an end time, and can host one or multiple observations.

currentTimeNorm (astropy Quantity)
Current mission time normalized so that start date is 0, in units of *day*

currentTimeAbs (astropy Time)
Current absolute mission time in *MJD*

5.12.6 mission_is_over Method

The `mission_is_over` method does not take any explicit inputs. It compares the updated module attribute `currentTimeNorm` to `missionLife` and last `OBendTimes` for termination conditions. We also compare `exoplanetObsTime` to the total mission time dedicates to observations.

Input

Obs (Observatory object)
Observatory module for use of Obs.settlingTime

mode (dict)
mode dictionary for ohTime

Output

is_over (boolean)
True if the mission time is used up, else False

5.12.7 get_ObsDetectionMaxIntTime

Calculates the maximum integration times possible without exceeding allocate_time scheduling constraints: 1) OBendTime, 2) exoplanetObsTime, and 3) missionLife. The calculation includes the time multiplier, system settling time, and system overhead time.

Input

Obs (Observatory object)
Observatory module for use of Obs.settlingTime

mode (dict)
mode dictionary for ohTime

Output

maxIntTimeOBendTime (astropy Quantity)
The maximum integration time that can be passed into observation detection without exceeding the observing block end time in units of *day*

maxIntTimeExoplanetObsTime (astropy Quantity)
The maximum integration time that can be passed into observation detection without exceeding the exoplanet observation time in units of *day*

maxIntTimeMissionLife (astropy Quantity)
The maximum integration time that can be passed into observation detection without exceeding the mission life in units of *day*

5.13 Survey Simulation

This is the module that performs a specific simulation based on all of the input parameters and models. This module returns the mission timeline - an ordered list of simulated observations of various targets on the target list along with their outcomes. The output also includes an encoding of the final state of the simulated universe (so that a subsequent simulation can start from where a previous simulation left off) and the final state of the observatory definition (so that post-simulation analysis can determine the percentage of volatiles expended, and other engineering metrics).

The Survey Simulation module contains the following methods:

`run_sim`	Performs the survey simulation (see §5.13.2)
`next_target`	Finds index of next target star and calculates its integration time (see §5.13.3) - called by `run_sim`
`calc_targ_intTime`	Calculate integration times for a subset of targets give the start of observation time (see §5.13.4) - called by `intTimeFilter`
`choose_next_target`	Given a subset of targets, select the next target to observe OR specify some waitTime (see §5.13.5) - called by `next_target`
`observation_detection`	Determines detection status for a given integration time (see §5.13.6)
`observation_characterization`	Determines characterization time and status (see §5.13.8)
`calc_signal_noise`	Calculates the signal and noise fluxes for a given time interval (see §5.13.9) - called by `observation_detection` and `observation_characterization`
`update_occulter_mass`	Updates the occulter wet mass in the Observatory module, and stores all the occulter related values in the DRM array (see §5.13.10) - called by `run_sim`
`reset_sim`	Performs a full reset of the simulation (see §5.13.11)
`genOutSpec`	Joins outspec dicts from all modules into one output dict (see §5.13.12)
`generateHashName`	Generates a hashname for cached files tied to this json script (see §5.13.13)
`revisitFilter`	Filters a list of target stars by revisits (see §5.13.14) - called by `next_target`

5.13.1 Survey Simulation Object Attribute Initialization

Input

scriptfile (string)
JSON script file. If not set, assumes that dictionary has been passed through specs.

nt_flux (integer)
Observation time sampling, to determine the integration time interval. Default value is 1.

nVisitsMax (integer)
Maximum number of observations (in detection mode) per star. Default value is 5.

charMargin (float)
Integration time margin for characterization. Default value is 15.

seed (integer)
Random seed used to make all random number generation reproducible.

WAint (float)
Working angle used for integration time calculation in units of *arcsec*.

dMagint (float)
Delta magnitude used for integration time calculation.

cachedir (float)
Path to desired cache directory (default will be `$HOME/.EXOSIMS/cache`).

Attributes

StarCatalog (StarCatalog module)
StarCatalog class object (only retained if keepStarCatalog is True, see 5.1)

PlanetPopulation (PlanetPopulation module)
PlanetPopulation class object (see 5.2)

PlanetPhysicalModel (PlanetPhysicalModel module)
PlanetPhysicalModel class object (see 5.3)

OpticalSystem (OpticalSystem module)
OpticalSystem class object (see 5.4)

ZodiacalLight (ZodiacalLight module)
ZodiacalLight class object (see 5.5)

BackgroundSources (BackgroundSources module)
BackgroundSources class object (see 5.6)

PostProcessing (PostProcessing module)
PostProcessing class object (see 5.7)

Completeness (Completeness module)
Completeness class object (see 5.8)
TargetList (TargetList module)
TargetList class object (see 5.9)
SimulatedUniverse (SimulatedUniverse module)
SimulatedUniverse class object (see 5.10)
Observatory (Observatory module)
Observatory class object (see 5.11)
TimeKeeping (TimeKeeping module)
TimeKeeping class object (see 5.12)
fullSpectra (boolean ndarray)
Indicates if planet spectra have been captured
partialSpectra (boolean ndarray)
Indicates if planet partial spectra have been captured
propagTimes (astropy Quantity array)
Contains the last time the stellar system was propagated in units of *day*
lastObsTimes (astropy Quantity array)
Contains the last observation start time for future completeness update in units of *day*
starVisits (integer ndarray)
Contains the number of times each target was visited
starRevisit (float n$\times$2 ndarray)
Contains indices of targets to revisit and revisit times of these targets in units of *day*
lastDetected (float n$\times$4 ndarray)
For each target, contains 4 lists with planets' detected status, exozodi brightness (in units of $1/arcsec^2$), delta magnitude, and working angles (in units of *arcsec*)
DRM (list of dicts)
Design Reference Mission, contains the results of a survey simulation
nt_flux (integer)
Observation time sampling, to determine the integration time interval
nVisitsMax (integer)
Maximum number of observations (in detection mode) per star
charMargin (float)
Integration time margin for characterization
seed (integer)
Random seed used to make all random number generation reproducible
WAint (astropy Quantity array)
Working angle used for integration time calculation in units of *arcsec*
dMagint (float ndarray)
Delta magnitude used for integration time calculation
valfZmin (astropy Quantity array)
value of minimum zodiacal light with units 1/arcsec^2
absTimefZmin (astropy Time array)
absolute time in MJD and tai that minimum zodiacal light will occur next
cachedir (string)
Path to cache directory

5.13.2 run_sim Method

The `run_sim` method uses the inherited modules to generate a survey simulation, without any explicit inputs and outputs. It updates module attributes and populates the results in `SurveySimulation.DRM`.

Updated Module Attributes

SurveySimulation.DRM
Python list where each entry contains a dictionary of survey simulation results for each observation. The dictionary may include the following `key:value` pairs (from the prototype):

star_ind (integer)
Index of the observed target star

star_name (string)
Name of the observed target star

arrival_time (float)
Elapsed time since mission start when observation begins in units of *day*

OB_nb (integer)
Number/index of the observing block

plan_inds (integer ndarray)
Indices of planets orbiting the observed target star

det_mode (dict)
Observing mode selected for detection

det_time (astropy Quantity)
Integration time for detection in units of *day*

det_status (integer ndarray)
List of detection status for each planet orbiting the observed target star, where 1 is detection, 0 missed detection, -1 below IWA, and -2 beyond OWA

det_SNR (float ndarray)
List of detection SNR values for each planet orbiting the observed target star. Non-observable planets have their SNR set to 0.

det_fZ (astropy Quantity)]
Zodiacal surface brightnesses at detection in units of $1/arcsec^2$

det_params (dict)
Dictionary of system params at detection for each planet orbiting the observed target star, including d (AU), phi, fEZ ($1/arcsec2$), dMag, and WA ($arcsec$)

char_mode (dict)
Observing mode selected for characterization

char_time (astropy Quantity)
Integration time for characterization in units of *day*

char_status (integer ndarray)
List of characterization status for each planet orbiting the observed target star, where 1 is full spectrum, -1 partial spectrum, and 0 not characterized

char_SNR (float ndarray)
List of characterization SNR values for each planet orbiting the observed target star. Non-observable planets have their SNR set to 0.

char_fZ (astropy Quantity)]
Zodiacal surface brightnesses at characterization in units of $1/arcsec^2$

char_params (dict)
Dictionary of system params at characterization for each planet orbiting the observed target star, including d (AU), phi, fEZ ($1/arcsec2$), dMag, and WA ($arcsec$)

FA_det_status (integer)
Detection status for a false alarm signal (1 is a false alarm, 0 is not)

FA_char_status (integer ndarray)
(if false alarm) Characterization status for a false alarm signal, where 1 is full spectrum, -1 partial spectrum, and 0 not characterized

FA_char_SNR (float ndarray)
(if false alarm) Characterization SNR value for a false alarm signal

FA_char_fEZ (astropy Quantity)
(if false alarm) Exo-zodi surface brightness for a false alarm signal in units of $1/arcsec^2$

FA_char_dMag (float)
(if false alarm) Delta magnitude for a false alarm signal

FA_char_WA (astropy Quantity)
(if false alarm) Working angle for a false alarm signal in units of *arcsec*

slew_time (astropy Quantity)
(if occulter) Slew time to next target in units of *day*

slew_angle (astropy Quantity)
(if occulter) Slew angle to next target in units of *deg*

slew_dV (astropy Quantity)
(if occulter) Slew ΔV in units of m/s

slew_mass_used (astropy Quantity)

(if occulter) Slew fuel mass used in units of *kg*

sc_mass (float)

(if occulter) Maneuvering spacecraft mass at the end of target observation in units of *kg*

det_dV (astropy Quantity)

(if occulter) Detection station-keeping ΔV in units of m/s

det_mass_used (astropy Quantity)

(if occulter) Detection station-keeping fuel mass used in units of *kg*

det_dF_lateral (astropy Quantity)

(if occulter) Detection station-keeping lateral disturbance force on occulter in units of *N*

det_dF_axial (astropy Quantity)

(if occulter) Detection station-keeping axial disturbance force on occulter in units of *N*

char_dV (astropy Quantity)

(if occulter) Characterization station-keeping ΔV in units of m/s

char_mass_used (astropy Quantity)

(if occulter) Characterization station-keeping fuel mass used in units of *kg*

char_dF_lateral (astropy Quantity)

(if occulter) Characterization station-keeping lateral disturbance force on occulter in units of *N*

char_dF_axial (astropy Quantity)

(if occulter) Characterization station-keeping axial disturbance force on occulter in units of *N*

5.13.3 next_target Method

The `next_target` method filters target stars available by keepout, integration time, and revisits finds index of next target star and calculates its integration time. This method chooses the next target star index based on which stars are available, their integration time, and maximum completeness. Also updates DRM. Returns None if no target could be found.

Input

old_sInd (integer)

Index of the previous target star (set to None for the first observation)

mode (dict)

Selected observing mode (from OpticalSystem)

Output

DRM (dict)

Dictionary containing survey simulation results

sInd (integer)

Index of next target star. Defaults to None.

det_intTime (astropy Quantity)

Selected star integration time for detection in units of *day*. Defaults to None.

5.13.4 calc_targ_intTime Method

The calc_targ_intTime method is a helper method for next_target to aid in overloading for alternative implementations.

Input

sInds (integer array)

Indicies of available targets

startTimes (astropy Quantity array)

absolute start times of observations. must be the same size as sInds

mode (dict)

Selected observing mode for detection (from OpticalSystem)

Output

intTimes (integer array)

Integration times for detection same dimension as sInds

5.13.5 choose_next_target Method

Helper method for method next_target to simplify alternative implementation.

Input

old_sInd (integer)
Index of previous target star

sInds (integer array)
Indices of available targets

slewTimes (astropy quantity array)
slew times to all stars (must be indexed by sInds)

intTimes (astropy Quantity array)
Integration times for detection in units of day

Output

sInd (integer)
Index of next target star

5.13.6 observation_detection Method

The `observation_detection` method determines the detection status and updates the last detected list and the revisit list.

Input

sInd (integer)
Integer index of the star of interest

intTime (astropy Quantity)
Selected star integration time in units of *day*. Defaults to None.

mode (dict)
Selected observing mode (from OpticalSystem)

Output

detected (integer ndarray)
Detection status for each planet orbiting the observed target star, where 1 is detection, 0 missed detection, -1 below IWA, and -2 beyond OWA

fZ (astropy Quantity)
Zodiacal surface brightnesses at detection in units of $1/arcsec^2$

systemParams (dict)
Dictionary of time-dependant planet properties averaged over the duration of the integration

SNR (float ndarray)
Detection signal-to-noise ratio of the observable planets

FA (boolean)
False alarm (false positive) boolean

5.13.7 scheduleRevisit Method

A Helper Method for scheduling revisits after observation detection

Input

sInd (integer)
index of the star just detected

smin (AU)
minimum separation of the planet to star of planet just detected

pInds (astropy Quantity array)
Indices of planets around target star

Output

updates starRevisit Attribute

5.13.8 observation_characterization Method

The `observation_characterization` method finds if characterizations are possible and relevant information.

Input

sInd (integer)
Integer index of the star of interest

mode (dict)
Selected observing mode (from OpticalSystem)

Output

characterized (integer ndarray)
Characterization status for each planet orbiting the observed target star including False Alarm if any, where 1 is full spectrum, -1 partial spectrum, and 0 not characterized

fZ (astropy Quantity)
Zodiacal surface brightnesses at characterization in units of $1/arcsec^2$

systemParams (dict)
Dictionary of time-dependant planet properties averaged over the duration of the integration

SNR (float ndarray)
Characterization signal-to-noise ratio of the observable planets

char_intTime (astropy Quantity)
Selected star characterization time in units of *day*

5.13.9 calc_signal_noise Method

The `calc_signal_noise` method calculates the signal and noise fluxes for a given time interval.

Input

sInd (integer)
Integer index of the star of interest

pInds (integer)
Integer indices of the planets of interest

t_int (astropy Quantity)
Integration time interval interval in units of *day*

mode (dict)
Selected observing mode (from OpticalSystem)

Output

Signal (float)
Counts of signal

Noise (float)
Counts of background noise variance

5.13.10 update_occulter_mass Method

The `update_occulter_mass` method updates the occulter wet mass in the Observatory module, and stores all the occulter related values in the DRM array.

Input

DRM (dicts)
Contains the results of survey simulation

sInd (integer)
Integer index of the star of interest

t_int (astropy Quantity)
Selected star integration time (for detection or characterization) in units of *day*
skMode (string)
Station keeping observing mode type

Output
DRM (dicts)
Contains the results of survey simulation

5.13.11 reset_sim Method
Performs a full reset of the current simulation:

1. Re-initializes the TimeKeeping object with its own outspec (i.e., `TimeKeeping._outspec` is passed directly to the TimeKeeping constructor)
2. If genNewPlanets is True (default) generates all new planets based on the original input specification. If genNewPlanets is False, then the original planets will remain. Setting to True forces rewindPlanets to be True as well.
3. If rewindPlanets is True (default), then the current set of planets will be reset to their original orbital phases. If both genNewPlanets and rewindPlanet are False, the original planets will be retained and will not be rewound to their initial starting locations (i.e., all systems will remain at the times they were at the end of the last run, thereby effectively randomizing planet phases).
4. If seed is None (default) Re-initializes the SurveySimulation object, including resetting the DRM to [] and the random seed will be reset. If seed is specified, the seed will be set to the specified seed.

5.13.12 genOutSpec Method
Join all _outspec dicts from all modules into one output dict and optionally write out to JSON file on disk.

Input
tofile (string)
Name of the file containing all output specifications (outspecs). Default to None.

Output
out (dictionary)
Dictionary containing additional user specification values and desired module names.

5.13.13 generateHashfName Method
Generate cached file Hashname

Input
specs
The json script elements of the simulation to be run

Output
cachefname (string)
a string containing the file location, hashnumber of the cache name based off of the completeness to be computed (completeness specs if available else standard module)

5.13.14 revisitFilter Method
Helper method for overloading revisit filtering in next_target

Input
sInds [sInds]
indices of stars still in observation list
tmpCurrentTimeNorm (MJD)
the simulation time after overhead was added in MJD form

Output

sInds [sInds (integer array)]
indces of the stars still in observation list

5.13.15 refineOcculterSlews Method

Refines the selection of occulter slew times by filtering based on mission time constraints and selecting the best slew time for each star. This method calls on other occulter methods within SurveySimulation depending on how slew times were calculated prior to calling this function (i.e. depending on which implementation of the Observatory module is used).

Input

old_sInd (integer)
Integer index of the most recently observed star

sInds [sInds]
indices of stars still in observation list

slewTimes (astropy quantity array)
slew times to all stars (must be indexed by sInds)

observableTimes (astropy Time list)
A binary array with TargetList.nStars rows and (missionFinishAbs-missionStart)/dt columns where dt is 1 day by default. A value of 1 indicates the star is in keepout for (and therefore cannot be observed). A value of 0 indicates the star is not in keepout and may be observed.

sd (astropy Quantity)
Angular separation between stars in rad

tmpCurrentTimeNorm (MJD)
the simulation time after overhead was added in MJD form

Output

sInds [sInds (integer array)]
indces of the stars still in observation list

slewTimes (astropy quantity array)
slew times to all stars (must be indexed by sInds)

intTimes (astropy Quantity array)
Integration times for detection in units of day

dV (astropy Quantity)
Delta-V used to transfer to new star line of sight in units of m/s

5.13.16 filterOcculterSlews Method

Used by the refineOcculterSlews method when slew times have been selected a priori. This method filters out slews that are not within desired observing blocks, the maximum allowed integration time, and are within future keepouts.

Input

old_sInd (integer)
Integer index of the most recently observed star

sInds [sInds]
indices of stars still in observation list

slewTimes (astropy quantity array)
slew times to all stars (must be indexed by sInds)

observableTimes (astropy Time list)
A binary array with TargetList.nStars rows and (missionFinishAbs-missionStart)/dt columns where dt is 1 day by default. A value of 1 indicates the star is in keepout for (and therefore cannot be observed). A value of 0 indicates the star is not in keepout and may be observed.

tmpCurrentTimeNorm (MJD)
the simulation time after overhead was added in MJD form

Output

sInds [sInds (integer array)]
indces of the stars still in observation list

slewTimes (astropy quantity array)
slew times to all stars (must be indexed by sInds)

intTimes (astropy Quantity array)
Integration times for detection in units of day

dV (astropy Quantity)
Delta-V used to transfer to new star line of sight in units of m/s

5.13.17 findAllowableOcculterSlews Method

Used by the refineOcculterSlews method when slew times have not been selected, allowing an exploration of that parameter. This method returns an array of possible slew times for each star in which all are within desired observing blocks, allow integration times, and the stars remain outside of keepout throughout integration.

Input

old_sInd (integer)
Integer index of the most recently observed star

sInds [sInds]
indices of stars still in observation list

slewTimes (astropy quantity array)
slew times to all stars (must be indexed by sInds)

observableTimes (astropy Time list)
A binary array with TargetList.nStars rows and (missionFinishAbs-missionStart)/dt columns where dt is 1 day by default. A value of 1 indicates the star is in keepout for (and therefore cannot be observed). A value of 0 indicates the star is not in keepout and may be observed.

tmpCurrentTimeNorm (MJD)
the simulation time after overhead was added in MJD form

Output

sInds [sInds (integer array)]
indces of the stars still in observation list

slewTimesRange (astropy quantity array)
slew times to all stars (must be indexed by sInds)

intTimesRange (astropy Quantity array)
Integration times for detection in units of day

5.13.18 selectOcculterSlews Method

After finding a range of allowable slew times, this method selects the best slew time for each star based on desired criteria. Slew times for the default method are selected which allow the maximum possible amount of characterization time for each star should there be a detection.

Input

old_sInd (integer)
Integer index of the most recently observed star

sInds [sInds]
indices of stars still in observation list

slewTimesRange (astropy quantity array)
slew times to all stars (must be indexed by sInds)

intTimesRange (astropy Quantity array)
Integration times for detection in units of day

observableTimes (astropy Time list)
A binary array with TargetList.nStars rows and (missionFinishAbs-missionStart)/dt columns where dt is 1 day by default. A value of 1 indicates the star is in keepout for (and therefore cannot be observed). A value of 0 indicates the star is not in keepout and may be observed.

sd (astropy Quantity)
Angular separation between stars in rad
tmpCurrentTimeNorm (MJD)
the simulation time after overhead was added in MJD form

Output
sInds [sInds (integer array)]
indces of the stars still in observation list
slewTimes (astropy quantity array)
slew times to all stars (must be indexed by sInds)
intTimes (astropy Quantity array)
Integration times for detection in units of day
dV (astropy Quantity)
Delta-V used to transfer to new star line of sight in units of m/s

5.14 Survey Ensemble

The Survey Ensemble module's only task is to run multiple simulations. While the implementation of this module is not at all dependent on a particular mission design, it can vary to take advantage of available parallel-processing resources. As the generation of a survey ensemble is an embarrassingly parallel task—every survey simulation is fully independent and can be run as a completely separate process—significant gains in execution time can be achieved with parallelization. The baseline implementation of this module contains a simple looping function that executes the desired number of simulations sequentially, as well as a locally parallelized version based on IPython Parallel.

Depending on the local setup, the Survey Ensemble implementation can save significant time by cloning survey module objects and reinitializing only those sub-modules that have stochastic elements (i.e., the simulated universe). This is currently achieved with the `reset_sim` method in SurveySimulation.

Another possible implementation variation is to use the Survey Ensemble module to conduct investigations of the effects of varying any normally static parameter. This could be done, for example, to explore the impact on yield in cases where the non-coronagraph system throughput, or elements of the propulsion system, are mischaracterized prior to launch. This SE module implementation would overwrite the parameter of interest given in the input specification for every individual survey executed, and saving the true value of the parameter used along with the simulation output.

The parallel implementation in the Prototype is geared towards an ipyparallel-based parallelization scheme, where a single method is asynchronously mapped to *n* workers. The method is called `run_one` and takes as inputs two keyword parameters:

genNewPlanets (bool)
Generate new planets for every run (defaults true).
rewindPlanets (bool)
Reset all planets to their initial conditions on each sim (defaults True, is ignored if genNewPlanets is True).

5.14.1 run_ensemble Method

Input
sim (MissionSim object)
EXOSIMS.MissionSim.MissionSim object
nb_run_sim (integer)
number of simulations to run
run_one (method)
method to call for each simulation
genNewPlanets (boolean)
flag indicating whether to generate new planets each simulation (True means new planets will be generated)
rewindPlanets (boolean)
flag indicating whether planets will be rewound (True means planets will be rewound)
kwargs ()

Output

res (list dict)
simulation list of dictionaries

5.14.2 run_one Method

Input

SS (Survey Simulation Object)
SurveySimulation object

genNewPlanets (boolean)
flag indicating whether to rewind planets in reset_sim function call (True means rewind) (default is True)

rewindPlanets (boolean)
flag indicating whether to generate new plants in reset_sim function call (True means rewind) (default is True)

Output

res (list dict)
simulation list of dictionaries

Acknowledgements
EXOSIMS development is supported by NASA Grant Nos. NNX14AD99G (GSFC) and NNX15AJ67G (WPS).