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

Daniel Garrett, Christian Delacroix, 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: June 13, 2016

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-AFTA. 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. The modules include the Optical System, Star Catalog, Planet Population, Observatory, Planet Physical Model, Time Keeping, Zodiacal Light, Background Sources, and Post-Processing modules and Target List, Simulated Universe, Survey Simulation, and Survey Ensemble modules. Objects of all module classes can be instantiated independently, although most modules require the instantiation of other modules during their construction. 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 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

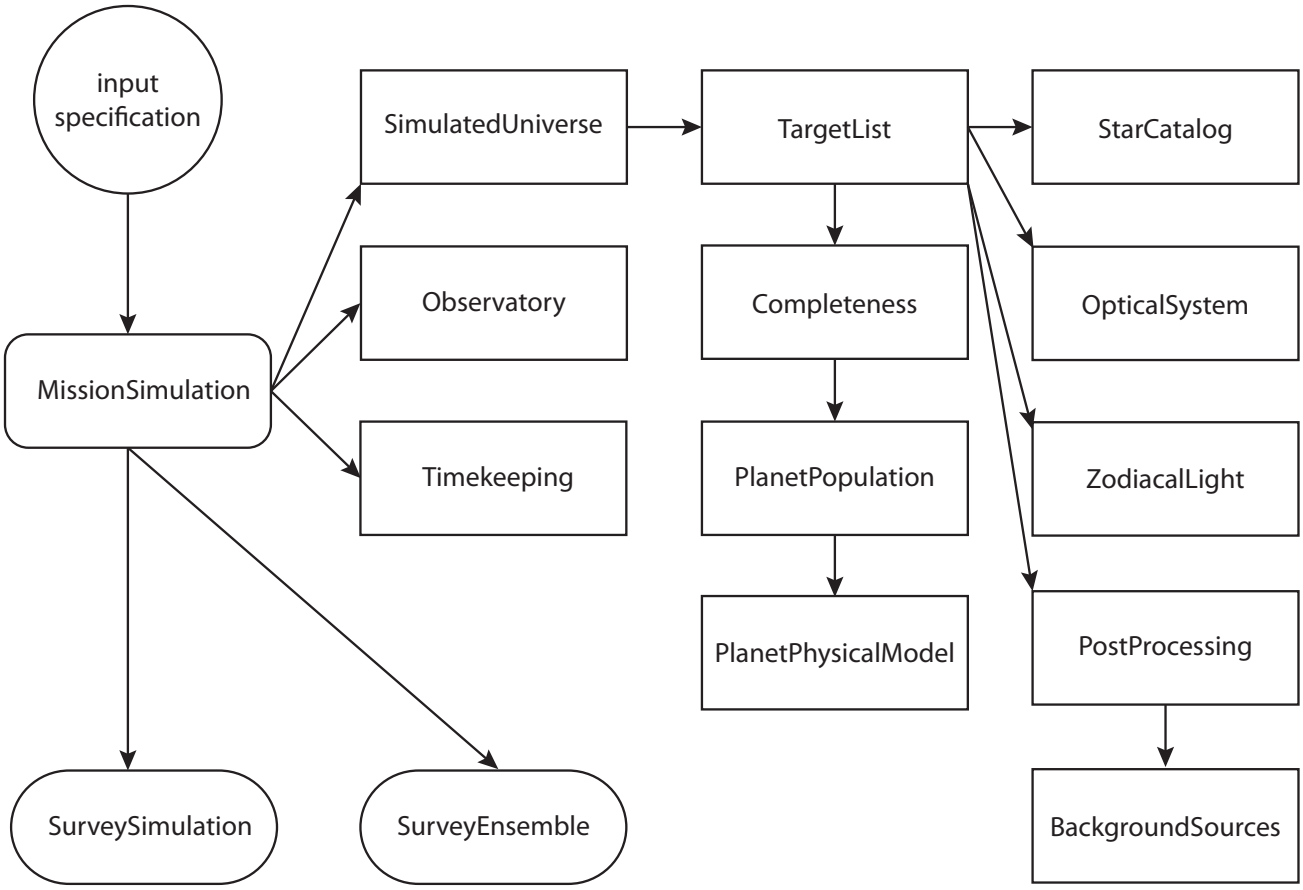

Fig. 1. Schematic depiction of the instantiation path of all EXOSIMS modules. The entry point to the backbone is the construction of a MissionSimulation object, which causes the instantiation of all other module objects. All objects are instantiated in the order shown here, with SurveySimulation and SurveyEnsemble constructed last. 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 at the end of construction, the MissionSimulation object has direct access to all other modules as its attributes.

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
    astropy.constants
    astropy.coordinates
    astropy.time
    astropy.units
copy
importlib
numpy
    numpy.linalg
os
    os.path
```

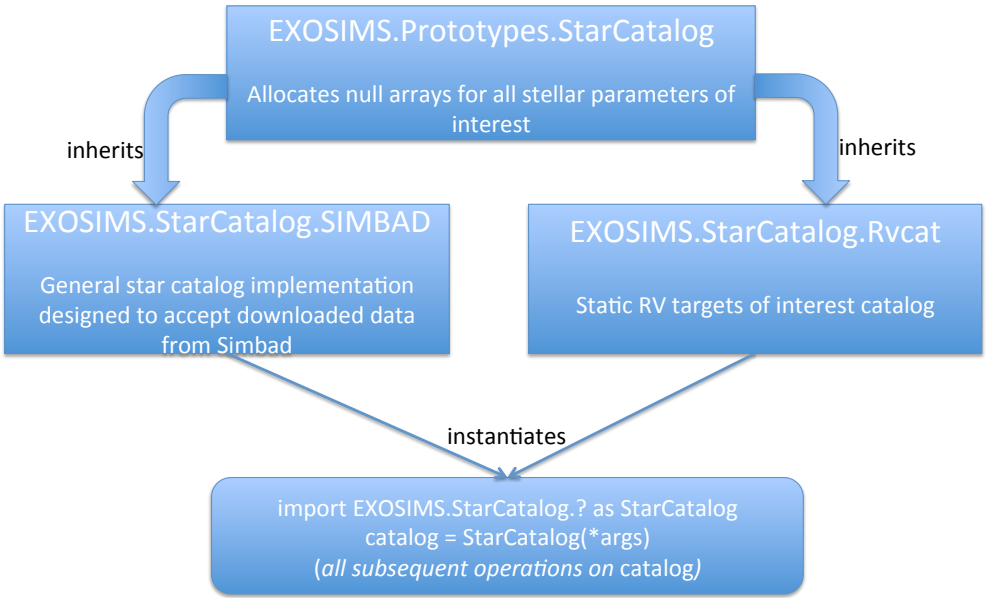

Fig. 2. Schematic of a sample implementation for the three module layers for the Star Catalog module. The Star Catalog prototype (top row) is immutable, specifies the input/output structure of the module along with all common functionality, and is inherited by all Star Catalog class implementations (middle 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.

`pickle/cPickle`
`scipy`
- `scipy.io`
- `scipy.special`
- `scipy.interpolate`

`jplephem (optional)`

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

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 (although not required) 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:

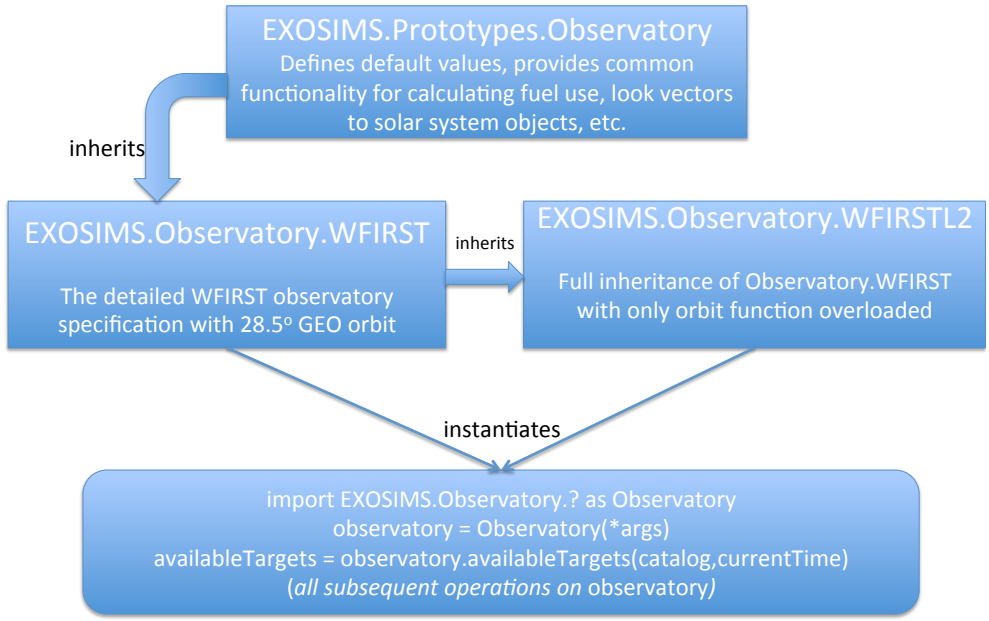

Fig. 3. Schematic of a sample implementation for the three module layers for the Observatory module. The Observatory prototype (top row) is immutable, specifies the input/output structure of the module along with all common functionality, and is inherited by all Observatory class implementations (middle 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.

```
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
```

4 Backbone

By default, the simulation execution will be performed via the backbone. 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. The backbone functionality will primarily be implemented in the MissionSimulation 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. Any mission-specific execution variations will be introduced by method overloading in the inherited survey simulation 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. The backbone 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 MissionSimulation method `genOutSpec`.

The backbone will also contain a specification parser that will check specification files for internal consistency. For example, if modules carry mutual dependencies, the specification parser will return an error if these are not met for a given specification. Similarly, if modules are selected with optional top level inputs, warnings will be generated if these are not set in the same specification files.

In addition to the specification parser, the backbone will contain a method for comparing two specification files and returning the difference between them. Assuming that the files specify all user-settable values, this will be equivalent to simply performing a `diff` operation on any POSIX system. The backbone diff function will add in the capability to automatically fill in unset values with their defaults. 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.

```
{
  "universalParam1": value,
  "universalParam2": value,
  ...
  "starlightSuppressionSystems": [
    {
      "starlightSuppressionSystemNumber": 1,
      "type": "external",
      "detectionTimeMultiplier": value,
      "characterizationTimeMultiplier": value,
      "occulterDiameter": value,
      "NocculterDistances": 2,
      "occulterDistances: [
        {
         "occulterDistanceNumber": 1,
         "occulterDistance": value,
         "occulterBlueEdge": value,
         "occulterRedEdge": value,
         "IWA": value,
         "OWA": value,
         "PSF": "/data/mdo1_psf.fits",
         "throughput": "/data/mdo1_thru.fits",
         "contrast":  "/data1/mdo1_contrast.fits"
        },
        {
         "occulterDistanceNumber": 2,
         "occulterDistance": value,
         "occulterBlueEdge": value,
         "occulterRedEdge": value,
         "IWA": value,
         "OWA": value,
         "PSF": "/data/mdo1_psf.fits",
```

```
        "throughput": "/data/mdo1_thru.fits",
        "contrast":  "/data1/mdo1_contrast.fits"
       }
     ],
     "occulterWetMass": value,
     "occulterDryMass": value,
   },
   {
     "starlightSuppressionSystemNumber": 2,
     "type": "internal",
     "IWA": value,
     "OWA": value,
     "PSF": "/data/coron1_psf.fits",
     "throughput": "/data/coron1_thru.fits",
     "contrast":  "/data1/coron1_contrast.fits",
     "detectionTimeMultiplier": value,
     "characterizationTimeMultiplier": value,
     "opticaloh": value
   }
 ],
 "scienceInstruments": [
   {
     "scienceInstrumentNumber": 1,
     "type": "imager-EMCCD",
     "QE": 0.88,
     "darkCurrent": 9e-5,
     "CIC": 0.0013,
     "readNoise": 16,
     "texp": 1000,
     "pixelPitch": 13e-6,
     "focalLength": 240,
     "ENF": 1.414,
     "G_EM": 500
   }
   {
     "scienceInstrumentNumber": 2,
     "type": "IFS-CCD",
     "QE": 0.88,
     "darkCurrent": 9e-5,
     "CIC": 0.0013,
     "readNoise": 3,
     "texp": 1000,
     "Rspec": 70.0,
   }
 ],
 modules: {
   "PlanetPopulation": "HZEarthTwins",
   "StarCatalog": "exocat3",
   "OpticalSystem": "hybridOpticalSystem1",
   "ZodiacalLight": "10xSolZodi",
   "BackgroundSources": "besanconModel",
   "PlanetPhysicalModel": "fortneyPlanets",
   "Observatory": "WFIRSTGeo",
   "TimeKeeping": "UTCtime",
   "PostProcessing": "KLIPpost",
   "Completeness": "BrownCompleteness",
   "TargetList": "WFIRSTtargets",
   "SimulatedUniverse": "simUniverse1",
```

```
    "SurveySimulation": "backbone1",
    "SurveyEnsemble": "localIpythonEnsemble"
  }
}
```

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.

4.3 Universal Parameters

These parameters apply to all simulations, and are described in detail in their specific module definitions:

`missionLifetime` (float) The total mission lifetime in *years*. 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 missionLifetime $\times$ missionPortion.

`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.

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

`lam` (float) Detection central wavelength in *nm*.

`deltaLam` (float) Detection bandwidth in *nm*.

`BW` (float) Detection bandwidth fraction = $\Delta\lambda/\lambda$.

`specLam` (float) Spectrograph central wavelength in *nm*.

`specDeltaLam` (float) Spectrograph bandwidth in *nm*.

`specBW` (float) Spectrograph bandwidth fraction = $\Delta\lambda_s/\lambda_s$.

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

`shapeFac` (float) Telescope aperture shape factor.

`pupilDiam` (float) Entrance pupil diameter in *m*.

`pupilArea` (float) Entrance pupil area in m^2.

`IWA` (float) Fundamental Inner Working Angle in *arcsec*. No planets can ever be observed at smaller separations.

`OWA` (float) Fundamental Outer Working Angle in *arcsec*. Set to Inf for no OWA. JSON values of 0 will be interpreted as Inf.

`dMagLim` (float) Fundamental limiting Δmag (difference in magnitude between star and planet).

`telescopeKeepout` (float) Telescope keepout angle in *deg*

`attenuation` (float) Non-coronagraph attenuation, equal to the throughput of the optical system without the coronagraph elements.

`intCutoff` (float) Maximum allowed integration time in *day*. No integrations will be started that would take longer than this value.

`exozodi` (float) Mean Exo-zodi level in zodi.

`exozodiVar` (float) Exo-zodi variation (variance of log-normal distribution). Zodi is constant if set to zero.

`FAP` (float) Detection false alarm probability

`MDP` (float) Missed detection probability

`SNimag` (float) Signal to Noise Ratio for imaging/detection.

`SNchar` (float) Signal to Noise Ratio for characterization.

`arange` 2 element list of minimum and maximum semi-major axes in *AU*.

`erange` 2 element list of minimum and maximum eccentricity.

`wrange` 2 element list of minimum and maximum argument of perigee in *deg*.

`Orange` 2 element list of minimum and maximum ascension of the ascending node in *deg*.

`Irange` 2 element list of minimum and maximum inclination in *deg*.

`prange` 2 element list of minimum and maximum planetary geometric albedo.

`Rrange`	2 element list of minimum and maximum planetary Radius in Earth radii.
`Mprange`	2 element list of minimum and maximum planetary mass in Earth masses.
`scaleOrbits`	(boolean) True means planetary orbits are scaled by the square root of stellar luminosity.
`missionStart`	(float) Mission start time in *MJD*.
`missionLife`	(float) Total length of mission in *years*.
`extendedLife`	(float) Extended mission time in *years*. Extended life typically differs from the primary mission in some way—most typically only revisits are allowed
`settlingTime`	(float) Amount of time needed for observatory to settle after a repointing in *day*.
`thrust`	(float) Occulter slew thrust in *mN*.
`slewIsp`	(float) Occulter slew specific impulse in *s*.
`scMass`	(float) Occulter (maneuvering spacecraft) initial wet mass in *kg*.
`dryMass`	(float) Occulter (maneuvering spacecraft) dry mass in *kg*.
`coMass`	(float) Telescope (or non-maneuvering spacecraft) mass in *kg*.
`skIsp`	(float) Specific impulse for station keeping in *s*.
`defburnPortion`	(float) Default burn portion for slewing.
`spkpath`	(string) Full path to SPK kernel file.
`forceStaticEphem`	(boolean) Force use of static solar system ephemeris if set to True, even if jplephem module is present.

5 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 Planet Population

The Planet Population module encodes the density functions of all required planetary parameters, both physical and orbital. These include semi-major axis, eccentricity, orbital orientation, radius, mass, and geometric albedo (see §5.1.2). 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 along the $\mathbf{s}_3$ axis, located at a distance $-d\mathbf{s}$ from the target at the time of observation. 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 $\hat{\mathbf{e}}$). The planet-star orbital radius vector $\mathbf{r}_{P/S}$ 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 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.

5.1.1 Planet Population Object Attribute Initialization

Inputs

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 implementation may choose to only use some for inputs and set the rest via the physical models.

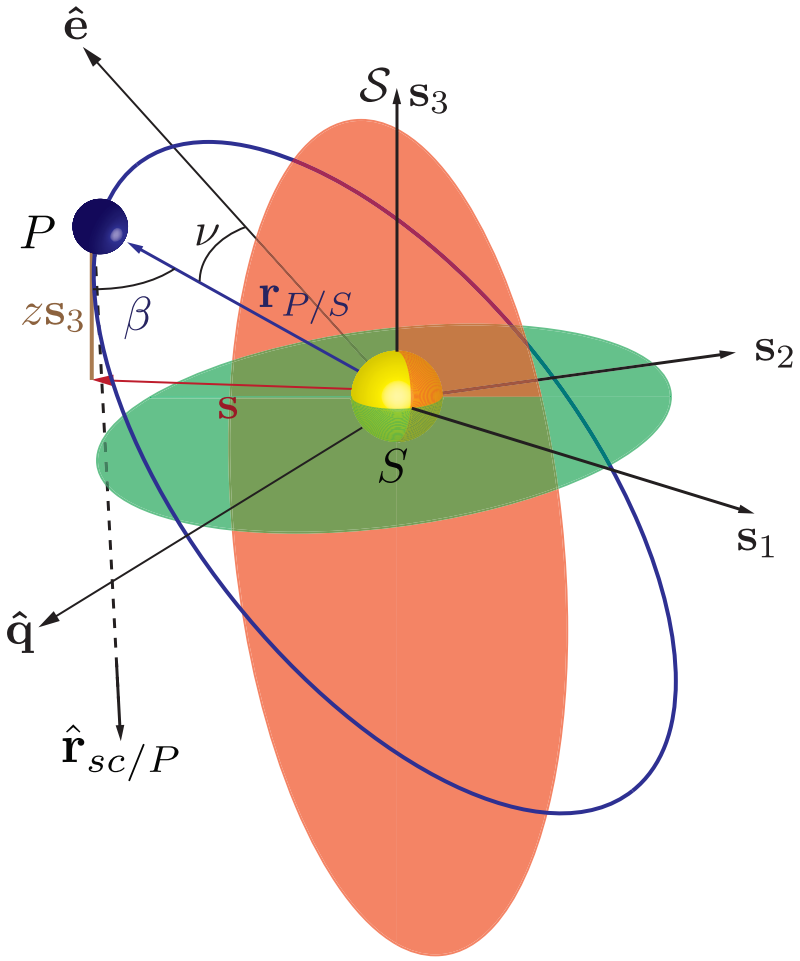

Fig. 4. Definition of reference frames and coordinates of simulated exosystems. The observer lies along the negative $\mathbf{s}_3$ axis so that the observer-star unit vector is $+\mathbf{s}_3$.

arange
: 2 element list of minimum and maximum semi-major axes in *AU*. Default value is [0.01, 100]

erange
: 2 element list of minimum and maximum eccentricity. Default value is [0.01,0.99]

wrange
: 2 element list of minimum and maximum argument of perigee in *deg*. Default value is [0,360]

Orange
: 2 element list of minimum and maximum ascension of the ascending node in *deg*. Default value is [0,360]

Irange
: 2 element list of minimum and maximum inclination in *deg*. Default value is [0,180]

prange
: 2 element list of minimum and maximum planetary geometric albedo. Default value is [0.1,0.6]

Rrange
: 2 element list of minimum and maximum planetary Radius in Earth radii. Default value is [1, 30]

Mprange
: 2 element list of minimum and maximum planetary mass in Earth masses. Default value is [1, 4131]

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

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

Attributes

arange
Semi-major axis range defined as [a_min, a_max] (astropy Quantity initially set in *AU*)

erange
Eccentricity range defined as [e_min, e_max]

wrange
Argument of perigee range defined as [w_min, w_max] (astropy Quantity initially set in *deg*)

Orange
Right ascension of the ascending node range defined as [O_min, O_max] (astropy Quantity initially set in *deg*)

prange
Planetary geometric albedo range defined as [p_min, p_max]

Irange
Planetary orbital inclination range defined as [I_min, I_max] (astropy Quantity initially set in *deg*)

Rrange
Planetary radius range defined as [R_min, R_max] (astropy Quantity initially set in *m*)

Mprange
Planetary mass range defined as [Mp_min, Mp_max] (astropy Quantity initially set in *kg*)

rrange
Planetary orbital radius range defined as [r_min, r_max] derived from PlanetPopulation.arange and PlanetPopulation.erange (astropy Quantity initially set in *AU*)

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

constrainOribts
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.

PlanetPhysicalModel
Planet physical model object

5.1.2 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 attribute, within the ranges specified by the attribute (so that, for example, there will be a `gen_sma` method 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 done by a helper function which stores the last sampled values in memory, and the individual functions (i.e., `gen_sma` and `gen_radius`) will act as getters for the values. 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_sma
Returns semi-major axis values (astropy Quantity initially set in *AU*)

gen_eccen
Returns eccentricity values (numpy ndarray)

gen_w
Returns argument of perigee values (astropy Quantity initially set in *deg*)

gen_O
Returns longitude of the ascending node values (astropy Quantity initially set in *deg*)

gen_radius
Returns planetary radius values (astropy Quantity initially set in *m*)

gen_mass
Returns planetary mass values (astropy Quantity initially set in *kg*)

gen_albedo
Returns planetary geometric albedo (numpy ndarray)

gen_I
Returns values of orbital inclination (astropy Quantity initially set in *deg*)

gen_eccen_from_sma
Required only for populations that can take a `constrainOrbits=True` input. Takes an additional argument of array of semi-major axis values (astropy Quantity). Returns eccentricity values (numpy ndarray) such that $a(1-e) \geq a_{\min}$ and $a(1+e) \leq a_{\max}$.

5.2 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. 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
Provides a method to calculate planetary geometric albedo as a function of the semi-major axis

calc_mass_from_radius
Provides a method to calculate planetary masses from their radii

calc_radius_from_mass
Provides a method to calculate planetary radii from their masses

calc_Phi
Provides a method to calculate the value of the planet phase function given the phase. The prototype implementation uses the Lambert phase function.

5.3 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.

5.3.1 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.

Inputs

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

Attributes

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

5.4 Optical System

The Optical System module contains all of the necessary information to describe the effects of the telescope and starlight suppression system on the target star and planet wavefronts. This requires encoding the design of both the telescope optics and the specific starlight suppression system, whether it be an internal coronagraph or an external occulter. The encoding can be achieved by specifying Point Spread Functions (PSF) for on- and off-axis sources, along with angular separation and wavelength dependent contrast and throughput definitions. 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 imaging detector, 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.

Finally, the Optical System must also include a description of the science instrument. The baseline instrument is assumed to be an imaging spectrometer. The encoding must provide the spatial and wavelength coverage of the instrument as well as sampling for each, along with detector details such as read noise, dark current, and readout cycle.

The Optical System module has four methods used in simulation. `calc_maxintTime` is called from the Target List module to calculate the maximum integration time for each star in the target list (see §5.4.2). `calc_intTime` and `calc_charTime` are called from the Survey Simulation module to calculate integration and characterization times for a target system (see §5.4.3 and §5.4.4). `Cp_Cb` is called by `calc_intTime` and `calc_charTime` to calculate the electron count rates for planet signal and background noise (see §5.4.5). The inputs and outputs for the Optical System methods are depicted in Figure 5.

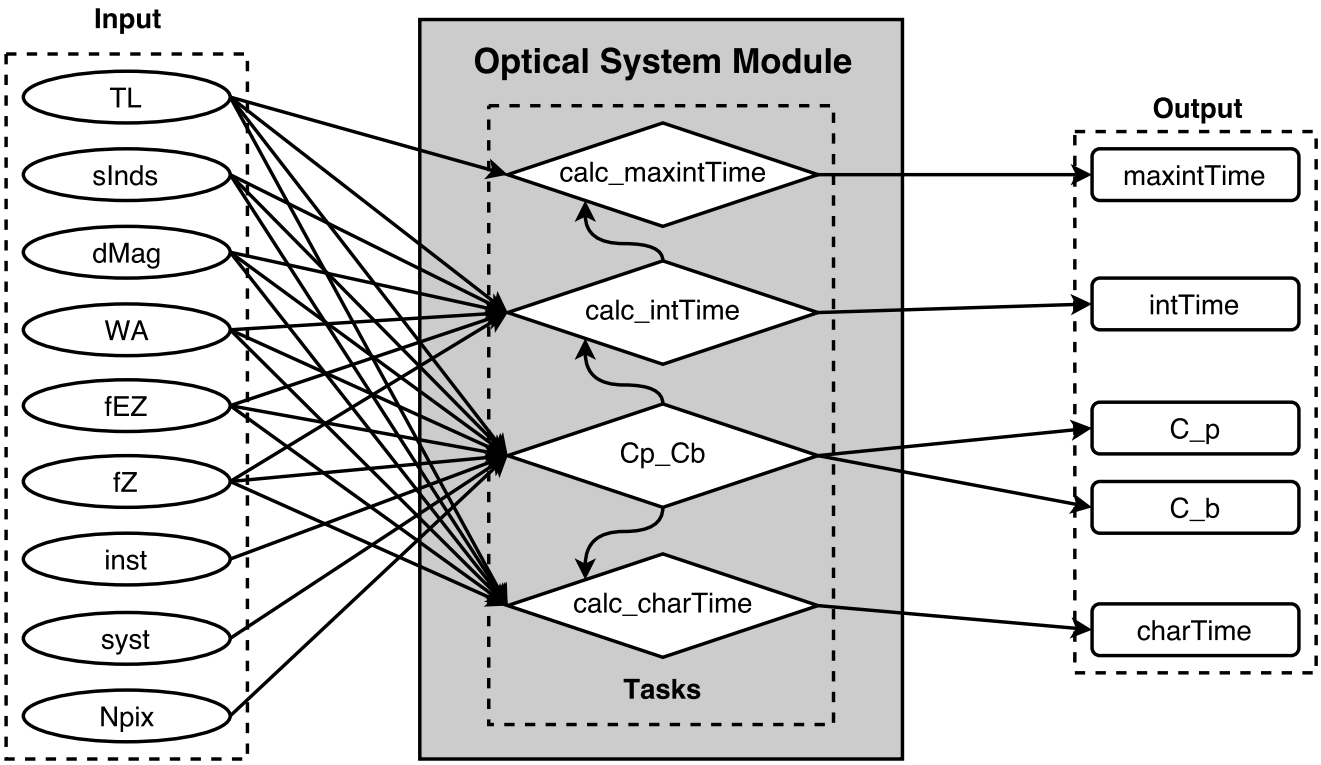

Fig. 5. Depiction of Optical System module methods including inputs and outputs (see §5.4.2, S5.4.3, S5.4.4 and §5.4.5).

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 coronagraph or occulter specific, and will be expected based on the "internal" or "external" starlight suppression system keyword, respectively.

Inputs

Information from simulation specification JSON file organized into a Python dictionary. For multiple systems, there will be an array of dictionaries. If the below `key:value` pairs are missing from the input specification, the Optical System object attributes will be assigned the default values listed. The following are all entries in the passed specs dictionary.

obscurFac
: Obscuration factor due to secondary mirror and spiders. Default value is 0.2.

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

pupilDiam
: Entrance pupil diameter in m. Default value is 4.

telescopeKeepout
: Telescope keepout angle in deg. Default value is 45.

attenuation
: Non-coronagraph attenuation, equal to the throughput of the optical system without the coronagraph elements. Default value is 0.57.

intCutoff
: Maximum allowed integration time in day. No integrations will be started that would take longer than this value. Default value is 50.

Npix
: Number of noise pixels. Default value is 14.3.

Ndark

Number of dark frames used. Default value is 10.

IWA

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

OWA

Fundamental Outer Working Angle in *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.

dMagLim

Fundamental limiting Δmag (difference in magnitude between star and planet). Default value is minimum of contrast.

scienceInstruments

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. In case of multiple instruments, specified wavelength values (lam, deltaLam, BW) of the first instrument become the new default values.

type

(Required) String indicating type of system. Standard values are 'imaging' and 'spectro'.

lam

Central wavelength λ in *nm*. Default value is 500.

deltaLam

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

BW

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

pitch

Pixel pitch in *m*. Default value is 13e-6.

focal

Focal length in *m*. Default value is 140.

idark

Detector dark-current rate in $electrons/s/pix$. Default value is 9e-5.

texp

Exposure time in $s/frame$. Default value is 1e3.

sread

Detector read noise in $electrons/frame$. Default value is 3.

CIC

(Specific to CCDs) Clock-induced-charge in $electrons/pix/frame$. Default value is 0.0013.

ENF

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

Gem

(Specific to EM-CCDs) Electron multiplication gain. Default value is 1.

Rs

(Specific to spectrometers) Spectral resolving power defined as $\lambda/d\lambda$. Default value is 70.

QE

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 *nm*. The ranges on all parameters must be consistent with the values for lam and deltaLam inputs. May be data or FITS filename. Default is scalar 0.9.

starlightSuppressionSystems

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.

type

(Required) String indicating the system type (e.g. internal, external, hybrid), should also contain the type of science instrument it can be used with (e.g. imaging, spectro).

throughput

System throughput: either a scalar for constant throughput, a two-column array for angular separation-dependent throughput, where the first column contains the separations in *arcsec*, or a 2D array for angular separation- and wavelength- dependent throughput, where the first column contains the angular separation values in *arcsec* and the first row contains the wavelengths in *nm*. The ranges on all parameters must be con-

sistent with the values for the IWA, OWA, lam and deltaLam inputs. May be data or FITS filename. Default is scalar 1e-2.

contrast
System contrast: either a scalar for constant contrast, a two-column array for angular separation-dependent contrast, where the first column contains the separations in *arcsec*, or a 2D array for angular separation- and wavelength- dependent contrast, where the first column contains the angular separation values in as and the first row contains the wavelengths in *nm*. The ranges on all parameters must be consistent with the values for the IWA, OWA, lam and deltaLam inputs. May be data or FITS filename. Default is scalar 1e-9.

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

OWA
Specific Outer Working Angle of this system in *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.

dMagLim
Limiting Δmag for this system. Default value is minimum of the system contrast.

PSF
Instrument point spread function. Either a 2D array of a single-PSF, or a 3D array of wavelength-dependent PSFs. May be data or FITS filename. Default is numpy.ones((3,3)).

samp
Sampling of the PSF in *arcsec* per pixel. Default value is 10.

ohTime
Optical system overhead time in *day*. Default value is 1 day. 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.

imagTimeMult
Duty cycle of a detection observation. If only a single integration is required for the initial detection observation, then this value is 1. Otherwise, it is equal to the number of discrete integrations needed to cover the full field of view (i.e., if a shaped pupil with a dark hole that covers 1/3 of the field of view is used for detection, this value would equal 3). Defaults to 1.

charTimeMult
Characterization duty cycle. If only a single integration is required for the initial detection observation, then this value is 1. Otherwise, it is equal to the number of discrete integrations needed to cover 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 by two (for the two polarization states). Defaults to 1.

occulterDiameter
Occulter diameter in *m*. Measured petal tip-to-tip.

NocculterDistances
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
Telescope-occulter separation in *km*.

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

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

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

telescopeKeepout (astropy Quantity)
Telescope keepout angle in units of deg

attenuation (float)
Non-coronagraph attenuation, equal to the throughput of the optical system without the coronagraph elements

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

Npix (float)
Number of noise pixels

Ndark (float)
Number of dark frames used

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 $arcsec$

dMagLim (float)
Fundamental Limiting Δmag (difference in magnitude between star and planet)

scienceInstruments (list of dicts)
List of dictionaries containing all supplied science instrument attributes. Typically the first instrument will be the one used for imaging, and the last one for spectroscopy. Only required attribute is 'type'. See above for other commonly used attributes.

Imager (dict)
Dictionary containing imaging camera attributes. Default to `scienceInstruments[0]`.

Spectro (dict)
Dictionary containing spectrograph attributes. Default to `scienceInstruments[-1]`.

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

ImagerSyst (dict)
Dictionary containing imaging coronagraph attributes. Default to `starlightSuppressionSystems[0]`.

SpectroSyst (dict)
Dictionary containing spectroscopy coronagraph attributes. Default to `starlightSuppressionSystems[-1]`.

5.4.2 calc_maxintTime Method

The `calc_maxintTime` method calculates the maximum integration time for each star in the target list. This method is called from the Target List module.

Inputs

TL (object)
TargetList class object, see §5.11 for definition of available attributes

Output

maxintTime (astropy Quantity array)

Maximum integration time for each target star in units of *day*

5.4.3 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.

Inputs

TL (object)

TargetList class object, see §5.11 for definition of available attributes

sInds (integer ndarray)

Integer indices of the stars of interest, with the length of the number of planets of interest. For instance, if a star hosts *n* planets, the index of this star must be repeated *n* times.

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

fEZ (astropy Quantity array)

Surface brightness of exo-zodiacal light in units of $1/arcsec^2$

fZ (astropy Quantity array)

Surface brightness of local zodiacal light in units of $1/arcsec^2$

Output

intTime (astropy Quantity array)

Integration time for each of the planets of interest in units of *day*

5.4.4 calc_charTime Method

The `calc_charTime` method calculates the characterization time required for a specific target system. This method is called from the Survey Simulation module.

Inputs

TL (object)

TargetList class object, see §5.11 for definition of available attributes

sInds (integer ndarray)

Integer indices of the stars of interest, with the length of the number of planets of interest. For instance, if a star hosts *n* planets, the index of this star must be repeated *n* times.

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

fEZ (astropy Quantity array)

Surface brightness of exo-zodiacal light in units of $1/arcsec^2$

fZ (astropy Quantity array)

Surface brightness of local zodiacal light in units of $1/arcsec^2$

Output

charTime (astropy Quantity array)

Characterization time for each of the planets of interest in units of *day*

5.4.5 Cp_Cb Method

The `Cp_Cb` method calculates the electron count rates for planet signal and background noise.

Inputs

TL (object)
TargetList class object, see §5.11 for definition of available attributes

sInds (integer ndarray)
Integer indices of the stars of interest, with the length of the number of planets of interest. For instance, if a star hosts n planets, the index of this star must be repeated n times.

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

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

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

inst (dict)
Selected Science Instrument

syst (dict)
Selected Starlight Suppression System

Npix (float)
Number of noise pixels

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 $1/s$

5.5 Zodiacal Light

The Zodiacal Light module contains the `fZ` and `fEZ` methods. The `fZ` method calculates the surface brightness of local zodiacal light. The `fEZ` calculates the surface brightness of exozodiacal light. The inputs and outputs for the Zodiacal Light method are depicted in Figure 6.

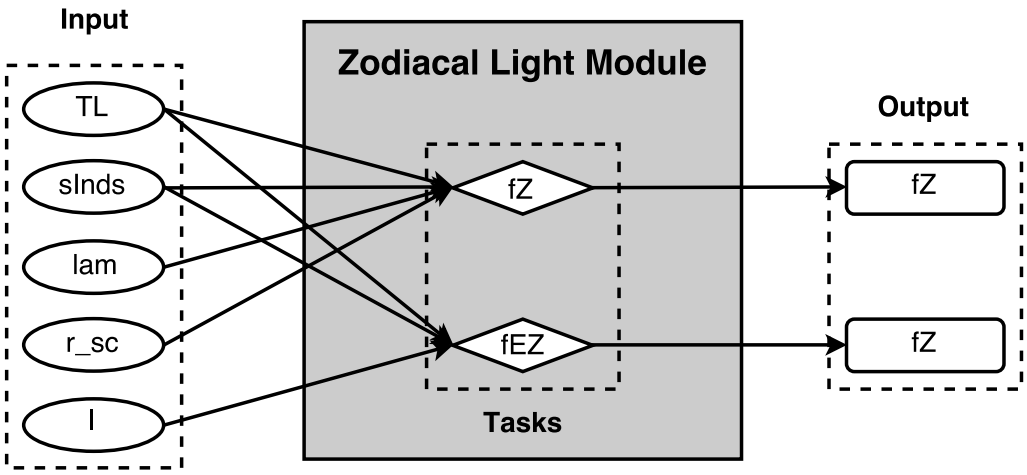

Fig. 6. Depiction of Zodiacal Light module method including inputs and outputs (see §5.5.2 and §5.5.3).

5.5.1 Zodiacal Light Object Attribute Initialization

Input

magZ
Zodi brightness magnitude (per $arcsec^2$). Defaults to 23.
magEZ
Exo-zodi brightness magnitude (per $arcsec^2$). Defaults to 22.
varEZ
Exo-zodiacal light variation (variance of log-normal distribution). Constant if set to 0. Defaults to 0.
nEZ
Exo-zodiacal light level in zodi. Defaults to 1.5.

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)
nEZ (float)
Exo-zodiacal light level in zodi

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.

Inputs

TL (object)
TargetList class object, see §5.11 for description of functionality and attributes
sInds (integer ndarray)
Integer indices of the stars of interest, with the length of the number of planets of interest
lam (astropy Quantity)
Central wavelength in units of *nm*
r_sc (astropy Quantity 1×3 array)
Observatory position vector in units of *km*

Outputs

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.

Inputs

TL (object)
TargetList class object, see §5.11 for description of functionality and attributes
sInds (integer ndarray)
Integer indices of the stars of interest, with the length of the number of planets of interest
I (astropy Quantity array)
Inclination of the planets of interest in units of *deg*

Outputs

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

5.6 Background Sources

The Background Sources module will provide density of background sources for a given target based on its coordinates and the integration depth. This will be used in the post-processing module to determine false alarms based on confusion. The prototype module has no inputs and only a single function: `dNbackground`.

5.6.1 dNbackground

Inputs

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 absolute magnitudes (in the detection band) of the dark hole to be produced for each target. Must be of same length as coords.

Outputs

dN (astropy Quantity array)

Number densities of background sources for given targets in units of $1/arcmin^2$. Same length as inputs.

5.7 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 two main methods: `orbit` and `keepout`, 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 and Target List module to determine which of the stars are observable at the current mission time.

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 such as the sun, moon, and solar system planets. 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 current mission time and Star Catalog or Target List module output 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.

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-AFTA with an L2 halo orbit, which is the new mission baseline.

The inputs, outputs, and updated attributes of the required Observatory module methods are depicted in Figure 7.

5.7.1 Observatory Object Attribute Initialization

Inputs

settlingTime

Amount of time needed for observatory to settle after a repointing in *day*. Default value is 1.

thrust

Occulter slew thrust in *mN*. Default value is 450.

slewIsp

Occulter slew specific impulse in *s*. Default value is 4160.

scMass

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

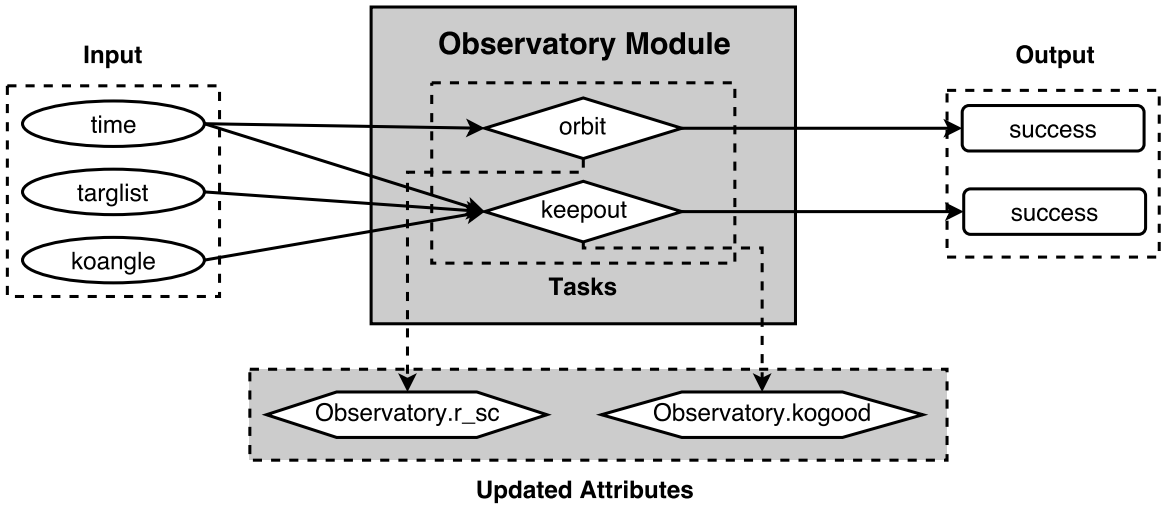

Fig. 7. Depiction of Observatory module methods including inputs, outputs, and updated attributes (see §5.7.2 and §5.7.3).

dryMass
Occulter (maneuvering spacecraft) dry mass in *kg*. Default value is 3400.
coMass
Telescope (or non-maneuvering spacecraft) mass in *kg*. Default value is 5800.
skIsp
Specific impulse for station keeping in *s*. Default value is 220.
defburnPortion
Default burn portion for slewing. Default value is 0.05
spkpath
String with full path to SPK kernel file (only used if using jplephem for solar system body propagation - see 5.7.4.
forceStaticEphem
Boolean, forcing use of static solar system ephemeris if set to True, even if jplephem module is present (see 5.7.4). Default value is False.

Attributes
settlingTime
Amount of time needed for observatory to settle after a repointing (astropy Quantity initially set in *day*)
thrust
Occulter slew thrust (astropy Quantity initially set in *mN*)
slewIsp
Occulter slew specific impulse (astropy Quantity initially set in *s*)
scMass
Occulter (maneuvering spacecraft) initial wet mass (astropy Quantity initially set in *kg*)
dryMass
Occulter (maneuvering spacecraft) dry mass (astropy Quantity initially set in *kg*)
coMass
Telescope (or non-maneuvering spacecraft) mass (astropy Quantity initially set in *kg*)
kogood
1D NumPy ndarray of Boolean values where True is a target unobstructed and observable in the keepout zone. Initialized to an empty array. This attribute is updated to the current mission time through the keepout method (see 5.7.3).
r_sc
Observatory orbit position in HE reference frame. Initialized to NumPy ndarray as numpy.array([0., 0., 0.]) and associated with astropy Quantity in *km*. This attribute is updated to the orbital position of the observatory at the current mission time through the orbit method (see 5.7.2).
skIsp

Specific impulse for station keeping (astropy Quantity initially set in *s*)

defburnPortion

Default burn portion for slewing

currentSep

Current occulter separation (astropy Quantity initially set in *km*

flowRate

Slew flow rate derived from thrust and slewIsp (astropy Quantity initially set in kg/day)

5.7.2 orbit Method

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

Inputs

time

astropy Time object which may be `TimeKeeping.currenttimeAbs` from Time Keeping module see 5.8.1 for definition

Outputs

success

Boolean indicating if orbit was successfully calculated

Updated Object Attributes

Observatory.r_sc

Observatory orbit position in HE reference frame at current mission time (astropy Quantity defined in *km*)

5.7.3 keepout Method

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

Inputs

time

astropy Time object which may be `TimeKeeping.currenttimeAbs` (see 5.8.1 for definition)

targlist

Instantiated Target List object from Target List module. See 5.11 for definition of available attributes

koangle

Telescope keepout angle in *deg* - `OpticalSystem.telescopeKeepout`

Outputs

success

Boolean indicating if orbit was successfully calculated

Updated Object Attributes

Observatory.kogood

1D NumPy ndarray of Boolean values for each target at given time where True is a target unobstructed in the keepout zone and False is a target unobservable due to obstructions in the keepout zone

5.7.4 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.

Inputs

time

astropy Time object which may be `TimeKeeping.currenttimeAbs` (see 5.8.1 for definition)

bodyname

String containing object name, capitalized by convention.

Outputs

r_body

(Quantity) heliocentric equatorial position vector (units of km)

5.8 Time Keeping

The Time Keeping module is responsible for keeping track of the current mission time. It encodes only the mission start time, the mission duration, and the current time within a simulation. 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 *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 the `update_times` and `duty_cycle` methods. These methods updates the mission time during a survey simulation. The duty cycle determines when during the mission timeline the observatory is allowed to perform planet-finding operations. The duty cycle function takes the current mission time as input and outputs the next available time when exoplanet observations may begin or resume, along with the duration of the observational period. The outputs of this method are used in the Survey Simulation module to determine when and how long exoplanet finding and characterization observations occur. The inputs and updated attributes for the Time Keeping methods are depicted in Figure 8.

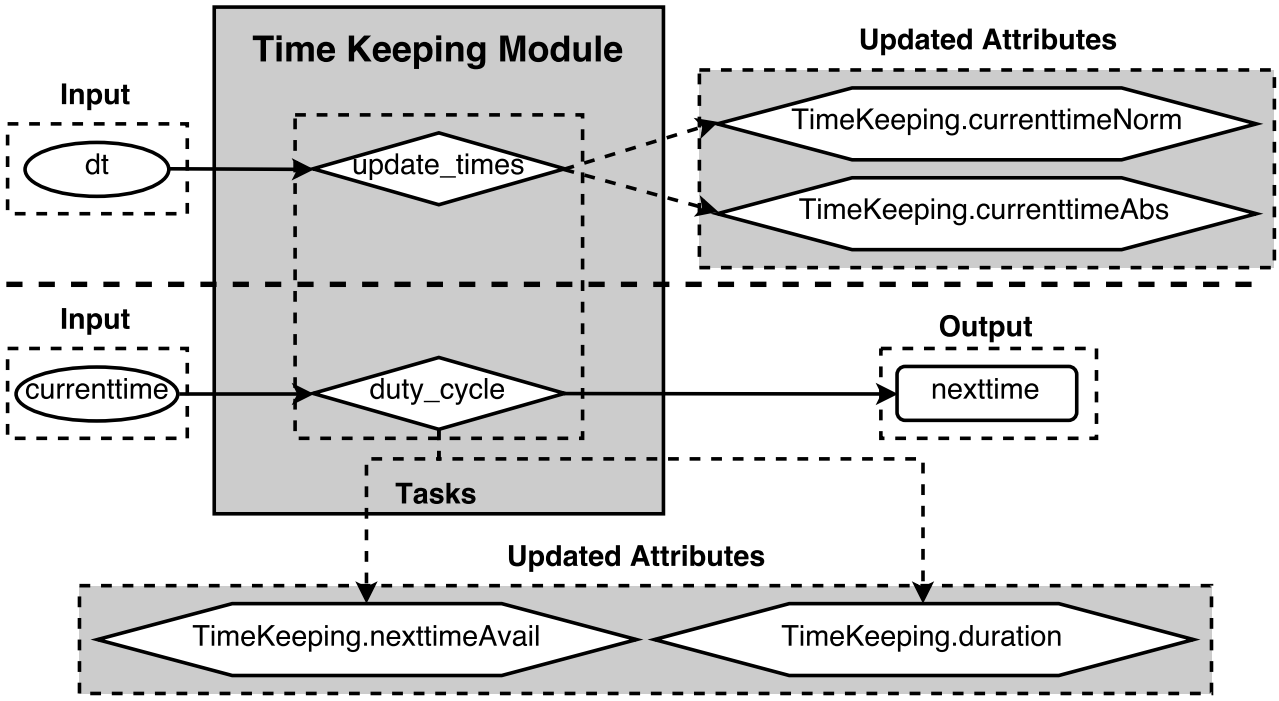

Fig. 8. Depiction of Time Keeping module method including input and updated attributes (see §5.8.2 and §5.8.3).

5.8.1 Time Keeping Object Attribute Initialization

Inputs

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

missionLife
Total length of mission in *years*. Default value is 6.

extendedLife
Extended mission time in *years*. Default value is 0. Extended life typically differs from the primary mission in some way—most typically only revisits are allowed.

missionPortion
Portion of mission time devoted to planet-finding. Default value is 1/6.

Attributes

missionStart
Mission start time (astropy Time object initially defined in *MJD*)

missionLife
Mission lifetime (astropy Quantity initially set in *years*)

extendedLife
Extended mission time (astropy Quantity initially set in *years*)

missionPortion
Portion of mission time devoted to planet-finding

duration
Duration of planet-finding operations (astropy Quantity initially set in *day*)

nexttimeAvail
Next time available for planet-finding (astropy Quantity initially set in *day*)

currenttimeNorm
Current mission time normalized so that start date is 0 (astropy Quantity initially set in *day*)

currenttimeAbs
Current absolute mission time (astropy Time object initially defined in *MJD*)

missionFinishNorm
Mission finish time (astropy Quantity initially set in *day*)

missionFinishAbs
Mission completion date (astropy Time object initially defined in *MJD*)

5.8.2 update_times Method

The `update_times` method updates the relevant mission times.

Inputs

dt
Time increment (astropy Quantity with units of time)

Updated Object Attributes

TimeKeeping.currenttimeNorm
Current mission time normalized so that start date is 0 (astropy Quantity with units of time)

TimeKeeping.currenttimeAbs
Current absolute mission time (astropy Time object)

5.8.3 duty_cycle Method

The `duty_cycle` method calculates the next time that the observatory will be available for exoplanet science and returns this time and the maximum amount of time afterwards during which an exolanet observation can run (if capped).

Inputs

currenttime
Current time in mission simulation (astropy Quantity with units of time often `TimeKeeping.currenttimeNorm`)

Outputs

nexttime

Next available time for planet-finding (astropy Quantity with units of time)

Updated Object Attributes

TimeKeeping.nexttimeAvail

Next time available for planet-finding (astropy Quantity with units of time)

TimeKeeping.duration

Duration of planet-finding operations (astropy Quantity with units of time)

5.9 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` task. This task determines if a planet detection occurs for a given observation. The inputs and outputs for this task are depicted in Figure 9.

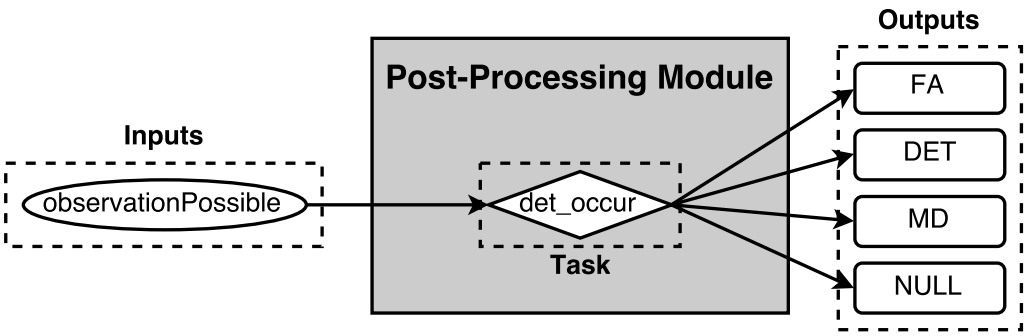

Fig. 9. Depiction of Post-Processing module task including inputs and outputs (see §5.9.2).

5.9.1 Post-Processing Object Attribute Initialization

Inputs

FAP

Detection false alarm probability. Default value is 3×10^{-5}.

MDP

Missed detection probability. Default value is 10^{-3}.

ppFact

Post-processing contrast factor, between 0 and 1. Default value is 1.

SNimag

Signal to Noise Ratio threshold for imaging/detection. Default value is 5.

SNchar

Signal to Noise Ratio threshold for characterization. Default value is 11.

Attributes

BackgroundSources (object)

BackgroundSources class object (see 5.6)

FAP

Detection false alarm probability

MDP

Missed detection probability

5.9.2 det_occur Method

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

Inputs

observationPossible

1D NumPy ndarray of booleans signifying if a planet in the system being observed is observable

Outputs

FA

Boolean where True means False Alarm

DET

Boolean where True means DETection

MD

Boolean where True means Missed Detection

NULL

Boolean where True means Null Detection

5.10 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` and `completeness_update`. `target_completeness` generates initial completeness values for each star in the target list (see §5.10.2). `completeness_update` updates the completeness values following an observation (see §5.10.4).

5.10.1 Completeness Object Attribute Initialization

Input

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

Attributes

PlanetPopulation

Planet Population module object (see 5.1)

PlanetPhysicalModel

Planet Physical Model module object (see 5.2)

5.10.2 target_completeness Method

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

Inputs

targlist

Instantiated Target List object from Target List module see §5.11 for definition of functionality and attributes

Outputs

comp0

1D NumPy ndarray containing completeness values for each star in the target list

5.10.3 gen_update Method

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

Input

TL

Instantiated Target List object from Target List module see §5.11 for definition of functionality and attributes

5.10.4 completeness_update Method

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

Inputs

s_ind
index of star in target list just observed

targlist
Instantiated Target List object from Target List module see §5.11 for definition of functionality and attributes

obsbegin
Mission time when the observation of `s_ind` began (astropy Quantity with units of time)

obsend
Mission time when the observation of `s_ind` ended (astropy Quantity with units of time)

nexttime
Mission time of next observational period (astropy Quantity with units of time)

Output

comp0
1D NumPy ndarray of updated completeness values for each star in the target list

5.11 Target List

The Target List module takes in information from the Optical System, Star Catalog, Planet Population, and Observatory modules and Completeness module 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.

5.11.1 Target List Object Attribute Initialization

Inputs

keepStarCatalog
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.

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

Attributes

(StarCatalog values)
Mission specific filtered star catalog values from StarCatalog class object (see 5.3)

PlanetPopulation (object)
PlanetPopulation class object (see 5.1)

PlanetPhysicalModel (object)
PlanetPhysicalModel class object (see 5.2)

StarCatalog (object)
StarCatalog class object (only retained if keepStarCatalog is True, see 5.3)

OpticalSystem (object)
OpticalSystem class object (see 5.4)

ZodiacalLight (object)
ZodiacalLight class object (see 5.5)

BackgroundSources (object)
BackgroundSources class object (see 5.6)

PostProcessing (object)
PostProcessing class object (see 5.9)

Completeness (object)
Completeness class object (see 5.10)

maxintTime (astropy Quantity array)
Maximum integration time for each target star in units of *day*. Calculated from `OpticalSystem.calc_maxintTime` §5.4.2
comp0 (float ndarray)
Completeness value for each target star. Calculated from `Completeness.target_completeness` §5.10.2
minComp (float)
Minimum completeness value for inclusion in target list.
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

5.11.2 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).

Inputs
sInds (integer ndarray)
Indices of the stars of interest, with the length of the number of planets of interest
lam (astropy Quantity)
Wavelength in units of *nm*

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

5.11.3 populate_target_list method

This 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 (§5.10.2) and the `calc_maxintTime` in the Optical System module (§5.4.2) to generate the initial completeness and maximum integration time for all targets. It also generates 'true' and 'approximate' star masses using object method `stellar_mass` (see below).

5.11.4 filter_target_list method

This 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 limiting delta mag is above the population maximum delta mag, where the integration time for the brightest planet in the modeled population is above the specified maximum integration time, and all targets where the initial completeness is below the specified threshold.

5.11.5 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 implementation implements the following methods:

nan_filter: Filters any target list entires with NaN values
binary_filter: Filters any targets with attribute Binary_Cut set to True
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
outside_IWA_filter: Filters out targets with all planets inside of the IWA
int_cutoff_filter: Filters out all targets with maximum integration times above the specified (in the input spec) threshold integration time
max_dmag_filter: Filters out all targets with minimum delta mag above the limiting delta mag (from input spec)
completeness_filter: Filters out all targets with single visit completeness below the specified (in the input spec) threshold completeness
revise_lists: General helper function for applying filters.

5.12 Simulated Universe

The Simulated Universe module instantiate 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 that maps planets to the stars in the target list.

The Simulated Universe module contains the following methods:

```
gen_planetary_systems
planet_pos_vel
prop_system
get_current_WA
```

gen_planetary_systems populates the orbital elements and physical characteristics of all planets (see §5.12.2). planet_pos_vel finds initial position and velocity vectors for each planet (see §5.12.3). prop_system propagates planet position, velocity, and star separation in time (see §5.12.4). get_current_WA calculates planet current working angle (see §5.12.5). All planetary parameters are generated in the constructor via calls to the appropriate value generating functions in the planet population module.

5.12.1 Attributes

PlanetPopulation (object)
PlanetPopulation class object (see 5.1)
PlanetPhysicalModel (object)
PlanetPhysicalModel class object (see 5.2)
OpticalSystem (object)
OpticalSystem class object (see 5.4)
ZodiacalLight (object)
ZodiacalLight class object (see 5.5)
BackgroundSources (object)
BackgroundSources class object (see 5.6)
PostProcessing (object)
PostProcessing class object (see 5.9)
Completeness (object)
Completeness class object (see 5.10)
TargetList (object)
TargetList class object (see 5.11)
eta (float)
Global occurrence rate defined as expected number of planets per star in a given universe
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*
r (astropy Quantity n×3 array)
Planet position vector in units of *km*
v (astropy Quantity n×3 array)
Planet velocity vector in units of km/s
s (astropy Quantity array)
Planet-star apparent separations in units of *km*
d (astropy Quantity array)
Planet-star distances in units of *km*
Mp (astropy Quantity array)
Planet mass in units of *kg*
Rp (astropy Quantity array)
Planet radius in units of *km*
p (float ndarray)
Planet albedo
fEZ (astropy Quantity array)
Surface brightness of exozodiacal light in units of $1/arcsec2$, determined from `ZodiacalLight.fEZ` §5.5.3

5.12.2 gen_planetary_system Method

The `gen_planetary_system` 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.

Inputs
This method does not take any explicit inputs. It uses the inherited TargetList and PlanetPopulation objects.

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

v (astropy Quantity n×3 array)
Planet velocity vector in units of km/s
s (astropy Quantity array)
Planet-star apparent separations in units of km
d (astropy Quantity array)
Planet-star distances in units of km
Mp (astropy Quantity array)
Planet mass in units of kg
Rp (astropy Quantity array)
Planet radius in units of km
p (float ndarray)
Planet albedo
fEZ (astropy Quantity array)
Surface brightness of exozodiacal light in units of $1/arcsec2$, determined from `ZodiacalLight.fEZ` §5.5.3

5.12.3 planet_pos_vel Method

The `planet_pos_vel` method assigns each planet an initial position and velocity vector with appropriate astropy Quantity units attached.

Inputs

This method does not take any explicit inputs. It uses the following attributes assigned before calling this method:

```
SimulatedUniverse.a
SimulatedUniverse.e
SimulatedUniverse.Mp
SimulatedUniverse.I
SimulatedUniverse.w
SimulatedUniverse.O
```

Outputs

r (astropy Quantity n×3 array)
Planet position vector in units of km
v (astropy Quantity n×3 array)
Planet velocity vector in units of km/s

5.12.4 prop_system Method

The `prop_system` method propagates planet state vectors (position and velocity) in time.

Inputs

r (astropy Quantity n×3 array)
Initial position vector of each planet in units of km
v (astropy Quantity n×3 array)
Initial velocity vector of each planet in units of km/s
Mp (astropy Quantity array)
Planet masses in units of kg
Ms (float ndarray)
Target star masses in M_sun
dt (astropy Quantity)
Time increment to propagate the system in units of day

Outputs

rnew (astropy Quantity n×3 array)
Propagated position vectors in units of km
vnew (astropy Quantity n×3 array)
Propagated velocity vectors in units of km/s

snew (astropy Quantity array)
Propagated apparent separations in units of *km*
dnew (astropy Quantity array)
Propagated planet-star distances in units of *km*

5.12.5 get_current_WA Method

The `get_current_WA` method calculates the current working angles for planets specified by the given indices.

Inputs
pInds (integer ndarray)
Integer indices of the planets of interest

Outputs
WA (astropy Quantity array)
Working angles in units of *arcsec*

5.13 Survey Simulation

The Survey Simulation module takes as input instances of the Simulated Universe module and the Time Keeping, and Post-Processing modules. 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).

Survey Simulation TASKS: `run_sim()` - perform survey simulation §5.13.2

Survey Simulation SUBTASKS: `initial_target()` - find initial target star §5.13.3

`observation_detection(pInds, s_ind, DRM, planPosTime)` - finds if planet detections are possible and returns relevant information §5.13.4

`det_data(s, dMag, Ip, DRM, FA, DET, MD, s_ind, pInds, observationPossible, observed)` - determines detection status §5.13.5

```
observation_characterization(observationPossible, pInds, s_ind, spectra,\
  s, Ip, DRM, FA, t_int)
```

finds if characterizations are possible and returns relevant information §5.13.6

`next_target(s_ind, revisit_list, extended_list, DRM)` - find next target (scheduler) §5.13.7

5.13.1 Survey Simulation Object Attribute Initialization

Inputs
OpticalSystem
Instance of Optical System module inherited from Simulated Universe module (see 5.4)
PlanetPopulation
Instance of Planet Population module inherited from Simulated Universe module (see 5.1)
ZodiacalLight
Instance of Zodiacal Light module inherited from Simulated Universe module (see 5.5)
Completeness
Instance of Completeness module inherited from Simulated Universe module (see 5.10)
TargetList
Instance of Target List module inherited from Simulated Universe module (see 5.11)
PlanetPhysicalModel
Instance of Planet Physical Model module inherited from Simulated Universe module (see 5.2)
SimulatedUniverse
Instance of Simulated Universe module (see 5.12)
Observatory
Instance of Observatory module (see 5.7)
TimeKeeping
Instance of Time Keeping module (see 5.8)

PostProcessing
Instance of Post-Processing module (see 5.9)

Attributes

OpticalSystem
Instance of Optical System module (see 5.4)

PlanetPopulation
Instance of Planet Population module (see 5.1)

ZodiacalLight
Instance of Zodiacal Light module (see 5.5)

Completeness
Instance of Completeness module (see 5.10)

TargetList
Instance of Target List module (see 5.11)

PlanetPhysicalModel
Instance of Planet Physical Model module (see 5.2)

SimulatedUniverse
Instance of Simulated Universe module (see 5.12)

Observatory
Instance of Observatory module (see 5.7)

TimeKeeping
Instance of Time Keeping module (see 5.8)

PostProcessing
Instance of Post-Processing module (see 5.9)

DRM
Contains the results of survey simulation

5.13.2 run_sim Method

The `run_sim` method performs the survey simulation and populates the results in `SurveySimulation.DRM`.

Inputs

This method does not take any explicit inputs. It uses the inherited modules to generate a survey simulation.

Updated Object 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):

target_ind
Index of star in target list observed

arrival_time
Days since mission start when observation begins

sc_mass
Maneuvering spacecraft mass (if simulating an occulter system)

dF_lateral
Lateral disturbance force on occulter in N if simulating an occulter system

dF_axial
Axial disturbance force on occulter in N if simulating an occulter system

det_dV
Detection station-keeping ΔV in m/s if simulating an occulter system

det_mass_used
Detection station-keeping fuel mass used in kg if simulating an occulter system

det_int_time
Detection integration time in day

det_status
Integer or list where
1 = detection

0 = null detection
-1 = missed detection
-2 = false alarm

det_WA
Detection WA in *mas*

det_dMag
Detection Δmag

char_1_time
Characterization integration time in *day*

char_1_dV
Characterization station-keeping ΔV in m/s if simulating an occulter system

char_1_mass_used
Characterization station-keeping fuel mass used in *kg* if simulating an occulter system

char_1_success
Characterization success where value which may be:

1 - successfull characterization
effective wavelength found during characterization in *nm*

slew_time
Slew time to next target in *day* if simulating an occulter system

slew_dV
Slew ΔV in m/s if simulating an occulter system

slew_mass_used
Slew fuel mass used in *kg* if simulating an occulter system

slew_angle
Slew angle to next target in *rad*

5.13.3 initial_target Sub-task

The `initial_target` sub-task is called from the `run_sim` method to determine the index of the initial target star in the target list.

Inputs

This sub-task does not take any explicit inputs. It may use any of the inherited modules to generate the initial target star index.

Outputs

s_ind
Index of the initial target star

5.13.4 observation_detection Sub-task

The `observation_detection` sub-task is called from the `run_sim` task to determine if planets may be detected and calculate information needed later in the simulation.

Inputs

pInds
1D NumPy ndarray of indices of planets belonging to the target star (used to get relevant attributes from the `SimulatedUniverse` module)

s_ind
Index of target star in target list

DRM
Python dictionary containing survey simulation results of current observation as key:value pairs

planPosTime
1D NumPy ndarray containing the times at which the planet positions and velocities contained in `SimulatedUniverse.r` and `SimulatedUniverse.v` are current (astropy Quantity with units of time)

Outputs

observationPossible
1D NumPy ndarray (length is number of planets in the system under observation) containing boolean values where True is an observable planet

t_int
Integration time (astropy Quantity with units of time)

DRM
Python dictionary containing survey simulation results of current observation as key:value pairs

s
1D NumPy ndarray (length is number of planets in the system under observation) containing apparent separation of planets (astropy Quantity with units of distance)

dMag
1D NumPy ndarray (length is number of planets in the system under observation) containing Δmag for each planet

Ip
1D NumPy ndarray (length is number of planets in the system under observation) containing irradiance (astropy Quantity with units of $\frac{1}{m^2 \cdot nm \cdot s}$)

5.13.5 det_data Sub-task

The `det_data` sub-task is called from the `run_sim` task to assign a detection status to the dictionary of current observation results.

Inputs

s
1D NumPy array (length is number of planets in the system under observation) containing apparent separation of planets (astropy Quantity with units of distance)

dMag
1D NumPy ndarray (length is number of planets in the system under observation) containing Δmag for each planet

Ip 1D NumPy ndarray (length is number of planets in the system under observation) containing irradiance (astropy Quantity with units of $\frac{1}{m^2 \cdot nm \cdot s}$)

DRM
Python dictionary containing survey simulation results of current observation as key:value pairs

FA
Boolean where True is False Alarm

DET
Boolean where True is DETection

MD
Boolean where True is Missed Detection

s_ind
Index of target star in target list

pInds
1D NumPy ndarray of indices of planets belonging to the target star (used to get relevant attributes from the `SimulatedUniverse` module)

observationPossible
1D NumPy ndarray (length is number of planets in the system under observation) containing boolean values where True is an observable planet

observed
1D NumPy ndarray which contains the number of observations for each planet in the simulated universe

Outputs

s
1D NumPy array (length is number of planets in the system under observation) containing apparent separation of planets (astropy Quantity with units of distance)

dMag
1D NumPy ndarray (length is number of planets in the system under observation) containing Δmag for each planet

Ip
1D NumPy ndarray (length is number of planets in the system under observation) containing irradiance (astropy

Quantity with units of $\frac{1}{m^2 \cdot nm \cdot s}$)

DRM

Python dictionary containing survey simulation results of current observation as key:value pairs

observed

1D NumPy ndarray which contains the number of observations for each planet in the simulated universe

5.13.6 observation_characterization Sub-task

The `observation_characterization` sub-task is called by the `run_sim` task to determine if characterizations are to be performed and calculate relevant characterization information to be used later in the observation simulation.

Inputs

observationPossible

1D NumPy ndarray (length is number of planets in the system under observation) containing boolean values where True is an observable planet

pInds

1D NumPy ndarray of indices of planets belonging to the target star (used to get relevant attributes from the `SimulatedUniverse` module)

s_ind

Index of target star in target list

spectra

NumPy ndarray where 1 denotes spectra for a planet that has been captured, 0 denotes spectra for a planet that has not been captured

s

1D NumPy array (length is number of planets in the system under observation) containing apparent separation of planets (astropy Quantity with units of distance)

Ip

1D NumPy ndarray (length is number of planets in the system under observation) containing irradiance (astropy Quantity with units of $\frac{1}{m^2 \cdot nm \cdot s}$)

DRM

Python dictionary containing survey simulation results of current observation as key:value pairs

FA

Boolean where True is False Alarm

t_int

Integration time (astropy Quantity with units of time)

Outputs

DRM

Python dictionary containing survey simulation results of current observation as key:value pairs

FA

Boolean where True is False Alarm

spectra

NumPy ndarray where 1 denotes spectra for a planet that has been captured, 0 denotes spectra for a planet that has not been captured

5.13.7 next_target Sub-task

The `next_target` sub-task is called from the `run_sim` task to determine the index of the next star from the target list for observation.

Inputs

s_ind

Index of current star from the target list

targlist

Target List module (see 5.11)

revisit_list

NumPy ndarray containing index of target star and time in *day* of target stars from the target list to revisit

extended_list
1D NumPy ndarray containing the indices of stars in the target list to consider if in extended mission time
DRM
Python dictionary containing survey simulation results of current observation as key:value pairs

Outputs
new_s_ind
Index of next target star in the target list
DRM
Python dictionary containing survey simulation results of current observation as key:value pairs

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 could also potentially save time by cloning survey module objects and reinitializing only those sub-modules that have stochastic elements (i.e., the simulated universe).

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.

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