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

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

ABSTRACT

This document describes the extensible, modular, open source software framework EXOSIMS. EXOSIMS creates end-to-end simulations of space-based exoplanet imaging missions using stand-alone software modules. The input/output interfaces of each module and interactions of modules with each other are presented to give guidance on mission specific modifications to the EXOSIMS framework. Last Update: October 5, 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. 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

Common Reference Frame

Heliocentric Equatorial (HE)

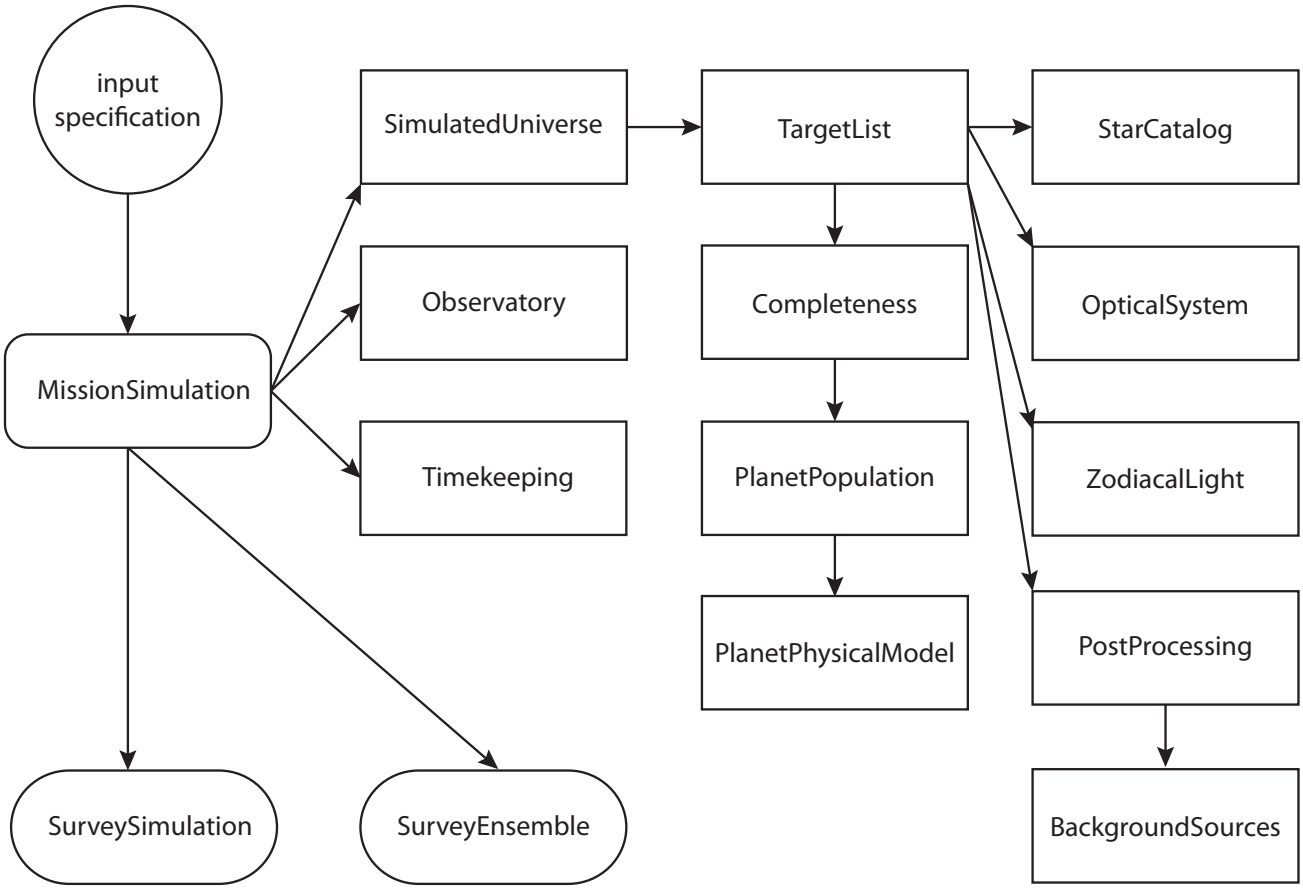

Fig. 1. Schematic depiction of the instantiation path of all EXOSIMS modules. The entry point to the backbone is the construction of a MissionSimulation object, which causes the instantiation of all other module objects. All objects are instantiated in the order shown here, with SurveySimulation and 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.

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
hashlib
importlib
numpy
    numpy.linalg
os
    os.path
pickle/cPickle
scipy
```

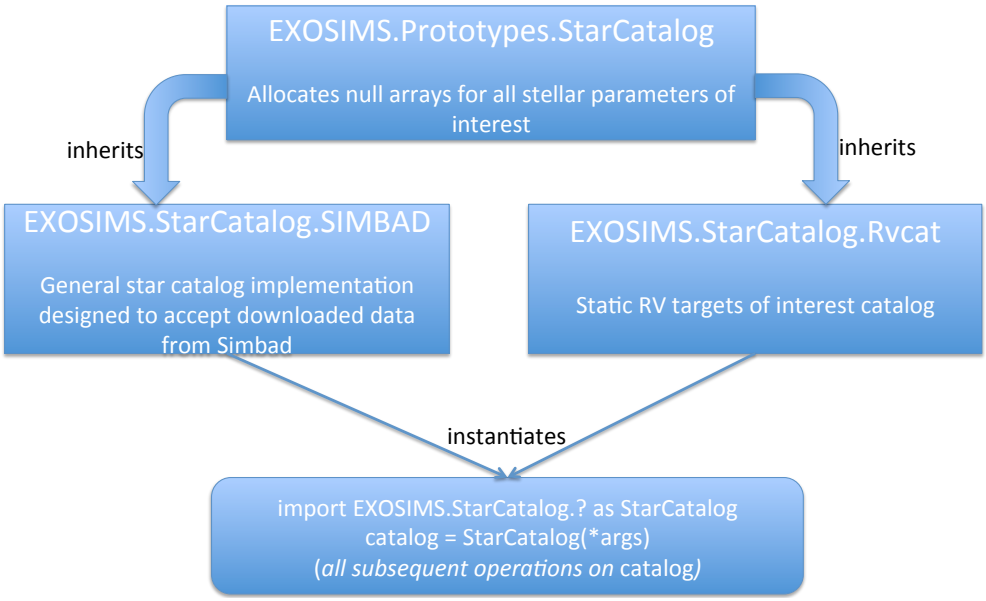

Fig. 2. Schematic of a sample implementation for the three module layers for the Star Catalog module. The Star Catalog prototype (top row) is immutable, specifies the input/output structure of the module along with all common functionality, and is inherited by all Star Catalog class implementations (middle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

```
    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:

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

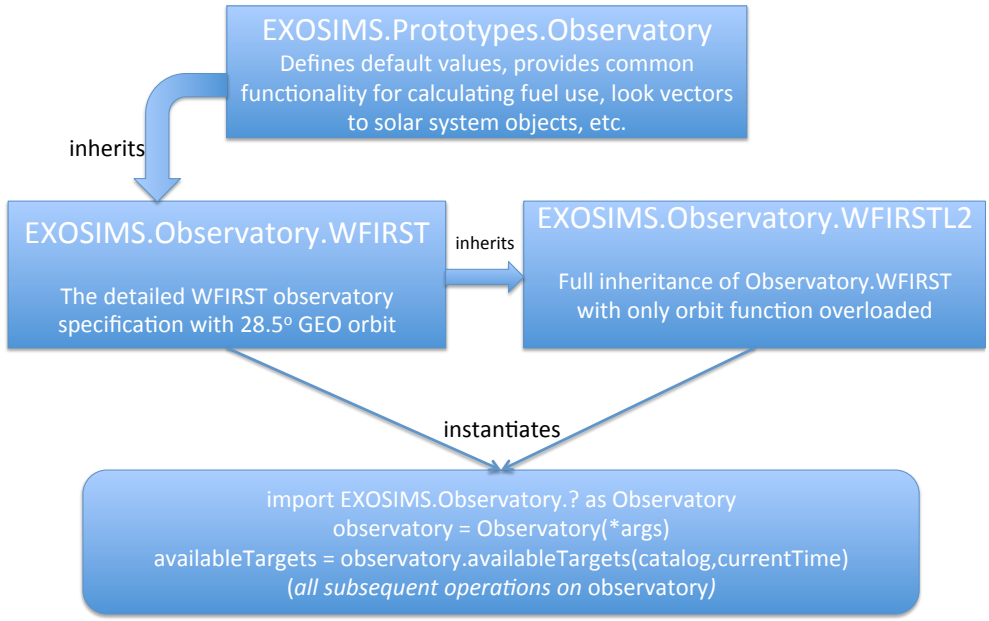

Fig. 3. Schematic of a sample implementation for the three module layers for the Observatory module. The Observatory prototype (top row) is immutable, specifies the input/output structure of the module along with all common functionality, and is inherited by all Observatory class implementations (middle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

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,
  ...
  "scienceInstruments": [
    {
      "name": "imager-EMCCD-PC",
      "QE": value,
      "pitch": value,
      "focal": value,
      "idark": value,
      "CIC": value,
      "sread": value,
      "texp": value,
      "ENF": value
    },
    {
      "type": "spectro-CCD",
      "QE": value,
      "pitch": value,
      "focal": value,
      "idark": value,
      "CIC": value,
      "sread": value,
      "texp": value,
      "ENF": value,
      "Rs": value
    }
  ],
  "starlightSuppressionSystems": [
    {
      "name": "HybridLyotCoronagraph",
      "lam": value,
      "BW": value,
```

```
    "occ_trans": "/data/hlc/occ_trans.fits",
    "core_thruput": "/data/hlc/core_thruput.fits",
    "core_contrast": "/data/hlc/core_contrast.fits",
    "core_area": "/data/hlc/core_area.fits",
    "PSF": "/data/hlc/PSF.fits",
    "IWA": value,
    "OWA": value,
    "ohTime": value
  },
  {
    "name": "MultipleDistanceOcculter",
    "occulter": true,
    "lam": value,
    "BW": value,
    "occ_trans": "/data/mdo/occ_trans.fits",
    "core_thruput": "/data/mdo/core_thruput.fits",
    "core_contrast": "/data/mdo/core_contrast.fits",
    "core_area": "/data/mdo/core_area.fits",
    "PSF": "/data/mdo/PSF.fits",
    "IWA": value,
    "OWA": value,
    "occulterDiameter": value,
    "NocculterDistances": 2,
    "occulterDistances: [
      {
       "occulterDistance": value,
       "occulterBlueEdge": value,
       "occulterRedEdge": value
      },
      {
       "occulterDistance": value,
       "occulterBlueEdge": value,
       "occulterRedEdge": value
      }
    ],
    "occulterWetMass": value,
    "occulterDryMass": value
  }
],
  "observingModes": [
  {
    "instName": "imager-EMCCD-PC",
    "systName": "HybridLyotCoronagraph",
    "detection": true
  },
  {
    "instName": "imager-EMCCD-PC",
    "systName": "HybridLyotCoronagraph",
    "lam": value,
    "BW": value
  },
  {
    "instName": "spectro-CCD",
    "systName": "HybridLyotCoronagraph",
    "SNR": value,
    "timeMultiplier": value
  },
  {
```

```
      "instName": "spectro-CCD",
      "systName": "MultipleDistanceOcculter"
    }
  ],
  "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:

`missionLife`	(float) The total mission lifetime in units of *year*. When the mission time is equal or greater to this value, the mission simulation stops.
`missionPortion`	(float) The portion of the mission dedicated to exoplanet science, given as a value between 0 and 1. The mission simulation stops when the total integration time plus observation overhead time is equal to the missionLife $\times$ missionPortion.
`missionStart`	(float) Mission start time in *MJD*.
`extendedLife`	(float) Extended mission time in units of *year*. Extended life typically differs from the primary mission in some way—most typically only revisits are allowed
`pupilDiam`	(float) Entrance pupil diameter in units of *m*.
`shapeFac`	(float) Telescope aperture shape factor.
`obscurFac`	(float) Obscuration factor due to secondary mirror and spiders.
`attenuation`	(float) Non-coronagraph attenuation, equal to the throughput of the optical system without the coronagraph elements.
`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.
`telescopeKeepout`	(float) Telescope keepout angle in units of *deg*
`forceStaticEphem`	(boolean) Force use of static solar system ephemeris if set to True, even if jplephem module is present.
`spkpath`	(string) Full path to SPK kernel file.
`settlingTime`	(float) Amount of time needed for observatory to settle after a repointing in units of *day*.
`lam`	(float) Default detection central wavelength in units of *nm*.
`deltaLam`	(float) Default detection bandwidth in units of *nm*.

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

`SNR` (float) Default signal to Noise Ratio threshold.

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

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

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

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

`FAP` (float) Detection false alarm probability

`MDP` (float) Missed detection probability

`arange` (float) 1×2 list of semi-major axis range in units of AU.

`erange` (float) 1×2 list of eccentricity range.

`Irange` (float) 1×2 list of inclination range in units of *deg*.

`Orange` (float) 1×2 list of ascension of the ascending node range in units of *deg*.

`wrange` (float) 1×2 list of argument of perigee range in units of *deg*.

`prange` (float) 1×2 list of planetary geometric albedo range.

`Rprange` (float) 1×2 list of planetary radius range in Earth radii.

`Mprange` (float) 1×2 list of planetary mass range in Earth masses.

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

`constrainOrbits` (boolean) True means planetary orbits are constrained to never leave the semi-major axis range (arange).

`thrust` (float) Occulter slew thrust in units of *mN*.

`slewIsp` (float) Occulter slew specific impulse in units of *s*.

`scMass` (float) Occulter (maneuvering spacecraft) initial wet mass in units of *kg*.

`dryMass` (float) Occulter (maneuvering spacecraft) dry mass in units of *kg*.

`coMass` (float) Telescope (or non-maneuvering spacecraft) mass in units of *kg*.

`skIsp` (float) Specific impulse for station keeping in units of *s*.

`defburnPortion` (float) Default burn portion for slewing.

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 Star Catalog

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

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

5.1.1 Star Catalog Object Attribute Initialization

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

Input

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

Attributes

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

5.2 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 §5.2.2).

Certain parameter models may be empirically derived while others may come from analyses of observational surveys. This module also encodes the limits on all parameters to be used for sampling the distributions and determining derived cutoff values such as the maximum target distance for a given instrument's IWA.

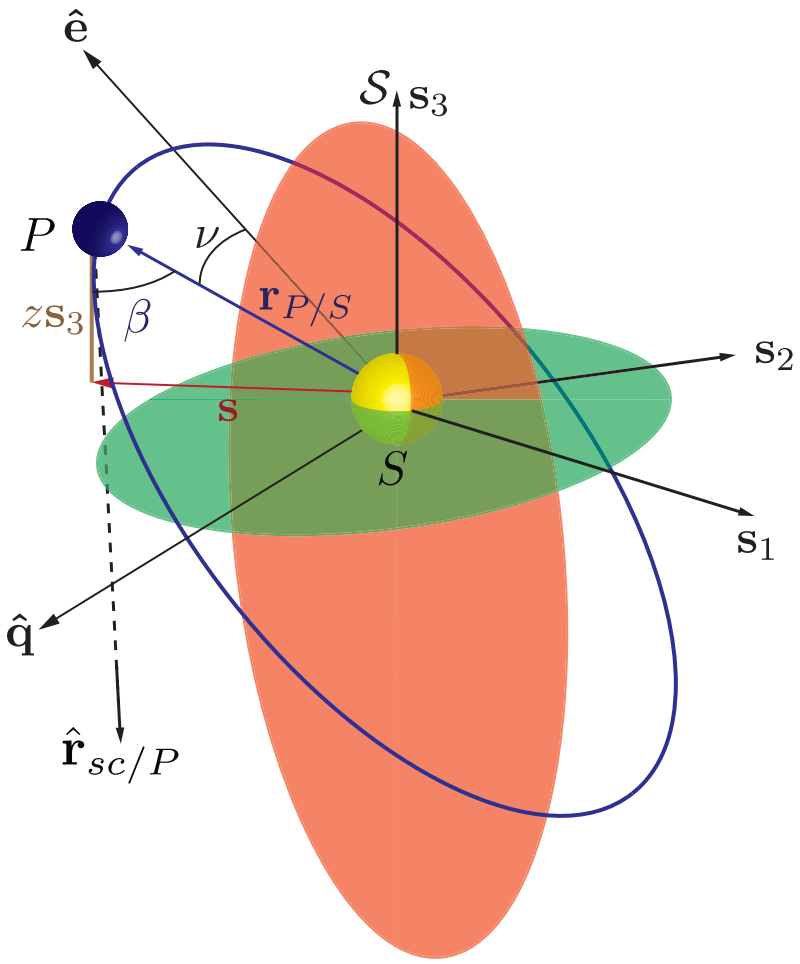

Fig. 4. Definition of reference frames and coordinates of simulated exosystems. The observer lies along the negative $\mathbf{s}_3$ axis so that the observer-star unit vector is $+\mathbf{s}_3$.

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.

5.2.1 Planet Population Object Attribute Initialization

Input

The following are all entries in the passed specs dictionary (derived from the JSON script file or another dictionary). Values not specified will be replaced with defaults, as listed. It is important to note that many of these (in particular mass and radius) may be mutually dependent, and so some implementation may choose to only use some for inputs and set the rest via the physical models.

arange (float 1×2 array)

Semi-major axis range in units of *AU*. Default value is [0.01, 100]

erange (float 1×2 array)
Eccentricity range. Default value is [0.01,0.99]
Irange (float 1×2 array)
Inclination range in units of *deg*. Default value is [0,180]
Orange (float 1×2 array)
Ascension of the ascending node range in units of *deg*. Default value is [0,360]
wrange (float 1×2 array)
Perigee range in units of *deg*. Default value is [0,360]
prange (float 1×2 array)
Planetary geometric albedo range. Default value is [0.1,0.6]
Rprange (float 1×2 array)
Planetary Radius in Earth radii. Default value is [1, 30]
Mprange (float 1×2 array)
Planetary mass in Earth masses. Default value is [1, 4131]
scaleOrbits (boolean)
Boolean where True means planetary orbits are scaled by the square root of stellar luminosity. Default value is False.
constrainOrbits (boolean)
Boolean where True means planetary orbits are constrained to never leave the semi-major axis range (arange). Default value is False.
eta (float)
The average occurrence rate of planets per star for the entire population. The expected number of planets generated per simulation is equal to the product of eta with the total number of targets. Note that this is the expectation value *only*—the actual number of planets generated in a given simulation may vary depending on the specific method of sampling the population.

Attributes
PlanetPhysicalModel (PlanetPhysicalModel module)
PlanetPhysicalModel class object
arange (astropy Quantity 1×2 array)
Semi-major axis range defined as [a_min, a_max] in units of *AU*
erange (float 1×2 ndarray)
Eccentricity range defined as [e_min, e_max]
Irange (astropy Quantity 1×2 array)
Planetary orbital inclination range defined as [I_min, I_max] in units of *deg*
Orange (astropy Quantity 1×2 array)
Right ascension of the ascending node range defined as [O_min, O_max] in units of *deg*
wrange (astropy Quantity 1×2 array)
Argument of perigee range defined as [w_min, w_max] in units of *deg*
prange (float 1×2 ndarray)
Planetary geometric albedo range defined as [p_min, p_max]
Rprange (astropy Quantity 1×2 array)
Planetary radius range defined as [R_min, R_max] in units of *km*
Mprange (astropy Quantity 1×2 array)
Planetary mass range defined as [Mp_min, Mp_max] in units of *kg*
rrange (astropy Quantity 1×2 array)
Planetary orbital radius range defined as [r_min, r_max] derived from PlanetPopulation.arange and PlanetPopulation.erange, in units of *AU*
scaleOrbits (boolean)
Boolean where True means planetary orbits are scaled by the square root of stellar luminosity.
constrainOribts (boolean)
Boolean where True means planetary orbits are constrained to never leave the semi-major axis range (arange). If set to True, an additional method (`gen_eccen_from_sma`) must be provided by the implementation—see below.
eta (float)
The average occurrence rate of planets per star for the entire population.

5.2.2 Planet Population Value Generators

For each of the parameters represented by the input attributes, the planet population object will provide a method that returns random values for the 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 in units of *AU*
`gen_eccen`	Returns eccentricity float values
`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 float values such that $a(1-e) \geq a_{\text{min}}$ and $a(1+e) \leq a_{\text{max}}$.
`gen_I`	Returns values of orbital inclination in units of *deg*
`gen_O`	Returns longitude of the ascending node values in units of *deg*
`gen_w`	Returns argument of perigee values in units of *deg*
`gen_albedo`	Returns planetary geometric albedo float values
`gen_radius`	Returns planetary radius values in units of *m*
`gen_mass`	Returns planetary mass values in units of *kg*
`dist_sma`	Provides the probability density function for the semi-major axis
`dist_eccen`	Provides the probability density function for the eccentricity
`dist_albedo`	Provides the probability density function for the albedo
`dist_radius`	Provides the probability density function for the radius

5.3 Planet Physical Model

The Planet Physical Model module contains models of the light emitted or reflected by planets in the wavelength bands under investigation by the current mission simulation. It takes as inputs the physical quantities sampled from the distributions in the Planet Population module and generates synthetic spectra (or band photometry, as appropriate). The specific implementation of this module can vary greatly, and can be based on any of the many available planetary geometric albedo, spectra and phase curve models. 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_radius_from_mass`	Provides a method to calculate planetary radii from their masses
`calc_mass_from_radius`	Provides a method to calculate planetary masses from their radii
`calc_Phi`	Provides a method to calculate the value of the planet phase function given its phase angle. The prototype implementation uses the Lambert phase function.

5.4 Optical System

The Optical System module contains all of the necessary information to describe the planet signal and the background noise, and calculate the integration time for a given observation. This first requires encoding the design of the telescope such as the optics attenuation, the diameter of the entrance pupil and the fraction of it that is obscured (by spiders and secondary mirror). A description of the science instruments is also required, with detector details such as read noise, dark current, and readout cycle. The baseline is assumed to be an imager and a spectrograph. Finally, the Optical System must include the performance of every selected starlight suppression systems, whether it be an internal coronagraph or an external occulter. The Optical System module also contains all the mission observing modes. Each mode is defined by a combination of a science instrument and a starlight suppression system, operating in a given spectral window.

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

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

`calc_maxintTime` Called to calculate the maximum integration time for observable stars in the target list (see §5.4.2)

`calc_intTime` Called by `calc_maxintTime` to calculate the integration times for specific values of planet angular separation and flux ratio (see §5.4.3)

`Cp_Cb_Csp` Called by `calc_intTime` to calculate the electron count rates for planet signal, background noise, and speckle residuals (see §5.4.4)

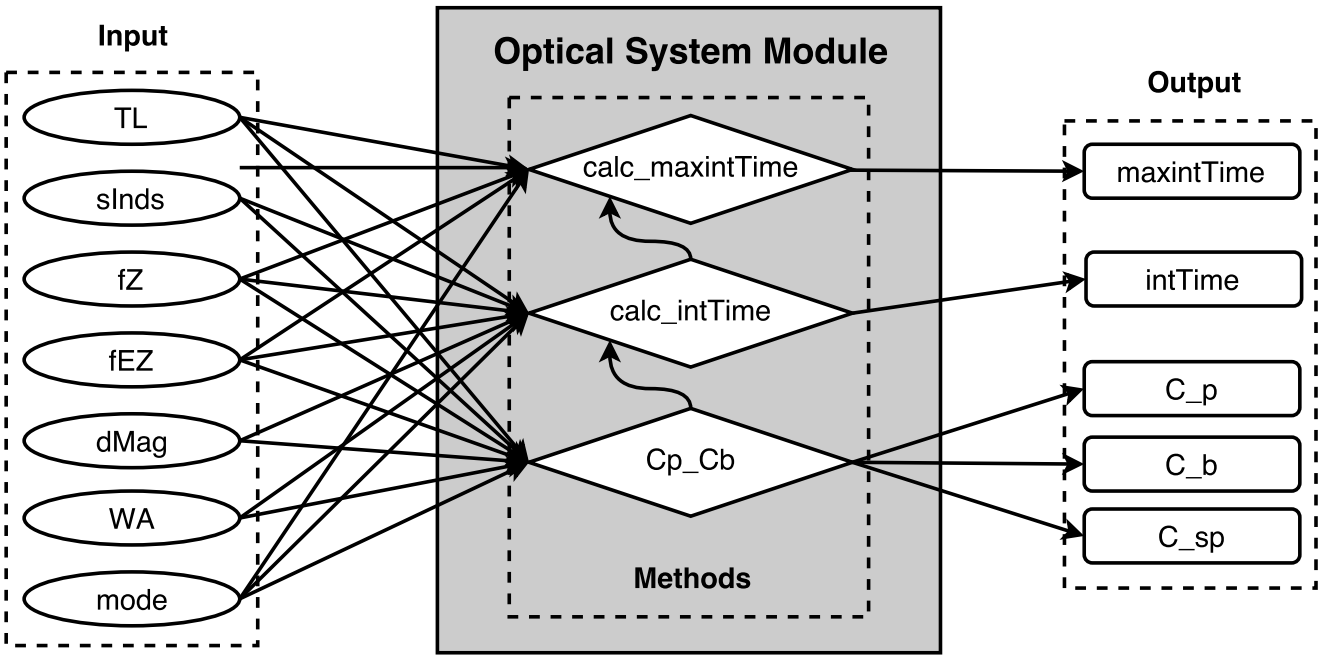

Fig. 5. Depiction of Optical System module methods including input and output (see §5.4.2, S5.4.3 and §5.4.4).

5.4.1 Optical System Object Attribute Initialization

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

Input

obscurFac (float)

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

shapeFac (float)

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

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

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

attenuation (float)
Non-coronagraph attenuation, equal to the throughput of the optical system without the coronagraph elements. Default value is 0.5.

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

Ndark (float)
Number of dark frames used. Default value is 10.

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

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

dMagLim (float)
Fundamental limiting Δmag (difference in magnitude between star and planet). Default value is 22.5.

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

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

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

focal (float)
Focal length in units of *m*. Default value is 100.

idark (float)
Detector dark-current rate of electrons per pixel in units of $1/s$. Default value is 5e-4.

CIC (float)
(Specific to CCDs) Clock-induced-charge of electrons per frame. Default value is 5e-3.

sread (float)
Detector effective read noise rate of electrons per frame, including any gain (e.g. electron multiplication gain). Default value is 0.2.

texp (float)
Exposure time per frame in units of *s*. Default value is 1000.

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

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

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

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

core_mean_intensity, *core_area*.

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

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

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

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

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

core_thruput (float, callable)
System throughput in the FWHM region of the planet PSF core. Default is scalar 1e-2.

core_contrast (float, callable)
System contrast defined as the starlight residual normalized intensity in the PSF core, divided by the core throughput. Default is scalar 1e-9.

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

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

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

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

PSF (float, callable)
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 (float)
Sampling of the PSF in units of *arcsec* per pixel. Default value is 10.

ohTime (float)
Optical system overhead time in units of *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.

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

occulterDiameter (float)
Occulter diameter in units of *m*. Measured petal tip-to-tip.

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

occulterDistance (float)
Telescope-occulter separation in units of *km*.

occulterBlueEdge (float)
Occulter blue end of wavelength band in units of *nm*.

occulterRedEdge (float)
Occulter red end of wavelength band in units of *nm*.

observingModes (list of dicts)
List of dictionaries containing specific attributes of all mission observing modes. Each observing mode is a combi-

nation of an instrument and a system, operating at a given wavelength, which by default is the wavelength defined in the starlight suppression system of the observing mode. If an observing mode is operating at a different wavelength than the system default wavelength, then this new wavelength must be added to the observing mode, and the system performance will be automatically rescaled to the new wavelength. If no observing mode is defined, the default observing mode simply combines the first instrument and the first system.

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

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

inst (dict)
Selected instrument of the observing mode.

syst (dict)
Selected system of the observing mode.

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

SNR (float)
Signal-to-noise ratio threshold. Defaults to 5.

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

lam (float)
Central wavelength in units of nm. Defaults to corresponding system value.

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

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

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

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

Attributes

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

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

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

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

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

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

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 imager, and the second the spectrograph (IFS). Only required attribute is 'name'. See above for other commonly used attributes.
starlightSuppressionSystems (list of dicts)
List of dictionaries containing all supplied starlight suppression system attributes. Typically the first system will be used with the imager, and the second with the IFS. Only required attribute is 'name'. See above for other commonly used attributes.
observingModes (list of dicts)
List of dictionaries containing all mission observing modes. Only required attribute are 'instName' and 'systName'. See above for other commonly used attributes.
detectionMode (dict)
Dictionary containing the observing mode used for detection. Default detection mode is first imager mode.

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 SurveySimulation module.

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

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.

Input
TL (TargetList module)
TargetList class object, see §5.9 for definition of available attributes
sInds (integer ndarray)
Integer indices of the stars of interest, with the length of the number of planets of interest
fZ (astropy Quantity array)
Surface brightness of local zodiacal light in units of $1/arcsec^2$

fEZ (astropy Quantity array)
Surface brightness of exo-zodiacal light in units of $1/arcsec^2$
dMag (float ndarray)
Differences in magnitude between planets and their host star.
WA (astropy Quantity array)
Working angles of the planets of interest in units of *mas*
mode (dict)
Selected observing mode

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

5.4.4 Cp_Cb_Csp Method

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

Input
TL (TargetList module)
TargetList class object, see §5.9 for definition of available attributes
sInds (integer ndarray)
Integer indices of the stars of interest, with the length of the number of planets of interest
fZ (astropy Quantity array)
Surface brightness of local zodiacal light in units of $1/arcsec^2$
fEZ (astropy Quantity array)
Surface brightness of exo-zodiacal light in units of $1/arcsec^2$
dMag (float ndarray)
Differences in magnitude between planets and their host star.
WA (astropy Quantity array)
Working angles of the planets of interest in units of *mas*
mode (dict)
Selected observing mode

Output
C_p (astropy Quantity array)
Planet signal electron count rate in units of $1/s$
C_b (astropy Quantity array)
Background noise electron count rate in units of $1/s$
C_sp (astropy Quantity array)
Residual speckle spatial structure (systematic error) in units of $1/s$

5.5 Zodiacal Light

The input and output of the Zodiacal Light methods are depicted in Figure 6. The Zodiacal Light module contains two methods:

`fZ` Calculates the surface brightness of local zodiacal light (see §5.5.2)
`fEZ` Calculates the surface brightness of exozodiacal light (see §5.5.3)

5.5.1 Zodiacal Light Object Attribute Initialization

Input
magZ (float)
Zodiacal light brightness magnitude (per $arcsec^2$). Defaults to 23.
magEZ (float)
Exo-zodiacal light brightness magnitude (per $arcsec^2$). Defaults to 22.
varEZ (float)
Exo-zodiacal light variation (variance of log-normal distribution). Defaults to 0 (constant exo-zodiacal light).

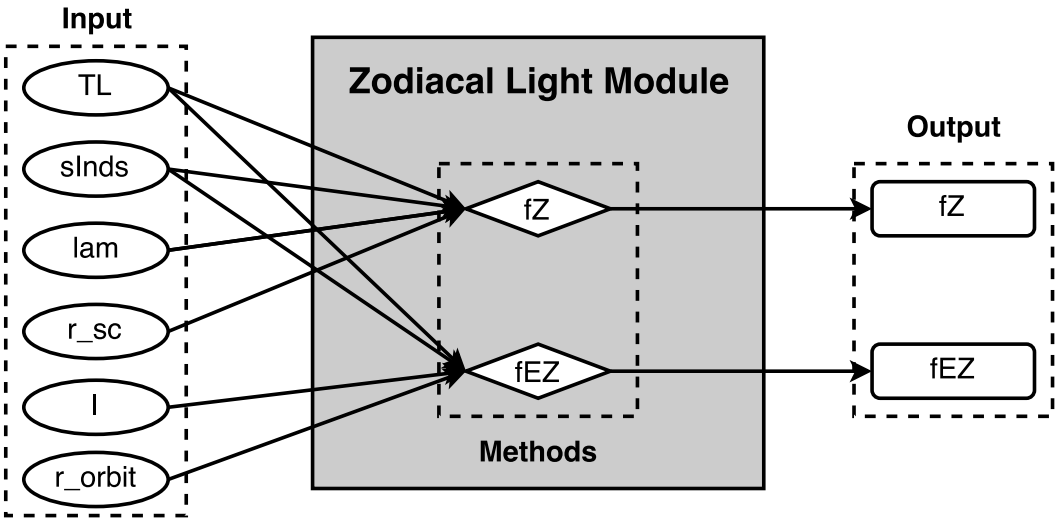

Fig. 6. Depiction of Zodiacal Light module methods including input and output (see §5.5.2 and §5.5.3).

Attributes

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

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

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

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

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

5.5.2 fZ Method

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

Input

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

sInds (integer ndarray)
Integer indices of the stars of interest, with the length of the number of planets of interest

lam (astropy Quantity)
Central wavelength in units of *nm*

r_sc (astropy Quantity n×3 array)
Observatory position vector in units of *km*, for each planet of interest.

Output

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

5.5.3 fEZ Method

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

Input

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

sInds (integer ndarray)
Integer indices of the stars of interest, with the length of the number of planets of interest

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

r_orbit (astropy Quantity n×3 array)
Orbital radii of the planets of interest in units of *AU*

Output

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

5.6 Background Sources

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

5.6.1 dNbackground Method

Input

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

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

Output

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

5.7 Post-Processing

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

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

5.7.1 Post-Processing Object Attribute Initialization

Input

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

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

ppFact (float, callable)
Post-processing contrast factor, between 0 and 1: either a scalar for constant gain, or a two-column array for separation-dependent gain, where the first column contains the angular separation in units of *arcsec*. May be data or FITS filename. Default value is 1.

maxFAfluxratio (float, callable)
Maximum flux ratio that can be obtained by a false alarm: either a scalar for constant flux ratio, or a two-column

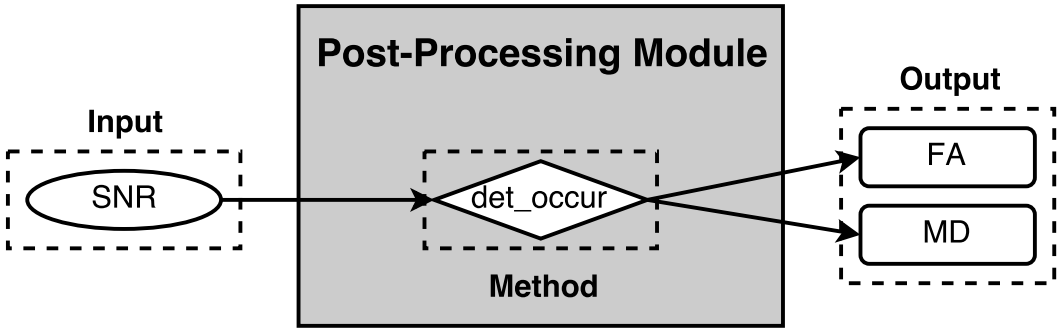

Fig. 7. Depiction of Post-Processing module method including input and output (see §5.7.2).

array for separation-dependent flux ratio, where the first column contains the angular separation in units of arcsec. May be data or FITS filename. Default value is 1e-6.

Attributes

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

FAP (float)
Detection false alarm probability

MDP (float)
Missed detection probability

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

maxFAfluxratio (float, callable)
Maximum flux ratio that can be obtained by a false alarm, between 0 and 1.

5.7.2 det_occur Method

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

Input

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

Output

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

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

5.8 Completeness

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

The Completeness module contains the following methods:

`target_completeness` Generates initial completeness values for each star in the target list (see §5.8.2)
`completeness_update` Updates the completeness values following an observation (see §5.8.4)

5.8.1 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 (PlanetPopulation module)
PlanetPopulation object (see 5.2)

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

5.8.2 target_completeness Method

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

Input

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

Output

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

5.8.3 gen_update Method

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

Input

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

5.8.4 completeness_update Method

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

Input

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

sInds (integer)
Indices of stars to update

dt (astropy Quantity)
Time since initial completeness

Output

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

5.9 Target List

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

5.9.1 Target List Object Attribute Initialization

Input

keepStarCatalog (boolean)
Boolean representing whether to delete the star catalog object after the target list is assembled (defaults to False).

If True, object reference will be available from TargetList class object.

minComp (float)

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 5.1)

StarCatalog (StarCatalog module)

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

PlanetPopulation (PlanetPopulation module)

PlanetPopulation class object (see 5.2)

PlanetPhysicalModel (PlanetPhysicalModel module)

PlanetPhysicalModel class object (see 5.3)

OpticalSystem (OpticalSystem module)

OpticalSystem class object (see 5.4)

ZodiacalLight (ZodiacalLight module)

ZodiacalLight class object (see 5.5)

BackgroundSources (BackgroundSources module)

BackgroundSources class object (see 5.6)

PostProcessing (PostProcessing module)

PostProcessing class object (see 5.7)

Completeness (Completeness module)

Completeness class object (see 5.8)

tint0 (astropy Quantity array)

Integration time for each target star in units of *day*, for a minimum detectable delta magnitude *dMagLim*. Calculated from `OpticalSystem.calc_intTime` §5.4.3

comp0 (float ndarray)

Completeness value for each target star. Calculated from `Completeness.target_completeness` §5.8.2

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

5.9.2 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).

Input

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

5.9.3 populate_target_list Method

The `populate_target_list` method is responsible for populating values from the star catalog (or any other source) into the target list attributes. It has not specific inputs and outputs, but is always passed the full specification dictionary, and

updates all relevant Target List attributes. This method is called from the prototype constructor, and does not need to be called from the implementation constructor when overloaded in the implementation. The prototype implementation copies values directly from star catalog and removes stars with any NaN attributes. It also calls the `target_completeness` in the Completeness module (§5.8.2) and the `calc_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).

5.9.4 filter_target_list Method

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

5.9.5 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

5.10 Simulated Universe

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

All planetary parameters are generated in the constructor via calls to the appropriate value generating functions in the planet population module. The input and updated attributes of the Simulated Universe methods are depicted in Figure 8. The Simulated Universe module contains the following methods:

`gen_physical_properties`	Populates the orbital elements and physical characteristics of all planets (see §5.10.2)
`init_systems`	Finds initial time-dependant parameters such as position and velocity vectors, along with exo-zodiacal surface brightness, delta magnitude, and working angle (see §5.10.3)
`propag_system`	Propagates planet time-dependant parameters (position, velocity, distance, separation, exozodiacal brightness, delta magnitude, and working angle) in time (see §5.10.4)

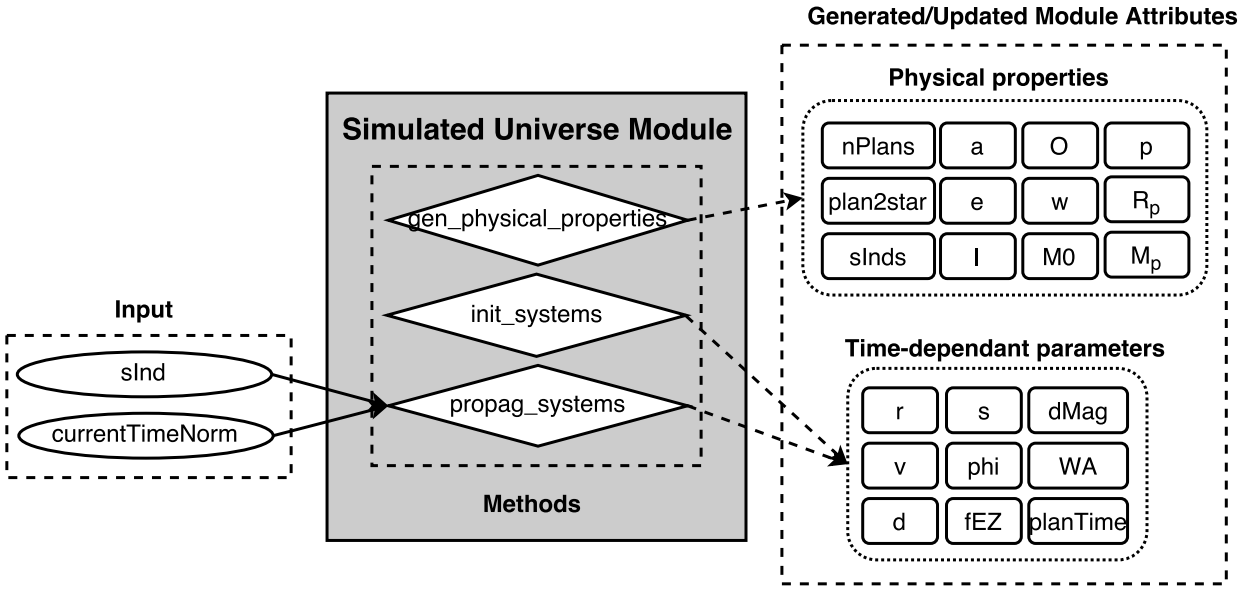

Fig. 8. Depiction of Simulated Universe module methods including input and updated attributes (see 5.10.2, 5.10.3, and 5.10.4).

5.10.1 Attributes

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

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

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

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

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

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

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

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

nPlans (integer)
Total number of planets

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

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

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

e (float ndarray)
Planet eccentricity

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

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

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

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

p (float ndarray)
Planet albedo
Rp (astropy Quantity array)
Planet radius in units of *km*
Mp (astropy Quantity array)
Planet mass in units of *kg*
r (astropy Quantity n×3 array)
Planet position vector in units of *AU*
v (astropy Quantity n×3 array)
Planet velocity vector in units of AU/day
d (astropy Quantity array)
Planet-star distances in units of *AU*
s (astropy Quantity array)
Planet-star apparent separations in units of *AU*
phi (float ndarray)
Planet phase function, given its phase angle
fEZ (astropy Quantity array)
Surface brightness of exozodiacal light in units of $1/arcsec2$, determined from `ZodiacalLight.fEZ` §5.5.3
dMag (float ndarray)
Differences in magnitude between planets and their host star
WA (astropy Quantity array)
Working angles of the planets of interest in units of *mas*
planTime (astropy Quantity array)
Contains the last time the planet was observed in units of day, for planet position propagation

5.10.2 gen_physical_properties Method

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

Generated Module Attributes
nPlans (integer)
Total number of planets
plan2star (integer ndarray)
Indices mapping planets to target stars in TargetList
sInds (integer ndarray)
Unique indices of stars with planets in TargetList
a (astropy Quantity array)
Planet semi-major axis in units of *AU*
e (float ndarray)
Planet eccentricity
I (astropy Quantity array)
Planet inclination in units of *deg*
O (astropy Quantity array)
Planet right ascension of the ascending node in units of *deg*
w (astropy Quantity array)
Planet argument of perigee in units of *deg*
M0 (astropy Quantity array)
Initial mean anomaly in units of *deg*
p (float ndarray)
Planet albedo
Mp (astropy Quantity array)
Planet mass in units of *kg*
Rp (astropy Quantity array)
Planet radius in units of *km*

5.10.3 init_systems Method

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

```
SimulatedUniverse.a
SimulatedUniverse.e
SimulatedUniverse.I
SimulatedUniverse.O
SimulatedUniverse.w
SimulatedUniverse.M0
SimulatedUniverse.Mp
```

Generated Module Attributes

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

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

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

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

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

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

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

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

planTime (astropy Quantity array)
Contains the last time the planet was observed in units of day, for system propagation

5.10.4 propag_system Method

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

Input

sInd (integer)
Index of the target system of interest

currentTimeNorm (astropy Quantity)
Current mission time normalized to zero at mission start in units of *day*

Updated Module Attributes

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

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

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

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

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

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

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

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

planTime (astropy Quantity array)
Contains the last time the planet was observed in units of day, for system propagation

5.11 Observatory

The Observatory module contains all of the information specific to the space-based observatory not included in the Optical System module. The module has 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 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 Target List module, current mission time and orbit position 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 with an L2 halo orbit, which is the new mission baseline.

The input and output of the Observatory module methods are depicted in Figure 9.

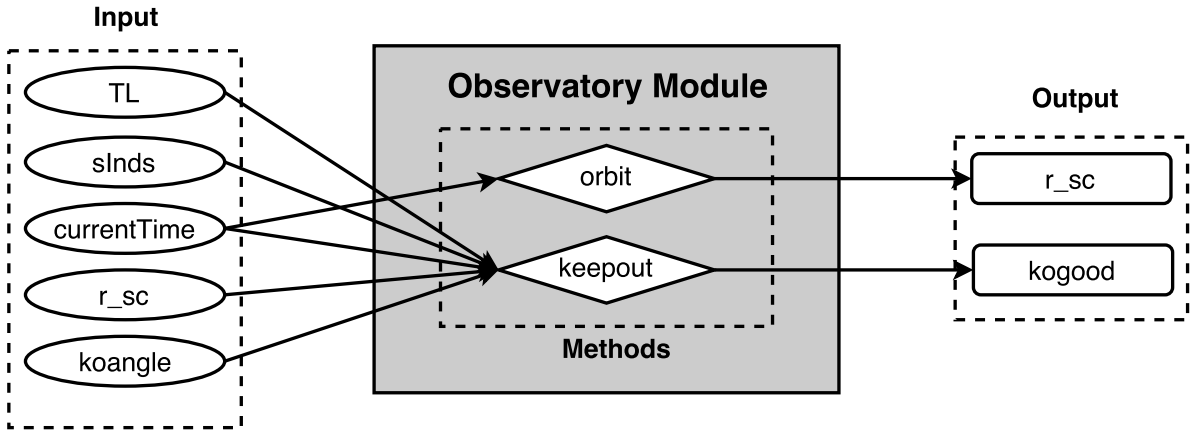

Fig. 9. Depiction of Observatory module methods including input and output (see §5.11.2 and §5.11.3).

5.11.1 Observatory Object Attribute Initialization

Input

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

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

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

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

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

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

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

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

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

spkpath (string)
String with full path to SPK kernel file (only used if using jplephem for solar system body propagation - see 5.11.4).

forceStaticEphem (boolean)
Boolean, forcing use of static solar system ephemeris if set to True, even if jplephem module is present (see 5.11.4). Default value is False.

Attributes

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

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

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

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

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

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

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

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

defburnPortion (float)
Default burn portion for slewing

flowRate (astropy Quantity)
Slew flow rate derived from thrust and slewIsp in units of kg/day

5.11.2 orbit Method

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

Input

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

Output

r_sc (astropy Quantity n×3 array)

Observatory orbit position in HE reference frame at current mission time in units of *km*

5.11.3 keepout Method

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

Input

TL (TargetList module)

TargetList class object, see §5.9 for definition of available attributes

sInds (integer ndarray)

Integer indices of the stars of interest, with the length of the number of planets of interest

currentTime (astropy Time array)

Current absolute mission time in MJD

r_sc (astropy Quantity n×3 array)

Observatory orbit position in HE reference frame at current mission time in units of *km*

koangle (astropy Quantity)

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

Output

kogood (boolean ndarray)

True is a target unobstructed and observable, and False is a target unobservable due to obstructions in the keepout zone.

5.11.4 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.

Input

currentTime (astropy Time)

Current absolute mission time in MJD

bodyname (string)

Solar system object name, capitalized by convention

Output

r_body (astropy Quantity 1×3 array)

Heliocentric equatorial position vector in units of *km*

5.12 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 units of *day*). The Time Keeping module also provides functionality for converting between this time measure and standard measures such as Julian Day Number and UTC time.

The input, output and updated attributes of the Time Keeping methods are depicted in Figure 10. The Time Keeping module contains two methods:

`allocate_time` Allocates a temporal block of width *dt*, advancing the observation window if needed, and updates the mission time during a survey simulation (see §5.12.2)

`mission_is_over` Checks if the time allocated for the mission is used up (see §5.12.3)

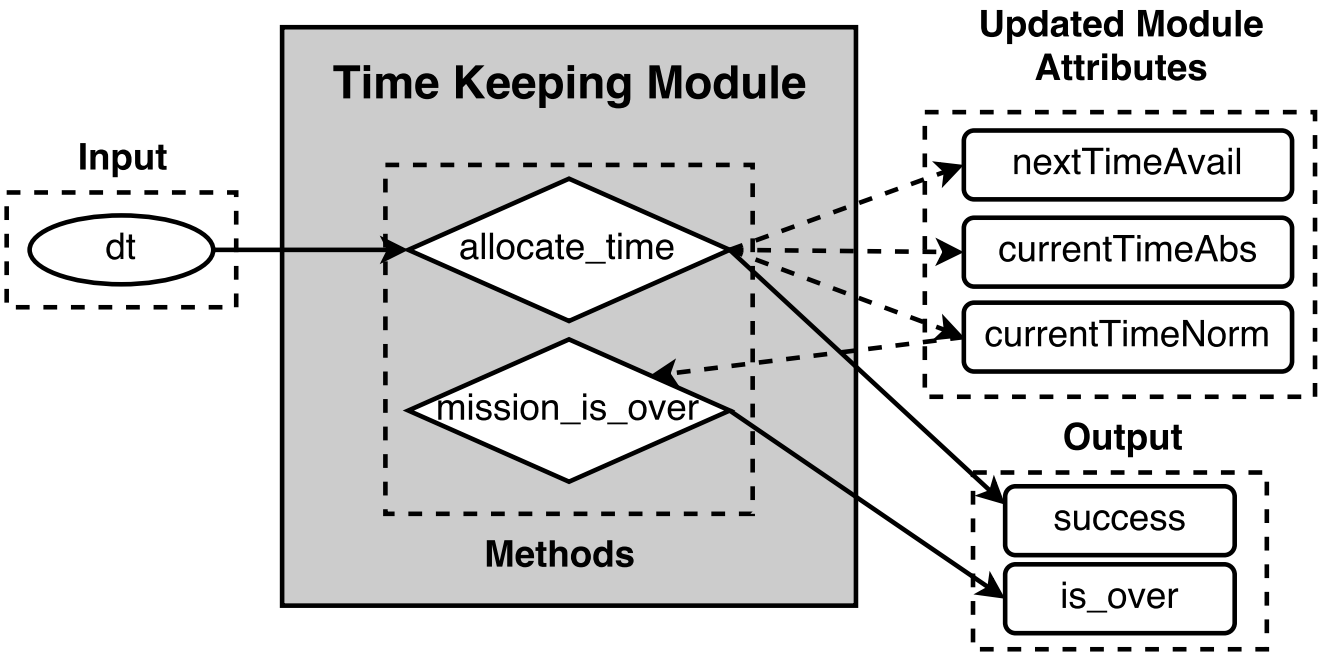

Fig. 10. Depiction of Time Keeping module method including input, output, and updated attributes (see §5.12.2 and §5.12.3).

5.12.1 Time Keeping Object Attribute Initialization

Input

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

missionLife (float)
Total length of mission in units of *year*. Default value is 6.

extendedLife (float)
Extended mission time in units of *year*. Default value is 0. Extended life typically differs from the primary mission in some way—most typically only revisits are allowed.

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

dtAlloc (float)
Default allocated temporal block in units of *day*. Default value is 1.

Attributes

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

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

extendedLife (astropy Quantity)
Extended mission time in units of *year*

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

dtAlloc (astropy Quantity)
Default allocated temporal block in units of *day*

duration (astropy Quantity)
Duration of planet-finding operations in units of *day*, value copied from `OpticalSystem.intCutoff`
missionFinishNorm
Mission finish time in units of *day*
missionFinishAbs (astropy Time)
Mission completion date in *MJD*
nextTimeAvail (astropy Quantity)
Next time available for planet-finding in units of *day*
currentTimeNorm (astropy Quantity)
Current mission time normalized so that start date is 0, in units of *day*
currentTimeAbs (astropy Time)
Current absolute mission time in *MJD*

5.12.2 allocate_time Method

Input

dt (astropy Quantity)
Amount of time requested in units of day

Output

success (boolean)
True if the requested time fits in the widest window, otherwise False

Updated Module Attributes

nexttimeAvail (astropy Quantity)
Next time available for planet-finding in units of *day*
currenttimeNorm (astropy Quantity)
Current mission time normalized so that start date is 0, in units of *day*
currenttimeAbs (astropy Time)
Current absolute mission time in *MJD*

5.12.3 mission_is_over Method

The `mission_is_over` method does not take any explicit inputs. It uses the updated module attribute `currentTimeNorm`.

Output

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

5.13 Survey Simulation

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

The input, output and updated attributes of the Survey Simulation methods are depicted in Figure 11. The Survey Simulation module contains the following methods:

`run_sim`	Performs the survey simulation (see §5.13.2)
`next_target`	Finds index of next target star and calculates its integration time (see §5.13.3)
`observation_detection`	Determines detection status for a given integration time (see §5.13.4)
`observation_characterization`	Determines characterization time and status (see §5.13.5)
`calc_signal_noise`	Calculates the signal and noise fluxes for a given time interval (see §5.13.6) - called by `observation_detection` and `observation_characterization`
`update_occulter_mass`	Updates the occulter wet mass in the Observatory module, and stores all the occulter related values in the DRM array (see §5.13.7)

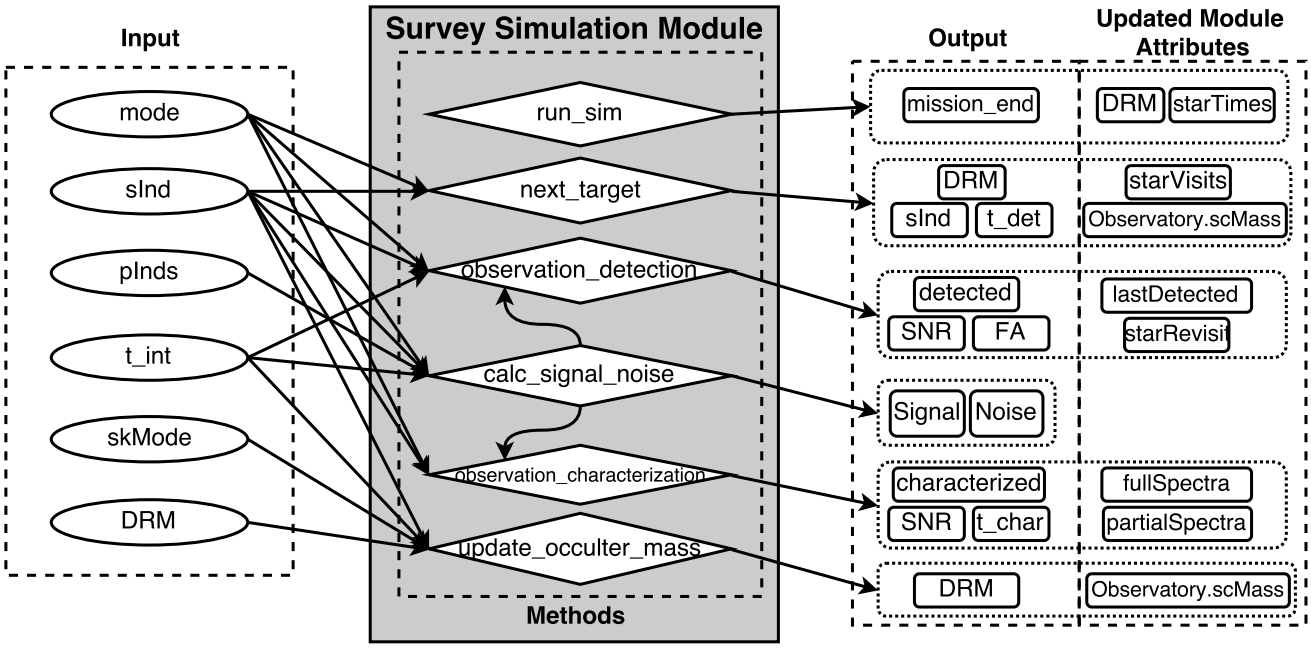

Fig. 11. Depiction of Survey Simulation module method including input, output, and updated attributes (see §5.13.2, §5.13.3, §5.13.4, §5.13.5, §5.13.6 and §5.13.7).

5.13.1 Survey Simulation Object Attribute Initialization

Input

nt_flux (integer)
Observation time sampling, to determine the integration time interval

logLevel (string)
Defines a logging level for the logger handler. Valid levels are: INFO, CRITICAL, ERROR, WARNING, DEBUG (case is ignored). Defaults to INFO.

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

Attributes

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

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

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

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

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

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

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

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

SimulatedUniverse (SimulatedUniverse module)
SimulatedUniverse class object (see 5.10)

Observatory (Observatory module)
Observatory class object (see 5.11)

TimeKeeping (TimeKeeping module)

TimeKeeping class object (see 5.12)

nt_flux (integer)
Observation time sampling, to determine the integration time interval

fullSpectra (boolean ndarray)
Indicates if planet spectra have been captured

partialSpectra (boolean ndarray)
Indicates if planet partial spectra have been captured

starVisits (integer ndarray)
Contains the number of times each target was visited

starTimes (astropy Quantity array)
Contains the last time the star was observed in units of *day*

starRevisit (float n×2 ndarray)
Contains indices of targets to revisit and revisit times of these targets in units of *day*

starExtended (integer ndarray)
Contains indices of targets with detected planets, updated throughout the mission

lastDetected (float n×4 ndarray)
For each target, contains 4 lists with planets' detected status, exozodi brightness (in units of $1/arcsec^2$), delta magnitude, and working angles (in units of *mas*)

DRM (list of dicts)
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`. This method does not take any explicit inputs. It uses the inherited modules to generate a survey simulation.

Output

mission_end (string)
Messaged printed at the end of a survey simulation.

Updated Module Attributes

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

star_ind (integer)
Index of the observed target star

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

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

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

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

det_SNR (float list)
List of detection SNR values, only for observable planets

det_fEZ (float list)
List of exo-zodi surface brightnesses at detection in units of $1/arcsec^2$ for each planet orbiting the observed target star

det_dMag (float list)
List of delta magnitudes at detection for each planet orbiting the observed target star

det_WA (float list)
List of working angles at detection in units of *mas* for each planet orbiting the observed target star

char_mode (dict)
Observing mode selected for characterization. Default is first spectro/IFS mode.

char_time (float)
Integration time for characterization in units of *day*
char_status (integer list)
List of characterization status for each planet orbiting the observed target star, where 1 is full spectrum, -1 partial spectrum, and 0 not characterized
char_SNR (float list)
List of characterization SNR values only for observable planets
char_fEZ (float list)
List of exo-zodi surface brightnesses at characterization in units of $1/arcsec^2$ for each planet orbiting the observed target star
char_dMag (float list)
List of delta magnitudes at characterization for each planet orbiting the observed target star
char_WA (float list)
List of working angles at characterization in units of *mas* for each planet orbiting the observed target star
FA_status (integer)
(if false alarm) Characterization status for a false alarm signal, where 1 is full spectrum, -1 partial spectrum, and 0 not characterized
FA_SNR (float)
(if false alarm) Characterization SNR value for a false alarm signal
FA_fEZ (float)
(if false alarm) Exo-zodi surface brightness for a false alarm signal in units of $1/arcsec^2$
FA_dMag (float)
(if false alarm) Delta magnitude for a false alarm signal
FA_WA (float)
(if false alarm) Working angle for a false alarm signal in units of *mas*
slew_time (float)
(if occulter) Slew time to next target in units of *day*
slew_angle (float)
(if occulter) Slew angle to next target in units of *deg*
slew_dV (float)
(if occulter) Slew ΔV in units of m/s
slew_mass_used (float)
(if occulter) Slew fuel mass used in units of *kg*
det_dV (float)
(if occulter) Detection station-keeping ΔV in units of m/s
det_mass_used (float)
(if occulter) Detection station-keeping fuel mass used in units of *kg*
det_dF_lateral (float)
(if occulter) Detection station-keeping lateral disturbance force on occulter in units of *N*
det_dF_axial (float)
Detection station-keeping axial disturbance force on occulter in units of *N*
char_dV (float)
(if occulter) Characterization station-keeping ΔV in units of m/s
char_mass_used (float)
(if occulter) Characterization station-keeping fuel mass used in units of *kg*
char_dF_lateral (float)
(if occulter) Characterization station-keeping lateral disturbance force on occulter in units of *N*
char_dF_axial (float)
(if occulter) Characterization station-keeping axial disturbance force on occulter in units of *N*
sc_mass (float)
(if occulter) Maneuvering spacecraft mass at the end of target observation in units of *kg*

5.13.3 next_target Method

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

Input

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

mode (dict)
Selected observing mode (from OpticalSystem)

Output

DRM (dict)
Dictionary containing survey simulation results

sInd (integer)
Index of next target star. Defaults to None.

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

5.13.4 observation_detection Method

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

Input

sInd (integer)
Integer index of the star of interest

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

mode (dict)
Selected observing mode (from OpticalSystem)

Output

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

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

FA (boolean)
False alarm (false positive) boolean

5.13.5 observation_characterization Method

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

Input

sInd (integer)
Integer index of the star of interest

mode (dict)
Selected observing mode (from OpticalSystem)

Output

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

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

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

5.13.6 calc_signal_noise Method

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

Input

sInd (integer)
Integer index of the star of interest

pInds (integer)
Integer indices of the planets of interest

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

mode (dict)
Selected observing mode (from OpticalSystem)

Output

Signal (float)
Counts of signal

Noise (float)
Counts of background noise variance

5.13.7 update_occulter_mass Method

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

Input

DRM (dicts)
Contains the results of survey simulation

sInd (integer)
Integer index of the star of interest

t_int (astropy Quantity)
Selected star integration time (for detection or characterization) in units of *day*

skMode (string)
Station keeping observing mode type

Output

DRM (dicts)
Contains the results of survey simulation

5.14 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).