

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

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. These modules are split into input and simulation module groups. 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.

CONTENTS

| | | |
|----------|--|----------|
| 1 | Introduction | 2 |
| 1.1 | Purpose and Scope | 2 |
| 2 | Overview | 2 |
| 3 | Global Specifications | 3 |
| 3.1 | Python Packages | 4 |
| 4 | Backbone | 4 |
| 4.1 | Specification Format | 4 |
| 5 | Input Modules | 5 |
| 5.1 | Optical System Description (OSD) NEEDS UPDATING | 5 |
| 5.1.1 | Optical System Object Attribute Initialization Input/Output Description | 5 |
| 5.2 | Star Catalog (SC) | 7 |
| 5.2.1 | Star Catalog Object Attribute Initialization Input/Output Description | 7 |
| 5.3 | Planet Population Description (PPD) NEEDS UPDATING | 8 |
| 5.3.1 | Planet Population Object Attribute Initialization Input/Output Description | 8 |
| 5.4 | Observatory Definition (OD) | 9 |
| 5.4.1 | Observatory Object Attribute Initialization Input/Output Description | 10 |
| 5.4.2 | Orbit Task Input/Output Description | 10 |
| 5.4.3 | Keepout Task Input/Output Description | 10 |
| 5.5 | Planet Physical Model (PPM) NEEDS UPDATING | 10 |
| 5.6 | Time | 11 |
| 5.6.1 | Time Object Attribute Initialization Input/Output Description ORGANIZE LATER | 11 |
| 5.7 | Rules NEEDS UPDATING | 11 |
| 5.7.1 | Rules Object Attribute Initialization Input/Output Description | 11 |
| 5.8 | Post-Processing (PP) NEEDS UPDATING | 12 |
| 5.9 | Zodiacal Light (ZL) NEEDS UPDATING | 12 |
| 5.9.1 | Zodiacal Light Object Attribute Initialization Input/Output Description ORGANIZE LATER | 12 |

| | |
|--|-----------|
| 6 Simulation Modules | 12 |
| 6.1 Target List (TL) NEEDS UPDATING | 12 |
| 6.1.1 Target List Object Attribute Initialization Input/Output Description | 13 |
| 6.2 Simulated Universe (SU) NEEDS UPDATING | 13 |
| 6.3 Survey Simulation (SS) NEEDS UPDATING | 13 |
| 6.4 Survey Ensemble (SE) NEEDS UPDATING | 14 |

Nomenclature

| | |
|---------|---|
| EXOSIMS | Exoplanet Open-Source Imaging Mission Simulator |
| HE | Heliocentric Equatorial |
| ICD | Interface Control Document |
| MJD | Modified Julian Day |
| OD | Observatory Definition |
| OSD | Optical System Description |
| PP | Post-Processing |
| PPD | Planet Population Description |
| PPM | Planet Physical Model |
| SC | Star Catalog |
| SE | Survey Ensemble |
| SS | Survey Simulation |
| SU | Simulated Universe |
| TL | Target List |
| ZL | Zodiacal Light |

1 Introduction

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

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

1.1 Purpose and Scope

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

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

2 Overview

The terminology used to describe the software implementation is loosely based on the object-oriented Python framework upon 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 of the prototype, or an instance of this object 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 with the larger framework, and thus allows flexibility and adaptability.

The overall framework of EXOSIMS is depicted in Fig. 1 where the component stand-alone software modules are classified as input modules and simulation modules. The input modules include the Optical System Description (OSD), Star Catalog (SC), Planet Population Description (PPD), Observatory Definition (OD), Planet Physical Model (PPM), Time, Rules, and Post-Processing (PP) modules. These modules contain specific mission design parameters. The simulation modules include Target List (TL), Simulated Universe (SU), Survey Simulation (SS), and Survey Ensemble (SE) modules. The simulation modules take information contained in the input modules and perform mission simulation tasks. Any module may perform any number or kind of calculations using any or all of the input parameters provided. They are only constrained by their input and output specification contained in this document.

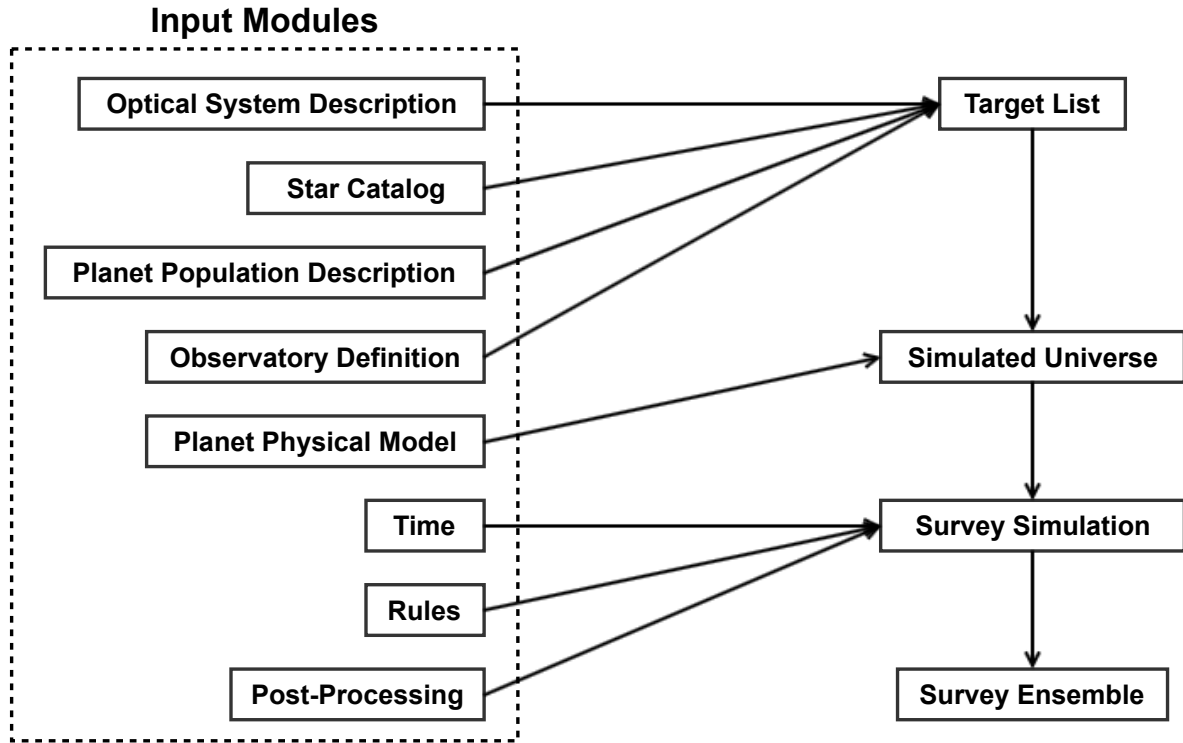


Fig. 1. Flowchart of mission simulation. Each box represents a component software module which interacts with other modules as indicated by the arrows. The simulation modules (those that are not classified as input modules) pass all input modules along with their own output. Thus, the Survey Ensemble module has access to all of the input modules and all of the upstream simulation modules.

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)

Common Time

Modified Julian Day ($\text{MJD} \triangleq \text{JD} - 2400000.5$)

Relative Working Directory

EXOSIMS/

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

importlib
numpy
    numpy.linalg

os
    os.path

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

4 Backbone

All simulation execution will be performed by the backbone. This set of functions will have very limited built-in functionality, and will primarily be tasked with parsing the input specification described below, and then calling the specified instances of each of the framework modules, detailed in §5.

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. 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.

The backbone will contain a standalone 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.

The backbone will contain an interactive function to help users generate specification files via a series of questions.

4.1 Specification Format

```
{
  "missionLifetime": 6,
  "missionDutyCycle": 0.24,
  "starlightSupressionSystems": [
    {
      "type": "SDO",
      "occulterDiameter": 50,
      "occulterDistance": 50000,
      "PSFfile": "/data/sd01_psf.fits",
      "throughputFile": "/data/sd01_thru.fits"
    },
    {
      "type": "coronagraph",
      "IWA": 3,
      "PSFfile": "/data/coron1_psf.fits",
      "throughputFile": "/data/coron1_thru.fits"
    }
  ],
  "OSDmod": "hybridOSD1"
}
```

5 Input Modules

The input modules include Optical System Description (OSD), Star Catalog (SC), Planet Population Description (PPD), Observatory Definition (OD), Planet Physical Model (PPM), Time, Rules, and Post-Processing (PP). These modules encode and/or generate all of the information necessary to perform mission simulations. The specific mission design determines the functionality of each input module, while inputs and outputs of these modules remain the same (in terms of data type and variable representations). This section defines the functionality, major tasks, input, output, and interface of each of these modules.

5.1 Optical System Description (OSD) NEEDS UPDATING

The OSD module contains all of the necessary information to describe the effects of the telescope and starlight suppression system on the target star and planet wavefronts. This requires encoding the design of both the telescope optics and the specific starlight suppression system, whether it be an internal coronagraph or an external occulter. The encoding can be achieved by specifying Point Spread Functions (PSF) for on- and off-axis sources, along with (potentially angular separation-dependent) contrast and throughput definitions. At the opposite level of complexity, the encoded portions of this module may be a description of all of the optical elements between the telescope aperture and the imaging detector, along with a method of propagating an input wavefront to the final image plane. Intermediate implementations can include partial propagations, or collections of static PSFs representing the contributions of various system elements. The encoding of the optical train will allow for the extraction of specific bulk parameters including the instrument inner working angle (IWA), outer working angle (OWA), and mean and max contrast and throughput.

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

5.1.1 Optical System Object Attribute Initialization Input/Output Description

Inputs

User specification

Information from simulation specification JSON file organized into a Python dictionary. If the below key: value pairs are missing from the dictionary, the Optical System object attributes will be assigned the default values listed.

specs["lam"]

Central or detection wavelength in *nm*. Default value is 500.

specs["shapeFac"]

Shape factor so that $shapeFac \times diameter^2 = Area$. Default value is $\frac{\pi}{4}$.

specs["pupilArea"]

Entrance pupil area in m^2 . Default value is 4π .

specs["SNchar"]

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

specs["haveOcculter"]

Boolean signifying if system includes an occulter. Default value is False.

specs["pixelArea"]

Pixel area in m^2 . Default value is $1e-10$.

specs["focalLength"]

Focal length in *m*. Default value is 240.

specs["IWA"]

Inner Working Angle in *arcseconds*. Default value is 0.075.

specs["OWA"]

Outer Working Angle in *arcseconds*. Default value is inf.

specs["dMagLim"]

Limiting Δmag (difference in magnitude between star and planet). Default value is 26.

specs["throughput"]

Optical system throughput at IWA. Default value is 0.5.

specs["contrast"]

Optical system suppression level at IWA. Default value is $1e-10$.

specs["dr"]

Detector dark current rate per pixel. Default value is 0.001.

specs["sigma_r"]

Detector read noise. Default value is 3.

specs["t_exp"]
 Exposure time per read in s . Default value is 1000.
 specs["QE"]
 Detector quantum efficiency. Default value is 0.5.
 specs["eta2"]
 Post coronagraph attenuation. Default value is 0.57.
 specs["deltaLambda"]
 Detection bandwidth $\Delta\lambda$ in nm . Default value is 100.

Outputs

OpticalSys.lam

Central or detection wavelength (astropy unit object initially defined in nm)

OpticalSys.shapeFac

Shape factor so that $shapeFac \times diameter^2 = Area$

OpticalSys.pupilArea

Entrance pupil area (astropy unit object initially defined in m^2)

OpticalSys.SNchar

Signal to Noise Ratio for characterization

OpticalSys.haveOcculter

Boolean signifying if system includes an occulter

OpticalSys.pixelArea

Pixel area (astropy unit object initially defined in m^2)

OpticalSys.focalLength

Focal length (astropy unit object initially defined in m)

OpticalSys.IWA

Inner Working Angle (astropy unit object initially defined in $arcseconds$)

OpticalSys.OWA

Outer Working Angle (astropy unit object initially defined in as)

OpticalSys.dMagLim

Limiting Amag (difference in magnitude between star and planet)

OpticalSys.throughput

Optical system throughput at IWA

OpticalSys.contrast

Designed optical system suppression level at IWA

OpticalSys.dr

Detector dark-current rate per pixel (astropy unit object initially defined in $\frac{1}{s}$)

OpticalSys.sigma_r

Detector read noise

OpticalSys.t_exp

Exposure time per read (astropy unit object initially defined in s)

OpticalSys.QE

Detector quantum efficiency

OpticalSys.eta2

Post coronagraph attenuation

OpticalSys.deltaLambda

Detection bandwidth $\Delta\lambda$ (astropy unit object initially defined in nm)

OpticalSys.dAlpha

Pixel size in square arcseconds derived from OpticalSys.lam, OpticalSys.shapeFac, and OpticalSys.pupilArea (astropy unit object initially defined in as^2)

OpticalSys.Psi

Nondimensional instrument specific value derived from PSF

OpticalSys.Xi

Nondimensional instrument specific value derived from PSF

OpticalSys.P1

Nondimensional instrument specific value derived from PSF

OpticalSys.alphaBar

Nondimensional instrument specific value derived from OpticalSys.pixelArea, OpticalSys.pupilArea, Optical-

Sys.shapeFac, OpticalSys.focalLength, and OpticalSys.lam.

OpticalSys.Ta

Airy throughput derived from OpticalSys.throughput, OpticalSys.shapeFac, OpticalSys.P1, and OpticalSys.alphaBar

5.2 Star Catalog (SC)

The SC 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 will 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 TL module as it requires interactions with the PPD (to determine the population of interest), OSD (to define the capabilities of the instrument), and OD (to determine if the view of the target is unobstructed) modules.

5.2.1 Star Catalog Object Attribute Initialization Input/Output Description

Inputs

star catalog information

Information from an external star catalog

Outputs

StarCatalog.Name

Numpy array of star names

StarCatalog.Type

Numpy array of star types

StarCatalog.Spec

Numpy array of spectral types

StarCatalog.parx

Numpy array of parallax in milliarcseconds

StarCatalog.Umag

Numpy array of U magnitude

StarCatalog.Bmag

Numpy array of B magnitude

StarCatalog.Vmag

Numpy array of V magnitude

StarCatalog.Rmag

Numpy array of R magnitude

StarCatalog.Imag

Numpy array of I magnitude

StarCatalog.Jmag

Numpy array of J magnitude

StarCatalog.Hmag

Numpy array of H magnitude

StarCatalog.Kmag

Numpy array of K magnitude

StarCatalog.dist

Numpy array of distance in parsecs

StarCatalog.BV

Numpy array of B-V Johnson magnitude

StarCatalog.MV

Numpy array of absolute V magnitude

StarCatalog.BC

Numpy array of bolometric correction

StarCatalog.L

Numpy array of stellar luminosity in Solar luminosities

StarCatalog.coords

Astropy [SkyCoord](#) object containing list of star positions (e.g., right ascension and declination)

StarCatalog.pmra

Numpy array of proper motion in right ascension in milliarcseconds/year

StarCatalog.pmdec

Numpy array of proper motion in declination in milliarcseconds/year

StarCatalog.rv

Numpy array of radial velocity in kilometers/second

StarCatalog.Binary_Cut

Boolean Numpy array where True is companion star closer than 10 arcseconds

5.3 Planet Population Description (PPD) NEEDS UPDATING

The PPD module encodes the density functions of all required planetary parameters, both physical and orbital. These include semi-major axis, eccentricity, orbital orientation, and planetary radius and mass. Certain parameter models may be empirically derived while others may come from analyses of observational surveys. This module also encodes the limits on all parameters to be used for sampling the distributions and determining derived cutoff values such as the maximum target distance for a given instrument's IWA.

The PPD 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.

5.3.1 Planet Population Object Attribute Initialization Input/Output Description**Inputs****User specification**

Information from simulation specification JSON file organized into a Python dictionary. If the below key: value pairs are missing from the dictionary, the Planet Population object attributes will be assigned the default values listed.

specs["a_min"]

Minimum semi-major axis in *AU*. Default value is 0.1.

specs["a_max"]

Maximum semi-major axis in *AU*. Default value is 1.0.

specs["e_min"]

Minimum eccentricity. Default value is 0.

specs["e_max"]

Maximum eccentricity. Default value is 0.8.

specs["p_min"]

Minimum planetary albedo. Default value is 0.1.

specs["p_max"]

Maximum planetary albedo. Default value is 0.5.

specs["R_min"]

Minimum planetary radius in *km*. Default value is `astropy.constants.R_earth` in *km*.

specs["R_max"]

Maximum planetary radius in *km*. Default value is $15 \times \text{astropy.constants.R_earth}$ in *km*.

specs["scaleOrbits"]

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

Outputs**PlanetPopulation.arange**

Semi-major axis range defined as `numpy.array([a_min, a_max])` (astropy unit object with units of *AU*)

PlanetPopulation.erange

Eccentricity range defined as `numpy.array([e_min, e_max])`

PlanetPopulation.prange

Planetary albedo range defined as `numpy.array([p_min, p_max])`

PlanetPopulation.Rrange

Planetary radius range defined as `numpy.array([R_min, R_max])` (astropy unit object or astropy constant objects, e.g., `astropy.constant.R_earth` with units of *km*)

PlanetPopulation.scaleOrbits

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

PlanetPopulation.rrange

Planetary orbital radius range defined as `numpy.array([r_min, r_max])` (astropy unit object with units of *km*)

5.4 Observatory Definition (OD)

The OD module contains all of the information specific to the space-based observatory not included in the OSD module. The module has two main tasks: orbit and keepout definition, which are implemented as functions within the module. The inputs and outputs for these functions are represented schematically in Fig. 2.

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 OD module's orbit function 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 definition task and TL module to determine which of the stars from the SC module may be targeted for observation at the current mission time.

The keepout definition 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 definition function takes the current mission time and SC module output as inputs and outputs a list of the target stars which are observable at the current time. It constructs position vectors of the target stars and bright objects which may interfere with observations with respect to the observatory. These position vectors are used to determine if bright objects are in the field of view for each of the potential stars under exoplanet finding observation. If there are no bright objects obstructing the view of the target star, it becomes a candidate for observation in the SS module.

In addition to these functions, 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 cryogenics 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 baseline geosynchronous 28.5° inclined orbit for WFIRST-AFTA with an L2 halo orbit proposed for other exoplanet imaging mission concepts.

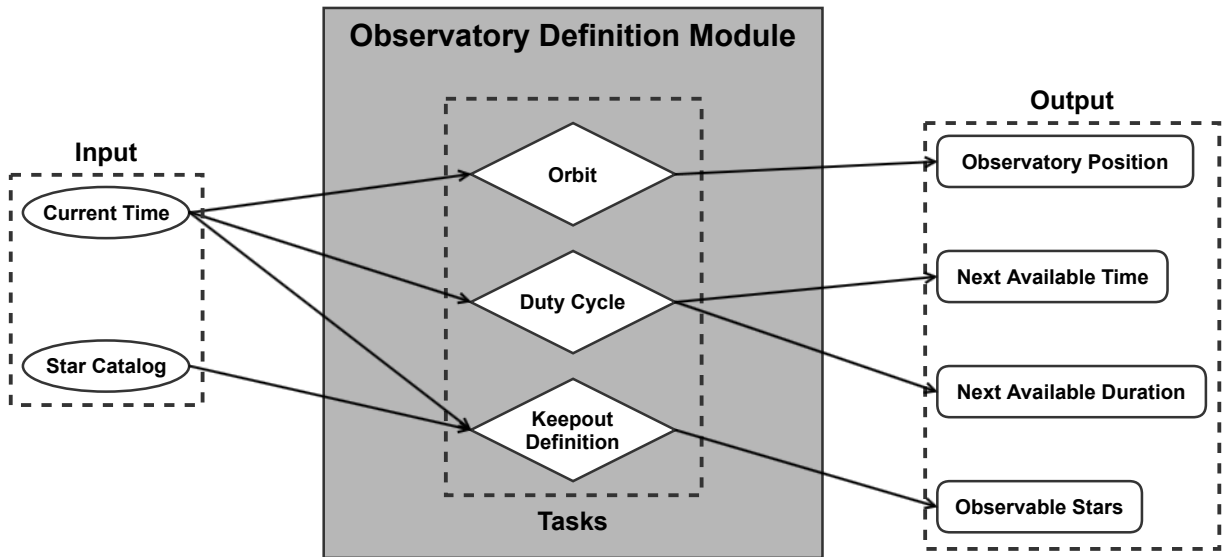


Fig. 2. Depiction of Observatory Definition module including inputs, tasks, and outputs.

5.4.1 Observatory Object Attribute Initialization Input/Output Description

Inputs

User specification

Information from simulation specification JSON file organized into a Python dictionary. If the below key: value pairs are missing from the dictionary, the Observatory object attributes will be assigned the default values listed.

specs["settling_time"]

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

specs["thrust"]

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

specs["slewIsp"]

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

Outputs

Observatory.settling_time

Amount of time needed for observatory to settle after a repointing (astropy unit object initially set in days)

Observatory.thrust

Occulter slew thrust (astropy unit object initially set in *mN*)

Observatory.slewIsp

Occulter slew specific impulse (astropy unit object initially set in *s*)

Observatory.flowRate

Slew flow rate derived from Observatory.thrust and Observatory.slewIsp (astropy unit object initially set in *kg/day*)

Observatory.r_sc

Observatory orbit position in HE reference frame. Initialized to numpy array as `numpy.array([0., 0., 0.])` and associated with astropy unit object in *km*. This attribute is updated to the orbital position of the observatory at the current mission time through the orbit task (see 5.4.2).

Observatory.kogood

List of Boolean values where True is a target unobstructed and observable in the keepout zone. Initialized to an empty list. This attribute is updated to the current mission time through the keepout task (see 5.4.3).

5.4.2 Orbit Task Input/Output Description

Inputs

time.currentTimeAbs

Current absolute mission time (astropy Time object) from Time module see 5.6.1 for definition

Outputs

Observatory.r_sc

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

5.4.3 Keepout Task Input/Output Description

Inputs

time.currentTimeAbs

Current Absolute mission time (astropy Time object) from Time module see 5.6.1 for definition

StarCatalog

Output of SC module. See 5.2 for definitions

Outputs

Observatory.kogood

List of Boolean values for each target at current mission time where True is when a target is unobstructed in the keepout zone and False is when a target cannot be observed due to obstructions in the keepout zone

5.5 Planet Physical Model (PPM) NEEDS UPDATING

The PPM 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 PPD 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 albedo, spectra and phase curve models.

5.6 Time

The Time 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 days). The Time module also provides functionality for converting between this time measure and standard measures such as Julian Day Number and UTC time.

5.6.1 Time Object Attribute Initialization Input/Output Description ORGANIZE LATER

Inputs

User specification

Information from simulation specification JSON file organized into a Python dictionary. If the below key: value pairs are missing from the dictionary, the Time object attributes will be assigned the default values listed.

specs["missionStart"]

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

specs["missionLife"]

Total length of mission in *days*. Default value is 2191.5.

Outputs

TimeKeeping.missionStart

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

TimeKeeping.missionLife

Mission lifetime (astropy TimeDelta object initially defined in *JD* increments, where $JD = MJD = 86400s$)

TimeKeeping.currenttimeNorm

Current mission time normalized so that start date is 0 (astropy TimeDelta object initially defined in *JD* increments as above)

TimeKeeping.currenttimeAbs

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

TimeKeeping.missionFinish

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

5.7 Rules NEEDS UPDATING

The Rules module contains additional constraints placed on the mission design not contained in other modules. These constraints are passed into the SS module to control the simulation. For example, a constraint in the Rules module could include prioritization of revisits to stars with detected exoplanets for characterization when possible. This rule would force the SS module to simulate observations for target stars with detected exoplanets when the OD module determines those stars are observable.

The Rules module also encodes the calculation of integration time for an observation. This can be based on achieving a pre-determined signal to noise (SNR) metric (with various possible definitions), or via a probabilistic description. This requires also defining a model for the background contribution due to all astronomical sources and especially due to zodiacal and exozodiacal light.

The duty cycle determines when during the mission timeline the observatory is allowed to perform planet-finding operations. The duty cycle function takes the current mission time as input and outputs the next available time when exoplanet observations may begin or resume, along with the duration of the observational period. The outputs of this task are used in the SS module to determine when and how long exoplanet finding and characterization observations occur.

5.7.1 Rules Object Attribute Initialization Input/Output Description

Inputs

User specification

Information from simulation specification JSON file organized into a Python dictionary. If the below key: value pairs are missing from the dictionary, the Rules object attributes will be assigned the default values listed.

specs["FAP"]

Detection false alarm probability. Default value is 0.01/1000.

specs["MDP"]

Missed detection probability. Default value is 0.001.

specs["dutyCycle"]

Observation duty cycle. Default value is 1.

specs["intCutoff"]

Maximum allowed integration time in *days*. Default value is 50.

Outputs

Rules.FAP

Detection false alarm probability

Rules.MDP

Missed detection probability

Rules.dutyCycle

Observation duty cycle

Rules.intCutoff

Maximum allowed integration time (astropy unit object initially set to *days*)

Rules.K

Threshold value determined by Rules.FAP

Rules.gamma

Threshold value determined by Rules.MDP

5.8 Post-Processing (PP) NEEDS UPDATING

The PP 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 PP 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.

5.9 Zodiacal Light (ZL) NEEDS UPDATING

5.9.1 Zodiacal Light Object Attribute Initialization Input/Output Description ORGANIZE LATER

Input

User specification

Information from simulation specification JSON file organized into a Python dictionary. If the below key: value pairs are missing from the dictionary, the Zodiacal Light object attributes will be assigned the default values listed.

specs["exozodi"]

Exo-zodi level in zodi

specs["exozodiVar"]

Exo-zodi variation (variance of log-normal distribution)

Output

ZodiacalLight.exozodi

Exo-zodi level in zodi

ZodiacalLight.exozodiVar

Exo-zodi variation (variance of log-normal distribution)

6 Simulation Modules

The simulation modules include Target List (TL), Simulated Universe (SU), Survey Simulation (SS) and Survey Ensemble (SE). These modules perform tasks which require inputs from one or more input modules as well as calling function implementations in other simulation modules.

6.1 Target List (TL) NEEDS UPDATING

The TL module takes in information from the OSD, SC, PPD, and OD input modules and generates 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 SC module.

6.1.1 Target List Object Attribute Initialization Input/Output Description

Inputs

User specification

Information from simulation specification JSON file organized into a Python dictionary. If the below key: value pairs are missing from the dictionary, the Target List object attributes will be assigned the default values listed.

specs["catalogname"]

String containing name of desired SC module

specs["optsysname"]

String containing name of desired OSD module

specs["popname"]

String containing name of desired PPD module

specs["rulesname"]

String containing name of desired Rules module

specs["zodiname"]

String containing name of desired ZL module

StarCatalog

Output of SC module given by above specs["catalogname"] (see 5.2)

OpticalSys

Output of OSD module given by above specs["optsysname"] (see 5.1)

PlanetPopulation

Output of PPD module given by above specs["popname"] (see 5.3)

Rules

Output of Rules module given by above specs["rulesname"] (see 5.7)

ZodiacalLight

Output of ZL module given by above specs["zodiname"] (see 5.9)

Outputs

StarCatalog values

Mission specific filtered star catalog values from SC module (see 5.2)

TargetList.intTime

Numpy array of maximum integration time for each target star (astropy unit object initially defined in s)

TargetList.MsEst

Approximate stellar mass in M_{sun}

TargetList.MsTrue

Stellar mass with an error component included in M_{sun}

6.2 Simulated Universe (SU) NEEDS UPDATING

The SU module takes as input the outputs of the TL simulation module to create a synthetic universe composed of the systems in the target list. For each target, a planetary system is generated based on the statistics encoded in the PPD 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. This universe is encoded as a list where each entry corresponds to one element of the target list, and where the list entries are arrays of planet physical parameters. In cases of empty planetary systems, the corresponding list entry contains a null array.

The SU module also takes as input the PPM module instance, so that it can return the specific spectra due to every simulated planet at an arbitrary observation time throughout the mission simulation.

6.3 Survey Simulation (SS) NEEDS UPDATING

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

6.4 Survey Ensemble (SE) NEEDS UPDATING

The SE 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.