Your Project Documentation

Release 0.1

Your Name

Sep 13, 2023

CONTENTS:

CHAPTER

ONE

FSPSIM PACKAGE

1.1 Subpackages

1.1.1 fspsim.utils package

1.1.1.1 Submodules

1.1.1.2 fspsim.utils.Conversions module

fspsim.utils.Conversions.**TLE_time**(*TLE*)

Returns the Epoch Year of the TLE as a Julian Date.

Parameters
TLE (*string*) – Two Line Element

Returns
Julian Date

Return type
datetime.datetime

fspsim.utils.Conversions.**UTC_step**(*date_list*, *steps*, *h*)

Make an array of datetime strings in UTC format.

Parameters

- **date_list** (*list[int]*) – List containing year, month, day, hour, minute, and second.
- **steps** (*int*) – Number of steps in the propagation.
- **h** (*float*) – Step size the propagation uses in seconds.

Returns
List of datetime strings in UTC format.

Return type
list[datetime.datetime]

fspsim.utils.Conversions.**calculate_eccentricity**(*position: List[float]*, *velocity: List[float]*, *mu: float*)

Calculate the eccentricity of an orbit given position, velocity, and gravitational parameter.

The function computes the eccentricity using the specific mechanical energy and specific angular momentum of the object in orbit.

Parameters

- **position** (*List[float]*) – Position vector of the object in orbit. Expected to be a 3-element list representing [x, y, z].
- **velocity** (*List[float]*) – Velocity vector of the object in orbit. Expected to be a 3-element list representing [vx, vy, vz].
- **mu** (*float*) – Gravitational parameter, product of the gravitational constant (G) and the mass (M) of the primary body.

Returns
Eccentricity of the orbit.

Return type
float

fspsim.utils.Conversions.**calculate_energy_from_semi_major_axis**(*a*, *m*)

Use the vis-viva law to calculate the total mechanical energy of an object in orbit around the Earth.

Parameters

- **a** (*float*) – Semi-major axis in meters.
- **m** (*float*) – Mass of the object in kg.

Returns
Total mechanical energy in joules.

Return type
float

fspsim.utils.Conversions.**car2kep**(*x*, *y*, *z*, *vx*, *vy*, *vz*, *mean_motion=False*, *arg_l=False*)

Convert Cartesian coordinates to Keplerian elements in radians.

Parameters

- **x** (*float*) – Position coordinate in km.
- **y** (*float*) – Position coordinate in km.
- **z** (*float*) – Position coordinate in km.
- **vx** (*float*) – Velocity component in km/s.
- **vy** (*float*) – Velocity component in km/s.
- **vz** (*float*) – Velocity component in km/s.
- **mean_motion** (*bool*) – If True, return mean motion. Default is False.
- **arg_l** (*bool*) – If True, return the argument of latitude. Default is False.

Returns
Tuple containing semi-major axis, eccentricity, inclination, RAAN, argument of perigee, and true anomaly. Optionally returns mean motion and argument of latitude.

Return type
tuple

fspsim.utils.Conversions.**earth_moon_vec**(*jd*, *unit=True*)

Calculate the Earth-Moon vector in ECI coordinates. Makes use of the JPL ephemerides to calculate the Earth-Moon vector.

Parameters

- **jd** (*float*) – Julian Date for which we want the Earth-Moon vector.

- **unit** (*bool*) – If True, returns the unit vector. If False, returns the vector itself. Default is True.

Returns
Earth-Moon vector in ECI coordinates.

Return type
np.array

fspsim.utils.Conversions.**earth_sun_vec**(*jd*, *unit=True*)

Calculate the Earth-Sun vector in ECI coordinates. Makes use of the JPL ephemerides to calculate the Earth-Sun vector.

Parameters

- **jd** (*float*) – Julian Date for which we want the Earth-Sun vector.
- **unit** (*bool*) – If True, returns the unit vector. If False, returns the vector itself. Default is True.

Returns
Earth-Sun vector in ECI coordinates.

Return type
np.array

fspsim.utils.Conversions.**eccentric_to_mean_anomaly**(*eccentric_anomaly*, *eccentricity*)

Convert eccentric anomaly to mean anomaly.

Parameters

- **eccentric_anomaly** (*float*) – Eccentric anomaly in radians.
- **eccentricity** (*float*) – Orbital eccentricity.

Returns
Mean anomaly in radians.

Return type
float

fspsim.utils.Conversions.**ecef2eci_astropy**(*ecef_pos: ndarray*, *ecef_vel: ndarray*, *mjd: float*) → Tuple[ndarray, ndarray]

Convert ECEF (Earth-Centered, Earth-Fixed) coordinates to ECI (Earth-Centered Inertial) coordinates using Astropy.

Parameters

- **ecef_pos** (*np.ndarray*) – ECEF position vectors.
- **ecef_vel** (*np.ndarray*) – ECEF velocity vectors.
- **mjd** (*float*) – Modified Julian Date.

Returns
Tuple containing ECI position and ECI velocity vectors.

Return type
tuple of np.ndarray

fspsim.utils.Conversions.**generate_cospar_id**(*launch_year*, *launch_number*, *launch_piece*)

Generates a placeholder (fake) COSPAR ID for a spacecraft based on the launch year, launch number, and launch piece.

Parameters

- **launch_year** (*int*) – The launch year of the spacecraft.
- **launch_number** (*int*) – The launch number of the spacecraft.
- **launch_piece** (*str*) – The piece of the launch.

Returns
The generated COSPAR ID.

Return type
str

fspsim.utils.Conversions.**get_day_of_year_and_fractional_day**(*epoch*)

Compute the day of the year and fractional day for a given date.

Parameters
epoch (*datetime.datetime*) – Datetime object representing the epoch.

Returns
Day of the year and fractional day.

Return type
float

fspsim.utils.Conversions.**jd_to_utc**(*jd*)

Converts Julian Date to UTC time tag (datetime object) using Astropy.

Parameters
jd (*float*) – Julian Date

Returns
UTC time tag (datetime object)

Return type
datetime.datetime

fspsim.utils.Conversions.**kep2car**(*a*, *e*, *i*, *w*, *W*, *V*, *epoch*)

Convert Keplerian elements to Cartesian coordinates.

Parameters

- **a** – Semi-major axis (km).
- **e** – Eccentricity.
- **i** – Inclination (rad).
- **w** – Argument of perigee (rad).
- **W** – Right ascension of ascending node (RAAN) (rad).
- **V** – True anomaly (rad).
- **epoch** (*Time*) – Time at which the elements are given.

Returns
Tuple containing x, y, z coordinates and vx, vy, vz velocities.

Return type
tuple of floats

fspsim.utils.Conversions.**orbit_classify**(*altitude*)

Classify the orbit based on the altitude.

Parameters
altitude (*float*) – Altitude of the orbit in km.

Returns
Orbit classification as a string.

Return type
str

fspsim.utils.Conversions.**orbital_period**(*semi_major_axis*)

Calculate the Orbital Period of a satellite based on the semi-major-axis

Parameters
semi_major_axis (*int*) – Semi Major Axis

Returns
Orbital Period in Minutes

Return type
int

fspsim.utils.Conversions.**probe_sun_vec**(*r*, *jd*, *unit=False*)

Calculate the probe-Sun vector in ECI coordinates.

Parameters

- **r** (*array*) – Probe position in ECI coordinates.
- **jd** (*float*) – Julian Date for which we want the probe-Sun vector.
- **unit** (*bool*) – If True, returns the unit vector. If False, returns the vector itself. Default is False.

Returns
Probe-Sun vector in ECI coordinates.

Return type
np.array

fspsim.utils.Conversions.**tle_checksum**(*line*)

Perform the checksum for one TLE line

Parameters
line (*str*) – TLE line

Returns
Checksum

Return type
int

fspsim.utils.Conversions.**tle_convert**(*tle_dict*, *display=False*)

Convert a TLE dictionary into the corresponding Keplerian elements.

Parameters

- **tle_dict** (*dict*) – Dictionary of TLE data as provided by the tle_parse function.
- **display** (*bool*) – If True, print out the Keplerian elements. Default is False.

Returns
Dictionary containing Keplerian elements.

Return type
dict

fspsim.utils.Conversions.**tle_exponent_format**(*value*)

Convert a value to the TLE-specific exponent format.

Parameters
value (*str*) – TLE Value

Returns
Formatted value

Return type
str

fspsim.utils.Conversions.**tle_parse**(*tle_2le*)

Parse a 2LE string (e.g. as provided by Celestrak) and return all the data in a dictionary.

Parameters
tle_2le (*str*) – 2LE string to be parsed.

Returns
Dictionary of all the data contained in the TLE string.

Return type
dict

fspsim.utils.Conversions.**true_to_eccentric_anomaly**(*true_anomaly*, *eccentricity*)

Convert true anomaly to eccentric anomaly.

Parameters

- **true_anomaly** (*float*) – True anomaly in radians.
- **eccentricity** (*float*) – Orbital eccentricity.

Returns
Eccentric anomaly in radians.

Return type
float

fspsim.utils.Conversions.**true_to_mean_anomaly**(*true_anomaly*, *eccentricity*)

Convert true anomaly to mean anomaly.

Parameters

- **true_anomaly** (*float*) – True anomaly in radians.
- **eccentricity** (*float*) – Orbital eccentricity.

Returns
Mean anomaly in radians.

Return type
float

fspsim.utils.Conversions.**utc_to_jd**(*time_stamps*)

Convert UTC datetime or string representations to Julian Date (JD).

This function takes in either a single datetime, string representation of a datetime, or a list of them. It then converts each of these into its corresponding Julian Date (JD) value. If a list is provided, it returns a list of JD values. If a single datetime or string is provided, it returns a single JD value.

Parameters

time_stamps (*datetime.datetime, str or list of datetime.datetime/ str*) – The datetime object(s) or string representation(s) of dates/times to be converted to Julian Date. Strings should be in the format '%Y-%m-%d' or '%Y-%m-%d %H:%M:%S'.

Returns

The corresponding Julian Date (JD) value(s) for the provided datetime(s) or string representation(s). Returns a single float if the input is a single datetime or string, and a list of floats if the input is a list.

Return type

float or list of float

fspsim.utils.Conversions.**v_rel**(*state*)

Calculate the speed of the satellite relative to the Earth's atmosphere. This assumes that the atmosphere is stationary and co-rotating with the surface Earth.

Parameters

state (*array-like*) – State vector of the satellite in ECI coordinates.

Returns

Velocity of the satellite with respect to the Earth's atmosphere in Km/s.

Return type

np.array

fspsim.utils.Conversions.**write_tle**(*catalog_number*, *classification*, *launch_year*, *launch_number*, *launch_piece*, *epoch_year*, *epoch_day*, *first_derivative*, *second_derivative*, *drag_term*, *ephemeris_type*, *element_set_number*, *inclination*, *raan*, *eccentricity*, *arg_perigee*, *mean_anomaly*, *mean_motion*, *revolution_number*)

This function builds a TLE string in the correct format when given the necessary parameters.

Parameters

- **catalog_number** (*int*) – Catalog number of the satellite.
- **classification** (*str*) – Classification type (usually 'U' for unclassified).
- **launch_year** (*int*) – Year of launch.
- **launch_number** (*int*) – Launch number of the year.
- **launch_piece** (*str*) – Piece of the launch.
- **epoch_year** (*int*) – Year of the epoch.
- **epoch_day** (*float*) – Day of year the epoch.
- **first_derivative** (*float*) – First time derivative of the mean motion (ballistic coefficient).
- **second_derivative** (*float*) – Second time derivative of the mean motion (delta-dot).
- **drag_term** (*float*) – B* drag term.
- **ephemeris_type** (*int*) – Ephemeris type.
- **element_set_number** (*int*) – Element set number.

- **inclination** (*float*) – Inclination (rad).
- **raan** (*float*) – Right ascension of the ascending node (RAAN) (rad).
- **eccentricity** (*float*) – Eccentricity.
- **arg_perigee** (*float*) – Argument of perigee (rad).
- **mean_anomaly** (*float*) – Mean anomaly (rad).
- **mean_motion** (*float*) – Mean motion (rad/s).
- **revolution_number** (*int*) – Revolution number at epoch.

Returns
A two-line element set (TLE) string.

Return type
str

1.1.1.3 fspsim.utils.Formatting module

fspsim.utils.Formatting.**calculate_form_factor**(*form_factor_str*)

Convert the form factor specified in the predictions csv to a characteristic length and area. For CubeSats, the area can be specified by writing the number of "U"s in the form factor string. e.g. "3U" or "6u" (the case does not matter) For other satellites the form factor must be three numbers (floats or ints), separated by stars. e.g. "1.2*2.3*4.5" or "1*2*3"

Parameters
form_factor_str (*string*) – string describing the form factor of satellties

Raises
ValueError – Form factor must be a string

Returns
characteristic length, characteristic area

Return type
tuple

fspsim.utils.Formatting.**future_constellations_csv_handler**(*file_path*)

Checks that the user supplied Future Constellation CSV is in the correct format for the simulation

Parameters
file_path (*str*) – File Path of the CSV

Returns
Dictionary of the constellations in a format the fspsim can read.

Return type
dict

1.1.1.4 fspsim.utils.LaunchModel module

fspsim.utils.LaunchModel.**Prediction2SpaceObjects**(*satellite_predictions_csv*, *simsettings*)

Generates a list SpaceObjects for each of the sub-constellation in the prediction data CSV file. Each object gets assigned a launch and decay date based on the launch model parameters and the parameters specified in the prediction data CSV file.

Parameters

- **satellite_predictions_csv** (*str*) – CSV file with the satellite predictions data.
- **simsettings** (*dict*) – Loaded JSON file with the simulation settings (contains launch model parameters).

Returns
A list of space objects generated from the prediction data.

Return type
list[*SpaceObject*]

fspsim.utils.LaunchModel.**create_subconstellation_Space_Objects**(*N*, *i*, *h*, *_soname*, *_application*, *_owner*, *launch_schedule*, *_mass*, *_area*, *_length*, *_maneuverable*, *_propulsion*)

Generate a list of SpaceObjects that are in a Walker-Delta constellation. This maximizes geometric coverage of the Earth given the number of satellites and the inclination and altitude of the orbit. Based on the parameters passed in the launch file.

Parameters

- **N** (*int*) – Number of satellites in the constellation.
- **i** (*float*) – Inclination of the satellite orbit.
- **h** (*float*) – Altitude of the satellite orbit.
- **_soname** (*str*) – Name of the sub-constellation.
- **_application** (*str*) – Application for which the satellite is used.
- **_owner** (*str*) – Owner or operator of the satellite.
- **launch_schedule** (*list[str]*) – Schedule of satellite launches.
- **_mass** (*float*) – Mass of the satellite.
- **_area** (*float*) – Characteristic area of the satellite.
- **_length** (*float*) – Characteristic length of the satellite.
- **_maneuverable** (*str*) – Indicates if the satellite is maneuverable.
- **_propulsion** (*str*) – Type of propulsion used in the satellite.

Raises
ValueError – If N is less than 1.

Returns
A list of space objects for the sub-constellation.

Return type
list[*SpaceObject*]

fspsim.utils.LaunchModel.**global_launch_schedule**(*sub_constellation_metadata_dicts*, *monthly_ton_capacity*, *launches_start_date*, *rocket='Falcon 9'*)

Determines the launch dates for various sub-constellations based on a greedy algorithm, limited by global montly launch capacity (tons/month), the date to start scheduling launches, and the type of rocket used.

Parameters

- **sub_constellation_metadata_dicts** (*list[dict]*) – List of metadata for each sub-constellation.
- **monthly_ton_capacity** (*float*) – The maximum weight capacity available for launching in tons per month.
- **launches_start_date** (*str*) – The initial date to begin scheduling launches.
- **rocket** (*str, optional*) – The type of rocket used for launches, defaults to "Falcon 9".

Raises

ValueError – If the provided rocket is not in the list of LEO launchers.

Returns

A dictionary of launch dates for each sub-constellation.

Return type

dict[str, list[str]]

fspsim.utils.LaunchModel.**import_configuration_json**(*filename*)

1.1.1.5 fspsim.utils.Propagators module

fspsim.utils.Propagators.**compute_eccentric_anomaly**(*Mo*, *e*, *n*, *step*, *step_size*, *r_tol*)

Computes the eccentric anomaly using Newton's method.

Parameters

- **Mo** (*float*) – Mean anomaly (first "guess") in radians
- **e** (*float*) – Eccentricity of the orbit
- **n** (*float*) – Mean motion of the orbit in radians/second
- **step** (*int*) – Current step number
- **step_size** (*float*) – Time step of propagation in seconds
- **r_tol** (*float*) – Relative tolerance for Newton's method

Returns

E – Approximated value of eccentric anomaly in radians

Return type

float

fspsim.utils.Propagators.**compute_gaussian_vectors**(*W*, *w*, *i*)

Computes the Gaussian vectors P and Q for coordinate transformation.

Parameters

- **W** (*float*) – Right ascension of the ascending node of the orbit in radians
- **w** (*float*) – Argument of perigee of the orbit in radians
- **i** (*float*) – Inclination of the orbit in radians

Returns

- **P** (*numpy.ndarray*) – Gaussian vector P
- **Q** (*numpy.ndarray*) – Gaussian vector Q

fspsim.utils.Propagators.**compute_velocity**(*GM_earth*, *a*, *e*, *E*, *P*, *Q*)

Computes the velocity vector of a satellite in a Keplerian orbit.

Parameters

- **GM_earth** (*float*) – Gravitational constant of the Earth in km^3/s^2
- **a** (*float*) – Semi-major axis of the orbit in km
- **e** (*float*) – Eccentricity of the orbit
- **E** (*float*) – Eccentric anomaly in radians
- **P** (*numpy.ndarray*) – Gaussian vector P
- **Q** (*numpy.ndarray*) – Gaussian vector Q

Returns
Velocity vector of the satellite in km/s

Return type
numpy.ndarray

fspsim.utils.Propagators.**kepler_prop**(*jd_start*, *jd_stop*, *step_size*, *a*, *e*, *i*, *w*, *W*, *V*, *area=None*, *mass=None*, *cd=None*, *drag_decay=False*)

Propagates the orbit of a satellite in a Keplerian orbit, taking drag decay into account. Uses Kepler's equation for propagation and Gaussian vectors for coordinate transformation. Stops the propagation if altitude drops below 200 km. Area, mass, and drag coefficient are required only for drag decay.

Parameters

- **jd_start** (*float*) – start time of propagation in Julian Date format
- **jd_stop** (*float*) – end time of propagation in Julian Date format
- **step_size** (*float*) – time step of propagation in seconds
- **a** (*float*) – semi-major axis of the orbit in km
- **e** (*float*) – eccentricity of the orbit
- **i** (*float*) – inclination of the orbit in degrees
- **w** (*float*) – argument of perigee of the orbit in degrees
- **W** (*float*) – right ascension of the ascending node of the orbit in degrees
- **V** (*float*) – true anomaly of the orbit in degrees
- **area** (*float, optional*) – cross-sectional area of the satellite in m^2. Defaults to None.
- **mass** (*float, optional*) – mass of the satellite in kg. Defaults to None.
- **cd** (*float, optional*) – drag coefficient of the satellite. Defaults to None.
- **drag_decay** (*bool, optional*) – If True, decay the semi-major axis due to drag. Defaults to False.

Returns: list: list of lists containing the propagated ephemeris

fspsim.utils.Propagators.**sgp4_prop_TLE**(*TLE*, *jd_start*, *jd_end*, *dt*)

Given a TLE, a start time, end time, and time step, propagate the TLE and return the time-series of Cartesian coordinates, and accompanying time-stamps (MJD)

Note: Simply a wrapper for the SGP4 routine in the sgp4.api package (Brandon Rhodes)

Parameters

- **TLE** (*string*) – TLE to be propagated
- **jd_start** (*float*) – start time of propagation in Julian Date format
- **jd_end** (*float*) – end time of propagation in Julian Date format
- **dt** (*float*) – time step of propagation in seconds
- **alt_series** (*bool, optional*) – If True, return the altitude series as well as the position series. Defaults to False.

Returns: list: list of lists containing the time-series of Cartesian coordinates, and accompanying time-stamps (JD)

1.1.1.6 fspsim.utils.SpaceCatalogue module

class fspsim.utils.SpaceCatalogue.**SpaceCatalogue**(*settings*, *future_constellations=None*)

Bases: object

A representation of a space catalogue that consolidates satellite data from various sources.

The class aims to process and manage satellite information from multiple sources like JSR and SpaceTrack. It provides functionality to merge active satellite catalogues, repull information from sources, and create a consolidated catalogue of space objects.

sim_object_type

Type of simulation objects. Can be 'active', 'all', or 'debris'.

Type
str

sim_object_catalogue

Source of the simulation object. Can be 'jsr', 'spacetrack', or 'both'.

Type
str

Catalogue

A list of SpaceObjects representing the space objects in the catalogue.

Type
list

repull_catalogues

Flag to determine if the catalogues should be repulled from sources.

Type
bool

Raises

- **Exception** – If an invalid sim_object_type or sim_object_catalogue is specified.
- **TypeError** – If loaded data from file is not an instance of SpaceCatalogue.

Catalogue2SpaceObjects()

Convert the current catalogue into a list of SpaceObjects.

Raises

Exception – If invalid *sim_object_type* is specified.

Returns

List of SpaceObjects.

Return type

list[*SpaceObject*]

CreateCatalogueActive()

Merge the JSR and SpaceTrack active satellite catalogues.

The resulting merged catalogue is stored in an attribute named CurrentCatalogueDF.

Raises

- **IOError** – If file paths are not found or unreadable.
- **ValueError** – If there's an issue with the data content or format.

Returns

None. But exports a CSV file with merged list of space objects to 'src/fspsim/data/external/active_jsr_spacetrack.csv'.

Return type

None

CreateCatalogueAll()

Merge the JSR catalogue with the SpaceTrack catalogue.

The function prioritizes SpaceTrack's data in case of conflicting information. The resulting merged catalogue is stored in an attribute named CurrentCatalogueDF.

Raises

- **IOError** – If file paths are not found or unreadable.
- **ValueError** – If there's an issue with the data content or format.

Returns

None. But exports a CSV file of merged space catalogue of all tracked objects to 'src/fspsim/data/catalogue/All_catalogue_latest.txt'.

Return type

None

Note

In the case of missing data for the simulation, the function will drop the rows that do not have the required data. At present this results in a couple thousands of objects being dropped.

DownloadJSRCatalogueIfNewer(*local_path*, *url*)

Download a file from a given URL if it is newer than the local file.

Parameters

- **local_path** (*str*) – Local file path.
- **url** (*str*) – URL of the file to be downloaded.

Returns

None. Downloads the file if newer.

Return type
None

PullCatalogueJSR(*external_dir*)

Update the JSR catalogue files by downloading them if a newer version is available online.

Parameters
external_dir (*str*) – The directory path where the files will be saved.

Returns
None. Updates the JSR catalogue files.

Return type
None

PullCatalogueSpaceTrack(*external_dir*)

Download the entire SpaceTrack catalogue.

This method requires SpaceTrack login credentials stored in a '.env' file.

Parameters
external_dir (*str*) – The directory path where the SpaceTrack catalogue file will be saved.

Raises
Exception – If there's an issue with the login credentials or fetching data.

Returns
None. Saves the SpaceTrack catalogue in a JSON file.

Return type
None

classmethod load_from_file(*file_path*)

fspsim.utils.SpaceCatalogue.**check_json_file**(*json_data*)

Checks the validity of the provided JSON content.

Parameters
json_data (*dict*) – Dictionary parsed from a JSON file that needs to be checked.

Raises

- **KeyError** – If an expected key is not found in the JSON content.
- **ValueError** – If the value of a key is not in the list of valid options.

fspsim.utils.SpaceCatalogue.**dump_pickle**(*file_path*, *data*)

Saves provided data to a pickle file.

Parameters

- **file_path** (*str*) – Relative path where the data will be saved in pickle format.
- **data** (*Any*) – Data to be saved in the pickle file.

fspsim.utils.SpaceCatalogue.**get_path**(**args*)

Constructs a full path by joining the current working directory with the provided arguments.

Parameters
args (*str*) – Arguments that constitute the relative path.

Returns
The full path generated by joining the current directory with the provided arguments.

Return type
str

fspsim.utils.SpaceCatalogue.**load_pickle**(*file_path*)

Loads and returns data from a pickle file.

Parameters
file_path (*str*) – Relative path to the pickle file to be loaded.

Returns
Data loaded from the pickle file.

Return type
Any

1.1.1.7 fspsim.utils.SpaceObject module

class fspsim.utils.SpaceObject.**ObjectType**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

DEB = 'DEB'

OTHER = '?'

PAYLOAD = 'PAY'

ROCKET_BODY = 'R/B'

UNKNOWN = 'UNK'

class fspsim.utils.SpaceObject.**OperationalStatus**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

B = 'B'

D = 'D'

NEGATIVE = '-'

PARTIAL = 'P'

POSITIVE = '+'

S = 'S'

UNKNOWN = '?'

X = 'X'

class fspsim.utils.SpaceObject.**SpaceObject**(*rso_name=None*, *rso_type=None*, *payload_operational_status=None*, *application=None*, *source=None*, *launch_site=None*, *mass=None*, *maneuverable=False*, *spin_stabilized=False*, *object_type=None*, *apogee=None*, *perigee=None*, *characteristic_area=None*, *characteristic_length=None*, *propulsion_type=None*, *epoch=None*, *sma=None*, *inc=None*, *argp=None*, *raan=None*, *tran=None*, *eccentricity=None*, *operator=None*, *launch_date=None*, *decay_date=None*, *tle=None*, *station_keeping=None*, *orbit_source=None*)

Bases: object

A representation of a space object detailing its orbital and physical properties.

This class provides comprehensive details about the space object, including its launch date, operational status, object type, application, and a variety of other orbital parameters.

launch_date

The date when the object was launched.

Type
datetime

decay_date

The date when the object is expected to decay.

Type
datetime

rso_name

Name of the Resident Space Object (RSO).

Type
str

rso_type

Type of the RSO.

Type
str

payload_operational_status

Operational status of the payload.

Type
str

object_type

Type of the object. Can be 'DEB', 'PAY', 'R/B', 'UNK', or '?'.

Type
str

application

Application purpose of the space object.

Type
str

operator

The operator or owner of the object.

Type
str

characteristic_length

The characteristic length of the object.

Type
float

characteristic_area

The characteristic area of the object.

Type
float

mass

The mass of the object.

Type
float

source

Source or country of origin of the object.

Type
str

launch_site

The site where the object was launched from.

Type
str

maneuverable

Indicates if the object is maneuverable.

Type
str

spin_stabilized

Indicates if the object is spin-stabilized.

Type
str

apogee

The maximum altitude of the object's orbit.

Type
float

perigee

The minimum altitude of the object's orbit.

Type
float

propulsion_type

Type of propulsion used by the object.

Type
str

epoch

Reference time for the object's orbital parameters.

Type
datetime

day_of_year

Day of the year derived from the epoch.

Type
int

station_keeping

Dates indicating station keeping period.

Type
list or bool or None

ephemeris

Ephemeris data for the object.

Type
list

sma

Semi-major axis of the object's orbit.

Type
float

orbital_period

Orbital period of the object.

Type
float

inc

Inclination of the object's orbit.

Type
float

argp

Argument of perigee.

Type
float

raan

Right ascension of the ascending node.

Type
float

tran

True anomaly.

Type
float

eccentricity

Eccentricity of the object's orbit.

Type
float

meananomaly

Mean anomaly.

Type
float

cart_state

Cartesian state vector [x,y,z,u,v,w].

Type
np.array

C_d

Drag coefficient.

Type
float

tle

Two-line element set representing the object's orbit.

Type
str

generate_cart()

Generates a cartesian state vector from keplerian elements.

Returns
Cartesian state.

Return type
np.array

impute_char_area(*char_area*)

Imputes a characteristic area based on the object type if the given value is None or 0.

Parameters
char_area (*float or None*) – Characteristic area to verify or impute.

Returns
Imputed or verified characteristic area.

Return type
float

Raises
ValueError – If the imputed or given char_area is None or 0.

impute_char_length(*char_length*)

Imputes a characteristic length based on the object type if the given value is None or 0.

Parameters
char_length (*float or None*) – Characteristic length to verify or impute.

Returns
Imputed or verified characteristic length.

Return type
float

Raises
ValueError – If the imputed or given char_length is None or 0.

impute_mass(*mass*)

Imputes a mass based on the object characteristic length if the given value is None or 0. The relationship between mass and characteristic length is based on the relationship between mass and length reported in From Alfano, Oltrogge and Sheppard, 2020

Parameters
mass (*float or None or str*) – Mass to verify or impute.

Returns
Imputed mass value.

Return type
float

prop_catobject(*jd_start*, *jd_stop*, *step_size*, *output_freq*)

Propagates the object based on initial conditions, propagator type, and station keeping preferences.

Parameters

- **jd_start** (*float*) – Julian start date for the simulation.
- **jd_stop** (*float*) – Julian stop date for simulation.
- **step_size** (*float*) – Step size for propagation.
- **output_freq** (*float*) – Frequency at which to output the ephemeris (in seconds).
- **integrator_type** (*str*) – Numerical integrator to use.
- **use_sgp4_propagation** (*bool*) – Propagate using SGP4 for 100-minute segments.

Returns
None. Updates the *ephemeris* attribute of the object.

fspsim.utils.SpaceObject.**verify_angle**(*value*, *name*, *random=False*)

Verifies that the value is a float and is between 0 and 360. If it is None or not a float or out of range, raise a ValueError.

Parameters

- **value** (*float or None or str*) – The value to verify
- **name** (*str*) – Name of the object
- **random** (*bool, optional*) – If *True*, return a random angle between 0 and 360 for invalid values, defaults to *False*

Returns
The verified angle or a random angle if *random* is *True*

Return type
float

Raises
ValueError – If the value is not a float or out of range and *random* is *False*

fspsim.utils.SpaceObject.**verify_eccentricity**(*value*)

Verifies that the value is a float and is between 0 and 1. If it is None or not a float or out of range, raise a ValueError.

Parameters
value (*float or None or str*) – The value to verify

Returns
The verified value

Return type
float

Raises
ValueError – If the value is not a float or out of range

fspsim.utils.SpaceObject.**verify_value**(*value*, *impute_function*)

Verifies that the value is a float and is not None. If it is None or not a float or too small, then impute the value using the impute_function.

Parameters

- **value** (*float or None or str*) – The value to verify
- **impute_function** (*function*) – Function to impute value if necessary

Returns
The verified or imputed value

Return type
float

1.1.1.8 Module contents

1.2 Submodules

1.3 fspsim.simulate module

fspsim.simulate.**dump_pickle**(*file_path*, *data*)

fspsim.simulate.**get_path**(**args*)

fspsim.simulate.**load_pickle**(*file_path*)

fspsim.simulate.**propagate_space_object**(*args*)

fspsim.simulate.**run_sim**(*settings: <module 'json' from '/Users/charlesc/anaconda3/lib/python3.11/json/__init__.py'>*, *future_constellations_file: str = None*) → None

Propagates a list of space objects over a specified time range.

The function retrieves space objects from a space catalogue (SATCAT), propagates each object up to a the specified time. This will save the output locally but will also return a list of propagated space ojects. A progress bar is displayed to track the propagation process.

Parameters

- **settings** (–) – A dictionary containing the following key-value pairs:
- **"sim_start_date"** (–) – UTC start date for simulation (str)
- **"sim_end_date"** (–) – UTC end date for simulation (str)
- **"integrator_step_size"** (–) – Time step size for integrator (str, converted to int)
- **"output_frequency"** (–) – Output frequency (str, converted to int)
- **"scenario_name"** (–) – Name of the scenario (str)
- **"integrator_type"** (–) – Type of integrator to use (str)
- **"sgp4_long_term"** (–) – Boolean indicating if SGP4 long term propagation should be used (str, converted to bool)
- **"force_model"** (–) – Force model settings (can be various types, depending on implementation)

Returns: SATCAT Catalogue with updated ephemeris of the locations. Results are saved to pickle files in the directory: 'src/fspsim/data/results/propagated_catalogs/'.

Notes: - Each batch consists of a set number of space objects (defined by the batch_size variable). - The pickle files are saved with a filename pattern: '<scenario_name>_batch_<batch_number>.pickle'. - The propagation function used is 'propagate_space_object' (not defined in the provided code snippet).

fspsim.simulate.**set_future_constellations**(*constellations*) → bool

Incorporates user-specified launch predictions into the simulation. This will check that it is in the correct format and can be used by the simulation.

Parameters
constellations (*str*) – The path of the specified csv file

Returns
If it is in the correct format.

Return type
bool

1.4 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f