Your Project Documentation

Release 0.1

Your Name

Sep 13, 2023

CONTENTS:

1 fspsim package **1**
- 1.1 Subpackages . . . 1
  - 1.1.1 fspsim.utils package . . . 1
    - 1.1.1.1 Submodules . . . 1
    - 1.1.1.2 fspsim.utils.Conversions module . . . 1
    - 1.1.1.3 fspsim.utils.Formatting module . . . 8
    - 1.1.1.4 fspsim.utils.LaunchModel module . . . 8
    - 1.1.1.5 fspsim.utils.Propagators module . . . 10
    - 1.1.1.6 fspsim.utils.SpaceCatalogue module . . . 10
    - 1.1.1.7 fspsim.utils.SpaceObject module . . . 13
    - 1.1.1.8 Module contents . . . 16
- 1.2 Submodules . . . 16
- 1.3 fspsim.simulate module . . . 16
- 1.4 Module contents . . . 17

2 Indices and tables **19**

Python Module Index **21**

Index **23**

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

Note: Requires *numpy* for vector operations.

---

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

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.

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.

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 COSPAR ID for a spacecraft.

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.

TODO: Expand to include all possible altitudes. Classifications must be set to match JSR/Celestrak orbit classifications.

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*)

Generate 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*)

Format the scientific notion used in TLEs (usually for BSTAR)

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.

NOTE: Does not work on 3LE strings.

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. Note: This function may return the absolute value due to a potential sign error that needs fixing.

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

Note: The function uses the 'astropy' library for the conversion, so ensure that 'astropy' is installed and available.

---

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

Calculate the speed of the satellite relative to the Earth's atmosphere.

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 generates a Two-Line Element Set (TLE) for a satellite based on provided input data.

param catalog_number
Catalog number of the satellite.

type catalog_number
int

param classification
Classification type (usually 'U' for unclassified).

type classification
str

param launch_year
Year of launch.

type launch_year
int

param launch_number
Launch number of the year.

type launch_number
int

param launch_piece
Piece of the launch.

type launch_piece
str

#INCOMPLETE … (and so on for other parameters)

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

rtype
str

1.1.1.3 fspsim.utils.Formatting module

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

Reads a string describing the form factor of satellties in a sub constellation and returns characteristic length and area of to populate the SpaceObject class metadata return error if form factor is not a string

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*)

Generate instances of the SpaceObject class for each of the satellites 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*)

Creates a list of space objects for a given sub-constellation.

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*, *settings*, *monthly_ton_capacity*, *launches_start_date*, *rocket='Falcon 9'*)

Determines the launch dates for various sub-constellations.

Parameters

- **sub_constellation_metadata_dicts** (*list[dict]*) – List of metadata for each sub-constellation.
- **settings** (*dict*) – Configuration settings.
- **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.**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.

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 (MJD)

1.1.1.6 fspsim.utils.SpaceCatalogue module

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

Bases: object

Catalogue2SpaceObjects()

Convert the current catalogue into a list of SpaceObjects.

This function will prioritize the SpaceTrack data in case of conflicting information between JSR and SpaceTrack. The function takes into account the simulation object type (*sim_object_type*) to determine the data source for the conversion.

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 information with the entire 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

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 there's a newer version 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*)

Checks the validity of the provided JSON content.

Parameters
json (*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.
- **TypeError** – If the type of value for a key doesn't match the expected type.
- **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

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.

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.

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 type.

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 a celestial 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

This allows one to provide their own future constellation guide. 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

fspsim, 17
fspsim.simulate, 16
fspsim.utils, 16
fspsim.utils.Conversions, 1
fspsim.utils.Formatting, 8
fspsim.utils.LaunchModel, 8
fspsim.utils.Propagators, 10
fspsim.utils.SpaceCatalogue, 10
fspsim.utils.SpaceObject, 13

INDEX

B

B (*fspsim.utils.SpaceObject.OperationalStatus attribute*), 13

C

calculate_eccentricity() (*in module fspsim.utils.Conversions*), 1
calculate_energy_from_semi_major_axis() (*in module fspsim.utils.Conversions*), 2
calculate_form_factor() (*in module fspsim.utils.Formatting*), 8
car2kep() (*in module fspsim.utils.Conversions*), 2
Catalogue2SpaceObjects() (*fspsim.utils.SpaceCatalogue.SpaceCatalogue method*), 10
check_json_file() (*in module fspsim.utils.SpaceCatalogue*), 12
create_subconstellation_Space_Objects() (*in module fspsim.utils.LaunchModel*), 9
CreateCatalogueActive() (*fspsim.utils.SpaceCatalogue.SpaceCatalogue method*), 11
CreateCatalogueAll() (*fspsim.utils.SpaceCatalogue.SpaceCatalogue method*), 11

D

D (*fspsim.utils.SpaceObject.OperationalStatus attribute*), 13
DEB (*fspsim.utils.SpaceObject.ObjectType attribute*), 13
DownloadJSRCatalogueIfNewer() (*fspsim.utils.SpaceCatalogue.SpaceCatalogue method*), 11
dump_pickle() (*in module fspsim.simulate*), 16
dump_pickle() (*in module fspsim.utils.SpaceCatalogue*), 12

E

earth_moon_vec() (*in module fspsim.utils.Conversions*), 2
earth_sun_vec() (*in module fspsim.utils.Conversions*), 3
eccentric_to_mean_anomaly() (*in module fspsim.utils.Conversions*), 3
ecef2eci_astropy() (*in module fspsim.utils.Conversions*), 3

F

fspsim
    module, 17
fspsim.simulate
    module, 16
fspsim.utils
    module, 16
fspsim.utils.Conversions
    module, 1
fspsim.utils.Formatting
    module, 8
fspsim.utils.LaunchModel
    module, 8
fspsim.utils.Propagators
    module, 10
fspsim.utils.SpaceCatalogue
    module, 10
fspsim.utils.SpaceObject
    module, 13
future_constellations_csv_handler() (*in module fspsim.utils.Formatting*), 8

G

generate_cart() (*fspsim.utils.SpaceObject.SpaceObject method*), 13
generate_cospar_id() (*in module fspsim.utils.Conversions*), 3
get_day_of_year_and_fractional_day() (*in module fspsim.utils.Conversions*), 4
get_path() (*in module fspsim.simulate*), 16
get_path() (*in module fspsim.utils.SpaceCatalogue*), 12
global_launch_schedule() (*in module fspsim.utils.LaunchModel*), 9

I

import_configuration_json() (*in module fspsim.utils.LaunchModel*), 10

impute_char_area() (*fspsim.utils.SpaceObject.SpaceObject method*), 14
impute_char_length() (*fspsim.utils.SpaceObject.SpaceObject method*), 14
impute_mass() (*fspsim.utils.SpaceObject.SpaceObject method*), 14

J

jd_to_utc() (*in module fspsim.utils.Conversions*), 4

K

kep2car() (*in module fspsim.utils.Conversions*), 4
kepler_prop() (*in module fspsim.utils.Propagators*), 10

L

load_from_file() (*fspsim.utils.SpaceCatalogue.SpaceCatalogue class method*), 12
load_pickle() (*in module fspsim.simulate*), 16
load_pickle() (*in module fspsim.utils.SpaceCatalogue*), 12

M

module
    fspsim, 17
    fspsim.simulate, 16
    fspsim.utils, 16
    fspsim.utils.Conversions, 1
    fspsim.utils.Formatting, 8
    fspsim.utils.LaunchModel, 8
    fspsim.utils.Propagators, 10
    fspsim.utils.SpaceCatalogue, 10
    fspsim.utils.SpaceObject, 13

N

NEGATIVE (*fspsim.utils.SpaceObject.OperationalStatus attribute*), 13

O

ObjectType (*class in fspsim.utils.SpaceObject*), 13
OperationalStatus (*class in fspsim.utils.SpaceObject*), 13
orbit_classify() (*in module fspsim.utils.Conversions*), 4
orbital_period() (*in module fspsim.utils.Conversions*), 5
OTHER (*fspsim.utils.SpaceObject.ObjectType attribute*), 13

P

PARTIAL (*fspsim.utils.SpaceObject.OperationalStatus attribute*), 13
PAYLOAD (*fspsim.utils.SpaceObject.ObjectType attribute*), 13
POSITIVE (*fspsim.utils.SpaceObject.OperationalStatus attribute*), 13
Prediction2SpaceObjects() (*in module fspsim.utils.LaunchModel*), 8
probe_sun_vec() (*in module fspsim.utils.Conversions*), 5
prop_catobject() (*fspsim.utils.SpaceObject.SpaceObject method*), 14
propagate_space_object() (*in module fspsim.simulate*), 16
PullCatalogueJSR() (*fspsim.utils.SpaceCatalogue.SpaceCatalogue method*), 11
PullCatalogueSpaceTrack() (*fspsim.utils.SpaceCatalogue.SpaceCatalogue method*), 12

R

ROCKET_BODY (*fspsim.utils.SpaceObject.ObjectType attribute*), 13
run_sim() (*in module fspsim.simulate*), 16

S

S (*fspsim.utils.SpaceObject.OperationalStatus attribute*), 13
set_future_constellations() (*in module fspsim.simulate*), 16
sgp4_prop_TLE() (*in module fspsim.utils.Propagators*), 10
SpaceCatalogue (*class in fspsim.utils.SpaceCatalogue*), 10
SpaceObject (*class in fspsim.utils.SpaceObject*), 13

T

tle_checksum() (*in module fspsim.utils.Conversions*), 5
tle_convert() (*in module fspsim.utils.Conversions*), 5
tle_exponent_format() (*in module fspsim.utils.Conversions*), 6
tle_parse() (*in module fspsim.utils.Conversions*), 6
TLE_time() (*in module fspsim.utils.Conversions*), 1
true_to_eccentric_anomaly() (*in module fspsim.utils.Conversions*), 6
true_to_mean_anomaly() (*in module fspsim.utils.Conversions*), 6

U

UNKNOWN (*fspsim.utils.SpaceObject.ObjectType attribute*), 13
UNKNOWN (*fspsim.utils.SpaceObject.OperationalStatus attribute*), 13

UTC_step() (*in module fspsim.utils.Conversions*), 1
utc_to_jd() (*in module fspsim.utils.Conversions*), 6

V

v_rel() (*in module fspsim.utils.Conversions*), 7
verify_angle() (*in module fspsim.utils.SpaceObject*), 15
verify_eccentricity() (*in module fspsim.utils.SpaceObject*), 15
verify_value() (*in module fspsim.utils.SpaceObject*), 15

W

write_tle() (*in module fspsim.utils.Conversions*), 7

X

X (*fspsim.utils.SpaceObject.OperationalStatus attribute*), 13