
SWVO

Release 1.0.0


Bernhard Haas, Sahil Jhawar

Sep 18, 2025

API:

1	SWVO @ GFZ	1
1.1	Introduction	1
1.2	Solar Indices Overview	1
1.3	Installation	2
2	API	3
2.1	swvo	3
3	User Guide	45
3.1	Changelog	45
3.2	Contributing to SWVO	45
	Python Module Index	47
	Index	49

SWVO @ GFZ

pypi package 1.0.0 docs passing  SWVO Tests python 3.11 | 3.12 | 3.13 coverage 76%

1.1 Introduction

This package provides a set of tools for managing solar data in Python. It includes functionalities for reading, writing, and processing data from various sources.

1.2 Solar Indices Overview

This package provides tools to read, process, and analyze several key solar and geomagnetic indices. For each index, the available data sources and the corresponding reader classes are listed below:

- **Kp Index:** A global geomagnetic activity index with a 3-hour cadence, ranging from 0 (quiet) to 9 (extremely disturbed). Used to assess geomagnetic storm conditions.
 - **Sources & Classes:**
 - * OMNI: `KpOMNI`
 - * SWPC: `KpSWPC`
 - * Niemegk: `KpNiemegk`
 - * Ensemble: `KpEnsemble`
 - * Combined: `read_kp_from_multiple_models`
- **Dst Index:** The Disturbance Storm Time (Dst) index measures the intensity of the Earth's ring current, related to geomagnetic storms. Provided hourly and is negative during storm conditions.
 - **Sources & Classes:**
 - * OMNI: `DSTOMNI`
 - * WDC: `DSTWDC`
 - * Combined: `read_dst_from_multiple_models`
- **Hp Index:** The Hp30 and Hp60 indices are high-cadence (30-minute and 60-minute) geomagnetic indices provided by GFZ, used for detailed geomagnetic activity studies.
 - **Sources & Classes:**
 - * GFZ: `HpGFZ`
 - * Ensemble: `HpEnsemble`
 - * Combined: `read_hp_from_multiple_models`

- **F10.7 Index:** The F10.7 solar radio flux index is a daily measure of solar activity (flux density at 10.7 cm), a standard proxy for solar EUV emissions.
 - **Sources & Classes:**
 - * OMNI: F107OMNI
 - * SWPC: F107SWPC
 - * Combined: `read_f107_from_multiple_models`
- **Solar Wind Parameters:** Access to solar wind data (speed, density, magnetic field components) from various spacecraft. Essential for solar-terrestrial interaction studies.
 - **Sources & Classes:**
 - * ACE: SWACE
 - * DSCOVR: DSCOVER
 - * OMNI: SWOMNI
 - * SWIFT: SWSWIFTEnsemble
 - * Combined: `read_solar_wind_from_multiple_models`

Each index can be accessed via these dedicated reader classes, which handle downloading and read methods. See the code in `swvo/io` or API documentation for details on each index's implementation.

1.3 Installation

To install the package, run the following command:

`uv venv`

```
source .venv/bin/activate
python -m ensurepip --upgrade
uv pip install --upgrade pip
uv pip install -e .
```

or it can be installed directly from PyPI:

```
uv pip install swvo
```

All the above `uv` commands assume you have `uv` installed, if not then remove `uv` prefix from the commands and run them directly.

swvo

2.1 swvo

Modules

io

2.1.1 swvo.io

Modules

RBMDataset

base_file_reader

decorators

dst

exceptions

f10_7

hp

kp

omni

plasmasphere

solar_wind

continues on next page

Table 3 – continued from previous page

utils

swvo.io.RBMDataset**Modules**

<i>RBMDataset</i> (start_time, end_time, ..., ...)	RBMDataset class for loading and managing data.
<i>RBMDatasetElPaso</i> (satellite, instrument, mfm)	RBMDatasetElPaso class for loading ElPaso data to RBMDataset.
<i>RBMDatasetManager</i> ()	RBMDatasetManager class for managing RBMDataset instances.
<i>bin_and_interpolate_to_model_grid</i>	
<i>custom_enums</i>	
<i>interp_functions</i>	
<i>scripts</i>	
<i>utils</i>	

swvo.io.RBMDataset.RBMDataset

class swvo.io.RBMDataset.**RBMDataset**(start_time: *datetime*, end_time: *datetime*, folder_path: *Path*, satellite: *Literal*['RBSPA', 'RBSPB', 'GOES13', 'GOES14', 'GOES15', 'GOESPrimary', 'GOESSecondary', 'ARASE', 'NOAA15', 'NOAA16', 'NOAA18', 'NOAA19', 'METOP1', 'METOP2', 'DSX'] | *SatelliteEnum* | *Satellite*, instrument: *InstrumentEnum*, mfm: *MfmEnum*, preferred_extension: *str* = 'pickle', * (Keyword-only parameters separator (PEP 3102)), verbose: *bool* = True)

Bases: **object**

RBMDataset class for loading and managing data.

Attributes

datetime
[list[dt.datetime]]

time
[NDArray[np.float64]]

energy_channels
[NDArray[np.float64]]

alpha_local
[NDArray[np.float64]]

alpha_eq_model
[NDArray[np.float64]]

alpha_eq_real
[NDArray[np.float64]]

InvMu
 [NDArray[np.float64]]
InvMu_real
 [NDArray[np.float64]]
InvK
 [NDArray[np.float64]]
InvV
 [NDArray[np.float64]]
Lstar
 [NDArray[np.float64]]
Flux
 [NDArray[np.float64]]
PSD
 [NDArray[np.float64]]
MLT
 [NDArray[np.float64]]
B_SM
 [NDArray[np.float64]]
B_total
 [NDArray[np.float64]]
B_sat
 [NDArray[np.float64]]
xGEO
 [NDArray[np.float64]]
P
 [NDArray[np.float64]]
R0
 [NDArray[np.float64]]
density
 [NDArray[np.float64]]

__init__(start_time: *datetime*, end_time: *datetime*, folder_path: *Path*, satellite: *Literal*['RBSPA', 'RBSPB', 'GOES13', 'GOES14', 'GOES15', 'GOESPrimary', 'GOESSecondary', 'ARASE', 'NOAA15', 'NOAA16', 'NOAA18', 'NOAA19', 'METOP1', 'METOP2', 'DSX'] | *SatelliteEnum* | *Satellite*, instrument: *InstrumentEnum*, mfm: *MfmEnum*, preferred_extension: *str* = 'pickle', *, verbose: *bool* = True) → *None*

Methods

```

__init__(start_time, end_time, folder_path, ...)

bin_and_interpolate_to_model_grid(sim_time,
...)
get_print_name()

```

continues on next page

Table 5 – continued from previous page

<code>get_satellite_and_instrument_name()</code>
<code>get_satellite_name()</code>
<code>get_var(var)</code>
<code>interp_flux(target_en, target_al, target_type)</code>
<code>set_file_cadence(file_cadence)</code>
<code>set_file_name_stem(file_name_stem)</code>
<code>set_file_path_stem(file_path_stem)</code>

Attributes

<code>datetime</code>
<code>time</code>
<code>energy_channels</code>
<code>alpha_local</code>
<code>alpha_eq_model</code>
<code>alpha_eq_real</code>
<code>InvMu</code>
<code>InvMu_real</code>
<code>InvK</code>
<code>InvV</code>
<code>Lstar</code>
<code>Flux</code>
<code>PSD</code>
<code>MLT</code>
<code>B_SM</code>
<code>B_total</code>
<code>B_sat</code>

continues on next page

Table 6 – continued from previous page

xGEO
P
R0
density

swvo.io.RBMDataset.RBMDatasetElPaso

```
class swvo.io.RBMDataset.RBMDatasetElPaso(satellite: Literal['RBSPA', 'RBSPB', 'GOES13', 'GOES14',
'GOES15', 'GOESPrimary', 'GOESSecondary', 'ARASE',
'NOAA15', 'NOAA16', 'NOAA18', 'NOAA19', 'METOP1',
'METOP2', 'DSX'] | SatelliteEnum | Satellite, instrument:
InstrumentEnum, mfm: MfmEnum)
```

Bases: `object`

RBMDatasetElPaso class for loading ElPaso data to RBMDataset.

Parameters

satellite
[*SatelliteLike*] Satellite identifier as enum or string.

instrument
[*InstrumentEnum*] Instrument enumeration.

mfm
[*MfmEnum*] Magnetic field model enum.

Attributes

datetime
[list[dt.datetime]]

time
[NDArray[np.float64]]

energy_channels
[NDArray[np.float64]]

alpha_local
[NDArray[np.float64]]

alpha_eq_model
[NDArray[np.float64]]

alpha_eq_real
[NDArray[np.float64]]

InvMu
[NDArray[np.float64]]

InvMu_real
[NDArray[np.float64]]

InvK
[NDArray[np.float64]]

InvV

[NDArray[np.float64]] Calculate InvV.

Lstar

[NDArray[np.float64]]

Flux

[NDArray[np.float64]]

PSD

[NDArray[np.float64]]

MLT

[NDArray[np.float64]]

B_SM

[NDArray[np.float64]]

B_total

[NDArray[np.float64]]

B_sat

[NDArray[np.float64]]

xGEO

[NDArray[np.float64]]

P

[NDArray[np.float64]] Calculate P.

R0

[NDArray[np.float64]]

density

[NDArray[np.float64]]

__init__(satellite: *Literal*['RBSPA', 'RBSPB', 'GOES13', 'GOES14', 'GOES15', 'GOESPrimary', 'GOESSecondary', 'ARASE', 'NOAA15', 'NOAA16', 'NOAA18', 'NOAA19', 'METOP1', 'METOP2', 'DSX'] | *SatelliteEnum* | *Satellite*, instrument: *InstrumentEnum*, mfm: *MfmEnum*) → *None*

Methods

__init__ (satellite, instrument, mfm)	
update_from_dict (source_dict)	Get data from ElPaso data dictionary and update the object.

Attributes

InvV	Calculate InvV.
P	Calculate P.
instrument	Returns the instrument enum.
mfm	Returns the MFM enum.
satellite	Returns the satellite enum.
variable_mapping	Returns the variable mapping dictionary.

continues on next page

Table 8 – continued from previous page

datetime
time
energy_channels
alpha_local
alpha_eq_model
alpha_eq_real
InvMu
InvMu_real
InvK
Lstar
Flux
PSD
MLT
B_SM
B_total
B_sat
xGEO
R0
density

property satellite: *SatelliteEnum*

Returns the satellite enum.

property instrument: *InstrumentEnum*

Returns the instrument enum.

property mfm: *MfmEnum*

Returns the MFM enum.

property variable_mapping: *dict[str, str]*

Returns the variable mapping dictionary.

update_from_dict(*source_dict: dict[str, Variable]*) → *None*

Get data from ElPaso data dictionary and update the object.

Parameters**source_dict**

[dict[str, Any]] Dictionary containing the data to be loaded into the object.

property P: `ndarray[tuple[Any, ...], dtype[float64]]`

Calculate P.

Returns`NDArray[np.float64]`

The P value calculated from the MLT.

property InvV: `ndarray[tuple[Any, ...], dtype[float64]]`

Calculate InvV.

Returns`NDArray[np.float64]`

The InvV value calculated from InvMu and InvK.

swvo.io.RBMDDataSet.RBMDDataSetManager**class** `swvo.io.RBMDDataSet.RBMDDataSetManager`Bases: `object`

RBMDDataSetManager class for managing RBMDDataSet instances.

Raises**RuntimeError**If the constructor is called directly instead of using the *load* method.**Notes**Use the *load* class method to create and retrieve datasets. Direct instantiation is not allowed.`__init__()` → `None`**Methods**

<code>__init__()</code>	
<code>load()</code>	Loads an RBMDDataSet or a list of RBMDDataSets based on the provided parameters.

Attributes

<code>data_set_dict</code>

classmethod `load(start_time: datetime, end_time: datetime, folder_path: Path, satellite: Literal['RBSPA', 'RBSPB', 'GOES13', 'GOES14', 'GOES15', 'GOESPrimary', 'GOESSSecondary', 'ARASE', 'NOAA15', 'NOAA16', 'NOAA18', 'NOAA19', 'METOP1', 'METOP2', 'DSX'] | SatelliteEnum | Satellite, instrument: InstrumentEnum, mfm: MfmEnum, folder_type: FolderTypeEnum = FolderTypeEnum.DataServer, *, verbose: bool = True, preferred_extension: str = 'pickle') → RBMDDataSet`

```
classmethod load(start_time: datetime, end_time: datetime, folder_path: Path, satellite:
    Iterable[Literal['RBSPA', 'RBSPB', 'GOES13', 'GOES14', 'GOES15', 'GOESPrimary',
        'GOESSecondary', 'ARASE', 'NOAA15', 'NOAA16', 'NOAA18', 'NOAA19', 'METOP1',
        'METOP2', 'DSX'] | SatelliteEnum | Satellite], instrument: InstrumentEnum, mfm:
    MfmEnum, folder_type: FolderTypeEnum = FolderTypeEnum.DataServer, *, verbose:
    bool = True, preferred_extension: str = 'pickle') → list[RBMDDataSet]
```

Loads an *RBMDDataSet* or a list of *RBMDDataSets* based on the provided parameters.

Parameters

start_time

[*datetime*] Start time of the data set.

end_time

[*datetime*] End time of the data set.

folder_path

[*Path*] Path to the folder where the data set is stored.

satellite

[*SatelliteLike* | *Iterable*[*SatelliteLike*]] Satellite identifier(s) as enum or string. If a single satellite is provided, it can be a string or an enum.

instrument

[*InstrumentEnum*] Instrument enumeration, e.g., *InstrumentEnum.HOPE*.

mfm

[*MfmEnum*] Magnetic field model enum, e.g., *MfmEnum.T89*.

folder_type

[*FolderTypeEnum*, optional] Type of folder where the data is stored, by default *FolderTypeEnum.DataServer*.

verbose

[*bool*, optional] Whether to print verbose output, by default *True*.

preferred_extension

[*str*, optional] Preferred file extension for the data set to be loaded, by default “pickle”.

Returns

Union[*RBMDDataSet*, *list*[*RBMDDataSet*]]

An instance of *RBMDDataSet* or a list of *RBMDDataSet* instances, depending on the input parameters. Variables are lazily loaded from the file system when accessed.

swvo.io.RBMDDataSet.bin_and_interpolate_to_model_grid

Functions

```
bin_and_interpolate_to_model_grid(self, ...)
```

```
plot_debug_figures(data_set, psd_binned, ...)
```

Classes

DebugPlotSettings(folder_path, ...)

```
class swvo.io.RBMDataSet.bin_and_interpolate_to_model_grid.DebugPlotSettings(folder_path:
                                                                            'Path',
                                                                            satellite_name:
                                                                            'str', target_V:
                                                                            'float', target_K:
                                                                            'float')
```

Bases: `object`

swvo.io.RBMDataSet.custom_enums

Classes

ElPasoMFMEnum(value)

FileCadenceEnum(value)

FolderTypeEnum(value)

InstrumentEnum(value)

MfmEnum(value)

Satellite(sat_name, mission, folder_type, ...)

SatelliteEnum(value)

Variable(var_name, data_server_file_prefix, ...)

VariableEnum(value)

```
class swvo.io.RBMDataSet.custom_enums.FolderTypeEnum(value)
```

Bases: `Enum`

```
class swvo.io.RBMDataSet.custom_enums.FileCadenceEnum(value)
```

Bases: `Enum`

```
class swvo.io.RBMDataSet.custom_enums.Variable(var_name: 'str', data_server_file_prefix: 'str',
                                                data_server_has_B: 'bool')
```

Bases: `object`

```
class swvo.io.RBMDataSet.custom_enums.VariableEnum(value)
```

Bases: `Variable`, `Enum`


```
class swvo.io.RBMDDataSet.custom_enums.Satellite(sat_name: 'str', mission: 'str', folder_type:
    'FolderTypeEnum' = <FolderTypeEnum.DataServer:
    2>, file_cadence: 'FileCadenceEnum' =
    <FileCadenceEnum.Monthly: 1>)
```

Bases: `object`

```
class swvo.io.RBMDDataSet.custom_enums.SatelliteEnum(value)
```

Bases: `Satellite`, `Enum`

```
class swvo.io.RBMDDataSet.custom_enums.InstrumentEnum(value)
```

Bases: `Enum`

```
class swvo.io.RBMDDataSet.custom_enums.MfmEnum(value)
```

Bases: `Enum`

```
class swvo.io.RBMDDataSet.custom_enums.ElPasoMFEnum(value)
```

Bases: `Enum`

swvo.io.RBMDDataSet.interp_functions

Functions

```
interp_flux(self, target_en, target_al, ...)
```

```
interp_psd(self, target_K, target_type[, ...])
```

Classes

```
TargetType(value)
```

```
class swvo.io.RBMDDataSet.interp_functions.TargetType(value)
```

Bases: `Enum`

swvo.io.RBMDDataSet.scripts

Modules

```
create_RBSP_line_data
```

swvo.io.RBMDDataSet.scripts.create_RBSP_line_data

Functions

```
create_RBSP_line_data(start_time, end_time, ...)    Create RBSP line data for given time, energy, and alpha
local targets.
```

```
swvo.io.RBMDDataSet.scripts.create_RBSP_line_data.create_RBSP_line_data(start_time: datetime,
                                                                           end_time: datetime,
                                                                           data_server_path:
                                                                           Path, target_en: float |
                                                                           Iterable[float],
                                                                           target_al: float |
                                                                           Iterable[float],
                                                                           target_type: TargetType
                                                                           | Literal['TargetPairs',
                                                                           'TargetMeshGrid'], energy_offset_threshold:
                                                                           float = 0.1, instruments:
                                                                           list[InstrumentEnum] |
                                                                           None = None, satellites:
                                                                           list[Literal['RBSPA',
                                                                           'RBSPB', 'GOES13',
                                                                           'GOES14', 'GOES15',
                                                                           'GOESPrimary',
                                                                           'GOESSecondary',
                                                                           'ARASE', 'NOAA15',
                                                                           'NOAA16', 'NOAA18',
                                                                           'NOAA19', 'METOP1',
                                                                           'METOP2', 'DSX'] |
                                                                           SatelliteEnum |
                                                                           Satellite] |
                                                                           Literal['RBSPA',
                                                                           'RBSPB', 'GOES13',
                                                                           'GOES14', 'GOES15',
                                                                           'GOESPrimary',
                                                                           'GOESSecondary',
                                                                           'ARASE', 'NOAA15',
                                                                           'NOAA16', 'NOAA18',
                                                                           'NOAA19', 'METOP1',
                                                                           'METOP2', 'DSX'] |
                                                                           SatelliteEnum | Satellite
                                                                           | None = None, mfm:
                                                                           MfmEnum =
                                                                           MfmEnum.T89, *,
                                                                           adjust_targets: bool =
                                                                           True, verbose: bool =
                                                                           True) →
                                                                           tuple[list[<module
                                                                           'swvo.io.RBMDDataSet.RBMDDataSet'
                                                                           from
                                                                           '/home/runner/work/SWVO/SWVO/swvo/i
                                                                           list[~swvo.io.RBMDDataSet.custom_enums
```

Create RBSP line data for given time, energy, and alpha local targets.

Parameters

start_time

[datetime] Start time of the data to be loaded.

end_time

[datetime] End time of the data to be loaded.

data_server_path

[Path] Path to the data server where the RBSP data is stored.

target_en

[float or Iterable[float]] Target energy in MeV or list of target energies in MeV.

target_al

[float or Iterable[float]] Target alpha local in degrees or list of target alpha locals in degrees.

target_type

[TargetType or Literal["TargetPairs", "TargetMeshGrid"]] Type of target data to create. Can be either TargetPairs or TargetMeshGrid.

energy_offset_threshold

[float, optional] Threshold for the energy offset in relative units (default is 0.1).

instruments

[list[InstrumentEnum] or None, optional] List of instruments to use for the data. If None, defaults to HOPE, MAGEIS, and REPT.

satellites

[list[SatelliteLike] or SatelliteLike or None, optional] List of satellites to use for the data. If None, defaults to RBSPA and RBSPB.

mfm

[MfmEnum, optional] Magnetic field model to use for the data. Default is MfmEnum.T89.

adjust_targets

[bool, optional] If True, the targets will be adjusted to the closest available energy and alpha local values. If False, the targets will be interpolated from the available data. Default is True.

verbose

[bool, optional] If True, print verbose output during processing. Default is True.

Returns

tuple[list[RBMDDataSet], list[InstrumentEnum]]

A tuple containing a list of RBMDDataSet objects with the line data and a list of instruments used.

swvo.io.RBMDDataSet.utils**Functions**

<code>cart2pol(x, y)</code>	transforms cartesian coordinates x, y to polar coordinates theta (in rad) and radius
<code>get_file_path_any_format(folder_path, ...)</code>	Get the file path for a given file stem and preferred extension.
<code>join_var(var1, var2)</code>	Join two variables along the first axis.
<code>load_file_any_format(file_path)</code>	Load a file in any supported format and return its content.
<code>matlab2python(datumum)</code>	Convert MATLAB datumum to Python datetime.
<code>pol2cart(theta, radius)</code>	transforms polar coordinates theta (in rad) and radius to cartesian coordinates x, y
<code>python2matlab(datumum)</code>	Convert Python datetime to MATLAB datumum.
<code>round_seconds(obj)</code>	Round datetime object to the nearest second.

`swvo.io.RBMDDataSet.utils.join_var(var1: ndarray[tuple[Any, ...], dtype[generic]], var2: ndarray[tuple[Any, ...], dtype[generic]]) → ndarray[tuple[Any, ...], dtype[generic]]`

Join two variables along the first axis.

`swvo.io.RBMDataSet.utils.get_file_path_any_format(folder_path: Path, file_stem: str, preferred_ext: str) → Path | None`

Get the file path for a given file stem and preferred extension.

`swvo.io.RBMDataSet.utils.load_file_any_format(file_path: Path) → dict[str, Any]`

Load a file in any supported format and return its content.

`swvo.io.RBMDataSet.utils.round_seconds(obj: datetime) → datetime`

Round datetime object to the nearest second.

`swvo.io.RBMDataSet.utils.python2matlab(datenum: datetime) → float`

Convert Python datetime to MATLAB datenum.

`swvo.io.RBMDataSet.utils.matlab2python(datenum: float | Iterable) → Iterable[datetime] | datetime`

Convert MATLAB datenum to Python datetime.

`swvo.io.RBMDataSet.utils.pol2cart(theta: ndarray[tuple[Any, ...], dtype[float64]], radius: ndarray[tuple[Any, ...], dtype[float64]]) → tuple[ndarray[tuple[Any, ...], dtype[float64]], ndarray[tuple[Any, ...], dtype[float64]]]`

transforms polar coordinates theta (in rad) and radius to cartesian coordinates x, y

`swvo.io.RBMDataSet.utils.cart2pol(x: ndarray[tuple[Any, ...], dtype[float64]], y: ndarray[tuple[Any, ...], dtype[float64]]) → tuple[ndarray[tuple[Any, ...], dtype[float64]], ndarray[tuple[Any, ...], dtype[float64]]]`

transforms cartesian coordinates x, y to polar coordinates theta (in rad) and radius

swvo.io.base_file_reader

Classes

BaseReader()

swvo.io.decorators

Functions

<code>add_attributes_to_class_docstring(cls)</code>	Automatically adds attributes to the class docstring, including inherited attributes for derived classes.
<code>add_methods_to_class_docstring(cls)</code>	Automatically adds methods to the class docstring, including inherited methods for derived classes.
<code>add_time_docs([action])</code>	A decorator to add start_time and end_time parameters to the docstring.

`swvo.io.decorators.add_time_docs(action=None)`

A decorator to add start_time and end_time parameters to the docstring. It will add them at the beginning of the Parameters section.

`swvo.io.decorators.add_attributes_to_class_docstring(cls)`

Automatically adds attributes to the class docstring, including inherited attributes for derived classes.

`swvo.io.decorators.add_methods_to_class_docstring(cls)`

Automatically adds methods to the class docstring, including inherited methods for derived classes.

swvo.io.dst

Modules

<code>omni</code>	Module for handling OMNI Dst data.
<code>read_dst_from_multiple_models(start_time, ...)</code>	Read DST data from multiple models.
<code>wdc</code>	Module for handling WDC Dst data.

swvo.io.dst.omni

Module for handling OMNI Dst data.

Classes

<code>DSTOMNI([data_dir])</code>	Class for reading F10.7 data from OMNI DST files.
----------------------------------	---

class `swvo.io.dst.omni.DSTOMNI(data_dir: str | Path | None = None)`

Bases: `OMNILowRes`

Class for reading F10.7 data from OMNI DST files. Inherits the `download_and_process`, other private methods and attributes from `OMNILowRes`.

read(`start_time: datetime, end_time: datetime, download: bool = False`) → `DataFrame`

Read OMNI DST data for the given time range.

Parameters

start_time

[datetime] Start time of the data to read. Must be timezone-aware.

end_time

[datetime] End time of the data to read. Must be timezone-aware.

download

[bool, optional] Download data on the go, defaults to False.

Returns

`pandas.DataFrame`

OMNI DST data.

download_and_process(`start_time: datetime, end_time: datetime, reprocess_files: bool = False`) → `None`

Download and process OMNI Low Resolution data files.

Parameters

start_time

[datetime] Start time for the data to be downloaded and processed.

end_time

[datetime] End time for the data to be downloaded and processed.

reprocess_files

[bool, optional] Downloads and processes the files again, defaults to False, by default False

Returns

None

swvo.io.dst.read_dst_from_multiple_models

```
swvo.io.dst.read_dst_from_multiple_models(start_time: datetime, end_time: datetime, model_order:
                                         list[DSTOMNI | DSTWDC] | None = None,
                                         historical_data_cutoff_time: datetime | None = None, *,
                                         synthetic_now_time: datetime | None = None, download: bool
                                         = False) → DataFrame
```

Read DST data from multiple models.

The model order represents the priorities of models. The first model in the model order is read. If there are still NaNs in the resulting data, the next model will be read, and so on. For ensemble predictions, a list will be returned; otherwise, a plain data frame will be returned.

Parameters**start_time**

[datetime] Start time of the data request.

end_time

[datetime] End time of the data request.

model_order

[list or None, optional] Order in which data will be read from the models. Defaults to [OMNI, WDC].

historical_data_cutoff_time

[datetime or None, optional] Time representing “now”. After this time, no data will be taken from historical models (OMNI, WDC). Defaults to None.

download

[bool, optional] Flag indicating whether new data should be downloaded. Defaults to False.

Returns**pandas.DataFrame**

A data frame containing data for the requested period.

swvo.io.dst.wdc

Module for handling WDC Dst data.

Classes

DSTWDC([data_dir])

This is a class for the WDC Dst data.

```
class swvo.io.dst.wdc.DSTWDC(data_dir: str | Path | None = None)
```

Bases: `object`

This is a class for the WDC Dst data.

Parameters**data_dir**

[str | Path, optional] Data directory for the WDC Dst data. If not provided, it will be read from the environment variable

Methods

<code>download_and_process(start_time, end_time[, ...])</code>	Download and process WDC Dst data files.
<code>read(start_time, end_time[, download])</code>	Read WDC Dst data for the given time range.

Raises

ValueError

Raises *ValueError* if necessary environment variable is not set.

download_and_process(*start_time*: *datetime*, *end_time*: *datetime*, *reprocess_files*: *bool* = *False*) → *None*

Download and process WDC Dst data files.

Parameters

start_time

[datetime] Start time of the data to download. Must be timezone-aware.

end_time

[datetime] End time of the data to download. Must be timezone-aware.

reprocess_files

[bool, optional] Downloads and processes the files again, defaults to False, by default False

Returns

None

read(*start_time*: *datetime*, *end_time*: *datetime*, *download*: *bool* = *False*) → *DataFrame*

Read WDC Dst data for the given time range. it always returns the data until the last day of the month or incase of current month, until the current day.

Parameters

start_time

[datetime] Start time of the data to read. Must be timezone-aware.

end_time

[datetime] End time of the data to read. Must be timezone-aware.

download

[bool, optional] Download data on the go, defaults to False.

Returns

pandas.DataFrame

WDC Dst data.

swvo.io.exceptions

Exceptions

<i>ModelError</i>

exception swvo.io.exceptions.**ModelError**

Bases: *Exception*

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

swvo.io.f10_7**Modules**

<i>omni</i>	Module for handling F10.7 data from OMNI low resolution files.
<i>read_f107_from_multiple_models</i> (start_time, ...)	Read F107 data from multiple models.
<i>swpc</i>	Module for handling F10.7 data from SWPC.

swvo.io.f10_7.omni

Module for handling F10.7 data from OMNI low resolution files.

Classes

<i>F107OMNI</i> ([data_dir])	Class for reading F10.7 data from OMNI low resolution files.
------------------------------	--

class swvo.io.f10_7.omni.F107OMNI(*data_dir*: *str* | *Path* | *None* = *None*)

Bases: *OMNILowRes*

Class for reading F10.7 data from OMNI low resolution files. Inherits the *download_and_process()*, other private methods and attributes from *OMNILowRes*.

read(*start_time*: *datetime*, *end_time*: *datetime*, *download*: *bool* = *False*) → *DataFrame*

Extract F10.7 data from OMNI Low Resolution files.

Parameters**start_time**

[datetime] Start time of the data to read. Must be timezone-aware.

end_time

[datetime] End time of the data to read. Must be timezone-aware.

download

[bool, optional] Download data on the go, defaults to False.

Returns**pandas.DataFrame**

F10.7 from OMNI Low Resolution data.

download_and_process(*start_time*: *datetime*, *end_time*: *datetime*, *reprocess_files*: *bool* = *False*) → *None*

Download and process OMNI Low Resolution data files.

Parameters**start_time**

[datetime] Start time for the data to be downloaded and processed.

end_time

[datetime] End time for the data to be downloaded and processed.

reprocess_files

[bool, optional] Downloads and processes the files again, defaults to False, by default False

Returns

None

swvo.io.f10_7.read_f107_from_multiple_models

```
swvo.io.f10_7.read_f107_from_multiple_models(start_time: datetime, end_time: datetime, model_order:
list[F107OMNI | F107SWPC] | None = None,
historical_data_cutoff_time: datetime | None = None, *,
synthetic_now_time: datetime | None = None, download:
bool = False) → DataFrame
```

Read F107 data from multiple models.

The model order represents the priorities of models. The first model in the model order is read. If there are still NaNs in the resulting data, the next model will be read, and so on. For ensemble predictions, a list will be returned; otherwise, a plain data frame will be returned.

Parameters**start_time**

[datetime] Start time of the data request.

end_time

[datetime] End time of the data request.

model_order

[list or None, optional] Order in which data will be read from the models. Defaults to [OMNI, SWPC].

historical_data_cutoff_time

[datetime or None, optional] Time representing “now”. After this time, no data will be taken from historical models (OMNI, SWPC). Defaults to None.

download

[bool, optional] Flag indicating whether new data should be downloaded. Defaults to False.

Returns**pandas.DataFrame**

A data frame containing data for the requested period.

swvo.io.f10_7.swpc

Module for handling F10.7 data from SWPC.

Classes*F107SWPC*([data_dir])

This is a class for the SWPC F107 data.

```
class swvo.io.f10_7.swpc.F107SWPC(data_dir: str | Path | None = None)
```

Bases: `object`

This is a class for the SWPC F107 data.

Parameters**data_dir**

[str | Path, optional] Data directory for the OMNI Low Resolution data. If not provided, it will be read from the environment variable

Methods

<code>download_and_process()</code>	Download and process the latest 30-day F10.7 data.
<code>read(start_time, end_time, *, download)</code>	Read F10.7 SWPC data for the given time range.

Raises**ValueError**

Returns *ValueError* if necessary environment variable is not set.

`download_and_process()` → *None*

Download and process the latest 30-day F10.7 data.

Returns

None

`read(start_time: datetime, end_time: datetime, *, download: bool = False)` → *DataFrame*

Read F10.7 SWPC data for the given time range.

Parameters**start_time**

[*datetime*] Start time of the data to read. Must be timezone-aware.

end_time

[*datetime*] End time of the data to read. Must be timezone-aware.

download

[*bool*, optional] Download data on the go, defaults to *False*.

Returns

pandas.DataFrame

F10.7 data.

Raises**ValueError**

Raises *ValueError* if *start_time* is *after end_time*.

swvo.io.hp**Modules**

<code>ensemble</code>	
<code>gfz</code>	
<code>read_hp_from_multiple_models(start_time, ...)</code>	Read Hp data from multiple models.

swvo.io.hp.ensemble**Classes**

<code>Hp30Ensemble([data_dir])</code>	A class to handle Hp30 ensemble data.
<code>Hp60Ensemble([data_dir])</code>	A class to handle Hp30 ensemble data.
<code>HpEnsemble(index[, data_dir])</code>	This is a base class for Hp ensemble data.

class swvo.io.hp.ensemble.**HpEnsemble**(*index*: *str*, *data_dir*: *str* | *Path* | *None* = *None*)

Bases: `object`

This is a base class for Hp ensemble data.

Parameters**index**

[*str*] Hp index Possible options are: hp30, hp60.

data_dir

[*str* | *Path*, optional] Data directory for the Hp data. If not provided, it will be read from the environment variable

Methods

<code>read(start_time, end_time)</code>	Read Hp ensemble data for the requested period.
---	---

Raises**ValueError**

Returns *ValueError* if necessary environment variable is not set.

FileNotFoundError

Returns *FileNotFoundError* if the data directory does not exist.

read(*start_time*: *datetime*, *end_time*: *datetime*) → *list*[*DataFrame*]

Read Hp ensemble data for the requested period.

Parameters**start_time**

[*datetime*] Start time of the data to read. Must be timezone-aware.

end_time

[*datetime*] End time of the data to read. Must be timezone-aware.

Returns

list[*pandas.DataFrame*]

List of ensemble data frames containing data for the requested period.

Raises**FileNotFoundError**

Returns *FileNotFoundError* if no ensemble file is found for the requested date.

class swvo.io.hp.ensemble.**Hp30Ensemble**(*data_dir*: *str* | *Path* | *None* = *None*)

Bases: `HpEnsemble`

A class to handle Hp30 ensemble data.

Parameters**data_dir**

[str | Path, optional] Data directory for the Hp30 ensemble data. If not provided, it will be read from the environment variable

read(*start_time*: *datetime*, *end_time*: *datetime*) → list[DataFrame]

Read Hp ensemble data for the requested period.

Parameters**start_time**

[datetime] Start time of the data to read. Must be timezone-aware.

end_time

[datetime] End time of the data to read. Must be timezone-aware.

Returns

list[pandas.DataFrame]

List of ensemble data frames containing data for the requested period.

Raises**FileNotFoundError**

Returns *FileNotFoundError* if no ensemble file is found for the requested date.

class swvo.io.hp.ensemble.Hp600Ensemble(*data_dir*: str | Path | None = None)

Bases: [HpEnsemble](#)

A class to handle Hp30 ensemble data.

Parameters**data_dir**

[str | Path, optional] Data directory for the Hp30 ensemble data. If not provided, it will be read from the environment variable

read(*start_time*: *datetime*, *end_time*: *datetime*) → list[DataFrame]

Read Hp ensemble data for the requested period.

Parameters**start_time**

[datetime] Start time of the data to read. Must be timezone-aware.

end_time

[datetime] End time of the data to read. Must be timezone-aware.

Returns

list[pandas.DataFrame]

List of ensemble data frames containing data for the requested period.

Raises**FileNotFoundError**

Returns *FileNotFoundError* if no ensemble file is found for the requested date.

swvo.io.hp.gfz

Classes

<code>Hp30GFZ([data_dir])</code>	A class to handle Hp30 data from GFZ.
<code>Hp60GFZ([data_dir])</code>	A class to handle Hp30 data from GFZ.
<code>HpGFZ(index[, data_dir])</code>	This is a base class for HpGFZ data.

class swvo.io.hp.gfz.**HpGFZ**(*index*: *str*, *data_dir*: *str* | *Path* | *None* = *None*)

Bases: `object`

This is a base class for HpGFZ data.

Parameters

index

[*str*] Hp index. Possible options are: hp30, hp60.

data_dir

[*str* | *Path*, optional] Data directory for the Hp data. If not provided, it will be read from the environment variable

Methods

<code>download_and_process(start_time, end_time, *)</code>	Download and process HpGFZ data.
<code>read(start_time, end_time, *[, download])</code>	Read HpGFZ data for the given time range.

Raises

ValueError

Returns *ValueError* if necessary environment variable is not set.

download_and_process(*start_time*: *datetime*, *end_time*: *datetime*, *, *reprocess_files*: *bool* = *False*) → *None*

Download and process HpGFZ data.

Parameters

start_time

[*datetime*] Start time of the data to be downloaded.

end_time

[*datetime*] End time of the data to be downloaded.

reprocess_files

[*bool*, optional] Downloads and processes the files again, defaults to False, by default False

Returns

None

read(*start_time*: *datetime*, *end_time*: *datetime*, *, *download*: *bool* = *False*) → *DataFrame*

Read HpGFZ data for the given time range.

Parameters

start_time

[*datetime*] Start time of the data to read. Must be timezone-aware.

end_time

[datetime] End time of the data to read. Must be timezone-aware.

download

[bool, optional] Download data on the go, defaults to False.

Returns

pandas.DataFrame

HpGFZ data for the given time range.

class swvo.io.hp.gfz.Hp30GFZ(*data_dir: str | Path | None = None*)

Bases: *HpGFZ*

A class to handle Hp30 data from GFZ.

Parameters

data_dir

[str | Path, optional] Data directory for the Hp30 data. If not provided, it will be read from the environment variable

download_and_process(*start_time: datetime, end_time: datetime, *, reprocess_files: bool = False*) → *None*

Download and process HpGFZ data.

Parameters

start_time

[datetime] Start time of the data to be downloaded.

end_time

[datetime] End time of the data to be downloaded.

reprocess_files

[bool, optional] Downloads and processes the files again, defaults to False, by default False

Returns

None

read(*start_time: datetime, end_time: datetime, *, download: bool = False*) → *DataFrame*

Read HpGFZ data for the given time range.

Parameters

start_time

[datetime] Start time of the data to read. Must be timezone-aware.

end_time

[datetime] End time of the data to read. Must be timezone-aware.

download

[bool, optional] Download data on the go, defaults to False.

Returns

pandas.DataFrame

HpGFZ data for the given time range.

class swvo.io.hp.gfz.Hp60GFZ(*data_dir: str | Path | None = None*)

Bases: *HpGFZ*

A class to handle Hp30 data from GFZ.

Parameters**data_dir**

[str | Path, optional] Data directory for the Hp30 data. If not provided, it will be read from the environment variable

download_and_process(*start_time*: *datetime*, *end_time*: *datetime*, *, *reprocess_files*: *bool* = *False*) → *None*

Download and process HpGFZ data.

Parameters**start_time**

[datetime] Start time of the data to be downloaded.

end_time

[datetime] End time of the data to be downloaded.

reprocess_files

[bool, optional] Downloads and processes the files again, defaults to False, by default False

Returns

None

read(*start_time*: *datetime*, *end_time*: *datetime*, *, *download*: *bool* = *False*) → *DataFrame*

Read HpGFZ data for the given time range.

Parameters**start_time**

[datetime] Start time of the data to read. Must be timezone-aware.

end_time

[datetime] End time of the data to read. Must be timezone-aware.

download

[bool, optional] Download data on the go, defaults to False.

Returns

pandas.DataFrame

HpGFZ data for the given time range.

swvo.io.hp.read_hp_from_multiple_models

swvo.io.hp.read_hp_from_multiple_models(*start_time*: *datetime*, *end_time*: *datetime*, *model_order*: *list*[*Hp30Ensemble* | *Hp30GFZ* | *Hp60Ensemble* | *Hp60GFZ*] | *None* = *None*, *hp_index*: *str* = 'hp30', *reduce_ensemble*: *Literal*['mean', 'median'] | *None* = *None*, *historical_data_cutoff_time*: *datetime* | *None* = *None*, *, *synthetic_now_time*: *datetime* | *None* = *None*, *download*: *bool* = *False*) → *DataFrame* | *list*[*DataFrame*]

Read Hp data from multiple models.

The model order represents the priorities of models. The first model in the model order is read. If there are still NaNs in the resulting data, the next model will be read. And so on. In the case of reading ensemble predictions, a list will be returned, otherwise a plain data frame will be returned.

Parameters

start_time

[datetime] Start time of the data request.

end_time

[datetime] End time of the data request.

model_order

[list, optional] Order in which data will be read from the models, defaults to [OMNI, Niemegek, Ensemble, SWPC].

reduce_ensemble

[{"mean"}, optional] The method to reduce ensembles to a single time series, defaults to None.

historical_data_cutoff_time

[datetime, optional] Time, which represents “now”. After this time, no data will be taken from historical models (OMNI, Niemegek), defaults to None.

download

[bool, optional] Flag which decides whether new data should be downloaded, defaults to False.

Returns

Union[pandas.DataFrame, list[pandas.DataFrame]]

A data frame or a list of data frames containing data for the requested period.

swvo.io.kp**Modules**

<i>ensemble</i>	Module for handling SWIFT Kp ensemble data.
<i>niemegek</i>	Module for handling Niemegek Kp data.
<i>omni</i>	Module holding the reader for reading Kp data from OMNI files.
<i>read_kp_from_multiple_models</i> (start_time, ...)	Read Kp data from multiple models.
<i>swpc</i>	Module for handling SWPC Kp data.

swvo.io.kp.ensemble

Module for handling SWIFT Kp ensemble data.

Classes

<i>KpEnsemble</i> ([data_dir])	This is a class for Kp ensemble data.
--------------------------------	---------------------------------------

```
class swvo.io.kp.ensemble.KpEnsemble(data_dir: str | Path | None = None)
```

Bases: `object`

This is a class for Kp ensemble data.

Parameters**data_dir**

[str | Path, optional] Data directory for the Hp data. If not provided, it will be read from the environment variable

Methods

<code>read(start_time, end_time)</code>	Read Kp ensemble data for the requested period.
---	---

Raises

ValueError

Returns *ValueError* if necessary environment variable is not set.

FileNotFoundError

Returns *FileNotFoundError* if the data directory does not exist.

read(*start_time*: *datetime*, *end_time*: *datetime*) → list

Read Kp ensemble data for the requested period.

Parameters

start_time

[datetime] Start time of the period for which to read the data.

end_time

[datetime] End time of the period for which to read the data.

Returns

list[pandas.DataFrame]

A list of data frames containing ensemble data for the requested period.

Raises

FileNotFoundError

Raises *FileNotFoundError* if no ensemble files are found for the requested date.

swvo.io.kp.niemegk

Module for handling Niemegek Kp data.

Classes

<code>KpNiemegek([data_dir])</code>	A class to handle Niemegek Kp data.
-------------------------------------	-------------------------------------

class swvo.io.kp.niemegk.**KpNiemegek**(*data_dir*: str | Path | None = None)

Bases: object

A class to handle Niemegek Kp data.

Parameters

data_dir

[str | Path, optional] Data directory for the Niemegek Kp data. If not provided, it will be read from the environment variable

Methods

<code>download_and_process(start_time, end_time[, Download and process Niemegk Kp data file. ...])</code>	
<code>read(start_time, end_time[, download])</code>	Read Niemegk Kp data for the specified time range.

Raises**ValueError**

Returns *ValueError* if necessary environment variable is not set.

download_and_process(*start_time*: *datetime*, *end_time*: *datetime*, *reprocess_files*: *bool* = *False*) → *None*

Download and process Niemegk Kp data file.

Parameters**start_time**

[datetime] Start time of the data to download and process.

end_time

[datetime] End time of the data to download and process.

reprocess_files

[bool, optional] Downloads and processes the files again, defaults to False, by default False

Raises**FileNotFoundError**

Raise *FileNotFoundError* if the file is not downloaded successfully.

read(*start_time*: *datetime*, *end_time*: *datetime*, *download*: *bool* = *False*) → *DataFrame*

Read Niemegk Kp data for the specified time range.

Parameters**start_time**

[datetime] Start time of the data to read.

end_time

[datetime] End time of the data to read.

download

[bool, optional] Download data on the go, defaults to False.

Returns**pandas.DataFrame**

Niemegk Kp dataframe.

swvo.io.kp.omni

Module holding the reader for reading Kp data from OMNI files.

Classes

<code>KpOMNI([data_dir])</code>	Class for reading Kp data from OMNI low resolution files.
---------------------------------	---

```
class swvo.io.kp.omni.KpOMNI(data_dir: str | Path | None = None)
```

Bases: `OMNILowRes`

Class for reading Kp data from OMNI low resolution files. Inherits the `download_and_process()`, other private methods and attributes from `OMNILowRes`.

```
read(start_time: datetime, end_time: datetime, *, download: bool = False) → DataFrame
```

Extract Kp data from OMNI Low Resolution files.

Parameters

start_time

[datetime] Start time of the data to read.

end_time

[datetime] End time of the data to read.

download

[bool, optional] Download data on the go, defaults to False.

Returns

`pandas.DataFrame`

Kp data from OMNI Low Resolution data.

```
download_and_process(start_time: datetime, end_time: datetime, reprocess_files: bool = False) → None
```

Download and process OMNI Low Resolution data files.

Parameters

start_time

[datetime] Start time for the data to be downloaded and processed.

end_time

[datetime] End time for the data to be downloaded and processed.

reprocess_files

[bool, optional] Downloads and processes the files again, defaults to False, by default False

Returns

`None`

swvo.io.kp.read_kp_from_multiple_models

```
swvo.io.kp.read_kp_from_multiple_models(start_time: datetime, end_time: datetime, model_order:
    list[KpEnsemble | KpNiemegk | KpOMNI | KpSWPC] | None =
    None, reduce_ensemble: Literal['mean', 'median'] | None =
    None, historical_data_cutoff_time: datetime | None = None, *,
    synthetic_now_time: datetime | None = None, download: bool =
    False, recurrence: bool = False, rec_model_order:
    list[KpOMNI | KpNiemegk] = None) → DataFrame |
    list[DataFrame]
```

Read Kp data from multiple models.

The model order determines the priority of models. Data is read from the first model in the model order. If there are still NaNs in the resulting data, the next model is read, and so on. For ensemble predictions, a list of data frames is returned; otherwise, a single data frame is returned.

Parameters

start_time

[datetime] The start time of the data request.

end_time

[datetime] The end time of the data request.

model_order

[list or None, optional] The order in which data will be read from the models. Defaults to [OMNI, Niemegek, Ensemble, SWPC].

reduce_ensemble

[{"mean", None}, optional] The method to reduce ensembles to a single time series. Defaults to None.

historical_data_cutoff_time

[datetime or None, optional] Represents "now". After this time, no data will be taken from historical models (OMNI, Niemegek). Defaults to None.

download

[bool, optional] Flag to decide whether new data should be downloaded. Defaults to False. Also applies to recurrence filling.

recurrence

[bool, optional] If True, fill missing values using 27-day recurrence from historical models (OMNI, Niemegek). Defaults to False.

rec_model_order

[list[KpOMNI | KpNiemegek], optional] The order in which historical models will be used for 27-day recurrence filling. Defaults to [OMNI, Niemegek].

Returns

Union[pandas.DataFrame, list[pandas.DataFrame]]

A data frame or a list of data frames containing data for the requested period.

swvo.io.kp.swpc

Module for handling SWPC Kp data.

Classes

KpSWPC([data_dir])

A class for handling SWPC Kp data.

class swvo.io.kp.swpc.**KpSWPC**(data_dir: str | Path | None = None)

Bases: `object`

A class for handling SWPC Kp data. In SWPC data, the file for current day always contains the forecast for the next 3 days. Keep this in mind when using the *read* and *download_and_process* methods.

Parameters**data_dir**

[str | Path, optional] Data directory for the SWPC Kp data. If not provided, it will be read from the environment variable

Methods

<code>download_and_process(target_date[, ...])</code>	Download and process SWPC Kp data file.
<code>read(start_time[, end_time, download])</code>	Read Kp data for the specified time range.

Raises

ValueError

Returns *ValueError* if necessary environment variable is not set

download_and_process(target_date: *datetime*, reprocess_files: *bool* = False)

Download and process SWPC Kp data file.

Parameters

target_date

[datetime] Target date for the Kp data,

reprocess_files

[bool, optional] Downloads and processes the files again, defaults to False, by default False

Raises

ValueError

Raises *ValueError* if the target date is in the past.

read(start_time: *datetime*, end_time: *datetime* = None, download: *bool* = False) → *DataFrame*

Read Kp data for the specified time range.

Parameters

start_time

[datetime] Start time of the data to read.

end_time

[datetime, optional] End time of the data to read. If not provided, it will be set to 3 days after *start_time*.

download

[bool, optional] Download data on the go, defaults to False.

Returns

pd.DataFrame

SWPC Kp dataframe.

Raises

ValueError

Raises *ValueError* if the time range is more than 3 days.

swvo.io.omni

Modules

<code>omni_high_res</code>	Module for handling OMNI high resolution data.
<code>omni_low_res</code>	Module for handling OMNI low resolution data.

swvo.io.omni.omni_high_res

Module for handling OMNI high resolution data.

Classes

<code>OMNIHighRes([data_dir])</code>	This is a class for the OMNI High Resolution data.
--------------------------------------	--

```
class swvo.io.omni.omni_high_res.OMNIHighRes(data_dir: str | Path | None = None)
```

Bases: `object`

This is a class for the OMNI High Resolution data.

Parameters

`data_dir`

[str | Path, optional] Data directory for the OMNI High Resolution data. If not provided, it will be read from the environment variable

Methods

<code>download_and_process(start_time, end_time[, ...])</code>	Download and process OMNI High Resolution data files.
<code>read(start_time, end_time[, cadence_min, ...])</code>	Read OMNI High Resolution data for the given time range.

Raises

`ValueError`

Returns *ValueError* if necessary environment variable is not set.

download_and_process(start_time: *datetime*, end_time: *datetime*, cadence_min: *float* = 1, reprocess_files: *bool* = False) → *None*

Download and process OMNI High Resolution data files.

Parameters

`start_time`

[datetime] Start time for data download.

`end_time`

[datetime] End time for data download.

`cadence_min`

[float, optional] Cadence of the data in minutes, defaults to 1

`reprocess_files`

[bool, optional] Downloads and processes the files again, defaults to False, by default False

Returns

None

Raises

`AssertionError`

Raises *AssertionError* if the cadence is not 1 or 5 minutes.

read(*start_time: datetime, end_time: datetime, cadence_min: float = 1, download: bool = False*) → `DataFrame`

Read OMNI High Resolution data for the given time range.

Parameters

start_time

[datetime] Start time for reading data.

end_time

[datetime] End time for reading data.

cadence_min

[float, optional] Cadence of the data in minutes, defaults to 1

download

[bool, optional] Download data on the go, defaults to False.

Returns

pandas.DataFrame

OMNI High Resolution data.

Raises

AssertionError

Raises *AssertionError* if the cadence is not 1 or 5 minutes.

swvo.io.omni.omni_low_res

Module for handling OMNI low resolution data.

Classes

<code>OMNILowRes([data_dir])</code>	This is a class for the OMNI Low Resolution data.
-------------------------------------	---

class swvo.io.omni.omni_low_res.**OMNILowRes**(*data_dir: str | Path | None = None*)

Bases: `object`

This is a class for the OMNI Low Resolution data.

Parameters

data_dir

[str | Path, optional] Data directory for the OMNI Low Resolution data. If not provided, it will be read from the environment variable

Methods

<code>download_and_process(start_time, end_time[, ...])</code>	Download and process OMNI Low Resolution data files.
<code>read(start_time, end_time[, download])</code>	Read OMNI Low Resolution data for the given time range.

Raises

ValueError

Returns *ValueError* if necessary environment variable is not set.

download_and_process(*start_time: datetime, end_time: datetime, reprocess_files: bool = False*) → *None*

Download and process OMNI Low Resolution data files.

Parameters**start_time**

[datetime] Start time for the data to be downloaded and processed.

end_time

[datetime] End time for the data to be downloaded and processed.

reprocess_files

[bool, optional] Downloads and processes the files again, defaults to False, by default False

Returns

None

read(*start_time: datetime, end_time: datetime, download: bool = False*) → *DataFrame*

Read OMNI Low Resolution data for the given time range.

Parameters**start_time**

[datetime] Start time for the data to be read.

end_time

[datetime] End time for the data to be read.

download

[bool, optional] Download data on the go, defaults to False.

Returns

pandas.DataFrame

OMNI Low Resolution data.

swvo.io.plasmasphere**Modules**

*read_plasmasphere**read_plasmasphere_combined_inputs*

swvo.io.plasmasphere.read_plasmasphere**Classes**

PlasmaspherePredictionReader(folder)

swvo.io.plasmasphere.read_plasmasphere_combined_inputs**Classes**

PlasmasphereCombinedInputsReader(folder)
--

swvo.io.solar_wind**Modules**

<i>ace</i>	Module for handling ACE Solar Wind data.
<i>dscovr</i>	Module for handling DSCOVR Solar Wind data.
<i>omni</i>	Module handling SW data from OMNI High Resolution files.
<i>read_solar_wind_from_multiple_models(...[, ...])</i>	Read solar wind data from multiple models.
<i>swift</i>	Module for handling SWIFT solar wind ensemble data.

swvo.io.solar_wind.ace

Module for handling ACE Solar Wind data.

Classes

<i>SWACE</i> ([data_dir])	This is a class for the ACE Solar Wind data.
---------------------------	--

class swvo.io.solar_wind.ace.**SWACE**(data_dir: *str* | *Path* | *None* = *None*)

Bases: *object*

This is a class for the ACE Solar Wind data.

Parameters**data_dir**

[*str* | *Path*, optional] Data directory for the ACE Solar Wind data. If not provided, it will be read from the environment variable

Methods

<i>download_and_process</i> (request_time)	Download and process ACE data, splitting data across midnight into appropriate day files.
<i>read</i> (start_time, end_time[, download, ...])	Read ACE data for the specified time range.

Raises**ValueError**

Returns *ValueError* if necessary environment variable is not set.

download_and_process(request_time: *datetime*) → *None*

Download and process ACE data, splitting data across midnight into appropriate day files.

Parameters

request_time

[datetime] The time for which the data is requested. Must be in the past and within the last two hours.

Returns

None

Raises**AssertionError**

If the request_time is in the future.

FileNotFoundError

If the downloaded files are empty.

read(start_time: *datetime*, end_time: *datetime*, download: *bool* = False, propagation: *bool* = False) → *DataFrame*

Read ACE data for the specified time range.

Parameters**start_time**

[datetime] Start time of the data to read.

end_time

[datetime] End time of the data to read.

download

[bool, optional] Download data on the go, defaults to False.

propagation

[bool, optional] Propagate the data from L1 to near-Earth, defaults to False.

Returns

pandas.DataFrame

ACE data

Raises**AssertionError**

Raises *AssertionError* if the start time is before the end time.

swvo.io.solar_wind.dscovr

Module for handling DSCOVR Solar Wind data.

Classes

DSCOVR([data_dir])

This is a class for the DSCOVR Solar Wind data.

class swvo.io.solar_wind.dscovr.*DSCOVR*(data_dir: *str* | *Path* | *None* = *None*)

Bases: *object*

This is a class for the DSCOVR Solar Wind data.

Parameters**data_dir**

[*str* | *Path*, optional] Data directory for the DSCOVR Solar Wind data. If not provided, it will be read from the environment variable

Methods

<code>download_and_process(request_time)</code>	Download and process DSCOVR data, splitting data across midnight into appropriate day files.
<code>read(start_time, end_time[, download, ...])</code>	Read DSCOVR data for the specified time range.

Raises

ValueError

Returns *ValueError* if necessary environment variable is not set.

download_and_process(*request_time: datetime*) → *None*

Download and process DSCOVR data, splitting data across midnight into appropriate day files.

Parameters

request_time

[datetime] The time for which the data is requested. Must be in the past and within the last two hours.

Returns

None

Raises

AssertionError

If the *request_time* is in the future.

FileNotFoundError

If the downloaded files are empty.

read(*start_time: datetime, end_time: datetime, download: bool = False, propagation: bool = False*) → *DataFrame*

Read DSCOVR data for the specified time range.

Parameters

start_time

[datetime] Start time of the data to read. Must be timezone-aware.

end_time

[datetime] End time of the data to read. Must be timezone-aware. If not provided, it defaults to 3 days after the start time. If *propagation* is True, it defaults to 2 days after the start time. If *propagation* is False, it defaults to 3 days after the start time.

download

[bool, optional] Download data on the go, defaults to False.

propagation

[bool, optional] Propagate the data from L1 to near-Earth, defaults to False.

Returns

pandas.DataFrame

DataFrame containing DSCOVR Solar Wind data for the requested period.

Raises

AssertionError

Raises *AssertionError* if the start time is before the end time.

swvo.io.solar_wind.omni

Module handling SW data from OMNI High Resolution files.

Classes

<code>SWOMNI([data_dir])</code>	Class for reading SW data from OMNI High resolution files.
---------------------------------	--

```
class swvo.io.solar_wind.omni.SWOMNI(data_dir: str | Path | None = None)
```

Bases: `OMNIHighRes`

Class for reading SW data from OMNI High resolution files. Inherits the `download_and_process()`, other private methods and attributes from `OMNIHighRes`.

download_and_process(start_time: *datetime*, end_time: *datetime*, cadence_min: *float* = 1, reprocess_files: *bool* = False) → None

Download and process OMNI High Resolution data files.

Parameters

start_time

[datetime] Start time for data download.

end_time

[datetime] End time for data download.

cadence_min

[float, optional] Cadence of the data in minutes, defaults to 1

reprocess_files

[bool, optional] Downloads and processes the files again, defaults to False, by default False

Returns

None

Raises

AssertionError

Raises *AssertionError* if the cadence is not 1 or 5 minutes.

read(start_time: *datetime*, end_time: *datetime*, cadence_min: *float* = 1, download: *bool* = False) → `DataFrame`

Read OMNI High Resolution data for the given time range.

Parameters

start_time

[datetime] Start time for reading data.

end_time

[datetime] End time for reading data.

cadence_min

[float, optional] Cadence of the data in minutes, defaults to 1

download

[bool, optional] Download data on the go, defaults to False.

Returns

`pandas.DataFrame`

OMNI High Resolution data.

Raises**AssertionError**Raises *AssertionError* if the cadence is not 1 or 5 minutes.**swvo.io.solar_wind.read_solar_wind_from_multiple_models**

```
swvo.io.solar_wind.read_solar_wind_from_multiple_models(start_time: datetime, end_time: datetime,
                                                         model_order: list[DSCOVER | SWACE |
                                                         SWOMNI | SWSWIFTEnsemble] | None =
                                                         None, reduce_ensemble: str | None = None,
                                                         historical_data_cutoff_time: datetime |
                                                         None = None, *, synthetic_now_time:
                                                         datetime | None = None, download: bool =
                                                         False) → DataFrame | list[DataFrame]
```

Read solar wind data from multiple models.

The model order represents the priorities of models. The first model in the model order is read. If there are still NaNs in the resulting data, the next model will be read. And so on. In the case of reading ensemble predictions, a list will be returned, otherwise a plain data frame will be returned.

Parameters**start_time**

[datetime] Start time of the data request.

end_time

[datetime] End time of the data request.

model_order

[list, optional] Order in which data will be read from the models. Defaults to [OMNI, ACE, SWIFT].

reduce_ensemble

[{'mean'}, optional] The method to reduce ensembles to a single time series. Defaults to None.

historical_data_cutoff_time

[datetime, optional] Time which represents “now”. After this time, no data will be taken from historical models (OMNI, ACE). Defaults to None.

download

[bool, optional] Flag which decides whether new data should be downloaded. Defaults to False.

Returns`Union[pandas.DataFrame, list[pandas.DataFrame]]`

A data frame or a list of data frames containing data for the requested period.

swvo.io.solar_wind.swift

Module for handling SWIFT solar wind ensemble data.

Classes

`SWSWIFTEnsemble([data_dir])`This is a class for SWIFT ensemble data.

```
class swvo.io.solar_wind.swift.SWSWIFTEnsemble(data_dir: str | Path | None = None)
```

Bases: `object`

This is a class for SWIFT ensemble data.

Parameters

data_dir

[str | Path, optional] Data directory for the SWIFT Ensemble data. If not provided, it will be read from the environment variable

Methods

`read([start_time, end_time, propagation, ...])`Read SWIFT ensemble data for the requested period.

Raises

ValueError

Returns *ValueError* if necessary environment variable is not set.

FileNotFoundError

Returns *FileNotFoundError* if the data directory does not exist.

```
read(start_time: datetime | None = None, end_time: datetime | None = None, propagation: bool = False,
      truncate: bool = True) → list
```

Read SWIFT ensemble data for the requested period.

Parameters

start_time

[datetime] Start time of the data to read. Must be timezone-aware.

end_time

[datetime] End time of the data to read. Must be timezone-aware. If not provided, it defaults to 3 days after the start time. If *propagation* is True, it defaults to 2 days after the start time. If *propagation* is False, it defaults to 3 days after the start time.

propagation

[bool, optional] Propagate the data from L1 to near-Earth, defaults to False.

truncate

[bool, optional] If True, truncate the data to the requested period, defaults to True.

Returns

```
list[pandas.DataFrame]
```

A list of data frames containing ensemble data for the requested period.

swvo.io.utils

Functions

<code>any_nans(data)</code>	Calculate if a list of data frames contains any nans.
<code>construct_updated_data_frame(data, ...)</code>	Construct an updated data frame providing the previous data frame and the data frame of the current model call.
<code>datenum(date_input[, month, year, hour, ...])</code>	Convert a date to a MATLAB serial date number.
<code>datestr(datenum)</code>	Convert MATLAB datenum to a formatted date string.
<code>nan_percentage(data)</code>	Calculate the percentage of NaN values in the data column of data frame and log it.
<code>sw_mag_propagation(sw_data)</code>	Propagate the solar wind magnetic field to the bow shock and magnetopause.

`swvo.io.utils.any_nans(data: list[DataFrame] | DataFrame) → bool`

Calculate if a list of data frames contains any nans.

Parameters

data

[list[pd.DataFrame] | pd.DataFrame] Data frame or list of data frames to process

Returns

bool

Bool if any data frame of the list contains any nan values

`swvo.io.utils.nan_percentage(data: DataFrame) → float`

Calculate the percentage of NaN values in the data column of data frame and log it.

Parameters

data

[pd.DataFrame] The data frame to process

Returns

float

Nan percentage in the data frame

`swvo.io.utils.construct_updated_data_frame(data: list[DataFrame] | DataFrame, data_one_model: list[DataFrame] | DataFrame, model_label: str) → list[DataFrame]`

Construct an updated data frame providing the previous data frame and the data frame of the current model call.

Also adds the model label to the data frame. :param data: The data frame or list of data frames to update. :type data: list[pd.DataFrame] | pd.DataFrame :param data_one_model: The data frame or list of data frames from the current model call. :type data_one_model: list[pd.DataFrame] | pd.DataFrame :param model_label: The label of the model to add to the data frame. :type model_label: str

Returns

list[pd.DataFrame]

The updated data frame or list of data frames with the model label added.

`swvo.io.utils.datenum(date_input: datetime | int, month: int = None, year: int = None, hour: int = 0, minute: int = 0, seconds: int = 0) → float`

Convert a date to a MATLAB serial date number.

Parameters

date_input

[datetime | int] A datetime object or an integer representing the day of the month.

month

[int, optional] The month of the date. Required if date_input is an integer.

year

[int, optional] The year of the date. Required if date_input is an integer.

hour

[int, optional] The hour of the date, by default 0

minute

[int, optional] The minute of the date, by default 0

seconds

[int, optional] The seconds of the date, by default 0

Returns**float**

The MATLAB serial date number.

Raises**ValueError**

If the input is invalid, i.e., if date_input is an integer and month or year is not provided.

`swvo.io.utils.datestr(datenum: float) → str`

Convert MATLAB datenum to a formatted date string.

Parameters**datenum**

[float] The MATLAB datenum to convert.

Returns**str**

The formatted date string in the format “YYYYMMDDHHMM00”.

`swvo.io.utils.sw_mag_propagation(sw_data: DataFrame) → DataFrame`

Propagate the solar wind magnetic field to the bow shock and magnetopause.

Parameters**sw_data**

[pd.DataFrame] Data frame containing solar wind data with a ‘speed’ column.

Returns**pd.DataFrame**

Data frame with propagated solar wind data, indexed by time.

3.1 Changelog

3.2 Contributing to SWVO

Thank you for your interest in contributing to [SWVO](#). This guide explains the development workflow, coding standards, and testing requirements.

3.2.1 Getting Started

1. Fork the repository and clone it locally:

```
git clone https://github.com/<your-username>/SWVO.git  
cd SWVO
```

2. Create a feature branch:

```
git checkout -b my-feature
```

3.2.2 Code Style and Linting

We use [Ruff](#) for linting, formatting, and import sorting.

1. To check your code for issues:

```
ruff check .
```

2. To automatically apply available fixes:

```
ruff check . --fix
```

3. If Ruff cannot auto-fix an issue, fix it manually.
4. [Optional] Ruff is also integrated with `pre-commit`. To enable it:

```
pip install pre-commit  
pre-commit install
```

This ensures Ruff runs automatically on changed files before each commit.

3.2.3 Running Tests

We use pytest for testing.

1. Install dependencies:

```
pip install -e .
```

2. Run the test suite locally:

```
python -m pytest tests/io
```

All new code should include tests when applicable.

3.2.4 Pull Requests

1. Push your branch:

```
git push origin my-feature
```

2. Open a Pull Request (PR) against main.

- Clearly describe the purpose of the changes.
- Reference related issues if applicable.

CI will automatically run Ruff and the test suite on your PR.

- If Ruff fails and cannot auto-fix, you must resolve the issues before merging.
- If tests fail, update your code or test cases accordingly.

3.2.5 Commit Messages

- Use clear, descriptive commit messages.
- Example:

```
Fix handling of missing satellite data in solar wind reader
```

3.2.6 Review Process

- At least one maintainer must review and approve your PR.
- Be open to feedback and make the requested changes.

3.2.7 Additional Notes

- Keep PRs focused and small when possible.
- Document new functions or modules.

PYTHON MODULE INDEX

S

- swvo, 3
- swvo.io, 3
- swvo.io.base_file_reader, 16
- swvo.io.decorators, 16
- swvo.io.dst, 17
- swvo.io.dst.omni, 17
- swvo.io.dst.wdc, 18
- swvo.io.exceptions, 19
- swvo.io.f10_7, 20
- swvo.io.f10_7.omni, 20
- swvo.io.f10_7.swpc, 21
- swvo.io.hp, 22
- swvo.io.hp.ensemble, 23
- swvo.io.hp.gfz, 25
- swvo.io.kp, 28
- swvo.io.kp.ensemble, 28
- swvo.io.kp.niemegk, 29
- swvo.io.kp.omni, 30
- swvo.io.kp.swpc, 32
- swvo.io.omni, 33
- swvo.io.omni.omni_high_res, 34
- swvo.io.omni.omni_low_res, 35
- swvo.io.plasmasphere, 36
- swvo.io.plasmasphere.read_plasmasphere, 36
- swvo.io.plasmasphere.read_plasmasphere_combined_inputs,
37
- swvo.io.RBMDataSet, 4
- swvo.io.RBMDataSet.bin_and_interpolate_to_model_grid,
11
- swvo.io.RBMDataSet.custom_enums, 12
- swvo.io.RBMDataSet.interp_functions, 13
- swvo.io.RBMDataSet.scripts, 13
- swvo.io.RBMDataSet.scripts.create_RBSP_line_data,
13
- swvo.io.RBMDataSet.utils, 15
- swvo.io.solar_wind, 37
- swvo.io.solar_wind.ace, 37
- swvo.io.solar_wind.dscovr, 38
- swvo.io.solar_wind.omni, 40
- swvo.io.solar_wind.swift, 41
- swvo.io.utils, 42

Symbols

`__init__()` (*swvo.io.RBMDDataSet.RBMDDataSet* method), 5
`__init__()` (*swvo.io.RBMDDataSet.RBMDDataSetElPaso* method), 8
`__init__()` (*swvo.io.RBMDDataSet.RBMDDataSetManager* method), 10

A

`add_attributes_to_class_docstring()` (*in module swvo.io.decorators*), 16
`add_methods_to_class_docstring()` (*in module swvo.io.decorators*), 16
`add_note()` (*swvo.io.exceptions.ModelError* method), 19
`add_time_docs()` (*in module swvo.io.decorators*), 16
`any_nans()` (*in module swvo.io.utils*), 43

C

`cart2pol()` (*in module swvo.io.RBMDDataSet.utils*), 16
`construct_updated_data_frame()` (*in module swvo.io.utils*), 43
`create_RBSP_line_data()` (*in module swvo.io.RBMDDataSet.scripts.create_RBSP_line_data*), 13

D

`datenum()` (*in module swvo.io.utils*), 43
`datestr()` (*in module swvo.io.utils*), 44
`DebugPlotSettings` (class *in swvo.io.RBMDDataSet.bin_and_interpolate_to_model_grid*), 12
`download_and_process()` (*swvo.io.dst.omni.DSTOMNI* method), 17
`download_and_process()` (*swvo.io.dst.wdc.DSTWDC* method), 19
`download_and_process()` (*swvo.io.f10_7.omni.F107OMNI* method), 20
`download_and_process()` (*swvo.io.f10_7.swpc.F107SWPC* method), 22

`download_and_process()` (*swvo.io.hp.gfz.Hp30GFZ* method), 26
`download_and_process()` (*swvo.io.hp.gfz.Hp60GFZ* method), 27
`download_and_process()` (*swvo.io.hp.gfz.HpGFZ* method), 25
`download_and_process()` (*swvo.io.kp.niemegk.KpNiemegk* method), 30
`download_and_process()` (*swvo.io.kp.omni.KpOMNI* method), 31
`download_and_process()` (*swvo.io.kp.swpc.KpSWPC* method), 33
`download_and_process()` (*swvo.io.omni.omni_high_res.OMNIHighRes* method), 34
`download_and_process()` (*swvo.io.omni.omni_low_res.OMNILowRes* method), 36
`download_and_process()` (*swvo.io.solar_wind.ace.SWACE* method), 37
`download_and_process()` (*swvo.io.solar_wind.dscovr.DSCOVr* method), 39
`download_and_process()` (*swvo.io.solar_wind.omni.SWOMNI* method), 40
`DSCOVr` (class *in swvo.io.solar_wind.dscovr*), 38
`DSTOMNI` (class *in swvo.io.dst.omni*), 17
`DSTWDC` (class *in swvo.io.dst.wdc*), 18

E

`ElPasoMFMEEnum` (class *in swvo.io.RBMDDataSet.custom_enums*), 13

F

`F107OMNI` (class *in swvo.io.f10_7.omni*), 20
`F107SWPC` (class *in swvo.io.f10_7.swpc*), 21
`FileCadenceEnum` (class *in swvo.io.RBMDDataSet.custom_enums*), 12

FolderTypeEnum (class in *swvo.io.RBMDataset.custom_enums*), 12

G

get_file_path_any_format() (in module *swvo.io.RBMDataset.utils*), 16

H

Hp30Ensemble (class in *swvo.io.hp.ensemble*), 23

Hp30GFZ (class in *swvo.io.hp.gfz*), 26

Hp60Ensemble (class in *swvo.io.hp.ensemble*), 24

Hp60GFZ (class in *swvo.io.hp.gfz*), 26

HpEnsemble (class in *swvo.io.hp.ensemble*), 23

HpGFZ (class in *swvo.io.hp.gfz*), 25

I

instrument (*swvo.io.RBMDataset.RBMDatasetElPaso* property), 9

InstrumentEnum (class in *swvo.io.RBMDataset.custom_enums*), 13

InvV (*swvo.io.RBMDataset.RBMDatasetElPaso* property), 10

J

join_var() (in module *swvo.io.RBMDataset.utils*), 15

K

KpEnsemble (class in *swvo.io.kp.ensemble*), 28

KpNiemegk (class in *swvo.io.kp.niemegk*), 29

KpOMNI (class in *swvo.io.kp.omni*), 30

KpSWPC (class in *swvo.io.kp.swpc*), 32

L

load() (*swvo.io.RBMDataset.RBMDatasetManager* class method), 10

load_file_any_format() (in module *swvo.io.RBMDataset.utils*), 16

M

matlab2python() (in module *swvo.io.RBMDataset.utils*), 16

mfm (*swvo.io.RBMDataset.RBMDatasetElPaso* property), 9

MfmEnum (class in *swvo.io.RBMDataset.custom_enums*), 13

ModelError, 19

module

swvo, 3

swvo.io, 3

swvo.io.base_file_reader, 16

swvo.io.decorators, 16

swvo.io.dst, 17

swvo.io.dst.omni, 17

swvo.io.dst.wdc, 18

swvo.io.exceptions, 19

swvo.io.f10_7, 20

swvo.io.f10_7.omni, 20

swvo.io.f10_7.swpc, 21

swvo.io.hp, 22

swvo.io.hp.ensemble, 23

swvo.io.hp.gfz, 25

swvo.io.kp, 28

swvo.io.kp.ensemble, 28

swvo.io.kp.niemegk, 29

swvo.io.kp.omni, 30

swvo.io.kp.swpc, 32

swvo.io.omni, 33

swvo.io.omni.omni_high_res, 34

swvo.io.omni.omni_low_res, 35

swvo.io.plasmasphere, 36

swvo.io.plasmasphere.read_plasmasphere, 36

swvo.io.plasmasphere.read_plasmasphere_combined_inputs, 37

swvo.io.RBMDataset, 4

swvo.io.RBMDataset.bin_and_interpolate_to_model_grid, 11

swvo.io.RBMDataset.custom_enums, 12

swvo.io.RBMDataset.interp_functions, 13

swvo.io.RBMDataset.scripts, 13

swvo.io.RBMDataset.scripts.create_RBSP_line_data, 13

swvo.io.RBMDataset.utils, 15

swvo.io.solar_wind, 37

swvo.io.solar_wind.ace, 37

swvo.io.solar_wind.dscovr, 38

swvo.io.solar_wind.omni, 40

swvo.io.solar_wind.swift, 41

swvo.io.utils, 42

N

nan_percentage() (in module *swvo.io.utils*), 43

O

OMNIHighRes (class in *swvo.io.omni.omni_high_res*), 34

OMNILowRes (class in *swvo.io.omni.omni_low_res*), 35

P

P (*swvo.io.RBMDataset.RBMDatasetElPaso* property), 10

pol2cart() (in module *swvo.io.RBMDataset.utils*), 16

python2matlab() (in module *swvo.io.RBMDataset.utils*), 16

R

RBMDataset (class in *swvo.io.RBMDataset*), 4

RBMDataSetElPaso (class in swvo.io.RBMDataSet), 7
 RBMDataSetManager (class in swvo.io.RBMDataSet), 10
 read() (swvo.io.dst.omni.DSTOMNI method), 17
 read() (swvo.io.dst.wdc.DSTWDC method), 19
 read() (swvo.io.f10_7.omni.F107OMNI method), 20
 read() (swvo.io.f10_7.swpc.F107SWPC method), 22
 read() (swvo.io.hp.ensemble.Hp30Ensemble method), 24
 read() (swvo.io.hp.ensemble.Hp60Ensemble method), 24
 read() (swvo.io.hp.ensemble.HpEnsemble method), 23
 read() (swvo.io.hp.gfz.Hp30GFZ method), 26
 read() (swvo.io.hp.gfz.Hp60GFZ method), 27
 read() (swvo.io.hp.gfz.HpGFZ method), 25
 read() (swvo.io.kp.ensemble.KpEnsemble method), 29
 read() (swvo.io.kp.niemegk.KpNiemegk method), 30
 read() (swvo.io.kp.omni.KpOMNI method), 31
 read() (swvo.io.kp.swpc.KpSWPC method), 33
 read() (swvo.io.omni.omni_high_res.OMNIHighRes method), 34
 read() (swvo.io.omni.omni_low_res.OMNILowRes method), 36
 read() (swvo.io.solar_wind.ace.SWACE method), 38
 read() (swvo.io.solar_wind.dscovr.DSCOV method), 39
 read() (swvo.io.solar_wind.omni.SWOMNI method), 40
 read() (swvo.io.solar_wind.swift.SWSWIFTEnsemble method), 42
 read_dst_from_multiple_models() (in module swvo.io.dst), 18
 read_f107_from_multiple_models() (in module swvo.io.f10_7), 21
 read_hp_from_multiple_models() (in module swvo.io.hp), 27
 read_kp_from_multiple_models() (in module swvo.io.kp), 31
 read_solar_wind_from_multiple_models() (in module swvo.io.solar_wind), 41
 round_seconds() (in module swvo.io.RBMDataSet.utils), 16

S
 Satellite (class in swvo.io.RBMDataSet.custom_enums), 12
 satellite (swvo.io.RBMDataSet.RBMDataSetElPaso property), 9
 SatelliteEnum (class in swvo.io.RBMDataSet.custom_enums), 13
 sw_mag_propagation() (in module swvo.io.utils), 44
 SWACE (class in swvo.io.solar_wind.ace), 37
 SWOMNI (class in swvo.io.solar_wind.omni), 40
 SWSWIFTEnsemble (class in swvo.io.solar_wind.swift), 42
 swvo
 module, 3
 swvo.io
 module, 3
 swvo.io.base_file_reader
 module, 16
 swvo.io.decorators
 module, 16
 swvo.io.dst
 module, 17
 swvo.io.dst.omni
 module, 17
 swvo.io.dst.wdc
 module, 18
 swvo.io.exceptions
 module, 19
 swvo.io.f10_7
 module, 20
 swvo.io.f10_7.omni
 module, 20
 swvo.io.f10_7.swpc
 module, 21
 swvo.io.hp
 module, 22
 swvo.io.hp.ensemble
 module, 23
 swvo.io.hp.gfz
 module, 25
 swvo.io.kp
 module, 28
 swvo.io.kp.ensemble
 module, 28
 swvo.io.kp.niemegk
 module, 29
 swvo.io.kp.omni
 module, 30
 swvo.io.kp.swpc
 module, 32
 swvo.io.omni
 module, 33
 swvo.io.omni.omni_high_res
 module, 34
 swvo.io.omni.omni_low_res
 module, 35
 swvo.io.plasmasphere
 module, 36
 swvo.io.plasmasphere.read_plasmasphere
 module, 36
 swvo.io.plasmasphere.read_plasmasphere_combined_inputs
 module, 37
 swvo.io.RBMDataSet
 module, 4
 swvo.io.RBMDataSet.bin_and_interpolate_to_model_grid
 module, 11
 swvo.io.RBMDataSet.custom_enums

- module, [12](#)
- swvo.io.RBMDataSet.interp_functions
 - module, [13](#)
- swvo.io.RBMDataSet.scripts
 - module, [13](#)
- swvo.io.RBMDataSet.scripts.create_RBSP_line_data
 - module, [13](#)
- swvo.io.RBMDataSet.utils
 - module, [15](#)
- swvo.io.solar_wind
 - module, [37](#)
- swvo.io.solar_wind.ace
 - module, [37](#)
- swvo.io.solar_wind.dscovr
 - module, [38](#)
- swvo.io.solar_wind.omni
 - module, [40](#)
- swvo.io.solar_wind.swift
 - module, [41](#)
- swvo.io.utils
 - module, [42](#)

T

TargetType (*class in swvo.io.RBMDataSet.interp_functions*),
[13](#)

U

update_from_dict() (*swvo.io.RBMDataSet.RBMDataSetElPaso*
method), [9](#)

V

Variable (*class in swvo.io.RBMDataSet.custom_enums*),
[12](#)

variable_mapping (*swvo.io.RBMDataSet.RBMDataSetElPaso*
property), [9](#)

VariableEnum (*class in*
swvo.io.RBMDataSet.custom_enums), [12](#)

W

with_traceback() (*swvo.io.exceptions.ModelError*
method), [20](#)