esrin
European Space Research Institute
Largo Galileo Galilei 1
00044 Frascati
Italy
T +39 06 9418 01
F +39 06 9418 0280
www.esa.int

ESA NEOCC PYTHON INTERFACE

Prepared by C. Álvaro Arroyo Parejo

Document Type Software User Manual
Reference ESA-S2P-PD-MAN-0002
Issue/Revision 1.4.0
Date of Issue 29/10/2021
Status Final

APPROVAL

Title	ESA NEOCC Python Interface		
Issue Number	1	Revision Number	4.0
Author	C. Álvaro Arroyo Parejo	Date	29/10/2021
Approved by		Date of Approval	

CHANGE LOG

Reason for change	Issue Nr.	Revision Number	Date
Initial version	1	0	24/02/2021
Release of version 1.1	1	1	26/03/2021
Release of version 1.2	1	2	17/05/2021
Release of version 1.3	1	3	16/06/2021
Release of version 1.3.1	1	3.1	29/06/2021
Release of version 1.4	1	4	29/10/2021

CHANGE RECORD

Issue Number 1	Revision Number 4.0		
Reason for change	Date	Pages	Paragraphs(s)
New document	24/02/2021	All	
Adding ESA LaTeXtemplate	26/03/2021	All	
Update requirements	17/05/2021	5	2
Update change log tables	17/05/2021	6, 13, 15	Tables
Adding clarification within examples	17/05/2021	9, 12	Examples
Adding note for use of *help* property in data frames	17/05/2021	10	
Change *orbit_elements* to *orbital_elements*	17/05/2021	12	
Change in *impacts* example	17/05/2021	9	
Adding *close_encounter* and *impacted_objects* to *query_list* method	17/05/2021	10	
Change in requirements and version	16/06/2021	5	
Change name *neocc* to *core*	16/06/2021	5	Section
Update change log tables	16/06/2021	6, 14, 17	Tables
Adding section *Library Examples*	16/06/2021	24-28	New Section
Adding section *ESANEOCC Change Log*	16/06/2021	28-29	New Section
Adding version 1.3.1 to *ESANEOCC Change Log*	29/06/2021	28	Bug fixed
Update change log tables	29/06/2021	7, 15, 18	Tables
Update requirements	29/10/2021	6	
Adding catalogues of NEAs to *query_list*	29/10/2021	8	
Remove note about *physical_properties*	29/10/2021	13	
Update *orbit_properties* example adding *orb_type* as attribute	29/10/2021	14	
Remove function *get_dec_year*	29/10/2021	13	
Update *get_list_url* parameters	29/10/2021	16	
Added definition of *orb_type* parameter	29/10/2021	22	
Update change log tables	29/10/2021	7, 15, 18	Tables
Adding version 1.4 to *ESANEOCC Change Log*	29/10/2021	30-31	Changes and Bug fixed

DISTRIBUTION

Name/Organisational Unit
Feel Free To Distribute (ESA)

Table of Contents:

1. Introduction 5
2. Installation 5
3. Requirements 6
4. Modules 6
5. Library examples 25
6. ESANEOCC Change Log 30

This is the documentation for the ESA NEOCC Portal Python interface library.

1. INTRODUCTION

ESA NEOCC Portal Python interface library makes the data that ESA NEOCC provides easily accessible through a Python program.

The main functionality of this library is to allow a programmer to easily retrieve:

- All the NEAs
- Other data that the NEOCC provides (risk list, close approach list, etc.)
- All basic and advanced information regarding a NEA
- An ephemeris service for NEAs

2. INSTALLATION

The library is contained in ESANEOCC folder. In order to install the library:

1. Navigate to the proper directory where the *setup.py* is located.
2. The installation is doable through *pip install* command:

```
$ pip install .
```

Note: Consider installing **ESANEOCC** library into a virtualenv. This will avoid problems with previous installed dependencies.

The previous installation will install the library and its dependencies, but the dependencies will not be updated in case they are previously installed. In order to asssure that the packages version is the one determined in the **Requirements** the following command must be written:

```
$ pip install -r requirements.txt
```

This can be done in one command line:

```
$ pip install . && pip install -r requirements.txt
```

Warning! The previous command will force to install the package version of the requirements. This will upgrade/downgrade the version of any previous installed package that **ESANEOCC** library depends on.

Another installation method that will install the library and will update the dependencies is the following:

```
$ pip install . -upgrade-strategy eager
```

In this case, dependencies are upgraded regardless of whether the currently installed version satisfies the requirements of the upgraded package(s).

If you want to make sure none of your existing dependencies get upgraded, you can also do:

```
$ pip install . -no-deps
```

Note that, in the latter case, it is possible that some library functionalities will not work if the dependencies do not satisfy the **Requirements**.

3. REQUIREMENTS

ESA NEOCC Portal Python Interface Library works with Python 3.

The following packages are required for the library installation & use:

- astropy = 4.3.1
- beautifulsoup4 = 4.10.0
- lxml = 4.6.3
- pandas = 1.3.4
- parse = 1.19.0
- requests = 2.26.0
- scipy = 1.7.1

For tests the following packages are required:

- pytest = 6.2.1
- astroquery = 0.4.3

4. MODULES

ESANEOCC.core

Main module from ESA NEOCCS library. This module contains the two main methods of the library: *query_list* and *query_object*. The information is obtained from ESA Near-Earth Object Coordination Centres (NEOCC) web portal: https://neo.ssa.esa.int/.

- Project: NEOCC portal Python interface
- Property: European Space Agency (ESA)
- Developed by: Elecnor Deimos

- Author: C. Álvaro Arroyo Parejo
- Issue: 1.4.0
- Date: 29-10-2021
- Purpose: Main module which gets NEAs data from https://neo.ssa.esa.int/
- Module: core.py
- History:

Version	Date	Change History
1.0	26-02-2021	Initial version
1.1	24-03-2021	New docstrings
1.2	17-05-2021	Adding new docstrings for *help* property in dataframes and *tab* specification for obtaining attributes. For orbit properties *orbit_elements* changes to *orbital_elements*. Adding impacted objects lists Minor typos changes.
1.3	16-06-2021	Renamed module from *neocc* to *core* specification for obtaining attributes. Define methods as static
1.3.1	29-06-2021	No changes
1.4	29-10-2021	Adding new docstrings. Change method for obtaining physical properties

class ESANEOCC.core.**ESAneoccClass**

Class to init ESA NEOCC Python interface library.

static query_list(*list_name*)

Get requested list data from ESA NEOCC.

Different lists that can be requested are:

- All NEA list: *nea_list*
- Updated NEA list: *updated_nea*
- Monthly computation date: *monthly_update*
- Risk list (normal): *risk_list*
- Risk list (special): *risk_list_special*
- Close approaches (upcoming): *close_approaches_upcoming*

- Close approaches (recent): *close_approaches_recent*
- Priority list (normal): *priority_list*
- Priority list (faint): *priority_list_faint*
- Close encounter list: *close_encounter*
- Impacted objects: *impacted_objects*
- Catalogue of NEAs (current date): *neo_catalogue_current*
- Catalogue of NEAs (middle arc): *neo_catalogue_middle*

These lists are referenced in https://neo.ssa.esa.int/computer-access

Parameters **`list_name`** (*str*) – Name of the requested list. Valid names are: *nea_list, risk_list, risk_list_special, close_approaches_upcoming, close_approaches_recent, priority_list, priority_list_faint, close_encounter, impacted_objects, neo_catalogue_current and neo_catalogue_middle*.

Returns neocc_lst – Data Frame which contains the information of the requested list

Return type *pandas.Series* or *pandas.DataFrame*

Examples

NEA list: The output of this list is a *pandas.Series* which contains the list of all NEAs currently considered in the NEOCC system.

```
>>> from ESANEOCC import neocc
>>> list_data = neocc.query_list(list_name='nea_list')
>>> list_data
0                433 Eros
1              719 Albert
2              887 Alinda
3            1036 Ganymed
4               1221 Amor
             ...
25191             2021DY
25192            2021DY1
25193             2021DZ
25194            2021DZ1
25195            6344P-L
Name: 0, Length: 25196, dtype: object
```

Each asteroid can be accessed using its index. This information can be used as input for *query_object* method.

```
>>> list_data[4]
'1221 Amor'
```

Other lists: The output of this list is a *pandas.DataFrame* which contains the information of the requested list.

```
>>> from ESANEOCC import neocc
>>> list_data = neocc.query_list(list_name='close_approaches_upcoming')
>>> list_data
         Object Name         Date   ...   Rel. vel in km/s
0             2021DE  2021.156164  ...               26.0
1             2021DM  2021.158904  ...               10.2
2             2011DW  2021.161644  ...               13.6
3           2011EH17  2021.161644  ...               16.8
4            2016DV1  2021.164384  ...               18.6
..               ...          ...  ...                ...
141           2020DF  2022.120548  ...                8.6
142          2018CW2  2022.131507  ...               10.8
143          2020CX1  2022.131507  ...                8.2
144  455176 1999VF22  2022.142466  ...               25.1
145          2017CX1  2022.145205  ...                5.0
[146 rows x 10 columns]
```

The information of the columns can be accessed through (see pandas for further information about data access):

```
>>> list_data['Object Name']
0               2021DE
1               2021DM
2               2011DW
3             2011EH17
4              2016DV1
            ...
141             2020DF
142            2018CW2
143            2020CX1
144    455176 1999VF22
145            2017CX1
Name: Object Name, Length: 146, dtype: object
```

And the information of the rows can be accessed using:

```
>>> list_data.iloc[2]
Object Name              2011DW
Date                   2021.16
Miss Distance in km    5333057
Miss Distance in au   0.035649
Miss Distance in LD     13.874
Diameter in m               90
*=Yes                        *
H                         22.9
```

(continues on next page)

(continued from previous page)

```
Max Bright                  16.4
Rel. vel in km/s            13.6
Name: 2, dtype: object
```

Note: If the contents request fails the following message will be printed:

Initial attempt to obtain list failed. Reattempting

Then a second request will be automatically sent to the NEOCC portal.

---

static query_object(*name*, *tab*, ***kwargs*)

Get requested object data from ESA NEOCC.

Parameters

- **name** (*str*) – Name of the requested object
- **tab** (*str*) – Name of the request tab. Valid names are: summary, orbit_properties, physical_properties, observations, ephemerides, close_approaches and impacts.
- ****kwargs** (*str*) – Tabs orbit_properties and ephemerides tabs required additional arguments to work:
  - *orbit_properties*: the required additional arguments are:
    - *orbital_elements* : str (keplerian or equinoctial)
    - *orbit_epoch* : str (present or middle)
  - *ephemerides*: the required additional arguments are:
    - *observatory* : str (observatory code, e.g. 500, J04, etc.)
    - *start* : str (start date in YYYY-MM-DD HH:MM)
    - *stop* : str (end date in YYYY-MM-DD HH:MM)
    - *step* : str (time step, e.g. 2, 15, etc.)
    - *step_unit* : str (e.g. days, minutes, etc.)

Returns neocc_obj – Object data which contains different attributes depending on the tab selected.

Return type object

Examples

Impacts, Physical Properties and Observations: This example tries to summarize how to access the data of this tabs and how to use it. Note that this classes only require as inputs the name of the object and the requested tab.

The information can be obtained introducing directly the name of the object, but it can be also added from the output of a *query_list* search:

```
>>> from ESANEOCC import neocc
>>> ast_impacts = neocc.query_object(name='1979XB', tab='impacts')
```

or

```
>>> nea_list = neocc.query_list(list_name='nea_list')
>>> nea_list[2921]
'1979XB'
>>> ast_impacts = neocc.query_object(name=nea_list[2921], tab='impacts
↪')
```

or

```
>>> risk_list = neocc.query_list(list_name='risk_list')
>>> risk_list['Object Name'][1]
'1979XB'
>>> ast_impacts = neocc.query_object(name=risk_list['Object Name'][1],␣
↪tab='impacts')
```

The output provides an object with the different attributes:

```
>>> ast_impacts.<tab>
ast_impacts.additional_note       ast_impacts.impacts
ast_impacts.arc_end               ast_impacts.info
ast_impacts.arc_start             ast_impacts.observation_accepted
ast_impacts.computation           ast_impacts.observation_rejected
```

By adding the attribute its information can be accessed:

```
>>> ast_impacts.impacts
             date          MJD    sigma      ...   Exp. Energy in MT    PS␣
↪ TS
0  2056-12-12.902   72344.902    0.255      ...            0.011500 -3.22␣
↪  0
1  2065-12-16.462   75635.463   -1.110      ...            0.000090 -5.36␣
↪  0
2  2086-12-16.663   83305.664   -1.101      ...            0.002390 -4.03␣
↪  0
3  2101-12-14.203   88781.204   -0.384      ...            0.000131 -5.36␣
↪  0
```

(continues on next page)

(continued from previous page)

```
4  2105-12-12.764   90240.765   1.003     ...           0.000574 -4.75
↪  0
5  2113-12-14.753   93164.753   0.706     ...           0.018500 -3.25
↪  0
6  2113-12-14.756   93164.756   0.708     ...           0.000163 -5.30
↪  0
7  2117-12-15.496   94626.496  -1.316     ...           0.000069 -5.68
↪  0
[8 rows x 11 columns]
```

Note: Most of the dataframes of the object tabs contain the help property which contains information about the fields of the dataframe.

```
>>> print(ast_impacts.impacts.help)
Data frame with possible impacts information:
-Date: date for the potential impact in YYYY-MM-DD.ddd format
-MJD: Modified Julian Day for the potential impact
-sigma: approximate location along the Line Of Variation (LOV)
 in sigma space
-sigimp: The lateral distance in sigma-space from the LOV to
 the Earth surface. A zero implies that the LOV passes through
 the Earth-dist: Minimum Distance in Earth radii. The lateral
 distance from the LOV to the centre of the Earth
-width: one-sigma semi-width of the Target Plane confidence
 region in Earth radii
-stretch: Stretching factor. It indicates how much the
 confidence region at the epoch has been stretched by the time
 of the approach. This is a close cousin of the Lyapounov
 exponent. Units are in Earth radii divided by sigma (RE/sig)
-p_RE: probability of Earth Impact (IP)
-Exp. Energy in MT: Expected energy. It is the product of the
 impact energy and the impact probability
-PS: Palermo Scale
-TS: Torino Scale
```

Another example is shown to obtain the physical properties:

```
>>> from ESANEOCC import neocc
>>> properties = neocc.query_object(name='433', tab='physical_
↪properties')
```

Again, the output provides an object with different attributes:

```
>>> properties.<tab>
```

(continues on next page)

(continued from previous page)

```
properties.physical_properties  properties.sources
>>> properties.physical_properties
                   Property     Values Unit Source
0           Rotation Period       5.27    h    [4]
1                   Quality          4    -    [4]
2                 Amplitude  0.04-1.49  mag    [4]
3        Rotation Direction        PRO    -    [1]
4              Spinvector L         16  deg    [1]
5              Spinvector B          9  deg    [1]
6                  Taxonomy         Sq    -    [2]
7            Taxonomy (all)          S    -    [3]
8    Absolute Magnitude (H)      10.31  mag    [5]
9       Slope Parameter (G)     0.46**  mag    [6]
10                  Albedo       0.24    -    [9]
11                Diameter      23300    m   [10]
12  Color Index Information       0.39  R-I   [11]
13                Sightings   Visual S    -   [13]
```

Note: For the case of tab Observations there are objects which contain Roving observer and satellite observations. In the original requested data the information of these observations produces two lines of data, where the second line does not fit the structure of the data frame (https://www.minorplanetcenter.org/iau/info/OpticalObs.html). In order to solve this problem those second lines have been extracted in another attribute (e.g. sat_observations or roving_observations) to make the data more readable.

Since this information can be requested in pairs, i.e. it is needed to access both lines of data, this can be made using the date of the observations which will be the same for both attributes:

```
>>> ast_observations = neocc.query_object(name='99942',
tab='observations')
>>> sat_obs = ast_observations.sat_observations
>>> sat_obs
    Design.  K  T  N  YYYY  MM  DD.dddddd  ... Obs Code
0     99942  S  s     2020  12   18.97667  ...      C51
1     99942  S  s     2020  12   19.10732  ...      C51
...     ...           ...            ...  ...      ...
10    99942  S  s     2021   1   16.92315  ...      C53
11    99942  S  s     2021   1   19.36233  ...      C53
12    99942  S  s     2021   1   19.36927  ...      C53
>>> opt_obs = ast_ast_observations.optical_observations
>>> opt_obs.loc[opt_obs['DD.dddddd'] == sat_obs['DD.dddddd'][0]]
        Design.  K  T N  YYYY  MM  ...  Obs Code   Chi  A  M
4582    99942    S  S    2020  12  ...       C51  1.13  1  1
[1 rows x 33 columns]
```

Close Approaches: This example corresponds to the class close approaches. As for the previous example, the information can be obtained by directly introducing the name of the object or from a previous *query_list* search.

In this particular case, there are no attributes and the data obtained is a DataFrame which contains the information for close approaches:

```
>>> close_appr = neocc.query_object(name='99942', tab='close_approaches
 ↪')
>>> close_appr
    BODY     CALENDAR-TIME  ...          WIDTH  PROBABILITY
0   EARTH  1957/04/01.13908  ...  1.318000e-08        1.000
1   EARTH  1964/10/24.90646  ...  1.119000e-08        1.000
2   EARTH  1965/02/11.51118  ...  4.004000e-09        1.000
...   ...               ...  ...           ...          ...
16  EARTH  2080/05/09.23878  ...  1.206000e-06        0.821
17  EARTH  2087/04/07.54747  ...  1.254000e-08        0.327
[18 rows x 10 columns]
```

Orbit Properties: In order to access the orbit properties information, it is necessary to provide two additional inputs to *query_object* method: **orbital_elements** and **orbit_epoch**.

It is mandatory to write these two parameters as: *orbit_epoch=* to make the library works.

```
>>> ast_orbit_prop = neocc.query_object(name='99942',
tab='orbit_properties',orbital_elements='keplerian', orbit_epoch=
 ↪'present')
>>> ast_orbit_prop.<tab>
ast_orbit_prop.anode           ast_orbit_prop.ngr
ast_orbit_prop.aphelion        ast_orbit_prop.orb_type
ast_orbit_prop.cor             ast_orbit_prop.perihelion
ast_orbit_prop.cov             ast_orbit_prop.period
ast_orbit_prop.dnode           ast_orbit_prop.pha
ast_orbit_prop.epoch           ast_orbit_prop.rectype
ast_orbit_prop.form            ast_orbit_prop.refsys
ast_orbit_prop.kep             ast_orbit_prop.rms
ast_orbit_prop.lsp             ast_orbit_prop.u_par
ast_orbit_prop.mag             ast_orbit_prop.vinfty
ast_orbit_prop.moid
```

Ephemerides: In order to access ephemerides information, it is necessary to provide five additional inputs to *query_object* method: **observatory**, **start**, **stop**, **step** and **step_unit***.

It is mandatory to write these five parameters as: *observatory=* to make the library works.

```
>>> ast_ephemerides = neocc.query_object(name='99942',
tab='ephemerides', observatory='500', start='2019-05-08 01:30',
```

(continues on next page)

(continued from previous page)

```
stop='2019-05-23 01:30', step='1', step_unit='days')
>>> ast_ephemerides.
ast_ephemerides.ephemerides  ast_ephemerides.tinit
ast_ephemerides.observatory  ast_ephemerides.tstep
ast_ephemerides.tfinal
```

ESANEOCC.lists

This module contains all the methods required to request the list data, obtain it from the ESA NEOCC portal and parse it to show it properly.

- Project: NEOCC portal Python interface
- Property: European Space Agency (ESA)
- Developed by: Elecnor Deimos
- Author: C. Álvaro Arroyo Parejo
- Issue: 1.4
- Date: 29-10-2021
- Purpose: Module which request and parse list data from ESA NEOCC
- Module: lists.py
- History:

Version	Date	Change History
1.0	26-02-2021	Initial version
1.1	24-03-2021	New docstrings
1.2	17-05-2021	Adding new docstrings for *help* property in dataframes. Adding timeout of 90 seconds. Adding *parse_impacted* function for new list
1.3	16-06-2021	URL and Timeout from configuration file for astroquery implementation. Change dateformat to datetime ISO format.
1.3.1	29-06-2021	Hotfix in Risk list (TS and Velocity)
1.4	29-10-2021	Adding Catalogue of NEAS (current date and middle arc). Update docstrings.

ESANEOCC.lists.**get_list_data**(*url*, *list_name*)

Get requested parsed list from url.

Parameters

- **list_name** (*str*) – Name of the requested list.
- **url** (*str*) – URL of the requested list.

Returns **neocc_lst** – Data frame which contains the data of the requested list.

Return type *pandas.Series* or *pandas.DataDrame*

ESANEOCC.lists.**get_list_url**(*list_name*)

Get url from requested list name.

Parameters **list_name** (*str*) – Name of the requested list. Valid names are: *nea_list, risk_list, risk_list_special, close_appr_upcoming, close_appr_recent, priority_list, priority_list_faint, close_encounter, impacted_objects, neo_catalogue_current and neo_catalogue_middle*.

Returns **url** – Final URL string.

Return type str

Raises **KeyError** – If the requested list_name is not in the dictionary

ESANEOCC.lists.**parse_clo**(*data_byte_d*)

Parse and arrange close approaches lists.

Parameters **data_byte_d** (*object*) – Decoded StringIO object.

Returns **neocc_lst** – Data frame with close approaches list data parsed.

Return type *pandas.Series* or *pandas.DataFrame*

ESANEOCC.lists.**parse_encounter**(*data_byte_d*)

Parse and arrange close encounter list.

Parameters **data_byte_d** (*object*) – Decoded StringIO object.

Returns **neocc_lst** – Data frame with close encounter list data parsed.

Return type *pandas.Series* or *pandas.DataFrame*

ESANEOCC.lists.**parse_impacted**(*data_byte_d*)

Parse and arrange close encounter list.

Parameters **data_byte_d** (*object*) – Decoded StringIO object.

Returns **neocc_lst** – Data frame with impacted objects list data parsed.

Return type *pandas.Series* or *pandas.DataFrame*

ESANEOCC.lists.**parse_list**(*list_name*, *data_byte_d*)

Switch function to select parse method.

Parameters

- **list_name** (*str*) – Name of the requested list.

- **data_byte_d** (*object*) – Decoded StringIO object.

Returns **neocc_lst** – Data frame with data from the list parsed.

Return type *pandas.Series* or *pandas.DataFrame*

ESANEOCC.lists.**parse_nea**(*data_byte_d*)
Parse and arrange all NEA list.

Parameters **data_byte_d** (*object*) – Decoded StringIO object.

Returns **neocc_lst** – Data frame with NEA list data parsed.

Return type *pandas.Series* or *pandas.DataFrame*

ESANEOCC.lists.**parse_neo_catalogue**(*data_byte_d*)
Parse neo catalogues (current or middle arc) lists.

Parameters **data_byte_d** (*object*) – Decoded StringIO object.

Returns **neocc_lst** – Data frame with catalogues of NEAs list data parsed.

Return type *pandas.Series* or *pandas.DataFrame*

ESANEOCC.lists.**parse_pri**(*data_byte_d*)
Parse and arrange priority lists.

Parameters **data_byte_d** (*object*) – Decoded StringIO object.

Returns **neocc_lst** – Data frame with priority list data parsed.

Return type *pandas.Series* or *pandas.DataFrame*

ESANEOCC.lists.**parse_risk**(*data_byte_d*)
Parse and arrange risk lists.

Parameters **data_byte_d** (*object*) – Decoded StringIO object.

Returns **neocc_lst** – Data frame with risk list data parsed.

Return type *pandas.Series* or *pandas.DataFrame*

ESANEOCC.tabs

This module contains all the methods required to request the data from a particular object, obtain it from the ESA NEOCC portal and parse it to show it properly. The information of the object is shows in the ESA NEOCC in different tabs that correspond to the different classes within this module.

- Project: NEOCC portal Python interface
- Property: European Space Agency (ESA)
- Developed by: Elecnor Deimos
- Author: C. Álvaro Arroyo Parejo
- Issue: 1.4

- Date: 29-10-2021
- Purpose: Module which request and parse list data from ESA NEOCC
- Module: tabs.py
- History:

Version	Date	Change History
1.0	26-02-2021	Initial version
1.1 1.2	24-03-2021 17-05-2021	Physical properties functionality added Adding *help* property for dataframes. Parsing of diameter property in summary and *physical_properties* has been modified to add robustness. In *physical_properties* the parsing of properties has been modified to include cases with more information. Adding timeout of 90 seconds.
1.3	16-06-2021	URLs and timeout from configuration file for astroquery implementation Change time format to datetime ISO format Change to correct types in attributes (e.g., matrices, etc.) Change ephemerides skipfooter to fix bug Change *get_matrix from orbit_properties* for objects with 2 non-gravitational parameters
1.3.1	29-06-2021	No changes
1.4	29-10-2021	Tab physical_properties has been recoded to parse the information through a request in the portal instead of parsing the html. Get URL function now contains the file extension for physical properties. Parsing of ephemerides has been change to adapt new format. orb_type attribute added in tab orbit_properties. Bug fix in tab observations. Adding redundancy for tab summary parsing.

class ESANEOCC.tabs.**AsteroidObservations**

This class contains information of asteroid observations.

version

File version.

Type int

errmod
Error model for the data.

Type str

rmsast
Root Mean Square for asteroid observations.

Type float

rmsmag
Root Mean Square for magnitude.

Type float

optical_observations
Data frame which contains optical observations (without roving observer and satellite observation).

Type pandas.DataFrame

radar_observations
Data structure which contains radar observations.

Type pandas.DataFrame

roving_observations
Data structure which contains roving observer observations.

Type pandas.DataFrame

sat_observations
Data structure which contains satellite observations.

Type pandas.DataFrame

class ESANEOCC.tabs.**CloseApproaches**
This class contains information of object close approaches.

static clo_appr_parser(*data_obj*, *name*)
Parse and arrange the close approaches data.

Parameters **data_obj** (*object*) – Object in byte format.

Returns **df_close_appr** – Data frame with the close approaches information.

Return type pandas.DataFrame

Raises **ValueError** – If file is empty.

class ESANEOCC.tabs.**Ephemerides**
This class contains information of object ephemerides.

observatory
Name of the observatory from which ephemerides are obtained.

Type str

tinit
Start date from which ephemerides are obtained.

Type str

tfinal
End date from which ephemerides are obtained.

Type str

tstep
Time step and time unit used during ephemerides calculation.

Type str

ephemerides
Data frame which contains the information of the object ephemerides

Type pandas.DataFrame

class ESANEOCC.tabs.**EquinoctialOrbitProperties**
This class contains information of equinoctial asteroid orbit properties. This class inherits the attributes from OrbitProperties.

equinoctial
Data frame which contains the equinoctial elements information.

Type pandas.DataFrame

rms
Root Mean Square for equinoctial elements.

Type DataFrame

eig
Eigenvalues for the covariance matrix.

Type pandas.DataFrame

wea
Eigenvector corresponding to the largest eigenvalue.

Type pandas.DataFrame

cov
Covariance matrix for equinoctial elements.

Type pandas.DataFrame

nor
Normalization matrix for equinoctial elements.

Type pandas.DataFrame

class ESANEOCC.tabs.**Impacts**

This class contains information of object possible impacts.

impacts

Data frame where are listed all the possible impactors.

Type pandas.DataFrame

arc_start

Starting date for optical observations.

Type str

arc_end

End date for optical observations.

Type str

observations_accepted

Total number of observations subtracting rejected observations.

Type int

observations_rejected

Number of observations rejected.

Type int

computation

Date of computation (in format YYYYMMDD MJD TimeSys)

Type str

info

Information from the footer of the requested file.

Type str

additional_note

Additional information. Some objects (e.g. 99942 Apophis) have an additional note after the main footer.

Type str

class ESANEOCC.tabs.**KeplerianOrbitProperties**

This class contains information of keplerian asteroid orbit properties. This class inherits the attributes from OrbitProperties.

kep

Data frame which contains the keplerian elements information.

Type pandas.DataFrame

perihelion
Orbit perihelion in au.

Type int

aphelion
Orbit aphelion in au.

Type int

anode
Ascending node-Earth separation in au.

Type int

dnode
Descending node-Earth separation in au.

Type int

moid
Minimum Orbit Intersection distance in au.

Type int

period
Orbit period in days.

Type int

pha
Potential hazardous asteroid classification.

Type string

vinfty
Infinite velocity.

Type int

u_par
Uncertainty parameter as defined by MPC.

Type int

orb_type
Type of orbit.

Type int

rms
Root mean square for keplerian elements

Type pandas.DataFrame

cov
Covariance matrix for keplerian elements

Type pandas.DataFrame

cor
Correlation matrix for keplerian elements

Type pandas.DataFrame

class ESANEOCC.tabs.**OrbitProperties**
This class contains information of asteroid orbit properties.

form
File format.

Type str

rectype
Record type.

Type str

refsys
Default reference system.

Type str

epoch
Epoch in MJD format.

Type str

mag
Data frame which contains magnitude values.

Type pandas.DataFrame

lsp
Data structure with information about non-gravitational parameters (model, numer of parameters, dimension, etc.).

Type pandas.DataFrame

ngr
Data frame which contains non-gravitational parameters.

Type pandas.DataFrame

class ESANEOCC.tabs.**PhysicalProperties**
This class contains information of asteroid physical properties

physical_properties
Data structure containing property, value, units and source from the complete set of physical properties

Type DataFrame

sources
Data structure containing source number, name and additional information

Type DataFrame

Raises **ValueError** – If the name of the object is not found

class ESANEOCC.tabs.**Summary**
This class contains the information from the Summary tab.

physical_properties
Data frame which contains the information of the object physical properties, their value and their units.

Type pandas.DataFrame

discovery_date
Provides the object discovery date

Type str

observatory
Provides the name of the observatory where object was discovered

Type str

ESANEOCC.tabs.**get_indexes**(*dfobj*, *value*)
Get a list with location index of a value or string in the DataFrame requested.

Parameters

- **dfobj** (*pandas.DataFrame*) – Data frame where the value will be searched.
- **value** (*str, int, float*) – String, integer or float to be searched.

Returns **listofpos** – List which contains the location of the value in the Data frame. The first elements will correspond to the index and the second element to the columns

Return type list

ESANEOCC.tabs.**get_object_data**(*url*)
Get object in byte format from requested url.

Parameters **url** (*str*) – URL of the requested data.

Returns **data_obj** – Object in byte format.

Return type object

ESANEOCC.tabs.**get_object_url**(*name*, *tab*, ***kwargs*)
Get url from requested object and tab name.

Parameters

- **name** (*str*) – Name of the requested object.
- **tab** (*str*) – Name of the request tab. Valid names are: *summary, orbit_properties, physical_properties, observations, ephemerides, close_approaches and impacts*.
- ****kwargs** (*str*) – orbit_properties and ephemerides tabs required additional arguments to work:
  - *orbit_properties*: the required additional arguments are:
    - *orbital_elements* : str (keplerian or equinoctial)
    - *orbit_epoch* : str (present or middle)
  - *ephemerides*: the required additional arguments are:
    - *observatory* : str (observatory code, e.g. 500, J04, etc.)
    - *start* : str (start date in YYYY-MM-DD HH:MM)
    - *stop* : str (end date in YYYY-MM-DD HH:MM)
    - *step* : str (time step, e.g. 2, 15, etc.)
    - *step_unit* : str (e.g. days, minutes, etc.)

Returns **url** – Final url from which data is requested.

Return type string

Raises

- **KeyError** – If the requested tab is not in the dictionary.
- **ValueError** – If the elements requested are not valid.

5. LIBRARY EXAMPLES

How to export data

To JSON

Most of the data obtained from ESANEOCC Python Interface is collected as *pandas.Series* or *pandas.DataFrame* and, therefore it can be easily converted to JSON format. Here is a template that you may use in Python to export pandas DataFrame to JSON:

```
>>> df.to_json(r'Path to store the exported JSON file/ FileName.json')
```

The complete use of this function can be found in pandas.DataFrame.to_json, where different examples are shown. It also shows how to obtain the different types of JSON files.

```
>>> from ESANEOCC import neocc
>>> from astropy.table import Table
>>> ast = neocc.query_object(name='99942', tab='physical_properties')
>>> ast.physical_properties
                  Property     Values Unit Source
0           Rotation Period      30.56    h    [4]
1                   Quality          3    -    [4]
2                 Amplitude        1.0  mag    [4]
3        Rotation Direction      RETRO    -    [1]
4              Spinvector L        250  deg    [1]
5              Spinvector B    -7.50E1  deg    [1]
6                  Taxonomy       S/Sq    -    [2]
7            Taxonomy (all)   Sq,Scomp    -    [3]
8    Absolute Magnitude (H)      19.09  mag    [5]
9       Slope Parameter (G)       0.24  mag    [1]
10                  Albedo      0.285    -    [8]
11                Diameter        375    m    [9]
12  Color Index Information      0.362  R-I   [10]
13                Sightings    Radar R    -   [11]
>>> ast_json = ast.physical_properties.to_json(orient='split')
>>> parsed = json.loads(ast_json)
>>> print(json.dumps(parsed, indent= 4))
{
    'columns': [
        'Property',
        'Values',
        'Unit',
        'Source'
    ],
    'index': [
        0,
        1,
        2,
        3,
        4,
        5,
        6,
        7,
        8,
        9,
        10,
        11,
        12,
        13
    ],
    'data': [
        [
```

(continues on next page)

(continued from previous page)

```
            'Rotation Period',
            '30.56',
            'h',
            '[4]'
        ],
        [
            'Quality',
            '3',
            '-',
            '[4]'
        ],
        [
            'Amplitude',
            '1.0',
            'mag',
            '[4]'
        ],
        [
            'Rotation Direction',
            'RETRO',
            '-',
            '[1]'
        ],
        [
            'Spinvector L',
            '250',
            'deg',
            '[1]'
        ],
        [
            'Spinvector B',
            '-7.50E1',
            'deg',
            '[1]'
        ],
        [
            'Taxonomy',
            'S/Sq',
            '-',
            '[2]'
        ],
        [
            'Taxonomy (all)',
            'Sq,Scomp',
            '-',
            '[3]'
```

(continues on next page)

(continued from previous page)


```
            ],
            [
                'Absolute Magnitude (H)',
                '19.09',
                'mag',
                '[5]'
            ],
            [
                'Slope Parameter (G)',
                '0.24',
                'mag',
                '[1]'
            ],
            [
                'Albedo',
                '0.285',
                '-',
                '[8]'
            ],
            [
                'Diameter',
                '375',
                'm',
                '[9]'
            ],
            [
                'Color Index Information',
                '0.362',
                'R-I',
                '[10]'
            ],
            [
                'Sightings',
                'Radar R',
                '-',
                '[11]'
            ]
        ]
    }
```


```
>>> ast_json = ast.physical_properties.to_json(r'Path to store the␣
↪exported JSON file/ ast.json', orient='split')
```

To Tables (VO Tables)

Virtual Observatory (VO) tables are a new format developed by the International Virtual Observatory Alliance to store one or more tables. This format is included within ATpy library. However, most of ATpys functionalities has now been incorporated into Astropy library and the developers recommended to use the Astropy Tables.

Astropy documentation details how to interface with the Pandas library, i.e., how to convert data in pandas.Series or pandas.DataFrame formats into Astropy Tables and vice versa.

```
>>> from ESANEOCC import neocc
>>> from astropy.table import Table
>>> ast = neocc.query_object(name='433', tab='physical_properties')
>>> ast.physical_properties
                  Property     Values Unit Source
0           Rotation Period       5.27    h    [4]
1                   Quality          4    -    [4]
2                 Amplitude  0.04-1.49  mag    [4]
3        Rotation Direction        PRO    -    [1]
4              Spinvector L         16  deg    [1]
5              Spinvector B          9  deg    [1]
6                  Taxonomy         Sq    -    [2]
7             Taxonomy (all)          S    -    [3]
8    Absolute Magnitude (H)      10.31  mag    [5]
9       Slope Parameter (G)     0.46**  mag    [6]
10                   Albedo       0.24    -    [9]
11                 Diameter      23300    m   [10]
12  Color Index Information       0.39  R-I   [11]
13                Sightings   Visual S    -   [13]
>>> ast_astropy = Table.from_pandas(ast.physical_properties)
```

```
>>> ast_astropy
<Table length=14>
           Property            Values Unit Source
            str23               str8  str3  str4
-------------------------  --------- ---- ------
           Rotation Period       5.27    h    [4]
                   Quality          4    -    [4]
                 Amplitude  0.04-1.49  mag    [4]
        Rotation Direction        PRO    -    [1]
              Spinvector L         16  deg    [1]
              Spinvector B          9  deg    [1]
                  Taxonomy         Sq    -    [2]
             Taxonomy (all)          S    -    [3]
    Absolute Magnitude (H)      10.31  mag    [5]
       Slope Parameter (G)     0.46**  mag    [6]
                    Albedo       0.24    -    [9]
                  Diameter      23300    m   [10]
   Color Index Information       0.39  R-I   [11]
                 Sightings   Visual S    -   [13]
```

Visit Interfacing with the Pandas Package for further information.

6. ESANEOCC CHANGE LOG

Version 1.4

Changes

- The type of the orbit is now obtained as attribute in tab *orbit_properties*.
- The format of the ephemerides data has been modified to adapt to the change of format of the ephemerides provided by NEOCC portal. This modification includes new parameters for the columns and also changes the precion of the Magnitude values up to two decimal digits.
- Tab *physical_properties* now contains all the values (if exist) for the different properties.
- There are two new lists implemented: Catalogue of NEAs (current date) and Catalogue of NEAs (middle arc).
- Parsing of tab *summary* is more robust.

Bug Fixes

- Fixed tab *observations* which failed to process the information of some asteroids.

Version 1.3.1

Bug Fixes

- Fixed bug where *risk_list* and *risk_list_special* were not displaying Torino Scale and Velocity.

Version 1.3

Changes

- astropy library has been added as required package.
- neocc.py module has been renamed to core.py in order to be consistent with Astroquery.
- core.py has been modified in order to be consistent with Astroquery (main class and static methods).
- Dates/time columns or data have been converted to datetime ISO format.
- Abbreviations contain now complete expressions (e.g., close_appr_upcoming to close approaches_upcoming)
- The documentation explains how to obtain JSON and Table format from data retrieved from the library.
- Time performance improvement in obtaining physical properties.

Bug Fixes

- Fixed two-points ephemerides generation fails
- Fixed physical properties generation for objects with Area-to-mass ratio and Yarkovsky parameter.
- Fixed orbit properties generation for objects with Area-to-mass ratio and Yarkovsky parameter.