pyTOPSScrape

Release 0.5

Thomas Boudreaux

Sep 06, 2022

CONTENTS

1 pyTOPSScrape.api package **1**
- 1.1 Submodules . . . 1
- 1.2 pyTOPSScrape.api.api module . . . 1
- 1.3 pyTOPSScrape.api.convert module . . . 2
  - 1.3.1 Functions . . . 3
- 1.4 pyTOPSScrape.api.utils module . . . 7
- 1.5 Module contents . . . 8

2 pyTOPSScrape.err package **9**
- 2.1 Submodules . . . 9
- 2.2 pyTOPSScrape.err.err module . . . 9
- 2.3 Module contents . . . 9

3 pyTOPSScrape.ext package **11**
- 3.1 Submodules . . . 11
- 3.2 pyTOPSScrape.ext.utils module . . . 11
- 3.3 Module contents . . . 12

4 pyTOPSScrape.parse package **13**
- 4.1 Submodules . . . 13
- 4.2 pyTOPSScrape.parse.abundance module . . . 13
- 4.3 Module contents . . . 17

5 pyTOPSScrape.scripts package **19**
- 5.1 Module contents . . . 19

6 pyTOPSScrape.misc package **21**
- 6.1 Subpackages . . . 21
  - 6.1.1 pyTOPSScrape.misc.dataFiles package . . . 21
    - 6.1.1.1 Module contents . . . 21
- 6.2 Submodules . . . 21
- 6.3 pyTOPSScrape.misc.utils module . . . 21
  - 6.3.1 Functions . . . 21
- 6.4 Module contents . . . 22

Python Module Index **23**

Index **25**

CHAPTER

ONE

PYTOPSSCRAPE.API PACKAGE

1.1 Submodules

1.2 pyTOPSScrape.api.api module

Author: Thomas M. Boudreaux

Created: September 2021

Last Modified: September 2021

Psuedo API for querying TOPS webform

pyTOPSScrape.api.api.**TOPS_query**(*mixString: str*, *mixName: str*, *nAttempts: int*) → bytes
Query TOPS form and retry n times

Parameters

- **mixString** (*string*) – string in the form of: "massFrac0 Element0 massFrac1 Element1 ..." which will be sumbittedi in the webform for mixture
- **mixName** (*string*) – name to be used in the webform
- **nAttemptes** (*int*) – How many times to reattempt after a failure.

Returns **tableHTML** – Table quired from TOPS cite.

Return type bytes

pyTOPSScrape.api.api.**TOPS_query_async_distributor**(*compList*, *outputDirectory*, *njobs=10*)
Distributes TOPS query jobs to different threads and gathers the results together. Writes out output.

Parameters

- **aFiles** (*list of TextIO*) – List of file like objects to abundance files to be parsed
- **outputDirectory** (*str*) – Path to directory to save TOPS query results to.
- **njobs** (*int, default=10*) – Number of concurrent jobs to allow at a time.

pyTOPSScrape.api.api.**call**(*aMap: str*, *aTable: str*, *outputDir: str*, *jobs: int*)
Main TOPS psuedo API call function. Will save results to outputDir with file format OP:IDX_X_Y_Z.dat where IDX is the ID of the composition (parallel to DSEP composition ID), X is the classical Hydrogen mass fraction, Y is the classical Helium mass fraction, and Z is the classical metal mass fraction.

Parameters

- **aMap** (*str*) – Path to the list of classical compositions to be used. List should be given as an ascii file where ecach row is X,Y,Z

- **aTable** (*str*) – Path to chemical abundance table to be used as base composition.
- **outputDir** (*str*) – Path to directory save TOPS query results into
- **jobs** (*int*) – Number of threads to query TOPS webform on

pyTOPSScrape.api.api.**parse_table**(*html: bytes*) → str
Parse the bytes table returned from mechanize into a string

Parameters **html** (*bytes*) – bytes table retuend from mechanize bowser at second TOPS submission form

Returns **table** – parsed html soruce in the form of a string

Return type string

pyTOPSScrape.api.api.**query_and_parse**(*compList: list*, *outputDirectory: int*, *i: int*, *nAttempts: int = 10*)
Async coroutine to query TOPS webform, parse the output, and write that to disk.

Parameters

- **file** (*TextIO*) – Abundance file to be parsed for the form as defined in the docstring for the function parse_numfrac_file
- **outputDirectory** (*str*) – Path to write out results of TOPS webquery
- **i** (*int*) – Index of composition so file name can properly keep track of where it is, even in parallel processing.
- **nAttempts** (*int, default=10*) – Number of time to retry TOPS query before failing out

pyTOPSScrape.api.api.**submit_TOPS_form**(*mixString: str*, *mixName: str*, *massFrac: bool = True*) → bytes
Open the Los Alamos opacity website, submit a given composition and then return the resultant table.

Parameters

- **mixString** (*string*) – string in the form of: "massFrac0 Element0 massFrac1 Element1 ..." which will be sumbittedi in the webform for mixture
- **mixName** (*string*) – name to be used in the webform
- **massFrac** (*bool, default=True*) – Submit as massFrac instead of numberFrac

Returns **tableHTML** – Table quired from TOPS cite.

Return type bytes

1.3 pyTOPSScrape.api.convert module

Author: Thomas M. Boudreaux

Created: September 2021

Last Modified: September 2021

Main conversion code for TOPS api, responsible for takine many TOPS results and merging them into a single OPAL formate high temperature opacity file.

1.3.1 Functions

- **comp_list_2_dict** Take a list containing compsoition information for a star in the form of [('Element Symbol', massFraction, numberFraction),...] and convert that into a dictionary of the form: {'Element Symbol': (massFraction, numberFraction),..}.
- **parse_RMO_TOPS_table_file** Given the path to a file queried from the TOPS webform put it into a computer usable form of 3 arrays. One array of mass density, one of LogT and one of log Rossland Mean Opacity
- **convert_rho_2_LogR** Maps a given kappa(rho,logT) parameter space onto a kappa(LogR, LogT) field through interpolation. The final field is the field that DSEP needs.
- **extract_composition_path** Given the name of a TOPS return file (named in the format OP:n_X_Y_Z.dat) extract X, Y, and Z
- **format_opal_comp_table** Take in all the information from a given TOPS tables and format it to the proper format for DSEP to undersand. Leave in some placeholders so that in future table can be labeld as the proper number.
- **format_OPAL_header** Writes the header of the opacity table that DSEP expects. This is written to be the same length (and basically the same contents) of the header from the OPACITY project. Not sure if that is required; however, if so I am matching it.
- **format_OPAL_table** Given a dictionary of tables and a composition Dictionary for solar composition in a given mixture (AGSS08, GS98, etc...) merge all the information together into a string which can be written to disk and would be the format of an opacoty project table (what DSEP expects)
- **format_TOPS_to_OPAL** Take the path to a table queried from the TOPS web form and fully convert it into a table which can be directly read by DSEP. (Note this function does not write anything to disk; however, the return products can be written to disk)
- **rebuild_formated_table** Iterate over a list of opacity tables and a list of desired chemical compositions then replace the contents of the table list with the newly updated RMOs frmo the interpolation.
- **TOPS_2_OPAL** Main conversoin utility to go between some set of TOPS tables and an OPAl table. Will take a set of 126 TOPS tables where each one is the opacity for one composition over a number of temperature and densities and rearange them into one large file with 126 tables within it. Each table will be over a range of temperatures and R values. To get to R val interpolation is used.

pyTOPSScrape.api.convert.**TOPS_2_OPAL**(*outputDirectory: str*, *aTable: str*, *aMap: str*, *output: str*, *nonRect: bool = False*)

Main conversoin utility to go between some set of TOPS tables and an OPAl table. Will take a set of 126 TOPS tables where each one is the opacity for one composition over a number of temperature and densities and rearange them into one large file with 126 tables within it. Each table will be over a range of temperatures and R values. To get to R val interpolation is used.

Parameters

- **outputDirectory** (*str*) – Path to directory where TOPS query results are stored
- **aTable** (*str*) – Path to a reference abundance table to use when filling the header with compositional information
- **aMap** (*str*) – Path to the abundance map. This should be an ascii file where each row is X,Y,Z. Each row will correspond to one rescaled composition which will be queried.
- **output** (*str*) – Path to save final OPAL formated table too
- **nonRect** (*bool, default=False*) – Flag to control whether output tables will be rectangular or have their corners cut off in a way consistant which how DSEP expects OPAL tables.

pyTOPSScrape.api.convert.**comp_list_2_dict**(*compList: collections.abc.Iterator*) → dict

Take a list containing compsoition information for a star in the form of [('Element Symbol', massFraction, numberFraction),...] and convert that into a dictionary of the form: {'Element Symbol': (massFraction, numberFraction),..}.

Parameters **compList** (*Iterator*) – list of the form: [(ElementSymbol, massFrac, numberFrac, ZmassFrac, massFrac Uncertanity, numberFrac Uncertanity, ZMassFrac Uncertanity),...]

Returns Dictionary of the form {'Element':(massFrac, numberFrac, ZmassFrac, massFrac Uncertanity, numberFrac Uncertanity, ZMassFrac Uncertanity),...}

Return type dict

pyTOPSScrape.api.convert.**convert_rho_2_LogR**(*rho: numpy.ndarray*, *LogT: numpy.ndarray*, *RMO: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Maps a given kappa(rho,logT) parameter space onto a kappa(LogR, LogT) field through interpolation. The final field is the field that DSEP needs.

Parameters

- **rho** (*np.ndarray*) – mass density array of size n
- **LogT** (*np.ndarray*) – LogT array of size m
- **RMO** (*np.ndarray*) – Opacity Array of size m x n

Returns

- **targetLogR** (*np.ndarray(shape=19)*) – Log R values which dsep requires
- **targetLogR** (*np.ndarray(shape=70)*) – Lof T values which dsep requires
- **Opacity** (*np.ndarray(shape=(70, 19))*) – Opacity array now interpolated into LogR, LogT space from rho LogT space and sampled at the exact LogR and LogT values required.

pyTOPSScrape.api.convert.**extract_composition_path**(*path: str*) → Tuple[float, float, float]

Given the name of a TOPS return file (named in the format OP:n_X_Y_Z.dat) extract X, Y, and Z

Parameters **path** (*string*) – path to TOPS return file

Returns

- **X** (*float*) – Hydrogen mass fraction
- **Y** (*float*) – Helium mass fraction
- **Z** (*Metal mass fraction*)

pyTOPSScrape.api.convert.**format_OPAL_header**(*compDict: dict*) → str

Writes the header of the opacity table that DSEP expects. This is written to be the same length (and basically the same contents) of the header from the OPACITY project. Not sure if that is required; however, if so I am matching it.

Parameters **compDict** (*dict*) – dictionary in the form: {'Element': (massFrac, numFrac), ...} used to fill up the header with composition information. This is meant to be the "solar" compositon of whatever mix you are using so... [Fe/H] = 0.0, [alpha/H] = 0.0, a(He) = 10.93

Returns The Header to be prepended to the opacity table file

Return type string

pyTOPSScrape.api.convert.**format_OPAL_table**(*tableDict: dict*, *compDict: dict*) → str

Given a dictionary of tables and a composition Dictionary for solar composition in a given mixture (AGSS08, GS98, etc…) merge all the information together into a string which can be written to disk and would be the format of an opacoty project table (what DSEP expects)

Parameters

- **tableDict** (*dict*) – dictionary of table elements, containing a "Summary" entry (metadata) and a "Table" entry. All occcurences of the the string "TNUM" will be replaced with the index+1 of where that table occurs in the file.
- **compDict** (*dict*) – dictionary in the form: {'Element': (massFrac, numFrac), … } used to fill up the header with composition information. This is meant to be the "solar" compositon of whatever mix you are using so… [Fe/H] = 0.0, [alpha/H] = 0.0, a(He) = 10.93

Returns **OPALFormatted** – Opacity Project formated table as a string which can be written to disk

Return type string

pyTOPSScrape.api.convert.**format_TOPS_to_OPAL**(*TOPSTable: str*, *comp: tuple*, *tnum: int*, *upperNonRect: Optional[numpy.ndarray] = None*) → Tuple[str, float, float, float, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Take the path to a table queried from the TOPS web form and fully convert it into a table which can be directly read by DSEP. (Note this function does not write anything to disk; however, the return products can be written to disk)

Parameters

- **TOPSTable** (*string*) – path to table queried from TOPS web form
- **comp** (*dict*) – composition dictionary
- **tnum** (*int*) – table number
- **upperNonRect** (*np.ndarray, optional*) – array describing how to fill the top of the table non rectangurally

Returns

- **metaLine** (*str*) – metadata extracted from table
- **X** (*float*) – Hydrogen mass fraction
- **Y** (*float*) – Helium mass fraction
- **Z** (*float*) – Metal mass fraction
- **LogR** (*np.ndarray(shape=19)*) – Log R values which dsep expects
- **LogT** (*np.ndarray(shape=70)*) – Log Temperature values which dsep expects.
- **LogRMO** (*np.ndarray(shape=(70, 19))*) – Log rossland mean opacities for the LogT and LogR arrays

pyTOPSScrape.api.convert.**format_opal_comp_table**(*LogR: numpy.ndarray*, *LogT: numpy.ndarray*, *LogRMO: numpy.ndarray*, *TNUM: int*, *comp: Optional[dict] = None*, *upperNonRect: Optional[numpy.ndarray] = None*, *nonRect: bool = False*) → Tuple[str, str]

Take in all the information from a given TOPS tables and format it to the proper format for DSEP to undersand. Leave in some placeholders so that in future table can be labeld as the proper number.

Parameters

- **LogR** (*ndarray*) – The Log R value array (horizontal axis of table)
- **LogT** (*ndarray*) – The Log Temperature value array (vertical axis)
- **LogRMO** (*ndarray*) – all of the RMO values associated with R and T
- **TNUM** (*int*) – Table number
- **comp** (*dict, optional*) – composition dictionary. If not provided placeholders are left in place so that it may be filled later on
- **upperNonRect** (*ndarray, default=None*) – Array describing how to fill the top of the table non rectangurally This array should be of the shape (nRowsPerTable * nTables, 3). So if you have 5 tables each with 70 rows then this array should have a shape of (350,3). The first column of this array correspond to the table that the row is a member of, the second column correspond to the row in that table that the row is. so the first row of the first table would be at upperNonRect[0,:] = [0,0,...] while the first row of the second table would be at upperNonRect[n,:] [1,0,...]. The final column desribes how many of the elements, counting from the left of the opacity table should be blanked out to 99.999 (the sentinal value DSEP uses for non entries). So a row of [2,55,8] would mean that for the 56th row in the 3rd table blank out the firts 8 opacity values (opacities for the first 8 values of logR).
- **nonRect** (*bool, default = False*) – Flag to control whether output tables will be rectangular or have their corners cut off in a way consistant which how DSEP expects OPAL tables.

Returns

- **metaLine** (*str*) – header line for each table, may or may not have placeholders in it
- **fullTable** (*str*) – full table to be places in opacity file

pyTOPSScrape.api.convert.**parse_RMO_TOPS_table_file**(*TOPSTable: str*, *n: int = 100*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Given the path to a file queried from the TOPS webform put it into a computer usable form of 3 arrays. One array of mass density, one of LogT and one of log Rossland Mean Opacity

Parameters

- **TOPSTable** (*string*) – Path to file queried from TOPS webform
- **n** (*int, default=100*) – The size of density grid used in TOPS query form.

Returns

- **rho** (*np.ndarray(shape=n)*) – Array of mass densities (in cgs) parsed from TOPS table.
- **LogT** (*np.ndarray(shape=m)*) – Array of temperatures (in Kelvin) parsed from TOPS table.
- **OPALTableInit** (*np.ndarray(shape=(m,n))*) – Array of Rossland Mean Opacities parsed from TOPS table.

pyTOPSScrape.api.convert.**rebuild_formated_tables**(*formatedTables*, *interpRMO*, *pContents*, *upperNonRect*, *nonRect=False*)

Iterate over a list of opacity tables and a list of desired chemical compositions then replace the contents of the table list with the newly updated RMOs frmo the interpolation.

Parameters

- **formatedTables** (*list of dicts*) – List of dictionaries holding three axis each. X, Z, and LogRMO. These are the "observed" values to be interpolared

- **interpRMO** (*list of ndarrays*) – RMOs after interpolation to LogR-LogT space from rho-LogT space.
- **pContents** (*np.array(shape=(n, 3))*) – Numpy array of all the compositions of length n. For a dsep n=126. Along the second axis the first column is X, the second is Y, and the third is Z.
- **upperNonRect** (*ndarray*) – Array describing how to fill the top of the table non rectangularally This array should be of the shape (nRowsPerTable * nTables, 3). So if you have 5 tables each with 70 rows then this array should have a shape of (350,3). The first column of this array correspond to the table that the row is a member of, the second column correspond to the row in that table that the row is. so the first row of the first table would be at upperNonRect[0,:] = [0,0,...] while the first row of the second table would be at upperNonRect[n,:] [1,0,...]. The final column desribes how many of the elements, counting from the left of the opacity table should be blanked out to 99.999 (the sentinal value DSEP uses for non entries). So a row of [2,55,8] would mean that for the 56th row in the 3rd table blank out the firts 8 opacity values (opacities for the first 8 values of logR).
- **nonRect** (*bool, default = False*) – Flag to control whether output tables will be rectangular or have their corners cut off in a way consistant which how DSEP expects OPAL tables.

Returns **formatedTables** – List of dictionaries holding three axis each. X, Z, and LogRMO. These have been updated to reflect the compositions in pContents.

Return type list of dicts

1.4 pyTOPSScrape.api.utils module

Author: Thomas M. Boudreaux

Created: September 2021

Last Modified: September 2021

Utilities to help with the TOPS query api

pyTOPSScrape.api.utils.**format_TOPS_string**(*compList: list*) → str

Format the composition list from pasrse_abundance_file into a string in the form that the TOPS web form expects for a mass fraction input.

Parameters **compList** (*list*) – composition list in the form of: [('Element', massFrac, numFrac),...]

Returns **TOPS_abundance_string** – string in the form of: "massFrac0 Element0 massFrac1 Element1 ..."

Return type string

pyTOPSScrape.api.utils.**validate_extant_tables**(*path: str*, *prefix: str*) → bool

Check if there is a quiried table from TOPS for every number frac file generated by the program passed to call_num_frac.

Parameters

- **path** (*string*) – Path to where the results of the number frac and TOPS query files are stored
- **prefix** (*string*) – start prefix given to all abundance / number frac files

Returns **validated** – Whether or not all number frac files have a corresponding TOPS opacity table

Return type bool

1.5 Module contents

CHAPTER

TWO

PYTOPSSCRAPE.ERR PACKAGE

2.1 Submodules

2.2 pyTOPSScrape.err.err module

2.3 Module contents

CHAPTER

THREE

PYTOPSSCRAPE.EXT PACKAGE

3.1 Submodules

3.2 pyTOPSScrape.ext.utils module

pyTOPSScrape.ext.utils.**call_num_frac**(*abunTable: str*, *feh: Union[float, Tuple[float, float, int]]*, *alpha: Union[float, Tuple[float, float, int]]*, *Y: Union[float, Tuple[float, float, int]]*, *Xc: float*, *Yc: float*) → _io.BytesIO

Given some grid of [Fe/H], [Alpha/Fe], and a(He) generate the number fractions files for every point on that grid. This is done using the libnumfrac shared library in ext/lib.

Parameters

- **abunTable** (*string*) – Table to parse abundances from.
- **feh** (*float or tuple of floats*) – [Fe/H] value to pass to program. If this is a float the numFrac program will only be evaluate at that point, if a tuple it will be evaluated at every point within linspace(feh[0], feh[1], feh[2]).
- **alpha** (*float of tuple of floats*) – [alpha/H] value to pass to program. If this is a float the numFrac program will only be evaluate at that point, if a tuple it will be evaluated at every point within linspace(alpha[0], alpha[1], alpha[2]).
- **Y** (*float or tuple of floats*) – a(He) value to pass to program. If this is a float the numFrac program will only be evaluate at that point, if a tuple it will be evaluated at every point within linspace(Y[0], Y[1], Y[2]).
- **Xc** (*float*) – Current X to use as a reference
- **Yc** (*float*) – Current Y to use as a referece (only used if Xc == 0)

Returns **fp** – Temporary file object storing results

Return type BytesIO

pyTOPSScrape.ext.utils.**get_base_composition**(*aTablePath: str*) → Tuple[list, float, float, float]

For some abundance path return the "base" composition, this is mainly to be used for headers.

Parameters **aTablePath** (*str*) – Path to the abundance table in the form as described in the parseChemFile module documentation

Returns

- *list* – list of the composition in the form [('Element',massFrac,numberFrac),...]
- *float* – Hydrogen mass fraction
- *float* – Helium mass fraction

- *float* – Metal mass fraction

pyTOPSScrape.ext.utils.**parse_numfrac_file**(*file: _io.BytesIO*, *big: bool = False*, *pbar: bool = True*) → Tuple[numpy.ndarray, float, float, float]

Given a file generated by the executable used in call_num_frac parse that file into a usable form. This includes the hydrogen, helium, and metall mass fractions. And a list in the form of [('Element', massFrac, numberFrac),...]

Parameters

- **file** (*BytesIO*) – file like object to abundance file
- **big** (*bool*) – single composition file or one file with many composition
- **pbad** (*bool*) – display progress bar

Returns

- *list* – list of the composition in the form [('Element',massFrac,numberFrac),...]
- *float* – Hydrogen mass fraction
- *float* – Helium mass fraction
- *float* – Metal mass fraction

3.3 Module contents

CHAPTER

FOUR

PYTOPSSCRAPE.PARSE PACKAGE

4.1 Submodules

4.2 pyTOPSScrape.parse.abundance module

Author: Thomas M. Boudreaux

Created: May 2021

Last Modified: May 2021

Module responsible for the parsing and handeling of chemical composition files in the form of

```
#STD [Fe/H] [alpha/Fe] [C/Fe] [N/Fe] [O/Fe] [r/Fe] [s/Fe] C/O X Y,Z
F -1.13 0.32 -0.43 -0.28 0.31 -1.13 -1.13 0.10 0.7584 0.2400,1.599E-03
#H He Li Be B C N O F Ne
12.00 10.898 -0.08 0.25 1.57 6.87 6.42 7.87 3.43 7.12
#Na Mg Al Si P S Cl Ar K Ca
5.11 6.86 5.21 6.65 4.28 6.31 -1.13 5.59 3.90 5.21
#Sc Ti V Cr Mn Fe Co Ni Cu Zn
2.02 3.82 2.80 4.51 4.30 6.37 3.86 5.09 3.06 2.30
#Ga Ge As Se Br Kr Rb Sr Y Zr
0.78 1.39 0.04 1.08 0.28 0.99 0.26 0.61 1.08 1.45
#Nb Mo Tc Ru Rh Pd Ag Cd In Sn
-0.80 -0.38 -99.00 -0.51 -1.35 -0.69 -1.32 -0.55 -1.46 -0.22
#Sb Te I Xe Cs Ba La Ce Pr Nd
-1.25 -0.08 -0.71 -0.02 -1.18 1.05 -0.03 0.45 -1.54 0.29
#Pm Sm Eu Gd Tb Dy Ho Er Tm Yb
-99.00 -1.30 -0.61 -1.19 -1.96 -1.16 -1.78 -1.34 -2.16 -1.42
#Lu Hf Ta W Re Os Ir Pt Au Hg
-2.16 -1.41 -2.38 -1.41 -2.00 -0.86 -0.88 -0.64 -1.34 -1.09
#Tl Pb Bi Po At Rn Fr Ra Ac Th
-1.36 -0.51 -1.61 -99.00 -99.00 -99.00 -99.00 -99.00 -99.00 -2.20
#Pa U
-99.00 -2.80
```

Where each number is a(i) for the ith element and lines starting with # are comments.

pyTOPSScrape.parse.abundance.**a_to_mfrac**(*a*, *amass*, *X*)

Convert $a(i)$ for the i^{th} element to a mass fraction using the expression

$$a(i) = \log(1.008) + \log(F_i) - [\log(X) + \log(m_i)] + 12$$

Or, equivilenetly, to go from $a(i)$ to mass fraction

$$F_i = \left[\frac{Xm_i}{1.008}\right] \times 10^{a(i)-12}$$

Where F_i is the math fraction of the i^{th} element, X is the Hydrogen mass fraction, and m_i is the ith element mass in hydrogen masses.

Parameters

- **a** (*float*) – $a(i)$ for the i^{th} element. For example for He chem might be 10.93. For Hydrogen it would definititionally be 12.
- **amass** (*float*) – Mass of i^{th} element given in atomic mass units.
- **X** (*float*) – Hydrogen mass fraction

Returns **mf** – Mass fraction of i^{th} element.

Return type float

pyTOPSScrape.parse.abundance.**est_feh_from_Z_and_X**(*abunTable: dict*, *Xt: float*, *Zt: float*) → float

Analytically estimate feh from Z and X

Parameters

- **abunTable** (*dict*) – Abundance Table dictionary in the form described in the docs for pysep.misc.abun.util.open_and_parse.
- **Xt** (*float*) – Target X to move to
- **Zt** (*float*) – Target Z to move to.

Returns **FeH** – [Fe/H] value to add to every a(i) for every tracked element i where i > 2 (i.e all the metals).

Return type float

pyTOPSScrape.parse.abundance.**gen_abun_map**(*abunTable*)

Generate an analytic mapping between X, Y, Z and FeH given an abundance table.

Parameters **abunTable** (*str*) – Path of checmical abundance table to use for composition. Format of this table is defined in the ext module documentation.

Returns **MetalAbunMap** – Function build from interpolation of a grid of FeH, alpha/Fe, and a(He) which will returned the set of those values giving the composition most similar to an input X, Y, and Z.

Return type function(X,Y,Z) -> (Fe/H,0.0,a(He))

pyTOPSScrape.parse.abundance.**get_atomic_masses**()

Return a dict of atomic masses from Hydrogen all the way to plutonium

Returns **amasses** – Dicionary of atomic masses in atomic mass units indexed by elemental symbol.

Return type dict of floats

pyTOPSScrape.parse.abundance.**mfrac_to_a**(*mfrac*, *amass*, *X*, *Y*)

Convert mass fracition of a given element to a for that element at a given hydrogen mass fraction using the equation

$$a(i) = \log(1.008) + \log(F_i) - [\log(X) + \log(m_i)]$$

Where F_i is the mass fraction for the i^{th} element and m_i is the mass fraction for the i^{th} element.

Parameters

- **mfrac** (*float*) – Mass fraction of the ith element.
- **amass** (*float*) – Mass of the ith element in atomic mass units.
- **X** (*float*) – Hydrogen mass fraction
- **Y** (*float*) – Helium mass fraction, will be used as reference if X = 0

Returns **a** – a for the ith element

Return type float

pyTOPSScrape.parse.abundance.**open_and_parse**(*path*)
Open and parse the contents of a chemical composition file

Parameters **path** (*str*) – Path to open file

Returns

parsed –

Dictionary with two indexes.

- **Abundance Ratio** Includes the indexes:
  - STD (*str*)
  - [Fe/H] (*float*)
  - [alpha/Fe] (*float*)
  - [C/Fe] (*float*)
  - [N/Fe] (*float*)
  - [O/Fe] (*float*)
  - [r/Fe] (*float*)
  - [s/Fe] (*float*)
  - C/O (*float*)
  - X (*float*)
  - Y (*float*)
  - Z (*float*)
- **RelativeAbundance** Includes an index for each chemical symbol given in the file format definition provided in the module documentation. These are all floats.

Return type dict

pyTOPSScrape.parse.abundance.**open_chm_file**(*path*)
Open a chemical composition file (format defined in the module documentation). Split the contents by line then remove all lines which start with #. Finally split each line by both whitespace and commas.

Parameters **path** (*str*) – Path to file to open

Returns **contents** – List of list of strings. The outter index selects the row, the inner index selectes the column within the row.

Return type list

pyTOPSScrape.parse.abundance.**parse**(*contents: list*) → dict
Parse chem file in the format described in the module documentation.

The abuundance ratios and abundances on the first row are added to a dict under the key ['AbundanceRatio'] and sub indexed by the comments above each entry (Note that these are not read; rather, they are assumed to be the same in every file). The subsequent values (on all other rows) are added to the same dict under the key ['RelativeAbundance'] and sub indexed by their chemical symbols.

Parameters `contents` (*list*) – List of list of strings. The outter index selects the row, the inner index selected the column in the row and at each coordinate is a string which can be cast as a float. The one exception is that string at 0,0 is a charectar.

Returns

extracted –

Dictionary with two indexes.

- **Abundance Ratio** Includes the indexes:
  - STD (*str*)
  - [Fe/H] (*float*)
  - [alpha/Fe] (*float*)
  - [C/Fe] (*float*)
  - [N/Fe] (*float*)
  - [O/Fe] (*float*)
  - [r/Fe] (*float*)
  - [s/Fe] (*float*)
  - C/O (*float*)
  - X (*float*)
  - Y (*float*)
  - Z (*float*)
- **RelativeAbundance** Includes an index for each chemical symbol given in the file format from the module documentation. These are all floats.

Return type dict

pyTOPSScrape.parse.abundance.**parse_abundance_map**(*path: str*) → numpy.ndarray

Parse Hydrogen, Helium, and metal mass fraction out of a csv where each row is one composition, the first column is X, second is Y, and the third is Z. Comments may be included in the file if the first non white space character on the line is a hash.

Parameters `path` (*str*) – Path to the abundance map. This should be an ascii file where each row contains X, Y, and Z (comma delimited with no white space). Each row will define one set of tables to be queried with the idea being that the entire file describes the entire set of tables to be queried.

Returns **pContents** – numpy array of all the compositions of length n where n is the number of rows whos first non white space character was not a hash. For a DSEP n=126. Along the second axis the first column is X, the second is Y, and the third is Z.

Return type np.ndarray(shape=(n,3))

4.3 Module contents

CHAPTER

FIVE

PYTOPSSCRAPE.SCRIPTS PACKAGE

5.1 Module contents

CHAPTER

SIX

PYTOPSSCRAPE.MISC PACKAGE

6.1 Subpackages

6.1.1 pyTOPSScrape.misc.dataFiles package

6.1.1.1 Module contents

6.2 Submodules

6.3 pyTOPSScrape.misc.utils module

Author: Thomas M. Boudreaux

Created: Febuary 2021

Last Modified: July 2021

Opacity utility functions

6.3.1 Functions

- **get_target_log_R** Return a numpy array with the LogR values required by DSEP for high temperature opacity tables.
- **get_target_log_T** Return a numpy array with the LogT values required by DSEP for high temperature opacity tables.

pyTOPSScrape.misc.utils.**get_target_log_R**() → numpy.ndarray
Get the ndarray for the LogR values that DSEP expects

Returns **targetLogR** – Array of LogR values expected by DSEP in opacity table

Return type np.ndarray

pyTOPSScrape.misc.utils.**get_target_log_T**() → numpy.ndarray
Get the ndarray for the LogT values that DSEP expects

Returns **targetLogT** – Array of LogT values expected by DSEP in opacity table

Return type np.ndarray

pyTOPSScrape.misc.utils.**load_non_rect_map**() → numpy.ndarray
Load the upper non rectabtular map from numpy binary which DSEP requires for the high temperature opacity files.

Returns **upperNonRect** – Upper non rectangular map which DSEP requires.

Return type np.ndarray

6.4 Module contents

PYTHON MODULE INDEX

p

pyTOPSScrape.api, 8
pyTOPSScrape.api.api, 1
pyTOPSScrape.api.convert, 2
pyTOPSScrape.api.utils, 7
pyTOPSScrape.err, 9
pyTOPSScrape.err.err, 9
pyTOPSScrape.ext, 12
pyTOPSScrape.ext.utils, 11
pyTOPSScrape.misc, 22
pyTOPSScrape.misc.dataFiles, 21
pyTOPSScrape.misc.utils, 21
pyTOPSScrape.parse, 17
pyTOPSScrape.parse.abundance, 13
pyTOPSScrape.scripts, 19

INDEX

A

a_to_mfrac() (*in module pyTOPSScrape.parse.abundance*), 13

C

call() (*in module pyTOPSScrape.api.api*), 1
call_num_frac() (*in module pyTOPSScrape.ext.utils*), 11
comp_list_2_dict() (*in module pyTOPSScrape.api.convert*), 3
convert_rho_2_LogR() (*in module pyTOPSScrape.api.convert*), 4

E

est_feh_from_Z_and_X() (*in module pyTOPSScrape.parse.abundance*), 14
extract_composition_path() (*in module pyTOPSScrape.api.convert*), 4

F

format_opal_comp_table() (*in module pyTOPSScrape.api.convert*), 5
format_OPAL_header() (*in module pyTOPSScrape.api.convert*), 4
format_OPAL_table() (*in module pyTOPSScrape.api.convert*), 4
format_TOPS_string() (*in module pyTOPSScrape.api.utils*), 7
format_TOPS_to_OPAL() (*in module pyTOPSScrape.api.convert*), 5

G

gen_abun_map() (*in module pyTOPSScrape.parse.abundance*), 14
get_atomic_masses() (*in module pyTOPSScrape.parse.abundance*), 14
get_base_composition() (*in module pyTOPSScrape.ext.utils*), 11
get_target_log_R() (*in module pyTOPSScrape.misc.utils*), 21
get_target_log_T() (*in module pyTOPSScrape.misc.utils*), 21

L

load_non_rect_map() (*in module pyTOPSScrape.misc.utils*), 21

M

mfrac_to_a() (*in module pyTOPSScrape.parse.abundance*), 14
module
    pyTOPSScrape.api, 8
    pyTOPSScrape.api.api, 1
    pyTOPSScrape.api.convert, 2
    pyTOPSScrape.api.utils, 7
    pyTOPSScrape.err, 9
    pyTOPSScrape.err.err, 9
    pyTOPSScrape.ext, 12
    pyTOPSScrape.ext.utils, 11
    pyTOPSScrape.misc, 22
    pyTOPSScrape.misc.dataFiles, 21
    pyTOPSScrape.misc.utils, 21
    pyTOPSScrape.parse, 17
    pyTOPSScrape.parse.abundance, 13
    pyTOPSScrape.scripts, 1, 19

O

open_and_parse() (*in module pyTOPSScrape.parse.abundance*), 15
open_chm_file() (*in module pyTOPSScrape.parse.abundance*), 15

P

parse() (*in module pyTOPSScrape.parse.abundance*), 15
parse_abundance_map() (*in module pyTOPSScrape.parse.abundance*), 16
parse_numfrac_file() (*in module pyTOPSScrape.ext.utils*), 12
parse_RMO_TOPS_table_file() (*in module pyTOPSScrape.api.convert*), 6
parse_table() (*in module pyTOPSScrape.api.api*), 2
pyTOPSScrape.api
    module, 8
pyTOPSScrape.api.api

module, 1
pyTOPSScrape.api.convert
module, 2
pyTOPSScrape.api.utils
module, 7
pyTOPSScrape.err
module, 9
pyTOPSScrape.err.err
module, 9
pyTOPSScrape.ext
module, 12
pyTOPSScrape.ext.utils
module, 11
pyTOPSScrape.misc
module, 22
pyTOPSScrape.misc.dataFiles
module, 21
pyTOPSScrape.misc.utils
module, 21
pyTOPSScrape.parse
module, 17
pyTOPSScrape.parse.abundance
module, 13
pyTOPSScrape.scripts
module, 1, 19

Q

query_and_parse() (*in module pyTOPSScrape.api.api*), 2

R

rebuild_formated_tables() (*in module pyTOPSScrape.api.convert*), 6

S

submit_TOPS_form() (*in module pyTOPSScrape.api.api*), 2

T

TOPS_2_OPAL() (*in module pyTOPSScrape.api.convert*), 3
TOPS_query() (*in module pyTOPSScrape.api.api*), 1
TOPS_query_async_distributor() (*in module pyTOPSScrape.api.api*), 1

V

validate_extant_tables() (*in module pyTOPSScrape.api.utils*), 7