nmrstarlib Documentation

Release 2.0.2

Andrey Smelter, Hunter N.B. Moseley

Jun 09, 2017

Contents

CHAPTER 1

nmrstarlib

The *nmrstarlib* package is a Python library that facilitates reading and writing NMR-STAR formatted files used by the Biological Magnetic Resonance Data Bank (BMRB) for archival of Nuclear Magnetic Resonance (NMR) data.

The *nmrstarlib* package provides facilities to convert NMR-STAR formatted files into their equivalent JSONized (JavaScript Object Notation, an open-standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs) representation and vice versa.

The *nmrstarlib* package also provides facilities to create simulated peak lists for different types of standard solution and solid-state NMR experiments from chemical shifts and assignment information deposited in NMR-STAR files.

In addition, the *nmrstarlib* package provides methods to visualize chemical shift data.

The *nmrstarlib* package can be used in several ways:

- As a library for accessing and manipulating data stored in NMR-STAR format files.
- As a command-line tool to convert between NMR-STAR format and its equivalent JSONized NMR-STAR format, to create a large number of simulated peak lists, and also to visualize chemical shift data.

Citation

When using *nmrstarlib* in published work, please cite the following paper:

- Smelter, Andrey, Morgan Astra, and Hunter NB Moseley. "A fast and efficient python library for interfacing with the Biological Magnetic Resonance Data Bank." *BMC Bioinformatics* 18.1 (2017): 175. doi: 10.1186/s12859-017-1580-5.

Links

- nmrstarlib @ GitHub
- nmrstarlib @ PyPI

- Documentation @ ReadTheDocs

Installation

The *nmrstarlib* package runs under Python 2.7 and Python 3.4+, use pip to install. Starting with Python 3.4, pip is included by default.

Install on Linux, Mac OS X

```
python3 -m pip install nmrstarlib
```

Install on Windows

```
py -3 -m pip install nmrstarlib
```

Quickstart

Import *nmrstarlib* library and create generator function that will yield *nmrstarlib.nmrstarlib.StarFile* instance(s):

```
>>> from nmrstarlib import nmrstarlib
>>>
>>> # "path": path_to_file / path_to_dir / path_to_archive / bmrb_id / file_url
>>> starfile_gen = nmrstarlib.read_files("path")
>>>
>>> for starfile in starfile_gen:
...     print(starfile.bmrbid)          # print BMRB id of StarFile
...     print(starfile.source)          # print source of StarFile
...     print(list(starfile.keys()))    # print StarFile saveframe categories
>>>
>>> # For example, let's read two files: one using BMRB id and the other one using␣
↪URL:
>>> starfile_gen = nmrstarlib.read_files("15000", "http://rest.bmrb.wisc.edu/bmrb/NMR-
↪STAR3/18569")
>>>
>>> for starfile in starfile_gen:
...     print("BMRB id:", starfile.bmrbid)
...     print("Source:", starfile.source)
...     print("List of saveframes and comments:", list(starfile.keys()))
>>>
```

Note: Read the User Guide and The nmrstarlib Tutorial on ReadTheDocs to learn more and to see code examples on using the *nmrstarlib* as a library and as a command-line tool.

License

This package is distributed under the MIT *license*.

CHAPTER 2

Documentation index:

User Guide

Description

The *nmrstarlib* package provides a simple Python interface for parsing and manipulating data stored in NMR-STAR format files used by Biological Magnetic Resonance Data Bank (BMRB) for archival of Nuclear Magnetic Resonance (NMR) experimental data.

The *nmrstarlib* package provides facilities to convert NMR-STAR formatted files into their equivalent JSONized (JavaScript Object Notation, an open-standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs) representation and visa versa.

The *nmrstarlib* package also provides facilities to create simulated peak lists for different types of standard solution and solid-state NMR experiments from chemical shifts and assignment information deposited in NMR-STAR files.

In addition, the *nmrstarlib* package provides facilities to visualize assigned chemical shift data.

Installation

The *nmrstarlib* package runs under Python 2.7 and Python 3.4+. Starting with Python 3.4, pip is included by default. To install system-wide with pip run the following:

Install on Linux, Mac OS X

```
python3 -m pip install nmrstarlib
```

Install on Windows

```
py -3 -m pip install nmrstarlib
```

Install inside virtualenv

For an isolated install, you can run the same inside a virtualenv.

```
$ virtualenv -p /usr/bin/python3 venv  # create virtual environment, use python3␣
↪interpreter

$ source venv/bin/activate             # activate virtual environment

$ python3 -m pip install nmrstarlib    # install nmrstarlib as usually

$ deactivate                           # if you are done working in the virtual␣
↪environment
```

Get the source code

Code is available on GitHub: https://github.com/MoseleyBioinformaticsLab/nmrstarlib

You can either clone the public repository:

```
$ https://github.com/MoseleyBioinformaticsLab/nmrstarlib.git
```

Or, download the tarball and/or zipball:

```
$ curl -OL https://github.com/MoseleyBioinformaticsLab/nmrstarlib/tarball/master

$ curl -OL https://github.com/MoseleyBioinformaticsLab/nmrstarlib/zipball/master
```

Once you have a copy of the source, you can embed it in your own Python package, or install it into your system site-packages easily:

```
$ python3 setup.py install
```

Dependencies

The *nmrstarlib* package depends on several Python libraries, it will install all dependencies automatically, but if you wish to install them manually run the following commands:

- **docopt for creating *nmrstarlib* command-line interface.**
  - To install docopt run the following:

```
python3 -m pip install docopt  # On Linux, Mac OS X
py -3 -m pip install docopt    # On Windows
```

- **graphviz for visualizing assigned chemical shift values.**
  - To install the graphviz Python library run the following:

```
python3 -m pip install graphviz  # On Linux, Mac OS X
py -3 -m pip install graphviz    # On Windows
```

  - The only dependency of the graphviz Python library is a working installation of Graphviz (Graphviz download page).

- **numpy for generating noise values from random normal distribution during peak list simulation.**
  - To install the numpy Python library run the following:

```
python3 -m pip install numpy  # On Linux, Mac OS X
py -3 -m pip install numpy    # On Windows
```

Basic usage

The *nmrstarlib* package can be used in several ways:

- As a library for accessing and manipulating data stored in NMR-STAR format files.
  - Create the *StarFile* generator function that will generate (yield) single *StarFile* instance at a time.
  - Process each *StarFile* instance:
    - Process NMR-STAR files in a for-loop one file at a time.
    - Process as an iterator calling the `next()` built-in function.
    - Convert the generator into a `list` of *StarFile* objects.
- As a command-line tool:
  - Convert from NMR-STAR file format into its equivalent JSON file format and vice versa.
  - Create standard solution and solid-state NMR simulated peak lists from chemical shift values and assignment information.
  - Visualize (organize) assigned chemical shift values.

Note: Read *The nmrstarlib Tutorial* to learn more and see code examples on using the *nmrstarlib* as a library and as a command-line tool.

The nmrstarlib Tutorial

The *nmrstarlib* package provides classes and other facilities for parsing, accessing, and manipulating data stored in NMR-STAR and JSONized NMR-STAR formats. Also, the *nmrstarlib* package provides simple command-line interface.

Using nmrstarlib as a library

Importing nmrstarlib module

If the *nmrstarlib* package is installed on the system, the *nmrstarlib.nmrstarlib* module can be imported:

```
>>> from nmrstarlib import nmrstarlib
```

Constructing StarFile generator

The *nmrstarlib* module provides the *read_files()* generator function that yields *StarFile* instances. Constructing a *StarFile* generator is easy - specify the path to a local NMR-STAR file, directory of NMR-STAR files, archive of NMR-STAR files or BMRB id:

```
>>> from nmrstarlib import nmrstarlib
>>>
>>> single_starfile = nmrstarlib.read_files("bmr18569.str")  # single NMR-STAR file
>>>
>>> starfiles = nmrstarlib.read_files("bmr18569.str", "bmr336.str") # several NMR-
↪STAR files
>>>
>>> dir_starfiles = nmrstarlib.read_files("starfiles_dir")    # directory of NMR-STAR␣
↪files
>>>
>>> arch_starfiles = nmrstarlib.read_files("starfiles.zip")  # archive of NMR-STAR␣
↪files
>>>
>>> url_starfile = nmrstarlib.read_files("18569")            # BMRB id of NMR-STAR␣
↪file
>>>
```

Processing StarFile generator

The *StarFile* generator can be processed in several ways:

- Feed it to a for-loop and process one file at a time:

```
>>> for starfile in dir_starfiles:
...     print(starfile.bmrbid)                  # print BMRB id of StarFile
...     print(starfile.source)                  # print source of StarFile
...     for saveframe_name in starfile.keys():  # print saveframe names
...         print(saveframe_name)
>>>
```

Note: Once the generator is consumed, it becomes empty and needs to be created again.

- Since the *StarFile* generator behaves like an iterator, we can call the `next()` built-in function:

```
>>> starfile1 = next(dir_starfiles)
>>> starfile2 = next(dir_starfiles)
>>> ...
```

Note: Once the generator is consumed, `StopIteration` will be raised.

- Convert the *StarFile* generator into a `list` of *StarFile* objects:

```
>>> starfiles_list = list(dir_starfiles)
>>>
```

Warning: Do not convert the *StarFile* generator into a `list` if the generator can yield a large number of files, e.g. several thousand, otherwise it can consume all available memory.

Accessing and manipulating data from a single StarFile

Since a *StarFile* is a Python `collections.OrderedDict`, data can be accessed and manipulated as with any regular Python `dict` object using bracket accessors.

- Accessing data in *StarFile*:

```
>>> list(starfile.keys())  # list StarFile-level keys, i.e. saveframe names
['data', 'save_entry_information', 'save_entry_citation', 'save_assembly',
 'save_EVH1', 'save_natural_source', 'save_experimental_source',
 'save_sample_1', 'save_sample_2', 'save_sample_3', 'save_sample_4',
 'save_sample_conditions_1', 'save_sample_conditions_2',
 'save_sample_conditions_3', 'save_sample_conditions_4', 'save_AZARA',
 'save_xwinnmr', 'save_ANSIG', 'save_CNS', 'save_spectrometer_1',
 'save_spectrometer_2', 'save_NMR_spectrometer_list', 'save_experiment_list',
 'save_chemical_shift_reference_1', 'save_assigned_chem_shift_list_1',
 'save_combined_NOESY_peak_list']
>>>
>>> starfile["data"]
'18569'
>>>
>>> starfile["save_entry_information"]
OrderedDict([
 ('Entry.Sf_category', 'entry_information'),
 ('Entry.Sf_framecode', 'entry_information'),
 ('Entry.ID', '18569'),
 ('Entry.Title', ';\n13C, 15N and 1H backbone and sidechain assignments\n of␣
↪the
                  ENA-VASP homology 1 (EVH1) domain of the human
                  vasodilator-stimulated phosphoprotein (VASP)\n;'),
 ('Entry.Type', '.'),
 ('Entry.Version_type', 'original'),
 ('Entry.Submission_date', '2012-07-05'),
 ('Entry.Accession_date', '2012-07-05'), ...
])
>>>
>>> list(starfile["save_entry_information"].keys())  # list saveframe-level␣
↪keys
['Entry.Sf_category', 'Entry.Sf_framecode', 'Entry.ID', 'Entry.Title',
 'Entry.Type', 'Entry.Version_type', 'Entry.Submission_date',
 'Entry.Accession_date', 'Entry.Last_release_date', 'Entry.Original_release_
↪date',
 'Entry.Origination', 'Entry.NMR_STAR_version', 'Entry.Original_NMR_STAR_
↪version',
 'Entry.Experimental_method', 'Entry.Experimental_method_subtype', 'Entry.
↪Details',
 'Entry.BMRB_internal_directory_name', 'loop_0', 'loop_1', 'loop_2', 'loop_3
↪', 'loop_4']
>>>
```

```
>>> starfile["save_entry_information"]["Entry.Submission_date"]
'2012-07-05'
>>>
>>> starfile["save_entry_information"]["loop_0"]
(['Entry_author.Ordinal', 'Entry_author.Given_name', 'Entry_author.Family_
↪name',
  'Entry_author.First_initial', 'Entry_author.Middle_initials',
  'Entry_author.Family_title', 'Entry_author.Entry_ID'],
  [OrderedDict([('Entry_author.Ordinal', '1'),
                ('Entry_author.Given_name', 'Linda'),
                ('Entry_author.Family_name', 'Ball'),
                ('Entry_author.First_initial', '.'),
                ('Entry_author.Middle_initials', 'J.'),
                ('Entry_author.Family_title', '.'),
                ('Entry_author.Entry_ID', '18569')]),
   OrderedDict([('Entry_author.Ordinal', '2'),
                ('Entry_author.Given_name', 'Schmieder'),
                ('Entry_author.Family_name', 'Peter'),
                ('Entry_author.First_initial', '.'),
                ('Entry_author.Middle_initials', '.'),
                ('Entry_author.Family_title', '.'),
                ('Entry_author.Entry_ID', '18569')])
])
>>>
>>> starfile["save_entry_information"]["loop_0"][0]  # list loop-level keys
['Entry_author.Ordinal', 'Entry_author.Given_name', 'Entry_author.Family_name
↪',
'Entry_author.First_initial', 'Entry_author.Middle_initials',
'Entry_author.Family_title', 'Entry_author.Entry_ID']
>>>
>>> # loop values is a list of dictionaries:
>>> starfile["save_entry_information"]["loop_0"][1]
[OrderedDict([('Entry_author.Ordinal', '1'),
              ('Entry_author.Given_name', 'Linda'),
              ('Entry_author.Family_name', 'Ball'),
              ('Entry_author.First_initial', '.'),
              ('Entry_author.Middle_initials', 'J.'),
              ('Entry_author.Family_title', '.'),
              ('Entry_author.Entry_ID', '18569')]),
 OrderedDict([('Entry_author.Ordinal', '2'),
              ('Entry_author.Given_name', 'Schmieder'),
              ('Entry_author.Family_name', 'Peter'),
              ('Entry_author.First_initial', '.'),
              ('Entry_author.Middle_initials', '.'),
              ('Entry_author.Family_title', '.'),
              ('Entry_author.Entry_ID', '18569')])]
>>>
>>> # every loop entry is accessed by index:
>>> starfile["save_entry_information"]["loop_0"].[1][0]["Entry_author.Family_
↪name"]
'Ball'
>>> starfile["save_entry_information"]["loop_0"].[1][1]["Entry_author.Family_
↪name"]
'Peter'
```

- Manipulating data in a *StarFile* is easy - access data using bracket accessors and set a new value:

```
>>> starfile["data"]
'18569'
>>>
>>> starfile["data"] = "18569_modified"
'18569_modified'
>>>
>>> # change submission date
>>> starfile["save_entry_information"]["Entry.Submission_date"]
'2012-07-05'
>>>
>>> starfile["save_entry_information"]["Entry.Submission_date"] = "2015-07-05
↪"
'2015-07-05'
>>>
```

- Printing a *StarFile* and its components (*saveframe* and *loop* data):

```
>>> starfile.print_starfile(file_format="nmrstar")
data_18569
save_entry_information
    _Entry.Sf_category     entry_information
    _Entry.Sf_framecode    entry_information
    _Entry.ID      18569
...
>>>
>>> starfile.print_starfile(file_format="json")
{
 "data": "18569",
 "save_entry_information": {
     "Entry.Sf_category": "entry_information",
     "Entry.Sf_framecode": "entry_information",
     "Entry.ID": "18569",
 ...
}
>>>
>>> starfile.print_saveframe("save_entry_information", file_format=
↪"nmrstar")
_Entry.Sf_category         entry_information
_Entry.Sf_framecode        entry_information
_Entry.ID  18569
_Entry.Title
;
13C, 15N and 1H backbone and sidechain assignments of the
ENA-VASP homology 1 (EVH1) domain of the human
vasodilator-stimulated phosphoprotein (VASP)
;
_Entry.Type        .
_Entry.Version_type        original
_Entry.Submission_date     2012-07-05
_Entry.Accession_date      2012-07-05
_Entry.Last_release_date   2012-07-18
_Entry.Original_release_date       2012-07-18
_Entry.Origination         author
_Entry.NMR_STAR_version    3.1.1.61
_Entry.Original_NMR_STAR_version   3.1
_Entry.Experimental_method         NMR
_Entry.Experimental_method_subtype         solution
_Entry.Details     'ANSIG v3.3 exported crosspeaks file'
```

```
_Entry.BMRB_internal_directory_name        .
...
>>>
>>> starfile.print_saveframe("save_entry_information", file_format="json
↪")
{
    "Entry.Sf_category": "entry_information",
    "Entry.Sf_framecode": "entry_information",
    "Entry.ID": "18569",
    "Entry.Title": ";\n13C, 15N and 1H backbone and sidechain␣
↪assignments of the
                    ENA-VASP homology 1 (EVH1) domain of the human
                    vasodilator-stimulated phosphoprotein (VASP)\n;",
    "Entry.Type": ".",
    "Entry.Version_type": "original",
    "Entry.Submission_date": "2012-07-05",
    "Entry.Accession_date": "2012-07-05",
    "Entry.Last_release_date": "2012-07-18",
    "Entry.Original_release_date": "2012-07-18",
    "Entry.Origination": "author",
    "Entry.NMR_STAR_version": "3.1.1.61",
    "Entry.Original_NMR_STAR_version": "3.1",
    "Entry.Experimental_method": "NMR",
    "Entry.Experimental_method_subtype": "solution",
    "Entry.Details": "'ANSIG v3.3 exported crosspeaks file'",
    "Entry.BMRB_internal_directory_name": ".",
    ...
}
>>>
>>> starfile.print_loop("save_entry_information", "loop_1", file_format=
↪"nmrstar")
_Data_set.Type
_Data_set.Count
_Data_set.Entry_ID
assigned_chemical_shifts 1 18569
spectral_peak_list 1 18569
>>>
>>> starfile.print_loop("save_entry_information", "loop_1", file_format=
↪"json")
[
    [
        "Data_set.Type",
        "Data_set.Count",
        "Data_set.Entry_ID"
    ],
    [
        {
            "Data_set.Type": "assigned_chemical_shifts",
            "Data_set.Count": "1",
            "Data_set.Entry_ID": "18569"
        },
        {
            "Data_set.Type": "spectral_peak_list",
            "Data_set.Count": "1",
            "Data_set.Entry_ID": "18569"
        }
    ]
]
```

```
>>>
```

- Accessing chemical shift data:

Chemical shift data can be accessed using bracket accessors as described above using a *saveframe* name and *loop* name:

```
>>> starfile["save_assigned_chem_shift_list_1"]["loop_1"][0]
['Atom_chem_shift.ID', 'Atom_chem_shift.Assembly_atom_ID',
 'Atom_chem_shift.Entity_assembly_ID', 'Atom_chem_shift.Entity_ID',
 'Atom_chem_shift.Comp_index_ID', 'Atom_chem_shift.Seq_ID',
 'Atom_chem_shift.Comp_ID', 'Atom_chem_shift.Atom_ID',
 'Atom_chem_shift.Atom_type', 'Atom_chem_shift.Atom_isotope_number',
 'Atom_chem_shift.Val', 'Atom_chem_shift.Val_err',
 'Atom_chem_shift.Assign_fig_of_merit', 'Atom_chem_shift.Ambiguity_code',
 'Atom_chem_shift.Occupancy', 'Atom_chem_shift.Resonance_ID',
 'Atom_chem_shift.Auth_entity_assembly_ID', 'Atom_chem_shift.Auth_asym_ID',
 'Atom_chem_shift.Auth_seq_ID', 'Atom_chem_shift.Auth_comp_ID',
 'Atom_chem_shift.Auth_atom_ID', 'Atom_chem_shift.Details',
 'Atom_chem_shift.Entry_ID', 'Atom_chem_shift.Assigned_chem_shift_list_ID']
>>>
>>> starfile["save_assigned_chem_shift_list_1"]["loop_1"][1][0]["Atom_chem_
↪shift.Seq_ID"]
'1'
>>> starfile["save_assigned_chem_shift_list_1"]["loop_1"][1][0]["Atom_chem_
↪shift.Comp_ID"]
'MET'
>>> starfile["save_assigned_chem_shift_list_1"]["loop_1"][1][0]["Atom_chem_
↪shift.Atom_ID"]
'H'
>>> starfile["save_assigned_chem_shift_list_1"]["loop_1"][1][0]["Atom_chem_
↪shift.Val"]
'8.55'
>>> starfile["save_assigned_chem_shift_list_1"]["loop_1"][1][1]["Atom_chem_
↪shift.Atom_ID"]
'HA'
>>> starfile["save_assigned_chem_shift_list_1"]["loop_1"][1][1]["Atom_chem_
↪shift.Val"]
'4.548'
>>> starfile["save_assigned_chem_shift_list_1"]["loop_1"][1][2]["Atom_chem_
↪shift.Atom_ID"]
'HB2'
>>> starfile["save_assigned_chem_shift_list_1"]["loop_1"][1][2]["Atom_chem_
↪shift.Val"]
'1.994'
>>>
```

Also the *StarFile* class provides a *chem_shifts_by_residue()* method that organizes chemical shits into a `list` of `collections.OrderedDict` data structures (*keys* - sequence id, *values* - chemical shift data) - one for each protein chain, if multiple chains are present within the file:

```
>>> starfile.chem_shifts_by_residue()
[OrderedDict([
    ('1', OrderedDict([('AA3Code', 'MET'),
                       ('Seq_ID', '1'),
                       ('H', '8.55'),
                       ('HA', '4.548'),
                       ('HB2', '1.994'),
```

```
                        ('HB3', '2.118'),
                        ('CA', '55.489'),
                        ('CB', '32.848'),
                        ('N', '122.221')])),
    ('2', OrderedDict([('AA3Code', 'SER'),
                        ('Seq_ID', '2'),
                        ('H', '8.225'),
                        ('HA', '4.420'),
                        ('HB2', '3.805'),
                        ('HB3', '3.857'),
                        ('CA', '58.593'),
                        ('CB', '64.057'),
                        ('N', '117.197')])),
    ('3', OrderedDict([('AA3Code', 'GLU'),
                        ('Seq_ID', '3'),
                        ('H', '8.002'),
                        ('HA', '4.848'),
                        ('HB2', '1.852'),
                        ('HB3', '1.963'),
                        ('HG2', '1.981'),
                        ('HG3', '2.191'),
                        ('CA', '55.651'),
                        ('CB', '32.952'),
                        ('CG', '37.425'),
                        ('N', '119.833')])), ...
...
]
>>>
>>> starfile.chem_shifts_by_residue(amino_acids=["SER"], atoms=["CA", "CB"])
[OrderedDict([
    ('2', OrderedDict([('AA3Code', 'SER'),
                        ('Seq_ID', '2'),
                        ('CA', '58.593'),
                        ('CB', '64.057')])),
    ('8', OrderedDict([('AA3Code', 'SER'),
                        ('Seq_ID', '8'),
                        ('CA', '57.456'),
                        ('CB', '64.863')])),
    ('9', OrderedDict([('AA3Code', 'SER'),
                        ('Seq_ID', '9'),
                        ('CA', '57.852'),
                        ('CB', '67.332')])),
    ('34', OrderedDict([('AA3Code', 'SER'),
                         ('Seq_ID', '34'),
                         ('CA', '59.113'),
                         ('CB', '66.248')])),
    ('46', OrderedDict([('AA3Code', 'SER'),
                         ('Seq_ID', '46'),
                         ('CA', '55.939'),
                         ('CB', '66.829')])),
    ('95', OrderedDict([('AA3Code', 'SER'),
                        ('Seq_ID', '95'),
                        ('CA', '57.013'),
                        ('CB', '66.501')])),
    ('108', OrderedDict([('AA3Code', 'SER'),
                         ('Seq_ID', '108'),
                         ('CA', '61.617'),
                         ('CB', '62.493')]))])
```

```
]
>>>
```

Writing data from a StarFile object into a file

Data from a *StarFile* can be written into file in original NMR-STAR format or in equivalent JSON format using *write()*:

- Writing into a NMR-STAR formatted file:

```
>>> with open("bmr18569_modified.str", "w") as outfile:
...     starfile.write(outfile, file_format="nmrstar")
>>>
```

- Writing into a JSONized NMR-STAR formatted file:

```
>>> with open("bmr18569_modified.json", "w") as outfile:
...     starfile.write(outfile, file_format="json")
>>>
```

Converting NMR-STAR files

NMR-STAR files can be converted between the NMR-STAR file format and a JSONized NMR-STAR file format using *nmrstarlib.converter* and *nmrstarlib.translator* modules.

One-to-one file conversions

- Converting from the NMR-STAR file format into its equivalent JSON file format:

```
from nmrstarlib.converter import Converter
from nmrstarlib.translator import StarFileToStarFile

# Using valid BMRB id to access file from URL: from_path="18569"
converter = Converter(StarFileToStarFile(from_path="18569", to_path=
↪"bmr18569.json",
                                         from_format="nmrstar", to_format=
↪"json"))
converter.convert()
```

- Converting from JSON file format into its equivalent NMR-STAR file format:

```
from nmrstarlib.converter import Converter
from nmrstarlib.translator import StarFileToStarFile

converter = Converter(StarFileToStarFile(from_path="bmr18569.json", to_path=
↪"bmr18569.str",
                                         from_format="json", to_format=
↪"nmrstar"))
converter.convert()
```

Many-to-many files conversions

- Converting from the directory of NMR-STAR formatted files into its equivalent JSON formatted files:

```
from nmrstarlib.converter import Converter
from nmrstarlib.translator import StarFileToStarFile

converter = Converter(StarFileToStarFile(from_path="starfiles_dir_nmrstar",
                                         to_path="starfiles_dir_json",
                                         from_format="nmrstar",
                                         to_format="json"))
converter.convert()
```

- Converting from the directory of JSONized NMR-STAR formatted files into NMR-STAR formatted files:

```
from nmrstarlib.converter import Converter
from nmrstarlib.translator import StarFileToStarFile

converter = Converter(StarFileToStarFile(from_path="starfiles_dir_json",
                                         to_path="starfiles_dir_nmrstar",
                                         from_format="json",
                                         to_format="nmrstar"))
converter.convert()
```

Note: Many-to-many files and one-to-one file conversions are available. See *nmrstarlib.converter* for full list of available conversions.

Creating simulated peak lists from NMR-STAR formatted files

Chemical shift values and assignment information deposited in NMR-STAR formatted files can be used to generate a large number of simulated peak lists for different types of solution and solid-state NMR experiments. Many different types of standard NMR experiments are defined in the *spectrum_description.json* configuration file. We will be using *HNcoCACB* spectrum type for the following examples.

- Creating a zero-variance *HNcoCACB* peak list file in *sparky*-like format from NMR-STAR formatted file:

```
from nmrstarlib.converter import Converter
from nmrstarlib.translator import StarFileToPeakList

# Using valid BMRB id to access file from URL: from_path="18569"
converter = Converter(StarFileToPeakList(from_path="18569", to_path="18569.
↪txt",
                                         from_format="nmrstar", to_format=
↪"sparky",
                                         spectrum_name="HNcoCACB"))
converter.convert()
```

The generated *18569.txt* peak list file should look like the following:

```
Assignment                      w1              w2              w3

```

```
GLN101H-GLN101N-ALA100CA          7.99            117.573          54.763
GLN101H-GLN101N-ALA100CB          7.99            117.573          18.2
PHE102H-PHE102N-GLN101CA          7.779           122.727          58.601
PHE102H-PHE102N-GLN101CB          7.779           122.727          28.439
ALA103H-ALA103N-PHE102CA          8.653           120.217          62.078
ALA103H-ALA103N-PHE102CB          8.653           120.217          40.21
ALA104H-ALA104N-ALA103CA          7.725           120.05           55.174
ALA104H-ALA104N-ALA103CB          7.725           120.05           18.25
GLY105H-GLY105N-ALA104CA          7.624           108.8            54.625
GLY105H-GLY105N-ALA104CB          7.624           108.8            17.714
...
```

- Creating a zero-variance *HNcoCACB* peak list file in *json* format from a NMR-STAR formatted file:

```
from nmrstarlib.converter import Converter
from nmrstarlib.translator import StarFileToPeakList

# Using valid BMRB id to access file from URL: from_path="18569"
converter = Converter(StarFileToPeakList(from_path="18569", to_path="18569.
↪json",
                                         from_format="nmrstar", to_format=
↪"json",
                                         spectrum_name="HNcoCACB"))
converter.convert()
```

The generated *18569.json* peak list file should look like the following:

```
[
 {"Assignment": ["GLN101H", "GLN101N", "ALA100CA"], "Dimensions": [7.99, 117.
↪573, 54.763]},
 {"Assignment": ["GLN101H", "GLN101N", "ALA100CB"], "Dimensions": [7.99, 117.
↪573, 18.2]},
 {"Assignment": ["PHE102H", "PHE102N", "GLN101CA"], "Dimensions": [7.779,␣
↪122.727, 58.601]},
 {"Assignment": ["PHE102H", "PHE102N", "GLN101CB"], "Dimensions": [7.779,␣
↪122.727, 28.439]},
 {"Assignment": ["ALA103H", "ALA103N", "PHE102CA"], "Dimensions": [8.653,␣
↪120.217, 62.078]},
 {"Assignment": ["ALA103H", "ALA103N", "PHE102CB"], "Dimensions": [8.653,␣
↪120.217, 40.21]},
 {"Assignment": ["ALA104H", "ALA104N", "ALA103CA"], "Dimensions": [7.725,␣
↪120.05, 55.174]},
 {"Assignment": ["ALA104H", "ALA104N", "ALA103CB"], "Dimensions": [7.725,␣
↪120.05, 18.25]},
 {"Assignment": ["GLY105H", "GLY105N", "ALA104CA"], "Dimensions": [7.624,␣
↪108.8, 54.625]},
 {"Assignment": ["GLY105H", "GLY105N", "ALA104CB"], "Dimensions": [7.624,␣
↪108.8, 17.714]},
 ...
]
```

- Creating a *HNcoCACB* peak list file in *sparky*-like format and adding noise values to peak dimensions from a single source of variance, i.e. 100% of peaks will have chemical shift values adjusted using noise values from the defined random normal distribution:

```
from nmrstarlib.converter import Converter
from nmrstarlib.translator import StarFileToPeakList
from nmrstarlib.noise import RandomNormalNoiseGenerator
```

```
# create parameters dictionary for random normal distribution
parameters = {"H_mean": [0], "C_mean": [0], "N_mean": [0],
              "H_std": [0.001], "C_std": [0.01], "N_std": [0.01]}

# create random normal noise generator
random_normal_noise_generator = RandomNormalNoiseGenerator(parameters)

# Using valid BMRB id to access file from URL: from_path="18569"
converter = Converter(StarFileToPeakList(from_path="18569", to_path="18569.
↪txt",
                                         from_format="nmrstar", to_format=
↪"sparky",
                                         spectrum_name="HNcoCACB",
                                         noise_generator=random_normal_noise_
↪generator))
converter.convert()
```

The generated *18569.txt* peak list file should look like the following (note chemical shift values differences for peaks that belong to the same spin system):

```
Assignment                w1                      w2                      w3

GLN101H-GLN101N-ALA100CA  7.99181036128894       117.58020101990542      54.
↪766688495100205
GLN101H-GLN101N-ALA100CB  7.990954825305333      117.56015058662396      18.
↪210671036513453
PHE102H-PHE102N-GLN101CA  7.778922720297377      122.72338031497752      58.
↪611256298615515
PHE102H-PHE102N-GLN101CB  7.779649007770076      122.73158449084175      28.
↪44847867136174
ALA103H-ALA103N-PHE102CA  8.655268275687266      120.21203154731162      62.
↪06020052346133
ALA103H-ALA103N-PHE102CB  8.652429780474138      120.24028818390909      40.
↪20871454076629
ALA104H-ALA104N-ALA103CA  7.726032805261596      120.0465086439804       55.
↪17721632833778
ALA104H-ALA104N-ALA103CB  7.723707420058092      120.05857764146538      18.
↪238751461431125
GLY105H-GLY105N-ALA104CA  7.624245263765136      108.79128538124017      54.
↪635163676566144
GLY105H-GLY105N-ALA104CB  7.622852045357025      108.80169379890037      17.
↪712279254335343
```

- Creating a *HNcoCACB* peak list file in *sparky*-like format and adding noise values to *H* and *N* peak dimensions but not *C* peak dimension from a single source of variance, i.e. 100% of peaks will have chemical shift values adjusted using noise values from the defined random normal distribution:

```
from nmrstarlib.converter import Converter
from nmrstarlib.translator import StarFileToPeakList
from nmrstarlib.noise import RandomNormalNoiseGenerator

# create parameters dictionary for random normal distribution
parameters = {"H_mean": [0], "C_mean": [0], "N_mean": [0],
              "H_std": [0.001], "C_std": [0], "N_std": [0.01]}

# create random normal noise generator
random_normal_noise_generator = RandomNormalNoiseGenerator(parameters)
```

```
# Using valid BMRB id to access file from URL: from_path="18569"
converter = Converter(StarFileToPeakList(from_path="18569", to_path="18569.
↪txt",
                                          from_format="nmrstar", to_format=
↪"sparky",
                                          spectrum_name="HNcoCACB",
                                          noise_generator=random_normal_noise_
↪generator))
converter.convert()
```

The generated *18569.txt* peak list file should look like the following (note the chemical shift values differences in *H* and *N* dimensions for peaks that belong to the same spin system):

```
Assignment                w1                      w2                      w3

GLN101H-GLN101N-ALA100CA  7.989218253134929       117.57857910858431      54.
↪763
GLN101H-GLN101N-ALA100CB  7.990510131020589       117.56823569837354      18.
↪2
PHE102H-PHE102N-GLN101CA  7.7779627574724515      122.73008978861516      58.
↪601
PHE102H-PHE102N-GLN101CB  7.7793926308291415      122.72231923701418      28.
↪439
ALA103H-ALA103N-PHE102CA  8.652917582133883       120.2346126620952       62.
↪078
ALA103H-ALA103N-PHE102CB  8.653219672092492       120.2181374169753       40.
↪21
ALA104H-ALA104N-ALA103CA  7.725520756033144       120.06129459019358      55.
↪174
ALA104H-ALA104N-ALA103CB  7.724202440439531       120.07284401661603      18.
↪25
GLY105H-GLY105N-ALA104CA  7.62504474457142        108.78804954461619      54.
↪625
GLY105H-GLY105N-ALA104CB  7.623712745737121       108.79003993468108      17.
↪714
```

- Creating a *HNcoCACB* peak list file in *sparky*-like format and adding noise values to peak dimensions from two sources of variance, i.e. chemical shift values will be adjusted using noise values from two random normal distributions. In order to specify two sources of variance, we need to provide how we want to split our peak list and provide statistical distribution parameters for both distributions. Let's say we want 70 % of peaks to have a smaller variance in *H* and *N* dimensions and 30 % of peaks to have a larger variance in *H* and *N* dimensions:

```
from nmrstarlib.converter import Converter
from nmrstarlib.translator import StarFileToPeakList
from nmrstarlib.noise import RandomNormalNoiseGenerator

# create parameters dictionary for random normal distribution
parameters = {"H_mean": [0, 0], "C_mean": [0, 0], "N_mean": [0, 0],
              "H_std": [0.001, 0.005], "C_std": [0, 0], "N_std": [0.01, 0.
↪05]}

# create random normal noise generator
noise_generator = RandomNormalNoiseGenerator(parameters)

# Using valid BMRB id to access file from URL: from_path="18569"
```

```
converter = Converter(StarFileToPeakList(from_path="18569", to_path="18569.
↪txt",
                                          from_format="nmrstar", to_format=
↪"sparky",
                                          spectrum_name="HNcoCACB",
                                          plsplit=(70,30),
                                          noise_generator=noise_generator))
converter.convert()
```

The generated *18569.txt* peak list file should look like the following (note the larger variance in the last four peaks especially in *N* dimension):

```
Assignment                w1                      w2                      w3

GLN101H-GLN101N-ALA100CA  7.989176427887494       117.57288229702456      54.
↪763
GLN101H-GLN101N-ALA100CB  7.989740864521572       117.5707086547982       18.
↪2
PHE102H-PHE102N-GLN101CA  7.780597368680522       122.7357007073057       58.
↪601
PHE102H-PHE102N-GLN101CB  7.777390419829074       122.72723395076358      28.
↪439
ALA103H-ALA103N-PHE102CA  8.651374019487395       120.22156605272194      62.
↪078
ALA103H-ALA103N-PHE102CB  8.654384679162527       120.23197871710906      40.
↪21
...
ASP98H-ASP98N-GLU97CA     7.869384379707692       120.72766991387383      60.
↪601
ASP98H-ASP98N-GLU97CB     7.872260831124177       120.66960671379097      28.
↪533
ALA99H-ALA99N-ASP98CA     7.1803123000354026      122.76636174425305      57.
↪799
ALA99H-ALA99N-ASP98CB     7.187801610413494       122.83147347445296      42.
↪138
```

Spectrum description configuration file

Spectrum description configuration file (*spectrum_description.json*) contains descriptions for standard solution and solid-state NMR experiments.

- List all available experiments:

```
>>> from nmrstarlib import nmrstarlib
>>> nmrstarlib.list_spectrums()
CANCO
CANCOCX
CBCANH
CBCAcoNH
CCcoNH
HBHAcoNH
HNCA
HNCACB
HNCO
HNcaCO
HNcoCA
HNcoCACB
```

```
HSQC
HccoNH
NCA
NCACX
NCO
NCOCX
>>>
```

- List all available spectrum descriptions:

```
>>> from nmrstarlib import nmrstarlib
>>> nmrstarlib.list_spectrum_descriptions()
{'CANCO': {'Labels': ['CA', 'N', 'CO-1'],
           'MinNumberPeaksPerSpinSystem': 1,
           'PeakDescriptions': [{'dimensions': ['CA', 'N', 'CO-1'], 'fraction
↪': 1}]},
 'CANCOCX': {'Labels': ['CA', 'N', 'CO-1', 'CX-1'],
             'MinNumberPeaksPerSpinSystem': 2,
             'PeakDescriptions': [
                 {'dimensions': ['CA', 'N', 'CO-1', 'CO-1'], 'fraction': 1},
                 {'dimensions': ['CA', 'N', 'CO-1', 'CA-1'], 'fraction': 1},
                 {'dimensions': ['CA', 'N', 'CO-1', 'CB-1'], 'fraction': 1},
                 {'dimensions': ['CA', 'N', 'CO-1', 'CG-1'], 'fraction': 1},
                 {'dimensions': ['CA', 'N', 'CO-1', 'CD-1'], 'fraction': 1},
                 {'dimensions': ['CA', 'N', 'CO-1', 'CE-1'], 'fraction': 1},
                 {'dimensions': ['CA', 'N', 'CO-1', 'CZ-1'], 'fraction': 1}]}
↪, ...
}
>>>
```

- List specific spectrum descriptions:

```
>>> from nmrstarlib import nmrstarlib
>>> nmrstarlib.list_spectrum_descriptions("HNcoCACB", "NCACX")
{'HNcoCACB': {'Labels': ['H', 'N', 'CA/CB-1'],
              'MinNumberPeaksPerSpinSystem': 2,
              'PeakDescriptions': [
                  {'dimensions': ['H', 'N', 'CA-1'], 'fraction': 1},
                  {'dimensions': ['H', 'N', 'CB-1'], 'fraction': 0.95}]}}
{'NCACX': {'Labels': ['N', 'CA', 'CX'],
           'MinNumberPeaksPerSpinSystem': 2,
           'PeakDescriptions': [
               {'dimensions': ['N', 'CA', 'CO'], 'fraction': 1},
               {'dimensions': ['N', 'CA', 'CA'], 'fraction': 1},
               {'dimensions': ['N', 'CA', 'CB'], 'fraction': 1},
               {'dimensions': ['N', 'CA', 'CG'], 'fraction': 1},
               {'dimensions': ['N', 'CA', 'CD'], 'fraction': 1},
               {'dimensions': ['N', 'CA', 'CE'], 'fraction': 1},
               {'dimensions': ['N', 'CA', 'CZ'], 'fraction': 1}]}}
>>>
```

- Adding a custom experiment description and simulating peak list based on it. Custom spectrum description can be added in several ways:
  - By creating additional json configuration with spectrum description and updating *SPECTRUM_DESCRIPTIONS* `dict`. Content of *custom_spectrum_description.json*:

```
{
    "NCACX_custom": {
        "Labels": ["N", "CA", "CX"],
        "MinNumberPeaksPerSpinSystem": 2,
        "PeakDescriptions": [
            {"fraction": 1, "dimensions": ["N", "CA", "CO"]},
            {"fraction": 1, "dimensions": ["N", "CA", "CA"]},
            {"fraction": 1, "dimensions": ["N", "CA", "CB"]},
            {"fraction": 1, "dimensions": ["N", "CA", "CG"]},
            {"fraction": 1, "dimensions": ["N", "CA", "CO-1"]},
            {"fraction": 1, "dimensions": ["N", "CA", "CA-1"]}]
    }
}
```

```
from nmrstarlib import nmrstarlib
from nmrstarlib.converter import Converter
from nmrstarlib.translator import StarFileToPeakList
from nmrstarlib.noise import RandomNormalNoiseGenerator

# update SPECTRUM_DESCRIPTIONS
nmrstarlib.update_constants(spectrum_descriptions_cfg="path/to/
↪custom_spectrum_description.json")

# create parameters dictionary for random normal distribution
parameters = {"H_mean": [0, 0], "C_mean": [0, 0], "N_mean": [0,␣
↪0],
              "H_std": [0, 0], "C_std": [0.01, 0.05], "N_std": [0.
↪01, 0.05]}

# create random normal noise generator
random_normal_noise_generator =␣
↪RandomNormalNoiseGenerator(parameters)

converter = Converter(StarFileToPeakList(from_path="18569", to_
↪path="18569.txt",
                                          from_format="nmrstar",␣
↪to_format="sparky",
                                          spectrum_name="NCACX_
↪custom",
                                          plsplit=(70,30),
                                          noise_generator=random_
↪normal_noise_generator))
converter.convert()
```

– By defining dictionary with new spectrum description and updating *SPECTRUM_DESCRIPTIONS* `dict`.

```
from nmrstarlib import nmrstarlib
from nmrstarlib.converter import Converter
from nmrstarlib.translator import StarFileToPeakList
from nmrstarlib.noise import RandomNormalNoiseGenerator

custom_experiment_type = {
   "NCACX_custom": {
      "Labels": ["N", "CA", "CX"],
      "MinNumberPeaksPerSpinSystem": 2,
      "PeakDescriptions": [
```

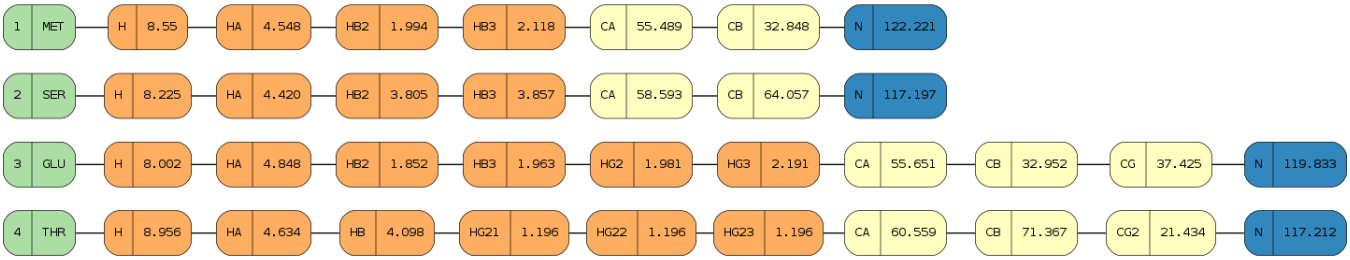
```
      {"fraction": 1, "dimensions": ["N", "CA", "CO"]},
      {"fraction": 1, "dimensions": ["N", "CA", "CA"]},
      {"fraction": 1, "dimensions": ["N", "CA", "CB"]},
      {"fraction": 1, "dimensions": ["N", "CA", "CG"]},
      {"fraction": 1, "dimensions": ["N", "CA", "CO-1"]},
      {"fraction": 1, "dimensions": ["N", "CA", "CA-1"]}]
   }
}

# update SPECTRUM_DESCRIPTION
nmrstarlib.SPECTRUM_DESCRIPTIONS.update(custom_experiment_type)

# create parameters dictionary for random normal distribution
parameters = {"H_mean": [0, 0], "C_mean": [0, 0], "N_mean": [0,␣
↪0],
              "H_std": [0.001, 0.005], "C_std": [0, 0], "N_std":␣
↪[0.01, 0.05]}

# create random normal noise generator
random_normal_noise_generator =␣
↪RandomNormalNoiseGenerator(parameters)

# Using valid BMRB id to access file from URL: from_path="18569"
converter = Converter(StarFileToPeakList(from_path="18569", to_
↪path="18569.txt",
                                         from_format="nmrstar",␣
↪to_format="sparky",
                                         spectrum_name="NCACX_
↪custom",
                                         plsplit=(70,30),
                                         noise_generator=random_
↪normal_noise_generator))
converter.convert()
```

Visualizing chemical shifts values

Chemical shifts values can be visualized using the *nmrstarlib.csviewer* Chemical Shifts Viewer module.

```
>>> from nmrstarlib.csviewer import CSViewer
>>>
>>> csviewer = CSViewer(from_path="18569", filename="18569_chem_shifts_all", csview_
↪format="png")
>>> csviewer.csview(view=True)
>>>
>>> csviewer = CSViewer(from_path="18569", amino_acids=["SER", "THR"], atoms=["CA",
↪"CB"],
...                     filename="18569_chem_shifts_SER_THR_CA_CB", csview_format="png
↪")
>>> csviewer.csview(view=True)  # open in a default image viewer or pdf viewer
>>> csviewer.csview(view=False) # save output file in current working directory
>>>
```

nmrstarlib.csviewer output example:



Command Line Interface

Command Line Interface functionality:

- Convert from the NMR-STAR file format into its equivalent JSON file format and vice versa.
- Create simulated peak list files using chemical shift and assignment information.
- Visualize assigned chemical shift values.

```
nmrstarlib command-line interface

Usage:
    nmrstarlib -h | --help
    nmrstarlib --version
    nmrstarlib convert (<from_path> <to_path>) [--from_format=<format>]
                                               [--to_format=<format>]
                                               [--bmrb_url=<url>]
                                               [--nmrstar_version=<version>]
                                               [--verbose]
    nmrstarlib csview <starfile_path> [--amino_acids=<aa>]
                                      [--atoms=<at>]
                                      [--csview_outfile=<path>]
                                      [--csview_format=<format>]
                                      [--bmrb_url=<url>]
                                      [--nmrstar_version=<version>]
                                      [--verbose]
    nmrstarlib plsimulate (<from_path> <to_path> <spectrum>) [--from_format=<format>]
                                                             [--to_plformat=<format>]
                                                             [--split=<%>]
                                                             [--H_std=<std>]
                                                             [--C_std=<std>]
                                                             [--N_std=<std>]
                                                             [--H_mean=<mean>]
                                                             [--C_mean=<mean>]
                                                             [--N_mean=<mean>]
                                                             [--bmrb_url=<url>]
                                                             [--nmrstar_version=
↪<version>]
                                                             [--verbose]

Options:
    -h, --help                      Show this screen.
    --version                       Show version.
    --verbose                       Print what files are processing.
    --from_format=<format>          Input file format, available formats: nmrstar,␣
↪json
                                    [default: nmrstar].
```

```
    --to_format=<format>            Output file format, available formats: nmrstar,␣
↪json
                                    [default: json].
    --nmrstar_version=<version>     Version of NMR-STAR format to use, available: 2, 3
                                    [default: 3].
    --bmrb_url=<url>                URL to BMRB REST interface
                                    [default: http://rest.bmrb.wisc.edu/bmrb/NMR-
↪STAR3/].
    --amino_acids=<aa>              Comma-separated amino acid three-letter codes.
    --atoms=<at>                    Comma-separated BMRB atom codes.
    --csview_outfile=<path>         Where to save chemical shifts table.
    --csview_format=<format>        Format to which save chamical shift table
                                    [default: svg].
    --split=<%>                     How to split peak list into chunks by percent␣
↪[default: 100].
    --H_std=<ppm>                   Standard deviation for H dimensions [default: 0].
    --C_std=<ppm>                   Standard deviation for C dimensions [default: 0].
    --N_std=<ppm>                   Standard deviation for N dimensions [default: 0].
    --H_mean=<ppm>                  Mean for H dimensions [default: 0].
    --C_mean=<ppm>                  Mean for C dimensions [default: 0].
    --N_mean=<ppm>                  Mean for N dimensions [default: 0].
    --spectrum_descriptions=<path>  Path to custom spectrum descriptions file.
```

Converting NMR-STAR files in bulk

One-to-one file conversions

- Convert from a local file in NMR-STAR format to a local file in JSON format:

```
$ python3 -m nmrstarlib convert bmr18569.str bmr18569.json \
          --from_format=nmrstar --to_format=json
```

- Convert from a local file in JSON format to a local file in NMR-STAR format:

```
$ python3 -m nmrstarlib convert bmr18569.json bmr18569.str \
          --from_format=json --to_format=nmrstar
```

- Convert from a compressed local file in NMR-STAR format to a compressed local file in JSON format:

```
$ python3 -m nmrstarlib convert bmr18569.str.gz bmr18569.json.gz \
          --from_format=nmrstar --to_format=json
```

- Convert from a compressed local file in JSON format to a compressed local file in NMR-STAR format:

```
$ python3 -m nmrstarlib convert bmr18569.json.gz bmr18569.str.gz \
          --from_format=json --to_format=nmrstar
```

- Convert from a uncompressed URL file in NMR-STAR format to a compressed local file in JSON format:

```
$ python3 -m nmrstarlib convert 18569 bmr18569.json.bz2 \
          --from_format=nmrstar --to_format=json
```

Note: See *nmrstarlib.converter* for full list of available conversions.

Many-to-many files conversions

- Convert from a directory of files in NMR-STAR format to a directory of files in JSON format:

```
$ python3 -m nmrstarlib convert starfiles_dir_nmrstar starfiles_dir_json \
          --from_format=nmrstar --to_format=json
```

- Convert from a directory of files in JSON format to a directory of files in NMR-STAR format:

```
$ python3 -m nmrstarlib convert starfiles_dir_json starfiles_dir_nmrstar \
          --from_format=json --to_format=nmrstar
```

- Convert from a directory of files in NMR-STAR format to a zip archive of files in JSON format:

```
$ python3 -m nmrstarlib convert starfiles_dir_nmrstar starfiles_json.zip \
          --from_format=nmrstar --to_format=json
```

- Convert from a compressed tar archive of files in JSON format to a directory of files in NMR-STAR format:

```
$ python3 -m nmrstarlib convert starfiles_json.tar.gz starfiles_dir_nmrstar \
          --from_format=json --to_format=nmrstar
```

- Convert from a zip archive of files in NMR-STAR format to a compressed tar archive of files in JSON format:

```
$ python3 -m nmrstarlib convert starfiles_nmrstar.zip starfile_json.tar.bz2 \
          --from_format=nmrstar --to_format=json
```

Note: See *nmrstarlib.converter* for full list of available conversions.

Creating simulated peak list files from NMR-STAR files in bulk

One-to-one file simulations

- Creating a zero-variance *HNcoCACB* peak list file in *sparky*-like format from local NMR-STAR formatted file (*bmr18569.txt*):

```
$ python3 -m nmrstarlib plsimulate bmr18569.txt 18569_peaklist.txt HNcoCACB \
          --from_format=nmrstar --to_format=sparky
```

- Creating a *HNcoCACB* peak list file in *sparky*-like format and adding noise values to peak dimensions from a single source of variance, i.e. 100% of peaks will have chemical shift values adjusted using noise values from the defined random normal distribution (note that we can use *18569* BMRB id instead of local file):

```
$ python3 -m nmrstarlib plsimulate 18569 18569_peaklist.txt HNcoCACB \
          --from_format=nmrstar --to_format=sparky \
          --H_std=0.001 --N_std=0.01 --C_std=0.01
```

- Creating a *HNcoCACB* peak list file in *sparky*-like format and adding noise values to *H* and *N* peak dimensions but not *C* peak dimension from a single source of variance, i.e. 100% of peaks will have chemical shift values adjusted using noise values from the defined random normal distribution (note that we can use compressed *bmr18569.str.gz* file):

```
$ python3 -m nmrstarlib plsimulate bmr18569.str.gz 18569_peaklist.txt␣
↪HNcoCACB \
          --from_format=nmrstar --to_format=sparky \
          --H_std=0.001 --N_std=0.01
```

- Creating a *HNcoCACB* peak list file in *sparky*-like format and adding noise values to peak dimensions from two sources of variance, i.e. chemical shift values will be adjusted using noise values from two random normal distributions. In order to specify two sources of variance, we need to provide how we want to split our peak list and provide statistical distribution parameters for both distributions. Let's say we want 70 % of peaks to have a smaller variance in *H* and *N* dimensions and 30 % of peaks to have a larger variance in *H* and *N* dimensions:

```
$ python3 -m nmrstarlib plsimulate 18569 18569_peaklist.txt HNcoCACB \
          --from_format=nmrstar --to_format=sparky \
          --plsplit=70,30 --H_std=0.001,0.005 --N_std=0.01,0.05
```

Note: See *nmrstarlib.converter* for full list of available one-to-one and many-to-many input and output formats.

Many-to-many files simulations

- Simulate zero-variance *HNcoCACB* peak lists from a directory of NMR-STAR formatted files to a directory of peak list files:

```
$ python3 -m nmrstarlib plsimulate starfiles_dir peaklists_dir HNcoCACB \
          --from_format=nmrstar --to_format=sparky
```

- Simulate *HNcoCACB* peak lists from a directory of NMR-STAR formatted files to a zip archive of peak list files, add random normal noise values to *H* and *N* peak dimensions:

```
$ python3 -m nmrstarlib plsimulate starfiles_dir peaklists.zip HNcoCACB \
          --from_format=nmrstar --to_format=sparky --H_std=0.001 --N_std=0.01
```

- Simulate *NCACX* peak lists from a directory of NMR-STAR formatted files to a tar.gz archive of peak list files, add random normal noise values to *C* and *N* peak dimensions using two sources of variance, 70 % of peaks will have smaller variance, 30 % of peaks will have larger variance:

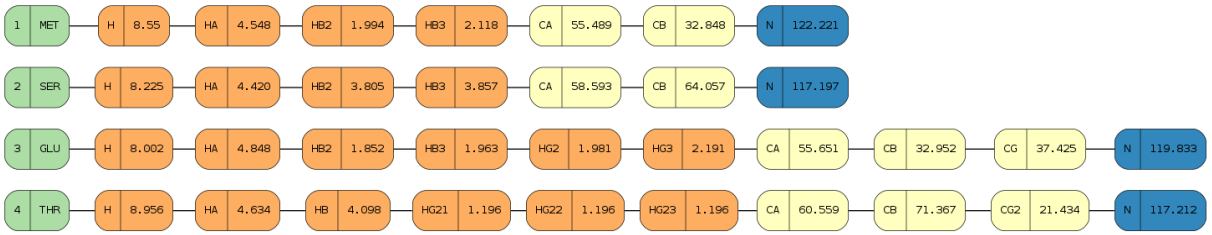
```
$ python3 -m nmrstarlib plsimulate starfiles_dir peaklists.tar.gz NCACX \
          --from_format=nmrstar --to_format=sparky --plsplit=70,30
          --C_std=0.01,0.05 --N_std=0.01,0.07
```

Note: See *nmrstarlib.converter* for full list of available one-to-one and many-to-many input and output formats.

Visualizing chemical shift values

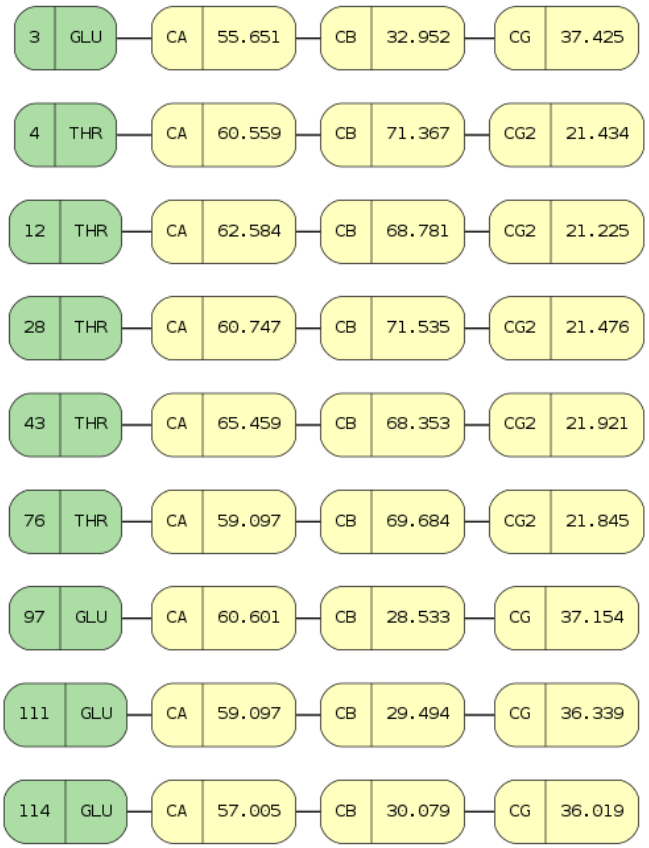
- Visualize chemical shift values for the entire sequence:

```
$ python3 -m nmrstarlib csview 18569 \
          --csview_outfile=18569_chem_shifts_all --csview_format=png
```



- Visualize *CA*, *CB*, *CG*, and *CG2* chemical shift values for *GLU* and *THR* amino acid residues:

```
$ python3 -m nmrstarlib csview 18569 \
          --amino_acids=GLU,THR --atoms=CA,CB,CG,CG2 \
          --csview_outfile=18569_chem_shifts_GLU_THR_CA_CB_CG_CG2 \
          --csview_format=png
```



The nmrstarlib API Reference

Routines for working with BMRB NMR-STAR format files.

This package includes the following modules:

nmrstarlib This module provides the *StarFile* class which is a python dictionary representation of a BMRB NMR-STAR file. Data can be accessed directly from the *StarFile* instance using bracket accessors. The *nmrstarlib* module relies on the *bmrblex* module for processing of tokens.

bmrblex This module provides the *bmrblex()* generator that is responsible for the syntax analysis of BMRB NMR-STAR files, processing word, number, single quoted, double quoted, multiline quoted BMRB tokens.

converter This module provides the *Converter* class that is responsible for the conversion of NMR-STAR formated files.

csviewer This module provides the *CSViewer* class that visualizes chemical shift values using the Graphviz (http://www.graphviz.org/) DOT Languge description and provides code example for utilizing the library.

noise This module provides the *NoiseGenerator* abstract class and derived *RandomNormalNoiseGenerator* for adding random normal noise values to peaks in simulated peak list.

plsimulator This module provides necessary interfaces in order to create a simulated *PeakList*.

translator This module provides *StarFileToStarFile* for conversion between NMR-STAR and JSONized NMR-STAR formatted files and *StarFileToPeakList* for conversion of NMR-STAR formatted files into peak list files using chemical shift values and assignment information.

nmrstarlib.nmrstarlib

This module provides the *StarFile* class that stores the data from a single NMR-STAR file in the form of an `OrderedDict`. Data can be accessed directly from the *StarFile* instance using bracket accessors.

The NMR-STAR format is a hierachical dictionary containing data on NMR experiments. The data is divided into a series of "saveframes" which each contain a number of key-value pairs and "loops".

Each saveframe has a unique name, which is used as the key in the dictionary, corresponding to another dictionary containing the information in the saveframe. Since loops in NMR-Star format do not have names, the keys for them inside the saveframe dictionary are simply loop_0, loop_1, etc.

nmrstarlib.nmrstarlib.**update_constants**(*nmrstar2cfg=''*, *nmrstar3cfg=''*, *resonance_classes_cfg=''*, *spectrum_descriptions_cfg=''*)

Update constant variables.

Returns None

Return type `None`

class nmrstarlib.nmrstarlib.**StarFile**(*source=''*, *frame_categories=None*, **args*, ***kwds*)

StarFile class that stores the data from a single NMR-STAR file in the form of an `OrderedDict`.

__init__(*source=''*, *frame_categories=None*, **args*, ***kwds*)

StarFile initializer. Leave *frame_categories* as `None` to read everything. Otherwise it can be a list of saveframe categories to read, skipping the rest.

Parameters

- **source** (*str*) – Source *StarFile* instance was created from - local file or URL address.
- **frame_categories** (*list*) – List of saveframe names.

read(*filehandle*)

Read data into a *StarFile* instance.

Parameters **filehandle** (io.TextIOWrapper, gzip.GzipFile, bz2.BZ2File, zipfile.ZipFile) – file-like object.

Returns None

Return type None

write(*filehandle*, *file_format*)
Write *StarFile* data into file.

Parameters

- **filehandle** (io.TextIOWrapper) – file-like object.
- **file_format** (*str*) – Format to use to write data: *nmrstar* or *json*.

Returns None

Return type None

writestr(*file_format*)
Write *StarFile* data into string.

Parameters **file_format** (*str*) – Format to use to write data: *nmrstar* or *json*.

Returns String representing the *StarFile* instance.

Return type str

_build_starfile(*nmrstar_str*)
Build *StarFile* object.

Parameters **nmrstar_str** (str or bytes) – NMR-STAR-formatted string.

Returns instance of *StarFile*.

Return type *StarFile*

_build_saveframe(*lexer*)
Build NMR-STAR file saveframe.

Parameters **lexer** (*bmrblex()*) – instance of the BMRB lexical analyzer.

Returns Saveframe dictionary.

Return type collections.OrderedDict

_build_loop(*lexer*)
Build saveframe loop.

Parameters **lexer** (*bmrblex()*) – instance of BMRB lexical analyzer.

Returns Fields and values of the loop.

Return type tuple

print_starfile(*f=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *file_format='nmrstar'*, *tw=3*)
Print *StarFile* into a file or stdout.

Parameters

- **f** (*io.StringIO*) – writable file-like stream.
- **file_format** (*str*) – Format to use: *nmrstar* or *json*.
- **tw** (*int*) – Tab width.

Returns None

Return type None

print_saveframe(*sf*, *f=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *file_format='nmrstar'*, *tw=3*)
Print saveframe into a file or stdout. We need to keep track of how far over everything is tabbed. The "tab width" variable tw does this for us.

Parameters

- **sf** (*str*) – Saveframe name.
- **f** (*io.StringIO*) – writable file-like stream.
- **file_format** (*str*) – Format to use: *nmrstar* or *json*.
- **tw** (*int*) – Tab width.

Returns None

Return type None

print_loop(*sf*, *sftag*, *f=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *file_format='nmrstar'*, *tw=3*)
Print loop into a file or stdout.

Parameters

- **sf** (*str*) – Saveframe name.
- **sftag** (*str*) – Saveframe tag, i.e. field name.
- **f** (*io.StringIO*) – writable file-like stream.
- **file_format** (*str*) – Format to use: *nmrstar* or *json*.
- **tw** (*int*) – Tab width.

Returns None

Return type None

_to_json()
Save *StarFile* into JSON string.

Returns JSON string.

Return type str

_to_nmrstar()
Save *StarFile* NMR-STAR format string.

Returns NMR-STAR string.

Return type str

_skip_saveframe(*lexer*)
Skip entire saveframe - keep emitting tokens until the end of saveframe.

Parameters **lexer** (*bmrblex*) – instance of the BMRB lexical analyzer class.

Returns None

Return type None

static _is_nmrstar(*string*)
Test if input string is in NMR-STAR format.

Parameters **string** (str or bytes) – Input string.

Returns Input string if in NMR-STAR format or False otherwise.

Return type `str` or `False`

static _is_json(*string*)
Test if input string is in JSON format.

Parameters **string** (`str` or `bytes`) – Input string.

Returns Input string if in JSON format or False otherwise.

Return type `str` or `False`

chem_shifts_by_residue(*amino_acids=None*, *atoms=None*, *amino_acids_and_atoms=None*, *nmrstar_version='3'*)
Organize chemical shifts by amino acid residue.

Parameters

- **amino_acids** (*list*) – List of amino acids three-letter codes.
- **atoms** (*list*) – List of BMRB atom type codes.
- **amino_acids_and_atoms** (*dict*) – Amino acid and its atoms key-value pairs.
- **nmrstar_version** (*str*) – Version of NMR-STAR format to use for look up chemical shifts loop.

Returns List of OrderedDict per each chain

Return type `list` of `collections.OrderedDict`

nmrstarlib.nmrstarlib.**_generate_filenames**(*sources*)
Generate filenames.

Parameters **sources** (*tuple*) – Sequence of strings representing path to file(s).

Returns Path to file(s).

Return type `str`

nmrstarlib.nmrstarlib.**_generate_handles**(*filenames*)
Open a sequence of filenames one at time producing file objects. The file is closed immediately when proceeding to the next iteration.

Parameters **filenames** (*generator*) – Generator object that yields the path to each file, one at a time.

Returns Filehandle to be processed into a *StarFile* instance.

nmrstarlib.nmrstarlib.**read_files**(**sources*)
Construct a generator that yields *StarFile* instances.

Parameters **sources** – One or more strings representing path to file(s).

Returns *StarFile* instance(s).

Return type *StarFile*

class nmrstarlib.nmrstarlib.**GenericFilePath**(*path*)
GenericFilePath class knows how to open local files or files over URL.

__init__(*path*)
Initialize path.

Parameters **path** (*str*) – String representing a path to local file(s) or valid URL address of file(s).

open()
Generator that opens and yields filehandles using appropriate facilities: test if path represents a local file or file over URL, if file is compressed or not.

Returns Filehandle to be processed into a *StarFile* instance.

static **is_compressed**(*path*)
Test if path represents compressed file(s).

Parameters **path** (*str*) – Path to file(s).

Returns String specifying compression type if compressed, "" otherwise.

Return type str

static **is_url**(*path*)
Test if path represents a valid URL.

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

Returns True if path is valid url string, False otherwise.

Return type True or False

__weakref__
list of weak references to the object (if defined)

nmrstarlib.nmrstarlib.**list_spectrums**()
List all available spectrum names that can be used for peak list simulation.

Returns None

Return type None

nmrstarlib.nmrstarlib.**list_spectrum_descriptions**(**args*)
List all available spectrum descriptions that can be used for peak list simulation.

Parameters **args** (*str*) – Spectrum name(s), e.g. list_spectrum_descriptions("HNCO", "HNcoCACB"), leave empty to list everything.

Returns None

Return type None

nmrstarlib.bmrblex

This module provides *bmrblex()* lexical analyzer for BMRB NMR-STAR format syntax. It is implemented as Python generator-based state machine which generates (yields) token one at a time when next() is invoked on *bmrblex()* instance.

Simplified description of parsing rules:

- Each word or number separated by whitespace characters is a separate BMRB token.
- Each single quoted (') string is a separate BMRB token, it should start with a single quote (') and end with a single quote *always* followed by whitespace character(s).
- Each double quoted (") string is a separate BMRB token, it should start with a double quote (") and end with a double quote *always* followed by whitespace character(s).
- Single quoted and double quoted strings have to be processed separately.
- Single quoted and double quoted strings are processed one character at a time.

- Multiline strings start with a semicolon *always* followed by new line character and ending with a semicolon *always* followed by whitespace character(s).
- Multiline strings are processed one line at a time.

Note:

- For a full description of NMR-STAR file format, see official documentation: http://www.bmrb.wisc.edu/dictionary/
- For a concise description of the NMR-STAR file format grammar see: https://github.com/mattfenwick/NMRPyStar#nmr-star-grammar

---

nmrstarlib.bmrblex.**bmrblex**(*text*)

A lexical analyzer for the BMRB NMR-STAR format syntax.

Parameters **text** (str or bytes) – Input text.

Returns Current token.

Return type str

nmrstarlib.converter

This module provides functionality for converting between the BMRB NMR-STAR format and its equivalent JSONized NMR-STAR format.

The following conversions are possible:

Local files:

- **One-to-one file conversions:**
  - textfile - to - textfile
  - textfile - to - textfile.gz
  - textfile - to - textfile.bz2
  - textfile.gz - to - textfile
  - textfile.gz - to - textfile.gz
  - textfile.gz - to - textfile.bz2
  - textfile.bz2 - to - textfile
  - textfile.bz2 - to - textfile.gz
  - textfile.bz2 - to - textfile.bz2
  - textfile / textfile.gz / textfile.bz2 - to - textfile.zip / textfile.tar / textfile.tar.gz / textfile.tar.bz2 (TypeError: One-to-many conversion)
- **Many-to-many files conversions:**
  - **Directories:**
    - directory - to - directory
    - directory - to - directory.zip
    - directory - to - directory.tar
    - directory - to - directory.tar.bz2

        * directory - to - directory.tar.gz
        * directory - to - directory.gz / directory.bz2 (TypeError: Many-to-one conversion)
    - **Zipfiles:**
        * zipfile.zip - to - directory
        * zipfile.zip - to - zipfile.zip
        * zipfile.zip - to - tarfile.tar
        * zipfile.zip - to - tarfile.tar.gz
        * zipfile.zip - to - tarfile.tar.bz2
        * zipfile.zip - to - directory.gz / directory.bz2 (TypeError: Many-to-one conversion)
    - **Tarfiles:**
        * tarfile.tar - to - directory
        * tarfile.tar - to - zipfile.zip
        * tarfile.tar - to - tarfile.tar
        * tarfile.tar - to - tarfile.tar.gz
        * tarfile.tar - to - tarfile.tar.bz2
        * tarfile.tar - to - directory.gz / directory.bz2 (TypeError: Many-to-one conversion)
        * tarfile.tar.gz - to - directory
        * tarfile.tar.gz - to - zipfile.zip
        * tarfile.tar.gz - to - tarfile.tar
        * tarfile.tar.gz - to - tarfile.tar.gz
        * tarfile.tar.gz - to - tarfile.tar.bz2
        * tarfile.tar.gz - to - directory.gz / directory.bz2 (TypeError: Many-to-one conversion)
        * tarfile.tar.bz2 - to - directory
        * tarfile.tar.bz2 - to - zipfile.zip
        * tarfile.tar.bz2 - to - tarfile.tar
        * tarfile.tar.bz2 - to - tarfile.tar.gz
        * tarfile.tar.bz2 - to - tarfile.tar.bz2
        * tarfile.tar.bz2 - to - directory.gz / directory.bz2 (TypeError: Many-to-one conversion)

URL files:

- **One-to-one file conversions:**
    - bmrbid - to - textfile
    - bmrbid - to - textfile.gz
    - bmrbid - to - textfile.bz2
    - bmrbid - to - textfile.zip / textfile.tar / textfile.tar.gz / textfile.tar.bz2 (TypeError: One-to-many conversion)
    - textfileurl - to - textfile

  - textfileurl - to - textfile.gz
  - textfileurl - to - textfile.bz2
  - textfileurl.gz - to - textfile
  - textfileurl.gz - to - textfile.gz
  - textfileurl.gz - to - textfile.bz2
  - textfileurl.bz2 - to - textfile
  - textfileurl.bz2 - to - textfile.gz
  - textfileurl.bz2 - to - textfile.bz2
  - textfileurl / textfileurl.gz / textfileurl.bz2 - to - textfile.zip / textfile.tar / textfile.tar.gz / textfile.tar.bz2 (TypeError: One-to-many conversion)

- **Many-to-many files conversions:**
  - **Zipfiles:**
    - zipfileurl.zip - to - directory
    - zipfileurl.zip - to - zipfile.zip
    - zipfileurl.zip - to - tarfile.tar
    - zipfileurl.zip - to - tarfile.tar.gz
    - zipfileurl.zip - to - tarfile.tar.bz2
    - zipfileurl.zip - to - directory.gz / directory.bz2 (TypeError: Many-to-one conversion)
  - **Tarfiles:**
    - tarfileurl.tar - to - directory
    - tarfileurl.tar - to - zipfile.zip
    - tarfileurl.tar - to - tarfile.tar
    - tarfileurl.tar - to - tarfile.tar.gz
    - tarfileurl.tar - to - tarfile.tar.bz2
    - tarfileurl.tar - to - directory.gz / directory.bz2 (TypeError: Many-to-one conversion)
    - tarfileurl.tar.gz - to - directory
    - tarfileurl.tar.gz - to - zipfile.zip
    - tarfileurl.tar.gz - to - tarfile.tar
    - tarfileurl.tar.gz - to - tarfile.tar.gz
    - tarfileurl.tar.gz - to - tarfile.tar.bz2
    - tarfileurl.tar.gz - to - directory.gz / directory.bz2 (TypeError: Many-to-one conversion)
    - tarfileurl.tar.bz2 - to - directory
    - tarfileurl.tar.bz2 - to - zipfile.zip
    - tarfileurl.tar.bz2 - to - tarfile.tar
    - tarfileurl.tar.bz2 - to - tarfile.tar.gz
    - tarfileurl.tar.bz2 - to - tarfile.tar.bz2

* tarfileurl.tar.bz2 - to - directory.gz / directory.bz2 (TypeError: Many-to-one conversion)

class nmrstarlib.converter.**Converter**(*file_generator*)

Converter class to convert BMRB NMR-STAR files from NMR-STAR to JSON or from JSON to NMR-STAR format.

__init__(*file_generator*)

Converter initializer.

Parameters **file_generator** (nmrstarlib.converter.Translator) –

convert()

Convert file(s) from NMR-STAR format to JSON format or from JSON format to NMR-STAR format.

Returns None

Return type None

_many_to_many()

Perform many-to-many files conversion.

Returns None

Return type None

_one_to_one()

Perform one-to-one file conversion.

Returns None

Return type None

_to_dir(*file_generator*)

Convert files to directory.

Returns None

Return type None

_to_zipfile(*file_generator*)

Convert files to zip archive.

Returns None

Return type None

_to_tarfile(*file_generator*)

Convert files to tar archive.

Returns None

Return type None

_to_bz2file(*file_generator*)

Convert file to bz2-compressed file.

Returns None

Return type None

_to_gzipfile(*file_generator*)

Convert file to gzip-compressed file.

Returns None

Return type None

_to_textfile (*file_generator*)
Convert file to regular text file.

Returns None

Return type None

_output_path (*inputpath*, *to_format*, *archive=False*)
Construct an output path string from an input path string.

Parameters **inputpath** (*str*) – Input path string.

Returns Output path string.

Return type str

__weakref__
list of weak references to the object (if defined)

nmrstarlib.csviewer

This module provides the *CSViewer* class - Chemical Shifts Viewer that visualizes chemical shifts values.

class nmrstarlib.csviewer.**CSViewer** (*from_path*, *amino_acids=None*, *atoms=None*, *filename=None*, *csview_format='svg'*, *nmrstar_version='3'*)
Chemical Shifts Viewer uses *chem_shifts_by_residue()* method to get chemical shifts organized by residue and visualizes chemical shifts values using the Graphviz (http://www.graphviz.org/) DOT Languge description.

__init__ (*from_path*, *amino_acids=None*, *atoms=None*, *filename=None*, *csview_format='svg'*, *nmrstar_version='3'*)
CSViewer initializer.

Parameters

- **from_path** (*str*) – Path to single NMR-STAR file or BMRB id.
- **amino_acids** (list or tuple) – Sequence of amino acids three letter codes, e.g. 'ALA', 'GLY', 'SER', etc. Leave as *None* to include everything.
- **atoms** (list or tuple) – Sequence of atom types, e.g. 'CA', 'CB', 'HA', etc. Leave as *None* to include everything.
- **filename** (*str*) – Output filename chemical shifts graph to be saved.
- **csview_format** (*str*) – *svg*, *png*, *pdf*. See http://www.graphviz.org/doc/info/output.html for all available formats.
- **nmrstar_version** (*str*) – Version of NMR-STAR format to use for look up chemichal shifts loop.

Returns None

Return type None

csview (*view=False*)
View chemical shift values organized by amino acid residue.

Parameters **view** (True or False) – Open in default image viewer or save file in current working directory quietly.

Returns None

Return type None

__weakref__
list of weak references to the object (if defined)

nmrstarlib.noise

This module provides the *NoiseGenerator* abstract class and *RandomNormalNoiseGenerator* for adding noise values to *Peak* dimensions within a *PeakList*.

class nmrstarlib.noise.**NoiseGenerator**(*parameters*)
Noise generator abstract class.

__init__(*parameters*)
Noise generator initializer.

Parameters **parameters** – Statistical distribution parameters per each peak list split.

generate(*labels*, *split_idx*)
Generate peak-specific noise abstract method, must be reimplemented in a subclass.

Parameters

- **labels** (*tuple*) – Dimension labels of a peak.
- **split_idx** (*int*) – Index specifying which peak list split parameters to use.

Returns List of noise values for dimensions ordered as they appear in a peak.

Return type list

__weakref__
list of weak references to the object (if defined)

class nmrstarlib.noise.**RandomNormalNoiseGenerator**(*parameters*)
Random normal noise generator concrete class.

__init__(*parameters*)
Random normal noise generator initializer.

Parameters **parameters** – Statistical distribution parameters from random normal distribution per each peak list split.

generate(*labels*, *split_idx*)
Generate peak-specific random noise from normal distribution.

Parameters

- **labels** (*tuple*) – Dimension labels of a peak.
- **split_idx** (*int*) – Index specifying which peak list split parameters to use.

Returns List of noise values for dimensions ordered as they appear in a peak.

Return type list

nmrstarlib.plsimulator

This module provides interface classes necessary to create simulated peak list file.

class nmrstarlib.plsimulator.**DimensionComponent**(*label*, *position*)
Dimensions component interface.

__init__(*label*, *position*)
Dimension component.

Parameters

- **label** (*str*) – Label of a dimension.
- **position** (*int*) – Position of dimensions within a peak according to sequence site position, e.g. -1, 0, or +1.

__weakref__
list of weak references to the object (if defined)

class nmrstarlib.plsimulator.**DimensionGroup**(*label*, *position*)
Composite dimension group.

__init__(*label*, *position*)
Dimension group.

Parameters

- **label** (*str*) – Label of a dimension.
- **position** (*int*) – Position of dimensions within a peak according to sequence site position, e.g. -1, 0, or +1.

class nmrstarlib.plsimulator.**Dimension**(*label*, *position*, *assignment=None*, *chemshift=None*)
Concrete dimension.

__init__(*label*, *position*, *assignment=None*, *chemshift=None*)
Concrete dimension intializer.

Parameters

- **label** (*str*) – Label of a dimension.
- **position** (*int*) – Position of dimensions within a peak according to sequence site position, e.g. -1, 0, or +1.
- **assignment** (*str*) – Chemical shift assignment of a dimension.
- **chemshift** (*float*) – Chemical shift value of a dimension.

class nmrstarlib.plsimulator.**Peak**(*labels*)
Peak within a peak list.

__init__(*labels*)
Peak initializer.

Parameters **labels** (*tuple*) – Dimension labels of peak.

assignments_list
List of assignments per each dimension within a peak.

Returns List of assignments.

Return type list

chemshifts_list
List of chemical shift values per each dimensions within a peak.

Returns List of chemical shifts.

Return type list

apply_noise(*noise_generator*, *split_idx*)
Apply noise to dimensions within a peak.

Parameters

- **noise_generator** – Noise generator object.
- **split_idx** (*int*) – Index specifying which peak list split parameters to use.

Returns None

Return type None

__weakref__
list of weak references to the object (if defined)

class nmrstarlib.plsimulator.**PeakList**(*spectrum_name*, *labels*, *source*, *chain_idx*)
Peak list contains chemical shift values and assignment information for each peak.

__init__(*spectrum_name*, *labels*, *source*, *chain_idx*)
Peak list initializer.

Parameters

- **spectrum_name** (*str*) – Spectrum name from which peak list will be simulated.
- **labels** (*list*) – Sequence of labels as they appear in a peak.
- **source** (*str*) – *StarFile* source.
- **chain_idx** (*int*) – *StarFile* chain index.

_to_sparky()
Save *PeakList* into Sparky-formatted string.

Returns Peak list representation in Sparky format.

Return type str

_to_autoassign()
Save *PeakList* into AutoAssign-formatted string.

Returns Peak list representation in AutoAssign format.

Return type str

_to_json()
Save *PeakList* into JSON string.

Returns Peak list representation in JSON format.

Return type str

write(*filehandle*, *fileformat*)
Write *PeakList* data into file.

Parameters

- **filehandle** (io.TextIOWrapper) – file-like object.
- **fileformat** (*str*) – Format to use to write data: *nmrstar* or *json*.

Returns None

Return type None

writestr(*fileformat*)
Write *PeakList* data into string.

Parameters **fileformat** (*str*) – Format to use to write data: *nmrstar* or *json*.

Returns String representing the *PeakList* instance.

Return type str

__weakref__
list of weak references to the object (if defined)

class nmrstarlib.plsimulator.**SpinSystem**
Spin system - collection of related resonances associated with specific atoms in a molecule.

__init__()
Spin system initializer.

__weakref__
list of weak references to the object (if defined)

class nmrstarlib.plsimulator.**SequenceSite**(*residues*)
Sequence site.

__init__(*residues*)
Sequence site initializer.

is_sequential()
Check if residues that sequence site is composed of are in sequential order.

Returns If sequence site is in valid sequential order (True) or not (False).

Return type True or False

__weakref__
list of weak references to the object (if defined)

class nmrstarlib.plsimulator.**PeakTemplate**(*dimensions*)
Peak templates defined as a list of concrete dimensions.

__init__(*dimensions*)
Peak template initializer.

dimension_labels
List of dimension labels.

Returns List of dimension labels of a peak template.

Return type list

dimension_positions
List of dimension positions.

Returns List of dimension positions of a peak template.

Return type list

__weakref__
list of weak references to the object (if defined)

class nmrstarlib.plsimulator.**PeakDescription**(*fraction*, *dimension_labels*)
Peak descriptions defined as list of general dimension groups.

__init__(*fraction*, *dimension_labels*)
Peak description initializer.

Parameters

- **fraction** (*float*) – Describes expected number of peaks.
- **dimension_labels** – List of dimension labels.

static create_dimension_groups(*dimension_positions*)
Create list of dimension groups.

Parameters **dimension_positions** (*zip*) – List of tuples describing dimension and its position within sequence site.

Returns List of dimension groups.

Return type list

__weakref__
list of weak references to the object (if defined)

class nmrstarlib.plsimulator.**Spectrum**(*name*, *labels*, *min_spin_system_peaks*, *amino_acids_and_atoms=None*)
Spectrum object described as a list of general peak descriptions.

__init__(*name*, *labels*, *min_spin_system_peaks*, *amino_acids_and_atoms=None*)
Spectrum initializer.

Parameters

- **name** (*str*) – Spectrum name.
- **labels** – Sequence of dimension labels as they appear in a peak.
- **min_spin_system_peaks** (*int*) – Minimum number of peaks per spin system.

peak_templates
Create a list of concrete peak templates from a list of general peak descriptions.

Returns List of peak templates.

Return type list

__weakref__
list of weak references to the object (if defined)

seq_site_length
Calculate length of a single sequence site based upon relative positions specified in peak descriptions.

Returns Length of sequence site.

Return type int

nmrstarlib.translator

This module provides the *Translator* abstract class and concrete classes: *StarFileToStarFile* for converting between NMR-STAR and JSONized NMR-STAR formats and *StarFileToPeakList* for converting NMR-STAR formatted file into simulated peak list file.

class nmrstarlib.translator.**Translator**(*from_path*, *to_path*, *from_format=None*, *to_format=None*)
Translator abstract class.

__init__(*from_path*, *to_path*, *from_format=None*, *to_format=None*)
Translator initializer.

Parameters

- **from_path** (*str*) – Path to input file(s).
- **to_path** (*str*) – Path to output file(s).
- **from_format** (*str*) – Input format.
- **to_format** (*str*) – Output format.

__iter__()
Abstract iterator must be implemented in a subclass.

__weakref__
list of weak references to the object (if defined)

class nmrstarlib.translator.**StarFileToStarFile**(*from_path*, *to_path*, *from_format=None*, *to_format=None*)
Translator concrete class that can convert between NMR-STAR and JSONized NMR-STAR formats.

__init__(*from_path*, *to_path*, *from_format=None*, *to_format=None*)
StarFileToStarFile translator initializer.

Parameters

- **from_path** (*str*) – Path to input file(s).
- **to_path** (*str*) – Path to output file(s).
- **from_format** (*str*) – Input format: *nmrstar* or *json*.
- **to_format** (*str*) – Output format: *nmrstar* or *json*.

__iter__()
Iterator that yields instances of *StarFile* instances.

Returns instance of *StarFile* object instance.

Return type *StarFile*

class nmrstarlib.translator.**StarFileToPeakList**(*from_path*, *to_path*, *from_format*, *to_format*, *spectrum_name*, *plsplit=None*, *noise_generator=None*, *nmrstar_version='3'*)
Translator concrete class that can convert NMR-STAR of JSONized NMR-STAR formatted file into peak list file.

__init__(*from_path*, *to_path*, *from_format*, *to_format*, *spectrum_name*, *plsplit=None*, *noise_generator=None*, *nmrstar_version='3'*)
StarFileToPeakList initializer.

Parameters

- **from_path** (*str*) – Path to input file(s).
- **to_path** (*str*) – Path to output file(s).
- **from_format** (*str*) – Input format: *nmrstar* or *json*.
- **to_format** (*str*) – Output format: *json* or *sparky*.
- **spectrum_name** (*str*) – Name of spectrum from which to simulate peak list.
- **plsplit** (*tuple*) – How to split peak list in order to account for multiple sources of variance.
- **nmrstar_version** (*str*) – Version of NMR-STAR format to use for look up chemical shifts loop.
- **noise_generator** (*NoiseGenerator*) – Subclasses of *NoiseGenerator* object.

static create_spectrum(*spectrum_name*)
Initialize spectrum and peak descriptions.

Parameters **spectrum_name** (*str*) – Name of the spectrum from which peak list will be simulated.

Returns Spectrum object.

Return type *Spectrum*

static create_sequence_sites(*chain*, *seq_site_length*)
Create sequence sites using sequence ids.

Parameters

- **chain** (*dict*) – Chain object that contains chemical shift values and assignment information.
- **seq_site_length** (*int*) – Length of a single sequence site.

Returns List of sequence sites.

Return type list

static calculate_intervals(*chunk_sizes*)
Calculate intervals for a given chunk sizes.

Parameters **chunk_sizes** (*list*) – List of chunk sizes.

Returns Tuple of intervals.

Return type tuple

split_by_percent(*spin_systems_list*)
Split list of spin systems by specified percentages.

Parameters **spin_systems_list** (*list*) – List of spin systems.

Returns List of spin systems divided into sub-lists corresponding to specified split percentages.

Return type list

create_peaklist(*spectrum*, *chain*, *chain_idx*, *source*)
Create peak list file.

Parameters

- **spectrum** (*Spectrum*) – Spectrum object instance.
- **chain** (*dict*) – Chain object that contains chemical shift values and assignment information.
- **chain_idx** (*int*) – Protein chain index.
- **source** (*str*) – *StarFile* source.

Returns Peak list object.

Return type *PeakList*

__iter__()
Iterator that yields instances of *PeakList* instances.

Returns instance of *PeakList* object instance.

Return type *PeakList*

License

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index

Symbols

A

B

C

D

G

I

L

N

O

P

R

S

T

U

W