nmrstarlib Documentation

Release 2.1.1

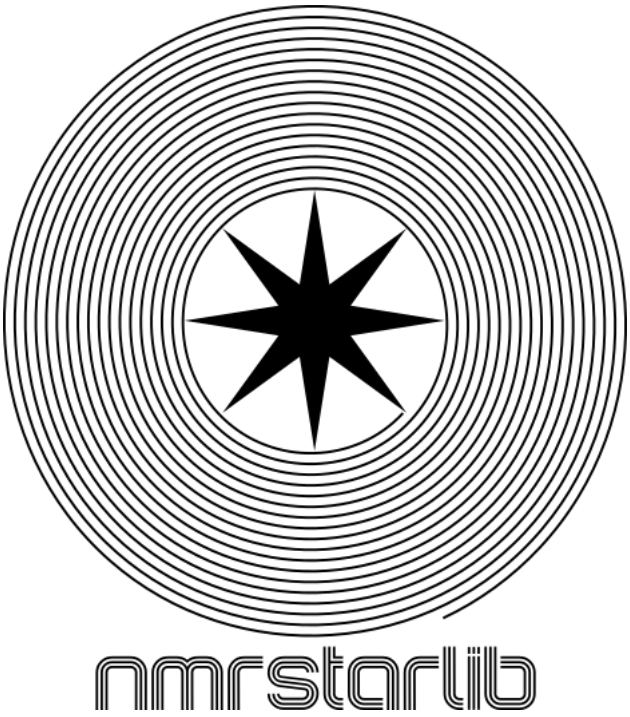
Andrey Smelter, Hunter N.B. Moseley

Nov 08, 2018

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 and CIF formatted files used by Protein Data Bank (PDB).

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

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 and CIF formatted files.
- As a command-line tool to convert between NMR-STAR/CIF format and its equivalent JSONized NMR-STAR/CIF format, to create a large number of simulated peak lists, and also to visualize chemical shift data from NMR-STAR formatted files.

1.1 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.

1.2 Links

- nmrstarlib @ GitHub
- nmrstarlib @ PyPI
- Documentation @ ReadTheDocs

1.3 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.

1.3.1 Install on Linux, Mac OS X

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

1.3.2 Install on Windows

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

1.4 Quickstart

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

```
>>> import nmrstarlib
>>>
>>> # "path": path_to_file / path_to_dir / path_to_archive / bmrb_id / pdb_id / file_
↪url
>>> for file in nmrstarlib.read_files("path"):
...     print(file.id)      # print BMRB/PDB id of a file
...     print(file.source)  # print source of a file
...     print(file.keys())  # print top-level 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.

1.5 License

This package is distributed under the MIT *license*.

CHAPTER 2

Documentation index:

2.1 User Guide

2.1.1 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.

2.1.2 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
```

2.1.3 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
```

2.1.4 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).

2.1.5 Optional dependencies

- **numpy for generating noise values from random 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
```

  - If the numpy is not installed distributions from the Python standard library `random` module will be used.

2.1.6 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.

2.2 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.

2.2.1 Using nmrstarlib as a library

Importing nmrstarlib package

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

```
In [1]: 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:

```
In [2]: 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:

```
In [3]: for starfile in nmrstarlib.read_files("18569", "15000"):
            print("BMRB id:", starfile.id)          # print BMRB id of StarFile
            print("File source:", starfile.source)  # print source of StarFile
            for saveframe_name in starfile.keys():  # print saveframe names
                print("\t", saveframe_name)
```

```
BMRB id: 18569
File source: http://rest.bmrb.wisc.edu/bmrb/NMR-STAR3/18569
         data
         comment_0
         save_entry_information
         comment_1
         save_entry_citation
         comment_2
         save_assembly
         comment_3
         save_EVH1
         comment_4
         save_natural_source
         comment_5
         save_experimental_source
         comment_6
         comment_7
         save_sample_1
         save_sample_2
         save_sample_3
         save_sample_4
         comment_8
         save_sample_conditions_1
         save_sample_conditions_2
```

```
         save_sample_conditions_3
         save_sample_conditions_4
         comment_9
         save_AZARA
         save_xwinnmr
         save_ANSIG
         save_CNS
         comment_10
         comment_11
         save_spectrometer_1
         save_spectrometer_2
         save_NMR_spectrometer_list
         comment_12
         save_experiment_list
         comment_13
         comment_14
         comment_15
         save_chemical_shift_reference_1
         comment_16
         comment_17
         save_assigned_chem_shift_list_1
         comment_18
         save_combined_NOESY_peak_list
BMRB id: 15000
File source: http://rest.bmrb.wisc.edu/bmrb/NMR-STAR3/15000
         data
         comment_0
         save_entry_information
         comment_1
         save_citation_1
         comment_2
         save_assembly
         comment_3
         save_F5-Phe-cVHP
         comment_4
         save_natural_source
         comment_5
         save_experimental_source
         comment_6
         save_chem_comp_PHF
         comment_7
         comment_8
         save_unlabeled_sample
         save_selectively_labeled_sample
         comment_9
         save_sample_conditions
         comment_10
         save_NMRPipe
         save_PIPP
         save_SPARKY
         save_CYANA
         save_X-PLOR_NIH
         comment_11
         comment_12
         save_spectrometer_1
         save_spectrometer_2
         save_spectrometer_3
         save_spectrometer_4
         save_spectrometer_5
```

```
 save_spectrometer_6
 save_NMR_spectrometer_list
 comment_13
 save_experiment_list
 comment_14
 comment_15
 comment_16
 save_chemical_shift_reference_1
 comment_17
 comment_18
 save_assigned_chem_shift_list_1
```

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:

```
In [4]: sf_generator = nmrstarlib.read_files("18569", "15000")

        starfile1 = next(sf_generator)
        starfile2 = next(sf_generator)
```

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

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

```
In [5]: starfiles_list = list(nmrstarlib.read_files("18569", "15000"))
```

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*:

```
In [7]: starfile = next(nmrstarlib.read_files("15000"))

        # list StarFile-level keys, i.e. saveframe names
        list(starfile.keys())
```

```
Out[7]: ['data',
         'comment_0',
         'save_entry_information',
         'comment_1',
         'save_citation_1',
         'comment_2',
         'save_assembly',
         'comment_3',
         'save_F5-Phe-cVHP',
         'comment_4',
         'save_natural_source',
         'comment_5',
         'save_experimental_source',
```

```
 'comment_6',
 'save_chem_comp_PHF',
 'comment_7',
 'comment_8',
 'save_unlabeled_sample',
 'save_selectively_labeled_sample',
 'comment_9',
 'save_sample_conditions',
 'comment_10',
 'save_NMRPipe',
 'save_PIPP',
 'save_SPARKY',
 'save_CYANA',
 'save_X-PLOR_NIH',
 'comment_11',
 'comment_12',
 'save_spectrometer_1',
 'save_spectrometer_2',
 'save_spectrometer_3',
 'save_spectrometer_4',
 'save_spectrometer_5',
 'save_spectrometer_6',
 'save_NMR_spectrometer_list',
 'comment_13',
 'save_experiment_list',
 'comment_14',
 'comment_15',
 'comment_16',
 'save_chemical_shift_reference_1',
 'comment_17',
 'comment_18',
 'save_assigned_chem_shift_list_1']
```

```
In [8]: # access "data" field
        starfile["data"]
```

```
Out[8]: '15000'
```

```
In [9]: # access saveframe
        starfile["save_entry_information"]
```

```
Out[9]: OrderedDict([('Entry.Sf_category', 'entry_information'),
                     ('Entry.Sf_framecode', 'entry_information'),
                     ('Entry.ID', '15000'),
                     ('Entry.Title',
                      '\nSolution structure of chicken villin headpiece subdomain containing a fluor
                     ('Entry.Type', 'macromolecule'),
                     ('Entry.Version_type', 'original'),
                     ('Entry.Submission_date', '2006-09-07'),
                     ('Entry.Accession_date', '2006-09-07'),
                     ('Entry.Last_release_date', '.'),
                     ('Entry.Original_release_date', '.'),
                     ('Entry.Origination', 'author'),
                     ('Entry.NMR_STAR_version', '3.1.1.61'),
                     ('Entry.Original_NMR_STAR_version', '.'),
                     ('Entry.Experimental_method', 'NMR'),
                     ('Entry.Experimental_method_subtype', 'solution'),
                     ('Entry.Details', '.'),
                     ('Entry.BMRB_internal_directory_name', '.'),
                     ('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', 'Claudia'),
                 ('Entry_author.Family_name', 'Cornilescu'),
                 ('Entry_author.First_initial', '.'),
                 ('Entry_author.Middle_initials', 'C.'),
                 ('Entry_author.Family_title', '.'),
                 ('Entry_author.Entry_ID', '15000')]),
    OrderedDict([('Entry_author.Ordinal', '2'),
                 ('Entry_author.Given_name', 'Gabriel'),
                 ('Entry_author.Family_name', 'Cornilescu'),
                 ('Entry_author.First_initial', '.'),
                 ('Entry_author.Middle_initials', '.'),
                 ('Entry_author.Family_title', '.'),
                 ('Entry_author.Entry_ID', '15000')]),
    OrderedDict([('Entry_author.Ordinal', '3'),
                 ('Entry_author.Given_name', 'Erik'),
                 ('Entry_author.Family_name', 'Hadley'),
                 ('Entry_author.First_initial', '.'),
                 ('Entry_author.Middle_initials', 'B.'),
                 ('Entry_author.Family_title', '.'),
                 ('Entry_author.Entry_ID', '15000')]),
    OrderedDict([('Entry_author.Ordinal', '4'),
                 ('Entry_author.Given_name', 'Samuel'),
                 ('Entry_author.Family_name', 'Gellman'),
                 ('Entry_author.First_initial', '.'),
                 ('Entry_author.Middle_initials', 'H.'),
                 ('Entry_author.Family_title', '.'),
                 ('Entry_author.Entry_ID', '15000')]),
    OrderedDict([('Entry_author.Ordinal', '5'),
                 ('Entry_author.Given_name', 'John'),
                 ('Entry_author.Family_name', 'Markley'),
                 ('Entry_author.First_initial', '.'),
                 ('Entry_author.Middle_initials', 'L.'),
                 ('Entry_author.Family_title', '.'),
                 ('Entry_author.Entry_ID', '15000')])])),
('loop_1',
 (['SG_project.SG_project_ID',
   'SG_project.Project_name',
   'SG_project.Full_name_of_center',
   'SG_project.Initial_of_center',
   'SG_project.Entry_ID'],
  [OrderedDict([('SG_project.SG_project_ID', '1'),
                ('SG_project.Project_name', 'not applicable'),
                ('SG_project.Full_name_of_center',
                 'not applicable'),
                ('SG_project.Initial_of_center', '.'),
                ('SG_project.Entry_ID', '15000')])])),
('loop_2',
 (['Struct_keywords.Keywords',
   'Struct_keywords.Text',
   'Struct_keywords.Entry_ID'],
  [OrderedDict([('Struct_keywords.Keywords',
                 'chicken villin headpiece'),
```

```
                ('Struct_keywords.Text', '.'),
                ('Struct_keywords.Entry_ID', '15000')]),
   OrderedDict([('Struct_keywords.Keywords', 'fluorinated Phe'),
                ('Struct_keywords.Text', '.'),
                ('Struct_keywords.Entry_ID', '15000')]),
   OrderedDict([('Struct_keywords.Keywords', 'VHP'),
                ('Struct_keywords.Text', '.'),
                ('Struct_keywords.Entry_ID', '15000')])])),
('loop_3',
 (['Data_set.Type', 'Data_set.Count', 'Data_set.Entry_ID'],
  [OrderedDict([('Data_set.Type', 'assigned_chemical_shifts'),
                ('Data_set.Count', '1'),
                ('Data_set.Entry_ID', '15000')])])),
('loop_4',
 (['Datum.Type', 'Datum.Count', 'Datum.Entry_ID'],
  [OrderedDict([('Datum.Type', '13C chemical shifts'),
                ('Datum.Count', '77'),
                ('Datum.Entry_ID', '15000')]),
   OrderedDict([('Datum.Type', '15N chemical shifts'),
                ('Datum.Count', '40'),
                ('Datum.Entry_ID', '15000')]),
   OrderedDict([('Datum.Type', '1H chemical shifts'),
                ('Datum.Count', '223'),
                ('Datum.Entry_ID', '15000')])])),
('loop_5',
 (['Release.Release_number',
   'Release.Format_type',
   'Release.Format_version',
   'Release.Date',
   'Release.Submission_date',
   'Release.Type',
   'Release.Author',
   'Release.Detail',
   'Release.Entry_ID'],
  [OrderedDict([('Release.Release_number', '2'),
                ('Release.Format_type', '.'),
                ('Release.Format_version', '.'),
                ('Release.Date', '2008-07-17'),
                ('Release.Submission_date', '2006-09-06'),
                ('Release.Type', 'update'),
                ('Release.Author', 'BMRB'),
                ('Release.Detail', 'complete entry citation'),
                ('Release.Entry_ID', '15000')]),
   OrderedDict([('Release.Release_number', '1'),
                ('Release.Format_type', '.'),
                ('Release.Format_version', '.'),
                ('Release.Date', '2006-10-20'),
                ('Release.Submission_date', '2006-09-06'),
                ('Release.Type', 'original'),
                ('Release.Author', 'author'),
                ('Release.Detail', 'original release'),
                ('Release.Entry_ID', '15000')])])),
('loop_6',
 (['Related_entries.Database_name',
   'Related_entries.Database_accession_code',
   'Related_entries.Relationship',
   'Related_entries.Entry_ID'],
  [OrderedDict([('Related_entries.Database_name', 'PDB'),
                ('Related_entries.Database_accession_code',
```

```
                                    '2JM0'),
                                   ('Related_entries.Relationship',
                                    'BMRB Entry Tracking System'),
                                   ('Related_entries.Entry_ID', '15000')])]))])
```

```
In [10]: # list saveframe-level keys
         list(starfile["save_entry_information"].keys())
```

```
Out[10]: ['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',
          'loop_5',
          'loop_6']
```

```
In [11]: # access 'key-value' pairs within saveframes
         starfile["save_entry_information"]["Entry.Submission_date"]
```

```
Out[11]: '2006-09-07'
```

```
In [12]: # access loops
         starfile["save_entry_information"]["loop_0"]
```

```
Out[12]: (['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', 'Claudia'),
                        ('Entry_author.Family_name', 'Cornilescu'),
                        ('Entry_author.First_initial', '.'),
                        ('Entry_author.Middle_initials', 'C.'),
                        ('Entry_author.Family_title', '.'),
                        ('Entry_author.Entry_ID', '15000')]),
           OrderedDict([('Entry_author.Ordinal', '2'),
                        ('Entry_author.Given_name', 'Gabriel'),
                        ('Entry_author.Family_name', 'Cornilescu'),
                        ('Entry_author.First_initial', '.'),
                        ('Entry_author.Middle_initials', '.'),
                        ('Entry_author.Family_title', '.'),
```

```
                    ('Entry_author.Entry_ID', '15000')]),
          OrderedDict([('Entry_author.Ordinal', '3'),
                       ('Entry_author.Given_name', 'Erik'),
                       ('Entry_author.Family_name', 'Hadley'),
                       ('Entry_author.First_initial', '.'),
                       ('Entry_author.Middle_initials', 'B.'),
                       ('Entry_author.Family_title', '.'),
                       ('Entry_author.Entry_ID', '15000')]),
          OrderedDict([('Entry_author.Ordinal', '4'),
                       ('Entry_author.Given_name', 'Samuel'),
                       ('Entry_author.Family_name', 'Gellman'),
                       ('Entry_author.First_initial', '.'),
                       ('Entry_author.Middle_initials', 'H.'),
                       ('Entry_author.Family_title', '.'),
                       ('Entry_author.Entry_ID', '15000')]),
          OrderedDict([('Entry_author.Ordinal', '5'),
                       ('Entry_author.Given_name', 'John'),
                       ('Entry_author.Family_name', 'Markley'),
                       ('Entry_author.First_initial', '.'),
                       ('Entry_author.Middle_initials', 'L.'),
                       ('Entry_author.Family_title', '.'),
                       ('Entry_author.Entry_ID', '15000')])])

In [13]: # list loop-level fields
         starfile["save_entry_information"]["loop_0"][0]

Out[13]: ['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']

In [14]: # list loop-level values (list of dictionaries)
         starfile["save_entry_information"]["loop_0"][1]

Out[14]: [OrderedDict([('Entry_author.Ordinal', '1'),
                       ('Entry_author.Given_name', 'Claudia'),
                       ('Entry_author.Family_name', 'Cornilescu'),
                       ('Entry_author.First_initial', '.'),
                       ('Entry_author.Middle_initials', 'C.'),
                       ('Entry_author.Family_title', '.'),
                       ('Entry_author.Entry_ID', '15000')]),
          OrderedDict([('Entry_author.Ordinal', '2'),
                       ('Entry_author.Given_name', 'Gabriel'),
                       ('Entry_author.Family_name', 'Cornilescu'),
                       ('Entry_author.First_initial', '.'),
                       ('Entry_author.Middle_initials', '.'),
                       ('Entry_author.Family_title', '.'),
                       ('Entry_author.Entry_ID', '15000')]),
          OrderedDict([('Entry_author.Ordinal', '3'),
                       ('Entry_author.Given_name', 'Erik'),
                       ('Entry_author.Family_name', 'Hadley'),
                       ('Entry_author.First_initial', '.'),
                       ('Entry_author.Middle_initials', 'B.'),
                       ('Entry_author.Family_title', '.'),
                       ('Entry_author.Entry_ID', '15000')]),
          OrderedDict([('Entry_author.Ordinal', '4'),
                       ('Entry_author.Given_name', 'Samuel'),
                       ('Entry_author.Family_name', 'Gellman'),
```

```
                        ('Entry_author.First_initial', '.'),
                        ('Entry_author.Middle_initials', 'H.'),
                        ('Entry_author.Family_title', '.'),
                        ('Entry_author.Entry_ID', '15000')]),
          OrderedDict([('Entry_author.Ordinal', '5'),
                        ('Entry_author.Given_name', 'John'),
                        ('Entry_author.Family_name', 'Markley'),
                        ('Entry_author.First_initial', '.'),
                        ('Entry_author.Middle_initials', 'L.'),
                        ('Entry_author.Family_title', '.'),
                        ('Entry_author.Entry_ID', '15000')])]
```

```
In [15]: # every loop entry is accessed by index
         starfile["save_entry_information"]["loop_0"][1][0]["Entry_author.Family_name"]
```

```
Out[15]: 'Cornilescu'
```

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

```
In [16]: # check submission date
         starfile["save_entry_information"]["Entry.Submission_date"]
```

```
Out[16]: '2006-09-07'
```

```
In [17]: # change submission date
         starfile["save_entry_information"]["Entry.Submission_date"] = "2015-07-05"
```

```
In [18]: # check that submission date is updated
         starfile["save_entry_information"]["Entry.Submission_date"]
```

```
Out[18]: '2015-07-05'
```

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

```
In [19]: starfile = next(nmrstarlib.read_files("bmr15000.str"))
```

```
In [20]: starfile.print_file(file_format="nmrstar")
```

```
data_15000

######################
#  Entry information  #
######################

save_entry_information
   _Entry.Sf_category  entry_information
   _Entry.Sf_framecode         entry_information
   _Entry.ID    15000
   _Entry.Title
;

Solution structure of chicken villin headpiece subdomain containing a fluorinated side chain in the c
;
   _Entry.Type  macromolecule
   _Entry.Version_type         original
   _Entry.Submission_date      2006-09-07
   _Entry.Accession_date       2006-09-07
   _Entry.Last_release_date   .
   _Entry.Original_release_date       .
   _Entry.Origination   author
   _Entry.NMR_STAR_version    3.1.1.61
   _Entry.Original_NMR_STAR_version  .
   _Entry.Experimental_method  NMR
   _Entry.Experimental_method_subtype         solution
```

```
   _Entry.Details       .
   _Entry.BMRB_internal_directory_name       .

   loop_
      _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

      1 Claudia Cornilescu . C. . 15000
      2 Gabriel Cornilescu . . . 15000
      3 Erik Hadley . B. . 15000
      4 Samuel Gellman . H. . 15000
      5 John Markley . L. . 15000

   stop_

   loop_
      _Datum.Type
      _Datum.Count
      _Datum.Entry_ID

      '13C chemical shifts' 77 15000
      '15N chemical shifts' 40 15000
      '1H chemical shifts' 223 15000

   stop_

save_


save_assigned_chem_shift_list_1
   _Assigned_chem_shift_list.Sf_category    assigned_chemical_shifts
   _Assigned_chem_shift_list.Sf_framecode   assigned_chem_shift_list_1
   _Assigned_chem_shift_list.Entry_ID       15000
   _Assigned_chem_shift_list.ID      1
   _Assigned_chem_shift_list.Sample_condition_list_ID     1
   _Assigned_chem_shift_list.Sample_condition_list_label  $sample_conditions
   _Assigned_chem_shift_list.Chem_shift_reference_ID      1
   _Assigned_chem_shift_list.Chem_shift_reference_label   $chemical_shift_reference_1
   _Assigned_chem_shift_list.Chem_shift_1H_err    .
   _Assigned_chem_shift_list.Chem_shift_13C_err   .
   _Assigned_chem_shift_list.Chem_shift_15N_err   .
   _Assigned_chem_shift_list.Chem_shift_31P_err   .
   _Assigned_chem_shift_list.Chem_shift_2H_err    .
   _Assigned_chem_shift_list.Chem_shift_19F_err   .
   _Assigned_chem_shift_list.Error_derivation_method       .
   _Assigned_chem_shift_list.Details         .
   _Assigned_chem_shift_list.Text_data_format      .
   _Assigned_chem_shift_list.Text_data      .

   loop_
      _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

      1 . 1 1 2 2 SER H H 1 9.3070 0.01 . . . . . . 2 SER H . 15000 1
      2 . 1 1 2 2 SER HA H 1 4.5970 0.01 . . . . . . 2 SER HA . 15000 1
      3 . 1 1 2 2 SER HB2 H 1 4.3010 0.01 . . . . . . 2 SER HB2 . 15000 1
      4 . 1 1 2 2 SER HB3 H 1 4.0550 0.01 . . . . . . 2 SER HB3 . 15000 1
      5 . 1 1 2 2 SER CB C 13 64.6000 0.1 . . . . . . 2 SER CB . 15000 1
      6 . 1 1 2 2 SER N N 15 121.5800 0.1 . . . . . . 2 SER N . 15000 1
      7 . 1 1 3 3 ASP H H 1 8.0740 0.01 . . . . . . 3 ASP H . 15000 1
      8 . 1 1 3 3 ASP HA H 1 4.5580 0.01 . . . . . . 3 ASP HA . 15000 1
      9 . 1 1 3 3 ASP HB2 H 1 2.835 0.01 . . . . . . 3 ASP HB2 . 15000 1
      10 . 1 1 3 3 ASP HB3 H 1 2.754 0.01 . . . . . . 3 ASP HB3 . 15000 1
      11 . 1 1 3 3 ASP CA C 13 57.6400 0.1 . . . . . . 3 ASP CA . 15000 1
      12 . 1 1 3 3 ASP N N 15 121.1040 0.1 . . . . . . 3 ASP N . 15000 1
      13 . 1 1 4 4 GLU H H 1 8.6520 0.01 . . . . . . 4 GLU H . 15000 1
      14 . 1 1 4 4 GLU HA H 1 4.1420 0.01 . . . . . . 4 GLU HA . 15000 1
      15 . 1 1 4 4 GLU HB2 H 1 2.0520 0.01 . . . . . . 4 GLU HB2 . 15000 1
      16 . 1 1 4 4 GLU HB3 H 1 2.0320 0.01 . . . . . . 4 GLU HB3 . 15000 1
      17 . 1 1 4 4 GLU HG2 H 1 2.4540 0.01 . . . . . . 4 GLU HG2 . 15000 1
      18 . 1 1 4 4 GLU CB C 13 28.1200 0.1 . . . . . . 4 GLU CB . 15000 1
      19 . 1 1 4 4 GLU CG C 13 33.2720 0.1 . . . . . . 4 GLU CG . 15000 1
      20 . 1 1 4 4 GLU N N 15 119.8900 0.1 . . . . . . 4 GLU N . 15000 1

   stop_

save_
```

```
In [21]: starfile.print_file(file_format="json")
```

```
{
    "data": "15000",
    "comment_0": "########################\n#  Entry information  #\n########################\n",
    "save_entry_information": {
        "Entry.Sf_category": "entry_information",
        "Entry.Sf_framecode": "entry_information",
        "Entry.ID": "15000",
        "Entry.Title": "\nSolution structure of chicken villin headpiece subdomain containing a fluor
        "Entry.Type": "macromolecule",
        "Entry.Version_type": "original",
```

```
"Entry.Submission_date": "2006-09-07",
"Entry.Accession_date": "2006-09-07",
"Entry.Last_release_date": ".",
"Entry.Original_release_date": ".",
"Entry.Origination": "author",
"Entry.NMR_STAR_version": "3.1.1.61",
"Entry.Original_NMR_STAR_version": ".",
"Entry.Experimental_method": "NMR",
"Entry.Experimental_method_subtype": "solution",
"Entry.Details": ".",
"Entry.BMRB_internal_directory_name": ".",
"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"
    ],
    [
        {
            "Entry_author.Ordinal": "1",
            "Entry_author.Given_name": "Claudia",
            "Entry_author.Family_name": "Cornilescu",
            "Entry_author.First_initial": ".",
            "Entry_author.Middle_initials": "C.",
            "Entry_author.Family_title": ".",
            "Entry_author.Entry_ID": "15000"
        },
        {
            "Entry_author.Ordinal": "2",
            "Entry_author.Given_name": "Gabriel",
            "Entry_author.Family_name": "Cornilescu",
            "Entry_author.First_initial": ".",
            "Entry_author.Middle_initials": ".",
            "Entry_author.Family_title": ".",
            "Entry_author.Entry_ID": "15000"
        },
        {
            "Entry_author.Ordinal": "3",
            "Entry_author.Given_name": "Erik",
            "Entry_author.Family_name": "Hadley",
            "Entry_author.First_initial": ".",
            "Entry_author.Middle_initials": "B.",
            "Entry_author.Family_title": ".",
            "Entry_author.Entry_ID": "15000"
        },
        {
            "Entry_author.Ordinal": "4",
            "Entry_author.Given_name": "Samuel",
            "Entry_author.Family_name": "Gellman",
            "Entry_author.First_initial": ".",
            "Entry_author.Middle_initials": "H.",
            "Entry_author.Family_title": ".",
            "Entry_author.Entry_ID": "15000"
        },
        {
```

```
                    "Entry_author.Ordinal": "5",
                    "Entry_author.Given_name": "John",
                    "Entry_author.Family_name": "Markley",
                    "Entry_author.First_initial": ".",
                    "Entry_author.Middle_initials": "L.",
                    "Entry_author.Family_title": ".",
                    "Entry_author.Entry_ID": "15000"
                }
            ]
        ],
        "loop_1": [
            [
                "Datum.Type",
                "Datum.Count",
                "Datum.Entry_ID"
            ],
            [
                {
                    "Datum.Type": "13C chemical shifts",
                    "Datum.Count": "77",
                    "Datum.Entry_ID": "15000"
                },
                {
                    "Datum.Type": "15N chemical shifts",
                    "Datum.Count": "40",
                    "Datum.Entry_ID": "15000"
                },
                {
                    "Datum.Type": "1H chemical shifts",
                    "Datum.Count": "223",
                    "Datum.Entry_ID": "15000"
                }
            ]
        ]
    },
    "save_assigned_chem_shift_list_1": {
        "Assigned_chem_shift_list.Sf_category": "assigned_chemical_shifts",
        "Assigned_chem_shift_list.Sf_framecode": "assigned_chem_shift_list_1",
        "Assigned_chem_shift_list.Entry_ID": "15000",
        "Assigned_chem_shift_list.ID": "1",
        "Assigned_chem_shift_list.Sample_condition_list_ID": "1",
        "Assigned_chem_shift_list.Sample_condition_list_label": "$sample_conditions",
        "Assigned_chem_shift_list.Chem_shift_reference_ID": "1",
        "Assigned_chem_shift_list.Chem_shift_reference_label": "$chemical_shift_reference_1",
        "Assigned_chem_shift_list.Chem_shift_1H_err": ".",
        "Assigned_chem_shift_list.Chem_shift_13C_err": ".",
        "Assigned_chem_shift_list.Chem_shift_15N_err": ".",
        "Assigned_chem_shift_list.Chem_shift_31P_err": ".",
        "Assigned_chem_shift_list.Chem_shift_2H_err": ".",
        "Assigned_chem_shift_list.Chem_shift_19F_err": ".",
        "Assigned_chem_shift_list.Error_derivation_method": ".",
        "Assigned_chem_shift_list.Details": ".",
        "Assigned_chem_shift_list.Text_data_format": ".",
        "Assigned_chem_shift_list.Text_data": ".",
        "loop_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"
    ],
    [
        {
            "Atom_chem_shift.ID": "1",
            "Atom_chem_shift.Assembly_atom_ID": ".",
            "Atom_chem_shift.Entity_assembly_ID": "1",
            "Atom_chem_shift.Entity_ID": "1",
            "Atom_chem_shift.Comp_index_ID": "2",
            "Atom_chem_shift.Seq_ID": "2",
            "Atom_chem_shift.Comp_ID": "SER",
            "Atom_chem_shift.Atom_ID": "H",
            "Atom_chem_shift.Atom_type": "H",
            "Atom_chem_shift.Atom_isotope_number": "1",
            "Atom_chem_shift.Val": "9.3070",
            "Atom_chem_shift.Val_err": "0.01",
            "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": "2",
            "Atom_chem_shift.Auth_comp_ID": "SER",
            "Atom_chem_shift.Auth_atom_ID": "H",
            "Atom_chem_shift.Details": ".",
            "Atom_chem_shift.Entry_ID": "15000",
            "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
        },
        {
            "Atom_chem_shift.ID": "2",
            "Atom_chem_shift.Assembly_atom_ID": ".",
            "Atom_chem_shift.Entity_assembly_ID": "1",
            "Atom_chem_shift.Entity_ID": "1",
            "Atom_chem_shift.Comp_index_ID": "2",
            "Atom_chem_shift.Seq_ID": "2",
            "Atom_chem_shift.Comp_ID": "SER",
            "Atom_chem_shift.Atom_ID": "HA",
            "Atom_chem_shift.Atom_type": "H",
```

```
        "Atom_chem_shift.Atom_isotope_number": "1",
        "Atom_chem_shift.Val": "4.5970",
        "Atom_chem_shift.Val_err": "0.01",
        "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": "2",
        "Atom_chem_shift.Auth_comp_ID": "SER",
        "Atom_chem_shift.Auth_atom_ID": "HA",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "3",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "2",
        "Atom_chem_shift.Seq_ID": "2",
        "Atom_chem_shift.Comp_ID": "SER",
        "Atom_chem_shift.Atom_ID": "HB2",
        "Atom_chem_shift.Atom_type": "H",
        "Atom_chem_shift.Atom_isotope_number": "1",
        "Atom_chem_shift.Val": "4.3010",
        "Atom_chem_shift.Val_err": "0.01",
        "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": "2",
        "Atom_chem_shift.Auth_comp_ID": "SER",
        "Atom_chem_shift.Auth_atom_ID": "HB2",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "4",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "2",
        "Atom_chem_shift.Seq_ID": "2",
        "Atom_chem_shift.Comp_ID": "SER",
        "Atom_chem_shift.Atom_ID": "HB3",
        "Atom_chem_shift.Atom_type": "H",
        "Atom_chem_shift.Atom_isotope_number": "1",
        "Atom_chem_shift.Val": "4.0550",
        "Atom_chem_shift.Val_err": "0.01",
        "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": "2",
        "Atom_chem_shift.Auth_comp_ID": "SER",
        "Atom_chem_shift.Auth_atom_ID": "HB3",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "5",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "2",
        "Atom_chem_shift.Seq_ID": "2",
        "Atom_chem_shift.Comp_ID": "SER",
        "Atom_chem_shift.Atom_ID": "CB",
        "Atom_chem_shift.Atom_type": "C",
        "Atom_chem_shift.Atom_isotope_number": "13",
        "Atom_chem_shift.Val": "64.6000",
        "Atom_chem_shift.Val_err": "0.1",
        "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": "2",
        "Atom_chem_shift.Auth_comp_ID": "SER",
        "Atom_chem_shift.Auth_atom_ID": "CB",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "6",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "2",
        "Atom_chem_shift.Seq_ID": "2",
        "Atom_chem_shift.Comp_ID": "SER",
        "Atom_chem_shift.Atom_ID": "N",
        "Atom_chem_shift.Atom_type": "N",
        "Atom_chem_shift.Atom_isotope_number": "15",
        "Atom_chem_shift.Val": "121.5800",
        "Atom_chem_shift.Val_err": "0.1",
        "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": "2",
        "Atom_chem_shift.Auth_comp_ID": "SER",
        "Atom_chem_shift.Auth_atom_ID": "N",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
```

```
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "7",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "3",
        "Atom_chem_shift.Seq_ID": "3",
        "Atom_chem_shift.Comp_ID": "ASP",
        "Atom_chem_shift.Atom_ID": "H",
        "Atom_chem_shift.Atom_type": "H",
        "Atom_chem_shift.Atom_isotope_number": "1",
        "Atom_chem_shift.Val": "8.0740",
        "Atom_chem_shift.Val_err": "0.01",
        "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": "3",
        "Atom_chem_shift.Auth_comp_ID": "ASP",
        "Atom_chem_shift.Auth_atom_ID": "H",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "8",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "3",
        "Atom_chem_shift.Seq_ID": "3",
        "Atom_chem_shift.Comp_ID": "ASP",
        "Atom_chem_shift.Atom_ID": "HA",
        "Atom_chem_shift.Atom_type": "H",
        "Atom_chem_shift.Atom_isotope_number": "1",
        "Atom_chem_shift.Val": "4.5580",
        "Atom_chem_shift.Val_err": "0.01",
        "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": "3",
        "Atom_chem_shift.Auth_comp_ID": "ASP",
        "Atom_chem_shift.Auth_atom_ID": "HA",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "9",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
```

```
        "Atom_chem_shift.Comp_index_ID": "3",
        "Atom_chem_shift.Seq_ID": "3",
        "Atom_chem_shift.Comp_ID": "ASP",
        "Atom_chem_shift.Atom_ID": "HB2",
        "Atom_chem_shift.Atom_type": "H",
        "Atom_chem_shift.Atom_isotope_number": "1",
        "Atom_chem_shift.Val": "2.835",
        "Atom_chem_shift.Val_err": "0.01",
        "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": "3",
        "Atom_chem_shift.Auth_comp_ID": "ASP",
        "Atom_chem_shift.Auth_atom_ID": "HB2",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "10",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "3",
        "Atom_chem_shift.Seq_ID": "3",
        "Atom_chem_shift.Comp_ID": "ASP",
        "Atom_chem_shift.Atom_ID": "HB3",
        "Atom_chem_shift.Atom_type": "H",
        "Atom_chem_shift.Atom_isotope_number": "1",
        "Atom_chem_shift.Val": "2.754",
        "Atom_chem_shift.Val_err": "0.01",
        "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": "3",
        "Atom_chem_shift.Auth_comp_ID": "ASP",
        "Atom_chem_shift.Auth_atom_ID": "HB3",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "11",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "3",
        "Atom_chem_shift.Seq_ID": "3",
        "Atom_chem_shift.Comp_ID": "ASP",
        "Atom_chem_shift.Atom_ID": "CA",
        "Atom_chem_shift.Atom_type": "C",
        "Atom_chem_shift.Atom_isotope_number": "13",
        "Atom_chem_shift.Val": "57.6400",
```

```
        "Atom_chem_shift.Val_err": "0.1",
        "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": "3",
        "Atom_chem_shift.Auth_comp_ID": "ASP",
        "Atom_chem_shift.Auth_atom_ID": "CA",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "12",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "3",
        "Atom_chem_shift.Seq_ID": "3",
        "Atom_chem_shift.Comp_ID": "ASP",
        "Atom_chem_shift.Atom_ID": "N",
        "Atom_chem_shift.Atom_type": "N",
        "Atom_chem_shift.Atom_isotope_number": "15",
        "Atom_chem_shift.Val": "121.1040",
        "Atom_chem_shift.Val_err": "0.1",
        "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": "3",
        "Atom_chem_shift.Auth_comp_ID": "ASP",
        "Atom_chem_shift.Auth_atom_ID": "N",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "13",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "4",
        "Atom_chem_shift.Seq_ID": "4",
        "Atom_chem_shift.Comp_ID": "GLU",
        "Atom_chem_shift.Atom_ID": "H",
        "Atom_chem_shift.Atom_type": "H",
        "Atom_chem_shift.Atom_isotope_number": "1",
        "Atom_chem_shift.Val": "8.6520",
        "Atom_chem_shift.Val_err": "0.01",
        "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": "4",
        "Atom_chem_shift.Auth_comp_ID": "GLU",
        "Atom_chem_shift.Auth_atom_ID": "H",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "14",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "4",
        "Atom_chem_shift.Seq_ID": "4",
        "Atom_chem_shift.Comp_ID": "GLU",
        "Atom_chem_shift.Atom_ID": "HA",
        "Atom_chem_shift.Atom_type": "H",
        "Atom_chem_shift.Atom_isotope_number": "1",
        "Atom_chem_shift.Val": "4.1420",
        "Atom_chem_shift.Val_err": "0.01",
        "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": "4",
        "Atom_chem_shift.Auth_comp_ID": "GLU",
        "Atom_chem_shift.Auth_atom_ID": "HA",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "15",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "4",
        "Atom_chem_shift.Seq_ID": "4",
        "Atom_chem_shift.Comp_ID": "GLU",
        "Atom_chem_shift.Atom_ID": "HB2",
        "Atom_chem_shift.Atom_type": "H",
        "Atom_chem_shift.Atom_isotope_number": "1",
        "Atom_chem_shift.Val": "2.0520",
        "Atom_chem_shift.Val_err": "0.01",
        "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": "4",
        "Atom_chem_shift.Auth_comp_ID": "GLU",
        "Atom_chem_shift.Auth_atom_ID": "HB2",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
```

```
{
    "Atom_chem_shift.ID": "16",
    "Atom_chem_shift.Assembly_atom_ID": ".",
    "Atom_chem_shift.Entity_assembly_ID": "1",
    "Atom_chem_shift.Entity_ID": "1",
    "Atom_chem_shift.Comp_index_ID": "4",
    "Atom_chem_shift.Seq_ID": "4",
    "Atom_chem_shift.Comp_ID": "GLU",
    "Atom_chem_shift.Atom_ID": "HB3",
    "Atom_chem_shift.Atom_type": "H",
    "Atom_chem_shift.Atom_isotope_number": "1",
    "Atom_chem_shift.Val": "2.0320",
    "Atom_chem_shift.Val_err": "0.01",
    "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": "4",
    "Atom_chem_shift.Auth_comp_ID": "GLU",
    "Atom_chem_shift.Auth_atom_ID": "HB3",
    "Atom_chem_shift.Details": ".",
    "Atom_chem_shift.Entry_ID": "15000",
    "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
},
{
    "Atom_chem_shift.ID": "17",
    "Atom_chem_shift.Assembly_atom_ID": ".",
    "Atom_chem_shift.Entity_assembly_ID": "1",
    "Atom_chem_shift.Entity_ID": "1",
    "Atom_chem_shift.Comp_index_ID": "4",
    "Atom_chem_shift.Seq_ID": "4",
    "Atom_chem_shift.Comp_ID": "GLU",
    "Atom_chem_shift.Atom_ID": "HG2",
    "Atom_chem_shift.Atom_type": "H",
    "Atom_chem_shift.Atom_isotope_number": "1",
    "Atom_chem_shift.Val": "2.4540",
    "Atom_chem_shift.Val_err": "0.01",
    "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": "4",
    "Atom_chem_shift.Auth_comp_ID": "GLU",
    "Atom_chem_shift.Auth_atom_ID": "HG2",
    "Atom_chem_shift.Details": ".",
    "Atom_chem_shift.Entry_ID": "15000",
    "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
},
{
    "Atom_chem_shift.ID": "18",
    "Atom_chem_shift.Assembly_atom_ID": ".",
    "Atom_chem_shift.Entity_assembly_ID": "1",
    "Atom_chem_shift.Entity_ID": "1",
    "Atom_chem_shift.Comp_index_ID": "4",
    "Atom_chem_shift.Seq_ID": "4",
```

```
        "Atom_chem_shift.Comp_ID": "GLU",
        "Atom_chem_shift.Atom_ID": "CB",
        "Atom_chem_shift.Atom_type": "C",
        "Atom_chem_shift.Atom_isotope_number": "13",
        "Atom_chem_shift.Val": "28.1200",
        "Atom_chem_shift.Val_err": "0.1",
        "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": "4",
        "Atom_chem_shift.Auth_comp_ID": "GLU",
        "Atom_chem_shift.Auth_atom_ID": "CB",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "19",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "4",
        "Atom_chem_shift.Seq_ID": "4",
        "Atom_chem_shift.Comp_ID": "GLU",
        "Atom_chem_shift.Atom_ID": "CG",
        "Atom_chem_shift.Atom_type": "C",
        "Atom_chem_shift.Atom_isotope_number": "13",
        "Atom_chem_shift.Val": "33.2720",
        "Atom_chem_shift.Val_err": "0.1",
        "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": "4",
        "Atom_chem_shift.Auth_comp_ID": "GLU",
        "Atom_chem_shift.Auth_atom_ID": "CG",
        "Atom_chem_shift.Details": ".",
        "Atom_chem_shift.Entry_ID": "15000",
        "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
    },
    {
        "Atom_chem_shift.ID": "20",
        "Atom_chem_shift.Assembly_atom_ID": ".",
        "Atom_chem_shift.Entity_assembly_ID": "1",
        "Atom_chem_shift.Entity_ID": "1",
        "Atom_chem_shift.Comp_index_ID": "4",
        "Atom_chem_shift.Seq_ID": "4",
        "Atom_chem_shift.Comp_ID": "GLU",
        "Atom_chem_shift.Atom_ID": "N",
        "Atom_chem_shift.Atom_type": "N",
        "Atom_chem_shift.Atom_isotope_number": "15",
        "Atom_chem_shift.Val": "119.8900",
        "Atom_chem_shift.Val_err": "0.1",
        "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": "4",
                    "Atom_chem_shift.Auth_comp_ID": "GLU",
                    "Atom_chem_shift.Auth_atom_ID": "N",
                    "Atom_chem_shift.Details": ".",
                    "Atom_chem_shift.Entry_ID": "15000",
                    "Atom_chem_shift.Assigned_chem_shift_list_ID": "1"
                }
            ]
        ]
    }
}
```

```
In [22]: starfile.print_saveframe("save_entry_information", file_format="nmrstar")
```

```
   _Entry.Sf_category  entry_information
   _Entry.Sf_framecode         entry_information
   _Entry.ID    15000
   _Entry.Title
;

Solution structure of chicken villin headpiece subdomain containing a fluorinated side chain in the c
;
   _Entry.Type  macromolecule
   _Entry.Version_type         original
   _Entry.Submission_date      2006-09-07
   _Entry.Accession_date       2006-09-07
   _Entry.Last_release_date   .
   _Entry.Original_release_date       .
   _Entry.Origination   author
   _Entry.NMR_STAR_version    3.1.1.61
   _Entry.Original_NMR_STAR_version  .
   _Entry.Experimental_method  NMR
   _Entry.Experimental_method_subtype         solution
   _Entry.Details       .
   _Entry.BMRB_internal_directory_name       .

   loop_
      _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

      1 Claudia Cornilescu . C. . 15000
      2 Gabriel Cornilescu . . . 15000
      3 Erik Hadley . B. . 15000
      4 Samuel Gellman . H. . 15000
      5 John Markley . L. . 15000

   stop_

   loop_
      _Datum.Type
```

```
      _Datum.Count
      _Datum.Entry_ID

      '13C chemical shifts' 77 15000
      '15N chemical shifts' 40 15000
      '1H chemical shifts' 223 15000

   stop_
```

```
In [23]: starfile.print_saveframe("save_entry_information", file_format="json")
```


```
{
    "Entry.Sf_category": "entry_information",
    "Entry.Sf_framecode": "entry_information",
    "Entry.ID": "15000",
    "Entry.Title": "\nSolution structure of chicken villin headpiece subdomain containing a fluorinat
    "Entry.Type": "macromolecule",
    "Entry.Version_type": "original",
    "Entry.Submission_date": "2006-09-07",
    "Entry.Accession_date": "2006-09-07",
    "Entry.Last_release_date": ".",
    "Entry.Original_release_date": ".",
    "Entry.Origination": "author",
    "Entry.NMR_STAR_version": "3.1.1.61",
    "Entry.Original_NMR_STAR_version": ".",
    "Entry.Experimental_method": "NMR",
    "Entry.Experimental_method_subtype": "solution",
    "Entry.Details": ".",
    "Entry.BMRB_internal_directory_name": ".",
    "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"
        ],
        [
            {
                "Entry_author.Ordinal": "1",
                "Entry_author.Given_name": "Claudia",
                "Entry_author.Family_name": "Cornilescu",
                "Entry_author.First_initial": ".",
                "Entry_author.Middle_initials": "C.",
                "Entry_author.Family_title": ".",
                "Entry_author.Entry_ID": "15000"
            },
            {
                "Entry_author.Ordinal": "2",
                "Entry_author.Given_name": "Gabriel",
                "Entry_author.Family_name": "Cornilescu",
                "Entry_author.First_initial": ".",
                "Entry_author.Middle_initials": ".",
                "Entry_author.Family_title": ".",
                "Entry_author.Entry_ID": "15000"
            },
            {
                "Entry_author.Ordinal": "3",
```

```
                "Entry_author.Given_name": "Erik",
                "Entry_author.Family_name": "Hadley",
                "Entry_author.First_initial": ".",
                "Entry_author.Middle_initials": "B.",
                "Entry_author.Family_title": ".",
                "Entry_author.Entry_ID": "15000"
            },
            {
                "Entry_author.Ordinal": "4",
                "Entry_author.Given_name": "Samuel",
                "Entry_author.Family_name": "Gellman",
                "Entry_author.First_initial": ".",
                "Entry_author.Middle_initials": "H.",
                "Entry_author.Family_title": ".",
                "Entry_author.Entry_ID": "15000"
            },
            {
                "Entry_author.Ordinal": "5",
                "Entry_author.Given_name": "John",
                "Entry_author.Family_name": "Markley",
                "Entry_author.First_initial": ".",
                "Entry_author.Middle_initials": "L.",
                "Entry_author.Family_title": ".",
                "Entry_author.Entry_ID": "15000"
            }
        ]
    ],
    "loop_1": [
        [
            "Datum.Type",
            "Datum.Count",
            "Datum.Entry_ID"
        ],
        [
            {
                "Datum.Type": "13C chemical shifts",
                "Datum.Count": "77",
                "Datum.Entry_ID": "15000"
            },
            {
                "Datum.Type": "15N chemical shifts",
                "Datum.Count": "40",
                "Datum.Entry_ID": "15000"
            },
            {
                "Datum.Type": "1H chemical shifts",
                "Datum.Count": "223",
                "Datum.Entry_ID": "15000"
            }
        ]
    ]
}
```


```
In [24]: starfile.print_loop("save_entry_information", "loop_0", file_format="nmrstar")
```

```
   _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

   1 Claudia Cornilescu . C. . 15000
   2 Gabriel Cornilescu . . . 15000
   3 Erik Hadley . B. . 15000
   4 Samuel Gellman . H. . 15000
   5 John Markley . L. . 15000
```

```
In [25]: starfile.print_loop("save_entry_information", "loop_0", file_format="json")
```


```
[
    [
        "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"
    ],
    [
        {
            "Entry_author.Ordinal": "1",
            "Entry_author.Given_name": "Claudia",
            "Entry_author.Family_name": "Cornilescu",
            "Entry_author.First_initial": ".",
            "Entry_author.Middle_initials": "C.",
            "Entry_author.Family_title": ".",
            "Entry_author.Entry_ID": "15000"
        },
        {
            "Entry_author.Ordinal": "2",
            "Entry_author.Given_name": "Gabriel",
            "Entry_author.Family_name": "Cornilescu",
            "Entry_author.First_initial": ".",
            "Entry_author.Middle_initials": ".",
            "Entry_author.Family_title": ".",
            "Entry_author.Entry_ID": "15000"
        },
        {
            "Entry_author.Ordinal": "3",
            "Entry_author.Given_name": "Erik",
            "Entry_author.Family_name": "Hadley",
            "Entry_author.First_initial": ".",
            "Entry_author.Middle_initials": "B.",
            "Entry_author.Family_title": ".",
            "Entry_author.Entry_ID": "15000"
        },
        {
            "Entry_author.Ordinal": "4",
            "Entry_author.Given_name": "Samuel",
            "Entry_author.Family_name": "Gellman",
            "Entry_author.First_initial": ".",
            "Entry_author.Middle_initials": "H.",
            "Entry_author.Family_title": ".",
            "Entry_author.Entry_ID": "15000"
        },
        {
            "Entry_author.Ordinal": "5",
```

```
            "Entry_author.Given_name": "John",
            "Entry_author.Family_name": "Markley",
            "Entry_author.First_initial": ".",
            "Entry_author.Middle_initials": "L.",
            "Entry_author.Family_title": ".",
            "Entry_author.Entry_ID": "15000"
        }
    ]
]
```

- Accessing chemical shift data:

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

```
In [26]: starfile["save_assigned_chem_shift_list_1"]["loop_0"][0]

Out[26]: ['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']

In [27]: starfile["save_assigned_chem_shift_list_1"]["loop_0"][1][0]["Atom_chem_shift.Seq_ID"]

Out[27]: '2'

In [28]: starfile["save_assigned_chem_shift_list_1"]["loop_0"][1][0]["Atom_chem_shift.Comp_ID"]

Out[28]: 'SER'

In [29]: starfile["save_assigned_chem_shift_list_1"]["loop_0"][1][0]["Atom_chem_shift.Atom_ID"]

Out[29]: 'H'

In [30]: starfile["save_assigned_chem_shift_list_1"]["loop_0"][1][0]["Atom_chem_shift.Val"]

Out[30]: '9.3070'

In [31]: starfile["save_assigned_chem_shift_list_1"]["loop_0"][1][1]["Atom_chem_shift.Atom_ID"]

Out[31]: 'HA'

In [32]: starfile["save_assigned_chem_shift_list_1"]["loop_0"][1][1]["Atom_chem_shift.Val"]

Out[32]: '4.5970'
```

```
In [33]: starfile["save_assigned_chem_shift_list_1"]["loop_0"][1][2]["Atom_chem_shift.Atom_ID"]
```

```
Out[33]: 'HB2'
```

```
In [34]: starfile["save_assigned_chem_shift_list_1"]["loop_0"][1][2]["Atom_chem_shift.Val"]
```

```
Out[34]: '4.3010'
```

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:

```
In [35]: # access all chemical shifts
         starfile.chem_shifts_by_residue()
```

```
Out[35]: [OrderedDict([('2',
                         OrderedDict([('AA3Code', 'SER'),
                                      ('Seq_ID', '2'),
                                      ('H', '9.3070'),
                                      ('HA', '4.5970'),
                                      ('HB2', '4.3010'),
                                      ('HB3', '4.0550'),
                                      ('CB', '64.6000'),
                                      ('N', '121.5800')])),
                        ('3',
                         OrderedDict([('AA3Code', 'ASP'),
                                      ('Seq_ID', '3'),
                                      ('H', '8.0740'),
                                      ('HA', '4.5580'),
                                      ('HB2', '2.835'),
                                      ('HB3', '2.754'),
                                      ('CA', '57.6400'),
                                      ('N', '121.1040')])),
                        ('4',
                         OrderedDict([('AA3Code', 'GLU'),
                                      ('Seq_ID', '4'),
                                      ('H', '8.6520'),
                                      ('HA', '4.1420'),
                                      ('HB2', '2.0520'),
                                      ('HB3', '2.0320'),
                                      ('HG2', '2.4540'),
                                      ('CB', '28.1200'),
                                      ('CG', '33.2720'),
                                      ('N', '119.8900')]))])]
```

```
In [36]: # access chemical shifts for "SER" and "GLU" amino acids
         starfile.chem_shifts_by_residue(amino_acids=["SER", "GLU"])
```

```
Out[36]: [OrderedDict([('2',
                         OrderedDict([('AA3Code', 'SER'),
                                      ('Seq_ID', '2'),
                                      ('H', '9.3070'),
                                      ('HA', '4.5970'),
                                      ('HB2', '4.3010'),
                                      ('HB3', '4.0550'),
                                      ('CB', '64.6000'),
                                      ('N', '121.5800')])),
                        ('4',
                         OrderedDict([('AA3Code', 'GLU'),
                                      ('Seq_ID', '4'),
                                      ('H', '8.6520'),
                                      ('HA', '4.1420'),
```

```
                                        ('HB2', '2.0520'),
                                        ('HB3', '2.0320'),
                                        ('HG2', '2.4540'),
                                        ('CB', '28.1200'),
                                        ('CG', '33.2720'),
                                        ('N', '119.8900')]))])]
```

```
In [37]: # access chemical shifts for "SER" and "GLU" amino acids for "CB" and "CG" atoms
         starfile.chem_shifts_by_residue(amino_acids=["SER", "GLU"], atoms=["CB", "CG"])
```

```
Out[37]: [OrderedDict([('2',
                         OrderedDict([('AA3Code', 'SER'),
                                      ('Seq_ID', '2'),
                                      ('CB', '64.6000')])),
                        ('4',
                         OrderedDict([('AA3Code', 'GLU'),
                                      ('Seq_ID', '4'),
                                      ('CB', '28.1200'),
                                      ('CG', '33.2720')]))])]
```

```
In [38]: # acceess chemical shifts for specific amino acid and specific atom
         starfile.chem_shifts_by_residue(amino_acids_and_atoms={"SER":["HA", "HB2", "HB3"], "ASP": ['
```

```
Out[38]: [OrderedDict([('2',
                         OrderedDict([('AA3Code', 'SER'),
                                      ('Seq_ID', '2'),
                                      ('HA', '4.5970'),
                                      ('HB2', '4.3010'),
                                      ('HB3', '4.0550')])),
                        ('3',
                         OrderedDict([('AA3Code', 'ASP'),
                                      ('Seq_ID', '3'),
                                      ('CA', '57.6400'),
                                      ('N', '121.1040')]))])]
```

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:

```
In [39]: with open("out/bmr15000_modified.str", "w") as outfile:
             starfile.write(outfile, file_format="nmrstar")
```

- Writing into a JSONized NMR-STAR formatted file:

```
In [40]: with open("out/bmr15000_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:

```
In [41]: 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="out/bmr18569.json",
                                                  from_format="nmrstar", to_format="json"))
         converter.convert()
```

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

```
In [42]: from nmrstarlib.converter import Converter
         from nmrstarlib.translator import StarFileToStarFile

         # Using generated above "bmr18569.json" file
         converter = Converter(StarFileToStarFile(from_path="bmr18569.json", to_path="out/bmr18569.st
                                                  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:

```
In [43]: from nmrstarlib.converter import Converter
         from nmrstarlib.translator import StarFileToStarFile

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

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

```
In [44]: from nmrstarlib.converter import Converter
         from nmrstarlib.translator import StarFileToStarFile

         converter = Converter(StarFileToStarFile(from_path="starfiles_dir_json", to_path="out/starfi
                                                  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

Creating simulated peak lists without variance

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:

```
In [45]: 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="out/18569_HNcoCACB.txt"
                                          from_format="nmrstar", to_format="sparky",
                                          spectrum_name="HNcoCACB"))
converter.convert()
```

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

```
Assignment           w1              w2              w3

SER2H-SER2N-MET1CA          8.225           117.197         55.489
SER2H-SER2N-MET1CB          8.225           117.197         32.848
GLU3H-GLU3N-SER2CA          8.002           119.833         58.593
GLU3H-GLU3N-SER2CB          8.002           119.833         64.057
THR4H-THR4N-GLU3CA          8.956           117.212         55.651
THR4H-THR4N-GLU3CB          8.956           117.212         32.952
...
```

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

```
In [46]: 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="out/18569_HNcoCACB.jso
                                                  from_format="nmrstar", to_format="json",
                                                  spectrum_name="HNcoCACB"))
         converter.convert()
```

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

```
[
 {"Assignment": ["SER2H", "SER2N", "MET1CA"], "Dimensions": [8.225, 117.197, 55.489]},
 {"Assignment": ["SER2H", "SER2N", "MET1CB"], "Dimensions": [8.225, 117.197, 32.848]},
 {"Assignment": ["GLU3H", "GLU3N", "SER2CA"], "Dimensions": [8.002, 119.833, 58.593]},
 {"Assignment": ["GLU3H", "GLU3N", "SER2CB"], "Dimensions": [8.002, 119.833, 64.057]},
 {"Assignment": ["THR4H", "THR4N", "GLU3CA"], "Dimensions": [8.956, 117.212, 55.651]},
 {"Assignment": ["THR4H", "THR4N", "GLU3CB"], "Dimensions": [8.956, 117.212, 32.952]},
 ...
]
```

Creating simulated peak lists with variance drawn from random normal distribution

- 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:

```
In [47]: from nmrstarlib.converter import Converter
         from nmrstarlib.translator import StarFileToPeakList
         from nmrstarlib.noise import NoiseGenerator

         # create parameters dictionary for random normal distribution
         parameters = {"H_loc": [0], "C_loc": [0], "N_loc": [0],
                       "H_scale": [0.001], "C_scale": [0.01], "N_scale": [0.01]}

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

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

```
converter = Converter(StarFileToPeakList(from_path="18569", to_path="out/18569_HNcoCACB_ssv_
                                          from_format="nmrstar", to_format="sparky",
                                          spectrum_name="HNcoCACB",
                                          noise_generator=random_normal_noise_generator))
converter.convert()
```

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

```
Assignment           w1              w2              w3

SER2H-SER2N-MET1CA          8.226026                117.193655              55.477204
SER2H-SER2N-MET1CB          8.224649                117.184255              32.845212
GLU3H-GLU3N-SER2CA          8.003282                119.841221              58.603253
GLU3H-GLU3N-SER2CB          8.002372                119.827019              64.067278
THR4H-THR4N-GLU3CA          8.955568                117.215237              55.663902
THR4H-THR4N-GLU3CB          8.955757                117.206167              32.96412
```

- 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:

```
In [48]: from nmrstarlib.converter import Converter
         from nmrstarlib.translator import StarFileToPeakList
         from nmrstarlib.noise import NoiseGenerator

         # create parameters dictionary for random normal distribution
         parameters = {"H_loc": [0], "C_loc": [None], "N_loc": [0],
                       "H_scale": [0.001], "C_scale": [None], "N_scale": [0.01]}

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

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

The generated *18569_HNcoCACB_ssv_HN.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

SER2H-SER2N-MET1CA          8.226085                117.191527              55.489
SER2H-SER2N-MET1CB          8.224509                117.204666              32.848
GLU3H-GLU3N-SER2CA          8.001657                119.846806              58.593
GLU3H-GLU3N-SER2CB          8.003165                119.8268                64.057
THR4H-THR4N-GLU3CA          8.956946                117.209486              55.651
THR4H-THR4N-GLU3CB          8.955755                117.209889              32.952
```

- 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:

```
In [49]: from nmrstarlib.converter import Converter
         from nmrstarlib.translator import StarFileToPeakList
         from nmrstarlib.noise import NoiseGenerator
```

```
# create parameters dictionary for random normal distribution
parameters = {"H_loc": [0, 0], "C_loc": [None, None], "N_loc": [0, 0],
              "H_scale": [0.001, 0.005], "C_scale": [None, None], "N_scale": [0.01, 0.05]}

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

# Using valid BMRB id to access file from URL: from_path="18569"
converter = Converter(StarFileToPeakList(from_path="18569", to_path="out/18569_HNcoCACB_tsv_
                                         from_format="nmrstar", to_format="sparky",
                                         spectrum_name="HNcoCACB",
                                         plsplit=(70,30),
                                         noise_generator=random_normal_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

SER2H-SER2N-MET1CA            8.223356              117.208041              55.489
SER2H-SER2N-MET1CB            8.22532             117.184278          32.848
GLU3H-GLU3N-SER2CA            8.00271             119.847153          58.593
GLU3H-GLU3N-SER2CB            8.002822              119.824752              64.057
...
GLU114H-GLU114N-LEU113CA            7.614195                118.672897
↪56.14
GLU114H-GLU114N-LEU113CB            7.628722                118.565859
↪43.249
GLY115H-GLY115N-GLU114CA            7.583248                113.45153
↪57.005
GLY115H-GLY115N-GLU114CB            7.596634                113.472049
↪30.079
```

Creating simulated peak lists with variance drawn from other distribution types

- It is also possible to generate the simulated peak lists using other types of statistical distribution functions. For example, let's simulate the peak list using noise values drawn from `chisquare` distribution for 5 degrees of freedom for *H* and *N* dimensions from single source of variance.

```
In [50]: from nmrstarlib.converter import Converter
         from nmrstarlib.translator import StarFileToPeakList
         from nmrstarlib.noise import NoiseGenerator

         # create parameters dictionary for distribution
         parameters = {"H_df": [5], "C_df": [None], "N_df": [5]}

         # create chisquare noise generator
         chisquare_noise_generator = NoiseGenerator(parameters, distribution_name="chisquare")

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

The generated *18569_HNcoCACB_ssv_HN_chi2.txt* peak list file should look like the following:

```
Assignment          w1              w2              w3

SER2H-SER2N-MET1CA          12.50083            127.197738              55.489
SER2H-SER2N-MET1CB          10.495158           121.039655              32.848
GLU3H-GLU3N-SER2CA          15.597162           124.603078              58.593
GLU3H-GLU3N-SER2CB          8.340404            126.784481              64.057
THR4H-THR4N-GLU3CA          10.010804           120.476893              55.651
THR4H-THR4N-GLU3CB          11.961498           121.681636              32.952
```

- Below is the list of all supported distribution functions along with their parameters if the `numpy` library is not installed:

```
{
    {"function": "uniform", "parameters": ["low", "high"]},
    {"function": "triangular", "parameters": ["left", "right", "mode"]},
    {"function": "beta", "parameters": ["a", "b"]},
    {"function": "exponential", "parameters": ["scale"]},
    {"function": "gamma", "parameters": ["shape", "scale"]},
    {"function": "gauss", "parameters": ["mu", "sigma"]},
    {"function": "normal", "parameters": ["loc", "scale"]},
    {"function": "lognormal", "parameters": ["mean", "sigma"]},
    {"function": "vonmises", "parameters": ["mu", "kappa"]},
    {"function": "pareto", "parameters": ["a"]}
}
```

- And the list of all supported distribution functions along with their parameters if the `numpy` library is installed:

```
{
    {"function": "beta", "parameters": ["a", "b"]},
    {"function": "binomial", "parameters": ["n", "p"]},
    {"function": "chisquare", "parameters": ["df"]},
    {"function": "exponential", "parameters": ["scale"]},
    {"function": "f", "parameters": ["dfnum", "dfden"]},
    {"function": "gamma", "parameters": ["shape", "scale"]},
    {"function": "geometric", "parameters": ["p"]},
    {"function": "gumbel", "parameters": ["loc", "scale"]},
    {"function": "hypergeometric", "parameters": ["ngood", "nbad", "nsample"]},
    {"function": "laplace", "parameters": ["loc", "scale"]},
    {"function": "logistic", "parameters": ["loc", "scale"]},
    {"function": "lognormal", "parameters": ["mean", "sigma"]},
    {"function": "logseries", "parameters": ["p"]},
    {"function": "negative_binomial", "parameters": ["n", "p"]},
    {"function": "noncentral_chisquare", "parameters": ["df", "nonc"]},
    {"function": "noncentral_f", "parameters": ["dfnum", "dfden", "nonc"]},
    {"function": "normal", "parameters": ["loc", "scale"]},
    {"function": "pareto", "parameters": ["a"]},
    {"function": "poisson", "parameters": ["lam"]},
    {"function": "power", "parameters": ["a"]},
    {"function": "rayleigh", "parameters": ["scale"]},
    {"function": "triangular", "parameters": ["left", "mode", "right"]},
    {"function": "uniform", "parameters": ["low", "high"]},
    {"function": "vonmises", "parameters": ["mu", "kappa"]},
    {"function": "wald", "parameters": ["mean", "scale"]},
    {"function": "weibull", "parameters": ["a"]},
    {"function": "zipf", "parameters": ["a"]}
}
```

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:

```
In [51]: 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:

```
In [52]: 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}]},
 'CBCANH': {'Labels': ['CA/CB', 'H', 'N'],
            'MinNumberPeaksPerSpinSystem': 2,
            'PeakDescriptions': [{'dimensions': ['CA', 'H', 'N'], 'fraction': 1},
                                 {'dimensions': ['CB', 'H', 'N'], 'fraction': 0.95},
                                 {'dimensions': ['CA', 'H+1', 'N+1'], 'fraction': 1},
                                 {'dimensions': ['CB', 'H+1', 'N+1'], 'fraction': 0.95}]},
 'CBCAcoNH': {'Labels': ['CA/CB', 'H+1', 'N+1'],
              'MinNumberPeaksPerSpinSystem': 2,
              'PeakDescriptions': [{'dimensions': ['CA', 'H+1', 'N+1'], 'fraction': 1},
                                   {'dimensions': ['CB', 'H+1', 'N+1'], 'fraction': 0.95}]},
 'CCcoNH': {'Labels': ['CX-1', 'N', 'H'],
            'MinNumberPeaksPerSpinSystem': 2,
            'PeakDescriptions': [{'dimensions': ['CA-1', 'N', 'H'], 'fraction': 1},
                                 {'dimensions': ['CB-1', 'N', 'H'], 'fraction': 1},
                                 {'dimensions': ['CG-1', 'N', 'H'], 'fraction': 1},
                                 {'dimensions': ['CD-1', 'N', 'H'], 'fraction': 1},
                                 {'dimensions': ['CE-1', 'N', 'H'], 'fraction': 1},
```

```
                                {'dimensions': ['CZ-1', 'N', 'H'], 'fraction': 1}],
           'ResonanceLimit': {'ALA': ['H', 'N', 'CA', 'CB'],
                              'ARG': ['H', 'N', 'CA', 'CB', 'CG', 'CD', 'CZ'],
                              'ASN': ['H', 'N', 'CA', 'CB', 'CG'],
                              'ASP': ['H', 'N', 'CA', 'CB', 'CG'],
                              'CYS': ['H', 'N', 'CA', 'CB'],
                              'GLN': ['H', 'N', 'CA', 'CB', 'CG', 'CD'],
                              'GLU': ['H', 'N', 'CA', 'CB', 'CG', 'CD'],
                              'GLY': ['H', 'N', 'CA'],
                              'HIS': ['H', 'N', 'CA', 'CB'],
                              'ILE': ['H', 'N', 'CA', 'CB', 'CG1', 'CG2', 'CD1'],
                              'LEU': ['H', 'N', 'CA', 'CB', 'CG', 'CD1', 'CD2'],
                              'LYS': ['H', 'N', 'CA', 'CB', 'CG', 'CD', 'CE'],
                              'MET': ['H', 'N', 'CA', 'CB', 'CG', 'CE'],
                              'PHE': ['H', 'N', 'CA', 'CB'],
                              'SER': ['H', 'N', 'CA', 'CB'],
                              'THR': ['H', 'N', 'CA', 'CB', 'CG2'],
                              'TRP': ['H', 'N', 'CA', 'CB'],
                              'TYR': ['H', 'N', 'CA', 'CB'],
                              'VAL': ['H', 'N', 'CA', 'CB', 'CG1', 'CG2']}},
 'HBHAcoNH': {'Labels': ['HA/HB-1', 'N', 'H'],
              'MinNumberPeaksPerSpinSystem': 2,
              'PeakDescriptions': [{'dimensions': ['HA-1', 'N', 'H'], 'fraction': 1},
                                   {'dimensions': ['HB-1', 'N', 'H'], 'fraction': 1}]},
 'HNCA': {'Labels': ['H', 'N', 'CA'],
          'MinNumberPeaksPerSpinSystem': 1,
          'PeakDescriptions': [{'dimensions': ['H', 'N', 'CA'], 'fraction': 1},
                               {'dimensions': ['H', 'N', 'CA-1'], 'fraction': 1}]},
 'HNCACB': {'Labels': ['H', 'N', 'CA/CB'],
            'MinNumberPeaksPerSpinSystem': 2,
            'PeakDescriptions': [{'dimensions': ['H', 'N', 'CA'], 'fraction': 1},
                                 {'dimensions': ['H', 'N', 'CB'], 'fraction': 0.95},
                                 {'dimensions': ['H', 'N', 'CA-1'], 'fraction': 1},
                                 {'dimensions': ['H', 'N', 'CB-1'], 'fraction': 0.95}]},
 'HNCO': {'Labels': ['H', 'N', 'CO-1'],
          'MinNumberPeaksPerSpinSystem': 1,
          'PeakDescriptions': [{'dimensions': ['H', 'N', 'CO-1'], 'fraction': 1}]},
 'HNcaCO': {'Labels': ['H', 'N', 'CO'],
            'MinNumberPeaksPerSpinSystem': 1,
            'PeakDescriptions': [{'dimensions': ['H', 'N', 'CO'], 'fraction': 1},
                                 {'dimensions': ['H', 'N', 'CO-1'], 'fraction': 1}]},
 'HNcoCA': {'Labels': ['H', 'N', 'CA'],
            'MinNumberPeaksPerSpinSystem': 1,
            'PeakDescriptions': [{'dimensions': ['H', 'N', 'CA-1'], 'fraction': 1}]},
 '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}]},
 'HSQC': {'Labels': ['H', 'N'],
          'MinNumberPeaksPerSpinSystem': 1,
          'PeakDescriptions': [{'dimensions': ['H', 'N'], 'fraction': 1}]},
 'HccoNH': {'Labels': ['HX-1', 'N', 'H'],
            'MinNumberPeaksPerSpinSystem': 2,
            'PeakDescriptions': [{'dimensions': ['HA-1', 'N', 'H'], 'fraction': 1},
                                 {'dimensions': ['HB-1', 'N', 'H'], 'fraction': 1},
                                 {'dimensions': ['HG-1', 'N', 'H'], 'fraction': 1},
                                 {'dimensions': ['HD-1', 'N', 'H'], 'fraction': 1},
                                 {'dimensions': ['HE-1', 'N', 'H'], 'fraction': 1},
                                 {'dimensions': ['HH-1', 'N', 'H'], 'fraction': 1},
```

```
                                {'dimensions': ['HZ-1', 'N', 'H'], 'fraction': 1}],
            'ResonanceLimit': {'ALA': ['H', 'N', 'HA', 'HB'],
                               'ARG': ['H', 'N', 'HA', 'HB2', 'HB3', 'HG2', 'HG3', 'HD2', 'HD3'],
                               'ASN': ['H', 'N', 'HA', 'HB2', 'HB3'],
                               'ASP': ['H', 'N', 'HA', 'HB2', 'HB3'],
                               'CYS': ['H', 'N', 'HA', 'HB2', 'HB3'],
                               'GLN': ['H', 'N', 'HA', 'HB2', 'HB3', 'HG2', 'HG3'],
                               'GLU': ['H', 'N', 'HA', 'HB2', 'HB3', 'HG2', 'HG3'],
                               'GLY': ['H', 'N', 'HA2', 'HA3'],
                               'HIS': ['H', 'N', 'HA', 'HB2', 'HB3'],
                               'ILE': ['H', 'N', 'HA', 'HB', 'HG12', 'HG13', 'HG2', 'HD1'],
                               'LEU': ['H', 'N', 'HA', 'HB2', 'HB3', 'HD1', 'HD2'],
                               'LYS': ['H', 'N', 'HA', 'HB2', 'HB3', 'HG2', 'HG3', 'HD2', 'HD3', 'HE2
                               'MET': ['H', 'N', 'HA', 'HB2', 'HB3', 'HG2', 'HG3', 'HE'],
                               'PHE': ['H', 'N', 'HA', 'HB2', 'HB3'],
                               'SER': ['H', 'N', 'HA', 'HB2', 'HB3'],
                               'THR': ['H', 'N', 'HA', 'HB', 'HG2'],
                               'TRP': ['H', 'N', 'HA', 'HB2', 'HB3'],
                               'TYR': ['H', 'N', 'HA', 'HB2', 'HB3'],
                               'VAL': ['H', 'N', 'HA', 'HB', 'HG1', 'HG2']}},
 'NCA': {'Labels': ['N', 'CA'],
         'MinNumberPeaksPerSpinSystem': 1,
         'PeakDescriptions': [{'dimensions': ['N', 'CA'], 'fraction': 1}]},
 '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}]},
 'NCO': {'Labels': ['N', 'CO-1'],
         'MinNumberPeaksPerSpinSystem': 1,
         'PeakDescriptions': [{'dimensions': ['N', 'CO-1'], 'fraction': 1}]},
 'NCOCX': {'Labels': ['N', 'CO-1', 'CX-1'],
           'MinNumberPeaksPerSpinSystem': 2,
           'PeakDescriptions': [{'dimensions': ['N', 'CO-1', 'CA-1'], 'fraction': 1},
                                {'dimensions': ['N', 'CO-1', 'CB-1'], 'fraction': 1},
                                {'dimensions': ['N', 'CO-1', 'CG-1'], 'fraction': 1},
                                {'dimensions': ['N', 'CO-1', 'CD-1'], 'fraction': 1},
                                {'dimensions': ['N', 'CO-1', 'CE-1'], 'fraction': 1},
                                {'dimensions': ['N', 'CO-1', 'CZ-1'], 'fraction': 1}]}}
```

- List specific spectrum descriptions:

```
In [53]: 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:

  1. Create additional json configuration with spectrum description and update *SPECTRUM_DESCRIPTIONS* `dict`. Content of *custom_spectrum_description.json*.

  2. Define dictionary with new spectrum description and update *SPECTRUM_DESCRIPTIONS* `dict`.

1. Create 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"]}
        ]
    }
}
```

```
In [54]: from nmrstarlib.converter import Converter
         from nmrstarlib.translator import StarFileToPeakList
         from nmrstarlib.noise import NoiseGenerator

         # update SPECTRUM_DESCRIPTIONS
         nmrstarlib.nmrstarlib.update_constants(spectrum_descriptions_cfg="path/to/custom_spectrum_de

         # create parameters dictionary for random normal distribution
         parameters = {"H_loc": [None, None], "C_loc": [0, 0], "N_loc": [0, 0],
                       "H_scale": [None, None], "C_scale": [0.01, 0.05], "N_scale": [0.01, 0.05]}

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

         converter = Converter(StarFileToPeakList(from_path="18569", to_path="out/18569_NCACX_custom
                                                  from_format="nmrstar", to_format="sparky",
                                                  spectrum_name="NCACX_custom",
                                                  plsplit=(70,30),
                                                  noise_generator=random_normal_noise_generator))
         converter.convert()
```

2. Define dictionary with new spectrum description and update *SPECTRUM_DESCRIPTIONS* `dict`.

```
In [55]: from nmrstarlib.converter import Converter
         from nmrstarlib.translator import StarFileToPeakList
         from nmrstarlib.noise import NoiseGenerator

         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"]}
                 ]
```

```
        }
    }

    # update SPECTRUM_DESCRIPTION
    nmrstarlib.nmrstarlib.SPECTRUM_DESCRIPTIONS.update(custom_experiment_type)

    # create parameters dictionary for random normal distribution
    parameters = {"H_loc": [0, 0], "C_loc": [None, None], "N_loc": [0, 0],
                  "H_scale": [0.001, 0.005], "C_scale": [None, None], "N_scale": [0.01, 0.05]}

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

    # Using valid BMRB id to access file from URL: from_path="18569"
    converter = Converter(StarFileToPeakList(from_path="18569", to_path="out/18569_NCACX_custom
                                             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.

- Visualize all available chemical shifts for all amino acids.

```
In [56]: from nmrstarlib.csviewer import CSViewer

    csviewer = CSViewer(from_path="18569", filename="out/18569_chem_shifts_all", csview_format="
    csviewer.csview(view=False)
```

nmrstarlib.csviewer output example:

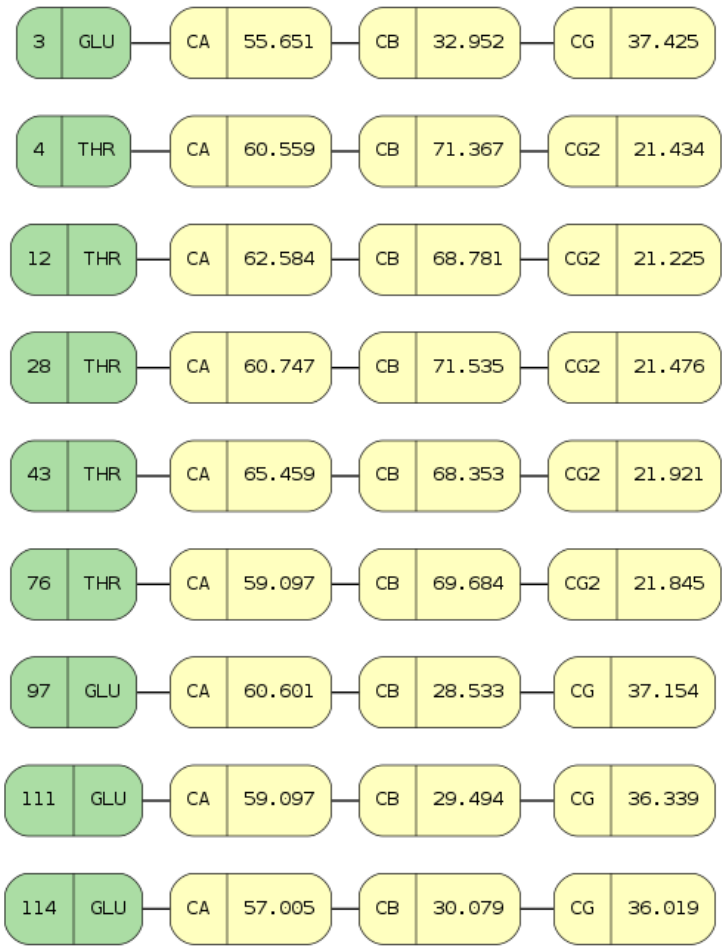
1	MET	H	8.55	HA	4.548	HB2	1.994	HB3	2.118	CA	55.489	CB	32.848	N	122.221		
2	SER	H	8.225	HA	4.420	HB2	3.805	HB3	3.857	CA	58.593	CB	64.057	N	117.197		
3	GLU	H	8.002	HA	4.848	HB2	1.852	HB3	1.963	HG2	1.981	HG3	2.191	CA	55.651	CB	32.952	CG	37.425	N	119.833
4	THR	H	8.956	HA	4.634	HB	4.098	HG21	1.196	HG22	1.196	HG23	1.196	CA	60.559	CB	71.367	CG2	21.434	N	117.212

- Visualize *CA*, *CB*, *CG*, and *CG2* chemical shifts for specific amino acids.

```
In [57]: from nmrstarlib.csviewer import CSViewer

    csviewer = CSViewer(from_path="18569", amino_acids=["GLU", "THR"], atoms=["CA", "CB", "CG",
                        filename="out/18569_chem_shifts_SER_THR_CA_CB_CG_CG2", csview_format="pr
    csviewer.csview(view=False)
```

nmrstarlib.csviewer output example:

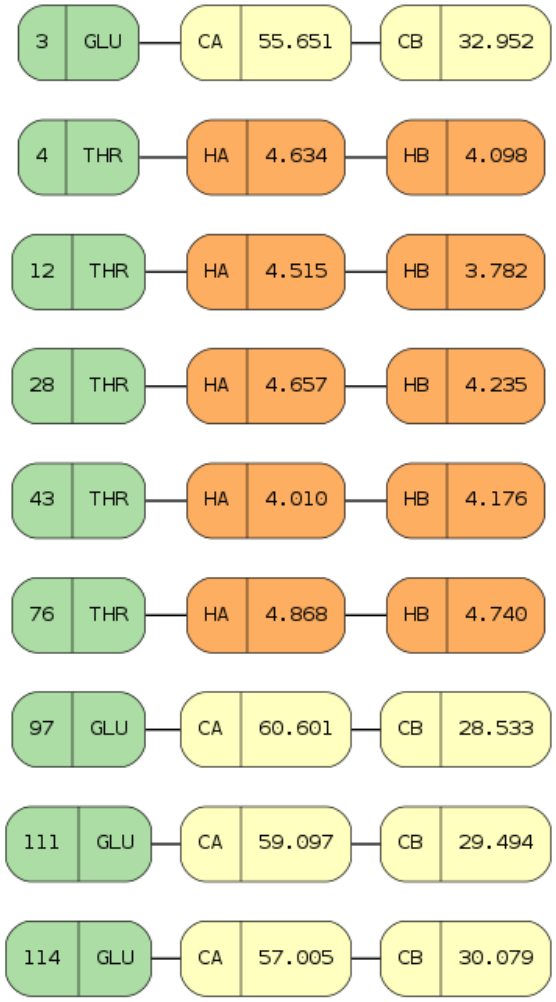


- Visualize specific atoms for specific amino acids.

```
In [58]: from nmrstarlib.csviewer import CSViewer

         csviewer = CSViewer(from_path="18569", amino_acids_and_atoms={"GLU": ["CA", "CB"], "THR": ["
                             filename="out/18569_chem_shifts_GLU_CA_CB_THR_HA_HB", csview_format="pn
         csviewer.csview(view=False)
```

nmrstarlib.csviewer output example:



2.2.2 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.

```
In [59]: ! python3 -m nmrstarlib --help
```

```
nmrstarlib command-line interface

Usage:
    nmrstarlib -h | --help
    nmrstarlib --version
    nmrstarlib convert (<from-path> <to-path>) [--from-format=<format>] [--to-format=<format>] [--bm
    nmrstarlib csview <starfile-path> [--aa=<aa>] [--at=<at>] [--aa-at=<aa-at>] [--csview-outfile=<pa
    nmrstarlib plsimulate (<from-path> <to-path> <spectrum>) [--from-format=<format>] [--to-format=<f

Options:
    -h, --help                      Show this screen.
    --version                       Show version.
```

```
    --verbose                       Print what files are processing.
    --show                          Display chemical shifts image generated by 'csview' command by de
    --from-format=<format>          Input file format, available formats: nmrstar, json [default: nm
    --to-format=<format>            Output file format, available formats: nmrstar, json [default: js
    --nmrstar-version=<version>     Version of NMR-STAR format to use, available: 2, 3 [default: 3].
    --bmrb-url=<url>                URL to BMRB interface [default: http://rest.bmrb.wisc.edu/bmrb/NM
    --pdb-url=<url>                 URL to PDB interface [default: https://files.rcsb.org/view/].
    --aa=<aa>                       Comma-separated amino acid three-letter codes (e.g. --aa=ALA,SER)
    --at=<at>                       Comma-separated BMRB atom codes (e.g. --at=CA,CB).
    --aa-at=<aa-at>                 Amino acid three-letter codes (keys) and corresponding atoms (val
    --csview-outfile=<path>         Where to save chemical shifts table.
    --csview-format=<format>        Format to which save chemical shift table [default: svg].
    --plsplit=<%>                   How to split peak list into chunks by percent [default: 100].
    --spectrum-descriptions=<path>  Path to custom spectrum descriptions file.
    --distribution=<func>           Statistical distribution function [default: normal].
    --seed=<value>                  Integer value used to initialize a pseudorandom number generator
    --H=<value>                     Statistical distribution parameter(s) for H dimension.
    --C=<value>                     Statistical distribution parameter(s) for C dimension.
    --N=<value>                     Statistical distribution parameter(s) for N dimension.
```

CLI Converting NMR-STAR files in bulk

CLI one-to-one file conversions

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

```
In [60]: ! python3 -m nmrstarlib convert bmr18569.str out/bmr18569.json \
                   --from-format=nmrstar --to-format=json
```

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

```
In [61]: ! python3 -m nmrstarlib convert bmr18569.json out/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:

```
In [62]: ! python3 -m nmrstarlib convert bmr18569.str.gz out/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:

```
In [63]: ! python3 -m nmrstarlib convert bmr18569.json.gz out/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:

```
In [64]: ! python3 -m nmrstarlib convert 18569 out/bmr18569.json.bz2 \
                   --from-format=nmrstar --to-format=json
```

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

CLI many-to-many files conversions

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

```
In [65]: ! python3 -m nmrstarlib convert starfiles_dir_nmrstar out/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:

```
In [66]: ! python3 -m nmrstarlib convert starfiles_dir_json out/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:

```
In [67]: ! python3 -m nmrstarlib convert starfiles_dir_nmrstar out/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:

```
In [68]: ! python3 -m nmrstarlib convert starfiles_json.tar.gz out/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:

```
In [69]: ! python3 -m nmrstarlib convert starfiles_nmrstar.zip out/starfiles_json.tar.bz2 \
                   --from-format=nmrstar --to-format=json
```

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

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

CLI one-to-one file simulations

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

```
In [70]: ! python3 -m nmrstarlib plsimulate bmr18569.str out/18569_HNcoCACB.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):

```
In [71]: ! python3 -m nmrstarlib plsimulate 18569 out/18569_HNcoCACB_ssv_HCN.txt HNcoCACB \
                   --from-format=nmrstar --to-format=sparky \
                   --H=0,0.001 --N=0,0.01 --C=0,0.01
```

- 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 chisquare distribution for degrees of freedom equal to 5:

```
In [72]: ! python3 -m nmrstarlib plsimulate 18569 out/18569_HNcoCACB_ssv_HCN_chi2.txt HNcoCACB \
                   --from-format=nmrstar --to-format=sparky \
                   --H=5 --N=5 --C=5 --distribution=chisquare
```

- 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):

```
In [73]: ! python3 -m nmrstarlib plsimulate bmr18569.str.gz out/18569_HNcoCACB_ssv_HN.txt HNcoCACB \
                   --from-format=nmrstar --to-format=sparky \
                   --H=0,0.001 --N=0,0.01
```

- 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. Note that values per split are separated by `:` and then parameters are separated by `,`.

```
In [74]: ! python3 -m nmrstarlib plsimulate 18569 out/18569_HNcoCACB_tsv_HN.txt HNcoCACB \
                   --from-format=nmrstar --to-format=sparky \
                   --plsplit=70,30 --H=0:0,0.001:0.005 --N=0:0,0.01:0.05
```

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

CLI 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:

```
In [75]: ! python3 -m nmrstarlib plsimulate starfiles_dir_nmrstar out/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:

```
In [76]: ! python3 -m nmrstarlib plsimulate starfiles_dir_nmrstar out/peaklists.zip HNcoCACB \
                   --from-format=nmrstar --to-format=sparky --H=0,0.001 --N=0,0.01
```

- 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:

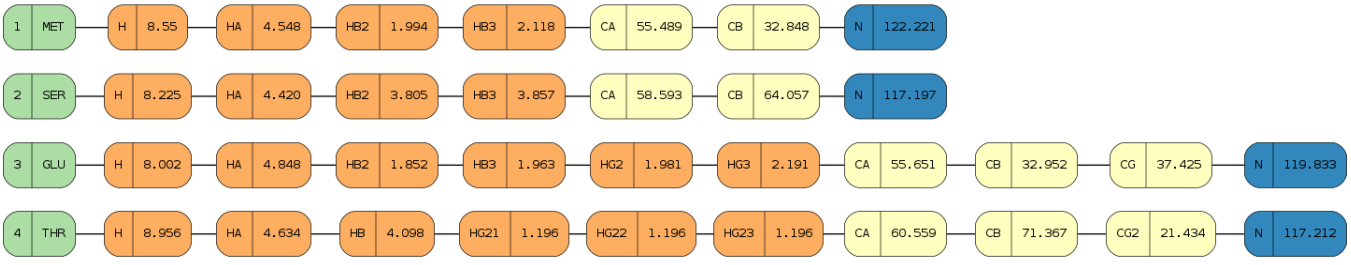
```
In [77]: ! python3 -m nmrstarlib plsimulate starfiles_dir_nmrstar out/peaklists.tar.gz NCACX \
                   --from-format=nmrstar --to-format=sparky --plsplit=70,30 \
                   --C=0:0,0.01:0.05 --N=0:0,0.01:0.07
```

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

CLI Visualizing chemical shift values

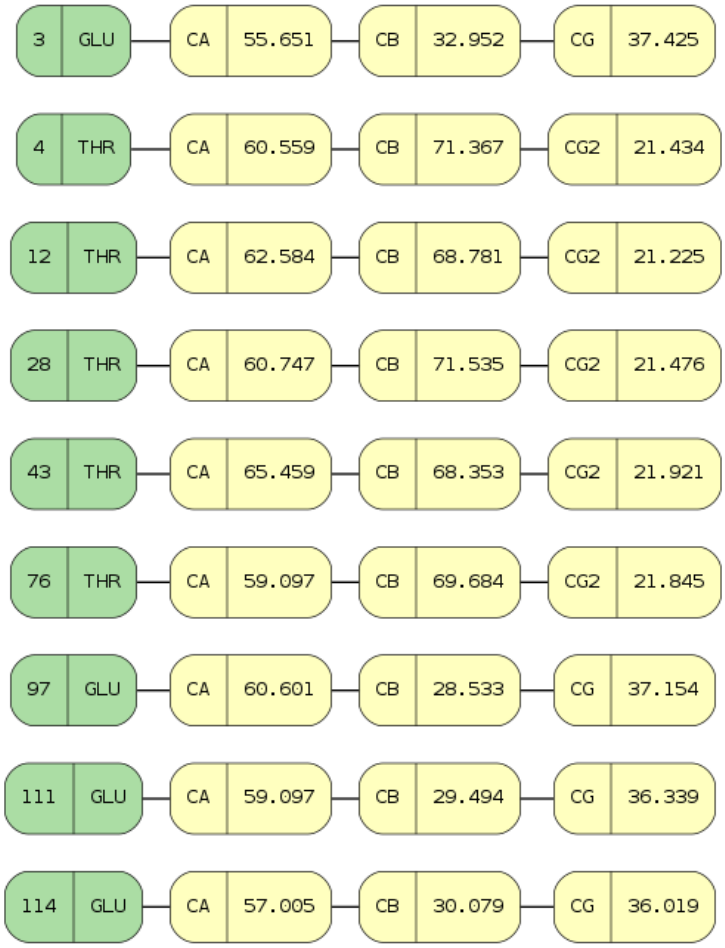
- Visualize chemical shift values for the entire sequence:

```
In [78]: ! python3 -m nmrstarlib csview 18569 --csview-outfile=out/18569_chem_shifts_all \
                   --csview-format=png
```



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

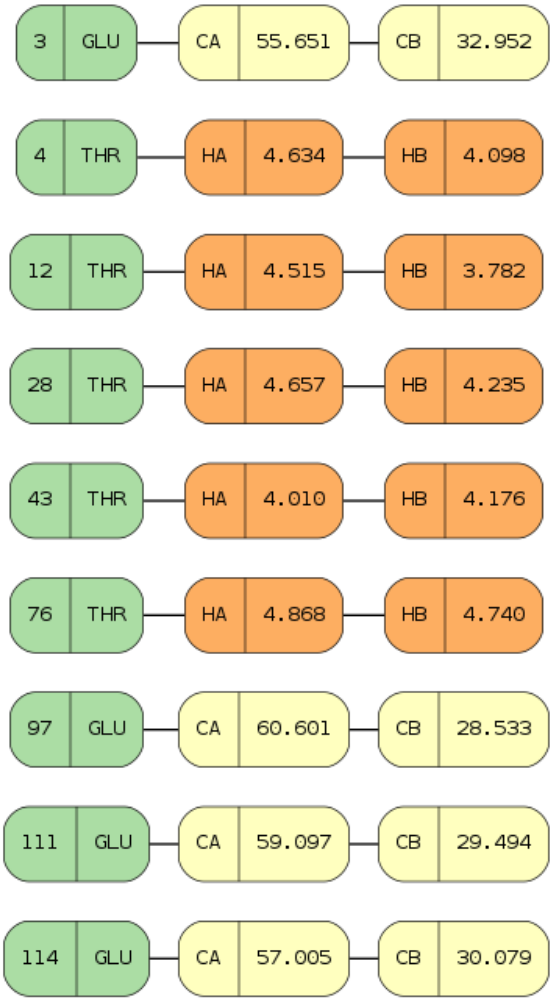
```
In [79]: ! python3 -m nmrstarlib csview 18569 --aa=GLU,THR --at=CA,CB,CG,CG2 \
                   --csview-outfile=out/18569_chem_shifts_GLU_THR_CA_CB_CG_CG2 \
                   --csview-format=png
```



- Visualize specific atoms for specific amino acids.

```
In [80]: ! python3 -m nmrstarlib csview 18569 --aa-at=GLU-CA,CB:THR-HA,HB \
                   --csview-outfile=out/18569_chem_shifts_GLU_CA_CB_THR_HA_HB \
                   --csview-format=png
```

nmrstarlib.csviewer output example:



2.3 The nmrstarlib API Reference

Routines for working with `BMRB NMR-STAR` and `PDB CIF` formatted files.

This package includes the following modules:

nmrstarlib This module provides the *StarFile* superclass and *NMRStarFile* and *CIFFile* which are python dictionary representation of a BMRB NMR-STAR file and PDB CIF file, respectively. Data can be accessed directly from the 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 and PDB CIF files, processing word, number, single quoted, double quoted, multiline quoted tokens.

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

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

noise This module provides the *NoiseGenerator* class for adding random normal noise values to peaks in simulated peak list for NMR-STAR formatted files.

plsimulator This module provides necessary interfaces in order to create a simulated *PeakList* from NMR-STAR formatted files.

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

fileio This module provides the *read_files()* generator to open files from different sources (single file/multiple files on a local machine, directory/archive of files, URL address of a file).

2.3.1 nmrstarlib.nmrstarlib

This module provides the *StarFile* superclass and *NMRStarFile* and *CIFFile* subclasses that store data from a single NMR-STAR file and CIF 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 hierarchical 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 unique names, the keys for them inside the "saveframe" dictionary are simply "loop_0", "loop_1", etc.

The CIF format is another STAR-derived format that is similar to NMR-STAR, i.e. it contains key-value pairs of data and contains "loops" but does not have "saveframe" data organization.

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

Update constant variables.

Returns None

Return type `None`

class nmrstarlib.nmrstarlib.**StarFile**(**args*, ***kwds*)

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

__init__(**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.

static read(*filehandle*, *source*)

Read data into a *StarFile* instance.

Parameters

- **filehandle** (`io.TextIOWrapper`, `gzip.GzipFile`, `bz2.BZ2File`, `zipfile.ZipFile`) – file-like object.
- **source** (*str*) – String indicating where file is coming from (path, url).

Returns subclass of *StarFile*.

Return type *NMRStarFile* or *CIFFile*

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*, *cif*, 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*, *cif*, or *json*.

Returns String representing the *StarFile* instance.

Return type str

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

Parameters

- **f** (*io.StringIO*) – writable file-like stream.
- **file_format** (*str*) – Format to use: *nmrstar*, *cif*, 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_star ()
Save *StarFile* into NMR-STAR or CIF formatted string.

Returns NMR-STAR string.

Return type str

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_cif (*string*)
Test if input string is in CIF format.

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

Returns Input string if in CIF 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

class nmrstarlib.nmrstarlib.**NMRStarFile**(*source=''*, *frame_categories=None*, **args*, ***kwds*)
NMRStarFile class that stores the data from a single NMR-STAR file in the form of an OrderedDict.

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

_build_file(*nmrstar_str*)
Build *NMRStarFile* object.

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

Returns instance of *NMRStarFile*.

Return type *NMRStarFile*

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

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

Returns Saveframe dictionary.

Return type collections.OrderedDict

_build_loop(*lexer*)
Build saveframe loop.

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

Returns Fields and values of the loop.

Return type tuple

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

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

Returns None

Return type None

print_file(*f=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *file_format='nmrstar'*, *tw=3*)
Print *NMRStarFile* 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

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

class nmrstarlib.nmrstarlib.**CIFFile**(*source=''*, **args*, ***kwds*)

CIFFile class that stores the data from a single CIF file in the form of an OrderedDict.

__init__(*source=''*, **args*, ***kwds*)

CIFFile initializer.

Parameters **source** (*str*) – Source *CIFFile* instance was created from - local file or URL address.

_build_file(*cif_str*)
Build *CIFFile* object.

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

Returns instance of *CIFFile*.

Return type *CIFFile*

_build_loop(*lexer*)
Build loop.

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

Returns Fields and values of the loop.

Return type tuple

print_file(*f=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>, file_format='cif', tw=0*)
Print *CIFFile* into a file or stdout.

Parameters

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

Returns None

Return type None

print_loop(*loop_number, f=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>, file_format='cif', tw=0*)
Print loop into a file or stdout.

Parameters

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

Returns None

Return type None

exception nmrstarlib.nmrstarlib.**InvalidToken**(*value*)

__init__(*value*)
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

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

exception nmrstarlib.nmrstarlib.**SkipSaveFrame**

__init__()
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

__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

2.3.2 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

2.3.3 nmrstarlib.converter

This module provides functionality for converting between the BMRB NMR-STAR/CIF format and its equivalent JSONized NMR-STAR/CIF 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:**
    - id - to - textfile
    - id - to - textfile.gz
    - id - to - textfile.bz2
    - id - 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 NMR-STAR/CIF files from NMR-STAR/CIF to JSON or from JSON to NMR-STAR/CIF format.

__init__(*file_generator*)

Converter initializer.

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

convert()

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

2.3.4 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*, *amino_acids_and_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*, *amino_acids_and_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.
- **amino_acids_and_atoms** (*dict*) – Amino acid and its atoms key-value pairs.
- **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)

2.3.5 nmrstarlib.noise

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

class nmrstarlib.noise.**NoiseGenerator**(*parameters=None*, *distribution_name='normal'*, *seed=None*)

Noise generator class.

__init__(*parameters=None*, *distribution_name='normal'*, *seed=None*)

Noise generator initializer.

Parameters

- **parameters** (*dict*) – Statistical distribution parameters per each peak list split.
- **distribution_name** (*str*) – Name of the statistical distribution function.

__weakref__

list of weak references to the object (if defined)

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

2.3.6 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 (-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, (-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, (-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*, *ndigits=6*)
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*) – *NMRStarFile* source.
- **chain_idx** (*int*) – *NMRStarFile* 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: *sparky*, *autoassign*, or *json*.

Returns None

Return type None

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

Parameters **fileformat** (*str*) – Format to use to write data: *sparky*, *autoassign*, 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

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

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

2.3.7 nmrstarlib.translator

This module provides the *Translator* abstract class and concrete classes: *StarFileToStarFile* for converting between NMR-STAR/CIF and JSONized NMR-STAR/CIF 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/CIF and JSONized NMR-STAR/CIF 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*, *cif*, or *json*.
- **to_format** (*str*) – Output format: *nmrstar*, *cif*, 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*

2.3.8 nmrstarlib.fileio

This module provides routines for reading NMR-STAR and CIF formatted files from difference kinds of sources:

- Single NMR-STAR or CIF formatted file on a local machine.
- Directory containing multiple NMR-STAR or CIF formatted files.
- Compressed zip/tar archive of NMR-STAR or CIF formatted files.
- URL address of NMR-STAR or CIF formatted file.
- BMRB ID of NMR-STAR or PDB ID of CIF formatted file.

nmrstarlib.fileio.**_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.fileio.**_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.fileio.**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.fileio.**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)

2.4 License

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

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