PyNMRSTAR

A Python module for reading, writing, and manipulating NMR-STAR files.

Python versions supported: 2.6, 2.7, 3.3, 3.4, 3.5, and 3.6

====

Command line tools

We have developed several command line tools to ease certain common actions
performed against NMR-STAR files. If you are looking for information on those
please go here.

====

PyNMRSTAR Overview

This module provides Entry, Saveframe, and Loop objects. Use python's built in help function for documentation.

There are eight module variables you can set to control our behavior.

- Setting pynmrstar.VERBOSE to True will print some of what is

going on to
the terminal.

- Setting pynmrstar.RAISE_PARSE_WARNINGS to True will raise an exception if
the parser encounters something problematic. Normally warnings are
suppressed.

- In addition, if you want to ignore some parse warnings but allow the
rest, you can specify warnings to ignore by adding the warning to ignore
to the “WARNINGS_TO_IGNORE” list.

Here are descriptions of the parse warnings that can be suppressed:

- “tag-only-loop”: A loop with no data was found.
- “empty-loop”: A loop with no tags or values was found.
- “tag-not-in-schema”: A tag was found in the entry that was not present
in the schema.
- “invalid-null-value”: A tag for which the schema disallows null values
had a null value.
- “bad-multiline”: A tag with an improper multi-line value was found.
Multiline values should look like this:

```
;
The multi-line
value here.
;
```

but the tag looked like this:

```
; The multi-line
value here.
;
```

- Setting pynmrstar.SKIP_EMPTY_LOOPS to True will suppress the printing of empty
  loops when calling **str** methods.

- Adding key->value pairs to pynmrstar.STR_CONVERSION_DICT will automatically
  convert tags whose value matches “key” to the string “value” when printing. This allows you to set the default conversion value for Booleans or other objects.

- Setting pynmrstar.ALLOW_V2_ENTRIES will allow parsing of NMR-STAR version
  2.1 entries. Most other methods will not operate correctly on parsed
  2.1 entries. This is only to allow you parse and access the data in

these entries - nothing else. Only set this if you have a really good reason to. Attempting to print a 2.1 entry will ‘work’ but tags that were after loops will be moved to before loops.

- Setting pynmrstar.DONT_SHOW_COMMENTS to True will supress the printing of
  comments before saveframes.

- Setting pynmrstar.CONVERT_DATATYPES to True will automatically convert
  the data loaded from the file into the corresponding python type as determined by loading the standard BMRB schema. This would mean that
  all floats will be represented as decimal.Decimal objects, all integers
  will be python int objects, strings and vars will remain strings, and dates will become datetime.date objects. When printing str() is called
  on all objects. Other that converting uppercase "E"s in scientific notation floats to lowercase "e"s this should not cause any change in
  the way re-printed NMR-STAR objects are displayed.

Some errors will be detected and exceptions raised, but this does not implement a full validator (at least at present).

Call directly (rather than importing) to run a self-test.

Variables

- `ALLOW_V2_ENTRIES`: False
- `CONVERT_DATATYPES`: False
- `DONT_SHOW_COMMENTS`: False
- `RAISE_PARSE_WARNINGS`: False
- `SKIP_EMPTY_LOOPS`: False
- `STR_CONVERSION_DICT`: {None: '.'}
- `VERBOSE`: False
- `WARNINGS_TO_IGNORE`: []

Functions

def `clean_value(value)`

Automatically quotes the value in the appropriate way. Don't quote values you send to this method or they will show up in another set of quotes as part of the actual data. E.g.:

clean_value('"e. coli"') returns '"e. coli"'

while

clean_value("e. coli") returns "'e. coli'"

This will automatically be called on all values when you use a str() method (so don't call it before inserting values into tags or loops).

Be mindful of the value of STR_CONVERSION_DICT as it will effect the way the value is converted to a string.

def `delete_empty_saveframes(entry_object, tags_to_ignore=['sf_category', 'sf_framecode'], allowed_null_values=['.', '?', None])`

This method will delete all empty saveframes in an entry (the loops in the saveframe must also be empty for the saveframe to be deleted). “Empty” means no values in tags, not no tags present.

def `diff(entry1, entry2)`

Prints the differences between two entries. Non-equal entries will always be detected, but specific differences detected depends on order of entries.

def `enable_nef_defaults()`

Sets the module variables such that our behavior matches the NEF standard. Specifically, suppress printing empty loops by default and convert True -> “true” and False -> “false” when printing.

def `enable_nmrstar_defaults()`

Sets the module variables such that our behavior matches the
BMRB standard (NMR-STAR). This is the default behavior of this module.
This method only exists to revert after calling enable_nef_defaults().

def `validate(entry_to_validate, schema=None)`

Prints a validation report of an entry.

Classes

class `Entry()`

An OO representation of a BMRB entry. You can initialize this
object several ways; (e.g. from a file, from the official database,
from scratch) see the class methods below.

Methods:

def `__init__()`

You should not directly instantiate an Entry using this method.
Instead use the class methods:"
Entry.from_database()
Entry.from_file()

Entry.from_string()

Entry.from_scratch()

Entry.from_json()

def `add_saveframe(frame)`

Add a saveframe to the entry.

def `compare(other)`

Returns the differences between two entries as a list. Otherwise returns 1 if different and 0 if equal. Non-equal entries will always be detected, but specific differences detected depends on order of entries.

def `frame_dict()`

Returns a dictionary of saveframe name -> saveframe object

def `from_database(cls, entry_num)`

Create an entry corresponding to the most up to date entry on the public BMRB server. (Requires ability to initiate outbound HTTP connections.)

def `from_file(cls, the_file)`

Create an entry by loading in a file. If the_file starts with http://, https://, or ftp:// then we will use those protocols to attempt to open the file.

def `from_json(cls, json_dict)`

Create an entry from JSON (serialized or unserialized JSON).

def `from_scratch(cls, entry_id)`

Create an empty entry that you can programatically add to.

You must pass a value corresponding to the Entry ID.

(The unique identifier “xxx” from “data_xxx”.)

def `from_string(cls, the_string)`

Create an entry by parsing a string.

def `get_json(serialize=True)`

Returns the entry in JSON format. If serialize is set to

False a dictionary representation of the entry that is

serializeable is returned.

def `get_loops_by_category(value)`

Allows fetching loops by category.

def `get_saveframe_by_name(frame)`

Allows fetching a saveframe by name.

def `get_saveframes_by_category(value)`

Allows fetching saveframes by category.

def `get_saveframes_by_tag_and_value(tag_name, value)`

Allows fetching saveframe(s) by tag and tag value.

def `get_tag(tag, whole_tag=False)`

Given a tag (E.g. _Assigned_chem_shift_list.Data_file_name) return a list of all values for that tag. Specify whole_tag=True and the [tag_name, tag_value (,tag_linenumber)] pair will be returned.

def `get_tags(tags)`

Given a list of tags, get all of the tags and return the results in a dictionary.

def `nef_string()`

Returns a string representation of the entry in NEF.

def `normalize(schema=None)`

Sorts saveframes, loops, and tags according to the schema provided (or BMRB default if none provided) and according to the assigned ID.

def `print_tree()`

Prints a summary, tree style, of the frames and loops in the entry.

def rename_saveframe(original_name, new_name)

Renames a saveframe and updates all pointers to that saveframe in the entry with the new name.

def validate(validate_schema=True, schema=None, validate_star=True)

Validate an entry in a variety of ways. Returns a list of errors found. 0-length list indicates no errors found. By default all validation modes are enabled.

validate_schema - Determines if the entry is validated against the NMR-STAR schema. You can pass your own custom schema if desired, otherwise the schema will be fetched from the BMRB servers.

validate_star - Determines if the STAR syntax checks are ran.

def write_to_file(file_name, format_="nmrstar")

Writes the entry to the specified file in NMR-STAR format.

Optionally specify format_=json to write to the file in JSON format.

class `Saveframe()`

A saveframe object. Create using the class methods, see below.

Methods:

def `__init__()`

Don’t use this directly. Use the class methods to construct:

Saveframe.from_scratch()

Saveframe.from_string()

Saveframe.from_template()

Saveframe.from_file()

and Saveframe.from_json()

def `add_loop(loop_to_add)`

Add a loop to the saveframe loops.

def `add_tag(name, value, linenum=None, update=False)`

Add a tag to the tag list. Does a bit of validation and parsing. Set update to true to update a tag if it exists rather than raise an exception.

def `add_tags(tag_list, update=False)`

Adds multiple tags to the list. Input should be a list of

tuples that are either [key, value] or [key]. In the latter case the value will be set to “.”. Set update to true to update a tag if it exists rather than raise an exception.

def `compare(other)`

Returns the differences between two saveframes as a list. Non-equal saveframes will always be detected, but specific differences detected depends on order of saveframes.

def `delete_tag(tag)`

Deletes a tag from the saveframe based on tag name.

def `from_file(cls, the_file, csv=False)`

Create a saveframe by loading in a file. Specify csv=True is the file is a CSV file. If the_file starts with http://, https://, or ftp:// then we will use those protocols to attempt to open the file.

def `from_json(cls, json_dict)`

Create a saveframe from JSON (serialized or unserialized JSON).

def `from_scratch(cls, sf_name, tag_prefix=None, source=from_scratch())`

Create an empty saveframe that you can programatically add to. You may also pass the tag prefix as the second argument. If

you do not pass the tag prefix it will be set the first time you add a tag.

def `from_string(cls, the_string, csv=False)`

Create a saveframe by parsing a string. Specify csv=True is the string is in CSV format and not NMR-STAR format.

def `from_template(cls, category, name=None, all_tags=False, schema=None)`

Create a saveframe that has all of the tags and loops from the schema present. No values will be assigned. Specify the category when calling this method. Optionally also provide the name of the saveframe as the 'name' argument.

The optional argument 'all_tags' forces all tags to be included rather than just the mandatory tags.

def `get_data_as_csv(header=True, show_category=True)`

Return the data contained in the loops, properly CSVd, as a string. Set header to False omit the header. Set show_category to False to omit the loop category from the headers.

def `get_json(serialize=True)`

Returns the saveframe in JSON format. If serialize is set to

False a dictionary representation of the saveframe that is serializeable is returned.

def `get_loop_by_category(name)`

Return a loop based on the loop name (category).

def `get_tag(query, whole_tag=False)`

Allows fetching the value of a tag by tag name. Specify whole_tag=True and the [tag_name, tag_value] pair will be returned.

def `loop_dict()`

Returns a hash of loop category -> loop.

def `loop_iterator()`

Returns an iterator for saveframe loops.

def `print_tree()`

Prints a summary, tree style, of the loops in the saveframe.

def `set_tag_prefix(tag_prefix)`

Set the tag prefix for this saveframe.

def `sort_tags(schema=None)`

Sort the tags so they are in the same order as a BMRB schema. Will automatically use the standard schema if none is provided.

def `tag_iterator()`

Returns an iterator for saveframe tags.

def `validate(validate_schema=True, schema=None, validate_star=True)`

Validate a saveframe in a variety of ways. Returns a list of errors found. 0-length list indicates no errors found. By default all validation modes are enabled.

validate_schema - Determines if the entry is validated against the NMR-STAR schema. You can pass your own custom schema if desired, otherwise the schema will be fetched from the BMRB servers.

validate_star - Determines if the STAR syntax checks are ran.

def `write_to_file(file_name, format_="nmrstar")`

Writes the saveframe to the specified file in NMR-STAR format.

Optionally specify format_=json to write to the file in JSON format.

class `Schema()`

A BMRB schema. Used to validate STAR files.

Methods:

def `__init__(schema_file=None)`

Initialize a BMRB schema. With no arguments the most
up-to-date schema will be fetched from the BMRB FTP site.
Otherwise pass a URL or a file to load a schema from using the
schema_file keyword argument.

def `add_tag(tag, tag_type, null_allowed, sf_category, loop_flag, after=None)`

Adds the specified tag to the tag dictionary. You must provide:

1. The full tag as such:
   "_Entry_interview.Sf_category"
2. The tag type which is one of the following:
   "INTEGER"
   "FLOAT"
   "CHAR(len)"
   "VARCHAR(len)"
   "TEXT"
   "DATETIME year to day"
3. A python True/False that indicates whether null values are allowed.

4. The sf_category of the parent saveframe.
5. A True/False value which indicates if this tag is a loop tag.
6. Optional: The tag to order this tag behind when normalizing saveframes.

def `convert_tag(tag, value, linenum=None)`

Converts the provided tag from string to the appropriate type as specified in this schema.

def `string_representation(search=None)`

Prints all the tags in the schema if search is not specified and prints the tags that contain the search string if it is.

def `val_type(tag, value, category=None, linenum=None)`

Validates that a tag matches the type it should have according to this schema.

class `Loop()`

A BMRB loop object. Create using the class methods, see below.

Methods:

def `__init__()`

You should not directly instantiate a Loop using this method.

Instead use the class methods:

Loop.from_scratch()

Loop.from_string()

Loop.from_template()

Loop.from_file()

Loop.from_json()

def `add_column(name, ignore_duplicates=False)`

Add a column to the column list. Does a bit of validation and parsing. Set ignore_duplicates to true to ignore attempts to add the same tag more than once rather than raise an exception.

You can also pass a list of column names to add more than one column at a time.

Note that adding a column only adds a new tag to the list of tags present in this loop. It does not automatically add a column of None values to the data array if the loop is already populated with data.

def `add_data(the_list, rearrange=False)`

Add a list to the data field. Items in list can be any type, they will be converted to string and formatted correctly. The

list must have the same cardinality as the column names or you must set the rearrange variable to true and have already set all the columns in the loop. Rearrange will break a longer list into rows based on the number of columns.

def `add_data_by_column(column_id, value)`

Add data to the loop one element at a time, based on column. Useful when adding data from SANS parsers.

def `clear_data()`

Erases all data in this loop. Does not erase the data columns or loop category.

def `compare(other)`

Returns the differences between two loops as a list. Order of loops being compared does not make a difference on the specific errors detected.

def `delete_data_by_tag_value(tag, value, index_tag=None)`

Deletes all rows which contain the provided value in the provided column. If index_tag is provided, that column is renumbered starting with 1. Returns the deleted rows.

def `filter(tag_list, ignore_missing_tags=False)`

Returns a new loop containing only the specified tags.
Specify ignore_missing_tags=True to bypass missing tags rather
than raising an error.

def `from_file(cls, the_file, csv=False)`

Create a saveframe by loading in a file. Specify csv=True if
the file is a CSV file. If the_file starts with http://,
https://, or ftp:// then we will use those protocols to attempt
to open the file.

def `from_json(cls, json_dict)`

Create a loop from JSON (serialized or unserialized JSON).

def `from_scratch(cls, category=None, source=from_scratch())`

Create an empty saveframe that you can programatically add
to. You may also pass the tag prefix as the second argument. If
you do not pass the tag prefix it will be set the first time you
add a tag.

def `from_string(cls, the_string, csv=False)`

Create a saveframe by parsing a string. Specify csv=True is
the string is in CSV format and not NMR-STAR format.

def `from_template(cls, tag_prefix, all_tags=False, schema=None)`

Create a loop that has all of the tags from the schema present. No values will be assigned. Specify the tag prefix of the loop.

The optional argument all_tags forces all tags to be included rather than just the mandatory tags.

def `get_columns()`

Return the columns for this entry with the category included. Throws ValueError if the category was never set.

def `get_data_as_csv(header=True, show_category=True)`

Return the data contained in the loops, properly CSVd, as a string. Set header to False to omit the header. Set show_category to false to omit the loop category from the headers.

def `get_data_by_tag(tags=None)`

Identical to get_tag but wraps the results in a list even if only fetching one tag. Primarily exists for legacy code.

def `get_json(serialize=True)`

Returns the loop in JSON format. If serialize is set to

False a dictionary representation of the loop that is serializeable is returned.

def `get_tag(tags=None, whole_tag=False)`

Provided a tag name (or a list of tag names), or ordinals corresponding to columns, return the selected tags by row as a list of lists.

If whole_tag=True return the full tag name along with the tag value, or if dict_result=True, as the tag key.

If dict_result=True, return the tags as a list of dictionaries in which the tag value points to the tag."""

def `print_tree()`

Prints a summary, tree style, of the loop.

def `renumber_rows(index_tag, start_value=1, maintain_ordering=False)`

Renumber a given column incrementally. Set start_value to initial value if 1 is not acceptable. Set maintain_ordering to preserve sequence with offset.

E.g. 2,3,3,5 would become 1,2,2,4.

def `set_category(category)`

Set the category of the loop. Useful if you didn't know the category at loop creation time.

def `sort_rows(tags, key=None)`

Sort the data in the rows by their values for a given column or columns. Specify the columns using their names or ordinals. Accepts a list or an int/float. By default we will sort numerically. If that fails we do a string sort. Supply a function as key and we will order the elements based on the keys it provides. See the help for sorted() for more details. If you provide multiple columns to sort by, they are interpreted as increasing order of sort priority.

def `sort_tags(schema=None)`

Rearranges the columns and data in the loop to match the order from the schema. Uses the BMRB schema unless one is provided.

def `validate(validate_schema=True, schema=None, validate_star=True, category=None)`

Validate a loop in a variety of ways. Returns a list of errors found. 0-length list indicates no errors found. By default all validation modes are enabled.

validate_schema - Determines if the entry is validated against the NMR-STAR schema. You can pass your own custom schema if desired,

otherwise the schema will be fetched from the BMRB servers.

validate_star - Determines if the STAR syntax checks are ran.