Trollsift Documentation

Release 0.1.0

Hrobjartur Thorsteinsson

Nov 21, 2022

Contents

Trollsift is a collection of modules that assist with formatting, parsing and filtering satellite granule file names. These modules are useful and necessary for writing higher level applications and api's for satellite batch processing.

The source code of the package can be found at github, github

CHAPTER 1

Contents

1.1 Installation

Trollsift is available from PyPI:

```
$ pip install trollsift
```

Alternatively, you can install it into a conda environment by using the conda-forge channel:

```
$ conda install -c conda-forge trollsift
```

Or you can install it directly from the GitHub repository:

```
$ pip install git+https://github.com/pytroll/trollsift.git
```

1.1.1 Developer Installation

You can download the trollsift source code from github:

```
$ git clone https://github.com/pytroll/trollsift.git
```

and then run:

```
$ pip install -e .
```

1.1.2 Testing

To check if your python setup is compatible with trollsift, you can run the test suite using pytest:

```
$ pytest trollsift/tests
```

1.2 Usage

Trollsift include collection of modules that assist with formatting, parsing and filtering satellite granule file names. These modules are useful and necessary for writing higher level applications and api's for satellite batch processing. Currently we are implementing the string parsing and composing functionality. Watch this space for further modules to do with various types of filtering of satellite data granules.

1.2.1 Parser

The trollsift string parser module is useful for composing (formatting) and parsing strings compatible with the Python Format String Syntax. In satellite data file name filtering, the library is useful for extracting typical information from granule filenames, such as observation time, platform and instrument names. The trollsift Parser can also verify that the string formatting is invertible, i.e. specific enough to ensure that parsing and composing of strings are bijective mappings (aka one-to-one correspondence) which may be essential for some applications, such as predicting granule

parsing

The Parser object holds a format string, allowing us to parse and compose strings:

```
>>> from trollsift import Parser
>>>
>>> p = Parser("/somedir/{directory}/hrpt_{platform:4s}{platnum:2s}_{time:%Y%m%d_%H%M}
↪_{orbit:05d}.l1b")
>>> data = p.parse("/somedir/otherdir/hrpt_noaa16_20140210_1004_69022.l1b")
>>> print(data) # doctest: +NORMALIZE_WHITESPACE
{'directory': 'otherdir', 'platform': 'noaa', 'platnum': '16',
 'time': datetime.datetime(2014, 2, 10, 10, 4), 'orbit': 69022}
```

Parsing in trollsift is not "greedy". This means that in the case of ambiguous patterns it will match the shortest portion of the string possible. For example:

```
>>> from trollsift import Parser
>>>
>>> p = Parser("{field_one}_{field_two}")
>>> data = p.parse("abc_def_ghi")
>>> print(data)
{'field_one': 'abc', 'field_two': 'def_ghi'}
```

So even though the first field could have matched to "abc_def", the non-greedy parsing chose the shorter possible match of "abc".

composing

The reverse operation is called 'compose', and is equivalent to the Python string class format method. Here we take the filename pattern from earlier, change the time stamp of the data, and write out a new file name,

```
>>> from datetime import datetime
>>>
>>> p = Parser("/somedir/{directory}/hrpt_{platform:4s}{platnum:2s}_{time:%Y%m%d_%H%M}
↪_{orbit:05d}.l1b")
>>> data = {'directory': 'otherdir', 'platform': 'noaa', 'platnum': '16', 'time':␣
↪datetime(2012, 1, 1, 1, 1), 'orbit': 69022}
```

(continues on next page)

(continued from previous page)

```
>>> p.compose(data)
'/somedir/otherdir/hrpt_noaa16_20120101_0101_69022.l1b'
```

It is also possible to compose only partially, i.e., compose by specifying values for only a subset of the parameters in the format string. Example:

```
>>> p = Parser("/somedir/{directory}/hrpt_{platform:4s}{platnum:2s}_{time:%Y%m%d_%H%M}
↪_{orbit:05d}.l1b")
>>> data = {'directory':'my_dir'}
>>> p.compose(data, allow_partial=True)
'/somedir/my_dir/hrpt_{platform:4s}{platnum:2s}_{time:%Y%m%d_%H%M}_{orbit:05d}.l1b'
```

In addition to python's builtin string formatting functionality trollsift also provides extra conversion options such as making all characters lowercase:

```
>>> my_parser = Parser("{platform_name!l}")
>>> my_parser.compose({'platform_name': 'NPP'})
'npp'
```

For all of the options see *StringFormatter*.

1.2.2 standalone parse and compose

The parse and compose methods also exist as standalone functions, depending on your requirements you can call,

```
>>> from trollsift import parse, compose
>>> fmt = "/somedir/{directory}/hrpt_{platform:4s}{platnum:2s}_{time:%Y%m%d_%H%M}_
↪{orbit:05d}.l1b"
>>> data = parse( fmt, "/somedir/otherdir/hrpt_noaa16_20140210_1004_69022.l1b" )
>>> data['time'] = datetime(2012, 1, 1, 1, 1)
>>> compose(fmt, data)
'/somedir/otherdir/hrpt_noaa16_20120101_0101_69022.l1b'
```

And achieve the exact same result as in the Parse object example above.

1.3 The `trollsift` API

1.3.1 trollsift parser

Main parsing and formatting functionality.

class trollsift.parser.**GlobifyFormatter**

UNPROVIDED_VALUE = '<trollsift unprovided value>'

format_field(*value*, *format_spec*)

get_value(*key*, *args*, *kwargs*)

class trollsift.parser.**Parser**(*fmt*)

Class-based interface to parsing and formatting functionality.

compose(*keyvals*, *allow_partial=False*)

Compose format string *self.fmt* with parameters given in the *keyvals* dict.

Parameters

- **keyvals** (*dict*) – "Parameter –> parameter value" map
- **allow_partial** (*bool*) – If True, then partial composition is allowed, i.e., not all parameters present in *fmt* need to be specified in *keyvals*. Unspecified parameters will, in this case, be left unchanged. (Default value = False).

Returns

Result of formatting the *self.fmt* string with parameter values extracted from the corresponding items in the *keyvals* dictionary.

Return type str

format(*keyvals*, *allow_partial=False*)
Compose format string *self.fmt* with parameters given in the *keyvals* dict.

Parameters

- **keyvals** (*dict*) – "Parameter –> parameter value" map
- **allow_partial** (*bool*) – If True, then partial composition is allowed, i.e., not all parameters present in *fmt* need to be specified in *keyvals*. Unspecified parameters will, in this case, be left unchanged. (Default value = False).

Returns

Result of formatting the *self.fmt* string with parameter values extracted from the corresponding items in the *keyvals* dictionary.

Return type str

globify(*keyvals=None*)
Generate a string useable with glob.glob() from format string *fmt* and *keyvals* dictionary.

is_one2one()
Runs a check to evaluate if this format string has a one to one correspondence. I.e. that successive composing and parsing opperations will result in the original data. In other words, that input data maps to a string, which then maps back to the original data without any change or loss in information.

Note: This test only applies to sensible usage of the format string. If string or numeric data is causes overflow, e.g. if composing "abcd" into {3s}, one to one correspondence will always be broken in such cases. This off course also applies to precision losses when using datetime data.

keys()
Get parameter names defined in the format string.

parse(*stri*, *full_match=True*)
Parse keys and corresponding values from *stri* using format described in *fmt* string.

validate(*stri*)
Validates that string *stri* is parsable and therefore complies with this string format definition. Useful for filtering strings, or to check if a string if compatible before passing it to the parser function.

class trollsift.parser.**RegexFormatter**
String formatter that converts a format string to a regular expression.

```
>>> regex_formatter = RegexFormatter()
>>> regex_str = regex_formatter.format('{field_one:5d}_{field_two}')
```

Can also be used to extract values from a string given the format spec for that string:

```
>>> regex_formatter.extract_values('{field_one:5d}_{field_two}', '12345_sometext')
{'field_one': '12345', 'field_two': 'sometext'}
```

Note that the regular expressions generated by this class are specially generated to reduce "greediness" of the matches found. For ambiguous patterns where a single field could match shorter or longer portions of the provided string, this class will prefer the shorter version of the string in order to make the rest of the pattern match. For example:

```
>>> regex_formatter.extract_values('{field_one}_{field_two}', 'abc_def_ghi')
{'field_one': 'abc', 'field_two': 'def_ghi'}
```

Note how *field_one* could have matched "abc_def", but the lower greediness of this parser caused it to only match against "abc".

ESCAPE_CHARACTERS = ['\\', '!', '"', '#', '$', '&', "'", '(', ')', '*', '+', ',', '-',

ESCAPE_SETS = [('\\', '\\\\'), ('!', '\\!'), ('"', '\\"'), ('#', '\\#'), ('$', '\\$'),

UNPROVIDED_VALUE = '<trollsift unprovided value>'

format

format_field(*value*, *format_spec*)

static format_spec_to_regex(*field_name*, *format_spec*)
Make an attempt at converting a format spec to a regular expression.

get_value(*key*, *args*, *kwargs*)

parse(*format_string*)

regex_field(*field_name*, *value*, *format_spec*)

class trollsift.parser.**StringFormatter**
Custom string formatter class for basic strings.

This formatter adds a few special conversions for assisting with common trollsift situations like making a parameter lowercase or removing hyphens. The added conversions are listed below and can be used in a format string by prefixing them with an *!* like so:

```
>>> fstr = "{!u}_{!l}"
>>> formatter = StringFormatter()
>>> formatter.format(fstr, "to_upper", "To_LowerCase")
"TO_UPPER_to_lowercase"
```

- c: Make capitalized version of string (first character upper case, all lowercase after that) by executing the parameter's *.capitalize()* method.
- l: Make all characters lowercase by executing the parameter's *.lower()* method.
- R: Remove all separators from the parameter including '-', '_', ' ', and ':'.
- t: Title case the string by executing the parameter's *.title()* method.
- u: Make all characters uppercase by executing the parameter's *.upper()* method.
- h: A combination of 'R' and 'l'.
- H: A combination of 'R' and 'u'.

CONV_FUNCS = {'H': 'upper', 'c': 'capitalize', 'h': 'lower', 'l': 'lower', 't': 't

convert_field(*value*, *conversion*)
Apply conversions mentioned above.

trollsift.parser.**compose**(*fmt*, *keyvals*, *allow_partial=False*)
Compose format string *self.fmt* with parameters given in the *keyvals* dict.

Parameters

- **fmt** (*str*) – Python format string to match against
- **keyvals** (*dict*) – "Parameter –> parameter value" map
- **allow_partial** (*bool*) – If True, then partial composition is allowed, i.e., not all parameters present in *fmt* need to be specified in *keyvals*. Unspecified parameters will, in this case, be left unchanged. (Default value = False).

Returns

Result of formatting the *self.fmt* string with parameter values extracted from the corresponding items in the *keyvals* dictionary.

Return type str

trollsift.parser.**extract_values**(*fmt*, *stri*, *full_match=True*)
Extract information from string matching format.

Parameters

- **fmt** (*str*) – Python format string to match against
- **stri** (*str*) – String to extract information from
- **full_match** (*bool*) – Force the match of the whole string. Default to `True`.

trollsift.parser.**get_convert_dict**
Retrieve parse definition from the format string *fmt*.

trollsift.parser.**globify**(*fmt*, *keyvals=None*)
Generate a string usable with glob.glob() from format string *fmt* and *keyvals* dictionary.

trollsift.parser.**is_one2one**(*fmt*)
Runs a check to evaluate if the format string has a one to one correspondence. I.e. that successive composing and parsing opperations will result in the original data. In other words, that input data maps to a string, which then maps back to the original data without any change or loss in information.

Note: This test only applies to sensible usage of the format string. If string or numeric data is causes overflow, e.g. if composing "abcd" into {3s}, one to one correspondence will always be broken in such cases. This of course also applies to precision losses when using datetime data.

trollsift.parser.**parse**(*fmt*, *stri*, *full_match=True*)
Parse keys and corresponding values from *stri* using format described in *fmt* string.

Parameters

- **fmt** (*str*) – Python format string to match against
- **stri** (*str*) – String to extract information from
- **full_match** (*bool*) – Force the match of the whole string. Default True.

trollsift.parser.**purge**()
Clear internal caches.

Not needed normally, but can be used to force cache clear when memory is very limited.

trollsift.parser.**regex_format**

trollsift.parser.**validate**(*fmt*, *stri*)

Validates that string *stri* is parsable and therefore complies with the format string, *fmt*. Useful for filtering string, or to check if string if compatible before passing the string to the parser function.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index