bufrpy Documentation

Release 0.2.1

Tuure Laurinolli / FMI

February 05, 2015

Contents

Bufrpy is a pure-Python BUFR message decoder. BUFR messages are typically used to transmit meteorological observations. Bufrpy was developed to work with NWCSAF RDT and HRW data, but there should not be any reason why it could not be used with any BUFR messages, with the following limitations:

- Messages with multiple data subsets are not supported
- Compressed messages are not supported
- Operator descriptors are not supported
- Sequence descriptors are not supported

Adding support for any of the above should be feasible, but we have not encountered such data in our use of the library.

CHAPTER 1

Examples

Usage examples

1.1 Examples

1.1.1 RDT Contour Extraction

```
import bufrpy
from bufrpy.template import safnwc
import re
import itertools
import sys

def to_rings(bufrmsg):
    """ Return a list of cloud contours as closed rings, in lat/lon ordinate order"""
    out = []
    subsets = bufrmsg.section4.subsets
    for subset in subsets:
        for cloudsystem in subset.values[19]: # cloud systems are at index 19
            contour = cloudsystem[0] # contour is at index 0 of cloud system
            point_vals = list(itertools.chain(contour, itertools.islice(contour,1))) # close contour
            points = []
            for (lat, lon) in point_vals:
                points.append((lat.value, lon.value)) # extract values
            out.append(points)
    return out

def to_wkt(rings):
    out = []
    for ring in rings:
        # Change ordinate order, convert to strings, join and stick in a POLYGON
        points = [" ".join(map(str, reversed(latlon))) for latlon in ring]
        edge = ",".join(points)
        out.append("POLYGON((" + edge + "))")
    return "\n".join(out)

if __name__ == '__main__':

    template = safnwc.read_template(open(sys.argv[1], 'rb'))
    msg = bufrpy.decode_file(open(sys.argv[2], 'rb'), template)
```

```
rings = to_rings(msg)
wkt = to_wkt(rings)

print wkt
```

CHAPTER 2

Data Model

Data model

2.1 Data Model

Bufrpy data model matches that of the BUFR messages fairly closely, though not as closely as libbufr or the Python libbufr wrappers. Typically BUFR messages are stored in a file so that the file contains a single BUFR message. Some applications (e.g. SAFNWC HWR) store multiple messages in a single file, requiring some tricks to read them.

The top-level data structure in `bufrpy` is `Message` that corresponds to one BUFR message. A BUFR message contains 6 sections, some of which contain metadata and some the actual data. The most important sections are numbers 3 and 4. They contain data descriptors and the data itself respectively. Logically BUFR files contain at least one *subset* of data. Each subset follows the same structure, defined by the data descriptors in section 3. In `bufrpy`, sections are attributes of `Message`. Section 4 contains the data subsets as a list of `BufrSubset` objects.

Logically each subset is a linear sequence of data values. All metadata about the data is stored in the descriptors. However, there are certain looping constructs (replication descriptors) that allow repetition of a sequence of data items. This leads to hierarchical logical structure. Physically the data is still stored in a linear sequence (or in interleaved fashion if a message has multiple data subsets). In `bufrpy` the data of each dataset is represented by a list of `BufrValue` objects. A replicated sequence in the data is represented by a list of lists of `BufrValue` objects so that the outer list contains one list for each repetition, and the inner lists are equivalent to `values` of a `BufrSubset`. Hierarchical data is thus naturally represented as nested lists.

CHAPTER 3

API Documentation

API documentation covers the public API

3.1 API

3.1.1 Reading BUFR messages

bufrpy.**decode_file**(*f*, *b_table*)

Decode BUFR message from a file into a Message object.

Parameters

- **f** (*file*) – File that contains the bufr message
- **b_table** (*Mapping|Template*) – Either a mapping from BUFR descriptor codes to descriptors or a Template describing the message

bufrpy.**decode**(*stream*, *b_table*, *skip_data=False*)

Decode BUFR message from stream into a Message object.

See WMO306_vl2_BUFR3_Spec_en.pdf for BUFR format specification.

Parameters

- **stream** (*ByteStream*) – Stream that contains the bufr message
- **b_table** (*Mapping|Template*) – Either a mapping from BUFR descriptor codes to descriptors or a Template describing the message
- **skip_data** (*bool*) – Skip decoding data? Can be used to get only extract metadata of a file to e.g. analysis of decoding errors.

bufrpy.**decode_all**(*stream*, *b_table*)

Decode all BUFR messages from stream into a list of Message objects and a list of decoding errors.

Reads through the stream, decoding well-formed BUFR messages. BUFR messages must start with BUFR and end with 7777. Data between messages is skipped.

Parameters

- **stream** (*ByteStream*) – Stream that contains the bufr message
- **b_table** (*Mapping|Template*) – Either a mapping from BUFR descriptor codes to descriptors or a Template describing the message

3.1.2 BUFR message representation

BUFR Message

BUFR Message is represented as a `Message`.

class `bufrpy.`**`Message`**
Represents a complete BUFR message, of either edition 3 or edition 4.

Variables

- **section0** (*Section0*) – Section 0, start token and length
- **section1** (*Section1v3|Section1v4*) – Section 1, time and source metadata
- **section2** (*Section2*) – Section 2, optional metadata, not processed
- **section3** (*Section3*) – Section 3, message structure
- **section4** (*Section4*) – Section 4, message contents
- **section5** (*Section5*) – Section 5, end token

The `Message` contains several sections. Of these, sections 1, 3 and 4 are the most interesting. Section 1 contains message-level metadata, section 3 describes message structure and section 4 contains the actual data. Note that section 1 comes in multiple variants. The variant used depends on which edition of the BUFR specification was used to code the message.

class `bufrpy.`**`Section1v3`**
Section 1 of a BUFR edition 3 message.

Variables

- **length** (*int*) – Length of Section 1
- **master_table_id** (*int*) – Master table identifier
- **originating_centre** (*int*) – Originating/generating centre
- **originating_subcentre** (*int*) – Originating/generating subcentre
- **update_sequence_number** (*int*) – Update sequence number
- **optional_section** (*int*) – 0 if optional section not present, 1 if present
- **data_category** (*int*) – Data category (identifies Table A to be used)
- **data_subcategory** (*int*) – Data subcategory
- **master_table_version** (*int*) – Master table version
- **local_table_version** (*int*) – Local table version
- **year** (*int*) – Year (originally of century, converted to AD by adding 1900)
- **month** (*int*) – Month
- **day** (*int*) – Day
- **hour** (*int*) – Hour
- **minute** (*int*) – Minute

class `bufrpy.`**`Section1v4`**
Section 1 of a BUFR edition 4 message.

Variables

- **length** (*int*) – Length of Section 1
- **master_table_id** (*int*) – Master table identifier
- **originating_centre** (*int*) – Originating/generating centre
- **originating_subcentre** (*int*) – Originating/generating subcentre
- **update_sequence_number** (*int*) – Update sequence number
- **optional_section** (*int*) – 0 if optional section not present, 1 if present
- **data_category** (*int*) – Data category (identifies Table A to be used)
- **data_subcategory** (*int*) – Data subcategory
- **local_subcategory** (*int*) – Local subcategory
- **master_table_version** (*int*) – Master table version
- **local_table_version** (*int*) – Local table version
- **year** (*int*) – Year (four digits)
- **month** (*int*) – Month
- **day** (*int*) – Day
- **hour** (*int*) – Hour
- **minute** (*int*) – Minute
- **second** (*int*) – Second

class bufrpy.**Section3**

Section 3 of a BUFR message.

Section 3 contains descriptors that describe the actual data format. Descriptors are instances of one of the descriptor classes: descriptors.ElementDescriptor, descriptors.ReplicationDescriptor, descriptors.OperatorDescriptor, descriptors.SequenceDescriptor.

Variables

- **length** (*int*) – Length of Section 3
- **n_subsets** (*int*) – Number of data subsets in the data section
- **flags** (*int*) – Flags describing the data set. See BUFR specification for details.
- **descriptors** – List of descriptors that describe the contents of each data subset.

class bufrpy.**Section4**

Section 4 of a BUFR message.

Section 4 contains the actual message data.

Variables

- **length** (*int*) – Length of Section 4
- **subsets** – Message data as a list of BufrSubsets.

BUFR Template

BUFR Template is represented as a `Template`.

class `bufrpy.template.`**`Template`**
BUFR message template

Describes a BUFR message by listing the descriptors that are present in the message. The same descriptors must be present in the same order in Section 3 of the actual message. The template includes descriptions, length, units and other information required to interpret the descriptor references in the message.

Possible descriptor types are `ElementDescriptor`, `ReplicationDescriptor`, `SequenceDescriptor` and `OperatorDescriptor`.

Variables

- **name** – Name of the template
- **descriptors** – List of descriptors

BUFR Descriptors

BUFR descriptors describe the data of the message. Atomic data elements are described by `ElementDescriptor`, multiple repetitions of a set of descriptors by `ReplicationDescriptor`, fixed sequences of elements by `SequenceDescriptor` and more complex structures by `OperatorDescriptor`.

class `bufrpy.descriptors.`**`ElementDescriptor`**
Describes single value

Data element described with an `ElementDescriptor` is decoded into a `BufrValue`, with either textual or numeric value.

Variables

- **code** (*int*) – Descriptor code
- **length** (*int*) – Length of data, in bits
- **scale** (*int*) – Scaling factor applied to value to get scaled value for encoding
- **ref** (*float*) – Reference value, subtracted from scaled value to get encoded value
- **significance** (*str*) – Semantics of the element
- **unit** (*str*) – Unit of the element, affects interpretation of the encoded data in case of e.g. textual vs. numeric values

class `bufrpy.descriptors.`**`ReplicationDescriptor`**
Describes a repeating collection of values

Data described with a `ReplicationDescriptor` is decoded into a list of lists of values. The outer list has one element per replication and the inner lists one element per replicated field.

Variables

- **code** (*int*) – Descriptor code
- **length** (*int*) – Length of data, always 0
- **fields** (*int*) – Number of following descriptors to replicate
- **count** (*int*) – Number of replications
- **significance** (*str*) – Meaning of the replicated list

class bufrpy.descriptors.**OperatorDescriptor**
Operators 201-208 are supported.

Variables

- **code** (*int*) – Descriptor code, the whole FXY as int
- **length** (*int*) – Length of data. Always 0.
- **operation** (*int*) – Opcode, X of FXY
- **operand** (*int*) – Operand, Y of FXY.
- **operator** (*Operator*) – Operator object for checking conflicts and parsing operand.
- **significance** (*str*) – Operator definition.

class bufrpy.descriptors.**SequenceDescriptor**
Describes a fixed sequence of elements in compact form

Similar to a replication with count 1, but the encoded form is more compact, since the sequence of fields is implicit. Except that at least in NWCSAF Templates the constituent elements of the sequence are also present in the template.

Variables

- **code** (*int*) – Descriptor code
- **length** (*int*) – Length of data, sum of lengths of constituent descriptors
- **descriptor_codes** (*int*) – Sequence containing constituent descriptor codes
- **significance** (*str*) – Meaning of the sequence, always empty string
- **descriptors** – Sequence containing constituent descriptors.

BUFR Values

BUFR values are represented a instances of `BufrValue`. The values of one BUFR message subset are contained in a `BufrSubset`.

class bufrpy.value.**BufrValue**
Contains single value

Contains single value, both in raw and decoded form, plus a link to its descriptor.

Variables

- **raw_value** (*str|int*) – Raw value. Either as hex-encoded string for textual values or an unsigned integer for numeric values
- **value** (*str|int|float|None*) – Decoded value. Value decoded according to its descriptor. Textual values are strings and numeric values floats or ints. Missing value is indicated by `None`.
- **descriptor** (*ElementDescriptor*) – The descriptor of this value

class bufrpy.value.**BufrSubset**
Single BUFR message data subset

Variables **values** – Subset data as a list of BufrValues.

3.1.3 Reading BUFR tables

bufrpy.table.libbufr.**read_tables**(*b_line_stream*, *d_line_stream=None*)
Read BUFR table(s) in from libbufr text file(s).

The return value is a dict that combines the tables read.

Parameters

- **b_line_stream** – Iterable of lines, contents of the B-table file
- **d_line_stream** – Iterable of lines, contents of the D-table file

Returns Mapping from FXY integers to descriptors

Return type dict

Raises

- **NotImplementedError** – if the table contains sequence descriptors
- **ValueError** – if the table contains descriptors with illegal class (outside range [0,3])

3.1.4 Reading BUFR templates

bufrpy.template.safnwc.**read_template**(*line_stream*)
Read SAFNWC message template into a `Template`

SAFNWC template lines look as follows:

```
1 001033 0 0 8 Code table Identification of originating/generating centre
```

Parameters **line_stream** – Lines of SAFNWC template file

Returns the template as Template

Return type Template

Raises **ValueError** if the template contains a desccriptor outside range [0,3]

3.1.5 JSON encoding/decoding

bufrpy.**to_json**(*msg*)
Convert a BUFR message into JSON-encodable form.

The conversion preserves descriptors in Sections 3 and data in Section 4 but not the other metadata.

The resulting JSON is considerably larger than BUFR messages themselves, but compresses to smaller size with e.g. gzip. The JSON is also faster to read and self-descriptive, i.e. a separate descriptor table is no longer necessary.

Parameters **msg** (*Message*) – Message to convert

Returns Dict that can be converted to JSON using the standard json module

Return type dict

bufrpy.**from_json**(*json_obj*)
Convert an object decoded from JSON into a BUFR message

The conversion reads partial contents of BUFR sections 3 and 4 from a dict generated by `to_json()`.

The resulting `bufrpy.Message` contains only section 3 and 4, and the sections only contain descriptors and data, respectively.

Parameters **json_obj** (*dict*) – JSON object to decode

Returns BUFR message with original descriptors and data

Return type Message

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

B

D

E

F

M

O

R

S

T