python-xml-microparser Documentation

Release 0.50beta

Claus Prüfer

Mar 08, 2019

CONTENTS

Contents:

CHAPTER

ONE

INTRO

This Python 3 Module is built to parse XML input data to easy to use internal python structs. As well a transformation from XML to JSON format is possible.

Note: The module does **NOT** provide XSLT/DTD parsing, just plain tag, attribute, value and recursive dependencies are part of this module.

1.1 Dependencies

On a ubuntu 18.04LTS system run the following to get everything working:

```
# install pip3 (pip for python3)
apt-get install python3-pip

# install sphinx documentation system
pip3 install sphinx

# install read the docs theme
pip3 install sphinx_rtd_theme

# install pytest to run integration- and unit tests
pip3 install -U pytest

# install pytest html output module
pip3 install pytest-html
```

1.2 How to run tests

To run all tests (unit and integration) run the following commands:

```
# run all tests (cd to root path)
export PYTHONPATH=./src && py.test --html=testresults.html
```

1.3 Documentation

To build documentation (html, pdf):

```
# build html documentation (found in doc/build/html/index.html)
cd ./doc && make html

# build html documentation (found in doc/build/latex/python-xml-microparser.pdf)
cd ./doc && make latexpdf
```

CHAPTER

TWO

API

2.1 Microparser

class microparser.**Parser**(*payload*)

Bases: object

XML MicroParser class.

Parses simple raw XML data not including DTD or XSLT model description. This data has to be ommitted and would lead to misbehaviour when provided. Also XML namespace parsing is not supported.

The XML will be transformed to internal JSON structs which can easily be iterated over or printed out.

Actually all lines **MUST** end with a "\n" otherwise input will be treated as a single line and tag closings will not be recognized correctly.

See *Examples* section for valid input, generated output and supported features.

Processing flow:

- Process xml input data line by line (add to internal processing list).
- Setup Element() instances for each found element in list, add properties.
- Add elements to Parser._elements list.
- Run Serializer (add/link child elements in OOP based manner).
- Easily iterate (automatically recursive) over the given result elements.

Class Inheritance/Dependencies:

- microparser.Element->microparser.Serializer->transformer.JSONTransformer

The microparser.Serializer class provides members/methods for recursive transformation processing for different transformer module/class types.

Actually only xml transformation is provided, the transformer module is built for future expansion (add multiple formats, e.g. yaml or else).

__init__(*payload*)

Parameters **payload** (*str*) – xml payload data

Variables

- **_elements** (*list[Element]*) – runtime xml item object storage
- **_current_element** (Element) – current processed element
- **_current_line_nr** (*int*) – current processed source line number

- **_current_item_id** (*int*) – current processed numerical internal xml item id

Example

```
>>> import microparser
>>>
>>> payload = '' \
>>>     '<tag1>\n' \
>>>     '    <tag2 a="1" b="value1">\n' \
>>>     '        <tag3 b="1" c="value2">value3</tag3>\n' \
>>>     '    </tag2>\n' \
>>>     '</tag1>\n'
>>>
>>> parser = microparser.Parser(payload)
>>>
>>> parser.build_serializer()
>>> parser.process_json()
>>>
>>> r1 = parser.get_root_element().get_json_dict()
>>> r2 = parser.get_element_by_name('tag2').get_json_dict()
>>> r3 = parser.get_element_by_id(2).get_json_dict()
>>>
>>> print(r1)
>>> print(r2)
>>> print(r3)
```

_add_child_elements_recursive(*element*)
Add child elements recursive.

Parameters **element** (Element) – xlm start element

_current_item_id_gen()
Current item id generator. Simple iterator. Start value 1, incremented by 1 (while True).

Return type Iterator[int]

_current_line_nr_gen()
Current line number generator. Simple iterator. Start value 0, incremented by 1 (while True).

Return type Iterator[int]

_get_last_unclosed_element_id()
Get last unprocessed/unclosed element id.

Returns last non closed element numerical id

Return type int or None

_parse_attributes(*attributes*)
Parse attribute var/value keypairs from string and add to current processed element item.

Parameters **attributes** (*str*) – tag attributes unparsed string

_parse_line(*line*)
Parse single xml data line.

build_serializer()
Build hierarchical serializer by adding all found elements starting with root element.

dump()
Send self (__repr__) to logger (debug)

get_child_elements_by_id(*id*)
Return elements children searched by element numerical id.

Parameters **id** (*int*) – element internal numerical id

Yield found item

Return type elements (list of objects) or None

get_element_by_id(*id*)
Get element by internal numerical id.

Parameters **id** (*int*) – internal numerical element id

Returns found element

Return type *Element* or None

get_element_by_name(*name*)
Get element by xml element id.

Parameters **element** (*str*) – xml tag id

Returns found element

Return type *Element* or None

get_elements()
Return processed elements list.

Returns processed elements

Return type list[*Element*] self._elements

get_root_element()
Return root element.

Returns root element

Return type self._elements[0]

process_json()
Process json transformation.

class microparser.**Element**(*, *name*, *id*, *line_nr*, *parent_id*)
Bases: *microparser.Serializer*

XML Element container class.

Provides methods for xml parsing. Inherits from Serializer class. As well hierarchical assignment by parent element/numerical id is part of this class.

#TODO: This class should be (syntactically correct) moved to own "xml" module (xml.Element).

__init__(*, *name*, *id*, *line_nr*, *parent_id*)

Parameters

- **name** (*str*) – xml element name (id)
- **id** (*int*) – xml element internal processing id
- **line_nr** (*int*) – line number of found xml opening tag in payload data
- **parent_id** (*int*) – parent element numerical id

Variables

- **_name** (*str*) – xml element id

- **_id** (*int*) – internal numerical element id
- **_parent_id** (*int*) – internal numerical parent id (self._id reference)
- **_attributes** (*dict[str]*) – xml tag attributes var/value pairs
- **_content** (*str*) – tag inner content (if no sub elements found)
- **_line_start** (*int*) – start line of found xml opening tag
- **_line_end** (*int*) – end line of found xml closing tag

add_attribute(*name*, *value*)
Add attribute/value pair to self._attributes dictionary.

Parameters

- **name** (*str*) – attribute name
- **value** (*str*) – attribute value

add_content(*content*)
Add content (value found between xml opening and closing tag) to self._content.

Parameters **content** (*str*) – tag value

get_attribute_by_name(*name*)
Get attribute value by given attribute name.

Returns attribute value

Return type str

get_attributes()
Get attributes.

Returns attributes dictionary

Return type dict

get_content()
Get content.

Returns content (value found between xml opening and closing tag)

Return type str

get_id()
Get numerical element id.

Returns internal numerical id

Return type int

get_line_end()
Get line number of found end tag.

Returns get line number of found end tag

Return type int

get_line_start()
Get line number of found start tag.

Returns get line number of found start tag

Return type int

get_name()
Get element name.

Returns element xml id

Return type str

get_parent_element()
Get parent element.

Returns parent element

Return type *Element*

get_parent_id()
Get numerical id of elements parent.

Returns internal numerical parent id

Return type int

set_line_end(*line_nr*)
Set line end (tag found at line nr. in payload)

Parameters **line_nr** (*int*) – line number

set_parent_element(*element*)
Set parent element.

Parameters **element** (Element) – parent element

class microparser.**Serializer**
Bases: *transformer.JSONTransformer*

Serializer class.

Provides methods for element dependency handling.

#TODO: This class should be (syntactically correct) moved to own "xml" module (xml.Element.Serializer).

__init__()
Elements hierarchical connector.

__iter__()
Overloaded iterator.

Will be called on 'yield self', iterate() method makes it recursive.

Returns iter(list[Element])

Return type iterator

_clear_child_elements()
Reset self._child_elements list.

_remove_child_element(*index*)
Remove_element from child_elements list.

Parameters **index** (*int*) – child element list position (index)

add_child_element(*element*)
Append object to self._child_elements list.

Parameters **element** (Element) – append element

get_child_element_count()
Return child element count.

Returns actual child element count

Return type int

get_child_elements()
Get child elements..

Returns child elements list

Return type list[*Element*]

get_element_by_element_id(*element_id*)
Get element by element numerical id.

Returns found element

Return type *Element* or None

get_element_by_element_name(*element_name*)
Get element by element numerical id.

Returns found element

Return type *Element* or None

iterate()
Recursive iterate through hierarchical objects.

2.2 Transformer

class transformer.**JSONTransformer**
Bases: object

JSON transformer class.

Transforms given JSON (text) data into XML format. Data in the first step will be transformed to internal JSON structs and afterwards converted (recursive) to xml.

This class will be inherited by the microparser.Serializer class which provides base members/methods for recursive transformation processing. .

__init__()
Builds json from serialized (connected) Element object hierarchy.

_set_json_attribute(*key*, *value*)
Set single json attribute.

Parameters

- **key** (*str*) – attribute key
- **value** (*mixed*) – attribute value (str or dict)

_set_json_value()
Set json value.

get_json()
Return json result.

Returns json result dictionary (json dumped)

Return type str

get_json_dict()
Return internal json dictionary.

Returns json result dictionary

Return type dict

json_transform()
Transform xml elements to python dictionary.

set_json_attributes()
Set json attributes.

2.3 Helper

class helper.**Looper**(*, *payload*, *function*, *methods=None*)
Bases: object

Looper Class.

Provides processing of list of input items (type should be irrelevant, actually set to type string) applied to multiple processing method references (list).

After single item has been processed by multiple methods specified in methods list (e.g. strip), it will be sent to the final processing function.

__init__(*, *payload*, *function*, *methods=None*)
Loops over payload items. For each item:

- applies methods given in methods list.
- calls function reference given in function argument using item as argument.

Parameters

- **payload** (*list[str]*) – payload list
- **function** (*str*) – function reference for item processing after methods processing
- **methods** (*list[str]*) – list of methods applied to item

Variables

- **_payload** (*list[str]*) – list of payload items to be processed
- **_function** (*str*) – stored function reference
- **_methods** (*list[str]*) – list of methods applied to payload items

Example

```
>>> from microparser import Looper
>>>
>>> def myfunction(payload):
>>>     print(payload)
>>>
>>> payload = 'one,two,three'
>>>
>>> args = {
>>>     'payload': payload.split(','),
>>>     'function': myfunction,
```

(continues on next page)

(continued from previous page)

```
>>>     'methods': ['strip']
>>> }
>>>
>>> Looper(**args).process()
```

generate_methods(*element*)
Generate methods when provided.

process()
Process payload elements.

static process_methods(*methods*, *element*)
Loop over methods.

CHAPTER

THREE

EXAMPLES

The following examples and use cases show how to cope with the XML parser.

Warning: Ech line **MUST** end with a "**\n**", otherwise expect undefined behaviour!

Note: **Loop over results** shows how to efficiently use XML data as configuration method for your projects.

3.1 Use complete XML

Transform the complete XML to internal dict/JSON.

```
import microparser

payload = '' \
    '<tag1>\n' \
    '    <tag2 a="1" b="value1">\n' \
    '        <tag3 c="2" d="value2">value3</tag3>\n' \
    '    </tag2>\n' \
    '</tag1>\n'

parser = microparser.Parser(payload)

parser.build_serializer()
parser.process_json()

r = parser.get_root_element().get_json_dict()

print(r)
```

Output (Printed Representation)

```
{'tag1': {'tag2': {'tag3': 'value3', 'attributes': {'a': '1', 'b': 'value1'}},
↪'attributes': {}}}
```

3.2 Get element by name

Get element value by name (if unique).

```
import microparser

payload = '' \
    '<tag1>\n' \
    '    <tag2 a="1" b="value1">\n' \
    '        <tag3 c="2" d="value2">value3</tag3>\n' \
    '    </tag2>\n' \
    '</tag1>\n'

parser = microparser.Parser(payload)

parser.build_serializer()
parser.process_json()

r = parser.get_element_by_name('tag2').get_json_dict()

print(r)
```

Output (Printed Representation)

```
{'tag2': {'tag3': 'value3', 'attributes': {'a': '1', 'b': 'value1'}}}
```

3.3 Get element by id

Get element value by id.

```
import microparser

payload = '' \
    '<tag1>\n' \
    '    <tag2 a="1" b="value1">\n' \
    '        <tag3 c="2" d="value2">value3</tag3>\n' \
    '    </tag2>\n' \
    '</tag1>\n'

parser = microparser.Parser(payload)

parser.build_serializer()
parser.process_json()

r = parser.get_element_by_id('tag2').get_json_dict()

print(r)
```

3.4 Duplicate elements (distinct name)

Duplicate elements can be used to process multiple configuration items (e.g. a webserver configuration with multiple virtual hosts).

Note: If element is duplicate, it will be appended (order retained) to an internal list (see result dict and **Loop over results** to see how to loop over.

Note: You also can add nested elements to group your vhost configuration, examples will be added in next realeases.

```
import microparser

payload = '' \
    '<config>\n' \
    '    <vhosts\n' \
    '        <vhost>\n' \
    '            <vhost name="vhost1" b="value1"></vhost>\n' \
    '        </vhost>\n' \
    '        <vhost>\n' \
    '            <vhost name="vhost2" b="value2"></vhost>\n' \
    '        </vhost>\n' \
    '        <vhost>\n' \
    '            <vhost name="vhost3" b="value3"></vhost>\n' \
    '        </vhost>\n' \
    '    </vhosts>\n' \
    '</config>\n'

parser = microparser.Parser(payload)

parser.build_serializer()
parser.process_json()

r = parser.get_root_element().get_json_dict()

print(r)
```

Output (Printed Representation)

```
{'config':
    {'vhosts':
        {
            'vhost': [
                {'vhost': {'attributes': {'name': 'vhost1', 'b': 'value1'}},
↪'attributes': {}},
                {'vhost': {'attributes': {'name': 'vhost2', 'b': 'value2'}},
↪'attributes': {}},
                {'vhost': {'attributes': {'name': 'vhost3', 'b': 'value3'}},
↪'attributes': {}}
            ],
            'attributes': {}
        },
        'attributes': {}
    }
}
```

3.5 Loop over results

To loop over the results (for each vhost) from the previous example, do the following:

```
parser = microparser.Parser(payload)
```

(continues on next page)

(continued from previous page)

```
parser.build_serializer()
parser.process_json()

r = parser.get_root_element().get_json_dict()

# note that you have to add the 'vhost' list at the end
for element in r['config']['vhosts']['vhost']:
    vhost_attributes = element['vhost']['attributes']
    vhost_name = vhost_attributes['name']
    vhost_attribute_b = vhost_attributes['b']
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

B

D

E

G

H

I

J

L

M

P

S

T