composite Documentation

Release 0.1.0

Nickolas Fox

Apr 04, 2017

Contents

Release v0.1.0. (*Changelog*)

composite is an ODM tool for converting complex datatypes, such as objects, to and from native Python datatypes and using them to further different file formats conversion such like as XML, JSON, etc

Jump to *Examples* to have get acquainted.

CHAPTER 1

Introduction

Composite project began as a layer between XML (and xml like) documents and JSON in a *already in production* project to move obsolete xml schema towards JSON compatible.

Though this didn't happen **composite** solved its issue designed for. It can maintain layer of compatibility between different document formats and allow user work with them in ODM/ORM style.

For example you have strict xml document schema:

```
<?xml version="1.0" encoding="utf-8"?>
<Company>
    <ceo first_name="Alexander" last_name="Pepyako" age="23"
          gender="male" phone="+79110010203" email="com@alexander.pepyako">
        <id>1</id>
        <sign>Pepyako inc.</sign>
    </ceo>
    <title>Pepyako industries</title>
    <address>Third Vydrokushskaya street, 7 building</address>
    <company_type>GHMb</company_type>
</Company>
```

You would probably want to have python like object to handle its data, bind different methods to it to maintain your own business logic and in final you want to serialize this document to JSON or vice versa.

```
{
"ceo": {
  "_attributes": {
    "first_name": "Alexander",
    "last_name": "Pepyako",
    "age": 23,
    "gender": "male",
    "phone": "+79110010203",
    "email": "com@alexander.pepyako"
  },
  "id": 1,
  "sign": "Pepyako inc."
```

```
},
  "title": "Pepyako industries",
  "address": "Third Vydrokushskaya street, 7 building",
  "company_type": "GHMb"
}
```

Useful links

- You search for ODM/ORM library to handle JSON-like schemas, look for marshmallow project

CHAPTER 2

Installation

Dependencies

- lxml-3.4.4 +
- six-1.9.0 +

Using PyPI

```
pip install composite
```

CHAPTER 3

Examples

XML to JSON

1. Make class for reading document (*documents.py*)

```
from composite import Document, fields


class User(Document):
    id = fields.Field(name='id', type=int)
    sign = fields.Field(name='sign', type=str)

    def get_user_name(self):
        if self.get_attributes():
            return '%s %s' % (self.attributes.first_name,
                              self.attributes.last_name)
        return self.id

    def __str__(self):
        return '%s' % self.attributes.first_name

    class Attributes:
        first_name = fields.AttributeField(name='first_name', type=str)
        last_name = fields.AttributeField(name='last_name', type=str)
        age = fields.AttributeField(name='age', type=int)
        gender = fields.AttributeField(name='gender', type=str)
        phone = fields.AttributeField(name='phone', type=str)
        email = fields.AttributeField(name='email', type=str)
```

```
class Company(Document):
"""
Company document

:param str title: title
:param str address: address
:param str company_type: company type
:param User ceo: company CEO user profile
"""
title = Field('title', str)
address = Field('address', str)
company_type = Field('company_type', str)
ceo = Node('ceo', type=User)

def __str__(self):
    return self.ceo.get_user_name()
```

2. Implement XML document

```
<?xml version="1.0" encoding="utf-8"?>
<Company>
    <ceo first_name="Alexander" last_name="Pepyako" age="23"
         gender="male" phone="+79110010203" email="com@alexander.pepyako">
        <id>1</id>
        <sign>Pepyako inc.</sign>
    </ceo>
    <title>Pepyako industries</title>
    <address>Third Vydrokushskaya street, 7 building</address>
    <company_type>GHMb</company_type>
</Company>
```

3. Read XML document with help of implemented classes

```
>>> from documents import Company
>>> from composite.builders import LXMLDocumentBuilder
>>> from lxml import etree
>>> xml_document = etree.fromstring(open('company.xml', 'rb').read())
>>> company = Company.parse(LXMLDocumentBuilder, xml_document)
>>> company
<Company: Alexander Pepyako>
>>> company.ceo.get_user_name()
'Alexander Pepyako'
```

4. Convert company to JSON

```
>>> from composite.builders import PythonDocumentBuilder
>>> import json
>>> company_dict = company.build(PythonDocumentBuilder, company)
>>> company_dict
{'address': 'Third Vydrokushskaya street, 7 building',
 'ceo': {'_attributes': {'age': 23,
   'email': 'com@alexander.pepyako',
   'first_name': 'Alexander',
   'gender': 'male',
   'last_name': 'Pepyako',
   'phone': '+79110010203'},
  'id': 1,
  'sign': 'Pepyako inc.'},
```

```
 'company_type': 'GHMb',
 'title': 'Pepyako industries'}
>>> json.dumps(company_dict)
'{"ceo": {"_attributes": {"phone": "+79110010203", "first_name":
↪"Alexander", "last_name": "Pepyako",
"gender": "male", "age": 23, "email": "com@alexander.pepyako"}, "id": 1,
"sign": "Pepyako inc."}, "title": "Pepyako industries",
"company_type": "GHMb", "address": "Third Vydrokushskaya street, 7␣
↪building"}'
```

CHAPTER 4

API Reference

Documents

class composite.documents.**Document**

Document (some call them schema) example usage:

```
from composite import Document, fields, builders

class User(Document):
    name = fields.Field('name', str)
    age = fields.Field('age', int)
    address = fields.Field('address', str)

user = User.parse(builders.PythonDocumentBuilder,
    {'name': 'Alice', 'age': 10, 'address': 'Wonderland'})
user.name  # Alice
User.build(builders.PythonDocumentBuilder, user)
{'name': 'Alice', 'age': 10, 'address': 'Wonderland'}
```

classmethod **build**(*build composite according to its build class business logic*)

Parameters

- **builder_class** – builder class (*composite.builders.BaseDocumentBuilder*)
- **document** (*composite.Document*) – python instance (composite.Document)

Return type any

Returns any object, for example lxml.etree.Element if build process had been managed with lxml builder class

get_attributes()
get attributes node

Return type *DocumentAttribute* or None

Returns attributes

has_attributes()
if document has attributes

Return type bool

Returns check if given document has attributes

classmethod parse(*builder_class*, *source*)
parse to python-object instance with build_class source data

Parameters

- **builder_class** – builder class (*composite.builders.BaseDocumentBuilder*)
- **source** – any data to process

Return type composite.Document

Returns document instance

class composite.documents.**DocumentAttribute**
Base class for building attributes inside *composite.documents.Document* instances example:

```
from composite import Document, fields, builders

class User(Document):
    name = fields.Field('name', str)

    class Attribute:
        age = fields.AttributeField('age', int)
        gender = fields.AttributeField('gender', str)

user = User.parse(builders.PythonDocumentBuilder,
    {'name': 'Alice', '_attributes': {'age': 10, 'gender': 'female'}})
user.attributes.gender  # female
User.build(builders.PythonDocumentBuilder, user)
{'name': 'Alice', '_attributes': {'age': 10, 'gender': 'female'}}
```

get_attribute_fields()
get attribute fields

Return type dict

Returns fields

values()
attribute values bind

Return type list

Returns attribute values

class composite.documents.**DocumentMeta**(*name*, *bases*, *attrs*)
Document Meta class for building documents

Fields

class composite.fields.**AttributeField**(*name*, *type*, *default=None*)
Same as *composite.fields.Field* but using only in attributes, as they include only simple data.

```
age = AttributeField('age', int)  # int() by default
```

visit(*visitor*, *source*)
invoke visitor's *composite.visitors.FieldVisitor.visit_attribute_field()* method call with given source.

Parameters **source** (*any*) – base data types source data

Return type None

Returns None

class composite.fields.**BaseField**(*name*, *type*, *default=None*)
Base Field abstract class

class composite.fields.**Field**(*name*, *type*, *default=None*)
Field processes simple types of data, for example: int, float, str:

```
age = Field('age', int, 21)
name = Field('name', str, '')
```

visit(*visitor*, *source*)
invoke visitor's *composite.visitors.FieldVisitor.visit_field()* method call with given source.

Parameters **source** (*any*) – any source data

Return type None

Returns None

class composite.fields.**ListField**(*name*, *type*, *default=None*)
List of fields combined together:

```
mana_cost = ListField('mana_cost', int, [90, 100, 110, 120])
player_names = ListField('player_names', str)  # empty list, []
```

visit(*visitor*, *source*)
invoke visitor's *composite.visitors.FieldVisitor.visit_list_field()* method call with given source.

Parameters **source** (*any*) – any source data

Return type None

Returns None

class composite.fields.**ListNode**(*name*, *type*, *default=None*)
List of nodes

```
class User(Document):
    age = Field('age', int)
    name = Field('name', str)

class Users(Document):
    users = ListNode('users', User)
```

visit(*visitor*, *source*)
invoke visitor's *composite.visitors.FieldVisitor.visit_list_node()* method call with given source.

Parameters **source** (*any*) – any source of data

Return type None

Returns None

class composite.fields.**MetaListField**(*name*, *type*, *default=None*)
For list fields, list nodes, list etc usage

class composite.fields.**Node**(*name*, *type*, *default=None*)
Node field serves to read different data types combined together (ADT).

```
class User(Document):
    age = Field('age', int)
    name = Field('name', str)

class Company(Document):
    title = Field('title', str)
    address = Field('address', str)

class Profile(Document):
    user = Node('user', User)
    company = Node('company', Company)
```

visit(*visitor*, *source*)
invoke visitor's *composite.visitors.FieldVisitor.visit_node()* method call with given source.

Parameters **source** (*any*) – any source of data

Return type None

Returns None

Builders

class composite.builders.**BaseDocumentBuilder**(*document*)
Base builder (abstract) class for building/parsing Documents

build(*node_name='document'*)
build instance from python object (document)

Parameters **node_name** (*str*) – name of the node to

Return type any

Returns built instance

build_attributes(*source_object*)
builds attributes

Parameters **source_object** (*str*) – source object

Returns attribute object

Raises **NotImplemented** –

- build object should be implemented in real classes

build_object(*node_name*)
builds object

Parameters **node_name** (*str*) – node name

Returns object

Raises **NotImplemented** –

- build object should be implemented in real classes

get_attribute_class()
get attribute class

Return type *composite.documents.DocumentAttribute*

Returns attribute class

get_attribute_fields()
get document attribute fields

Return type dict

Returns fields

get_build_visitor(*document*)
get build visitor

Return type *composite.visitors.FieldVisitor*

Returns build visitor class

get_build_visitor_class()
get build visitor class

Return type type

Returns build visitor class

get_document_fields()
get document fields

Return type dict

Returns document fields

get_document_fields_mapping()
get document fields mapping

Return type dict

Returns mapping

get_parse_visitor()
get parse visitor

Return type *composite.visitors.FieldVisitor*

Returns parse visitor class

get_parse_visitor_class()
get parse visitor class

Return type type

Returns parse visitor class

init_blank_attributes(*source_object*)
initiate blank attributes

Parameters **source_object** – source object

Raises **NotImplemented** –

- should be implement in children classes

classmethod iterate(*source*)
iterate through source

Parameters **source** (*any*) – any source data, for example dict

Return type generator

Returns generator with tuple[node name, node content]

parse(*source*)
parse source data with any format to final document

Parameters **source** – source data (dict, lxml.etree.Element element, etc)

Return type composite.Document

Returns document instance

class composite.builders.**LXMLDocumentBuilder**(*document*)
XML documents builder class for parse documents from raw format (via lxml) and build to them directly.

build_attributes(*source_object*)
attributes

Parameters **source_object** (*lxml.etree.Element*) – etree element

Return type dict

Returns attributes

build_object(*node_name*)
returns blank lxml.etree.Element object

Parameters **node_name** (*str*) – document node name

Return type lxml.etree.Element

Returns lxml Element instance

init_blank_attributes(*source_object*)
lxml etree Element objects already has blank initiated attribute children object

Parameters **source_object** (*lxml._etree.Element*) – source element

Returns None

Return type None

classmethod iterate(*source*)
iterate through source document

Parameters **source** (*lxml.etree.Element | lxml.etree._Attrib*) – xml document or its attribute

Return type generator

Returns tuple[node name, node]

class composite.builders.**PythonDocumentBuilder**(*document*)

Python documents builder class for parse documents from raw (dict) format and build to them directly. Could help to build json documents with json/simplejson/anyjson/etc.

build_attributes(*source_object*)

attributes

Parameters **source_object** (*dict*) – source element

Return type dict

Returns attributes

init_blank_attributes(*source_object*)

update source object with attributes

Parameters **source_object** (*dict*) – source object

Returns None

Return type None

classmethod **iterate**(*source*)

iterate through source document

Parameters **source** (*lxml.etree.Element*) – xml document

Return type generator

Returns tuple[node name, node]

Visitors

class composite.visitors.**FieldVisitor**(*builder_class*, *composite*)

Field visitor, helps to parse and build documents. There are 5 types of fields:

- `attribute field`, field attribute:

```
<font size="12em" color="black" weight="bold">
```

`size`, `color` and `weight` are attributes

- `field`, simple data field:

```
<Document>
    <x>0</x>
    <name>Jim</name>
    <pi>3.14</pi>
</Document>
```

- `list field`, simple data fields combined in array/list:

```
<Document>
    <name>Jim</name>
    <name>Jill</name>
    <name>Jerry</name>
</Document>
```

- `node`, block with several data in it:

```
<User>
    <name>Alexander</name>
    <last_name>Pepyako</last_name>
    <age>32</age>
    <email>com@alexander.pepyako</email>
</User>
```

- `list node`, several nodes combined in array/list:

```
<Clients>
    <user>
        <name>Alexander</name>
        <last_name>Pepyako</last_name>
        <age>32</age>
        <email>com@alexander.pepyako</email>
    </user>
    <user>
        <name>Mary</name>
        <last_name>Noname</last_name>
        <age>27</age>
        <email>mary@noname.nozone</email>
    </user>
</Clients>
```

visit_attribute_field(*field*, *source*)
visit attribute field

Parameters

- **field** (composite.fields.AttributeField) – field to visit
- **source** – any source of data

Return type None

Returns None

visit_field(*field*, *source*)
visit field

Parameters

- **field** (composite.fields.Field) – field to visit
- **source** – any source of data

Return type None

Returns None

visit_list_field(*field*, *source*)
visit list field

Parameters

- **field** (composite.fields.ListField) – field to visit
- **source** – any source of data

Return type None

Returns None

visit_list_node(*node*, *source*)
visit list node

Parameters

- **field** (composite.fields.ListNode) – field to visit
- **source** – any source of data

Return type None

Returns None

visit_node(*node*, *source*)
visit node

Parameters

- **node** (composite.fields.Node) – node field to visit
- **source** – any source of data

Return type None

Returns None

Exceptions

exception composite.exceptions.**ImproperlyConfigured**(*message*, *errors=None*)
Improperly configure exception

CHAPTER 5

Changelog

0.1.0

- Bump version
- Python 3.5+ (including python 3.6) compatibility check

0.0.1

Initial release

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

c

d

Index

I

L

M

N

P

V