paxb

Release 0.3.1

Nov 09, 2019

Contents

1 Requirements **3**

2 The User Guide **5**
- 2.1 Installation . . . 5
- 2.2 Quick start . . . 5
- 2.3 XML binding . . . 8
- 2.4 Serialization . . . 12
- 2.5 Deserialization . . . 14
- 2.6 Namespaces . . . 15
- 2.7 Errors . . . 17
- 2.8 attrs library integration . . . 18

3 The API Documentation **21**
- 3.1 Developer Interface . . . 21

4 Development **25**
- 4.1 Development . . . 25

Python Module Index **27**

Index **29**

`paxb` is a library that provides an API for mapping between XML documents and Python objects.

`paxb` library implements the following functionality:

- *Deserialize* XML documents to Python objects
- Validate deserialized fields
- Access and update Python object fields
- *Serialize* Python objects to XML documents

`paxb` provides an efficient way of mapping between an XML document and a Python object. Using `paxb` developers can write less boilerplate code emphasizing on application domain logic.

Since paxb is based on attrs library `paxb` and attrs API can be *mixed* together.

CHAPTER 1

Requirements

- attrs

CHAPTER 2

The User Guide

2.1 Installation

This part of the documentation covers the installation of *paxb* library.

2.1.1 Installation using pip

To install *paxb*, run:

```
$ pip install paxb
```

2.1.2 Installation from source code

You can clone the repository:

```
$ git clone git@github.com:dapper91/paxb.git
```

Then install it:

```
$ cd paxb
$ pip install .
```

2.2 Quick start

Suppose you have an xml document `user.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<doc:envelope xmlns="http://www.test.org"
              xmlns:doc="http://www.test1.org">
    <doc:user name="Alex" surname="Ivanov" age="26">

        <doc:birthdate year="1992" month="06" day="14"/>

        <doc:contacts>
            <doc:phone>+79204563539</doc:phone>
            <doc:email>alex@gmail.com</doc:email>
            <doc:email>alex@mail.ru</doc:email>
        </doc:contacts>

        <doc:documents>
            <doc:passport series="3127" number="836815"/>
        </doc:documents>

        <data:occupations xmlns:data="http://www.test2.org">
            <data:occupation title="yandex">
                <data:address>Moscow</data:address>
                <data:employees>8854</data:employees>
            </data:occupation>
            <data:occupation title="skbkontur">
                <data:address>Yekaterinburg</data:address>
                <data:employees>7742</data:employees>
            </data:occupation>
        </data:occupations>

    </doc:user>
</doc:envelope>
```

To deserialize the document you could use `xml` library api to parse the document and then access and modify the parsed xml DOM manually. Such an imperative code has a lot of boilerplate operations that takes a lot of time and can lead to bugs. Instead you can use `paxb` api to write a declarative style code. All you need to describe field mappings and types, `paxb` will serialize and deserialize data for you:

```
import json
import re
from datetime import date

import attr
import paxb as pb


@pb.model(name='occupation', ns='data', ns_map={'data': 'http://www.test2.org'})
class Occupation:
    title = pb.attr()
    address = pb.field()
    employees = pb.field(converter=int)


@pb.model(name='user', ns='doc', ns_map={'doc': 'http://www.test1.org'})
class User:
    name = pb.attr()
    surname = pb.attr()
    age = pb.attr(converter=int)
```

(continues on next page)

(continued from previous page)

```
    birth_year = pb.wrap('birthdate', pb.attr('year', converter=int))
    birth_month = pb.wrap('birthdate', pb.attr('month', converter=int))
    birth_day = pb.wrap('birthdate', pb.attr('day', converter=int))

    @property
    def birthdate(self):
        return date(year=self.birth_year, month=self.birth_month, day=self.birth_day)

    @birthdate.setter
    def birthdate(self, value):
        self.birth_year = value.year
        self.birth_month = value.month
        self.birth_day = value.day

    phone = pb.wrap('contacts', pb.field())
    emails = pb.wrap('contacts', pb.as_list(pb.field(name='email')))

    passport_series = pb.wrap('documents/passport', pb.attr('series'))
    passport_number = pb.wrap('documents/passport', pb.attr('number'))

    occupations = pb.wrap(
        'occupations', pb.lst(pb.nested(Occupation)), ns='data', ns_map={'data':
↪'http://www.test2.org'}
    )

    citizenship = pb.field(default='RU')

    @phone.validator
    def check(self, attribute, value):
        if not re.match(r'\+\d{11,13}', value):
            raise ValueError("phone number is incorrect")


with open('user.xml') as file:
    xml = file.read()
```

Then the deserialized object can be modified and serialized back to xml document or converted to json format:

```
try:
    user = pb.from_xml(User, xml, envelope='doc:envelope', ns_map={'doc': 'http://www.
↪test1.org'})
    user.birthdate = user.birthdate.replace(year=1993)

    with open('user.json') as file:
        json.dump(attr.asdict(user), file)

except (pb.exc.DeserializationError, ValueError) as e:
    print(f"deserialization error: {e}")
```

user.json:

```
{
    "age": 26,
    "birth_day": 14,
    "birth_month": 6,
    "birth_year": 1993,
    "citizenship": "RU",
```

(continues on next page)

(continued from previous page)

```
    "emails": ["alex@gmail.com", "alex@mail.ru"],
    "name": "Alexey",
    "occupations": [
        {
            "address": "Moscow",
            "employees": 8854,
            "title": "yandex"
        },
        {
            "address": "Yekaterinburg",
            "employees": 7742,
            "title": "skbkontur"
        }
    ],
    "passport_number": "836815",
    "passport_series": "3127",
    "phone": "+79204563539",
    "surname": "Ivanov"
}
```

2.3 XML binding

2.3.1 model

The `paxb.model()` decorator is used to describe a mapping of a python class to an xml element. All encountered class fields are mapped to the xml subelements. In the following example `User` class attributes `name` and `surname` are mapped to the corresponding xml elements. The model:

```
import paxb as pb

@pb.model
class User:
    name = pb.field()
    surname = pb.field()
```

has a complete mapping description for the document

```
<User>
    <name>Alex</name>
    <surname>Ivanov</surname>
</User>
```

By default class name is used as an xml tag name for a mapping. The default behavior can be altered using decorator `name` argument. The `User` class can be rewritten as follows:

```
import paxb as pb

@pb.model(name='user')
class User:
    name = pb.field()
    surname = pb.field()
```

```
<user>
    <name>Alex</name>
    <surname>Ivanov</surname>
</user>
```

2.3.2 field

The *paxb.field()* function describes a mapping of a python class field to an xml subelement. In the following example fields `name` and `surname` are mapped to the corresponding xml subelements. The name of the fields is used as an xml tag name for a mapping.

```
import paxb as pb

@pb.model
class User:
    name = pb.field()
    surname = pb.field()
```

```
<User>
    <name>Alex</name>
    <surname>Ivanov</surname>
</User>
```

Similarly to the `paxb.model()` decorator the default behavior can be altered using the `name` argument.

```
import paxb as pb

@pb.model
class User:
    name = pb.field(name="Name")
    surname = pb.field(name="Surname")
```

```
<User>
    <Name>Alex</Name>
    <Surname>Ivanov</Surname>
</User>
```

2.3.3 attribute

The *paxb.attr()* function describes a mapping of a python class field to an xml element attribute. In the following example fields `name` and `surname` are mapped to the corresponding `User` element attributes. The name of the fields is used as an xml tag name for a mapping.

```
import paxb as pb

@pb.model
class User:
    name = pb.attr()
    surname = pb.attr()
```

```
<User name="Alex" surname="Ivanov"/>
```

Similarly to the *paxb.field()* function the default behavior can be altered using `name` argument.

```
import paxb as pb

@pb.model
class User:
    name = pb.attribute(name="Name")
    surname = pb.attribute(name="Surname")
```

```
<User Name="Alex" Name="Ivanov"/>
```

2.3.4 nested

The *paxb.nested()* function is used to describe a mapping of a python class to an xml element. It is similar to the `paxb.model()` decorator, but declares a nested one. Beyond that it acts the same. The following example illustrates using nested classes:

```
imort paxb as pb

@pb.model
class Passport:
    series = pb.attribute()
    number = pb.attribute()

@pb.model
class User:
    name = pb.attribute()
    surname = pb.attribute()
    passport = pb.nested(Passport)
```

```
<User name="Alex" surname="Ivanov">
    <Passport series="4581" number="451672"/>
</User>
```

2.3.5 as_list

The *paxb.as_list()* function describes a mapping of a python class field to xml subelements. The corresponding subelements will be deserialized to a list. An element of a list can be field, nested class or wrapper (will be described later). Look at the example:

```
import paxb as pb

@pb.model
class User:
    emails = pb.as_list(pb.field(name="Email"))
```

```
<User>
    <Email>alex@mail.ru</Email>
    <Email>alex@gmail.com</Email>
    <Email>alex@yandex.ru</Email>
</User>
```

2.3.6 wrapper

It is common case when a mapped element is placed in a subelement but declaring a nested class is redundant. Here the *paxb.wrapper()* function comes forward. Let's look at the example:

```
import paxb as pb

@pb.model
class User:
    email = pb.wrapper('contacts', pb.field())
```

```
<User>
    <contacts>
        <email>alex@gmail.com</email>
    </contacts>
</User>
```

Here `email` is a direct field of the `User` class but in the xml tree it is placed in `contacts` subelement.

One *paxb.wrapper()* can be can be wrapped by another:

```
import paxb as pb

@pb.model
class User:
    email = pb.wrapper('info', pb.wrapper('contacts', pb.field()))
    ...
```

```
<User>
    <info>
        <contacts>
            <email>alex@gmail.com</email>
        </contacts>
    </info>
</User>
```

A path can be used instead of a tag name. The following model is equivalent to the former one:

```
import paxb as pb

@pb.model
class User:
    email = pb.wrapper('info/contacts', pb.field())
```

2.3.7 let's put it all together

All the functions can be mixed together. Look at the more advanced example:

```
<envelope>
    <user name="Alexey" surname="Ivanov" age="26">

        <birthdate year="1992" month="06" day="14"/>

        <contacts>
            <phone>+79204563539</phone>
```

(continues on next page)

(continued from previous page)

```
            <email>alex@gmail.com</email>
            <email>alex@mail.ru</email>
        </contacts>

        <documents>
            <passport series="3127" number="836815"/>
        </documents>

        <occupations>
            <occupation title="yandex">
                <address>Moscow</address>
                <employees>8854</employees>
            </occupation>
            <occupation title="skbkontur">
                <address>Yekaterinburg</address>
                <employees>7742</employees>
            </occupation>
        </occupations>

    </user>
</envelope>
```

```
import paxb as pb

@pb.model(name='occupation')
class Occupation:
    title = pb.attribute()
    address = pb.field()
    employees = pb.field()

@pb.model(name='user')
class User:
    name = pb.attribute()
    surname = pb.attribute()
    age = pb.attribute()

    phone = pb.wrap('contacts', pb.field())
    emails = pb.wrap('contacts', pb.as_list(pb.field(name='email')))

    passport_series = pb.wrap('documents/passport', pb.attribute('series'))
    passport_number = pb.wrap('documents/passport', pb.attribute('number'))

    occupations = pb.wrap('occupations', pb.lst(pb.nested(Occupation)))
```

2.4 Serialization

paxb implements an API for serializing an python object to an xml string. To serialize an object just pass it to a *paxb.to_xml()* method:

```
>>> import paxb as pb
>>>
>>> @pb.model
... class User:
...     name = pb.attribute()
```

(continues on next page)

(continued from previous page)

```
...     surname = pb.attribute()
...     email = pb.field()
...     phone = pb.field()
...
>>> obj = User(name='Alex', surname='Ivanov', email='alex@gmail.com', phone=
↪'+79123457323')
>>>
>>> xml_string = pb.to_xml(obj)
>>> print(xml_string)
b'<User name="Alex" surname="Ivanov"><email>alex@gmail.com</email><phone>+79123457323
↪</phone></User>'
```

By default *paxb.to_xml()* method serializes an object to a root element in an xml tree, class name is used as the element name, the element namespace is empty. The default behaviour can be altered using *paxb.to_xml()* argument. Look at the example:

```
>>> import paxb as pb
>>>
>>> @pb.model
... class User:
...     name = pb.attribute()
...     surname = pb.attribute()
...     email = pb.field()
...     phone = pb.field()
...
>>> obj = User(name='Alex', surname='Ivanov', email='alex@gmail.com', phone=
↪'+79123457323')
>>>
>>> xml_string = pb.to_xml(obj, envelope='root', name='user', ns='test', ns_map={'test
↪': 'http://www.test.org'}, encoding='unicode')
>>> print(xml_string)
<root xmlns:test="http://www.test.org"><test:user name="Alex" surname="Ivanov">
↪<test:email>alex@gmail.com</test:email><test:phone>+79123457323</test:phone></
↪test:user></root>
```

The `encoding` argument is an additional argument passed to `xml.etree.ElementTree.tostring()` method.

2.4.1 Encoder

By default an object fields serialized using the following rules:

- `str` field is serialized as it is.
- `bytes` field serialized using base64 encoding.
- `datetime.datetime` field serialized as iso formatted string.
- `datetime.date` field serialized as iso formatted string.
- other types serialized using `__str__()` method.

The default behaviour can be altered using `encoder` argument. Encoder must be a callable object that accepts an encoded value and returns its `str` representation.

Since `paxb` is based on `attr` library, `attr.asdict()` function can be used to serialize an object to a json string:

```
>>> import attr
>>> import json
>>> import paxb as pb
>>>
>>> @pb.model
... class User:
...     name = pb.attribute()
...     surname = pb.attr()
...     email = pb.field()
...     phone = pb.field()
...
>>> obj = User(name='Alex', surname='Ivanov', email='alex@gmail.com', phone=
↪'+79123457323')
>>>
>>> obj_dict = attr.asdict(obj)
>>> json.dumps(obj_dict)
'{"name": "Alex", "surname": "Ivanov", "email": "alex@gmail.com", "phone":
↪"+79123457323"}'
```

2.5 Deserialization

paxb implements an API for deserializing an xml string to a python object. To serialize an object just pass a class and an xml string to a *paxb.from_xml()* method:

```
>>> import paxb as pb
>>>
>>> @pb.model
... class User:
...     name = pb.attribute()
...     surname = pb.attribute()
...     email = pb.field()
...     phone = pb.field()
...
>>> xml_str = '<User name="Alex" surname="Ivanov"><email>alex@gmail.com</email><phone>
↪+79123457323</phone></User>'
>>> pb.from_xml(User, xml_str)
User(name='Alex', surname='Ivanov', email='alex@gmail.com', phone='+79123457323')
```

By default *paxb.from_xml()* method deserializes an object from a root element in an xml tree, class name is used as an element name, the element namespace is empty. The default behaviour can be altered using *paxb.from_xml()* additional arguments. Look at the example:

```
>>> import paxb as pb
>>>
>>> @pb.model
... class User:
...     name = pb.attribute()
...     surname = pb.attribute()
...     email = pb.field()
...     phone = pb.field()
...
>>> xml_str = '<root xmlns:test="http://www.test.org"><test:user name="Alex" surname=
↪"Ivanov"><test:email>alex@gmail.com</test:email><test:phone>+79123457323</
↪test:phone></test:user></root>'
>>> pb.from_xml(User, xml_str, envelope='root', name='user', ns='test', ns_map={'test
↪': 'http://www.test.org'}, required=True)
```

(continues on next page)

(continued from previous page)

```
User(name='Alex', surname='Ivanov', email='alex@gmail.com', phone='+79123457323')
```

The `required` argument tells the deserializer to raise an exception if the element is not found in the xml tree, otherwise `None` will be returned (see *Errors*).

By default all fields deserialized as `str` types. The default behaviour can be altered using a `converter` parameter. See `attr.ib()`.

```
>>> import datetime
>>> import paxb as pb
>>>
>>> @pb.model
... class User:
...     age = pb.attribute(converter=int)
...     birthdate = pb.field(converter=datetime.date.fromisoformat)
...
>>> xml_str = '<User age="26"><birthdate>1993-08-21</birthdate></User>'
>>> pb.from_xml(User, xml_str)
User(age=26, birthdate=datetime.date(1993, 8, 21))
```

To deserialize an object from a json document use python `json` package:

```
>>> import json
>>> import paxb as pb
>>>
>>> @pb.model
... class User:
...     name = pb.attribute()
...     surname = pb.attribute()
...     email = pb.field()
...     phone = pb.field()
...
>>> json_str = '{"name": "Alex", "surname": "Ivanov", "email": "alex@gmail.com",
↪"phone": "+79123457323"}'
>>> User(**json.loads(json_str))
User(name='Alex', surname='Ivanov', email='alex@gmail.com', phone='+79123457323')
```

2.6 Namespaces

2.6.1 Namespace inheritance

The default namespace of any element is an empty namespace. Functions *`paxb.field()`*, `paxb.model()`, *`paxb.wrapper()`* and *`paxb.nested()`* have a `ns` argument which alters the default (empty) namespace by a passed one. Compare two examples:

```
>>> import paxb as pb
>>>
>>> @pb.model
... class User:
...     name = pb.field()
...     surname = pb.field()
...
>>> user = User(name='Alex', surname='Ivanov')
```

(continues on next page)

(continued from previous page)

```
>>>
>>> pb.to_xml(user)
b'<User><name>Alex</name><surname>Ivanov</surname></User>'
```

```
>>> import paxb as pb
>>>
>>> @pb.model(ns='test1')
... class User:
...     name = pb.field(ns='test2')
...     surname = pb.field(ns='test3')
...
>>> user = User(name='Alex', surname='Ivanov')
>>>
>>> pb.to_xml(user, ns_map={
...     'test1': 'http://test1.org',
...     'test2': 'http://test2.org',
...     'test3': 'http://test3.org',
... })
b'<test1:User xmlns:test1="http://test1.org" xmlns:test2="http://test2.org"␣
↪xmlns:test3="http://test3.org"><test2:name>Alex</test2:name><test3:surname>Ivanov</
↪test3:surname></test1:User>'
```

The `ns_map` argument describes a mapping from a namespace prefix to a full name that will be used during serialization and deserializaion.

The namespace of *`paxb.field()`*, *`paxb.wrapper()`* and *`paxb.nested()`* is inherited from the containing model if it is not declared explicitly. Look at the example:

```
from xml.dom.minidom import parseString
import paxb as pb

@pb.model
class Passport:                     # implicit namespace, will be inherited from a␣
↪containing model
    series = pb.field()             # implicit namespace, the same as of Passport␣
↪model
    number = pb.field(ns='test3')   # explicit namespace 'test3'

@pb.model(ns='test2')               # namespace 'test2' explicitly set for␣
↪DrivingLicence and implicitly set for all contained elements
class DrivingLicence:               # explicit namespace 'test2'
    number = pb.field()             # implicit namespace 'test2'

@pb.model(ns='test1')               # namespace 'test1' explicitly set for User and␣
↪implicitly set for all contained elements
class User:                         # explicit namespace 'test1'
    name = pb.field()               # implicit namespace 'test1'
    surname = pb.field(ns='test2')  # explicit namespace 'test2'

    passport = pb.nested(Passport)                  # default namespace for the␣
↪contained model Passport will be set to 'test1'
    driving_licence = pb.nested(DrivingLicence)     # default namespace for the␣
↪contained model DrivingLicence will be set to 'test1'

passport = Passport(series="5425", number="541125")
licence = DrivingLicence(number="673457")
```

(continues on next page)

(continued from previous page)

```
user = User(name='Alex', surname='Ivanov', passport=passport, driving_licence=licence)

xml = pb.to_xml(user, ns_map={
    'test1': 'http://test1.org',
    'test2': 'http://test2.org',
    'test3': 'http://test3.org',
}, encoding='unicode')
print(parseString(xml).toprettyxml(indent=' ' * 4), end='')
```

Output:

```
<?xml version="1.0" ?>
<test1:User xmlns:test1="http://test1.org" xmlns:test2="http://test2.org" xmlns:test3=
↪"http://test3.org">
    <test1:name>Alex</test1:name>
    <test2:surname>Ivanov</test2:surname>
    <test1:Passport>
        <test1:series>5425</test1:series>
        <test3:number>541125</test3:number>
    </test1:Passport>
    <test2:DrivingLicence>
        <test2:number>673457</test2:number>
    </test2:DrivingLicence>
</test1:User>
```

2.7 Errors

The package has two main exceptions: *paxb.exceptions.SerializationError* and *paxb.exceptions.DeserializationError*.

paxb.exceptions.DeserializationError is raised when any deserialization error occurs. The most common case it is raised is a required element is not found in an xml tree. Look at the example:

```
>>> import paxb as pb
>>>
>>> @pb.model
... class User:
...     name = pb.attribute()
...
>>> xml_str = '<User/>'
>>> pb.from_xml(User, xml_str)
Traceback (most recent call last):
...
paxb.exceptions.DeserializationError: required attribute '/User[1]/name' not found
```

This error is raised when either of *paxb.field()*, *paxb.attr()*, *paxb.nested()* or *paxb.wrapper()* element not found. This behaviour can be altered by passing a default value to an element:

```
>>> import paxb as pb
>>>
>>> @pb.model
... class User:
...     name = pb.attr(default='Alex')
...
```

(continues on next page)

(continued from previous page)

```
>>> xml_str = '<User/>'
>>> pb.from_xml(User, xml_str)
User(name='Alex')
```

The same applies to *paxb.field()*, *paxb.nested()* and *paxb.wrapper()*.

paxb.exceptions.SerializationError is raised when any serialization error occurs. The most common case it is raised is a required element is not set. Look at the example:

```
>>> import paxb as pb
>>>
>>> @pb.model
... class User:
...     name = pb.attr()
...
>>> obj = User(name=None)
>>> pb.to_xml(obj)
Traceback (most recent call last):
...
paxb.exceptions.SerializationError: required attribute 'name' is not set
```

This error is raised when either of *paxb.field()*, *paxb.attr()*, *paxb.nested()* or *paxb.wrapper()* element is not set. This behaviour can be altered by passing a default value to an element:

```
>>> import paxb as pb
>>>
>>> @pb.model
... class User:
...     name = pb.attr(default='Alex')
...
>>> obj = User()
>>> pb.to_xml(obj)
b'<User name="Alex" />'
```

2.8 attrs library integration

Since `paxb` is based on attrs library `paxb` and `attrs` APIs can be mixed together.

Decorator `paxb.model()` accepts `attr.s()` function arguments as `**kwargs` and internally passes them to it. For example you can pass `str=True` argument to ask `attrs` library to generate `__str__` method for a class.

Functions *paxb.attr()*, *paxb.field()* and *paxb.nested()* accept `attr.ib()` function arguments as `**kwargs` and internally passes them to it. For example you can pass `default` or `factory` argument to set a default value for a class field or `converter` argument to convert a value to an appropriate type. Look at the example:

```
>>> import paxb as pb
>>>
>>> @pb.model
... class Model:
...     field = pb.field(default='1', converter=int)
...
>>> Model()
Model(field=1)
```

paxb in conjuction with attrs library can be used as a flexible xml-to-json converter and vise versa. All you need is just to declare a model, fields and field types, the rest is up to paxb.

Suppose you have an xml document user.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<doc:envelope xmlns:doc="http://www.test1.org">
    <doc:user name="Alex" surname="Ivanov" age="26">

        <doc:contacts>
            <doc:phone>+79204563539</doc:phone>
            <doc:email>alex@gmail.com</doc:email>
            <doc:email>alex@mail.ru</doc:email>
        </doc:contacts>

        <data:occupations xmlns:data="http://www.test2.org">
            <data:occupation title="yandex">
                <data:address>Moscow</data:address>
                <data:employees>8854</data:employees>
            </data:occupation>
            <data:occupation title="skbkontur">
                <data:address>Yekaterinburg</data:address>
                <data:employees>7742</data:employees>
            </data:occupation>
        </data:occupations>

    </doc:user>
</doc:envelope>
```

First you need to describe models. Then deserialize the document to an object and call attr.asdict() attrs library API method:

```
import json
import attr
import paxb as pb

xml = '''<?xml version="1.0" encoding="utf-8"?>
<doc:envelope xmlns:doc="http://www.test1.org">
    <doc:user name="Alex" surname="Ivanov" age="26">

        <doc:contacts>
            <doc:phone>+79204563539</doc:phone>
            <doc:email>alex@gmail.com</doc:email>
            <doc:email>alex@mail.ru</doc:email>
        </doc:contacts>

        <data:occupations xmlns:data="http://www.test2.org">
            <data:occupation title="yandex">
                <data:address>Moscow</data:address>
                <data:employees>8854</data:employees>
            </data:occupation>
            <data:occupation title="skbkontur">
                <data:address>Yekaterinburg</data:address>
                <data:employees>7742</data:employees>
            </data:occupation>
        </data:occupations>

    </doc:user>
```

(continues on next page)

(continued from previous page)

```
</doc:envelope>
'''

@pb.model(name="occupation")
class Occupation:
    title = pb.attribute()
    address = pb.field()
    employees = pb.field(converter=int)

@pb.model(name="user", ns="doc")
class User:
    name = pb.attribute()
    surname = pb.attribute()
    age = pb.attribute(converter=int)

    phone = pb.wrap("contacts", pb.field())
    emails = pb.wrap("contacts", pb.as_list(pb.field(name="email")))

    occupations = pb.wrap("occupations", pb.lst(pb.nested(Occupation)), ns="data")

user = pb.from_xml(User, xml, envelope="doc:envelope", ns_map={
    "doc": "http://www.test1.org",
    "data": "http://www.test2.org",
})

print(json.dumps(attr.asdict(user), indent=4))
```

Output:

```
{
    "name": "Alex",
    "surname": "Ivanov",
    "age": 26,
    "phone": "+79204563539",
    "emails": [
        "alex@gmail.com",
        "alex@mail.ru"
    ],
    "occupations": [
        {
            "title": "yandex",
            "address": "Moscow",
            "employees": 8854
        },
        {
            "title": "skbkontur",
            "address": "Yekaterinburg",
            "employees": 7742
        }
    ]
}
```

CHAPTER 3

The API Documentation

3.1 Developer Interface

`paxb` is a library that provides an API for mapping between XML documents and Python objects.

`paxb` library implements the following functionality:

- Deserialize XML documents to Python objects
- Validate deserialized data
- Access and update Python object fields
- Serialize Python objects to XML documents

`paxb` provides an efficient way of mapping between an XML document and a Python object. Using `paxb` developers can write less boilerplate code emphasizing on application domain logic.

Since `paxb` based on attrs library `paxb` and `attrs` API can be mixed together.

3.1.1 Binding

`paxb.`**`attribute`**(*name=None*, *ns=None*, *ns_map=None*, ***kwargs*)

The function maps a class field to an XML attribute. The field name is used as a default attribute name. The default name can be altered using the *name* argument.

Parameters

- **`name`** (*`str`*) – attribute name. If *None* field name will be used
- **`ns`** (*`str`*) – attribute namespace. If *None* empty namespace is used
- **`ns_map`** (*`dict`*) – mapping from a namespace prefix to a full name.
- **`kwargs`** – arguments that will be passed to `attr.ib()`

paxb.**field**(*name=None*, *ns=None*, *ns_map=None*, *idx=None*, ***kwargs*)
The function maps a class field to an XML element. The field name is used as a default element name. The default name can be altered using *name* argument. The *ns* argument defines the namespace of the element.

Internally the decorator adds some metainformation to `attr.ib.metadata`.

Parameters

- **name** (*str*) – element name. If *None* field name will be used
- **ns** (*str*) – element namespace. If *None* the namespace is inherited from the containing model
- **ns_map** (*dict*) – mapping from a namespace prefix to a full name.
- **idx** (*int*) – element index in the xml document. If *None* 1 is used
- **kwargs** – arguments that will be passed to `attr.ib()`

paxb.**nested**(*cls*, *name=None*, *ns=None*, *ns_map=None*, *idx=None*, ***kwargs*)
The function maps a class to an XML element. *nested* is used when a `paxb.model()` decorated class contains another one as a field.

Parameters

- **cls** – nested object class. *cls* must be an instance of `paxb.model()` decorated class
- **name** (*str*) – element name. If *None* model decorator *name* attribute will be used
- **ns** (*str*) – element namespace. If *None* model decorator ns attribute will be used
- **ns_map** (*dict*) – mapping from a namespace prefix to a full name. It is applied to the current model and it's elements and all nested models
- **idx** (*int*) – element index in the xml document. If *None* 1 is used
- **kwargs** – arguments that will be passed to `attr.ib()`

paxb.**as_list**(*wrapped*)
The function maps a class list field to an XML element list. Wrapped element can be field or nested model.

Parameters **wrapped** – list element type

paxb.**wrapper**(*path*, *wrapped*, *ns=None*, *ns_map=None*, *idx=None*)
The function is used to map a class field to an XML element that is contained by a subelement.

Parameters

- **path** (*str*) – full path to the *wrapped* element. Element names are separated by slashes
- **wrapped** – a wrapped element
- **ns** (*str*) – element namespace. If *None* the namespace is inherited from the containing model
- **ns_map** (*dict*) – mapping from a namespace prefix to a full name. It is applied to the current model and it's elements and all nested models
- **idx** (*int*) – element index in the xml document. If *None* 1 is used

paxb.**attr**
Alias for *paxb.attribute()*

paxb.**wrap**
Alias for *paxb.wrapper()*

paxb.**lst**
Alias for *paxb.as_list()*

3.1.2 Serialization/Deserialization

paxb.**from_xml**(*cls*, *xml*, *envelope=None*, *name=None*, *ns=None*, *ns_map=None*, *required=True*)
Deserializes xml string to object of *cls* type. *cls* must be a paxb.model() decorated class.

Parameters

- **cls** – class the deserialized object is instance of
- **xml** (str or xml.etree.ElementTree.ElementTree) – xml string or xml tree to deserialize the object from
- **envelope** (*str*) – root tag where the serializing object will be looked for
- **name** (*str*) – name of the serialized object element. If *None* model decorator *name* argument will be used
- **ns** (*str*) – namespace of the serialized object element. If *None* model decorator *ns* argument will be used
- **ns_map** (*dict*) – mapping from a namespace prefix to a full name
- **required** (*bool*) – is the serialized object element required. If element not found and *required* is True *paxb.exceptions.DeserializationError* will be raised otherwise None is returned

Returns deserialized object

paxb.**to_xml**(*obj*, *envelope=None*, *name=None*, *ns=None*, *ns_map=None*, *encoder=default_encoder*, ***kwargs*)
Serializes a paxb model object to an xml string. Object must be an instance of a paxb.model() decorated class.

Parameters

- **obj** – object to be serialized
- **envelope** (*str*) – root tag name the serialized object element will be added inside. If None object element will be a root
- **name** (*str*) – name of the serialized object element. If *None* model decorator *name* argument will be used
- **ns** (*str*) – namespace of the serialized object element. If *None* model decorator *ns* argument will be used
- **ns_map** (*dict*) – mapping from a namespace prefix to a full name.
- **encoder** – value encoder. If None *paxb.encoder.encode()* is used
- **kwargs** – arguments that will be passed to xml.etree.ElementTree.tostring() method

Returns serialized object xml string

Return type bytes or str

paxb.encoder.**encode**(*value*)
Default paxb value encoder. Encodes attributes or element text data during serialization. Supports str, bytes, datetime.date and datetime.datetime types.

Parameters **value** – value to be encoded

Returns encoded value string

Return type str

3.1.3 Exceptions

Package errors.

exception paxb.exceptions.**BaseError**
Base package exception. All package exception are inherited from it.

exception paxb.exceptions.**DeserializationError**
Deserialization error. Raised when any deserialization error occurs.

exception paxb.exceptions.**SerializationError**
Serialization error. Raised when any serialization error occurs.

CHAPTER 4

Development

4.1 Development

Install pre-commit hooks:

```
$ pre-commit install
```

For more information see pre-commit

You can run code check manually:

```
$ pre-commit run --all-file
```

Python Module Index

p

paxb, 21
paxb.exceptions, 24

Index

A

as_list() (*in module paxb*), 22
attr (*in module paxb*), 22
attribute() (*in module paxb*), 21

B

BaseError, 24

D

DeserializationError, 24

E

encode() (*in module paxb.encoder*), 23

F

field() (*in module paxb*), 21
from_xml() (*in module paxb*), 23

L

lst (*in module paxb*), 22

N

nested() (*in module paxb*), 22

P

paxb (*module*), 21
paxb.exceptions (*module*), 24

S

SerializationError, 24

T

to_xml() (*in module paxb*), 23

W

wrap (*in module paxb*), 22
wrapper() (*in module paxb*), 22