
xpdata-pydantic

Release 22.4

Christodoulos Tsoulloftas

Apr 10, 2022

CONTENTS

1	Install	3
2	Generate Models	5
3	XML Parsing	7
4	Why Pydantic?	9
5	Limitations	11

xpdata is a complete data binding library for python allowing developers to access and use XML and JSON documents as simple objects rather than using DOM.

Now powered by pydantic!

INSTALL

```
$ # Install with cli support  
$ pip install xsdata-pydantic[cli]
```


GENERATE MODELS

```
$ # Generate models
$ xsdata http://rss.cnn.com/rss/edition.rss --output pydantic
Parsing document edition.rss
Analyzer input: 9 main and 0 inner classes
Analyzer output: 9 main and 0 inner classes
Generating package: init
Generating package: generated.rss
```

```
from dataclasses import field
from pydantic.dataclasses import dataclass

@dataclass
class Rss:
    class Meta:
        name = "rss"

    version: Optional[float] = field(
        default=None,
        metadata={
            "type": "Attribute",
        }
    )
    channel: Optional[Channel] = field(
        default=None,
        metadata={
            "type": "Element",
        }
    )

...
```


XML PARSING

```
>>> from xsdata_pydantic.bindings import XmlParser
>>> from urllib.request import urlopen
>>> from generated.rss import Rss
>>>
>>> parser = XmlParser()
>>> with urlopen("http://rss.cnn.com/rss/edition.rss") as rq:
...     result = parser.parse(rq, Rss)
...
>>> result.channel.item[2].title
'A total lack of discipline': Clarissa Ward visits abandoned Russian foxholes"
>>> result.channel.item[2].pub_date
'Fri, 08 Apr 2022 22:56:33 GMT'
>>> result.channel.item[2].link
'https://www.cnn.com/videos/world/2022/04/08/ukraine-chernihiv-visit-ward-pkg-tsr-vpx.cnn
↵'
```

3.1 Changelog: xx.x (xxxx-xx-xx)

- Initial Release

WHY PYDANTIC?

Because you asked for! Pydantic offers out of the box validations and offers a dataclasses compatibility layer that we utilize to bring code generation and xml data binding with xsdata!

LIMITATIONS

Pydantic dataclasses don't behave with nested classes and self referencing types.

Check [issue](#) on github. Until this issue is fixed, you can use the xsdata generator `--unnest-classes` flag, that will move nested classes to the root level.

```
xpdata SOURCE --unnest-classes --output pydantic
```

The plugin is using xsdata's data bindings to parse json/xml, only xsdata's `types` are supported!

5.1 Installation

5.1.1 Install using pip

The recommended method is to use a virtual environment.

```
$ pip install xsdata-pydantic[cli,lxml,soap]
```

Hint:

- Install the cli requirements for the code generator
- Install the soap requirements for the builtin wsdl client
- Install lxml if you want to use one of the lxml handlers/writers instead of the builtin python xml implementations.

In order to use the latest updates you can also install directly from the git repo.

```
$ pip install git+https://github.com/tefra/xpdata-pydantic@master#egg=xpdata-pydantic[cli]
```

5.1.2 Install using conda

```
$ conda install -c conda-forge xpdata-pydantic
```

5.2 Code Generation

All the xpdata cli features are available. You only need to specify **pydantic** as the output format

5.2.1 Example from Schema

```
$ xpdata tests/fixtures/schemas/po.xsd --output pydantic --package tests.fixtures.po.
↳models --structure-style single-package
Parsing schema po.xsd
Compiling schema po.xsd
Builder: 6 main and 1 inner classes
Analyzer input: 6 main and 1 inner classes
Analyzer output: 5 main and 1 inner classes
Generating package: init
Generating package: tests.fixtures.po.models
```

5.2.2 Example with config

```
$ xpdata tests/fixtures/schemas/po.xsd --config tests/fixtures/pydantic.conf.xml
Parsing schema po.xsd
Compiling schema po.xsd
Builder: 6 main and 1 inner classes
Analyzer input: 6 main and 1 inner classes
Analyzer output: 5 main and 1 inner classes
Generating package: init
Generating package: tests.fixtures.po.models
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Config xmlns="http://pypi.org/project/xpdata" version="22.3">
  <Output maxLineLength="79">
    <Package>tests.fixtures.po.models</Package>
    <Format repr="true" eq="true" order="false" unsafeHash="false" frozen="false" slots=
↳"true" kwOnly="true">pydantic</Format>
    <Structure>single-package</Structure>
    <DocstringStyle>reStructuredText</DocstringStyle>
    <RelativeImports>false</RelativeImports>
    <CompoundFields defaultName="choice" forceDefaultName="false">false</CompoundFields>
    <PostponedAnnotations>false</PostponedAnnotations>
```


5.2.3 Generated Models

```

from dataclasses import field
from decimal import Decimal
from pydantic.dataclasses import dataclass
from typing import List, Optional
from xsdata.models.datatype import XmlDate

__NAMESPACE__ = "foo"

@dataclass(slots=True, kw_only=True)
class Items:
    item: List["Items.Item"] = field(
        default_factory=list,
        metadata={
            "type": "Element",
            "namespace": "foo",
        }
    )

@dataclass(slots=True, kw_only=True)
class Item:
    product_name: str = field(
        metadata={
            "name": "productName",
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
    quantity: int = field(
        metadata={
            "type": "Element",
            "namespace": "foo",
            "required": True,
            "max_exclusive": 100,
        }
    )
    usprice: Decimal = field(
        metadata={
            "name": "USPrice",
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
    comment: Optional[str] = field(
        default=None,
        metadata={
            "type": "Element",
            "namespace": "foo",
        }
    )

```

(continues on next page)

```
)
    ship_date: Optional[XmlDate] = field(
        default=None,
        metadata={
            "name": "shipDate",
            "type": "Element",
            "namespace": "foo",
        }
    )
    part_num: str = field(
        metadata={
            "name": "partNum",
            "type": "Attribute",
            "required": True,
            "pattern": r"\d{3}-[A-Z]{2}",
        }
    )

@dataclass(slots=True, kw_only=True)
class Usaddress:
    class Meta:
        name = "USAddress"

    name: str = field(
        metadata={
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
    street: str = field(
        metadata={
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
    city: str = field(
        metadata={
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
    state: str = field(
        metadata={
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
)
```

(continues on next page)

(continued from previous page)

```
zip: Decimal = field(
    metadata={
        "type": "Element",
        "namespace": "foo",
        "required": True,
    }
)
country: str = field(
    init=False,
    default="US",
    metadata={
        "type": "Attribute",
    }
)

@dataclass(slots=True, kw_only=True)
class Comment:
    class Meta:
        name = "comment"
        namespace = "foo"

    value: str = field(
        default="",
        metadata={
            "required": True,
        }
    )

@dataclass(slots=True, kw_only=True)
class PurchaseOrderType:
    ship_to: Usaddress = field(
        metadata={
            "name": "shipTo",
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
    bill_to: Usaddress = field(
        metadata={
            "name": "billTo",
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
    comment: Optional[str] = field(
        default=None,
        metadata={
            "type": "Element",
```

(continues on next page)

```

        "namespace": "foo",
    }
)
items: Items = field(
    metadata={
        "type": "Element",
        "namespace": "foo",
        "required": True,
    }
)
order_date: Optional[XmlDate] = field(
    default=None,
    metadata={
        "name": "orderDate",
        "type": "Attribute",
    }
)
)

@dataclass(slots=True, kw_only=True)
class PurchaseOrder(PurchaseOrderType):
    class Meta:
        name = "purchaseOrder"
        namespace = "foo"

```

5.3 Data Bindings

All the xsdata `XML` and `JSON` bindings are available. There is an extra requirement to specify the class type of the data models to the `XmlContext` that among other stuff also acts as a compatibility layer between `dataclasses` and pydantic dataclasses

Warning: The plugin is using xsdata's data bindings to parse json/xml, only xsdata's `types` are supported!

5.3.1 Specify ClassType

```

>>> from xsdata.formats.dataclass.parsers import XmlParser
>>> from xsdata.formats.dataclass.parsers import JsonParser
>>> from xsdata.formats.dataclass.serializers import XmlSerializer
>>> from xsdata.formats.dataclass.serializers import JsonSerializer
>>> from xsdata.formats.dataclass.context import XmlContext
...
>>> context = XmlContext(class_type="pydantic") # Specify class type pydantic
>>> xml_parser = XmlParser(context=context)
>>> xml_serializer = XmlSerializer(context=context)
>>> json_parser = JsonParser(context=context)
>>> json_serializer = JsonSerializer(context=context)

```

5.3.2 Binding Shortcuts

For convenience this plugin comes with subclasses for all the xodata binding modules with the pydantic context auto initialized.

```
>>> from xodata_pydantic.bindings import XmlContext
>>> from xodata_pydantic.bindings import XmlParser
>>> from xodata_pydantic.bindings import XmlSerializer
>>> from xodata_pydantic.bindings import JsonParser
>>> from xodata_pydantic.bindings import JsonSerializer
>>> from xodata_pydantic.bindings import UserXmlParser
```

5.4 Changelog

5.4.1 21.11 (2021-11-xx)

- Initial Release