xsdata-attrs

Release 21.8

Christodoulos Tsoulloftas

Aug 03, 2021

CONTENTS

1 Install 3

2 Generate Models 5

3 XML Parsing 7

4 Why attrs? 9

xsData 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 attrs!

CHAPTER

ONE

INSTALL

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

CHAPTER

TWO

GENERATE MODELS

```
$ # Generate models
$ xsdata http://rss.cnn.com/rss/edition.rss --output attrs
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
```

```
...

@attr.s
class Rss:
    class Meta:
        name = "rss"

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

...
```

CHAPTER

THREE

XML PARSING

```
>>> from xsdata_attrs.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
'Vatican indicts 10 people, including a Cardinal, over an international financial scandal
↪'
>>> result.channel.item[2].pub_date
'Sat, 03 Jul 2021 16:37:14 GMT'
>>> result.channel.item[2].link
'https://www.cnn.com/2021/07/03/europe/vatican-financial-scandal-intl/index.html'
```

CHAPTER

FOUR

WHY ATTRS?

Attrs is the original python data class without boilerplate with a huge following. It comes with features not found in the stdlib dataclasses, like keyword only arguments and slotted classes, that won't be available until Python 3.10

It's so similar to dataclasses that also makes it the perfect candidate to test the xsdata's plugin hooks.

4.1 Installation

4.1.1 Install using pip

The recommended method is to use a virtual environment.

```
$ pip install xsdata-attrs[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/xsdata-attrs@master#egg=xsdata-attrs[cli]
```

4.1.2 Install using conda

```
$ conda install -c conda-forge xsdata-attrs
```

4.2 Code Generation

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

4.2.1 Example from Schema

```
$ xsdata tests/fixtures/schemas/po.xsd --output attrs --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
```

4.2.2 Example with config

```
$ xsdata tests/fixtures/schemas/po.xsd --config tests/fixtures/attrs.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/xsdata" version="21.8">
  <Output maxLineLength="79">
    <Package>tests.fixtures.po.models</Package>
    <Format repr="true" eq="true" order="false" unsafeHash="false" frozen="false" slots=
↪"true" kwOnly="true">attrs</Format>
    <Structure>single-package</Structure>
    <DocstringStyle>reStructuredText</DocstringStyle>
    <RelativeImports>false</RelativeImports>
    <CompoundFields>false</CompoundFields>
  </Output>
```

4.2.3 Generated Models

```
import attr
from decimal import Decimal
from typing import List, Optional
from xsdata.models.datatype import XmlDate

__NAMESPACE__ = "foo"
```

(continues on next page)

(continued from previous page)

```
@attr.s(slots=True, kw_only=True)
class Items:
    item: List["Items.Item"] = attr.ib(
        factory=list,
        metadata={
            "type": "Element",
            "namespace": "foo",
        }
    )

    @attr.s(slots=True, kw_only=True)
    class Item:
        product_name: str = attr.ib(
            metadata={
                "name": "productName",
                "type": "Element",
                "namespace": "foo",
                "required": True,
            }
        )
        quantity: int = attr.ib(
            metadata={
                "type": "Element",
                "namespace": "foo",
                "required": True,
                "max_exclusive": 100,
            }
        )
        usprice: Decimal = attr.ib(
            metadata={
                "name": "USPrice",
                "type": "Element",
                "namespace": "foo",
                "required": True,
            }
        )
        comment: Optional[str] = attr.ib(
            default=None,
            metadata={
                "type": "Element",
                "namespace": "foo",
            }
        )
        ship_date: Optional[XmlDate] = attr.ib(
            default=None,
            metadata={
                "name": "shipDate",
                "type": "Element",
                "namespace": "foo",
            }
        )
```

(continues on next page)

(continued from previous page)

```
        part_num: str = attr.ib(
            metadata={
                "name": "partNum",
                "type": "Attribute",
                "required": True,
                "pattern": r"\d{3}-[A-Z]{2}",
            }
        )


@attr.s(slots=True, kw_only=True)
class Usaddress:
    class Meta:
        name = "USAddress"

    name: str = attr.ib(
        metadata={
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
    street: str = attr.ib(
        metadata={
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
    city: str = attr.ib(
        metadata={
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
    state: str = attr.ib(
        metadata={
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
    zip: Decimal = attr.ib(
        metadata={
            "type": "Element",
            "namespace": "foo",
            "required": True,
        }
    )
    country: str = attr.ib(
        init=False,
```

(continues on next page)

(continued from previous page)

```
        default="US",
        metadata={
            "type": "Attribute",
        }
    )


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

    value: str = attr.ib(
        default="",
        metadata={
            "required": True,
        }
    )


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

(continues on next page)

(continued from previous page)

```
    )
    order_date: Optional[XmlDate] = attr.ib(
        default=None,
        metadata={
            "name": "orderDate",
            "type": "Attribute",
        }
    )


@attr.s(slots=True, kw_only=True)
class PurchaseOrder(PurchaseOrderType):
    class Meta:
        name = "purchaseOrder"
        namespace = "foo"
```

4.3 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 attrs.

4.3.1 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="attrs")  # Specify class type attrs
>>> xml_parser = XmlParser(context=context)
>>> xml_serializer = XmlSerializer(context=context)
>>> json_parser = JsonParser(context=context)
>>> json_serializer = JsonSerializer(context=context)
```

4.3.2 Binding Shortcuts

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

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

4.4 Changelog

4.4.1 21.8 (2021-08-03)

- Initial Release