inverter

Release 0.1.0

Izhar Firdaus

Dec 24, 2020

CONTENTS:

CHAPTER

ONE

INVERTER: CONVERT DATACLASS TO ANOTHER CLASS

`inverter` is a library that help convert D(ata)C(lass) to A(nother) C(lass). Supported output classes / schemas are:

- colander schema model.
- Apache Avro schema JSON.
- JSON Schema through Python JSL library model.
- SQLALchemy model (current converter outputs primarily PostgreSQL compatible model).

Note: This library was originally part of `morpfw` project, and might still have some coupling leftover to morpfw.

CHAPTER

TWO

QUICKSTART

2.1 Installation

```
pip install inverter
```

2.2 Creating Schema

Following example creates `colander` schema from a `dataclass`

```
from dataclasses import dataclass, field
import typing
from deform.widget import PasswordWidget
from inverter import dc2colander
import colander

@dataclass
class LoginForm(object):

    username: typing.Optional[str] = field(metadata={'required': True})
    password: typing.Optional[str] = field(
                    metadata={'required': True,
                              'deform.widget': PasswordWidget()})


request = {}
cschema = dc2colander.convert(LoginForm, request)

assert issubclass(cschema, colander.Schema)
```

The same schema can also be converted to Avro Schema

```
from inverter import dc2avsc

avsc = dc2avsc.convert(LoginForm, request, namespace='myapp')

assert avsc == {
    'namespace': 'myapp',
    'type': 'record',
    'name': 'LoginForm',
    'fields': [
```

(continues on next page)

(continued from previous page)

```
        {'name': 'username', 'type': ['string', 'null']},
        {'name': 'password', 'type': ['string', 'null']}]}
```

CHAPTER

THREE

COLANDER CONVERTERS

`inverter` provide several converters to `colander` based schema:

- `dc2colander` - converts `dataclass` to standard `colander` schema.
- `dc2colanderjson` - converts `dataclass` to a `colander` schema that serialize to JSON
  - date is serialized as number days from epoch
  - datetime is serialized as number of miliseconds from epoch
- `dc2colanderavro` - converts `dataclass` to a `colander` schema that serialize to `avro` compatible data
  - date is serialized as number days from epoch
  - datetime is serialized as number of miliseconds from epoch
  - dictionary (JSON field) is serialized as JSON string
- `dc2colanderESjson` - converts `dataclass` to a `colander` schema that serialize to ES compatible JSON
  - date is serialized as YYYY-MM-DD string
  - datetime is serialized as iso8601 string

`inverter.dc2colander.`**`convert`**(*schema: type*, ***, *request: Optional[Any] = None*, *mode='default'*, *include_fields: Optional[List[str]] = None*, *exclude_fields: Optional[List[str]] = None*, *hidden_fields: Optional[List[str]] = None*, *readonly_fields: Optional[List[str]] = None*, *include_schema_validators: bool = True*, *colander_schema_type: Type[colander.Schema] = <class 'colander.Schema'>*, *oid_prefix: str = 'deformField'*, *default_tzinfo=<UTC>*, *field_metadata=None*, *dataclass_field_to_colander_schemanode=<function dataclass_field_to_colander_schemanode>*) → Type[colander.Schema]

Converts `dataclass` to `colander.Schema`

Parameters

- **`schema`** – `dataclass` class to be used as schema
- **`request`** – a request object. This is mainly be passed down to downstream factories, accepts anything.
- **`mode`** – this flag is used to affect the decision on how validators validate data. accepts one of the following: - `'default'` - `'edit'` - flag used when generating edit form - `'edit-process'` - flag used when generating edit processing form
- **`include_fields`** (*`typing.List[str]`*) – List of field names to include
- **`exclude_fields`** (*`typing.List[str]`*) – List of field names to exclude

- **hidden_fields** (*typing.List[str]*) – List of field names to hide
- **readonly_fields** (*typing.List[str]*) – List of field names to made readonly
- **include_schema_validators** (*bool*) – Set whether to include `__validators__` from `dataclass` during convertion
- **colander_schema_type** – base class to use as created output, defaults to colander.MappingSchema.
- **oid_prefix** – string to use as deform OID prefix
- **default_tzinfo** – default timezone for `datetime` handling, defaults to `pytz.UTC`
- **field_metadata** – a dictionary for overriding field metadata. Structure: `{'<fieldname>': {'metadatakey': 'metadataval'}}`
- **dataclass_field_to_colander_schemanode** – `colander.SchemaNode` factory function.

Returns `colander.Schema` class

Field metadata handling

This function will read several metadata from fields and use it to derive the parameters.

- `required: bool` - flag field as required
- `title: str` - field title
- `description: str` - field description
- `validators: typing.List[typing.Callable]` - a list of validator callables
- `preparers: typing.List[typing.Callable]` - a list of preparer callables
- `deform.widget: deform.widget.Widget - deform.widget.Widget` object to use as widget.
- `deform.widget_factory: typing.Callable` - a callable with that accept `request` and returns a `deform.widget.Widget` object.
- `colander.field_factory: typing.Callable` - a callable that accept `request` and returns a `colander.SchemaType` object.

inverter.dc2colanderjson.**convert**(*schema*, ***, *include_fields: Optional[List[str]] = None*, *exclude_fields: Optional[List[str]] = None*, *hidden_fields: Optional[List[str]] = None*, *readonly_fields: Optional[List[str]] = None*, *include_schema_validators: bool = True*, *colander_schema_type: Type[colander.Schema] = <class 'colander.Schema'>*, *oid_prefix: str = 'deformField'*, *request=None*, *mode='default'*, *default_tzinfo=<UTC>*, *field_metadata=None*) → Type[colander.Schema]

Converts `dataclass` to `colander.Schema` that serializes to JSON.

- date is serialized as number days from epoch
- datetime is serialized as number of miliseconds from epoch

Accepted parameters are the same as `inverter.dc2colander.convert`.

inverter.dc2colanderavro.**convert**(*schema*, *, *include_fields: Optional[List[str]] = None*, *exclude_fields: Optional[List[str]] = None*, *hidden_fields: Optional[List[str]] = None*, *readonly_fields: Optional[List[str]] = None*, *include_schema_validators: bool = True*, *colander_schema_type: Type[colander.Schema] = <class 'colander.Schema'>*, *oid_prefix: str = 'deformField'*, *request=None*, *mode='default'*, *default_tzinfo=None*, *field_metadata=None*) → Type[colander.Schema]

Converts `dataclass` to `colander.Schema` that serializes to Avro compatible dictionary.

- date is serialized as number days from epoch
- datetime is serialized as number of miliseconds from epoch
- dictionary (JSON field) is serialized as JSON string

Accepted parameters are the same as `inverter.dc2colander.convert`.

inverter.dc2colanderESjson.**convert**(*schema*, *, *include_fields: Optional[List[str]] = None*, *exclude_fields: Optional[List[str]] = None*, *hidden_fields: Optional[List[str]] = None*, *readonly_fields: Optional[List[str]] = None*, *include_schema_validators: bool = True*, *colander_schema_type: Type[colander.Schema] = <class 'colander.Schema'>*, *oid_prefix: str = 'deformField'*, *request=None*, *default_tzinfo=<UTC>*, *mode='default'*, *field_metadata=None*) → Type[colander.Schema]

Converts `dataclass` to `colander.Schema` that serializes to ElasticSearch compatible dictionary.

- date is serialized as YYYY-MM-DD string
- datetime is serialized as iso8601 string

Accepted parameters are the same as `inverter.dc2colander.convert`.

CHAPTER

FOUR

AVRO SCHEMA CONVERTER

`inverter` provides converter from `dataclass` to Avro Schema .

`inverter.dc2avsc.`**`convert`**(*schema*, ***, *request=None*, *include_fields: Optional[List[str]] = None*, *exclude_fields: Optional[List[str]] = None*, *namespace='inverter'*, *ignore_required=True*)

Converts `dataclass` to Avro Schema JSON dictionary

Parameters

- **`schema`** – `dataclass` class
- **`request`** – request object, accepts Any
- **`include_fields`** (*`typing.List[str]`*) – List of field names to include
- **`exclude_fields`** (*`typing.List[str]`*) – List of field names to exclude
- **`namespace`** – Avro schema namespace, defaults to 'inverter'
- **`ignore_required`** (*`bool`*) – if True, force all fields to be non-required

Returns dictionary representing Avro Schema.

CHAPTER

FIVE

JSON SCHEMA CONVERTER

`inverter` provides converter from `dataclass` to JSON Schema through Python JSL library

`inverter.dc2jsl.`**`convert`**(*schema*, ***, *ignore_required=False*, *additional_properties=False*, *mode='default'*)

Convert `dataclass` to `jsl` JSON Schema.

Parameters

- **`schema`** – `dataclass` class
- **`ignore_required`** – if `True`, set all fields as nullable
- **`additional_properties`** – Allow `additional_properties` in JSON Schema
- **`mode`** – mode flag

Returns `jsl.Document` class

CHAPTER

SIX

SQLALCHEMY CONVERTER

`inverter` provides converter from `dataclass` to SQLAlchemy ORM model.

Currently, only PostgreSQL compatible model is can be generated.

`inverter.dc2pgsqla.`**`convert`**(*schema*, *metadata*, *, *name=None*) → sqlalchemy.sql.schema.Table

Convert `dataclass` to `sqlalchemy` ORM model

Parameters

- **`schema`** – `dataclass` class
- **`metadata`** – `sqlalchemy.MetaData` object
- **`name`** – model name

Returns `sqlalchemy` ORM class

Field metadata handling

- `primary_key: bool` - primary key flag
- `index: bool` - flag on whether to index the column
- `autoincrement: bool` - flag on whether to make column autoincrement
- `unique: bool` - flag on whether to make column unique
- `searchable: bool` - flag on whether to make column searchable using PGSQL Trigram
- `format: str` - format of data: `uuid`, `text`, `fulltextindex`, `bigint`, `numeric`. This forces SQLAlchemy to use specific data type for the format.

INDEX

C