marshmallow-sqlalchemy

Release 1.5.0

Author name not set

Jun 22, 2026

CONTENTS

SQLAlchemy integration with the marshmallow (de)serialization library.

Release v1.5.0 (*Changelog*)

CHAPTER

ONE

DECLARE YOUR MODELS

SQLAlchemy 1.4

```python
import sqlalchemy as sa
from sqlalchemy.orm import (
    DeclarativeBase,
    backref,
    relationship,
    sessionmaker,
)

from marshmallow_sqlalchemy import SQLAlchemySchema, auto_field

engine = sa.create_engine("sqlite:///:memory:")
Session = sessionmaker(engine)


class Base(DeclarativeBase):
    pass


class Author(Base):
    __tablename__ = "authors"
    id = sa.Column(sa.Integer, primary_key=True)
    name = sa.Column(sa.String, nullable=False)

    def __repr__(self):
        return f"<Author(name={self.name!r})>"


class Book(Base):
    __tablename__ = "books"
    id = sa.Column(sa.Integer, primary_key=True)
    title = sa.Column(sa.String)
    author_id = sa.Column(sa.Integer, sa.ForeignKey("authors.id"))
    author = relationship("Author", backref=backref("books"))
```

SQLAlchemy 2

```
import sqlalchemy as sa
from sqlalchemy.orm import (
    DeclarativeBase,
    backref,
    relationship,
    sessionmaker,
    mapped_column,
    Mapped,
)

from marshmallow_sqlalchemy import SQLAlchemySchema, auto_field

engine = sa.create_engine("sqlite:///:memory:")
Session = sessionmaker(engine)


class Base(DeclarativeBase):
    pass


class Author(Base):
    __tablename__ = "authors"
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(nullable=False)

    def __repr__(self):
        return f"<Author(name={self.name!r})>"


class Book(Base):
    __tablename__ = "books"
    id: Mapped[int] = mapped_column(primary_key=True)
    title: Mapped[str] = mapped_column()
    author_id: Mapped[int] = mapped_column(sa.ForeignKey("authors.id"))
    author: Mapped["Author"] = relationship("Author", backref=backref("books"))
```

CHAPTER

TWO

GENERATE MARSHMALLOW SCHEMAS

```
from marshmallow_sqlalchemy import SQLAlchemySchema, auto_field


class AuthorSchema(SQLAlchemySchema):
    class Meta:
        model = Author
        load_instance = True  # Optional: deserialize to model instances

    id = auto_field()
    name = auto_field()
    books = auto_field()


class BookSchema(SQLAlchemySchema):
    class Meta:
        model = Book
        load_instance = True

    id = auto_field()
    title = auto_field()
    author_id = auto_field()
```

You can automatically generate fields for a model's columns using `SQLAlchemyAutoSchema`. The following schema classes are equivalent to the above.

```
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema


class AuthorSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Author
        include_relationships = True
        load_instance = True


class BookSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Book
        include_fk = True
        load_instance = True
```

Make sure to declare `Models` before instantiating `Schemas`. Otherwise sqlalchemy.orm.configure_mappers() will run too soon and fail.

CHAPTER

THREE

(DE)SERIALIZE YOUR DATA

```
author = Author(name="Chuck Paluhniuk")
author_schema = AuthorSchema()
book = Book(title="Fight Club", author=author)

with Session() as session:
    session.add(author)
    session.add(book)
    session.commit()

    dump_data = author_schema.dump(author)
    print(dump_data)
    # {'id': 1, 'name': 'Chuck Paluhniuk', 'books': [1]}

with Session() as session:
    load_data = author_schema.load(dump_data, session=session)
    print(load_data)
    # <Author(name='Chuck Paluhniuk')>
```

CHAPTER

FOUR

GET IT NOW

```
$ pip install -U marshmallow-sqlalchemy
```

CHAPTER

FIVE

LEARN

5.1 Recipes

5.1.1 Base `Schema` I

A common pattern with marshmallow is to define a base `Schema` class which has common configuration and behavior for your application's schemas.

You may want to define a common session object, e.g. a `scoped_session` to use for all `Schemas`.

```
# myproject/db.py
import sqlalchemy as sa
from sqlalchemy import orm

Session = orm.scoped_session(orm.sessionmaker())
Session.configure(bind=engine)
```

```
# myproject/schemas.py

from marshmallow_sqlalchemy import SQLAlchemySchema

from .db import Session


class BaseSchema(SQLAlchemySchema):
    class Meta:
        sqla_session = Session
```

```
# myproject/users/schemas.py

from ..schemas import BaseSchema
from .models import User


class UserSchema(BaseSchema):
    # Inherit BaseSchema's options
    class Meta(BaseSchema.Meta):
        model = User
```

5.1.2 Base Schema II

Here is an alternative way to define a BaseSchema class with a common `Session` object.

```
# myproject/schemas.py

from marshmallow_sqlalchemy import SQLAlchemySchemaOpts, SQLAlchemySchema
from .db import Session


class BaseOpts(SQLAlchemySchemaOpts):
    def __init__(self, meta, ordered=False):
        if not hasattr(meta, "sqla_session"):
            meta.sqla_session = Session
        super(BaseOpts, self).__init__(meta, ordered=ordered)


class BaseSchema(SQLAlchemySchema):
    OPTIONS_CLASS = BaseOpts
```

This allows you to define class Meta options without having to subclass `BaseSchema.Meta`.

```
# myproject/users/schemas.py

from ..schemas import BaseSchema
from .models import User


class UserSchema(BaseSchema):
    class Meta:
        model = User
```

5.1.3 Using *Related* to serialize relationships

The *Related* field can be used to serialize a SQLAlchemy `relationship` as a nested dictionary.

```
import sqlalchemy as sa
from sqlalchemy.orm import DeclarativeBase, relationship

from marshmallow_sqlalchemy import SQLAlchemyAutoSchema, auto_field
from marshmallow_sqlalchemy.fields import Related


class Base(DeclarativeBase):
    pass


class User(Base):
    __tablename__ = "user"
    id = sa.Column(sa.Integer, primary_key=True)
    full_name = sa.Column(sa.String(255))


class BlogPost(Base):
```

(continues on next page)

(continued from previous page)

```
    __tablename__ = "blog_post"
    id = sa.Column(sa.Integer, primary_key=True)
    title = sa.Column(sa.String(255), nullable=False)

    author_id = sa.Column(sa.Integer, sa.ForeignKey(User.id), nullable=False)
    author = relationship(User)


class BlogPostSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = BlogPost

    id = auto_field()
    # Blog's author will be serialized as a dictionary with
    # `id` and `name` pulled from the related User.
    author = Related(["id", "full_name"])
```

Serialization will look like this:

```
from pprint import pprint

from sqlalchemy.orm import sessionmaker

engine = sa.create_engine("sqlite:///:memory:")
Session = sessionmaker(engine)

Base.metadata.create_all(engine)

with Session() as session:
    user = User(full_name="Freddie Mercury")
    post = BlogPost(title="Bohemian Rhapsody Revisited", author=user)

    session.add_all([user, post])
    session.commit()

    blog_post_schema = BlogPostSchema()
    data = blog_post_schema.dump(post)
    pprint(data, indent=2)
    # { 'author': {'full_name': 'Freddie Mercury', 'id': 1},
    #   'id': 1,
    #   'title': 'Bohemian Rhapsody Revisited'}
```

5.1.4 Introspecting generated fields

It is often useful to introspect what fields are generated for a *SQLAlchemyAutoSchema*.

Generated fields are added to a `Schema's _declared_fields` attribute.

```
AuthorSchema._declared_fields["books"]
# <fields.RelatedList(default=<marshmallow.missing>, ...>
```

You can also use marshmallow-sqlalchemy's conversion functions directly.

```
from marshmallow_sqlalchemy import property2field

id_prop = Author.__mapper__.attrs.get("id")

property2field(id_prop)
# <fields.Integer(default=<marshmallow.missing>, ...>
```

5.1.5 Overriding generated fields

Any field generated by a *SQLAlchemyAutoSchema* can be overridden.

```
from marshmallow import fields
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema
from marshmallow_sqlalchemy.fields import Nested


class AuthorSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Author

    # Override books field to use a nested representation rather than pks
    books = Nested(BookSchema, many=True, exclude=("author",))
```

You can use the *auto_field* function to generate a marshmallow `Field` based on single model property. This is useful for passing additional keyword arguments to the generated field.

```
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema, field_for


class AuthorSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Author

    # Generate a field, passing in an additional dump_only argument
    date_created = auto_field(dump_only=True)
```

If a field's external data key differs from the model's column name, you can pass a column name to *auto_field*.

```
class AuthorSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Author
        # Exclude date_created because we're aliasing it below
        exclude = ("date_created",)

    # Generate "created_date" field from "date_created" column
    created_date = auto_field("date_created", dump_only=True)
```

5.1.6 Automatically generating schemas for SQLAlchemy models

It can be tedious to implement a large number of schemas if not overriding any of the generated fields as detailed above. SQLAlchemy has a hook that can be used to trigger the creation of the schemas, assigning them to `Model.__marshmallow__`.

```
from marshmallow_sqlalchemy import ModelConversionError, SQLAlchemyAutoSchema


def setup_schema(Base, session):
    # Create a function which incorporates the Base and session information
    def setup_schema_fn():
        for class_ in Base._decl_class_registry.values():
            if hasattr(class_, "__tablename__"):
                if class_.__name__.endswith("Schema"):
                    raise ModelConversionError(
                        "For safety, setup_schema can not be used when a"
                        "Model class ends with 'Schema'"
                    )

                class Meta(object):
                    model = class_
                    sqla_session = session

                schema_class_name = "%sSchema" % class_.__name__

                schema_class = type(
                    schema_class_name, (SQLAlchemyAutoSchema,), {"Meta": Meta}
                )

                setattr(class_, "__marshmallow__", schema_class)

    return setup_schema_fn
```

Usage:

```
import sqlalchemy as sa
from sqlalchemy.orm import declarative_base, sessionmaker
from sqlalchemy import event
from sqlalchemy.orm import mapper

# Either import or declare setup_schema here

engine = sa.create_engine("sqlite:///:memory:")
Session = sessionmaker(engine)
Base = declarative_base()


class Author(Base):
    __tablename__ = "authors"
    id = sa.Column(sa.Integer, primary_key=True)
    name = sa.Column(sa.String)

    def __repr__(self):
        return "<Author(name={self.name!r})>".format(self=self)


# Listen for the SQLAlchemy event and run setup_schema.
# Note: This has to be done after Base and session are setup
```

(continues on next page)

(continued from previous page)

```
event.listen(mapper, "after_configured", setup_schema(Base, session))

Base.metadata.create_all(engine)

with Session() as session:
    author = Author(name="Chuck Paluhniuk")
    session.add(author)
    session.commit()

    # Model.__marshmallow__ returns the Class not an instance of the schema
    # so remember to instantiate it
    author_schema = Author.__marshmallow__()

    print(author_schema.dump(author))
```

This is inspired by functionality from ColanderAlchemy.

5.1.7 Smart `Nested` field

To serialize nested attributes to primary keys unless they are already loaded, you can use this custom field.

```
from marshmallow_sqlalchemy.fields import Nested


class SmartNested(Nested):
    def serialize(self, attr, obj, accessor=None):
        if attr not in obj.__dict__:
            return {"id": int(getattr(obj, attr + "_id"))}
        return super().serialize(attr, obj, accessor)
```

An example of then using this:

```
from marshmallow_sqlalchemy import SQLAlchemySchema, auto_field


class BookSchema(SQLAlchemySchema):
    id = auto_field()
    author = SmartNested(AuthorSchema)

    class Meta:
        model = Book
        sqla_session = Session


book = Book(id=1)
book.author = Author(name="Chuck Paluhniuk")
session.add(book)
session.commit()

book = Book.query.get(1)
print(BookSchema().dump(book)["author"])
# {'id': 1}
```

(continues on next page)

(continued from previous page)

```
book = Book.query.options(joinedload("author")).get(1)
print(BookSchema().dump(book)["author"])
# {'id': 1, 'name': 'Chuck Paluhniuk'}
```

5.1.8 Transient object creation

Sometimes it might be desirable to deserialize instances that are transient (not attached to a session). In these cases you can specify the `transient` option in the *Meta* class of a *SQLAlchemySchema*.

```
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema


class AuthorSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Author
        load_instance = True
        transient = True


dump_data = {"id": 1, "name": "John Steinbeck"}
print(AuthorSchema().load(dump_data))
# <Author(name='John Steinbeck')>
```

You may also explicitly specify an override by passing the same argument to *load*.

```
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema


class AuthorSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Author
        sqla_session = Session
        load_instance = True


dump_data = {"id": 1, "name": "John Steinbeck"}
print(AuthorSchema().load(dump_data, transient=True))
# <Author(name='John Steinbeck')>
```

Note that transience propagates to relationships (i.e. auto-generated schemas for nested items will also be transient).

See also

See State Management to understand session state management.

5.1.9 Controlling instance loading

You can override the schema `load_instance` flag by passing in a `load_instance` argument when creating the schema instance. Use this to switch between loading to a dictionary or to a model instance:

```
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema


class AuthorSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = Author
        sqla_session = Session
        load_instance = True


dump_data = {"id": 1, "name": "John Steinbeck"}
print(AuthorSchema().load(dump_data))  # loading an instance
# <Author(name='John Steinbeck')>
print(AuthorSchema(load_instance=False).load(dump_data))  # loading a dict
# {"id": 1, "name": "John Steinbeck"}
```

CHAPTER

SIX

API REFERENCE

6.1 Top-level API

class marshmallow_sqlalchemy.**SQLAlchemySchema**(**args*, ***kwargs*)

Schema for a SQLAlchemy model or table. Use together with *auto_field* to generate fields from columns.

Example:

```
from marshmallow_sqlalchemy import SQLAlchemySchema, auto_field

from mymodels import User


class UserSchema(SQLAlchemySchema):
    class Meta:
        model = User

    id = auto_field()
    created_at = auto_field(dump_only=True)
    name = auto_field()
```

get_instance(*data*)

Retrieve an existing record by primary key(s). If the schema instance is transient, return None.

Parameters
data – Serialized data to inform lookup.

Return type
_ModelType | None

load(*data*, ** (Keyword-only parameters separator (PEP 3102))*, *session=None*, *instance=None*, *transient=False*, ***kwargs*)

Deserialize data. If `load_instance` is set to `True` in the schema meta options, load the data as model instance(s).

Parameters

- **data** (*_LoadDataT*) – The data to deserialize.
- **session** (*Session | None*) – SQLAlchemy `session`.
- **instance** (*_ModelType | None*) – Existing model instance to modify.
- **transient** (*bool*) – If `True`, load transient model instance(s).
- **kwargs** – Same keyword arguments as `marshmallow.Schema.load`.

Return type
Any

make_instance(*data*, ***kwargs*)

Deserialize data to an instance of the model if self.load_instance is True.

Update an existing row if specified in `self.instance` or loaded by primary key(s) in the data; else create a new row.

Parameters
data – Data to deserialize.

Return type
_ModelType

property session: Session | None

The SQLAlchemy session used to load models.

property transient: bool

Whether model instances are loaded in a transient state.

validate(*data*, *, *session=None*, ***kwargs*)

Same as `marshmallow.Schema.validate` but allows passing a `session`.

Parameters

- **data** (*_LoadDataT*)
- **session** (*Session | None*)

Return type
dict[str, list[str]]

class marshmallow_sqlalchemy.**SQLAlchemyAutoSchema**(**args*, ***kwargs*)

Schema that automatically generates fields from the columns of
a SQLAlchemy model or table.

Example:

```
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema, auto_field

from mymodels import User


class UserSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = User
        # OR
        # table = User.__table__

    created_at = auto_field(dump_only=True)
```

get_instance(*data*)

Retrieve an existing record by primary key(s). If the schema instance is transient, return `None`.

Parameters
data – Serialized data to inform lookup.

Return type
_ModelType | None

load(*data*, *, *session=None*, *instance=None*, *transient=False*, ***kwargs*)

Deserialize data. If `load_instance` is set to `True` in the schema meta options, load the data as model instance(s).

Parameters

- **data** (*_LoadDataT*) – The data to deserialize.
- **session** (*Session | None*) – SQLAlchemy `session`.
- **instance** (*_ModelType | None*) – Existing model instance to modify.
- **transient** (*bool*) – If `True`, load transient model instance(s).
- **kwargs** – Same keyword arguments as `marshmallow.Schema.load`.

Return type
Any

make_instance(*data*, ***kwargs*)

Deserialize data to an instance of the model if self.load_instance is True.

Update an existing row if specified in `self.instance` or loaded by primary key(s) in the data; else create a new row.

Parameters
data – Data to deserialize.

Return type
_ModelType

property session: Session | None

The SQLAlchemy session used to load models.

property transient: bool

Whether model instances are loaded in a transient state.

validate(*data*, *, *session=None*, ***kwargs*)

Same as `marshmallow.Schema.validate` but allows passing a `session`.

Parameters

- **data** (*_LoadDataT*)
- **session** (*Session | None*)

Return type
dict[str, list[str]]

exception marshmallow_sqlalchemy.**ModelConversionError**

Raised when an error occurs in converting a SQLAlchemy construct to a marshmallow object.

class marshmallow_sqlalchemy.**ModelConverter**(*schema_cls=None*)

Converts a SQLAlchemy model into a dictionary of corresponding marshmallow `Fields`.

Parameters
schema_cls (*type[ma.Schema] | None*)

fields_for_model(*model*, *, *include_fk=False*, *include_relationships=False*, *fields=None*, *exclude=None*, *base_fields=None*, *dict_cls=<class 'dict'>*)

Generate a dict of field_name: `marshmallow.fields.Field` pairs for the given model. Note: SynonymProperties are ignored. Use an explicit field if you want to include a synonym.

Parameters

- **model** (*type[DeclarativeMeta]*) – The SQLAlchemy model
- **include_fk** (*bool*) – Whether to include foreign key fields in the output.
- **include_relationships** (*bool*) – Whether to include relationships fields in the output.
- **fields** (*Iterable[str] | None*)
- **exclude** (*Iterable[str] | None*)
- **base_fields** (*dict | None*)
- **dict_cls** (*type[dict]*)

Returns
dict of field_name: Field instance pairs

Return type
dict[str, fields.Field]

property2field(*prop: MapperProperty*, *, *instance: Literal[True] = True*, *field_class: type[Field] | None = None*, ***kwargs*) → Field

property2field(*prop: MapperProperty*, *, *instance: Literal[False] = True*, *field_class: type[Field] | None = None*, ***kwargs*) → type[Field]

Convert a SQLAlchemy `Property` to a field instance or class.

Parameters

- **prop** (*Property*) – SQLAlchemy Property.
- **instance** (*bool*) – If `True`, return `Field` instance, computing relevant kwargs from the given property. If `False`, return the `Field` class.
- **kwargs** – Additional keyword arguments to pass to the field constructor.
- **field_class** (*type[fields.Field] | None*)

Returns
A `marshmallow.fields.Field` class or instance.

Return type
fields.Field | type[fields.Field]

column2field(*column*, *, *instance: Literal[True] = True*, ***kwargs*) → Field

column2field(*column*, *, *instance: Literal[False] = True*, ***kwargs*) → type[Field]

Convert a SQLAlchemy `Column` to a field instance or class.

Parameters

- **column** (*sqlalchemy.schema.Column*) – SQLAlchemy Column.
- **instance** (*bool*) – If `True`, return `Field` instance, computing relevant kwargs from the given property. If `False`, return the `Field` class.

Returns
A `marshmallow.fields.Field` class or instance.

Return type
Field | type[*Field*]

field_for(*model: type[DeclarativeMeta]*, *property_name: str*, *, *instance: Literal[True] = True*, *field_class: type[Field] | None = None*, ***kwargs*) → Field

field_for(*model: type[DeclarativeMeta]*, *property_name: str*, ***, *instance: Literal[False] = True*, *field_class: type[Field] | None = None*, ***kwargs*) → type[Field]

Convert a property for a mapped SQLAlchemy class to a marshmallow `Field`. Example:

```
date_created = field_for(Author, "date_created", dump_only=True)
author = field_for(Book, "author")
```

Parameters

- **model** (*type*) – A SQLAlchemy mapped class.
- **property_name** (*str*) – The name of the property to convert.
- **kwargs** – Extra keyword arguments to pass to *property2field*
- **instance** (*bool*)
- **field_class** (*type[fields.Field] | None*)

Returns
A `marshmallow.fields.Field` class or instance.

Return type
fields.Field | type[fields.Field]

class marshmallow_sqlalchemy.**SQLAlchemyAutoSchemaOpts**(*meta*, **args*, ***kwargs*)

Options class for *SQLAlchemyAutoSchema*. Has the same options as *SQLAlchemySchemaOpts*, with the addition of:

- `include_fk`: Whether to include foreign fields; defaults to `False`.
- `include_relationships`: Whether to include relationships; defaults to `False`.

class marshmallow_sqlalchemy.**SQLAlchemySchemaOpts**(*meta*, **args*, ***kwargs*)

Options class for *SQLAlchemySchema*. Adds the following options:

- `model`: The SQLAlchemy model to generate the `Schema` from (mutually exclusive with `table`).
- `table`: The SQLAlchemy table to generate the `Schema` from (mutually exclusive with `model`).
- `load_instance`: Whether to load model instances.
- **`sqla_session`: SQLAlchemy session to be used for deserialization.**
  This is only needed when `load_instance` is `True`. You can also pass a session to the Schema's `load` method.
- **`transient`: Whether to load model instances in a transient state (effectively ignoring the session).**
  Only relevant when `load_instance` is `True`.
- `model_converter`: *ModelConverter* class to use for converting the SQLAlchemy model to marshmallow fields.

marshmallow_sqlalchemy.**auto_field**(*column_name=None*, ***, *model=None*, *table=None*, ***kwargs*)

Mark a field to autogenerate from a model or table.

Parameters

- **column_name** (*str | None*) – Name of the column to generate the field from. If `None`, matches the field name. If `attribute` is unspecified, `attribute` will be set to the same value as `column_name`.
- **model** (*type[DeclarativeMeta] | None*) – Model to generate the field from. If `None`, uses `model` specified on `class Meta`.

- **table** (*sa.Table | None*) – Table to generate the field from. If `None`, uses `table` specified on `class Meta`.
- **kwargs** – Field argument overrides.

Return type
SQLAlchemyAutoField

marshmallow_sqlalchemy.**column2field**(*column*, *, *instance=True*, ***kwargs*)

Convert a SQLAlchemy `Column` to a field instance or class.

Parameters

- **column** (*sqlalchemy.schema.Column*) – SQLAlchemy Column.
- **instance** (*bool*) – If `True`, return `Field` instance, computing relevant kwargs from the given property. If `False`, return the `Field` class.

Returns
A `marshmallow.fields.Field` class or instance.

Return type
Field | type[*Field*]

marshmallow_sqlalchemy.**field_for**(*model*, *property_name*, *, *instance=True*, *field_class=None*, ***kwargs*)

Convert a property for a mapped SQLAlchemy class to a marshmallow `Field`. Example:

```
date_created = field_for(Author, "date_created", dump_only=True)
author = field_for(Book, "author")
```

Parameters

- **model** (*type*) – A SQLAlchemy mapped class.
- **property_name** (*str*) – The name of the property to convert.
- **kwargs** – Extra keyword arguments to pass to *property2field*
- **instance** (*bool*)
- **field_class** (*type[fields.Field] | None*)

Returns
A `marshmallow.fields.Field` class or instance.

Return type
fields.Field | type[fields.Field]

marshmallow_sqlalchemy.**fields_for_model**(*model*, *, *include_fk=False*, *include_relationships=False*, *fields=None*, *exclude=None*, *base_fields=None*, *dict_cls=<class 'dict'>*)

Generate a dict of field_name: `marshmallow.fields.Field` pairs for the given model. Note: SynonymProperties are ignored. Use an explicit field if you want to include a synonym.

Parameters

- **model** (*type[DeclarativeMeta]*) – The SQLAlchemy model
- **include_fk** (*bool*) – Whether to include foreign key fields in the output.
- **include_relationships** (*bool*) – Whether to include relationships fields in the output.
- **fields** (*Iterable[str] | None*)

- **exclude** (*Iterable[str] | None*)
- **base_fields** (*dict | None*)
- **dict_cls** (*type[dict]*)

Returns
dict of field_name: Field instance pairs

Return type
dict[str, fields.Field]

marshmallow_sqlalchemy.**property2field**(*prop*, *, *instance=True*, *field_class=None*, ***kwargs*)

Convert a SQLAlchemy `Property` to a field instance or class.

Parameters

- **prop** (*Property*) – SQLAlchemy Property.
- **instance** (*bool*) – If `True`, return `Field` instance, computing relevant kwargs from the given property. If `False`, return the `Field` class.
- **kwargs** – Additional keyword arguments to pass to the field constructor.
- **field_class** (*type[fields.Field] | None*)

Returns
A `marshmallow.fields.Field` class or instance.

Return type
fields.Field | type[fields.Field]

6.2 Fields

class marshmallow_sqlalchemy.fields.**RelatedList**(*cls_or_instance*, ***kwargs*)

Parameters

- **cls_or_instance** (*Field[_InternalT] | type[Field[_InternalT]]*)
- **kwargs** (*Unpack[_BaseFieldKwargs]*)

class marshmallow_sqlalchemy.fields.**Related**(*columns=None*, *column=None*, ***kwargs*)

Related data represented by a SQLAlchemy `relationship`. Must be attached to a `Schema` class whose options includes a SQLAlchemy `model`, such as *SQLAlchemySchema*.

Parameters

- **columns** (*list[str] | str | None*) – Optional column names on related model. If not provided, the primary key(s) of the related model will be used.
- **column** (*str | None*)

class marshmallow_sqlalchemy.fields.**Nested**(*nested*, *, *only=None*, *exclude=()*, *many=None*, *unknown=None*, ***kwargs*)

Nested field that inherits the session from its parent.

Parameters

- **nested** (*Schema | SchemaMeta | str | dict[str, Field] | Callable[[], Schema | SchemaMeta | str | dict[str, Field]]*)
- **only** (*types.StrSequenceOrSet | None*)

- **exclude** (*types.StrSequenceOrSet*)
- **many** (*bool | None*)
- **unknown** (*types.UnknownOption | None*)
- **kwargs** (*Unpack[_BaseFieldKwargs]*)

CHAPTER

SEVEN

PROJECT INFO

7.1 Changelog

7.1.1 unreleased

Other changes:

- Drop support for marshmallow 3, which is EOL.

7.1.2 1.5.0 (2026-04-01)

Bug fixes:

- Fix memory usage regression from 1.4.1 (1.4.2 didn't address the root cause) (#665).

Other changes:

- Drop support for Python 3.9, which is EOL. Support Python 3.10-3.14.

7.1.3 1.4.2 (2025-04-09)

Bug fixes:

- Fix memory usage regression in 1.4.1 (#665). Thanks @mistercrunch for reporting and sending a PR.

7.1.4 1.4.1 (2025-02-10)

Bug fixes:

- Fix inheritance of declared fields that match then name of a foreign key column when the `include_fk` option is set to `False` (#657). Thanks @carterjc for the PR.

7.1.5 1.4.0 (2025-01-19)

Bug fixes:

- Fix handling of arrays of enums and multidimensional arrays (#653). Thanks @carterjc for reporting and investigating the fix.
- Fix handling of `sqlalchemy.PickleType` columns (#394) Thanks @Eyon42 for reporting.

Other changes:

- Passing arbitrary keyword arguments to *`auto_field`* is no longer supported (#647). Use the `metadata` argument to pass metadata to the generated field instead.

```
# Before
auto_field(description="The name of the artist")
# On marshmallow 3, this raises a warning: "RemovedInMarshmallow4Warning: Passing field␣
↪metadata as keyword arguments is deprecated."
# On marshmallow 4, this raises an error: "TypeError: Field.__init__() got an unexpected␣
↪keyword argument 'description'"

# After
auto_field(metadata=dict(description="The name of the artist"))
```

7.1.6 1.3.0 (2025-01-11)

Features:

- Typing: Add type annotations to *fields*.

Bug fixes:

- Fix auto-generation of marshmallow.fields.Enum field from sqlalchemy.Enum columns (#615). Thanks @joaquimvl for reporting.
- Fix behavior of include_fk = False in options when parent schema sets include_fk = True (#440). Thanks @uhnomoli for reporting.
- Fields generated from non-nullable sqlalchemy.orm.relationship correctly set required=True and allow_none=False (#336, #163). Thanks @AbdealiLoKo for reporting.

Other changes:

- Docs: Add more documentation for *marshmallow_sqlalchemy.fields.Related* (#162). Thanks @GabrielC101 for the suggestion.
- Docs: Document methods of SQLAlchemySchema and SQLAlchemyAutoSchema (#619).
- Docs: Various documentation improvements (#635, #636, #639, #641, #642).

7.1.7 1.2.0 (2025-01-09)

Features:

- Typing: Improve type coverage (#631, #632, #634).

Other changes:

- Drop support for Python 3.8, which is EOL. Support Python 3.9-3.13.

7.1.8 1.1.1 (2025-01-06)

Bug fixes:

- Fix compatibility with marshmallow 3.24.0 and 4.0.0 (#628).

Other changes:

- Test against Python 3.13 (#629).

7.1.9 1.1.0 (2024-08-14)

Features:

- `sqlalchemy.Enum` fields generate a corresponding `marshmallow.fields.Enum` field (#485, #112). Thanks @panda-byte for the PR.

Support:

- Drop support for marshmallow<3.18.0.

7.1.10 1.0.0 (2024-01-30)

- Remove `__version__` attribute. Use feature detection or `importlib.metadata.version("marshmallow-sqlalchemy")` instead (#568).
- Support marshmallow>=3.10.0 (#566).
- Passing `info={"marshmallow": ...}` to SQLAlchemy columns is removed, as it is redundant with the `auto_field` functionality (#567).
- Remove `packaging` as a dependency (#566).
- Support Python 3.12.

7.1.11 0.30.0 (2024-01-07)

Features:

- Use `Session.get()` load instances to improve deserialization performance (#548). Thanks @zippolyte for the PR.

Other changes:

- Drop support for Python 3.7, which is EOL (#540).

7.1.12 0.29.0 (2023-02-27)

Features:

- Support SQLAlchemy 2.0 (#494). Thanks @dependabot for the PR.
- Enable (in tests) and fix SQLAlchemy 2.0 compatibility warnings (#493).

Bug fixes:

- Use mapper `.attrs` rather than `.get_property` and `.iterate_properties` to ensure `registry.configure` is called (call removed in SQLAlchemy 2.0.2) (#487). Thanks @ddoyon92 for the PR.

Other changes:

- Drop support for SQLAlchemy 1.3, which is EOL (#493).

7.1.13 0.28.2 (2023-02-23)

Bug fixes:

- Use .scalar_subquery() for SQLAlchemy>1.4 to suppress a warning (#459). Thanks @indiVar0508 for the PR.

Other changes:

- Lock SQLAlchemy<2.0 in setup.py. SQLAlchemy 2.x is not supported (#486).
- Test against Python 3.11 (#486).

7.1.14 0.28.1 (2022-07-18)

Bug fixes:

- Address `DeprecationWarning` re: usage of `distutils` (#435). Thanks @Tenzer for the PR.

7.1.15 0.28.0 (2022-03-09)

Features:

- Add support for generating fields from `column_property` (#97). Thanks @mrname for the PR.

Other changes:

- Drop support for Python 3.6, which is EOL.
- Drop support for SQLAlchemy 1.2, which is EOL.

7.1.16 0.27.0 (2021-12-18)

Features:

- Distribute type information per PEP 561 (#420). Thanks @bruceadams for the PR.

Other changes:

- Test against Python 3.10 (#421).

7.1.17 0.26.1 (2021-06-05)

Bug fixes:

- Fix generating fields for `postgreql.ARRAY` columns (#392). Thanks @mjpieters for the catch and patch.

7.1.18 0.26.0 (2021-05-26)

Bug fixes:

- Unwrap proxied columns to handle models for subqueries (#383). Thanks @mjpieters for the catch and patch
- Fix setting `transient` on a per-instance basis when the `transient` Meta option is set (#388). Thanks again @mjpieters.

Other changes:

- *Backwards-incompatible*: Remove deprecated `ModelSchema` and `TableSchema` classes.

7.1.19 0.25.0 (2021-05-02)

- Add `load_instance` as a parameter to `SQLAlchemySchema` and `SQLAlchemyAutoSchema` (#380). Thanks @mjpieters for the PR.

7.1.20 0.24.3 (2021-04-26)

- Fix deprecation warnings from marshmallow 3.10 and SQLAlchemy 1.4 (#369). Thanks @peterschutt for the PR.

7.1.21 0.24.2 (2021-02-07)

- `auto_field` supports `association_proxy` fields with local multiplicity (`uselist=True`) (#364). Thanks @Unix-Code for the catch and patch.

7.1.22 0.24.1 (2020-11-20)

- `auto_field` works with `association_proxy` (#338). Thanks @AbdealiJK.

7.1.23 0.24.0 (2020-10-20)

- *Backwards-incompatible*: Drop support for marshmallow 2.x, which is now EOL.
- Test against Python 3.9.

7.1.24 0.23.1 (2020-05-30)

Bug fixes:

- Don't add no-op `Length` validator (#315). Thanks @taion for the PR.

7.1.25 0.23.0 (2020-04-26)

Bug fixes:

- Fix data keys when using `Related` with a `Column` that is named differently from its attribute (#299). Thanks @peterschutt for the catch and patch.
- Fix bug that raised an exception when using the `ordered = True` option on a schema that has an `auto_field` (#306). Thanks @KwonL for reporting and thanks @peterschutt for the PR.

7.1.26 0.22.3 (2020-03-01)

Bug fixes:

- Fix `DeprecationWarning` getting raised even when user code does not use `TableSchema` or `ModelSchema` (#289). Thanks @5uper5hoot for reporting.

7.1.27 0.22.2 (2020-02-09)

Bug fixes:

- Avoid error when using `SQLAlchemyAutoSchema`, `ModelSchema`, or `fields_for_model` with a model that has a `SynonymProperty` (#190). Thanks @TrilceAC for reporting.
- `auto_field` and `field_for` work with `SynonymProperty` (#280).

Other changes:

- Add hook in `ModelConverter` for changing field names based on SQLA columns and properties (#276). Thanks @davenquinn for the suggestion and the PR.

7.1.28 0.22.1 (2020-02-09)

Bug fixes:

- Fix behavior when passing `table` to `auto_field` (#277).

7.1.29 0.22.0 (2020-02-09)

Features:

- Add `SQLAlchemySchema` and `SQLAlchemyAutoSchema`, which have an improved API for generating marshmallow fields and overriding their arguments via `auto_field` (#240). Thanks @taion for the idea and original implementation.

```
# Before
from marshmallow_sqlalchemy import ModelSchema, field_for

from . import models


class ArtistSchema(ModelSchema):
    class Meta:
        model = models.Artist

    id = field_for(models.Artist, "id", dump_only=True)
    created_at = field_for(models.Artist, "created_at", dump_only=True)


# After
from marshmallow_sqlalchemy import SQLAlchemySchema, auto_field

from . import models


class ArtistSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = models.Artist

    id = auto_field(dump_only=True)
    created_at = auto_field(dump_only=True)
```

- Add `load_instance` option to configure deserialization to model instances (#193, #270).
- Add `include_relationships` option to configure generation of marshmallow fields for relationship properties (#98). Thanks @dusktreader for the suggestion.

Deprecations:

- `ModelSchema` and `TableSchema` are deprecated, since `SQLAlchemyAutoSchema` has equivalent functionality.

```
# Before
from marshmallow_sqlalchemy import ModelSchema, TableSchema

from . import models


class ArtistSchema(ModelSchema):
    class Meta:
        model = models.Artist


class AlbumSchema(TableSchema):
```

(continues on next page)

(continued from previous page)

```
    class Meta:
        table = models.Album.__table__


# After
from marshmallow_sqlalchemy import SQLAlchemyAutoSchema

from . import models


class ArtistSchema(SQLAlchemyAutoSchema):
    class Meta:
        model = models.Artist
        include_relationships = True
        load_instance = True


class AlbumSchema(SQLAlchemyAutoSchema):
    class Meta:
        table = models.Album.__table__
```

- Passing `info={"marshmallow": ...}` to SQLAlchemy columns is deprecated, as it is redundant with the `auto_field` functionality.

Other changes:

- *Backwards-incompatible*: `fields_for_model` does not include relationships by default. Use `fields_for_model(..., include_relationships=True)` to preserve the old behavior.

7.1.30 0.21.0 (2019-12-04)

- Add support for `postgresql.OID` type (#262). Thanks @petrus-v for the PR.
- Remove imprecise Python 3 classifier from PyPI metadata (#255). Thanks @ecederstrand.

7.1.31 0.20.0 (2019-12-01)

- Add support for `mysql.DATETIME` and `mysql.INTEGER` type (#204).
- Add support for `postgresql.CIDR` type (#183).
- Add support for `postgresql.DATE` and `postgresql.TIME` type.

Thanks @evelyn9191 for the PR.

7.1.32 0.19.0 (2019-09-05)

- Drop support for Python 2.7 and 3.5 (#241).
- Drop support for marshmallow<2.15.2.
- Only support sqlalchemy>=1.2.0.

7.1.33 0.18.0 (2019-09-05)

Features:

- `marshmallow_sqlalchemy.fields.Nested` propagates the value of `transient` on the call to `load` (#177, #206). Thanks @leonidumanskiy for reporting.

Note: This is the last release to support Python 2.7 and 3.5.

7.1.34 0.17.2 (2019-08-31)

Bug fixes:

- Fix error handling when passing an invalid type to `Related` (#223). Thanks @heckad for reporting.
- Address `DeprecationWarning` raised when using `Related` with marshmallow 3 (#243).

7.1.35 0.17.1 (2019-08-31)

Bug fixes:

- Add `marshmallow_sqlalchemy.fields.Nested` field that inherits its session from its schema. This fixes a bug where an exception was raised when using `Nested` within a `ModelSchema` (#67). Thanks @nickw444 for reporting and thanks @samueljsb for the PR.

User code should be updated to use marshmallow-sqlalchemy's `Nested` instead of `marshmallow.fields.Nested`.

```
# Before
from marshmallow import fields
from marshmallow_sqlalchemy import ModelSchema


class ArtistSchema(ModelSchema):
    class Meta:
        model = models.Artist


class AlbumSchema(ModelSchema):
    class Meta:
        model = models.Album

    artist = fields.Nested(ArtistSchema)


# After
from marshmallow import fields
from marshmallow_sqlalchemy import ModelSchema
from marshmallow_sqlalchemy.fields import Nested


class ArtistSchema(ModelSchema):
    class Meta:
        model = models.Artist


class AlbumSchema(ModelSchema):
    class Meta:
```

(continues on next page)

(continued from previous page)

```
        model = models.Album

    artist = Nested(ArtistSchema)
```

7.1.36 0.17.0 (2019-06-22)

Features:

- Add support for `postgresql.MONEY` type (#218). Thanks @heckad for the PR.

7.1.37 0.16.4 (2019-06-15)

Bug fixes:

- Compatibility with marshmallow 3.0.0rc7. Thanks @heckad for the catch and patch.

7.1.38 0.16.3 (2019-05-05)

Bug fixes:

- Compatibility with marshmallow 3.0.0rc6.

7.1.39 0.16.2 (2019-04-10)

Bug fixes:

- Prevent ValueError when using the `exclude` class Meta option with `TableSchema` (#202).

7.1.40 0.16.1 (2019-03-11)

Bug fixes:

- Fix compatibility with SQLAlchemy 1.3 (#185).

7.1.41 0.16.0 (2019-02-03)

Features:

- Add support for deserializing transient objects (#62). Thanks @jacksmith15 for the PR.

7.1.42 0.15.0 (2018-11-05)

Features:

- Add `ModelConverter._should_exclude_field` hook (#139). Thanks @jeanphix for the PR.
- Allow field `kwargs` to be overriden by passing `info['marshmallow']` to column properties (#21). Thanks @dpwrussell for the suggestion and PR. Thanks @jeanphix for the final implementation.

7.1.43 0.14.2 (2018-11-03)

Bug fixes:

- Fix behavior of `Related` field (#150). Thanks @zezic for reporting and thanks @AbdealiJK for the PR.
- `Related` now works with `AssociationProxy` fields (#151). Thanks @AbdealiJK for the catch and patch.

Other changes:

- Test against Python 3.7.
- Bring development environment in line with marshmallow.

7.1.44 0.14.1 (2018-07-19)

Bug fixes:

- Fix behavior of `exclude` with marshmallow 3.0 (#131). Thanks @yaheath for reporting and thanks @deckar01 for the fix.

7.1.45 0.14.0 (2018-05-28)

Features:

- Make `ModelSchema.session` a property, which allows session to be retrieved from `context` (#129). Thanks @gtxm.

Other changes:

- Drop official support for Python 3.4. Python>=3.5 and Python 2.7 are supported.

7.1.46 0.13.2 (2017-10-23)

Bug fixes:

- Unset `instance` attribute when an error occurs during a `load` call (#114). Thanks @vgavro for the catch and patch.

7.1.47 0.13.1 (2017-04-06)

Bug fixes:

- Prevent unnecessary queries when using the `fields.Related` (#106). Thanks @xarg for reporting and thanks @jmuhlich for the PR.

7.1.48 0.13.0 (2017-03-12)

Features:

- Invalid inputs for compound primary keys raise a `ValidationError` when deserializing a scalar value (#103). Thanks @YuriHeupa for the PR.

Bug fixes:

- Fix compatibility with marshmallow>=3.x.

7.1.49 0.12.1 (2017-01-05)

Bug fixes:

- Reset `ModelSchema.instance` after each `load` call, allowing schema instances to be reused (#78). Thanks @georgexsh for reporting.

Other changes:

- Test against Python 3.6.

7.1.50 0.12.0 (2016-10-08)

Features:

- Add support for TypeDecorator-based types (#83). Thanks @frol.

Bug fixes:

- Fix bug that caused a validation errors for custom column types that have the `python_type` of `uuid.UUID` (#54). Thanks @wkevina and thanks @kelvinhammond for the fix.

Other changes:

- Drop official support for Python 3.3. Python>=3.4 and Python 2.7 are supported.

7.1.51 0.11.0 (2016-10-01)

Features:

- Allow overriding field class returned by `field_for` by adding the `field_class` param (#81). Thanks @cancan101.

7.1.52 0.10.0 (2016-08-14)

Features:

- Support for SQLAlchemy JSON type (in SQLAlchemy>=1.1) (#74). Thanks @ewittle for the PR.

7.1.53 0.9.0 (2016-07-02)

Features:

- Enable deserialization of many-to-one nested objects that do not exist in the database (#69). Thanks @seanharr11 for the PR.

Bug fixes:

- Depend on SQLAlchemy>=0.9.7, since marshmallow-sqlalchemy uses `sqlalchemy.dialects.postgresql.JSONB` (#65). Thanks @alejom99 for reporting.

7.1.54 0.8.1 (2016-02-21)

Bug fixes:

- `ModelSchema` and `TableSchema` respect field order if the `ordered=True` class Meta option is set (#52). Thanks @jeffwidman for reporting and @jmcarp for the patch.
- Declared fields are not introspected in order to support, e.g. `column_property` (#57). Thanks @jmcarp.

7.1.55 0.8.0 (2015-12-28)

Features:

- `ModelSchema` and `TableSchema` will respect the `TYPE_MAPPING` class variable of Schema subclasses when converting `Columns` to `Fields` (#42). Thanks @dwieeb for the suggestion.

7.1.56 0.7.1 (2015-12-13)

Bug fixes:

- Don't make marshmallow fields required for non-nullable columns if a column has a default value or autoincrements (#47). Thanks @jmcarp for the fix. Thanks @AdrielVelazquez for reporting.

7.1.57 0.7.0 (2015-12-07)

Features:

- Add `include_fk` class Meta option (#36). Thanks @jmcarp.
- Non-nullable columns will generated required marshmallow Fields (#40). Thanks @jmcarp.
- Improve support for MySQL BIT field (#41). Thanks @rudaporto.
- *Backwards-incompatible*: Remove `fields.get_primary_columns` in favor of `fields.get_primary_keys`.
- *Backwards-incompatible*: Remove `Related.related_columns` in favor of `fields.related_keys`.

Bug fixes:

- Fix serializing relationships when using non-default column names (#44). Thanks @jmcarp for the fix. Thanks @repole for the bug report.

7.1.58 0.6.0 (2015-09-29)

Features:

- Support for compound primary keys. Thanks @jmcarp.

Other changes:

- Supports marshmallow>=2.0.0.

7.1.59 0.5.0 (2015-09-27)

- Add `instance` argument to `ModelSchema` constructor and `ModelSchema.load` which allows for updating existing DB rows (#26). Thanks @sssilver for reporting and @jmcarp for the patch.
- Don't autogenerate fields that are in `Meta.exclude` (#27). Thanks @jmcarp.
- Raise `ModelConversionError` if converting properties whose column don't define a `python_type`. Thanks @jmcarp.
- *Backwards-incompatible*: `ModelSchema.make_object` is removed in favor of decorated `make_instance` method for compatibility with marshmallow>=2.0.0rc2.

7.1.60 0.4.1 (2015-09-13)

Bug fixes:

- Now compatible with marshmallow>=2.0.0rc1.
- Correctly pass keyword arguments from `field_for` to generated `List` fields (#25). Thanks @sssilver for reporting.

7.1.61 0.4.0 (2015-09-03)

Features:

- Add `TableSchema` for generating `Schemas` from tables (#4). Thanks @jmcarp.

Bug fixes:

- Allow `session` to be passed to `ModelSchema.validate`, since it requires it. Thanks @dpwrussell.
- When serializing, don't skip overriden fields that are part of a polymorphic hierarchy (#18). Thanks again @dpwrussell.

Support:

- Docs: Add new recipe for automatic generation of schemas. Thanks @dpwrussell.

7.1.62 0.3.0 (2015-08-27)

Features:

- *Backwards-incompatible*: Relationships are (de)serialized by a new, more efficient `Related` column (#7). Thanks @jmcarp.
- Improve support for MySQL types (#1). Thanks @rmackinnon.
- Improve support for Postgres ARRAY types (#6). Thanks @jmcarp.
- `ModelSchema` no longer requires the `sqla_session` class Meta option. A `Session` can be passed to the constructor or to the `ModelSchema.load` method (#11). Thanks @dtheodor for the suggestion.

Bug fixes:

- Null foreign keys are serialized correctly as `None` (#8). Thanks @mitchej123.
- Properly handle a relationship specifies `uselist=False` (#17). Thanks @dpwrussell.

7.1.63 0.2.0 (2015-05-03)

Features:

- Add `field_for` function for generating marshmallow Fields from SQLAlchemy mapped class properties.

Support:

- Docs: Add "Overriding generated fields" section to "Recipes".

7.1.64 0.1.1 (2015-05-02)

Bug fixes:

- Fix `keygetter` class Meta option.

7.1.65 0.1.0 (2015-04-28)

- First release.

7.2 Authors

7.2.1 Leads

- Steven Loria @sloria

7.2.2 Contributors

- Rob MacKinnon @rmackinnon
- Josh Carp @jmcarp
- Jason Mitchell @mitchej123
- Douglas Russell @dpwrussell
- Rudá Porto Filgueiras @rudaporto
- Sean Harrington @seanharr11
- Eric Wittle @ewittle

- Alex Rothberg @cancan101
- Vlad Frolov @frol
- Kelvin Hammond @kelvinhammond
- Yuri Heupa @YuriHeupa
- Jeremy Muhlich @jmuhlich
- Ilya Chistyakov @ilya-chistyakov
- Victor Gavro @vgavro
- Maciej Barański @gtxm
- Jared Deckard @deckar01
- AbdealiJK @AbdealiJK
- jean-philippe serafin @jeanphix
- Jack Smith @jacksmith15
- Kazantcev Andrey @heckad
- Samuel Searles-Bryant @samueljsb
- Michaela Ockova @evelyn9191
- Pierre Verkest @petrus-v
- Erik Cederstrand @ecederstrand
- Daven Quinn @davenquinn
- Peter Schutt @peterschutt
- Arash Fatahzade @ArashFatahzade
- David Malakh @Unix-Code
- Martijn Pieters @mjpieters
- Bruce Adams @bruceadams
- Justin Crown @mrname
- Jeppe Fihl-Pearson @Tenzer
- Indivar @indiVar0508
- David Doyon @ddoyon92
- Hippolyte Henry @zippolyte
- Alexandre Detiste tchet@debian.org
- PandaByte @panda-byte

7.3 License

```
Copyright Steven Loria and contributors

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
```

(continues on next page)

(continued from previous page)

```
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

PYTHON MODULE INDEX

m

INDEX

T

V