Mercator - Data Mapper for Protobuf Documentation

Release 0.1.11

NewStore Inc.

Apr 06, 2020

Table of Contents:

Python DSL to leverage translation of dictionaries and SQLAlchemy into Protobuf objects.

Primarily created with the intention of migrating python services to support gRPC.

CHAPTER 1

What is Mercator ?

Mercator is a Python library that simplifies the following of serializing dictionary data into Protobuf binary data.

Mercator actually supports extracting data from:

- dictionaries
- SQLAlchemy model instances
- Any opaque python objects (e.g.: `namedtuple()`)

1.1 When should I use Mercator ?

- When migrating custom implementations of RPC to gGRPC.
- When migrating in-memory data to Protobuf.

1.2 When should I not use Mercator ?

- When writing gRPC services from scratch.
- When writing anything that uses Protobuf gets called.

CHAPTER 2

A Data Mapper for Protobuf

This library is primarily intended to aid the migration of python-based microservices to gRPC by leveraging a DSL that resembles ORM and ActiveRecord patterns.

It supports mapping key/values from dictionaries to Protobuf 3 as well as SQLAlchemy ORM models intro Protobuf 3.

2.1 Installing

```
pip install mercator
```

2.1.1 Proto Mapping in Detail

A *ProtoMapping* provides syntax sugar to define ways in which python dictionaries or objects can have its keys or properties serialized into predefined ProtoBuf messages.

__proto__

Every *ProtoMapping* must declare a *__proto__* attribute that points to a valid Message subclass.

Example:

```
from google.protobuf.timestamp_pb2 import Timestamp

from mercator import ProtoMapping
from mercator import ProtoKey


```

(continues on next page)

(continued from previous page)

```
class TimestampMapping(ProtoMapping):
    __proto__ = Timestamp

 seconds = ProtoKey('seconds', int)
```

Warning: Failing to declare a valid *__proto__* attribute will cause mercator to raise a `SyntaxError`

`__source_input_type__`

If declared, this property will be considered as base-class of opaque objects that can have its properties mapped into protobuf.

This feature was primarily designed to support SQLAlchemy ORM models out of the box but supports any opaque python objects, as long as their base classes are defined by this attribute.

```
from sqlalchemy.ext.declarative import declarative_base

from mercator import ProtoMapping
from mercator import ProtoKey


MySimpleBaseModel = declarative_base()

class User(MySimpleBaseModel):
    __tablename__ = 'user'
    __table_args__ = {'useexisting': True}

    login = sa.Column(sa.String(256))
    email = sa.Column(sa.String(256))
    password = sa.Column(sa.String(256))


class UserMapping(ProtoMapping):
    __proto__ = domain_pb2.User
    __source_input_type__ = User
```

Important: This attribute is optional when declaring proto mappings, but if defined it must be a `type`.

See also:

The section *SQLAlchemy ORM Support* for more information on how to use the `__source_input_type__` attribute.

Field mappings

Field mappings are either *`ProtoKey`* or *`ProtoList`* class-attributes defined in the body of your *`ProtoMapping`* subclass.

This gives you the power to gather data from dictionaries with keys that are different than in the protobuf model.

target_type

Field mappings are subclasses of *mercator.meta.FieldMapping* and share its __init__ signature:

```
FieldMapping(name_at_source: str, target_type: type)

ProtoKey(name_at_source: str, target_type: type)

ProtoList(name_at_source: str, target_type: type)
```

The target_type argument is optional, but when given, supports different types.

Let's dive into the possibilities.

Native python types

Ensures that the field value is cast into any python type, namely: str, int, float, long, dict, list

Mappings of Mappings

Allows recursively translating data into protobuf messages whose members contain sub-messages.

Example

```
from mercator import (
    ProtoMapping,
    ProtoKey,
)
from . import domain_pb2
from . import sql


class UserMapping(ProtoMapping):
    __proto__ = domain_pb2.User

    uuid = ProtoKey('id', str)
    email = ProtoKey('email', str)
    username = ProtoKey('login', str)


class MediaMapping(ProtoMapping):
    __proto__ = domain_pb2.UserMedia

    author = ProtoKey('owner', UserMapping)
    download_url = ProtoKey('link', str)
```

2.1.2 SQLAlchemy ORM Support

Sometimes it can be useful to map SQLAlchemy ORM objects when migrating other RPC implementations to gRPC.

On Power and Responsability

It is important to note that this practice can have pernicious effects on the separation of responsability of your codebase.

ProtoMappings provide a powerful way to make this transition easier, so use it wisely.

A simple recipe

This example was partially extracted from the functional tests and simplified for didactic purposes. There is a more complete example on github.

Here we simplified our recipe to a service to *"return user data from a SQL database"*

Ingredients

1. A SQLAlchemy model: `User`
2. A Protobuf definition of a `User` message and a `UserService`
3. The implementation of the python service that uses a `UserMapping` to map model fields into protobuf message fields.

A SQLAlchemy Model

```
from sqlalchemy.ext.declarative import declarative_base


MySimpleBaseModel = declarative_base()

class User(MySimpleBaseModel):
    __tablename__ = 'user'
    __table_args__ = {'useexisting': True}

    id = sa.Column(
        postgresql.UUID(as_uuid=True),
        primary_key=True,
        default=uuid4
    )
    login = sa.Column(sa.String(256))
    email = sa.Column(sa.String(256))
    password = sa.Column(sa.String(256))
```

A protobuf declaration

```
syntax = "proto3";
package services.simple_example.sqlalchemy;

message User {
  string uuid = 1;
  string username = 2;
  string email = 3;
}
```

(continues on next page)

(continued from previous page)

```
message UserDataRequest {
    string user_uuid = 1;
}

service UserService {
    rpc GetUser (UserDataRequest) returns (User){};
}
```

The service implementation

```
from . import user_pb2_grpc
from . import sql

class UserMapping(ProtoMapping):
    # the destination type, must come from a *_pb2.py file compiled from your *.proto␣
↪file

    __proto__ = domain_pb2.User

    # the base type of your sqlalchemy types
    __source_input_type__ = sql.MySimpleBaseModel

    uuid = ProtoKey('id', str)     # translate "id" into "uuid"
    email = ProtoKey('email', str)
    username = ProtoKey('login', str) # translate "login" into "username"


class business_logic:
    """isolates SQL queries returning objects
    ready for the protobuf serialization layer"""

    @staticmethod
    def get_user_by_uuid(uuid):
        result = sql.session.query(sql.User).where(sql.User.uuid==uuid)
        return result.one()


class UserService(user_pb2_grpc.UserService):
    def GetUser(self, request, context):
        # retrieve sqlalchemy instance of user by uuid
        user = business_logic.get_user_by_id(request.user_uuid)

        return UserMapping(user).to_protobuf()
```

Full example: ORM Relationships

Warning: while entirely supported, this feature can have pernicious impact in the coupling of SQL data model with gRPC protobuf data modeling. Use with caution.

The SQL Models

```
from uuid import uuid4
import sqlalchemy as sa
from sqlalchemy import orm as sa_orm
from sqlalchemy.dialects import postgresql
from sqlalchemy.ext.declarative import declarative_base


BaseModel = declarative_base()


def PrimaryKeyUUID():
    return sa.Column(
        postgresql.UUID(as_uuid=True),
        primary_key=True,
        default=uuid4
    )


class User(BaseModel):
    __tablename__ = 'user'
    __table_args__ = {'useexisting': True}

    uuid = PrimaryKeyUUID()
    login = sa.Column(sa.String(256))
    email = sa.Column(sa.String(256))
    password = sa.Column(sa.String(256))
    extra_info = sa.Column(
        postgresql.JSON,
        nullable=True,
    )


class AuthToken(BaseModel):
    __tablename__ = 'auth_token'
    __table_args__ = {'useexisting': True}

    uuid = PrimaryKeyUUID()
    data = sa.Column(sa.String(256))
    created_at = sa.Column(sa.Integer)
    owner_id = sa.Column(
        postgresql.UUID(as_uuid=True),
        sa.ForeignKey('User.uuid')
    )
    owner = sa_orm.relationship(
        User,
        primaryjoin='and_(User.uuid == foreign(AuthToken.owner_id))',
        backref='tokens',
        uselist=False,
    )


class Media(BaseModel):
    __tablename__ = 'media'
    __table_args__ = {'useexisting': True}
```

(continues on next page)

(continued from previous page)

```
    uuid = PrimaryKeyUUID()
    author_id = sa.Column(
        postgresql.UUID(as_uuid=True),
        sa.ForeignKey('User.uuid')
    )
    author = sa_orm.relationship(
        User,
        primaryjoin='and_(Media.author_id == foreign(User.uuid))',
        backref='media',
        uselist=False,
    )
    url = sa.Column(sa.String(256))
```

Protobuf declaration

For consistency with code examples let's consider this is saved with `social_platform.proto` and subsequently compiled to python with:

```
python -m grpc_tools.protoc -I ./ \
    --python_out=./
    --grpc_python_out=./
    ./social_platform.proto
```

```
syntax = "proto3";
package services.social_platform;

import "google/protobuf/timestamp.proto";
import "google/protobuf/struct.proto";

service Auth {
  // returns an User.AuthToken
  rpc AuthenticateUser (AuthRequest) returns (AuthResponse){};
}

message AuthRequest {
  string username = 1;
  string password = 2;
}
message AuthResponse {
  User.AuthToken token = 1;
}


message User {
  message AuthToken {
    string value = 1;
    google.protobuf.Timestamp created_at = 2;
    google.protobuf.Timestamp expires_at = 3;
  }

  string uuid = 1;
  string username = 2;
  string email = 3;
  repeated AuthToken tokens = 4;
```

(continues on next page)

(continued from previous page)

```
  google.protobuf.Struct metadata = 5;
}

message UserMedia {
  string uuid = 1;
  string name = 2;
  User author = 3; // the author of the media
  string download_url = 4; // the URL where the media can be downloaded
  bytes blob = 5; // the media itself, if available.

  enum ContentType {
    BLOG_POST = 0;
    IMAGE = 1;
    VIDEO = 2;
    QUOTE = 3;
    GIF = 4;
  }
  ContentType content_type = 4;
}

message MediaRequest {
   string media_uuid = 1;
   string media_name = 2;
}

service Media {
   rpc GetMedia (MediaRequest) returns (UserMedia){};
}
```

Service Implementation with Mappings of Mappings

```
import grpc

from mercator import (
    ProtoMapping,
    ProtoKey,
    ProtoList,
    SinglePropertyMapping,
)
from google.protobuf.timestamp_pb2 import Timestamp
from concurrent.futures import ThreadPoolExecutor

from . import social_platform_pb2
from . import social_platform_pb2_grpc
from . import sql


ProtobufTimestamp = SinglePropertyMapping(int, Timestamp, 'seconds')

class AuthRequestMapping(ProtoMapping):
    __proto__ = social_platform_pb2.AuthRequest

    username = ProtoKey('username', str)
    password = ProtoKey('password', str)
```

(continues on next page)

(continued from previous page)

```
class UserAuthTokenMapping(ProtoMapping):
    __proto__ = social_platform_pb2.User.AuthToken
    __source_input_type__ = sql.AuthToken

    value = ProtoKey('data', str)
    created_at = ProtoKey('created_at', ProtobufTimestamp)
    expires_at = ProtoKey('expires_at', ProtobufTimestamp)


class UserMapping(ProtoMapping):
    __proto__ = social_platform_pb2.User
    __source_input_type__ = sql.User

    uuid = ProtoKey('id', str)
    email = ProtoKey('email', str)
    username = ProtoKey('login', str)
    tokens = ProtoList('tokens', UserAuthTokenMapping)
    metadata = ProtoKey('extra_info', dict)


class MediaMapping(ProtoMapping):
    __proto__ = social_platform_pb2.UserMedia
    __source_input_type__ = sql.Media

    author = ProtoKey('author', UserMapping)
    download_url = ProtoKey('link', str)
    blob = ProtoKey('blob', bytes)
    content_type = ProtoKey('content_type', bytes)


class AuthResponseMapping(ProtoMapping):
    __proto__ = social_platform_pb2.AuthResponse

    token = ProtoKey('token', UserAuthTokenMapping)


class MediaRequestMapping(ProtoMapping):
    __proto__ = social_platform_pb2.MediaRequest


class MediaServicer(social_platform_pb2_grpc.MediaServicer):
    def GetMedia(self, request, context):
        media = business_logic_module.retrieve_media_from_sqlalchemy(
            uuid=request.media_uuid,
            name=request.media_name,
        )

        return MediaMapping(media).to_protobuf()


server = grpc.server(
    ThreadPoolExecutor(max_workers=10)
)
```

(continues on next page)

(continued from previous page)

```
social_platform_pb2_grpc.add_MediaServicer_to_server(MediaServicer(), server)
```

2.1.3 API Reference

ProtoMapping

class mercator.**ProtoMapping**(*data*)

Base class to define attribute mapping from `dict` or `declarative_base()` subclasses' instances into pre-filled protobuf messages.

Example:

```
from mercator import (
    ProtoMapping,
    ProtoKey,
    ProtoList,
    SinglePropertyMapping,
)
from google.protobuf.timestamp_pb2 import Timestamp

ProtobufTimestamp = SinglePropertyMapping(int, Timestamp, 'seconds')

class AuthRequestMapping(ProtoMapping):
    __proto__ = domain_pb2.AuthRequest

    username = ProtoKey('username', str)
    password = ProtoKey('password', str)


class UserAuthTokenMapping(ProtoMapping):
    __proto__ = domain_pb2.User.AuthToken
    value = ProtoKey('data', str)
    created_at = ProtoKey('created_at', ProtobufTimestamp)
    expires_at = ProtoKey('expires_at', ProtobufTimestamp)


class UserMapping(ProtoMapping):
    __proto__ = domain_pb2.User

    uuid = ProtoKey('id', str)
    email = ProtoKey('email', str)
    username = ProtoKey('login', str)
    tokens = ProtoList('tokens', UserAuthTokenMapping)
    metadata = ProtoKey('extra_info', dict)


class MediaMapping(ProtoMapping):
    __proto__ = domain_pb2.Media

    author = ProtoKey('author', UserMapping)
    download_url = ProtoKey('link', str)
    blob = ProtoKey('blob', bytes)
    content_type = ProtoKey('content_type', bytes)
```

(continues on next page)

(continued from previous page)

```
class AuthResponseMapping(ProtoMapping):
    __proto__ = domain_pb2.AuthResponse

    token = ProtoKey('token', UserAuthTokenMapping)
```

to_dict()

Returns a `dict` with keyword-arguments to construct a new instance of protobuf message defined by *__proto__*.

to_protobuf()

Returns a new *__proto__* instance with the data extracted with *to_dict()*.

ProtoKey

class mercator.**ProtoKey**(*name_at_source: str*, *target_type: type = None*)

Represents the intent to translate a object property or dictionary key into a protobuf message.

Use this to map specific values into a protobuf object.

Example:

```
class UserMapping(ProtoMapping):
    __proto__ = domain_pb2.User

    username = ProtoKey('login', str)
```

Parameters

- **name_at_source** – a string with the name of key or property to be extracted in an input object before casting into the target type.
- **target_type** – an optional *ProtoMapping* subclass or native python type. Check *target_type* for more details.

cast(*value*)

Parameters **value** – a python object that is compatible with the given `target_type`

Returns `value` coerced into the target type. Supports ProtoMappings by automatically calling *to_protobuf()*.

ProtoList

class mercator.**ProtoList**(*name_at_source: str*, *target_type: type = None*)

Represents the intent to translate a several object properties or dictionary keys into a list in a protobuf message.

Example:

```
class UserMapping(ProtoMapping):
    __proto__ = domain_pb2.User

    tokens = ProtoList('tokens', UserAuthTokenMapping)
```

Parameters

- **name_at_source** – a string with the name of key or property to be extracted in an input object before casting into the target type.
- **target_type** – an optional *ProtoMapping* subclass or native python type. Check *target_type* for more details.

cast(*value*)

Parameters **value** – a python object that is compatible with the given `target_type`

Returns list of items target type coerced into the `target_type`. Supports ProtoMappings by automatically calling *to_protobuf()*.

SinglePropertyMapping

class mercator.**SinglePropertyMapping**(*to_python*, *pb2_type*, *argname*)

creates a new instance of the given protobuf type populated with a single property preprocessing the input value with the given callable.

Example:

```
from mercator import (
    ProtoMapping,
    ProtoKey,
    SinglePropertyMapping,
)
from google.protobuf.timestamp_pb2 import Timestamp

ProtobufTimestamp = SinglePropertyMapping(int, Timestamp, 'seconds')

class UserAuthTokenMapping(ProtoMapping):
    __proto__ = domain_pb2.User.AuthToken
    value = ProtoKey('data', str)
    created_at = ProtoKey('created_at', ProtobufTimestamp)
    expires_at = ProtoKey('expires_at', ProtobufTimestamp)


 auth_token = UserAuthTokenMapping({'created_at': 12345}).to_protobuf()

 assert isinstance(auth_token.created_at, Timestamp)
 assert auth_token.created_at.seconds == 12345
```

mercator.errors

exception mercator.errors.**ProtobufCastError**

raised when failed to create a `Message` instance

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception mercator.errors.**TypeCastError**

Raised when trying to cast a value of the wrong type. Used primarily by *mercator.ProtoList.cast()* from *ProtoList*

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

mercator.meta

class mercator.meta.**FieldMapping**(*name_at_source: str*, *target_type: type = None*)
Base-class for field mapping declaration in *ProtoMapping* that is:

- *ProtoKey*
- *ProtoList*

This base-class resides in *mercator.meta* so the metaclass can capture the field mapping declarations during import-time.

Parameters

- **name_at_source** – a string with the name of key or property to be extracted in an input object before casting into the target type.
- **target_type** – an optional *ProtoMapping* subclass or native python type. Check *target_type* for more details.

cast(*value*)
coerces the given `value` into the target type. :param value: a python object that is compatible with the given `target_type`

class mercator.meta.**ImplicitField**(*name_at_source: str*, *target_type: type = None*)
Like *ProtoKey* but works is declared automagically by the metaclass.

cast(*value*)
coerces the given `value` into the target type. :param value: a python object that is compatible with the given `target_type`

class mercator.meta.**MercatorDomainClass**
The existence of this class is a trick to avoid redundant imports.

Any subclasses of this are automatically supported by *FieldMapping* as `target_type`.

This was introduced to support *SinglePropertyMapping* but can be used in the future in any new types that leverage type casting.

class mercator.meta.**MetaMapping**
Metaclass to leverage and enforce correct syntax sugar when declaring protomappings.

Check the source code for comments explaining how everything works.

mro()
Return a type's method resolution order.

mercator.meta.**field_properties_from_proto_class**(*proto_class*)
inspects all members of the given proto_class and returns a list of those who seem to be a message field (determined by *is_field_property()*)

mercator.meta.**is_field_property**(*obj*)
utility function to check if the give object is a field from a protobuf message.

For now there is not much metadata when inspecting a protobuf class compiled by protoc, so the current strategy is:

1. Take the `obj.__class__`
2. Take the `__name__` of the class.

3. Check that the name contains the string "FieldProperty"

`mercator.meta.`**`validate_and_register_base_model_class`**(*cls*, *name*, *attributes*)
Invoked by `MetaMapping` during "import time" to validate the declaration of *__source_input_type__*.

`mercator.meta.`**`validate_proto_attribute`**(*name*, *attributes*)
Invoked by `MetaMapping` during "import time" to validate the declaration of *__proto__*.

CHAPTER 3

Example Usage

```
from myproject.mappings import (
    UserMapping,
)

from . import avengers_pb2


class AvengersService(avengers_pb2.HeroServicer):
    def GetHulk(self):
        info = {
            'login': 'Hulk',
            'email': 'bruce@avengers.world',
            'tokens': [
                {
                    'data': 'this is the token',
                    'created_at': 1552240433,
                    'expires_at': 1552240733,
                }
            ],
        }
        return UserMapping(info).to_protobuf()
```

Python Module Index

m

Index