ProtoBuf Schematics Documentation

Release 0.4.1

Almog Cohen

Feb 02, 2019

Contents:

CHAPTER 1

ProtoBuf Schematics

Convert ProtoBuf `proto` file to cute Schematics classes file.

Google Protobuf is great when it comes to high performance schema aware APIs, but when Google designed Protobuf, it didn't tried to make the generated code idiomatic in Python, which brings a problem when exporting messages outside interface modules or having nice IDE auto-completions. Schematics is a cute and Pythonic schema library that goes well with most applications. Why not join both?

Currently this package does **not** support the Protobuf binary format and will work with a any other textual representation which is easily generated with the original Protobuf API for any language. Ease of use was prioritized while writing this package rather than mere performance.

- Free software: Apache Software License 2.0
- Documentation: https://protobuf-schematics.readthedocs.io.

1.1 Usage

1. Convert the `proto` file to python Schematics classes:

```
protobuf_schematics <path-to-file.proto> generated_schematics_proto.py # or any␣
↪output filename
```

2. Convert your ProtoBuf message to Json:

In Java:

```
import com.google.protobuf.util.JsonFormat;

FileWriter file = new FileWriter("protoBufMessage.json")
JsonFormat.Printer printer = JsonFormat.printer().preservingProtoFieldNames();
String message = printer.print(someProtoBufMessage);
file.write(message)
```

or from Python:

```
import json
from google.protobuf.json_format import MessageToJson

json = MessageToJson(org, preserving_proto_field_name=True, including_default_value_
↪fields=True)
with open("protoBufMessage.json", 'w') as output:
    json.dump(json, output)
```

3. In your project, load the message in python as Schematics object:

```
import json
from generated_schematics_proto import SomeMessage # import the schematics message␣
↪class

schematics_root_message = SomeMessage(json.load(open('protoBufMessage.json')))
```

Or use a message loaded in python by Protobuf:

```
import json
from generated_schematics_proto import SomeMessage # import the schematics message␣
↪class

# ... get your protobuf message as the pb class representation
schematics_root_message = SomeMessage.import_from_protobuf_message(protobuf_message)
```

1.2 Example

This `proto` file:

```
syntax = "proto3";

enum IPAddressFamily {
    INVALID = 0;
    IPv4 = 1;
    IPv6 = 2;
};

message ProtocolAndPorts {
    repeated uint32 ports = 3;
}

message FlowFilter {
    enum SomeEnum {
        VALUE = 0;
    };
    string id = 1 [deprecated = true];
    SomeEnum consumer_filter_id = 2;
    map<string, ProtocolAndPorts> ports = 3;
    repeated ProtocolAndPorts protocol_and_ports = 4;
}
```

Will be converted to:

```
class IPAddressFamily(Enum):
    INVALID = 0
```

(continues on next page)

(continued from previous page)

```
    IPv4 = 1
    IPv6 = 2


class ProtocolAndPorts(ProtobufMessageModel):
    ports = ListType(IntType())


class FlowFilter(ProtobufMessageModel):
    class InnerEnum(Enum):
        VALUE = 0

    id = StringType()
    consumer_filter_id = EnumType(SomeEnum)
    ports = DictType(ModelType(ProtocolAndPorts), str)
    protocol_and_ports = ListType(ModelType(ProtocolAndPorts))
```

1.3 Features

- Support both Protobuf syntax 2 and 3.
- Support builtin types such as StringType, `IntType`.
- Support proto map fields as Schematics `DictType`.
- Support `repeated` modifier as convert to `ListType`.
- Support Enum class generation and custom Schematics `EnumType`.
- Support custom schematics `ByteArrayType` base64 encoded byte arrays converted from Java.

1.4 Development

First, install the Pipfile and create the proper virtual environment:

```
pipenv install --dev
```

To check linting with **flake8**, run:

```
make lint
```

To run the unittests against your working python version:

```
py.test
```

To see coverage report:

```
make coverage
```

To run tests against all supported python versions:

```
tox
```

To make the docs (which will be automatically published to readthedocs on commits to the master branch):

```
make docs
```

1.5 Credits

The parsing work of **.proto** files is provided thanks to the awesome guys at PyroBuf.

This package was created with Cookiecutter and the elgertam/cookiecutter-pipenv project template, based on audreyr/cookiecutter-pypackage.

CHAPTER 2

Installation

2.1 Stable release

To install ProtoBuf Schematics, run this command in your terminal:

```
$ pip install protobuf_schematics
```

This is the preferred method to install ProtoBuf Schematics, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for ProtoBuf Schematics can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/AlmogCohen/protobuf_schematics
```

Or download the tarball:

```
$ curl  -OL https://github.com/AlmogCohen/protobuf_schematics/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use ProtoBuf Schematics in a project:

```
import protobuf_schematics
```

CHAPTER 4

protobuf_schematics

4.1 protobuf_schematics package

4.1.1 Submodules

4.1.2 protobuf_schematics.cli module

4.1.3 protobuf_schematics.engine module

4.1.4 protobuf_schematics.filters module

4.1.5 protobuf_schematics.models module

4.1.6 protobuf_schematics.types module

4.1.7 Module contents

Top-level package for ProtoBuf Schematics.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/AlmogCohen/protobuf_schematics/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

ProtoBuf Schematics could always use more documentation, whether as part of the official ProtoBuf Schematics docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/AlmogCohen/protobuf_schematics/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *protobuf_schematics* for local development.

1. Fork the *protobuf_schematics* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/protobuf_schematics.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv protobuf_schematics
$ cd protobuf_schematics/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 protobuf_schematics tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/AlmogCohen/protobuf_schematics/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_protobuf_schematics
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

History

6.1 TODO

- Add more meaningful unittests.
- Improve Protobuf support and integration.
- add support for parsing the Protobuf generated files as it is more comprehensive than the current Pyrobuf support.

6.2 Read for Release

6.3 0.4.0 (2019-01-16)

- Added more meaningful unittests.
- Export Types and Models out of the templates.
- Export heavy definition logic (such as field definition) from the jinja template to pythonic filters.
- Improve integration with Protobuf.

6.4 0.2.0 (2019-01-14)

- Improve README - Add usage examples.
- Update the setup.py to point to official PyroBuf 0.8.5 PyPi release.

6.5 0.1.0 (2019-01-13)

- First release on PyPI.
- CLI interface for compiling a given proto file to schematics definition file.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

P