python package generate protobuf code automatically as part of package

If you want to automatically generate Protobuf code as part of a Python package, you can use the `setuptools` library along with the `protobuf` package. This can be achieved by running the Protobuf compiler (`protoc`) as part of the build process when installing the package.

Here's how you can integrate Protobuf code generation into a Python package:

1. Install `protobuf`

You need to install the `protobuf` Python package first:

bash Copy code

```
pip install protobuf
```

You also need the `protoc` compiler, which you can install via your system's package manager or download from the Protocol Buffers releases page ↗.

2. Create your `.proto` file(s)

Define your Protobuf schema in `.proto` files. For example, create a file called `example.proto` :

proto Copy code

```
syntax = "proto3";

message Person {
    string name = 1;
    int32 id = 2;
    string email = 3;
}
```

3. Add `setup.py` to your Python package

Modify your `setup.py` file to automatically compile `.proto` files when the package is installed. You can use `setuptools` and the `protobuf` package for this.

python Copy code

```
from setuptools import setup, find_packages
from setuptools.command.build_py import build_py as _build_py
import os
import subprocess


class BuildProto(_build_py):
    def run(self):
        # Run the protoc compiler to generate Python code
        proto_dir = os.path.join(os.path.dirname(__file__), 'proto')
        if os.path.exists(proto_dir):
            for proto_file in os.listdir(proto_dir):
                if proto_file.endswith('.proto'):
                    proto_path = os.path.join(proto_dir, proto_file)
                    subprocess.run(['protoc', '--python_out=.', proto_path]
        # Now, proceed with the normal build process
        super().run()


setup(
    name='your_package',
    version='0.1',
    packages=find_packages(),
    include_package_data=True,
    cmdclass={
        'build_py': BuildProto,  # Hook to compile .proto files during the
    },
    install_requires=[
        'protobuf',  # Ensure protobuf is installed
    ],
)
```

In this setup:

- The `BuildProto` class extends the `build_py` command to add Protobuf compilation before the package is built.
- It runs `protoc --python_out=.` for each `.proto` file in the `proto` directory (you can change this to match your folder structure).

4. Create a `proto/` directory

Place all your `.proto` files in a `proto/` directory inside your project.

arduino Copy code

```
your_project/
├── proto/
│   └── example.proto
├── your_package/
├── setup.py
└── README.md
```

5. Install and build the package

When you install the package, the `.proto` files will automatically be compiled into Python classes. Use the following command to install your package in the development environment:

bash Copy code

```
pip install -e .
```

The Protobuf code will be generated as part of the installation process.

6. Using the Generated Code

After the package is installed, you can import and use the generated code like this:

python Copy code

```python
from your_package.example_pb2 import Person

person = Person(name="John Doe", id=1, email="john.doe@example.com")
print(person)
```

Notes:

- You may need to adjust the paths or flags depending on your system configuration, but this setup should give you a good starting point.
- If you're distributing this package and want others to install it, make sure you include the `proto/` directory and specify in your `MANIFEST.in` file that the `.proto` files should be included in the package.

ini Copy code

```
include proto/*.proto
```

This approach will ensure that Protobuf code is automatically generated as part of your package installation process!