Table of content

Auto-generating QObject from template in PySide...............1
The Auto-Generator function...1
How to use the Auto-Generator.. 2
Example usage of our generated class...............................2
The expected output..2

Auto-generating QObject from template in PySide

In the previous PySide tutorial, we have defined a QObject subclass “Car” with 4 properties which took about 36 lines of code. The code is repetitive, and it follows a scheme – every property has a getter, setter and notification method (in the case of the previous example, we shared the notification signal amongst all properties). For a very generic “entity” QObject, there is an easier way: Auto-generating the QObject from a template that specifies the names of the properties and the type of each property. Attributes, getters, setters and notification methods are automatically generated, and the object can be used as if it was a normal Python object.

The Auto-Generator function

This function gets one or more class_def tuples (length: two) that specify the name and the type of objects (see below for an example). The only optional keyword at the moment is “name”, which specifies the name of the class for debugging purposes.

```
from PySide import QtCore

def AutoQObject(*class_def, **kwargs):
    class Object(QtCore.QObject):
        def __init__(self, **kwargs):
            QtCore.QObject.__init__(self)
            for key, val in class_def:
                self.__dict__['_'+key] = kwargs.get(key, val())

        def __repr__(self):
            values = ('%s=%r' % (key, self.__dict__['_'+key]) \
                    for key, value in class_def)
            return '<%s (%s)>' % (kwargs.get('name', 'QObject'), ', '.join(values))

        for key, value in class_def:
            nfy = locals()['_nfy_'+key] = QtCore.Signal()

            def _get(key):
                def f(self):
                    return self.__dict__['_'+key]
                return f

            def _set(key):
                def f(self, value):
                    self.__dict__['_'+key] = value
                    self.__dict__['_nfy_'+key].emit()
                return f

            set = locals()['_set_'+key] = _set(key)
            get = locals()['_get_'+key] = _get(key)

            locals()[key] = QtCore.Property(value, get, set, notify=nfy)

    return Object
```

How to use the Auto-Generator

Taking the example of our previous tutorial, defining a “Car” object has become quite easy now:

```
Car = AutoQObject(
    ('model', str),
    ('brand', str),
    ('year', int),
    ('inStock', bool),
    name='Car'
)
```

The amount of lines has decreased from 36 to 7 (if you ignore the fact that AutoQObject does also take some lines). The definition is more straightforward, and it saves you a lot of typing and errors compared to if you were to do it manually.

Example usage of our generated class

We could simply drop this code into the previous tutorial example and see that it still works. We just do some basic instantiation and reading of the values to see if it works:

```
print Car

c = Car(model='Fiesta', brand='Ford', year=1337)
print c.model, c.brand, c.year, c.inStock
print c

c.inStock = True

print c.model, c.brand, c.year, c.inStock
print c
```

The expected output

If everything works as planned, this is what you should get as output when running the example code:

```
<class '__main__.Object'>
Fiesta Ford 1337 False
<Car (model='Fiesta', brand='Ford', year=1337, inStock=False)>
Fiesta Ford 1337 True
<Car (model='Fiesta', brand='Ford', year=1337, inStock=True)>
```