TrivialUI Documentation

Release 0.0.2

Tim Martin

December 10, 2016

Contents

TrivialUI is a Python library that aims to make it as easy as possible to make your Python code into GUI Python code. It's not intended as a full replacement for a GUI library like PySide; if you want full control so you can make the greatest interface, then TrivialUI won't give you what you want. But if you want a UI when you didn't even think you had time to create one, then this might be your thing.

At least, that's the plan. Currently it's at a very early stage of development.

CHAPTER 1

Table of contents

1.1 Examples

1.1.1 Application main function

If you want to get anything to display, you'll need to start an application like this:

```
import TrivialUI

if __name__ == '__main__':
    with TrivialUI.Application() as app:
        # Create your windows here...
        pass
```

1.1.2 Simple dashboard

To have a dashboard with some buttons:

```
def yo():
    print "yo"

class Dashboard(TrivialUI.MainWindow):
    widgets = [
        TrivialUI.Button("Yo", on_click=yo)
    ]

    def __init__(self):
        super(Dashboard, self).__init__(title='Dashboard')
```

Start the application like this:

```
with TrivialUI.Application():
    window = Dashboard()
    window.show()
```

The *on_click* of the button takes an ordinary Python callable.

1.2 Reference

class TrivialUI.**GenericProxy**(*data*, *children*, *parent=None*, *row=0*)
The proxy object makes a single piece of Python data navigable in a tree context. This is the base class that uses Template Method to allow various different sorts of Python data to be navigated in this way.

hasChild(*row*)
Check whether this dict has an entry for the specified row. In fact, this just checks whether the number of entries in the collection is sufficient.

class TrivialUI.**DictProxy**(*data*, *children*, *parent=None*, *row=0*)
Proxy object for making a dict of dicts navigable in a form usable by PyQt.

This gives nondeterministic ordering, unless you use an OrderedDict.

class TrivialUI.**ListModel**(*data*, *header=None*)
A model object that exposes the model interface that Qt expects, based on a bunch of data provided as (possibly nested) Python list objects.

The tree structure itself is built up inside of the ListProxy object stored as self.root_item. The tree is constructed on the fly as Qt queries using the index() method.

TrivialUI.**MainWindow**
alias of <Mock id='140131672847568'>

class TrivialUI.**TextEdit**(*name*)
Proxy class for creating a QLineEdit. This has to be a distinct class for two reasons:

- It's used in static initialisation when creating a MainWindow class, so it ends up being constructed before the Qt application is ready.
- Similarly, it's used statically where we actually want multiple instances to be created when the MainWindow is created.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

t

D

G

H

L

M

T