Yeahlowflicker Production

PRJ000 - Simple Text Editor

Table of Contents

Project ID: PRJ000
Project Name: Simple Text Editor

0 | Definitions and References

Qt:	A widget toolkit for creating graphical user interfaces
QML:	A user-interface markup language developed by Qt
PySide2:	A Python library that implements Qt

Variables and functions will be set to italic style for better understanding.

1 | Introduction

The project aims to build a simple text editing application with the following features:

- Basic text editing
- File saving/opening
- Switchable application themes
- Customizable display font size

The editor is purely a simple editor, therefore it does not support advanced text options such as font-weight options (e.g. bold, italic, underline...) and per-character font size options.

2 | Tools

The project is completed using the following tools and libraries:

- Python 3.8
- PyCharm Community
- Qt for Python (Pyside2)
- PyInstaller

PyCharm is the Python IDE used to develop the application. The Pyside2 library provides Qt GUI implementation, while PyInstaller handles the application export.

3 | Code Structure

The application contains the following scripts:

- Python:
    - bootstrap.py
    - main.py
    - lib.py

- QML:
    - main.qml
    - preferences.qml
    - unsavedWarning.qml

Bootstrap.py is the bootstrap program of the application.

Main.py is the management script that handles saving and loading. It contains functions that call functions in *lib.py* to perform saving and loading when requested by main.qml.

Lib.py is the code library with functions that take arguments and save/load to and from a specific file path.

Main.qml defines the user interface of the main window. When requested, this script will emit signals to *main.py* for storage manipulation.

Preferences.qml defines the window for modifying application preferences.

UnsavedWarning.qml defines the popup window that shows up when the currently opened file is not saved and the user tries to close the application.

4 | User Interface

4.1 | Main Interface

The main interface contains a toolbar with the following buttons:

- New
- Open
- Save
- Save As
- Preferences

It also contains the text area in which the user can edit the text file.

Figure 5.1a: Screenshot of main interface with light theme

Figure 5.1b: Screenshot of main interface with dark theme

4.2 | Preference Window

This window allows the user to modify the application settings. It consists of two options - theme and display font size. When an option is selected, the interface will be updated automatically by QML. If the user clicks the OK button, the preferences will be saved to an external file.

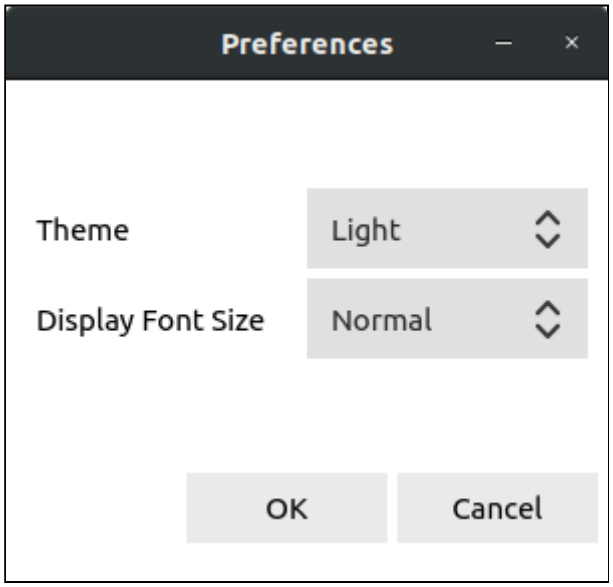

Figure 5.2a: Preference window (light theme)

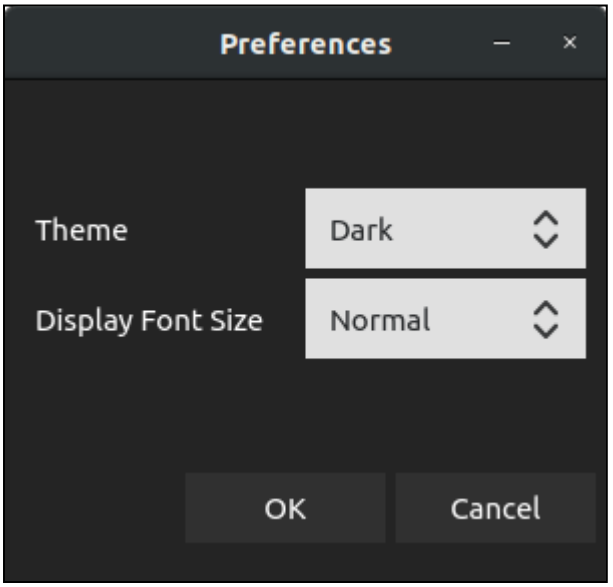

Figure 5.2b: Preference window (dark theme)

4.3 | Unsaved Warning

This is a warning dialog that pops up when the current file is not saved and the user attempts to close the application.

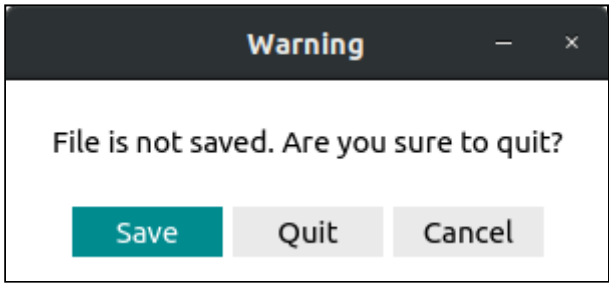

Figure 5.3a: Unsaved warning (light theme)

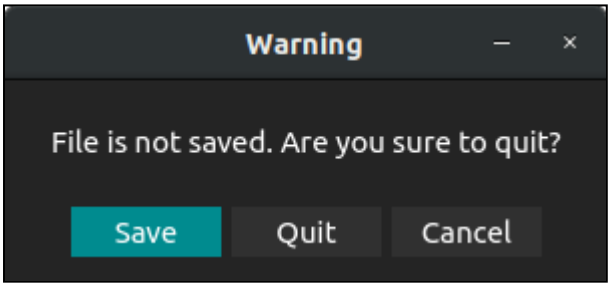

Figure 5.3b: Unsaved warning (dark theme)

5 | Problems

5.1 | Python Slot Return Values Undefined

When passing tuples from the Python slot, the QML script receives undefined return values. The followings are some examples:

@qtc.Slot(result=list)
@qtc.Slot(result=object)
@qtc.Slot(result=(bool, int))

This issue is fixed when switching to:

@qtc.Slot(result="QVariantList")

When using regular data types as the result, QML fails to read the data. QML uses variant as its property type, so passing the data as a QVariantList works because QML can read it.

5.2 | Sub-windows Opens Again And Again

When the user triggers a button that brings up a window, the system keeps making new sub-windows.

To limit this, the sub-window is set as a reference in *main.qml.*
Every time before instantiating a new sub-window, the system will check whether there is an existing reference. If not, the system will bring up a new sub-window, otherwise, it does nothing.

Moreover, when the sub-window is closed, it will set its reference in *main.qml* to null.

5.3 | Sub-windows Showing Below Of Main Window

The sub-windows will show underneath the main window, making it hidden and not accessible by the user.

To resolve this, the *raise()* function of the sub-window is called. This raises the window layer of the sub-window, making it appear on top of the main window.

6 | Conclusion

Making a simple text editor is an easy task with Qt/QML controls.

The tricky part is connecting signals and communication between Python and QML. Also, sub-window instantiation is confusing as well, as there is multiple information about the windows' positions, priority ,and references.

Overall the project is successful as all of the planned goals have been met.