Volume 2, Issue 4, April - 2024 **ISSN (E):** 2938-379X

OPTIMIZING PYQT5 DEVELOPMENT WITH QT DESIGNER

Soliev Bakhromjon Nabijonovich,
Senior Lecturer of the Fergana Branch of the Tashkent University of Information Technologies named after Muhammad al-Khorazmi,
bahromjonsoliev@gmail.com

G‘iyosiddinov Najmiddin,
Student of the Fergana Branch of the Tashkent University of Information Technologies named after Muhammad al-Khorazmi,
najmiddinweb@gmail.com

Abstract
This article explores the synergy between PyQt5 and Qt Designer in Python application development. PyQt5, a Python binding for the Qt framework, empowers developers to create feature-rich graphical user interfaces (GUIs) with ease. Qt Designer complements PyQt5 by providing a visual interface for designing UIs effortlessly. By integrating PyQt5 with Qt Designer, developers can streamline the UI design process, expedite prototyping, and enhance collaboration. This abstract sets the stage for delving into how PyQt5 and Qt Designer work together harmoniously to optimize development workflows and deliver polished applications efficiently.

Keywords: PyQt5, Qt Designer, Python, GUI development, User Interface, Integration, Streamlining, Prototyping, Collaboration, Workflow Optimization, Visual Design, Rapid Development, Modular Approach, Best Practices, Code Reusability, Event Handling, Signal-Slot Mechanism, Internationalization, Accessibility, Usability.

Introduction

In the realm of Python application development, crafting visually appealing and functionally robust graphical user interfaces (GUIs) stands as a pivotal task. PyQt5 emerges as a cornerstone in this domain, seamlessly integrating the powerful Qt framework with Python, thus offering developers an unparalleled toolkit for GUI design. Complementing PyQt5's prowess is Qt Designer, a versatile visual interface that facilitates the creation of dynamic UIs with remarkable ease.

At its core, PyQt5 empowers developers to leverage Qt's extensive widget library and event-driven architecture to build interactive and responsive applications. With PyQt5, developers can harness the power of Python's expressive syntax and object-oriented paradigm, coupled with Qt's robust features, to create applications that cater to a wide range of user needs.

However, the journey of GUI development does not end with PyQt5 alone. Enter Qt Designer—a graphical tool that complements PyQt5 by providing a visual interface for designing UIs effortlessly. Qt Designer allows developers to design and prototype UI layouts with a drag-and-drop interface, significantly reducing the time and effort required for manual coding. By seamlessly integrating PyQt5 with Qt Designer, developers can expedite the UI design process, iterate on designs rapidly, and achieve a level of visual fidelity that resonates with end-users.

In this article, we delve into the symbiotic relationship between PyQt5 and Qt Designer, exploring how these tools work together harmoniously to streamline the GUI development workflow. We'll discuss the seamless integration of PyQt5 with Qt Designer, highlighting the benefits of leveraging visual design tools in conjunction with Python's programming capabilities. Furthermore, we'll explore best practices for GUI development, including modular design principles, code reusability strategies, and techniques for optimizing workflow efficiency.

By the end of this article, readers will gain a comprehensive understanding of how PyQt5 and Qt Designer can be used in tandem to create polished, professional-grade GUI applications. Whether you're a seasoned developer looking to optimize your workflow or a newcomer eager to explore the world of GUI development, this article serves as a roadmap for harnessing the full potential of PyQt5 and Qt Designer in your Python projects.

Methods

Integration of PyQt5 and Qt Designer: Integrate PyQt5 with Qt Designer to leverage the visual design capabilities of Qt Designer while harnessing the power of PyQt5 for backend functionality. This integration allows developers to design UI layouts visually and seamlessly convert them into PyQt5 code for implementation.

Modular Design: Adopt a modular approach to GUI development by breaking down complex UIs into smaller, reusable components. Encapsulate these components into custom widgets or classes to promote code reusability and maintainability.

Rapid Prototyping: Utilize Qt Designer for rapid prototyping of UI designs. The drag-and-drop interface of Qt Designer allows developers to quickly iterate on UI layouts, experiment with different design options, and gather feedback early in the development process.

Signal-Slot Mechanism: Leverage PyQt5's signal-slot mechanism for efficient event handling and communication between UI elements. Define signals and slots to connect user interactions (e.g., button clicks, text input) with corresponding actions or functions in the application logic.

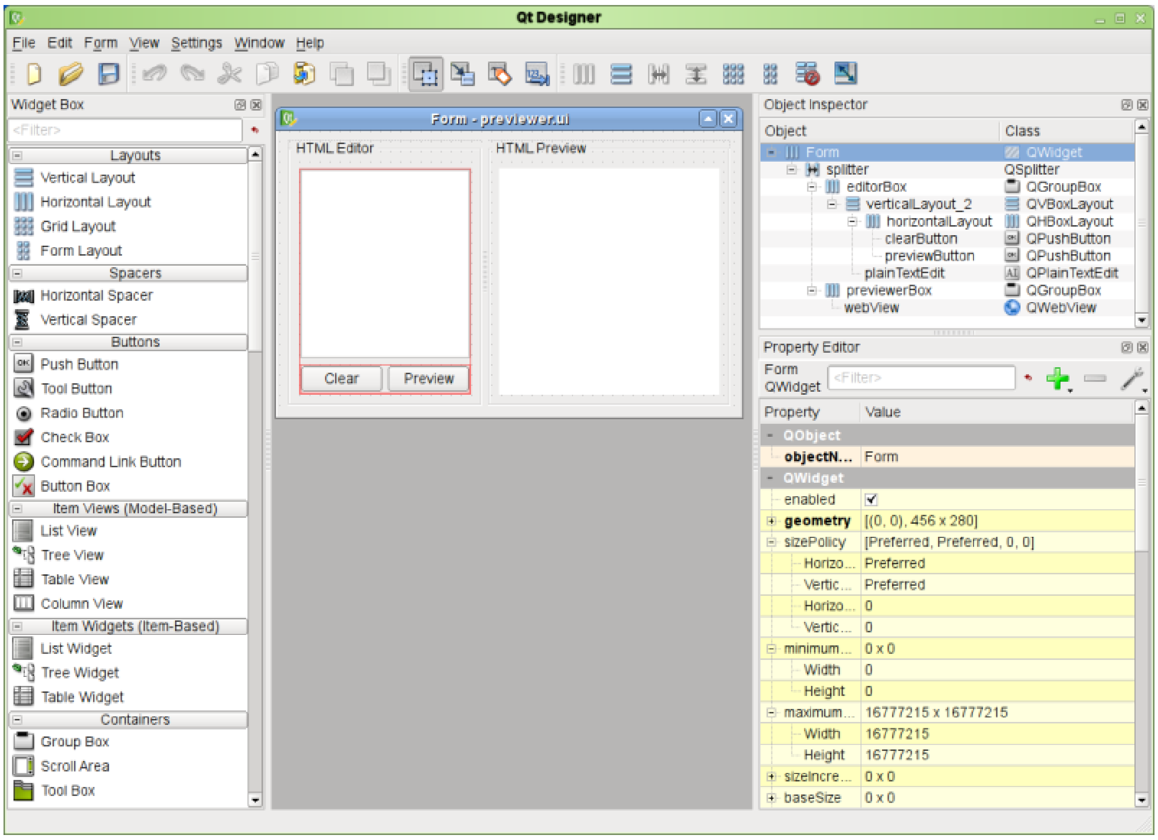
Results:

Efficient GUI Development: Integrating PyQt5 with Qt Designer allows for rapid prototyping and iteration of UI designs. Developers can create visually appealing layouts quickly, reducing the time and effort required for manual coding.

Modular and Reusable Code: Adopting a modular design approach and encapsulating UI components into custom widgets promotes code reusability and maintainability. Developers can reuse components across multiple projects, leading to more efficient development cycles and reducing redundancy.

Volume 2, Issue 4, April - 2024 **ISSN (E):** 2938-379X

Enhanced User Experience: Leveraging PyQt5's signal-slot mechanism for event handling and Qt Designer's intuitive interface for design enables developers to create responsive and interactive GUIs. This results in a more engaging user experience, with smooth navigation and seamless interaction.



Object	Class
Form	QWidget
splitter	QSplitter
editorBox	QGroupBox
verticalLayout_2	QVBoxLayout
horizontalLayout	QHBoxLayout
clearButton	QPushButton
previewButton	QPushButton
plainTextEdit	QPlainTextEdit
previewerBox	QGroupBox
webView	QWebView

Property	Value
QObject	
objectN…	Form
QWidget	
enabled	✓
geometry	[(0, 0), 456 x 280]
sizePolicy	[Preferred, Preferred, 0, 0]
Horizo…	Preferred
Vertic…	Preferred
Horizo…	0
Vertic…	0
minimum…	0 x 0
Width	0
Height	0
maximum…	16777215 x 16777215
Width	16777215
Height	16777215
sizeIncre…	0 x 0
baseSize	0 x 0

Fig.1 Interface of QT Designer

Code:

```
class window(QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.setGeometry(0, 0, 2400, 1000)
        tableResizeMode(self.tableWidget)
        tableResizeMode(self.tableWidget_2)
        tableResizeMode(self.tableWidget_3)
        tableResizeMode(self.tableWidget_4)
        tableResizeMode(self.tableWidget_5)
        tableResizeMode(self.tableWidget_6)
        tableResizeMode(self.tableWidget_7)
        tableResizeMode(self.tableWidget_8)
        tableResizeMode(self.tableWidget_9)
        self.addproductwidget = None
        self.tableWidget_2.setEditTriggers(QtWidgets.QTableWidget.NoEditTriggers)
        self.sotuv = {} # Dictionary tartib name: (List(book id, number))
```

Volume 2, Issue 4, April - 2024 **ISSN (E):** 2938-379X

```
self.curr_sotuv = "Sotuv 1"
self.prev_sotuv = 0
self.setCentralWidget(self.tabWidget)
self.handle_tabbar_clicked(0)
try: self.tabWidget.tabBarClicked.disconnect(self.handle_tabbar_clicked)
except: pass
self.tabWidget.tabBarClicked.connect(self.handle_tabbar_clicked)
self.label_change()
qmenu = self.menuBar()
qmenu.setStyleSheet("""
    QMenuBar::item:selected {
    background-color: blue;
    color: white;
}
    QMenuBar::item {
    border: 1px solid blue;
    padding: 5px;
}
""")

# menus
menubar = self.menuBar()
file_menu = menubar.addMenu('Funksiyalar')

# Creating QAction for each item
qidirish = QAction('Qidirish                F3', self)
sotish = QAction('Sotish                    F4', self)
# big = QAction('Kattalashtirish          F5', self)
# small = QAction('Kichiklashtirish         F6', self)

# Connecting triggered signal of each QAction to a function
qidirish.triggered.connect(self.Key_F3_function)
sotish.triggered.connect(self.F4_func_control)
# big.triggered.connect(self.make_big)
# small.triggered.connect(self.make_small)

# Adding actions to the menu
file_menu.addAction(qidirish)
file_menu.addAction(sotish)
# file_menu.addAction(big)
# file_menu.addAction(small)
```

```
        shortcut = QKeySequence(Qt.Key_F3)
        self.shortcut = QShortcut(shortcut, self)
        self.shortcut.activated.connect(self.Key_F3_function)

        shortcut = QKeySequence(Qt.Key_F4)
        self.shortcut = QShortcut(shortcut, self)
        self.shortcut.activated.connect(self.F4_func_control)
        self.Key_F3_function()
```

Listing 1. Sample code

Conclusions

The integration of PyQt5 with Qt Designer offers a powerful toolkit for Python developers to streamline GUI development workflows and deliver high-quality applications efficiently. By leveraging PyQt5's robust features and Qt Designer's intuitive visual interface, developers can expedite the UI design process, iterate on designs rapidly, and create visually appealing and functionally rich applications.

Through methods such as modular design, rapid prototyping, and efficient event handling, developers can optimize their PyQt5 development workflow, leading to more efficient code reuse, enhanced user experience, and improved collaboration among team members. Additionally, incorporating internationalization and accessibility features ensures that GUI applications reach a diverse audience and comply with accessibility standards.

Furthermore, version control practices, thorough testing, and clear documentation contribute to the overall success of PyQt5 projects by promoting collaboration, ensuring code quality, and facilitating knowledge sharing. By adhering to best practices and leveraging the full capabilities of PyQt5 and Qt Designer, developers can unlock the full potential of these tools and deliver polished, professional-grade GUI applications.

In essence, PyQt5 and Qt Designer empower developers to create visually stunning, responsive, and feature-rich applications that meet the needs and expectations of end-users effectively. With a focus on efficiency, collaboration, and quality, PyQt5 projects can thrive in today's competitive software development landscape, setting new standards for innovation and user experience.

References:

1. Ermatova Z. АЛЬТЕРНАТИВНЫЕ ПОДХОДЫ К ОБРАБОТКЕ ОШИБОК: СРАВНЕНИЕ EXCEPTIONS И STD:: EXPECTED В C++ //Потомки Аль-Фаргани. – 2023. – Т. 1. – №. 2. – С. 67-73.
2. Kayumov A. et al. PYTHON DASTURLASH TILIDA RASMLAR BILAN ISHLASH. PILLOW MODULI //Research and implementation. – 2023.
3. Zulunov R., Soliev B. Importance of Python language in development of artificial intelligence //Потомки Аль-Фаргани. – 2023. – Т. 1. – №. 1. – С. 7-12.

4. Nabijonovich S. B. et al. UNVEILING THE FUTURE OF DATA EXTRACTION USING PYTHON AND AI FOR VIDEO-BASED INFORMATION RECOGNITION //American Journal of Technology and Applied Sciences. – 2023. – T. 17. – C. 26-32.
5. Zulunov R. et al. Detecting mobile objects with ai using edge detection and background subtraction techniques //E3S Web of Conferences. – EDP Sciences, 2024. – T. 508. – C. 03004.
6. Akhundjanov U. et al. Off-line handwritten signature verification based on machine learning //E3S Web of Conferences. – EDP Sciences, 2024. – T. 508. – C. 03011.
7. PyQt5 Documentation. (riverbankcomputing.com/static/Docs/PyQt5/)
8. Qt Designer Documentation. (doc.qt.io/qt-5/qtdesigner-manual.html)
9. "Rapid GUI Programming with Python and Qt" by Mark Summerfield. Prentice Hall.
10. "PyQt6 By Example" by Nicholas J.C. Ralston. Packt Publishing.
11. "Learning Python Design Patterns" by Gennadiy Zlobin. Packt Publishing.
12. "Python GUI Programming Cookbook" by Burkhard Meier. Packt Publishing.
13. "Python GUI Programming with Tkinter" by Alan D. Moore. Packt Publishing.
14. "Qt5 C++ GUI Programming Cookbook" by Lee Zhi Eng. Packt Publishing.
15. "Modern Python Cookbook" by Steven Lott. Packt Publishing.

Web of Teachers: Inderscience Research
webofjournals.com/index.php/1