**Working with Advanced Widgets in PySide6**

**(TreeView, TableView, ComboBox)**

PySide6 provides several advanced widgets for building complex GUI applications. In this guide, we will cover three essential widgets:

1. **QTreeView** – Used for displaying hierarchical data.
2. **QTableView** – Provides a tabular data representation with a model-view architecture.
3. **QComboBox** – A dropdown menu widget for selecting predefined options.

**1. QTreeView (Hierarchical Data Display)**

**Example: Creating a Simple Tree View**

from PySide6.QtWidgets import QApplication, QTreeView, QFileSystemModel

import sys

app = QApplication(sys.argv)

# Create a file system model

model = QFileSystemModel()

model.setRootPath("") # Root directory (current filesystem)

# Create a tree view and set the model

tree = QTreeView()

tree.setModel(model)

tree.setRootIndex(model.index(".")) # Set to the current directory

tree.setWindowTitle("QTreeView Example")

tree.resize(600, 400)

tree.show()

sys.exit(app.exec())

**Key Features:**

* Uses **QFileSystemModel** to display the file system in a hierarchical format.
* setRootPath("") initializes the model with the system’s root.
* The QTreeView is connected to the model to display directory structures.

**2. QTableView (Displaying Tabular Data)**

**Example: Creating a Table View**

from PySide6.QtWidgets import QApplication, QTableView

from PySide6.QtGui import QStandardItemModel, QStandardItem

import sys

app = QApplication(sys.argv)

# Create a data model

model = QStandardItemModel(4, 3) # 4 rows, 3 columns

model.setHorizontalHeaderLabels(["Name", "Age", "Country"])

# Populate table with data

data = [

("Alice", 25, "USA"),

("Bob", 30, "UK"),

("Charlie", 28, "Canada"),

("David", 35, "Germany"),

]

for row, (name, age, country) in enumerate(data):

model.setItem(row, 0, QStandardItem(name))

model.setItem(row, 1, QStandardItem(str(age)))

model.setItem(row, 2, QStandardItem(country))

# Create a table view and set the model

table = QTableView()

table.setModel(model)

table.setWindowTitle("QTableView Example")

table.resize(400, 300)

table.show()

sys.exit(app.exec())

**Key Features:**

* Uses **QStandardItemModel** for storing tabular data.
* Data is added row by row using setItem().
* QTableView is used to display the model.

**3. QComboBox (Dropdown Selection)**

**Example: Creating a ComboBox**

from PySide6.QtWidgets import QApplication, QWidget, QVBoxLayout, QComboBox, QLabel

import sys

class ComboBoxExample(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("QComboBox Example")

self.resize(300, 200)

layout = QVBoxLayout()

self.label = QLabel("Select an option:")

layout.addWidget(self.label)

self.combo = QComboBox()

self.combo.addItems(["Python", "C++", "Java", "JavaScript"])

self.combo.currentIndexChanged.connect(self.on\_selection\_change)

layout.addWidget(self.combo)

self.setLayout(layout)

def on\_selection\_change(self, index):

self.label.setText(f"Selected: {self.combo.itemText(index)}")

app = QApplication(sys.argv)

window = ComboBoxExample()

window.show()

sys.exit(app.exec())

**Key Features:**

* **QComboBox** is used to create a dropdown list.
* **Signals & Slots**: currentIndexChanged.connect() triggers an event when a selection changes.
* The selected item updates the **QLabel** dynamically.

**Conclusion**

* **QTreeView** is ideal for **hierarchical data**, such as file systems.
* **QTableView** is perfect for **tabular data** representation.
* **QComboBox** is useful for **dropdown selections**.