**Event Handling and Signal-Slot Mechanism in PySide6**

QT's **Signal-Slot** mechanism is an **event-driven communication system** that allows widgets to communicate efficiently. It replaces traditional callback functions, making the code cleaner and more modular.

**1. Understanding Events and Signals-Slots in PySide6**

**What is an Event?**

An **event** occurs when the user interacts with the application (e.g., clicking a button, resizing a window, moving the mouse).  
Examples:

* Mouse click (mousePressEvent)
* Key press (keyPressEvent)
* Window resize (resizeEvent)

**What is a Signal?**

A **signal** is emitted when a specific event occurs (e.g., button clicked). PySide6 provides predefined signals like:

* clicked() → Emitted when a button is clicked.
* textChanged() → Emitted when text in a QLineEdit is changed.

**What is a Slot?**

A **slot** is a function that executes when a signal is emitted. It **"receives"** the signal and performs an action.

**2. Basic Example: Connecting a Button Click to a Function**

import sys

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

class SignalSlotExample(QWidget):

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

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

self.setWindowTitle("PySide6 Signal-Slot Example")

self.setGeometry(200, 200, 300, 200)

# QLabel

self.label = QLabel("Click the button", self)

# QPushButton

self.button = QPushButton("Click Me", self)

self.button.clicked.connect(self.on\_button\_click) # Connecting signal to slot

# Layout

layout = QVBoxLayout()

layout.addWidget(self.label)

layout.addWidget(self.button)

self.setLayout(layout)

def on\_button\_click(self):

""" Slot function that runs when button is clicked """

self.label.setText("Button Clicked!")

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

window = SignalSlotExample()

window.show()

sys.exit(app.exec())

**Explanation**

* self.button.clicked.connect(self.on\_button\_click):
  + **clicked()** is a predefined **signal**.
  + **on\_button\_click** is the **slot function** that updates the label text.
* Clicking the button updates the QLabel text dynamically.

**3. Custom Signals and Slots**

You can also create **custom signals** using Signal from PySide6.QtCore.

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

from PySide6.QtCore import Signal, QObject

import sys

class Communicate(QObject):

custom\_signal = Signal(str) # Custom signal with a string parameter

class CustomSignalExample(QWidget):

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

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

self.setWindowTitle("Custom Signal Example")

self.setGeometry(200, 200, 300, 200)

# QLabel

self.label = QLabel("Waiting for signal...", self)

# QPushButton

self.button = QPushButton("Send Signal", self)

# Custom Signal

self.comm = Communicate()

self.comm.custom\_signal.connect(self.update\_label) # Connecting custom signal to slot

# Connecting button click to emit signal

self.button.clicked.connect(lambda: self.comm.custom\_signal.emit("Custom Signal Received!"))

# Layout

layout = QVBoxLayout()

layout.addWidget(self.label)

layout.addWidget(self.button)

self.setLayout(layout)

def update\_label(self, message):

""" Slot function that updates label text when signal is received """

self.label.setText(message)

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

window = CustomSignalExample()

window.show()

sys.exit(app.exec())

**Explanation**

* **Signal(str)**: Creates a **custom signal** that sends a string.
* **self.comm.custom\_signal.connect(self.update\_label)**: Connects the signal to the update\_label slot.
* **self.comm.custom\_signal.emit("Custom Signal Received!")**: Emits the custom signal when the button is clicked.

**4. Event Handling Example (Mouse and Keyboard)**

Qt allows overriding event handlers to detect mouse and keyboard interactions.

import sys

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

from PySide6.QtGui import QMouseEvent, QKeyEvent

class EventHandlingExample(QWidget):

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

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

self.setWindowTitle("Event Handling Example")

self.setGeometry(200, 200, 400, 300)

# QLabel to display events

self.label = QLabel("Interact with the window...", self)

# Layout

layout = QVBoxLayout()

layout.addWidget(self.label)

self.setLayout(layout)

def mousePressEvent(self, event: QMouseEvent):

""" Detects mouse click and updates label """

self.label.setText(f"Mouse clicked at ({event.x()}, {event.y()})")

def keyPressEvent(self, event: QKeyEvent):

""" Detects key press and updates label """

self.label.setText(f"Key Pressed: {event.text()} (Key Code: {event.key()})")

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

window = EventHandlingExample()

window.show()

sys.exit(app.exec())

**Explanation**

* **mousePressEvent(self, event)**:
  + Detects a mouse click and **retrieves the x, y coordinates**.
* **keyPressEvent(self, event)**:
  + Detects a key press and **displays the key code**.

**5. Using Lambda Functions in Signal-Slot Connections**

Instead of defining a separate function, **lambda functions** can be used to handle signals.

self.button.clicked.connect(lambda: self.label.setText("Button Clicked!"))

* This makes the code **more concise**.

**6. Disconnecting a Signal**

You can **disconnect a signal** from a slot using .disconnect().

self.button.clicked.disconnect()

* This stops the button from triggering the slot.

**Summary**

* **Signal-Slot Mechanism**:
  + **Signal** = Emits an event (e.g., button click).
  + **Slot** = Function that responds to the event.
* **Predefined Signals**: PySide6 provides built-in signals (e.g., clicked()).
* **Custom Signals**: You can define **custom signals** using Signal().
* **Event Handling**:
  + **Override event handlers** (mousePressEvent, keyPressEvent) for direct event processing.
* **Lambda in Signals**: Use lambda for inline event handling.