# QT6 + ??????????

Платная лицензия Похоже надо углубиться в лицензирование opensource.

Страницы компонентов

| | | |
|---|---|---|
| Label | Push button | Radio button |
| Line edit | Check box | SpinBox |
| QLCD | ComboBox | Slider |
| ListWidget | Table | Calendar |
| ColorDialog | FontDialog | MessageBox |
| Dialogs (save) | | |

Установка:

```
pip install pyqt6
pip install pyqt6-tools
```

Минимальное приложение:

```
from PyQt6.QtWidgets import QApplication, QWidget
import sys

app = QApplication(sys.argv)
window = QWidget()
window.show()
sys.exit(app.exec())
```

Архитектура QT прячется под стандартную, но это не так. QT основывается на цикле событий, внутри реализованы используемые системные процедуры (таймер, ...), и приходится использовать соответствующие QT-модули, а не системные модули. Поэтому в составе QT много модулей.

| Модуль | Назначение |
|---|---|
| QtWidgets | Основной и шаблонные виджеты (окно, метка, ...) |

| Модуль | Назначение |
| --- | --- |
| QtGui | Классы для интеграции с оконной системой, обработки событий, 2D-графики, базовых изображений, шрифтов, иконок и текста.<br><br>QIcon класс работы с иконками |
| QtCore | Системные модули. |

### Типы окон

**QMainWindow** Главное окно приложения и связанные с ним классы для управления главным окном.

```
from PyQt6.QtWidgets import QApplication, QMainWindow
import sys

app = QApplication(sys.argv)
window = QMainWindow()
window.statusBar().showMessage("Welcome to pyqt6 coding")
window.show()
sys.exit(app.exec())
```

QMainWindow имеет свой собственный макет, содержащий QToolBars, QDockWidgets, QMenuBar и QStatusBar.

**QDialog** Базовый класс окна верхнего уровня, используемое для краткосрочных задач и краткого общения с пользователем.
QDialogs может быть модальным или немодальным.

**QWidget** Базовый класс всех объектов пользовательского интерфейса, получает мышь, клавиатуру и другие события
из оконной системы и отображает свое изображение на экране.

### Объектно-ориентированный подход настройки окна

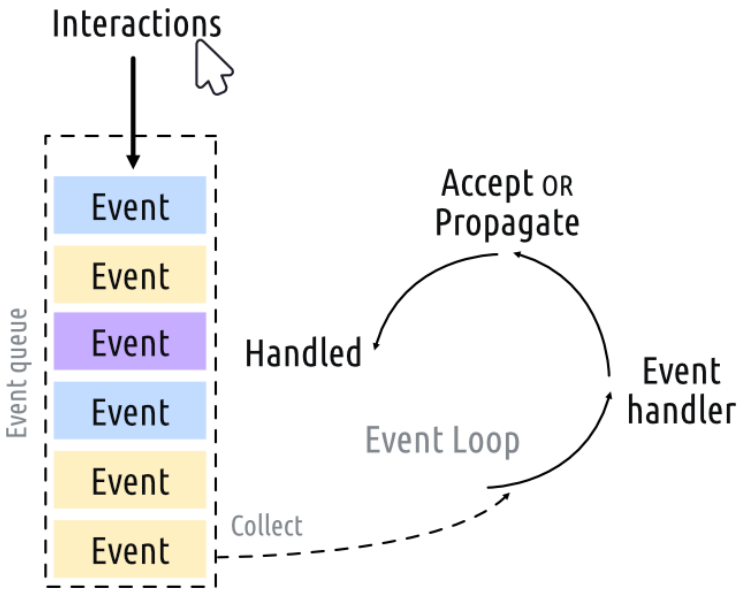
Создаем класс-потомок например от QWidget или QMainWindow, настраиваем свойства и

```
from PyQt6.QtWidgets import QApplication, QWidget
from PyQt6.QtGui import QIcon
import sys

class Window(QWidget):
    def __init__(self):
        super().__init__()
```

```
        self.setGeometry(200,200, 700, 400)
        self.setWindowTitle("Python GUI Development")
        self.setWindowIcon(QIcon('pyqt6lessons\images\python.png'))
        self.setStyleSheet('background-color:green')
        self.setWindowOpacity(0.5)

app = QApplication(sys.argv)
window = Window()
window.show()
sys.exit(app.exec())
```

**Управление событиями**

Основной элемент всех приложений в Qt — класс QApplication. Каждому приложению нужен только один объект QApplication, который содержит цикл событий приложения. Это основной цикл, управляющий всем взаимодействием пользователя с графическим интерфейсом.




При каждом взаимодействии с приложением генерируется событие, которое помещается в очередь событий. В цикле событий очередь проверяется на каждой итерации: если найдено ожидающее событие, оно вместе с управлением передаётся определённому обработчику этого события. Последний обрабатывает его, затем возвращает управление в цикл событий и ждёт новых событий. Для каждого приложения выполняется только один цикл событий.

Класс QApplication содержит цикл событий Qt (нужен один экземпляр QApplication). Приложение ждёт в цикле событий новое событие, которое будет сгенерировано при выполнении действия. Всегда выполняется только один цикл событий.

Сигналы — уведомления, отправляемые виджетами, когда что-то происходит. Это может быть нажатие кнопки, изменение текста в поле ввода, изменение текста в окне, ... Многие

сигналы инициируются в ответ на действия пользователя, но не только: в сигналах могут отправляться данные с дополнительным контекстом произошедшего.

Слоты — приёмники сигналов. Слотом можно сделать любую функцию (или метод), просто подключив к нему сигнал. Принимающая функция получает данные, отправляемые ей в сигнале. У многих виджетов Qt есть встроенные слоты, эти виджеты можно подключать друг к другу напрямую.

```
class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setGeometry(200,200, 700, 400)
        self.setWindowTitle("Python GUI Development")
        self.create_button()

    def create_button(self):
        btn = QPushButton("Click", self)
        btn.clicked.connect(self.the_button_was_clicked)

    def the_button_was_clicked(self):
        print("Clicked")
```

Соединение сигнала и слота происходит в функции btn.clicked.connect(self.the_button_was_clicked) Таблицы событий:

Мышь:

| Тип | Описание |
|---|---|
| `MouseButtonPress` | Нажата кнопка мыши |
| `MouseButtonRelease` | Отпущена кнопка мыши |
| `MouseButtonDblClick` | Двойной клик |
| `MouseMove` | Движение мыши |
| `Wheel` | Колёсико мыши |
| `Enter` | Курсор вошёл в виджет |
| `Leave` | Курсор покинул виджет |
| `HoverEnter` | Hover вошёл |
| `HoverMove` | Hover движение |
| `HoverLeave` | Hover вышел |

Клавиатура:

| Тип | Описание |
|---|---|
| `KeyPress` | Нажата клавиша |
| `KeyRelease` | Отпущена клавиша |
| `Shortcut` | Сработал shortcut |
| `ShortcutOverride` | Попытка перехвата shortcut |
| `InputMethod` | IME ввод |
| `InputMethodQuery` | Запрос IME |

### Фокус и активация

| Тип | Описание |
|---|---|
| `FocusIn` | Получен фокус |
| `FocusOut` | Потерян фокус |
| `ActivationChange` | Изменение активности окна |

### Окна и виджеты

| Тип | Описание |
|---|---|
| `Show` | Виджет показан |
| `Hide` | Виджет скрыт |
| `Close` | Закрытие |
| `Resize` | Изменение размера |
| `Move` | Перемещение |
| `Paint` | Перерисовка |
| `LayoutRequest` | Запрос layout |
| `UpdateRequest` | Запрос обновления |
| `Polish` | Финальная инициализация |
| `PolishRequest` | Запрос polish |
| `ParentChange` | Изменился родитель |
| `ParentAboutToChange` | Родитель изменится |
| `WindowStateChange` | Изменение состояния окна |
| `WindowActivate` | Окно активировано |
| `WindowDeactivate` | Окно деактивировано |
| `WindowTitleChange` | Заголовок окна |
| `WindowIconChange` | Иконка окна |

| Тип | Описание |
| --- | --- |
| `WindowBlocked` | Окно заблокировано |
| `WindowUnblocked` | Окно разблокировано |

### Геометрия и экран

| Тип | Описание |
| --- | --- |
| `ScreenChangeInternal` | Изменился экран |
| `ScreenChangeInternal` | DPI/Screen изменился |
| `OrientationChange` | Смена ориентации |
| `DevicePixelRatioChange` | Изменение DPR |

### Drag & Drop

| Тип | Описание |
| --- | --- |
| `DragEnter` | Drag вошёл |
| `DragMove` | Drag перемещение |
| `DragLeave` | Drag покинул |
| `Drop` | Drop |

### Буфер обмена

| Тип | Описание |
| --- | --- |
| `Clipboard` | Изменился буфер обмена |

### Таймеры

| Тип | Описание |
| --- | --- |
| `Timer` | Сработал таймер |
| `ZeroTimerEvent` | Таймер с нулевой задержкой |

### Touch / Tablet / Gesture

| Тип | Описание |
| --- | --- |
| `TouchBegin` | Touch начало |
| `TouchUpdate` | Touch обновление |
| `TouchEnd` | Touch конец |
| `TabletPress` | Перо нажато |

| Тип | Описание |
| --- | --- |
| `TabletMove` | Перо движение |
| `TabletRelease` | Перо отпущено |
| `Gesture` | Жест |
| `GestureOverride` | Перехват жеста |

### Состояние

| Тип | Описание |
| --- | --- |
| `EnabledChange` | Изменение enabled |
| `FontChange` | Изменение шрифта |
| `StyleChange` | Изменение стиля |
| `PaletteChange` | Изменение палитры |
| `LanguageChange` | Смена языка |
| `LocaleChange` | Смена локали |
| `ThemeChange` | Смена темы |
| `ApplicationStateChange` | Состояние приложения |

### Продвинутые опции

| Тип | Описание |
| --- | --- |
| `DynamicPropertyChange` | Изменение свойства |
| `ChildAdded` | Добавлен ребёнок |
| `ChildRemoved` | Удалён ребёнок |
| `ChildPolished` | Ребёнок отполирован |
| `MetaCall` | Вызов meta-object |
| `ThreadChange` | Смена потока |
| `DeferredDelete` | Отложенное удаление |
| `Quit` | Завершение приложения |
| `PlatformSurface` | Изменение поверхности |
| `PlatformPanel` | Platform panel |
| `User` | Начало пользовательских событий |

### Пользовательские события

| Тип | Описание |
| --- | --- |

| | |
|---|---|
| `User` | Базовый пользовательский event |
| `MaxUser` | Максимальный ID |

```
event = QEvent(QEvent.Type.User)
QCoreApplication.postEvent(obj, event)
```

Поиск событий:

```
def event(self, event):
    print(event.type())
    return super().event(event)

#или
widget.installEventFilter(self)

def eventFilter(self, obj, event):
    print(obj, event.type())
    return False
```

---

Revision #30
Created 27 January 2026 08:33:38 by Admin
Updated 10 February 2026 16:05:25 by Admin