UNIVERSITY OF LIVERPOOL

Application Development with C++ (ELEC362)

Lecture 16: Event handling, QActions and context menus

mihasan@liverpool.ac.uk

Previous lecture

- Further uses of Qt Designer were discussed.
- Formatting the lay out of the application and controlling its sizing were discussed.
- Controlling the style of a Widget was discussed.
- Adding resource files to an application was explained.
- The Signal-Slot mechanism was discussed and it was contrasted with connecting objects using variables.
- Classes introduced: **`QLayout, QStyle, QHBoxLayout, QVBoxLayout, QGridLayout`**

This lecture

- What is covered in this lecture?

  1. Event handling in Qt.   2. Introducing multiple to achieve a task in an application.

- Why it is covered?

  1. Even handling extends the capability of the app beyond what pre-defined widgets can do.

  2. Having multiple ways to achieve a task in an application makes applications user friendly.

- How are topics covered in this lecture: Live demonstration

What is an Event?

- In simple terms, an event is a user action through an input device.
- In more advanced terms, an event is a change in the state of an object.
- In Qt, there is a class for all types of events, all of which are subclasses from **`QEvent`**.

Event classes	What it describes
`QKeyEvent`	A key input from the keyboard by the user
`QMouseEvent`	A mouse click by the user
`QTouchEvent`	A touch on a smartphone’s screen or a tablet

- Unlike other classes, objects of event-based classes are constructed by the physical action of the user. Example: https://doc.qt.io/qt-5/qmouseevent.html
- Enumerators and flags are heavily used in event classes: https://doc.qt.io/qt-5/qt.html

What is an Event handler?

- An event handler is a function that is executed when a particular event occurs.
- Event handlers are typically member functions of the widgets receiving the event.
- The class **`QWidget`**, being the parent of all UI classes has a lot of virtual event handlers (https://doc.qt.io/qt-5/qwidget.html#protected-functions).
- Most of event handlers have the word "event" in their name.
- The input parameter of an event handler is an object of a class event.
- Example:

```
// In a cpp file:
MyWidget::mousePressEvent(QMouseEvent *event){};
```

Mouse event object accessed by pointer

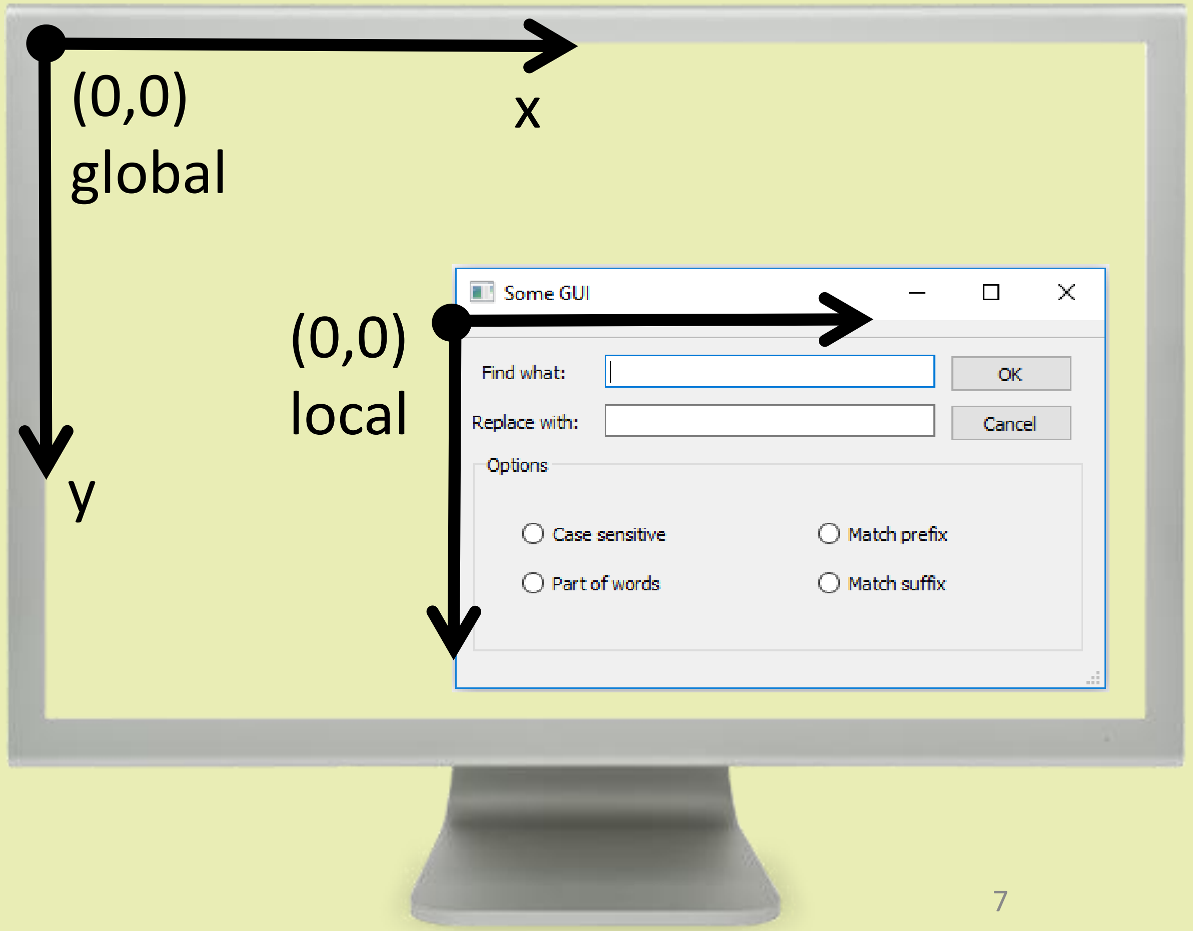
Event handler for **`QWidget`** class

A sample application: Mouse locator

- The task: Design and implement a Qt based GUI application that displays the coordinates at which the mouse is clicked in an empty rectangle.
- Applying the incremental model is an exercise for you!!
- To implement a virtual function in a class, it must be defined in the header file with a name that is identical to the documentation.
- Don’t forget to add the header file of the class if it is not already defined in Qt Designer.
- Build the application (demonstration).

`QMouseEvent` Class

- This class describes a mouse-clicking event by the user.
- Reference: https://doc.qt.io/qt-5/qmouseevent.html
- The event records which button was clicked and where (in local coordinates and in global coordinates).



Completion list in Qt creator

- When coding in Qt creator, one can get a completion list showing suggestions of what you type. It helps writing faster and reducing compiling mistakes.
- It can be shown by pressing Ctrl + Spacebar
- See the list of displayed symbols here: https://doc.qt.io/qtcreator/creator-completing-code.html
- For a list of keyboard shortcuts:

https://doc.qt.io/qtcreator/creator-keyboard-shortcuts.html

Actions

- The concept of events and event handling is universal to all GUI development libraries, the concept of Actions is specific to Qt.
- In any application there are multiple methods to do a certain task (through a menu, a toolbar, right click, etc..).
- Instead of implementing codes to do the same thing for every method, the code can be implemented once as an “action”, then all menus and toolbars simply link to that action.
- QUESTION: How many ways can one save a document in MS Word?

`QAction` Class

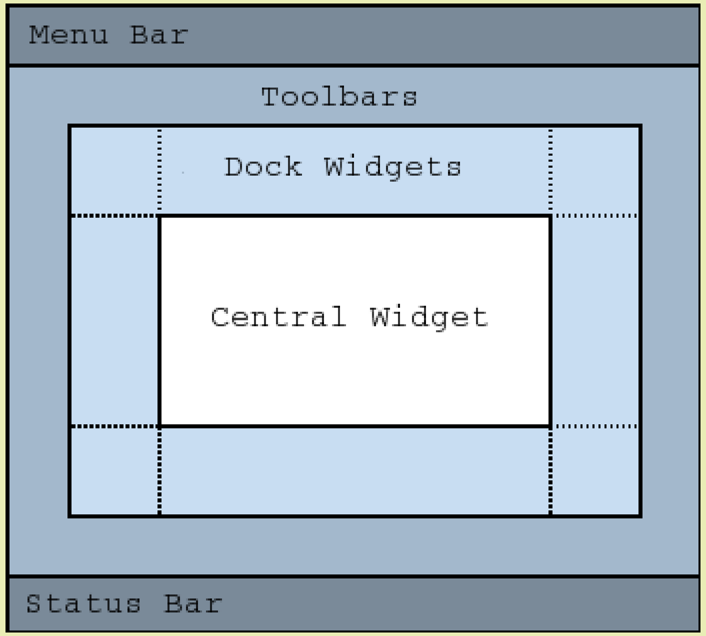
- This class defines a command that can be invoked by multiple widgets.
- Reference: https://doc.qt.io/qt-5/qaction.html
- Just like other classes in Qt, every **`QAction`** property has a setter and getter function.
- Every object of the class **`QAction`** can have an icon attached to it (using resource files).
- The functionality of the Action can be implemented by implementing the function **`triggered()`**. Such that whenever the action is called, this function is executed.
- Objects of the class **`QAction`** are automatically created by Qt Designer when the chosen class of the application is **`QMainWindow.`**

A sample application: Text formatting app

- The task: Design and implement a Qt based GUI application that allows the user to type paragraphs, change their font size and colour in the application. The app should have at least 3 different methods to change of the colour and the size of the text.
- Applying the incremental model is an exercise for you!!
- We choose **`QTextEdit`** object over **`QPlainTextEdit`** because it allows formatting.
- Build the application (demonstration).

QMainWindow revisited

- Any window based on **QMainWindow** class has a default layout, which can be activated from the constructor.



Good practice note:

When developing an application that relies on menus and toolboxes, use **QMainWindow** as the base class for the application.

`QMenu` Class

- This class constructs menus in the application.
- Reference: https://doc.qt.io/qt-5/qmenu.html
- The menus can belong to a menu bar, or they can be context menus (right-click menus).
- Menus can be nested (parent menu opens child menus)
- Qt Designer creates menus very easily when using **`QMainWindow`**, for every item in the menu, an action (object of **`QAction`**) is generated automatically.
- In Qt, menus do nothing on their own, they merely link the user to an action.

`QContextMenuEvent` Class

- This class constructs context menus (right-click menu).
- Reference: https://doc.qt.io/qt-5/qcontextmenuevent.html
- This class is a special case from **`QMouseEvent`** optimised for creating context menus.
- It is a virtual event handler for all classes derived from **`QWidget`**.
- The context menu must be constructed by code (not doable from Qt Designer).
- To create the menu, we need to create a function to construct the menu, and call it in the constructor.

Good practice note: For good code organisation create functions to construct menus and call them in the constructor

Create a child window

- Up to this point we only developed one-window applications.
- To create a child window, a class has to be defined representing the child window. The child window is displayed when an object of the class is constructed.
- Most of the time, the child window has to be of the **`QDialog`** type.
- To pass data from the **`QDialog`** object, the function **`exec()`** has to be used.
- Pass the data from the local variables in the <u>child</u> window to the local variables in the <u>parent</u> window.

Summary

- In this lecture, the concept of events and event handling was discussed.
- The coordinates systems in Qt were discussed.
- The concept of actions has been discussed.
- Constructing menus in menu bars, toolbars and context menus was shown
- Creating a child window in an application was shown.
- Classes introduced: **`QEvent, QKeyEvent, QMouseEvent, QTouchEvent, QAction, QMenu, QContextMenuEvent, QTextEdit`**