

# Qt Signals and Slots

Iuliana Bocicor  
*maria.bocicor@ubbcluj.ro*

Babes-Bolyai University

2023

# Overview

## Qt Signals and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- 1 Signals and slots
- 2 User defined signals and slots
- 3 Qt Designer
- 4 Notepad example
- 5 Drawing with Qt
- 6 Other useful Qt classes

# Callbacks I

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- When we change one widget, we often want another widget to be notified. E.g.:
  - when the close button is pressed on a main window in an application, all application related windows should be closed;
  - when a value is chosen in a combobox, a list should be populated with different values.
- In other toolkits, this kind of communication is achieved using *callbacks*. These are functions that are triggered in response to specific events or user actions.
- Callbacks play a crucial role in event-driven programming, where the flow of the program is driven by user interactions.

# Callbacks II

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- A callback is function that is called by another function, when an event happens.
- If you want a processing function to notify you about some event you pass a pointer to another function (the callback) to the processing function.
- The processing function then calls the callback when appropriate.
- E.g.
  - Comparison function passed to a sorting algorithm.
  - A progress bar object, which could be used by any client.

# Callbacks III

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

## Example - progress bar

- The prototype of the callback function is defined by the developer of the progress bar.

```
//callback function prototype
typedef void(*ProgressListener)(int percent);

//function that notifies progress using callback
void someComputation(ProgressListener callback)
{
    for (int i = 0; i < 100; i++)
    {
        cout << "Doing some computation\n";
        callback(i); //notification after each
                     step
    }
}
```

# Callbacks IV

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

```
#include <functional>

//function that notifies progress using callback
void someComputation(std::function<void(int)>
    callback)
{
    for (int i = 0; i < 100; i++)
    {
        //do stuff
        cout << "Doing some computation\n";
        callback(i); // notification after each
                     step
    }
}
```

# Callbacks V

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- The client code (which needs the notifications) will have to define a function with that specific prototype.

```
void onProgress(int percent)
{
    std::cout << "progress:" << percent;
}

int main()
{
    someComputation(onProgress);
}
```

# Callbacks VI

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- Callbacks in c++ have two fundamental disadvantages:
  - if there are several notifications needed, we either need separate callback functions, or we could use generic parameters (`void*`), which cannot be verified at compile-time.
  - the processing function is coupled to the callback function (it needs to know its signature, its parameters).



# Signals and slots I

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- Qt signals and slots are an alternative to the callback mechanism.
- They are used for communication between objects.
- The signals and slots mechanism is a central feature of Qt.

# Signals and slots II

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- A **signal** is emitted when a particular event occurs (e.g. a button is clicked).
- Qt widgets have many pre-defined signals which are emitted to indicate that a user action or a change of state has occurred.
- A **slot** is a function that is called in response to a particular signal.
- A signal can be connected to a function (called a slot), so that when the signal is emitted, the slot is automatically executed.

# Signals and slots III

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- The signature of a signal must match the signature of the receiving slot (in fact, a slot may have a shorter signature than the signal it receives because it can ignore extra arguments) ⇒ **the signals and slots mechanism is type safe.**
- Slots can be used for receiving signals, but they are also normal member functions.
- Just as an object does not know if anything receives its signals, a slot does not know if it has any signals connected to it ⇒ truly independent components can be created with Qt and there is a **loose coupling between signals and slots.**

# Signals and slots IV

## Qt Signals and Slots

Iuliana  
Bocicor

## Signals and slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- You can connect as many signals as you want to a single slot, and a signal can be connected to as many slots as you need.
- Qt widgets have many predefined signals, but we can subclass widgets to add new signals to them.
- Qt widgets have many pre-defined slots.
- It is common practice to subclass widgets and add your user-defined slots that can handle the signals.

# Signals and slots V

## Qt Signals and Slots

Iuliana  
Bocicor

## Signals and slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

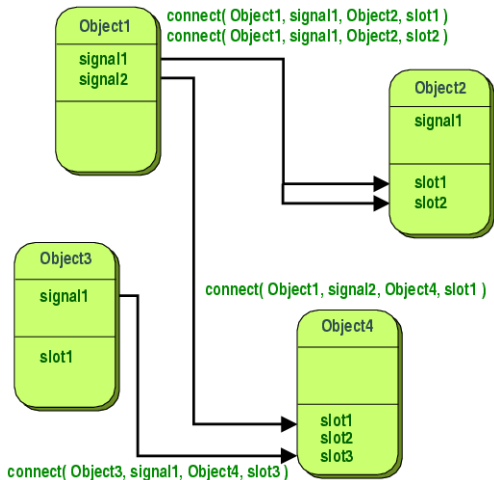


Figure source: [Qt Documentation - Signals Slots](#)

# Connecting signals and slots I

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- A slot is called when a signal connected to it is emitted. Slots are normal C++ functions and can be called normally; their only special feature is that signals can be connected to them.
- Several slots can be connected to one signal, the slots will be executed one after the other, in the order they have been connected, when the signal is emitted.

# Connecting signals and slots II

- To connect a signal and a slot we use the function `QObject::connect` and (optionally) the macros `SIGNAL` and `SLOT`.

```
QPushButton *closeButton = new QPushButton("Close");  
// Qt6, Qt5  
QObject::connect(closeButton, &QPushButton::clicked,  
                 QApplication::instance(), &QApplication::quit);  
  
// older Qt versions  
QObject::connect(closeButton, SIGNAL(clicked()),  
                 QApplication::instance(), SLOT(quit()));
```

# Spin and slider example I

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

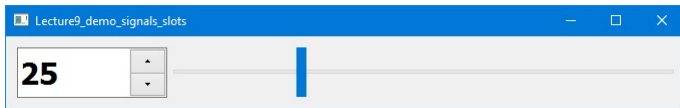
User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes



If the the user changes the value in the Spin Box:

- The spinbox will **emit the signal** `valueChanged(int)` with an int argument, the current value of the spinner.
- Because the spinner and slider are connected, the `setValue(int)` method of the slider will be invoked. The argument from the `valueChanged` method (the current value of the spinner) will be passed to the `setValue` method of the slider.



# Spin and slider example II

- The slider updates itself to reflect the new value, and emits a `valueChanged` signal because its own value has changed.
- Because the slider is also connected with the spinner, the `setValue` slot of the spinner is invoked, in response to the slider's signal.
- The `setValue` of the spinner will not emit any signal because the current value is the same as the value provided in the `setValue` (prevents infinite loop).

## DEMO

Simple example of connecting signals and slots (*Lecture10\_demo\_spin\_slider*).

# Subclassing QWidget

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- Create a separate class for the GUI that you are designing.
- This class must inherit from `QWidget`.
- Make it an independent, self-contained component, with its own signals and slots.

# The meta-object system

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- Qt introduces a mechanism called the **meta-object system** which provides two key services:
  - signals and slots;
  - introspection
    - the ability to introspect objects at runtime;
    - the the methods and properties of an object can be listed at runtime;
    - is necessary for implementing signals and slots;
    - allows application programmers to obtain "meta-information" about `QObject` subclasses at run-time, including the list of signals and slots supported by the object and its class name.

# The meta-object compiler I

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- The meta-object compiler (MOC) is the tool that provides introspection support.
- It is a code generator - parses the header files and generates an additional C++ file that is compiled with the rest of the program.
- The meta-object compiler takes all classes starting with the `Q_OBJECT` macro and generates a `moc_*.cpp` C++ source file.
- This file contains information about the class being moc-ed such as class name, inheritance tree and also the names and pointers to the signal and slot members.

# The meta-object compiler II

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- This means that emitting a signal is actually calling a function generated by the MOC.
- Macros used by the MOC:
  - `signals`
  - `slots`
  - `Q_OBJECT`
  - `emit`
  - `SIGNAL`
  - `SLOT`
- "We strongly recommend that all subclasses of `QObject` use the `Q_OBJECT` macro regardless of whether or not they actually use signals, slots, and properties." (Qt documentation: <https://doc.qt.io/qt-6/metaobjects.html>)

# The meta-object compiler III

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

## DEMO

Simple example of introspection (*Lecture10\_introspection*).

# Custom slots I

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- The syntax in Qt6 allows us to connect signals to any member function of QObject, not only slots.
- If connections are made via the **SIGNAL** and **SLOT** macros, then slots have to be declared using the **slots** keyword.
- A slot is just a regular method and the compiler will handle it as any other function.
- Using a signal-slot connection, slots can be invoked by any component.
- A signal emitted from an instance of class A can cause a private slot to be invoked in an instance of class B, even if A and B are unrelated.

# Custom signals I

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- Custom signals can be defined using the `signals` macro.

```
signals :  
    void genesUpdatedSignal();  
    void addGeneSignal(const std::string&  
        geneName, const std::string&  
        organismName, const std::string&  
        sequence);
```

- Signals can be emitted by an object when its internal state has changed in some way that might be interesting to another object.
- They can never have return types (use `void`).



# Custom signals II

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- The `emit` macro is used to emit signals.

```
void GenesGUI::deleteGene()  
{  
    // ...  
    // emit the signal: the genes were updated  
    emit genesUpdatedSignal();  
}
```

- Signals are public access functions and can be emitted from anywhere, but it is recommended to only emit them from the class that defines the signal and its subclasses.
- When a signal is emitted, the slots connected to it are usually executed immediately, just like a normal function call.

# Custom signals III

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- Execution of the code following the `emit` statement will occur once all slots have returned.
- If several slots are connected to one signal, the slots will be executed one after the other, in the order they have been connected.

## DEMO

Gene manager (*Lecture10\_genes*).

# Qt Designer I

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- Qt Designer is the Qt tool for designing and building GUIs.
- Windows and dialogs can be designed in a what-you-see-is-what-you-get manner.
- Objects can be dragged from the widget box and dropped on the form.
- Object properties can be modified interactively.

# Qt Designer II

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- Using the Qt Designer can be faster than hand-coding the interface.
- One can experiment with different designs quickly.
- A **.ui** file is created, representing the widget tree of the form in XML format. The *User Interface Compiler (uic)* can then be used to create a corresponding C++ header file.

## DEMO

Qt Designer (*Lecture10\_qt\_designer*).

# When should we write code programatically?

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- When the elements in the dialog must change dynamically.
- When we want to use custom widgets.

## How?

- 1 Create a new class, by inheriting from `QWidget`.
- 2 Implement the GUI.
- 3 Show the newly created widget.

# QMainWindow

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- A main window provides a framework for building an application's user interface.
- QMainWindow has its own layout to which you can add:
  - A menu bar (on the top): `QMenuBar;`
  - Toolbars: `QToolBar;`
  - A central widget: this will typically be a `QTextEdit` or a `QGraphicsView;`
  - A status bar (on the bottom) : `QStatusBar.`

# Using the menu bar

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- `QMainWindow` provides the function `menuBar()`, which allows adding `QMenus` to the menu bar and adding `QActions` to the pop-up menus.
- `QAction` can be used for common commands can be invoked via menus, toolbar buttons, and keyboard shortcuts.

```
QMenu* fileMenu = this->menuBar()->addMenu("&File");  
QAction *openAction = new QAction("&Open", this);  
fileMenu->addAction(openAction);
```

# Using the tool bar

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- `QToolBar` provides a movable panel that contains a set of controls.
- Toolbar buttons are added by adding actions, using the function `addAction`.

```
QToolBar* fileToolBar = addToolBar("&File");  
fileToolBar->addAction(openAction);  
fileToolBar->addAction(saveAction);
```



# Notepad example

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

**Notepad  
example**

Drawing with  
Qt

Other useful  
Qt classes

## DEMO

Notepad example (*Lecture10\_notepad*).

# QPainter I

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- **QPainter** is used for low-level painting on custom defined widgets.
- It can draw simple and complex shapes (from lines to pies).
- The **QPen** object, which defines how a painter should draw lines and outlines of shapes.
- The **QBrush** defines the fill pattern of shapes drawn by a QPainter.
- The **QPainterPath** is an object composed of building blocks such as rectangles, ellipses, lines.

# QPainter II

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- The `paintEvent` method (of the `QWidget` class) is invoked when the `QWidget` needs to repaint all or part of the widget.
- The `keyPressEvent` method is invoked when a key is pressed (**only if** the widget has the focus).
- The `mousePressEvent` method is invoked when a mouse key is pressed.

## DEMO

Drawing example (*Lecture10 - Drawing*).

## QString

- Unicode character string.
- The functions that have a `QString` as parameter will accept a `const char*`.
- There are methods to convert a `QString` in an `std::string` and viceversa.

```
QString s = "145";  
std::string ss = s.toStdString();  
QString s2 = QString::fromStdString(ss);
```

# Useful Qt classes II

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- There are methods to convert a `QString` in a number and viceversa.

```
QString s = "145";  
int a = s.toInt();  
double d = 3.8;  
QString s3 = QString::number(d);
```

# Useful Qt classes III

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

## QVector

- This is a template class that provides a dynamic array.
- It stores its items in adjacent memory locations and provides fast index-based access.
- It offers operations similar to the STL vector.
- It can be converted to an STL vector and viceversa (function `toStdVector`, `fromStdVector`).

## QMessageBox

- Provides a modal dialog to show information.
- It can display text, an item and standard buttons (OK, Cancel, Open, Close, Save, Discard, etc.) for user response.
- There are 4 message types, which really only differ in the predefined icon they each show:
  - Question
  - Information
  - Warning
  - Critical

# Useful Qt classes V

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

```
QMessageBox::information(this, "Info", "Selection  
    changed");  
// ...  
int ret = QMessageBox::warning(this, "My Application  
    ", "The document has been modified.Do you want  
    to save your changes?", QMessageBox::Save |  
    QMessageBox::Discard | QMessageBox::Cancel ,  
    QMessageBox::Save);  
if (ret == QMessageBox::Save) {  
    //do save  
}
```



# Useful Qt classes VI

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

## QDebug

- Provides an output stream for debugging information.
- It is useful to call the `QDebug()` function to obtain a default `QDebug` object to use for writing debugging information.
- In VS, the message will be displayed in the "Output" window.

```
QDebug() << "Here is a message for debug.";
```

# Summary

Qt Signals  
and Slots

Iuliana  
Bocicor

Signals and  
slots

User defined  
signals and  
slots

Qt Designer

Notepad  
example

Drawing with  
Qt

Other useful  
Qt classes

- Signals and slots are used for communication between objects.
- A signal is emitted when a particular event occurs.
- A slot is a function called in response to a particular signal.
- Signals and slots must be connected.