

Qt keretrendszer

Qt nyelvi könyvtárai,
a Qt programok fordítása,
és bevezetés az eseménykezelésbe



Code less.
Create more.
Deploy everywhere.

A Qt egy platformfüggetlen alkalmazás-fejlesztési keretrendszer, amellyel nemcsak asztali, de akár mobil vagy beágyazott alkalmazások is fejleszthetők.

- Lehetőséget ad az eseményvezérelt grafikus felhasználói felületű alkalmazások fejlesztésére, adatbázisok kezelésére, multimédiás szoftverek készítéséhez, 3D grafika alkalmazására, hálózati és webes kommunikáció megvalósítására.
- Rendelkezik nyílt forráskódú (LGPL) és kereskedelmi verzióval is.
- Fejlesztő nyelve elsősorban a C++, de más nyelvekre (pl. Python) is elérhető, sőt rendelkezik saját leíró nyelvvel (Qt Quick) is.
- A Qt 5.x keretrendszer, valamint QtCreator fejlesztőkörnyezet (benne a QtDesigner) elérhető a <https://www.qt.io/> oldalról.
 - Linux esetén: `apt-get install qt-sdk`

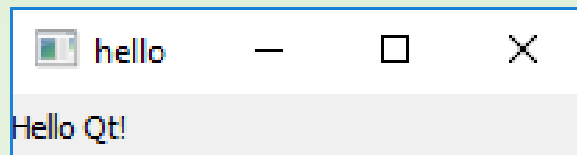
Segédanyag

- ❑ Qt : <https://doc.qt.io>
- ❑ Jasmin Blanchette, Mark Summerfield: C++ GUI programming with Qt4, Prentice Hall, 2006. ISBN 0-13-187249-4
- ❑ Lee Zhi Eng: Hands-On GUI Programming with C++ and Qt5 – Build stunning cross-platform applications and widgets with the most powerful GUI framework, Packt Publishing, 2016. ISBN 978-1-78646-712-6
- ❑ Lee Zhi Eng: Qt5 C++ GUI Programming Cookbook – Design and build a functional, appealing, and user-friendly graphical user interface, Packt Publishing, 2018. ISBN 978-1-78328-027-8
- ❑ Guillaume Lazar, Robin Penea: Mastering Qt5 – Master application development by writing succinct, robust, and reusable code with Qt5 Packt Publishing, 2018. ISBN 978-1-78839-782-7

Fejlesztés nyelve és környezete

- ❑ A fejlesztés C++/Qt nyelven.
 - Elérhető C++ a teljes utasításkészlete, nyelvi könyvtárai.
 - A standard C++ nyelv kiterjesztését Qt-s makrók biztosítják, amelyeket a *Meta Object Compiler (MOC)* fordít le ISO C++ kódra.
 - Speciális osztályokat biztosít, amelyek között számos C++ nyelven megszokott osztálynak Qt-s megfelelője (QString, QQueue, stb.) is megtalálható.
- ❑ Az alapértelmezett fejlesztőeszköz a *QtCreator*, de más környezetekben is lehetőség van a Qt fejlesztésre (pl. *Code::Blocks*, *Visual Studio*).
- ❑ Külön tervezőprogram (*QtDesigner*) áll rendelkezésre a grafikus felület létrehozásához, amelynek kimenete a felület XML nyelvű leírása.

“Hello Qt” alkalmazás



```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv); // alkalmazás példányosítása
    QLabel myLabel("Hello Qt!"); // címke a felirattal
    myLabel.show(); // címke megjelenítése
    return app.exec(); // alkalmazás futtatása
}
```

létrehozza az események
kezelését végző környezetet

megjeleníti egy önálló ablakban a címkét

elindítja az alkalmazást

Eseményvezérelt grafikus felületű alkalmazás

- Eseményvezérelt grafikus felületű alkalmazásnak nevezzük azt a programot, amely 2D-s interaktív felhasználói felületen (*GUI, Graphical User Interface*) keresztül kommunikál a felhasználóval, de más forrásból származó (pl.: egy időzítő, vagy egy másik alkalmazás) jelzésre is reagál.
 - A működését a felhasználói interakcióra történő várakozás jellemzi, és a programfutásba történő beavatkozáshoz egy konzolablaknál lényegesen változatosabb interakciós lehetőséget biztosít.
 - A felület grafikus megjelenítéssel ellátott ún. **vezérlők**ből (*control/widget*) áll (pl.: nyomógombok, listák, menük, stb.).
 - Egy grafikus vezérlő megjelenhet önállóan – **ablakként** (*form/window*) –, vagy egy másik vezérlő részeként. Több ablak esetén mindig van egy aktív ablak, és egy aktív vezérlő (ezen van a **fókusz**).

Példák GUI alkalmazásokra

ablakcím

vezérlő (táblanézet)

vezérlők

ablakfelület (rajzolva)

fejléc

vezérlőben lévő vezérlő (szám beállító)

vezérlő (nyomógomb)

fókuszált vezérlő (nyomógomb)

| | Track | Duration |
|---|---|----------|
| 1 | The Flying Dutchman: Overture | 10:30 |
| 2 | The Flying Dutchman: Wie aus der Fern laengst vergangner Zeiten | 06:14 |
| 3 | The Flying Dutchman: Steuermann, lass die Wacht | 02:32 |
| 4 | Die Walkuere: Ride of the Valkyries | 04:46 |
| 5 | Tannhaeuser: Freudig begruessen wir die edle Halle | 06:24 |
| 6 | Tannhaeuser: Wie Todesahnung - O du mein holder Abendstern | 04:17 |

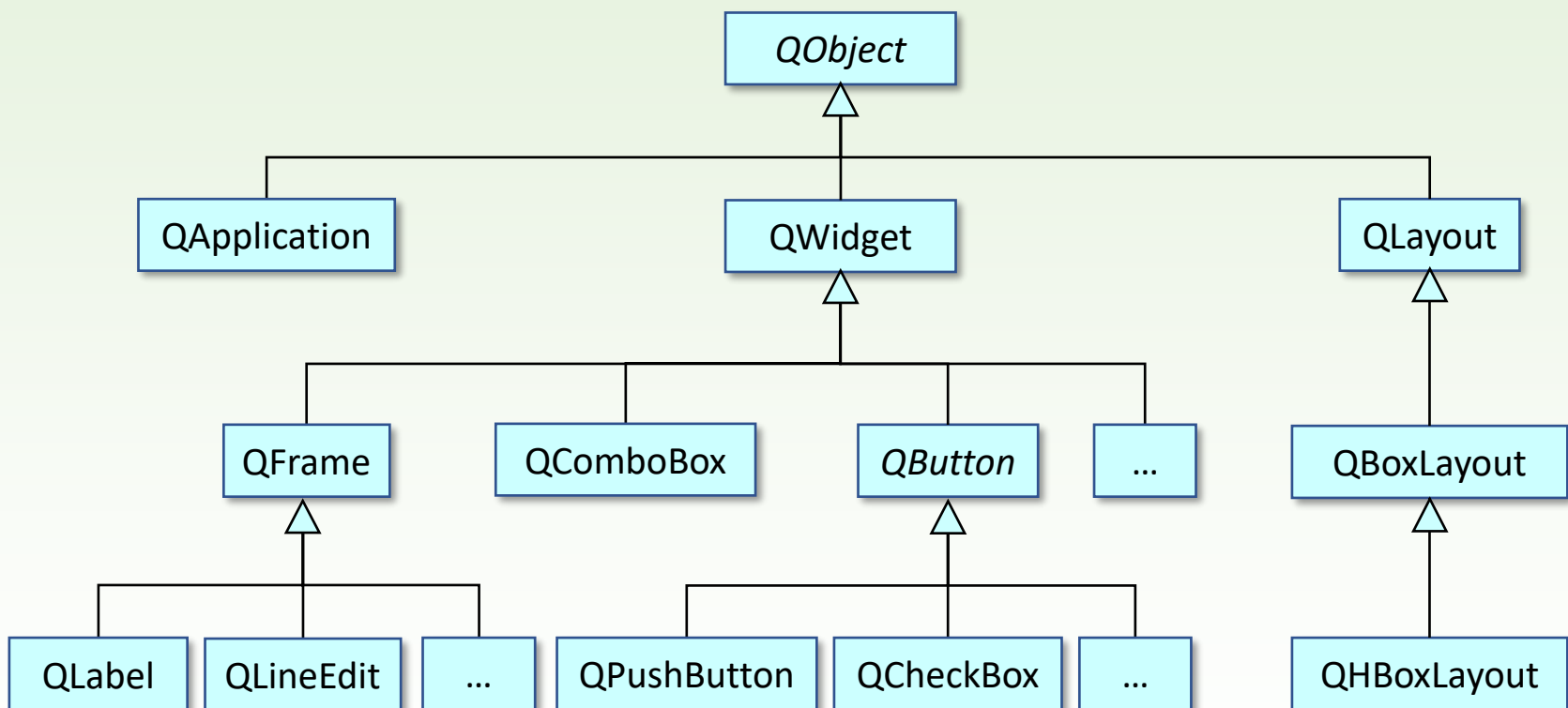
Objektumorientált grafikus felület

- ❑ Minden vezérlő egy objektum, amelyet az osztálya jellemez.
- ❑ Mivel a különféle vezérlők sok hasonló tulajdonsággal bírnak, így osztályaik **öröklődési hierarchiába** szervezhetők.
 - A Qt keretrendszer nyelvi könyvtárában találjuk az általános vezérlő osztályt, amelyből nevezetes vezérlők osztályai származnak.
 - Ha egyedi vezérlőre van szükségünk, akkor annak osztályát nekünk kell származtatással definiálni.
- ❑ Egy adott alkalmazás vezérlői **elhelyezkedési hierarchiába** rendeződnek aszerint, hogy egy vezérlő önállóan, azaz *ablakként* jelenik meg, vagy egy másik vezérlő részeként.

Qt osztálykönyvtár

- ❑ A Qt nyelvi könyvtár osztályai mind a `QObject` őssztály leszármazottjai. A `QObject` példányok nem másolhatók, ezért jórész mutatók és referenciák segítségével kezeljük őket
- ❑ Számos osztállyal rendelkezik:
 - grafikus megjelenéssel rendelkező nevezetes vezérlők osztályai, amelyek az általános `QWidget` osztály leszármazpottai
 - adatszerkezetek osztályai (`QVector`, `QStack`, `QLinkedList`, ...)
 - fájlok és fájlrendszerek kezelésének osztályai (`QFile`, `QTextStream`, `QDir`, ...)
 - adatbázis kezelés osztályai (`QSqlDataBase`, `QSqlQuery`, `QSqlResult`, `QSqlQueryModel`, `QSqlTableModel`)
 - párhuzamosság és aszinkron végrehajtást támogató osztályok (`QThread`, `QSemaphore`, `QFuture`, ...)

Vezérlők osztályhierarchiája



Vezérlők osztályai

```

QObject
+ staticMetaObject :QMetaObject [readOnly]
# d_ptr :QScopedPointer<QObjectData>
# staticQtMetaObject :QMetaObject [readOnly]

+ QObject(QObject*)
+ ~QObject()
+ event(Event*) :bool
+ eventFilter(QObject*, QEvent*) :bool
+ tr(char*, char*, int) :QString
+ trUtf8(char*, char*, int) :QString
+ metaObject() :QMetaObject * [query]
+ tr(char*, char*, int) :QString
+ tr(char*, char*) :QString
+ trUtf8(char*, char*, int) :QString
+ trUtf8(char*, char*) :QString
+ objectName() :QString [query]
+ setObjectName(QString&) :void
+ isWidgetType() :bool [query]
+ signalsBlocked() :bool [query]
+ blockSignals(bool) :bool
+ thread() :QThread * [query]
+ moveToThread(QThread*) :void
+ startTimer(int) :int
+ killTimer(int) :void
+ findChild(QString&) :T [query]
+ findChildren(QString&) :QList<T> [query]
+ findChildren(QRegExp&) :QList<T> [query]
+ children() :QObjectList & [query]
+ setParent(QObject*) :void
+ installEventFilter(QObject*) :void
+ removeEventFilter(QObject*) :void
+ connect(QObject*, char*, QObject*, char*, Qt::ConnectionType) :bool
+ connect(QObject*, QMetaMethod, QObject*, QMetaMethod, Qt::ConnectionType) :bool
+ connect(QObject*, char*, char*, Qt::ConnectionType) :bool [query]
+ disconnect(QObject*, char*, QObject*, char*) :bool
+ disconnect(QObject*, QMetaMethod, QObject*, QMetaMethod) :bool
+ disconnect(char*, QObject*, char*) :bool
+ disconnect(QObject*, char*) :bool
+ dumpObjectTree() :void
+ dumpObjectInfo() :void
+ setProperty(char*, QVariant&) :bool
+ property(char*) :QVariant [query]
+ dynamicPropertyName() :QList<QByteArray> [query]
+ registerUserData() :uint
+ setUserData(uint, QObjectUserData*) :void
+ userData(uint) :QObjectUserData * [query]
+ destroyed(QObject*) :void
+ parent() :QObject * [query]
+ inherits(char*) :bool [query]
+ deleteLater() :void
+ sender() :QObject * [query]
+ senderSignalIndex() :int [query]
+ receivers(char*) :int [query]
+ timerEvent(QTimerEvent*) :void
+ childEvent(QChildEvent*) :void
+ customEvent(QEvent*) :void
+ connectNotify(char*) :void
+ disconnectNotify(char*) :void
+ QObject(QObjectPrivate&, QObject*)
    
```

```

QWidget QPaintDevice
+ setEnabled(bool) :void
+ setDisabled(bool) :void
+ setWindowModified(bool) :void
+ frameGeometry() :QRect [query]
+ geometry() :QRect & [query]
+ normalGeometry() :QRect [query]
+ x() :int [query]
+ y() :int [query]
+ pos() :QPoint [query]
+ frameSize() :QSize [query]
+ size() :QSize [query]
+ width() :int [query]
+ height() :int [query]
+ rect() :QRect [query]
+ childrenRect() :QRect [query]
+ childrenRegion() :QRegion [query]
+ minimumSize() :QSize [query]
+ maximumSize() :QSize [query]
+ minimumWidth() :int [query]
+ minimumHeight() :int [query]
+ maximumWidth() :int [query]
+ maximumHeight() :int [query]
+ setMinimumSize(QSize&) :void
+ setMinimumSize(int, int) :void
+ setMaximumSize(QSize&) :void
+ setMaximumSize(int, int) :void
+ setMinimumWidth(int) :void
+ setMaximumWidth(int) :void
+ setMinimumHeight(int) :void
+ setMaximumHeight(int) :void
+ setupUi(QWidget*) :void
+ sizeIncrement() :QSize [query]
+ setSizeIncrement(QSize&) :void
+ setMinimumIncrement(int, int) :void
+ baseSize() :QSize [query]
+ setBaseSize(QSize&) :void
+ setBaseSize(int, int) :void
+ setFixedSize(QSize&) :void
+ setFixedSize(int, int) :void
+ setFixedSize(int) :void
+ ...()
    
```

```

«enumeration»
RenderFlag
DrawWindowBackground = 0x1
DrawChildren = 0x2
IgnoreMask = 0x4
    
```

```

«enumeration»
InsertPolicy
NoInsert
InsertAtTop
InsertAtCurrent
InsertAtBottom
InsertAfterCurrent
InsertBeforeCurrent
InsertAlphabetically
    
```

```

«enumeration»
SizeAdjustPolicy
AdjustToContents
AdjustToContentsOnFirstShow
AdjustToMinimumContentsLength
AdjustToMinimumContentsLengthWithIcon
    
```

```

QComboBox
+ QComboBox(QWidget*)
+ ~QComboBox()
+ maxVisibleItems() :int [query]
+ setMaxVisibleItems(int) :void
+ count() :int [query]
+ setMaxCount(int) :void
+ maxCount() :int [query]
+ autoComplete() :bool [query]
+ setAutoCompletionCaseSensitivity(Qt::CaseSensitivity) :void
+ setAutoCompletionCaseSensitivity(Qt::CaseSensitivity) :void
+ duplicatesEnabled() :bool [query]
+ setDuplicatesEnabled(bool) :void
+ setFrame(bool) :void
+ hasFrame() :bool [query]
+ findText(QString&, Qt::MatchFlags) :int [query]
+ findData(QVariant&, int, Qt::MatchFlags) :int [query]
+ insertPolicy() :InsertPolicy [query]
+ setInsertPolicy(InsertPolicy) :void
+ sizeAdjustPolicy() :SizeAdjustPolicy [query]
+ setSizeAdjustPolicy(SizeAdjustPolicy) :void
+ minimumContentsLength() :int [query]
+ setMinimumContentsLength(int) :void
+ iconSize() :QSize [query]
+ setIconSize(QSize&) :void
+ isEditable() :bool [query]
+ setEditable(bool) :void
+ setLineEdit(QLineEdit*) :void
+ lineEdit() :QLineEdit * [query]
+ setValidator(QValidator*) :void
+ validator() :QValidator * [query]
+ setCompleter(QCompleter*) :void
+ completer() :QCompleter * [query]
+ itemDelegate() :QAbstractItemDelegate * [query]
+ setItemDelegate(QAbstractItemDelegate*) :void
+ model() :QAbstractItemModel * [query]
+ setModel(QAbstractItemModel*) :void
+ rootModelIndex() :QModelIndex [query]
+ setRootModelIndex(QModelIndex&) :void
+ modelColumn() :int [query]
+ setModelColumn(int) :void
+ currentIndex() :int [query]
+ currentText() :QString [query]
+ itemText(int) :QString [query]
+ itemIcon(int) :QIcon [query]
+ itemData(int, int) :QVariant [query]
+ addItem(QIcon&, QString&, QVariant&) :void
+ addItems(QStringList&) :void
+ insertItem(int, QString&, QVariant&) :void
+ insertItem(int, QIcon&, QString&, QVariant&) :void
+ insertItems(int, QStringList&) :void
+ ...()
    
```

Modulok

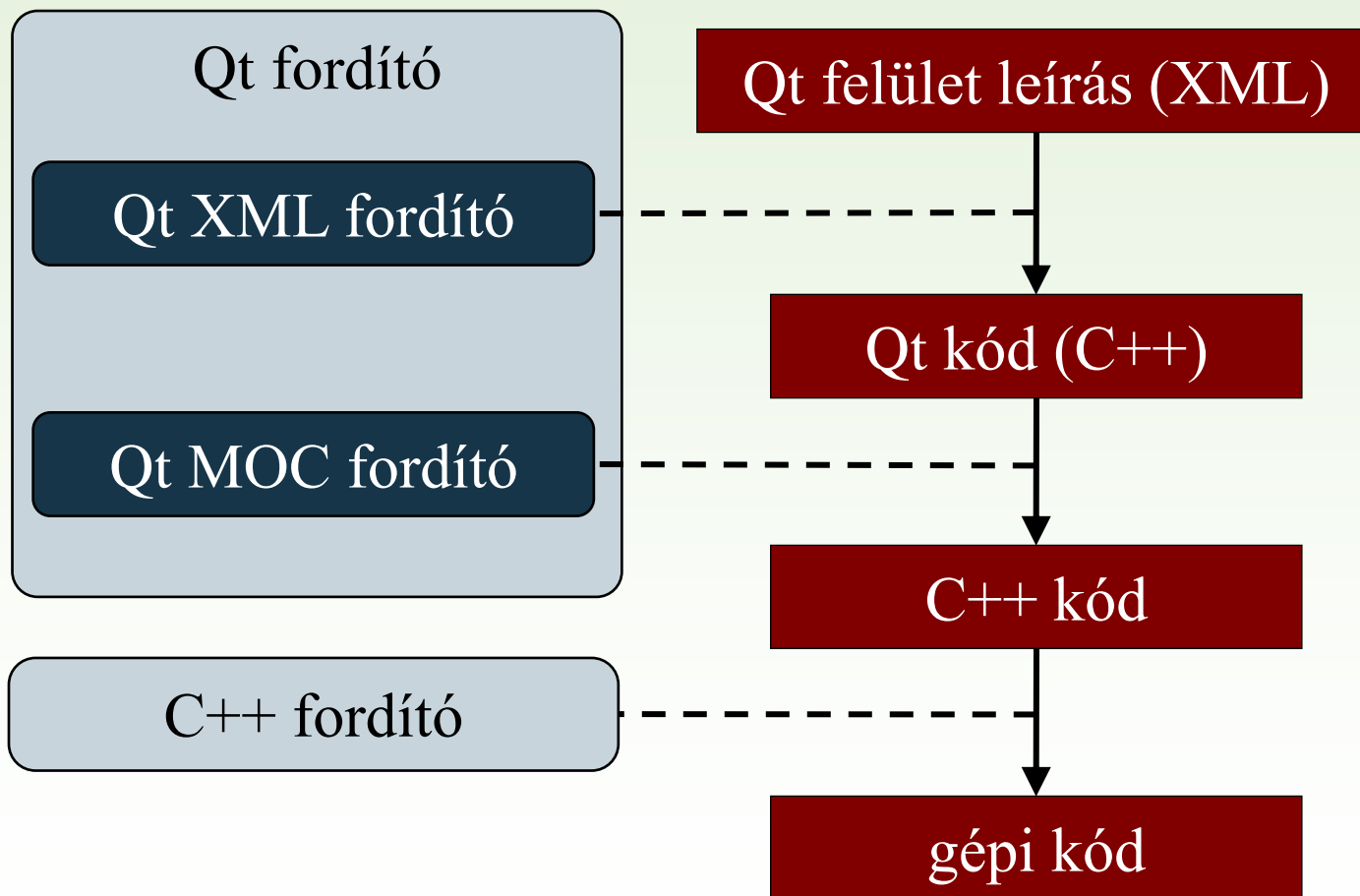
- ❑ A Qt keretrendszer felépítése modularizált. A legfontosabb modulok:
 - központi modul (`QtCore`)
 - grafikus felület (`QtGui`), grafikus vezérlők (`QtWidgets`)
 - adatbázis-kezelés (`QtSQL`)
 - hálózat-kezelés (`QtNetwork`)
- ❑ Egy modul tartalmát egyszerre (pl.: `#include <QtGui>`) is, de akár osztályonként is (pl.: `#include <QLabel>`) betölthetjük az aktuális fájlba.

Fordítás

- ❑ A fordítás projektszinten történik, az ehhez szükséges információk a *projekt fájlban* (`.pro`) tárolódnak, amely tartalmazza
 - a felhasznált modulokat, csatlakozásokat
pl.: `QT += core gui widgets sql network`
 - a forrás- (`.cpp`) és fejlécfájlok (`.h`), formok (`ui`) és erőforrások (`.png`, `.txt`, ...) listáját
 - sablon leírást (`app`)
- ❑ A fordítás a QtCreatorban egy kattintással, de akár közvetlenül (parancssorban) is elvégezhető:

```
qmake -project # könyvtár nevével azonos projekt fájl létrehozása
qmake          # fordító fájl (makefile) előállítás a projekt fájlból
make          # projekt fájlnak megfelelő fordítás és
               # szerkesztés végrehajtása
```

Fordítás lépései



Vezérlés

- ❑ Egy eseményvezérelt alkalmazás vezérlését egy ún. **alkalmazás objektum** (a Qt-ben a **QApplication** osztály példánya) látja el.
 - Beállítja az alkalmazás szintű tulajdonságokat (megjelenés, elérési útvonal).
 - Felügyeli a program futása során létrejött vezérlő objektumokat, (pl.: nyilvántartja azok elhelyezkedési hierarchiáját).
 - A programfutás során bekövetkező különféle történéseket, akciókat (pl. billentyűzet-, vagy egérhasználat, egy objektum üzenete, stb.) eseményként (speciális objektum) értelmezi.
 - Gondoskodik arról, hogy az események eljussanak azokhoz az objektumokhoz (**vezérlőkhöz**), amelyek megfelelő tevékenységek végrehajtásával reagálhatnak az eseményre.

Akció – Esemény – Kezelés

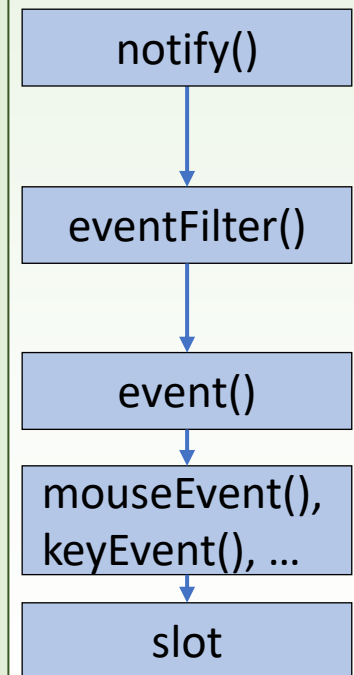
- ❑ Az **akció** az adott alkalmazás aktuális tevékenységétől függetlenül bekövetkező történés, amelyet kezdeményezhet a felhasználó (billentyűzet, egér, érintőképernyő, stb.), az alkalmazás egyik objektuma (pl. időzítő tikkélése), vagy egy másik alkalmazás.
- ❑ Az **esemény** (*event*) az alkalmazásban objektumként megjelenő akció.
 - Egy billentyű leütés akciójából a lenyomás és a felengedés eseménye lesz.
 - Az egér jobb fülével történő kattintás akcióból egy „egérgomb-lenyomás” és egy „egérgomb-felengedés” esemény születik.
- ❑ Egy eseményre az alkalmazás valamelyik vezérlő objektuma reagálhat úgy, hogy végrehajt valamilyen **tevékenységet**.
 - Ugyanazt az esemény akár több vezérlő objektum is megkapja, és többen is reagálhatnak rá.
 - Ugyanazt a reakciót több különböző esemény is kiválthatja (pl. egy fókuszban levő nyomógomb feletti egér-kattintás, vagy az <enter> leütése ugyanazon gomb megnyomását okozza).

Eseménykezelés formái

- ❑ **Eseményvizsgáló metódusok:** Egy vezérlő olyan metódusai, amelyeket az alkalmazás hív meg úgy, hogy átadja az adott eseményt. Ez a technika úgy tehető rugalmassá, ha
 - **Származtatás:** egy létező vezérlő osztályából származtatunk újat, és abban felüldefiniáljuk az eseményvizsgáló metódusokat.
pl.: a `KeyPressEvent()` metódust lehet (ha kell) ilyen módon felülírni.
 - **Függőség befecskendezés:** átadunk a vezérlőnek egy olyan objektumot, amelyiknek egy adott metódusa kezeli eseményt.
pl.: az `installEventFilter()` metódussal rendelhetünk futási időben egy ún. *monitoring* objektumot, amelynek `eventFilter()` metódusa kezeli a vezérlő eseményeit.
- ❑ **Signal-Slot kapcsolat:** egy vezérlő ún. eseményvizsgáló metódusai az esemény hatására egy szignált váltanak ki (*emit*), amelyre futási időben iratkozhat fel a szignált kiváltó esemény lekezelő metódusa (vagy metódusai).

Eseménykezelés szintjei

- Egy esemény feldolgozása az alkalmazás-objektum irányítása mellett elosztott módon történik a megfelelő sorrendben hívott **eseményvizsgáló metódusok** által.
 - Először az alkalmazás objektum `notify()` metódusa (amely felüldefiniálható) fut le. Ez juttatja el az eseményt a megfelelő (fókuszban levő vagy egér mutató alatti) vezérlőhöz.
 - A vezérlőkhöz opcionálisan hozzárendelt figyelő (*monitoring*) objektum (ugyanaz tartozhat több vezérlőhöz is) végezhet eseményfeldolgozást az `eventFilter()` metódusával.
 - Egy vezérlő a hozzá eljutó eseményt az `event()` metódusával (amely felüldefiniálható) dolgozza fel, és az esemény fajtájától függően lefut a `mouseXXXEvent()`, vagy a `keyPressEvent()`, vagy `paintEvent()` eseménykezelő is.
 - Ha ezek valamelyike szignált is kivált, akkor a szignálhoz rendelt tevékenységek (slot) végrehajtására is sor kerül.



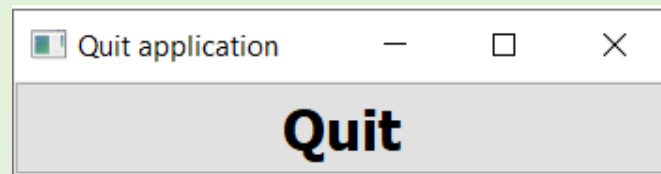
Események és szignálok

- ❑ Az esemény a **QEvent** osztályból származtatott objektum.
 - A több, mint száz különféle **esemény típust** *enum* értékek azonosítják, amelyeket a **QEvent::type()** ad meg.
 - Pl: **QEvent::KeyPress**, **QEvent::MouseButtonPress**
 - Egy eseménynek lehetnek **paramétere**i (*arguments*).
 - Pl. egérfül lenyomásakor: melyik fület nyomták le, mi az egérmutató pozíciója.
- ❑ Egy szignálra úgy gondoljunk, mint egy **törzs nélküli void-os függvényre**, amelyet épp emiatt nem „meghívunk”, hanem „kiváltunk”, és ha van hozzárendelve eseménykezelő tevékenység, akkor majd az hajtódik végre, különben nem történik semmi (a szignál lekezeletlen marad).
- ❑ Fejlesztés során
 - új vezérlő elkészítésekor annak eseményeire koncentrálnunk
 - egy vezérlő felhasználásakor annak szignáljaival foglalkozunk

Szignálok és slot-ok

- ❑ Egy szignálnak lehetnek **aktuális paramétere**i, amelyet a szignálhoz társított eseménykezelő formális paraméterváltozói vesznek át. Ezeknek sorrendben, típusban illeszkedni kell, de darabszámban nem.
- ❑ Egy szignálhoz több tevékenység is társítható, több különböző szignálhoz ugyanaz a tevékenység is, sőt egy szignálhoz egy mási szignál is.
- ❑ A társítást végző **connect** bármelyik **QObject**-ból leszármazott osztály metódusaként jelen van, de statikus metódus hivatkozással is elérhető. Négy paramétere van:
 - a szignált kiváltó küldő objektum (*sender*)
 - a szignált (**SIGNAL**)
 - a fogadó objektum (*receiver*)
 - a fogadó metódusaként definiált eseménykezelő (**SLOT**).
- ❑ Egy szignálhoz rendelt eseménykezelő metódusban mindig lekérdezhető a szignált kiváltó vezérlő objektum: a szignál **küldője** (*sender*).

“Quit” alkalmazás



```
#include <QApplication>
#include <QPushButton>
int main(int argc, char *argv[]) {
    QApplication app(argc, argv); // alkalmazás

    QPushButton quit;           // nyomógomb létrehozása
    quit.setText("Quit");        // ez konstruktorban is beállítható
    quit.resize(350, 50);       // méret beállítása
    quit.setFont(QFont("Times", 18, QFont::Bold)); // betűtípus
    quit.setWindowTitle("Quit application"); // ablak címe
    quit.setToolTip("You can exit"); // előugró szöveg

    QObject::connect(&quit, SIGNAL(clicked()), &app, SLOT(quit()));

    quit.show();                // nyomógomb megjelenítése
    return app.exec();          // alkalmazás indítása
}
```

Lehetne a quit objektum close() metódusa is

Felülettervező

A *felülettervező* (*Qt Designer*) lehetőséget ad a felület gyors elkészítésére

- ❑ Az elkészített felületterv XML-ben mentődik (`<ablaknév>.ui`), majd abból egy Qt osztály készül (`ui_<ablaknév>.h`)
- ❑ Az így generált osztály tulajdonságait egy saját, `QWidget`-ből származó osztály részévé úgy tehetjük, hogy
 - vagy annak másik őse a *Qt Designer*-rel generált osztály lesz,
 - vagy adattagként hivatkozunk (objektum befecskendezés) a generált osztály egy példányára, illetve annak vezérlőire
- ❑ A generált osztály vezérlőinek kialakítását a generált osztály `setupUi(QWidget* parent)` metódusának hívásával kell elvégezni.

The screenshot illustrates the Qt Creator interface during the development of a Qt widget. The top window, titled 'QuitWidget - Qt Creator', shows the project structure with 'quitwidget.ui' selected under the 'Forms' folder. The bottom window, titled 'quitwidget.ui @ QuitWidget - Qt Creator', shows the design view of the 'Quit' button. A red arrow points from the 'Push Button' widget in the widget toolbox to the button in the design view. Below the design view, a signal/slot connection table is shown, with the connection between 'pushButton.clicked()' and 'QuitWidget.close()' circled in red. The Properties panel on the right shows the properties of the selected 'pushButton : QPushButton' widget.

| Sender | Signal | Receiver | Slot |
|------------|-----------|------------|---------|
| pushButton | clicked() | QuitWidget | close() |

| Property | Value |
|--|-------------------------------------|
| QObject | |
| objectName | pushButton |
| QWidget | |
| enabled | <input checked="" type="checkbox"/> |
| geometry [(0, 0), 341 x 71] | |
| X | 0 |
| Y | 0 |
| Width | 341 |
| Height | 71 |
| sizePolicy [Minimum, Fixed, 0, 0] | |
| Horizontal P... | Minimum |
| Vertical Policy | Fixed |
| Horizontal S... | 0 |
| Vertical Stre... | 0 |
| minimumSize 0 x 0 | |
| Width | 0 |
| Height | 0 |
| maximumSize 16777215 x 16777215 | |
| Width | 16777215 |

Generált kód

```
namespace Ui { class QuitWidget; }

class QuitWidget : public QWidget
{
    Q_OBJECT
public:
    explicit QuitWidget(QWidget *parent = nullptr);
    ~QuitWidget();
private:
    Ui::QuitWidget *ui;
};
```

```
#include "quitwidget.h"
#include "ui_quitwidget.h"

QuitWidget::QuitWidget(QWidget *parent) :
    QWidget(parent), ui(new Ui::QuitWidget)
{
    ui->setupUi(this);
}
QuitWidget::~~QuitWidget()
{
    delete ui;
}
```

innenől használhatók a
tervezővel definiált
vezérlők: `ui->pushButton`