Multiplatform szoftverfejlesztés

Qt Quick 2

Qt Quick

JavaScript

QML JavaScript

- Jelenlegi JavaScript verzió: ES2016 (Qt 6.5-ben)
- JS kódot lehet írni
  - Property Binding
  - Eseménykezelő (Signal handler)
  - Bármilyen objektumhoz új függvényt
  - Importálható .js fájl
    - Ezek a változók és függvények felhasználhatók bárhol

QML JavaScript

- Nincs window, document, …
- Van Qt globális objektum
  - Számos Qt specifikus függvény
    - binding, exit, …
  - Segítő függvények
  - application object
    - Alaptulajdonságok mint name, version, …
  - platform (android, ios, windows, …)

QML import

- import <Module> <Version> as <Name>
  - <Module> lehet
    - Telepített modul, pl. QtQuick (az alapkontrollokhoz)
    - JS fájl
  - <Version> meg kell egyezzen, ha megadjuk
- Ha nincs név, akkor globális névtérbe kerül
  - import QtQuick 2.0

JavaScript - Property Binding

```
Rectangle {
    id: colorbutton
    width: 200; height: 80;
    color: mousearea.pressed ? "steelblue" : "lightsteelblue"
    MouseArea {
        id: mousearea
        anchors.fill: parent
    }
}
```

JavaScript - Eseménykezelő

```
Rectangle {
    width : 200; height: 80; color: "lightsteelblue"
    MouseArea {
        anchors.fill : parent
        onPressed : {label.text = "I am Pressed!"}
    }
    Text {
        id: label
        anchors.centerIn : parent
        text : "Press Me!"
    }
}
```

JavaScript – saját függvény

```
Item{
    function pi2() {
        return Math.PI * Math.PI;
    }

    MouseArea{
        anchors.fill: parent
        onClicked : console.log(pi2())
    }
}
```

JavaScript – külső .js fájl

- import kulcsszó
- as libname
  - Így lehet hivatkozni rá

```
import "hello.js" as MyFunctions
Item{
    MouseArea{
        anchors.fill: parent
        onClicked : console.log(MyFunctions.hello())
    }
}
```

JS, vagy C++

- Betöltés után szinte mindent meg lehet írni JS-ben
- És C++-ban is
  - QObject-ből származtatunk – ez látszik QML-ben
- Ezek után el is felejthetjük a C++-t?
- Válasz 1
  - Nem mindent lehet megírni JS-ben
  - JS lassabb, de ez vajon számít-e
  - Ha minden JS lenne, akkor miért nem HTML+JS?
    - QT többet tud, mint HTML
    - Van lehetőségünk átlépni C++-ba, ha szükséges

JS, vagy C++

- JS-ben célszerű megírni, ami a felülethez tartozik
  - Ha nem túl bonyolult
    - Nincsenek olyan szintű eszközök, mint C++-hoz
  - Ha nem számít a sebesség
    - C++ hozzáállás: mindig számít
    - Finomítunk: Ha nem kritikus a sebesség

Qt Quick

Alaposztályok

QtObject

- Egyszerű objektum
  - Csak egy neve van
  - C++ kódból név alapján meg lehet találni
  - Csoportosíthatunk mezőket (jobb oldal)
- Hierarchia gyökere
  - Szinte minden ebből származik
  - Item ebből származik

```
QtObject{
    id: attributes
    property string name
    property int size
    property variant attributes
}

Text{ text: attributes.name }
```

Component – sablon

```
Component{
    id: redSquare
    Rectangle{
        color: "red"
        width : 10
        height : 10
    }
}

Loader{ sourceComponent: redSquare; x: 20 } // QML-ben

redSquare.createObject(parent, {"x":20}); // JS függvényben
```

- Újrafelhasználható
- Létrehozza minden példányhoz
- ItemTemplate (XAML-ben) hasonló

Component

- Betöltés állapota
  - progress: 0..1
    - 1-nél nincs kész, status==Component.Ready-nél van kész
  - status: Null, Ready, Loading, Error
  - url: a qml fájl, amiben van
- Minden így töltődik be, az első qml fájlunk is
- Component.onCompleted
  - attached signal – mindennek van
  - (Lehet használni globális JS függvényt is – saját dolgait tudja inicializálni)
- Component.onDestruction()

Qt Quick

Adatkötés

QML Item – Adatkötés

- Minden köthető mindenhez
  - Alaptípusok tulajdonságai nem triggerelnek
- Tetszőleges JS kifejezés, vagy teljes kód írható be
  - Kifejezés esetén nem kell return, sem kapcsos zárójel

```
Rectangle {
        width: 100
        height: parent.height
        color: "blue"
}
```

QML Item – Adatkötés

- Property Binding csak egy kényelmi szintaktika
  - Valóságban egy binding objektum jön létre
    - Tetszőleges JS függvényt átadhatunk
- Qt.binding: JS szintaktika
- Függőségi gráf

```
Rectangle{
    width: parent.width
    Component.onCompleted: {
        height = Qt.binding(function() { return parent.height });
    }
}
```

QML Item – Adatkötés

- Adatkötés csak akkor történik, ha a fenti két szintaktikát használjuk
- Például itt space-re nem
  - Helyette simán kiszámolja az értéket, beleírja, majd nem változik többé

```
Rectangle {
    width: 100
    height: width * 2
    Keys.onSpacePressed: {
        height = width * 3
    }
}
```

QML Item – Adatkötés

- Két irányú adatkötés – nincs külön támogatás
  - Binding-loop kialakulhat (itt nem, mert property csak akkor süti el a changed signalt, ha tényleg változik az értéke)

```
Button {
    id: button1
    property int count: 0
    onClicked: count += 1
    text: count
}
Button {
    id: button2
    property int count: 0
    onClicked: count += 1
    text: count
}
```

```
Binding {
        target: button1
        property: 'count'
        value: button2.count
}
Binding {
    target: button2
    property: 'count'
    value: button1.count
}
```

Qt Quick

Állapotok

QML Item – Állapotok

- Aktuális állapot: state
- Összes állapot: states: list<State>
- Váltani a state=<state neve> módon lehet
- Mindenképpen van alapállapot
  - A neve üres string
  - Mindent visszaállít a kezdeti állapotba
  - Definiálni is lehet saját alapállapotot

QML Item – Állapotok

- State QML elem
  - name: string
  - changes: list<Change>
    - PropertyChanges, AnchorChange, ParentChange, …
  - extend: string
    - Átveszi a megnevezett state összes változását
    - Felül lehet írni, hozzá lehet adni
  - when: bool
    - JS kifejezést adunk meg, ami kiértékelődik
    - Ha igaz, akkor automatikusan állapotot vált
    - Ha több is igaz lesz egyszerre, akkor az elsőre vált

QML Item - Állapotok

```
Rectangle {
    id: rect
    width: 100; height: 100
    color: "black"
    MouseArea {
        id: mouseArea
        anchors.fill: parent
        onClicked: rect.state == 'clicked' ? rect.state = '' : rect.state = 'clicked';
    }
    states: [
        State {
            name: "clicked"
            when: width < 100
            PropertyChanges { target: rect; color: "red" }
        }
    ]
}
```

QML Item - Állapotok

- PropertyChanges
  - Leggyakrabban használt
  - Több tulajdonságot is állíthat
    - PropertyChanges { target: rect; color: "blue"; height: 5 }
    - undefined visszaállítja alapértékre (nem kezdeti értékre)
  - Lehet bindingot létrehozni
    - PropertyChanges { target: rect; height: parent.width }
  - Vagy az éppen aktuális értékét beírni binding nélkül
    - PropertyChanges { target: rect; explicit: true; height: parent.width }

Qt Quick

Átmenetek

QML Item - Átmenetek

- Ha nem azonnali váltás kell – sosem az kell
- transitions: list<Transition>
- Átmenet lehet
  - Állapotok között
    - Transition { from: "s1"; to: "s2"; ColorAnimation { … }}
    - Minden állapotban – from: "*", ez az alapértéke
  - Tulajdonság változásra – Behavior
    - Ez kicsi prioritású – ha van állapot animáció, akkor az érvényesül ütközés esetén
    - Behavior on width {NumberAnimation{ duration: 10 }}

QML Item - Átmenetek

- Minden animációban
  - Tulajdonságok: loops, paused, running
  - Események: started(), stopped()
  - Függvények: start(), stop(), restart(), pause(), resume(), complete()
- Animációk hierarchiába szervezése
  - Alapban párhuzamosan futnak a Transition-be felvett animációk, ha több van
  - SequentialAnimation
  - ParallelAnimation

QML Item - Átmenetek

- Nem animáló animációk
  - PropertyAction – nem animál, azonnal átállít
  - PauseAnimation – vár
  - ScriptAction – kód futtatása
- PropertyAnimation
  - NumberAnimation – lineáris
    - SpringAnimation – rugó
    - SmoothedAnimation – Easing function
  - ColorAnimation
  - RotationAnimation
  - Vector3dAnimation

QML Item - Átmenetek

```
Rectangle {
    id: rect; width: 100; height: 100; color: "red"
    MouseArea {
        id: mouseArea; anchors.fill: parent
    }
    states: State {
        name: "moved"; when: mouseArea.pressed
        PropertyChanges { target: rect; x: 50; y: 50 }
    }
    transitions: Transition {
        NumberAnimation { properties: "x,y"; duration: 300 }
    }
}
```

QML Item - Átmenetek

- Easing
  - PropertyAnimation tulajdonsága
  - Type
    - Egyenlet: Linear, Quad (x^2), Cubic (x^3), Quart (x^4), Quint (x^5), Sine(sin), Expo (2^x), Circ ($sqrt(1-x^2)$), Elastic, Back, Bounce, Bezier
    - Belépési/kilépési pont: In, Out, InOut, OutIn
  - Paraméterek: amplitude, overshoot, period

```
NumberAnimation {
    to: 100; easing.type: Easing.OutBounce; duration: 1000
}
```

QML Item - Átmenetek

- Speciális animációk
  - AnchorAnimation – változik az objektum
  - ParentAnimation – változik a szülő
  - PathAnimation – x,y animáció
- on szintaktika – <Animation> on <property>
  - Nem kell megadni: target, properties
    - PropertyAnimation on x { to: 100 }
  - Automatikusan indul, ha nem átmenetben van

Qt Quick

Style

QML Style

- Minden vezérlőnek testre szabható a kinézete
  - Amit mi írunk, annak csak akkor, ha megírjuk
- style tulajdonságnak kell megadni a saját stílust
- Minden vezérlőnek mást kell megadni
  - Pl. Button-nak ButtonStyle
  - Ezek egyszerű struktúrák, összefogják a részeket, amit megadhatunk
  - Nem kell minden részt felüldefiniálni

QML Style

```
Button{
    text: "Push"; width: 100; height: 30
    style : ButtonStyle{
        background: Rectangle{
            border.width : control.activeFocus ? 2 : 1
            border.color : "red"
            radius : 10
            gradient : Gradient{
                GradientStop{ position: 0; color: "white" }
                GradientStop{ position: 1; color: "blue" }
            }
        }
    }
}
```

QML Style

- A Style struktúrában vannak Component-ek az egyes részekhez, és egyéb adatok
- Például
  - ButtonStyle: background, label
  - CheckBox: background, label, indicator, spacing: int
  - Slider: groove, handle, panel, tickmarks

Qt Quick

Magas szintű szolgáltatások

Dialógus ablakok

- ColorDialog
- FileDialog
- FontDialog
- MessageDialog
- Dialog
  - Általános célú ablak, platform gombokkal
  - OK, Cancel, Open, Save, Close, Apply…

Dialógus ablakok

```
Dialog {
    id: dateDialog
    visible: true
    title: "Choose a date"
    standardButtons: StandardButton.Save | StandardButton.Cancel
    onAccepted: console.log("Saving the date " +
        calendar.selectedDate.toLocaleDateString())
    Calendar {
        id: calendar
        onDoubleClicked: dateDialog.click(StandardButton.Save)
    }
}
```

HTML megjelenítése

- Text vezérlő szöveget tud
- WebView teljes weboldalt
  - Url-t is meg lehet adni, ahonnan letölti
  - Böngészőt lehet írni vele
  - Chromium 108 motor

Szenzorok

- Szokásos: Accelometer, Altimeter, AbmientLightSensor, Compass, Gyroscope…
- Speciális eszközökben vannak: AmbientTemperatureSensor, DistanceSensor…
- Összesen 17 szenzor
  - Vannak átfedések (Pl. Light és AmbientLight)
- Nem csak mobil eszközökre van kitalálva, de azokkal is jól megy
  - Robotok vezérlése

Multimédia

- MediaPlayer
  - Audio
  - Video
- Camera
  - Az összes beállítása ki van vezetve (vaku, fókusz is)
- Radio
- Torch

```
Torch {
    power: 75          // 75% of full power
    enabled: true      // On
}
```

ShaderEffect

- Vertex és pixel shader
- Rectangle az objektum, ebben lehet más is
- Alapban 4 vertex
  - mesh megadásával bármennyi lehet (torzításhoz)
- cullMode állítható (backface culling)
- blending: alpha blending bekapcsolásához
  - Csak a source alpha és az összeadást tudja

ShaderEffect

- Vertex shader
  - Standard bemenő paraméterek
    - Állandó minden vertexre (uniform): qt_Matrix, qt_Opacity
    - Vertexenként (attribute): qt_Vertex, qt_MultiTexCoord0
  - Ezen kívül fel lehet venni bármi mást is, amit átadunk
  - Kimenetet mi definiáljuk
    - varying kulcsszó, ez megy a pixel shadernek

ShaderEffect

```
vertexShader: "
    uniform highp mat4 qt_Matrix;
    attribute highp vec4 qt_Vertex;
    attribute highp vec2 qt_MultiTexCoord0;
    varying highp vec2 coord;
    void main() {
        coord = qt_MultiTexCoord0;
        gl_Position = qt_Matrix * qt_Vertex;
    }"
```

ShaderEffect

- Pixel shader
  - Vertex shader kimenete jön interpolálva
  - Sampler a mintavételezéshez
    - Image-re kötve
  - Állandókat felvehetünk (uniform)

```
fragmentShader: "
    varying highp vec2 coord;
    uniform sampler2D src;
    void main() {
        gl_FragColor = texture2D(src, coord);
    }"
```

Kérdések?