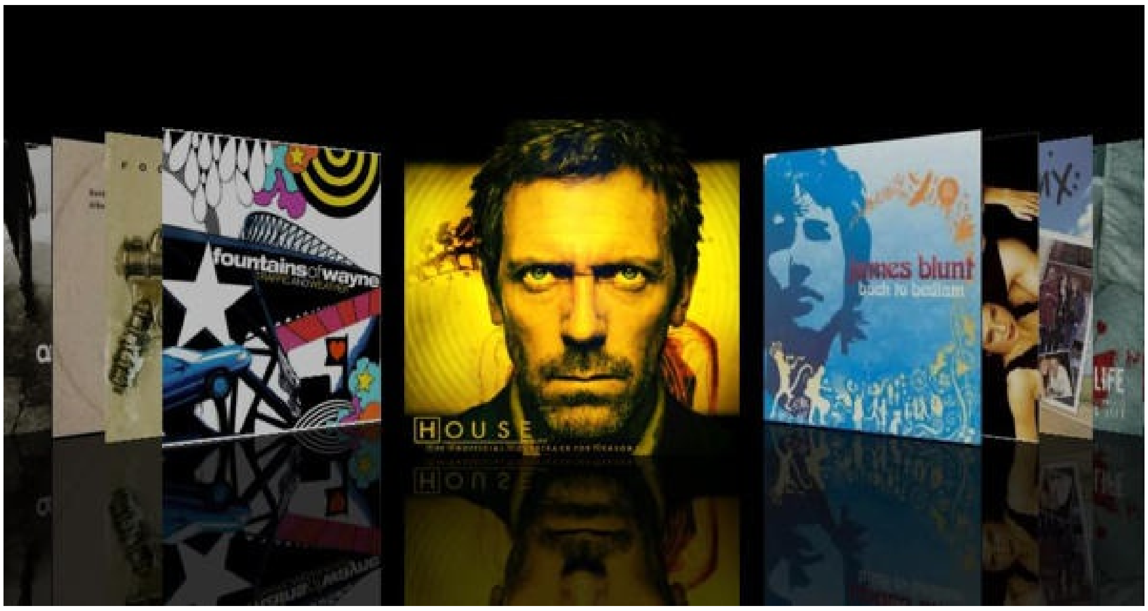
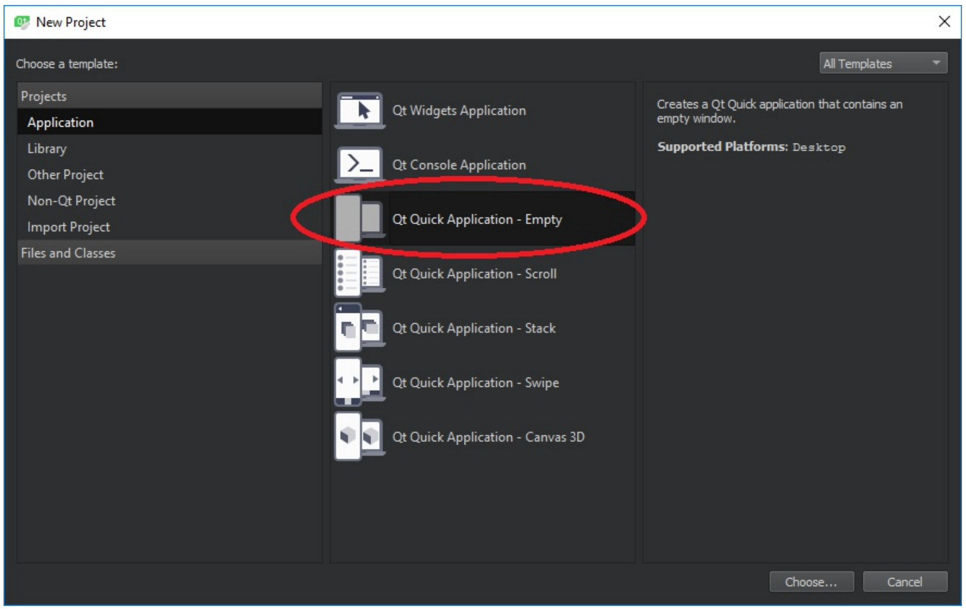
Introduction

The aim of this exercise is to create a QML UI solution similar to Apple's Cover Flow.



We will use the **Path View** QML component to solve the task.

1. Let us crea an empty project by using the ***Qt Quick Application Empty*** template.

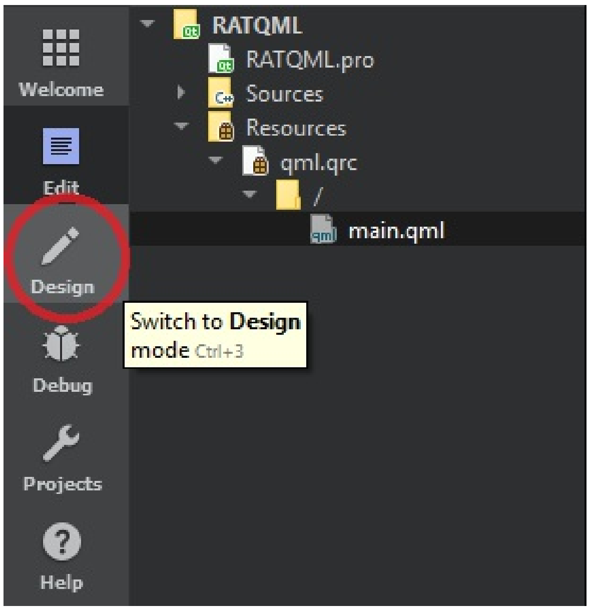


Creates a Qt Quick application that contains an empty window.

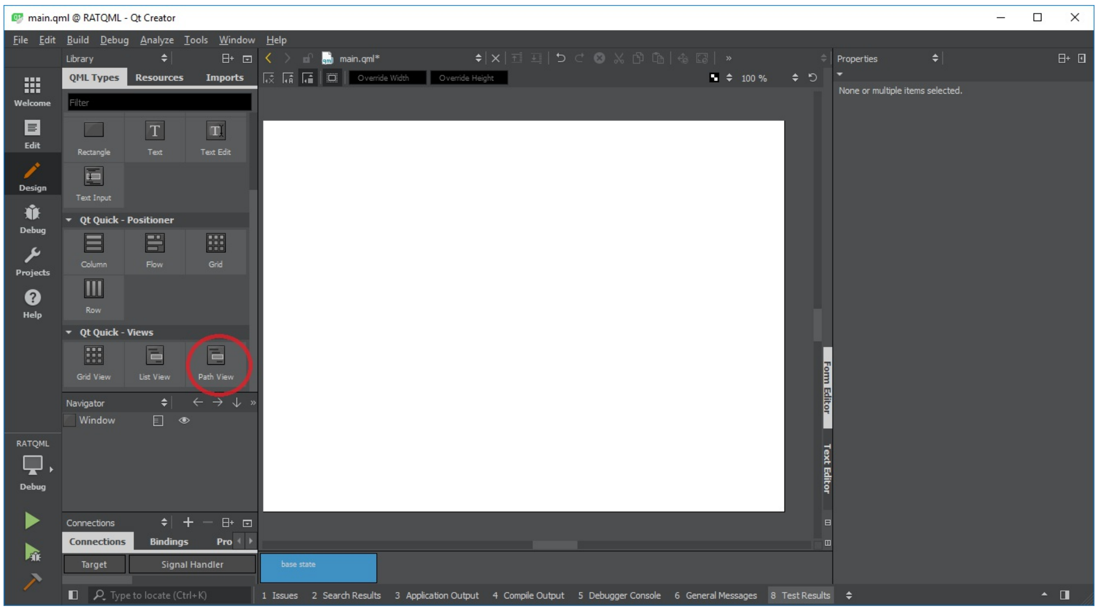
Supported Platforms: `Desktop`

2. Open the *main.qml* file.

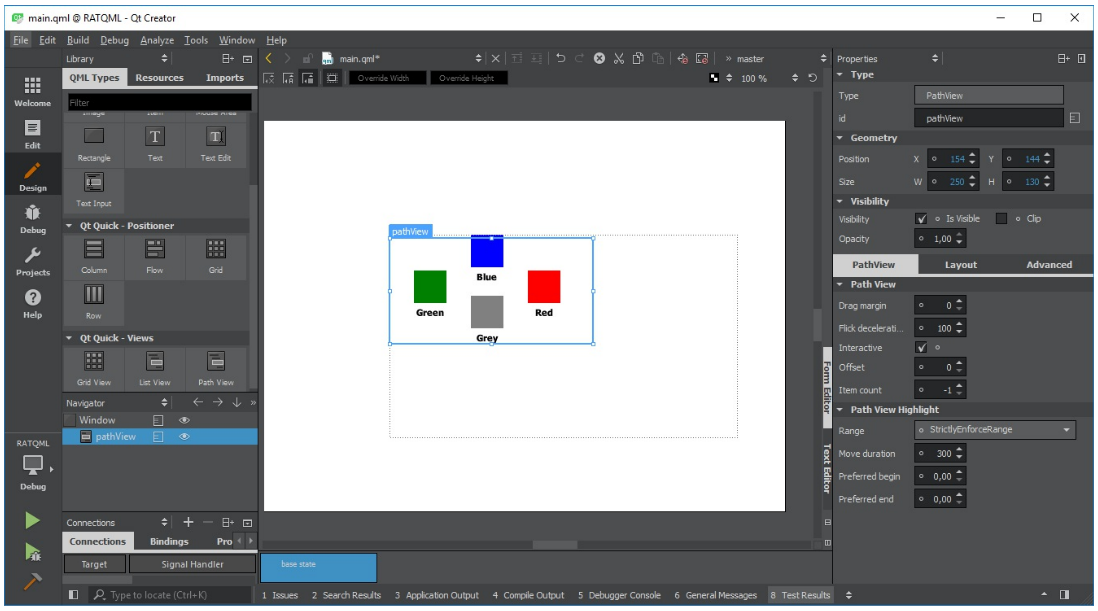
3. Change to *Design* mode.



4. In the **Qt Qucik - Views** section, select the **Path View**.


main.qml @ RATQML - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Library
QML Types Resources Imports
Rectangle
Text
Text Edit
Text Input
Qt Quick - Positioner
Column
Flow
Grid
Row
Qt Quick - Views
Grid View
List View
Path View
Navigator
Window
Connections
Bindings
Target
Signal Handler
Properties
None or multiple items selected.
base state
Form Editor
Text Editor
1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 Debugger Console 6 General Messages 8 Test Results

5. Add it to the *Window* by a simple *drag&drop*.


main.qml @ RATQML - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Library
QML Types Resources Imports
Rectangle Text Text Edit Text Input
Qt Quick - Positioner
Column Flow Grid Row
Qt Quick - Views
Grid View List View Path View
Navigator
Window
pathView
Connections
Connections Bindings Pro
Target Signal Handler
Welcome Edit Design Debug Projects Help
RATQML Debug
main.qml*
master
100 %
pathView
Blue
Green
Red
Grey
base state
Form Editor
Text Editor
Properties
PathView Layout Advanced
1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 Debugger Console 6 General Messages 8 Test Results
Type to locate (Ctrl+K)

Type: PathView
id: pathView
Geometry
Position X 154 Y 144
Size W 250 H 130
Visibility
Visibility ✓ Is Visible ☐ Clip
Opacity 1,00
Path View
Drag margin 0
Flick decelerati... 100
Interactive ✓
Offset 0
Item count -1
Path View Highlight
Range StrictlyEnforceRange
Move duration 300
Preferred begin 0,00
Preferred end 0,00

6. Run the application and check the functionality of the *Path View*. It can be seen that each element moves on a particular curve as the mouse moves.

7. Switch back to *Edit* mode and take a look at the generated code.

```
PathView {
        id: pathView
        x: 154
        y: 144
        width: 250
        height: 130
        path: Path {
            ...
        }
        delegate: Column {
             ...
        }
        model: ListModel {
             ...
        }
    }
```

7. To make our view more dynamic, make the following changes. First, get rid of the fixed position and size and use the anchors.

```
PathView {
        id: pathView
        anchors.fill: parent
        path: Path {
            ...
```

9. Add the following two properties to our PathView which will be used during the segmentation of the *Path*.

```
    property real centerX: width / 2
    property real vertOff: height / 2
```

9. Create a new QML file in which we will store the *path* of the items.Choose *File* from the top menu bar and select *New File or Project* menu item.

10. In the dialog window from the *Files and Classes* templates select *Qt* and then select *QML File (Qt Quick 2)*. Name the new file as **CoverFlowPath**.

11. Let us open our newly created file and change the type of the root element from *Item* to *Path*:

```
import QtQuick 2.0

Path {

}
```

12. Within the *Path*, create a *property* named **pathView**, which will point to the presenting PathView object. This is required in order to retreive the dimensions of our view.

```
Path {
    property PathView pathView
}
```

13. Create a straight path that overlaps the horizontal centerline of your *PathView*. To do this, you need a starting point and a straight path (PathLine), which contains the endpoint:

```
Path{
    //kezdőpont
    startX: 0
    startY: pathView.vertOff

    //egyenes vonal a végpontig
    PathLine {
        x: pathView.width
        y: pathView.vertOff
    }
}
```

14. Unpack the images.zip file and copy the images to a separate directory next to the project's source files.

15. Add the images to the project as resource. You can do this by moving the cursor over the **qml.qrc** file in the **Resources** folder, then press the right mouse button and select "*Add Existing Files*".

16. Select again the **qml.qrc** file, press the right mouse button and now select the "*Open in Editor*" menu item. In the *Editor* set the proper *Alias* for each image (i.e.: images/01.jpg -> 01).Don't forget to press the *Ctrl+S* to save the file.

17. Add the attached file **MyModell.qml** also to the **qml.qrc**. This is the model which stores a list of team name and their image source name.

18. Add a blank QML file to the project with the name *MyDelegate.qml*.

19. Replace generated *Item* with the following code:

```
Item {
    id: root

    width: 200
    height: width

    Image{
        id: delegate
        anchors.fill: parent
        source: imageName
        Rectangle{
            anchors.fill: parent
            color: "lightGrey"
            z: -1
        }
    }
}
```

20. Open the *main.qml* and set the *delegate* and *model* propteries of the *PathView*:

```
        delegate: MyDelegate {}

        model: MyModell {}
```

21. Build & Run the application! It can be seen that views are overlaped but there is no 3D effect on them.

22. Extend the *MyDelegate.qml* by adding the following lines of code:

```
Item {
    id: root
    width: 200
    height: width
    z: PathView.zOrder

    transform: [
        Rotation {
            angle: root.PathView.rotateY
            origin.x: delegate.width / 2
            origin.y: delegate.height * 0.3
            axis.x: 0
            axis.y: 1
            axis.z: 0
        }
    ]
    ...
```

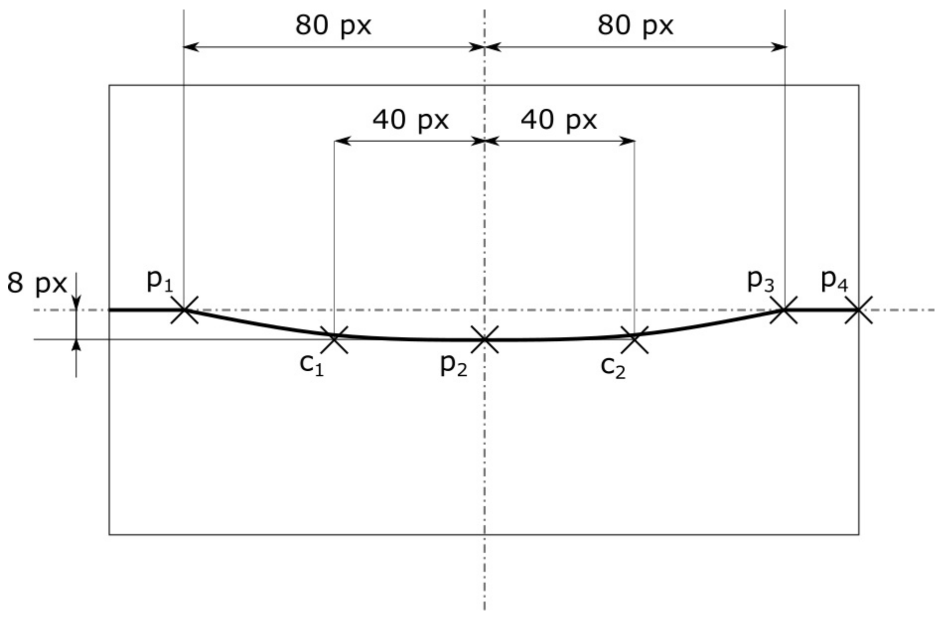
23. It can be seen that the *angle* parameter refere to *rotateY* property of the *PathView*, which is not a *built-in* parameter, howver it can be assigned to each path segment. Let's do this! Extend the initial path of the *Path* in the **CoverFlowPath.qml** file:

```
startX: 0
startY: pathView.vertOff

PathAttribute {
    name: "rotateY"
    value: 50.0
}

PathLine {
    x: pathView.width
    y: pathView.vertOff
}
```

24. Now that we have a base path and a rotated images, add the following points with _PathLine_s *p1,p4* and the following points and control points with _PathQuad_s *(p2,c1),(p3,c2)*. Mind the order! p2 comes righ after p1 and p3 comes after p2 and so on.



25. Once you've created the curve, begin to set the rotation of the elements in each segment:

- *start*: **50** *(this is already set)*
- *p1*: **50**
- *p2*: **0**
- *p3*: **-50**
- *p4*: **-50**

26. It can be seen that the pictures are already rotating. Now you have to make sure that the *z order* of the elements is correct, so the image in the middle should always be in foreround. To do this, you have to do the same as before with rotation add a new *PathAttribute* for each segment as follows:

```
startX: 0
startY: pathView.vertOff

PathAttribute {
    name: "rotateY"
    value: 50.0
}
PathAttribute {
    name: "zOrder"
    value: 1.0
}

PathLine {
    x: pathView.width
    y: pathView.vertOff
}
```

27. The *z order* values of the segments are the followings:

- *start*: **1.0** *(this is already set)*

- *p1*: **10.0**
- *p2*: **50.0**
- *p3*: **10.0**
- *p4*: **1.0**

28. The *z order* is now set correctly, but the image in the middle changes very fast. You can modify this by adding *PathPercent* component next to the *PathAttributes* according to the following pattern:

```
PathQuad {
    x: pathView.centerX
    y: pathView.vertOff + 8
    controlX: pathView.centerX - 40
    controlY: pathView.vertOff + 8
}

PathPercent {
    value: 0.5
}
PathAttribute {
    name: "rotateY"
    value: 0.0
}
```

29. However, this does not need to be set anywhere, except for the following points:

- *p1*: **0.44**
- *p2*: **0.5**
- *p3*: **0.56**

30. It's almost done, but you still have to solve the grow and shrink of the elements as they enter and exit the center part of the window. To solve this, extend the *transform* component in *MyDeleage.qml* with a *Scale* component as follows:

```
Item {
    id: root

    width: 200
    height: width
    z: PathView.zOrder

    transform: [
        Rotation {
            angle: root.PathView.rotateY
            origin.x: delegate.width / 2
            origin.y: delegate.height * 0.3
            axis.x: 0
            axis.y: 1
            axis.z: 0
        },
        Scale {
            xScale: 1.0
            yScale: root.PathView.scale
            origin.x: delegate.width / 2
            origin.y: delegate.height * 0.4
        }
    ]
```

31. Here we need a *scale* property (analogue to the *rotateY*) from *PathView* to determine the scale factor, which is also not a basic parameter of the *PathView*. So we must provide the proper *PathAttributes* to each segment:

```
startX: 0
startY: pathView.vertOff

PathAttribute {
    name: "rotateY"
    value: 50.0
}
PathAttribute {
    name: "zOrder"
    value: 1.0
}
PathAttribute {
    name: "scale"
    value: 0.7
}
```

32. The *scale* values of the segments are the followings::

- *start*: **0.7** *(this is already set)*
- *p1*: **0.7**
- *p2*: **1.0**
- *p3*: **0.7**
- *p4*: **0.7**

33. Finally add a reflection effect (*ShaderEffectSource*) to the *MyDelegate.qml* file:

```
Image{
    id: delegate
    anchors.fill: parent
    source: imageName
    Rectangle{
        anchors.fill: parent
        color: "lightGrey"
        z: -1
    }
}

ShaderEffectSource {
    id: reflection
    sourceItem: delegate
    y: sourceItem.height
    width: sourceItem.width
    height: sourceItem.height

    transform: [
        Rotation {
            origin.x: reflection.width / 2
            origin.y: reflection.height / 2
            axis.x: 1
            axis.y: 0
            axis.z: 0
            angle: 180
        }
    ]
}
```

34. It is not completely perfect since the reflection is too *perfect*. Add some gradient to the effect that dims the bottom half of the reflected image:

```
    Rectangle {
        anchors.fill: reflection

        gradient: Gradient {
            GradientStop {
                position: 0.0
                color: "#55ffffff"
            }
            GradientStop {
                position: 1.0
                color: "#ffffff"
            }
        }
    }
```