Introduction to Qt Design Studio 1.1.x

Customer, Date

Presented by presenter

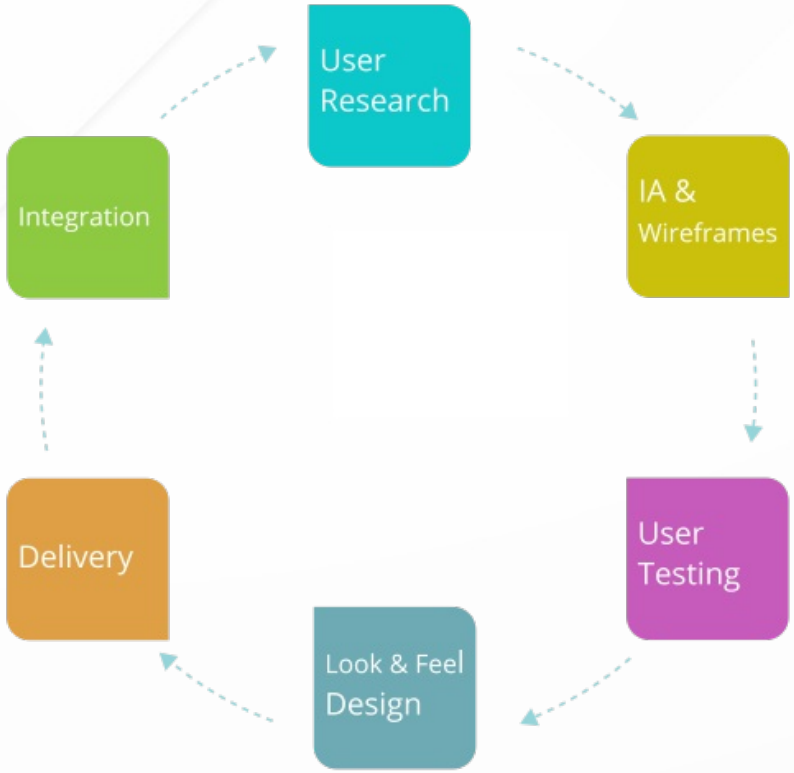
Created on July 19, 2023

- **Qt Design Studio Application**
  - Design workflow
  - Introduction to Qt Design Studio
- Import from Figma / Photoshop / etc
- Animations in QtDS
- 3D Incorporation
- Introduction to Qt Design Studio
- QML Integration

- **Design workflow**
  - Traditional workflow
  - Qt Design Studio pipeline
- Introduction to Qt Design Studio

- **Traditional workflow**
- Qt Design Studio pipeline

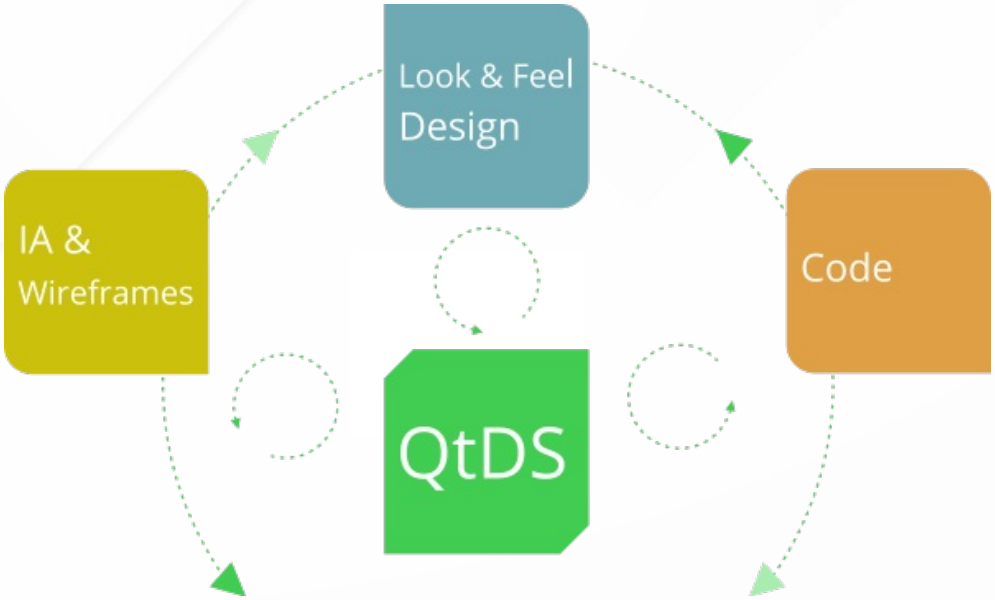
UX/UI -> Mocks & Wire-frames -> Code



Lost in translation problem

- Traditional workflow
- **Qt Design Studio pipeline**

QtDS way



Artist tools

- Primary sources for content such as Sketch, Figma, Photoshop, Blender, Inkscape
- Learning to produce high-quality content in these tools takes months or years
- Basic familiarity with the tools is a huge benefit in working with assets

Example flow..

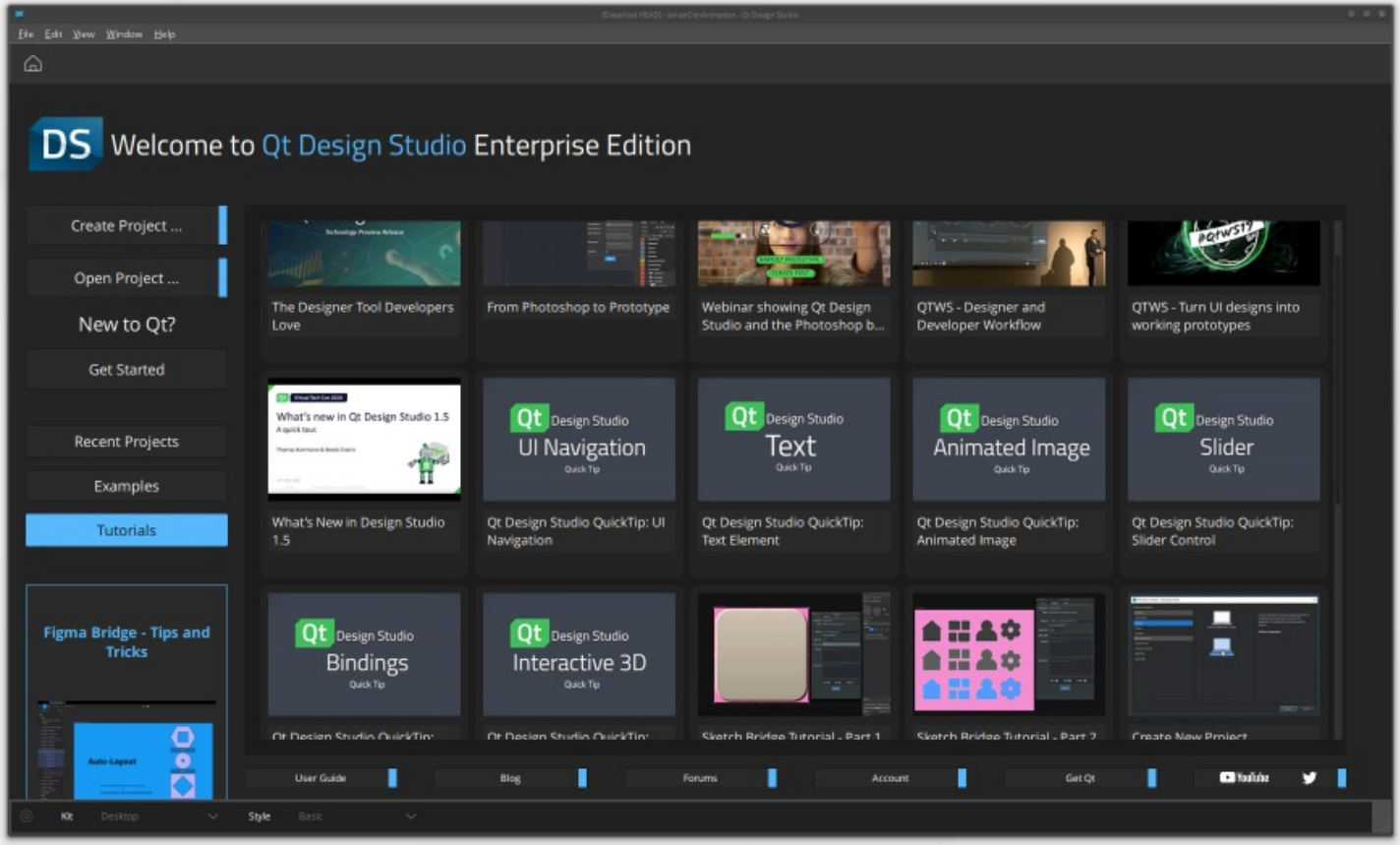
- Prototype in Figma.
- Export to QtDS
  - Animations and interaction defined by Designers
- Developer gets QML code
  - Developer adapts the code for use in the application
- Designer evaluates implantation
- Loop (goto start)

- Design workflow
- **Introduction to Qt Design Studio**
  - Qt Design Studio Overview
  - QML Fast Intro

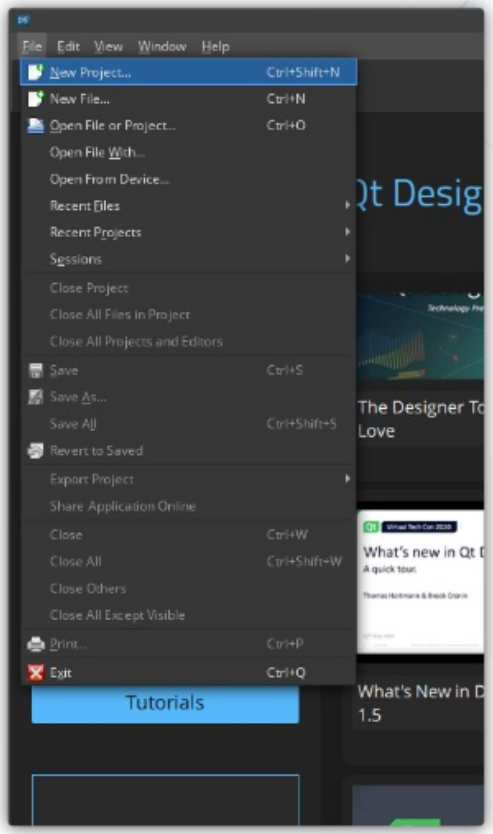
- **Qt Design Studio Overview**
  - Let's Start.
  - Main UI, Tabs & Setup.
- QML Fast Intro

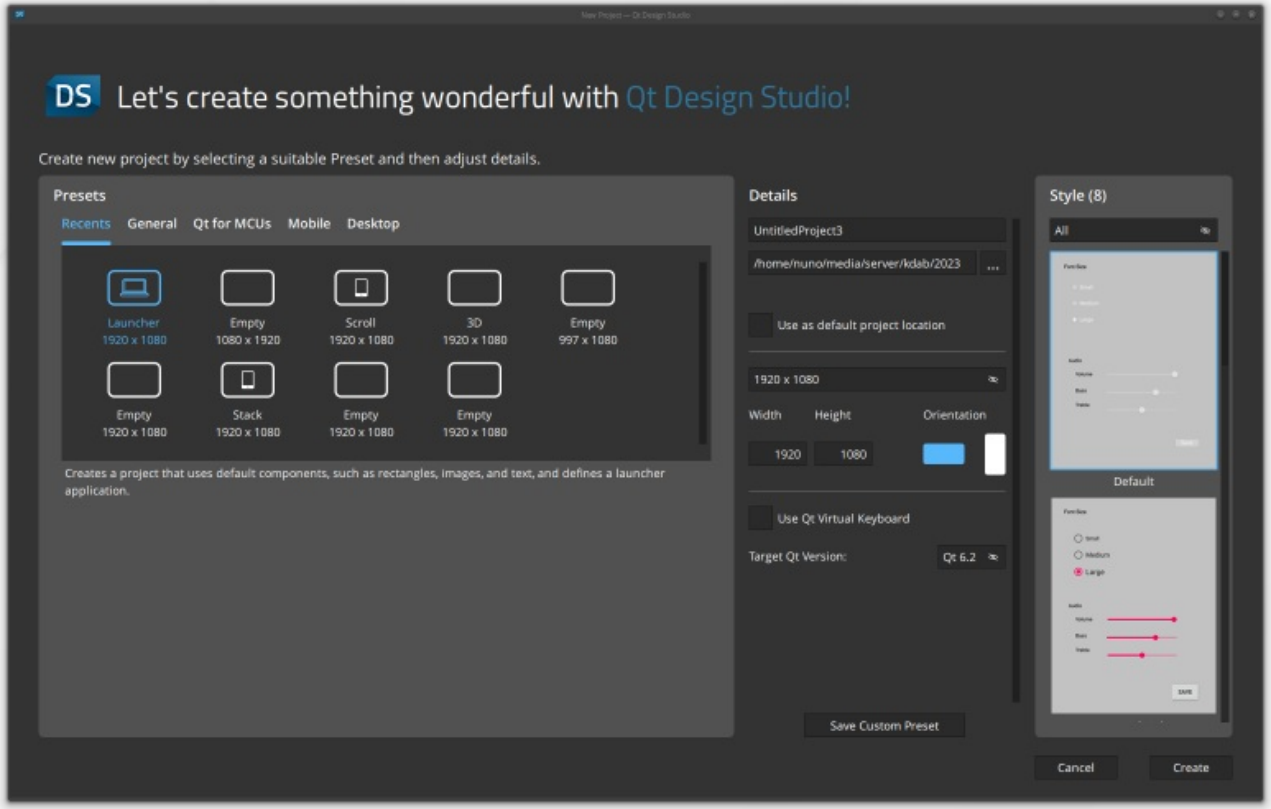
- **Let's Start.**
- Main UI, Tabs & Setup.

Welcome Page

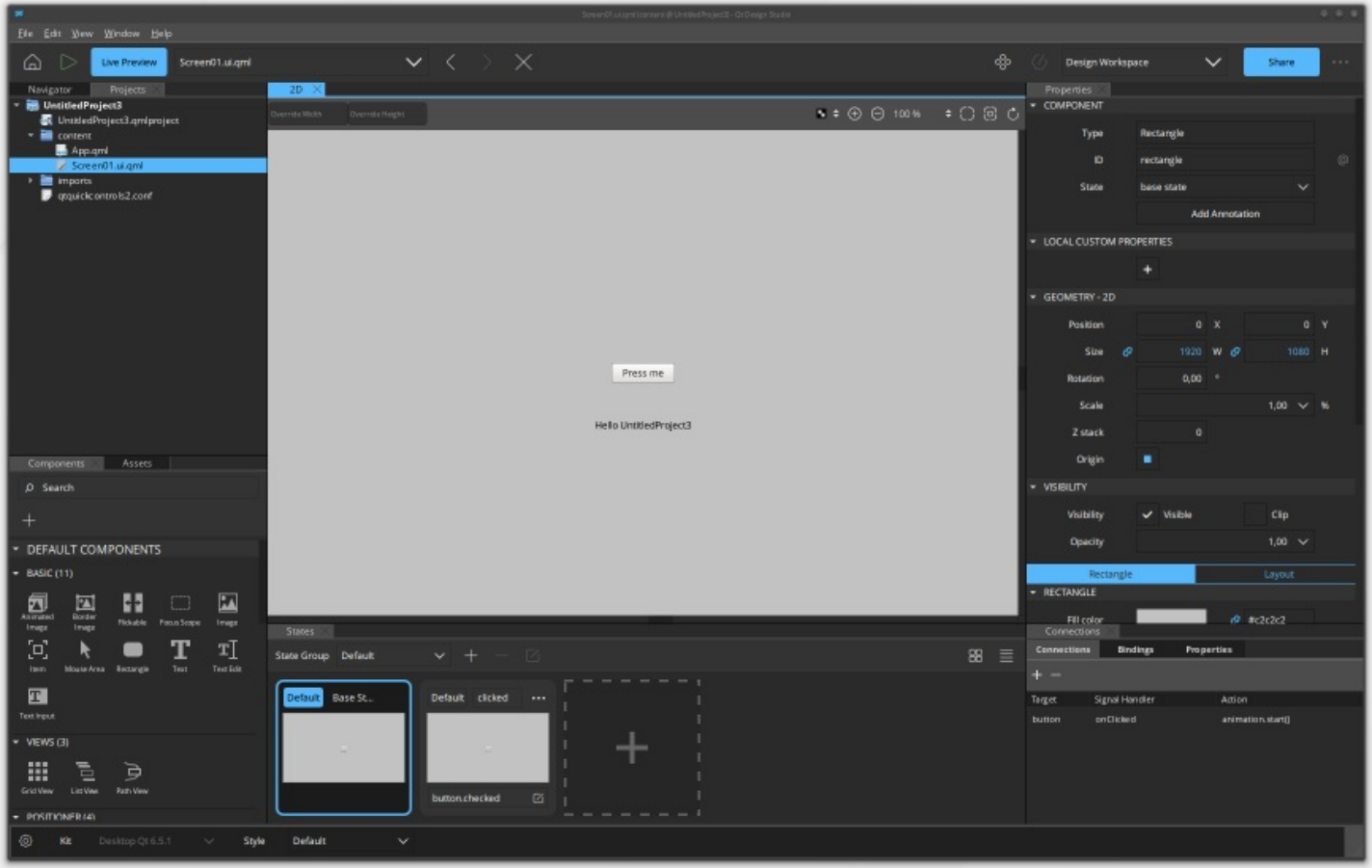


- File -> New File or Project
- Projects
  - Recent
  - General
  - Qt for MCUs
  - Mobile
  - Desktop




DS
Let's create something wonderful with Qt Design Studio!
Create new project by selecting a suitable Preset and then adjust details.
Presets
Recents
General
Qt for MCUs
Mobile
Desktop
Launcher
1920 x 1080
Empty
1080 x 1920
Scroll
1920 x 1080
3D
1920 x 1080
Empty
997 x 1080
Empty
1920 x 1080
Stack
1920 x 1080
Empty
1920 x 1080
Empty
1920 x 1080
Creates a project that uses default components, such as rectangles, images, and text, and defines a launcher application.
Details
UntitledProject3
/home/nuno/media/server/kdab/2023
Use as default project location
1920 x 1080
Width
Height
Orientation
1920
1080
Use Qt Virtual Keyboard
Target Qt Version:
Qt 6.2
Save Custom Preset
Style (8)
All
Default
Cancel
Create

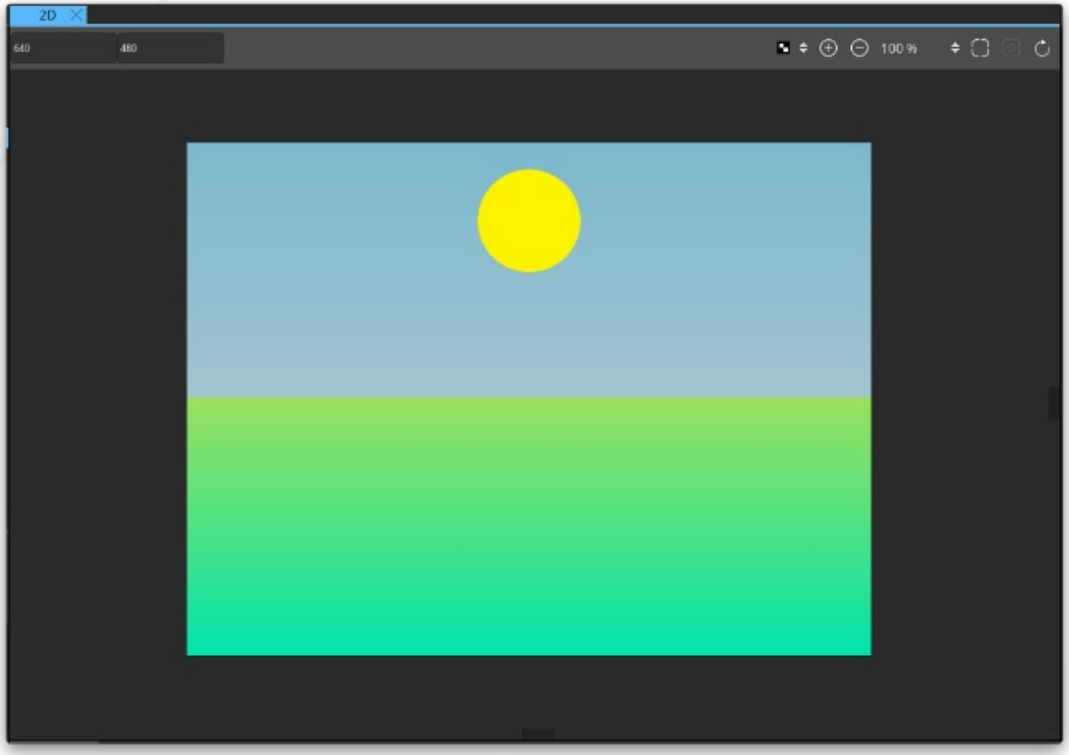
- Let's Start.
- **Main UI, Tabs & Setup.**


File Edit View Window Help
Live Preview
Screen01.ui.qml
Design Workspace
Share
Navigator
Projects
UntitledProject3
content
App.qml
Screen01.ui.qml
imports
qtquickcontrols2.conf
2D
Press me
Hello UntitledProject3
Properties
COMPONENT
Type
Rectangle
ID
rectangle
State
base state
Add Annotation
LOCAL CUSTOM PROPERTIES
GEOMETRY - 2D
Position
Size
Rotation
Scale
Z stack
Origin
VISIBILITY
Visibility
Visible
Clip
Opacity
Rectangle
Layout
RECTANGLE
Components
Assets
Search
DEFAULT COMPONENTS
BASIC (11)
VIEWS (3)
States
State Group
Default
Base St...
clicked
button.checked
Connections
Bindings
Properties
Target
Signal Handler
Action
button
onClicked
animation.start()
Kit
Desktop Qt 6.5.1
Style
Default

- Home
  - Play
  - Live Mode
- UI Files Selector
  - Component Creation
    - Edit Component
- Qt-DS Edit Mode
- Share to web assembly

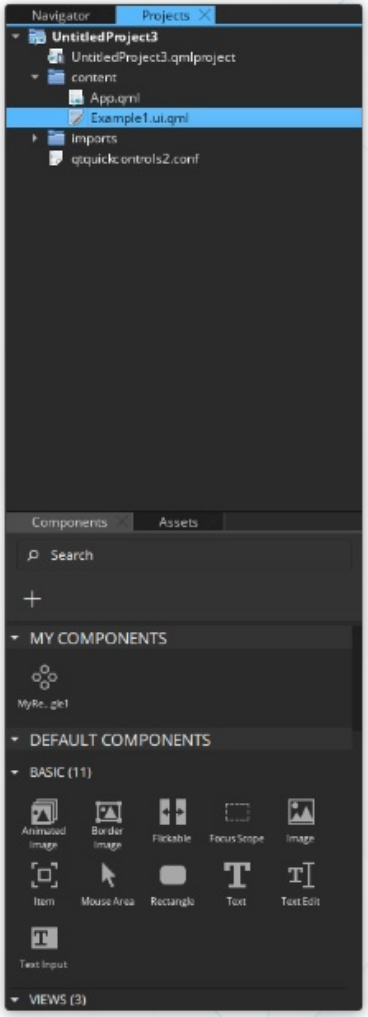
Form Editor

- WYSIWYG



Left Column

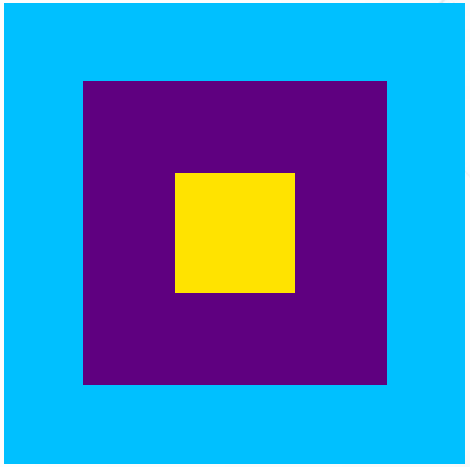
- Navigator
  - Navigator, Project
- Library
  - Components
    - QML types, Resources, Imports
  - Assets



- Properties
- Connections & Bindings

- Drag and drop.
  - Change properties in Form Editor.
  - More properties in Properties panel
  - Z Level defined by Navigator stack order

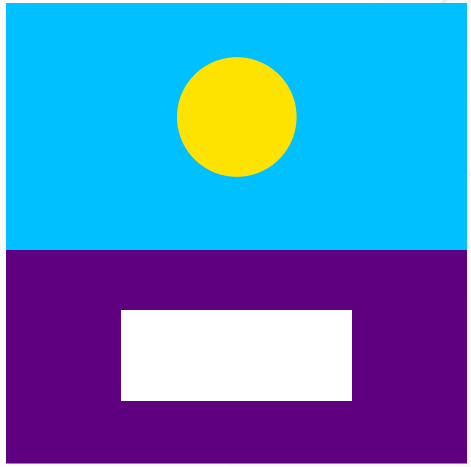
Demo: qtDesignStudio/ex-rect



Summary

- Application Based on Qt Creator.
  - Main UI area: Design mode.
  - Traditional Elements - WYSIWYG area - Properties

Try to recreate the image .



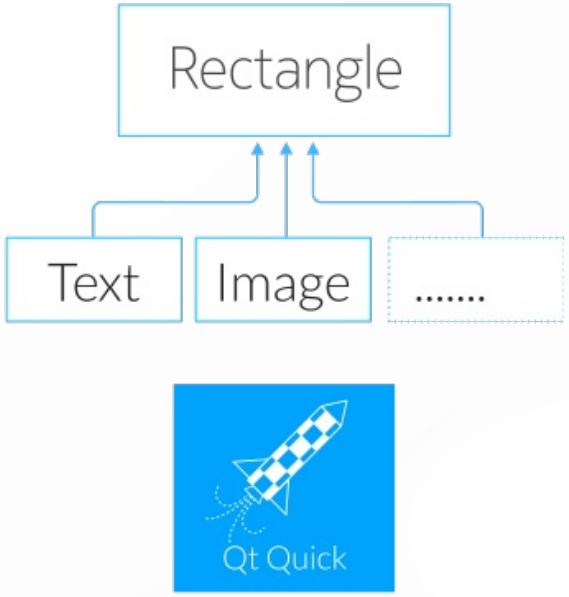
1. Recreate the scene using Rectangle items.

- Qt Design Studio Overview
- **QML Fast Intro**
  - Anchor Layout

Declarative language for user interface elements:

- Describes the user interface
  - How elements look
  - How elements behave
- UI specified as tree of elements with properties


Rectangle
width, height, color, ...
Text
x, y, font.pointSize, color, ...
Image
x, y, source, ...


Rectangle
Text
Image
.....
Qt Quick

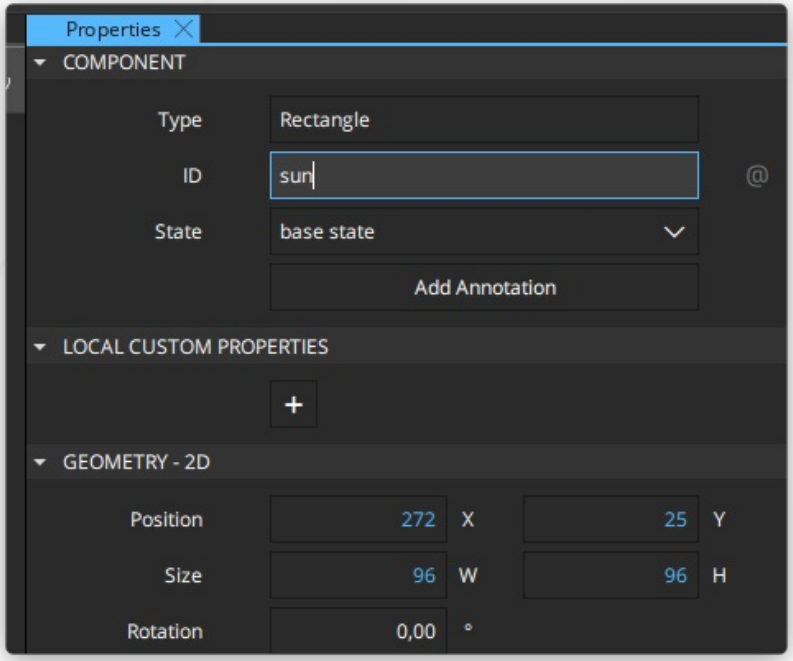
```
1  import QtQuick 2.0
2
3  Item {
4      width: 220; height:  220
5
6      Rectangle {
7          x:10; y: 10
8          width: 200; height: 200
9          color: "lightblue"
10         border.color: "#a06060ff"
11         border.width: 3
12     }
13 }
```

- Locate the example: *ex-rectangle.qml*
- Open in Text Editor:
- Blocks with list of properties
  - Block opened and closed by { }
  - Uppercase letter must be used for the block type
  - Properties start with a lowercase letter and has the value after a :

- Not visible itself
- Has a position and dimensions
- Usually used to group visual elements
- Often used as the top-level element

Elements are have specific properties:

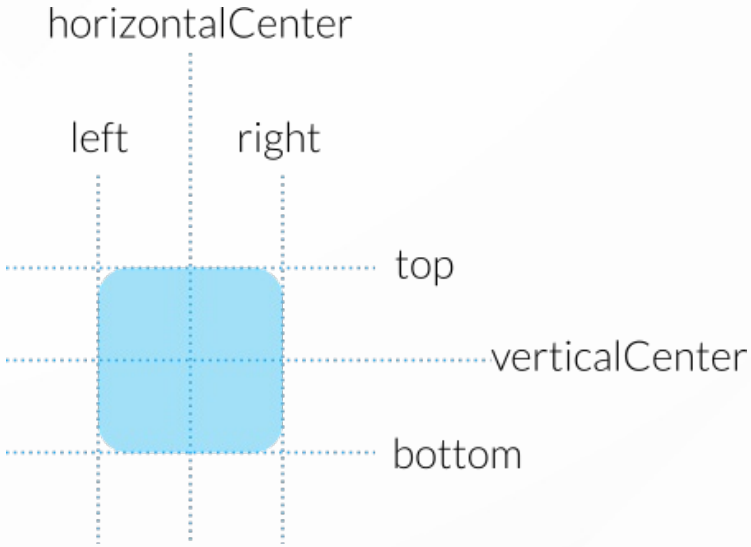
- Simple name-value definitions
  - width, height, color, ...
  - with default values
  - separated by semicolons or line breaks


Properties
COMPONENT
Type
Rectangle
ID
sun
@
State
base state
Add Annotation
LOCAL CUSTOM PROPERTIES
+
GEOMETRY - 2D
Position
272
X
25
Y
Size
96
W
96
H
Rotation
0,00
°

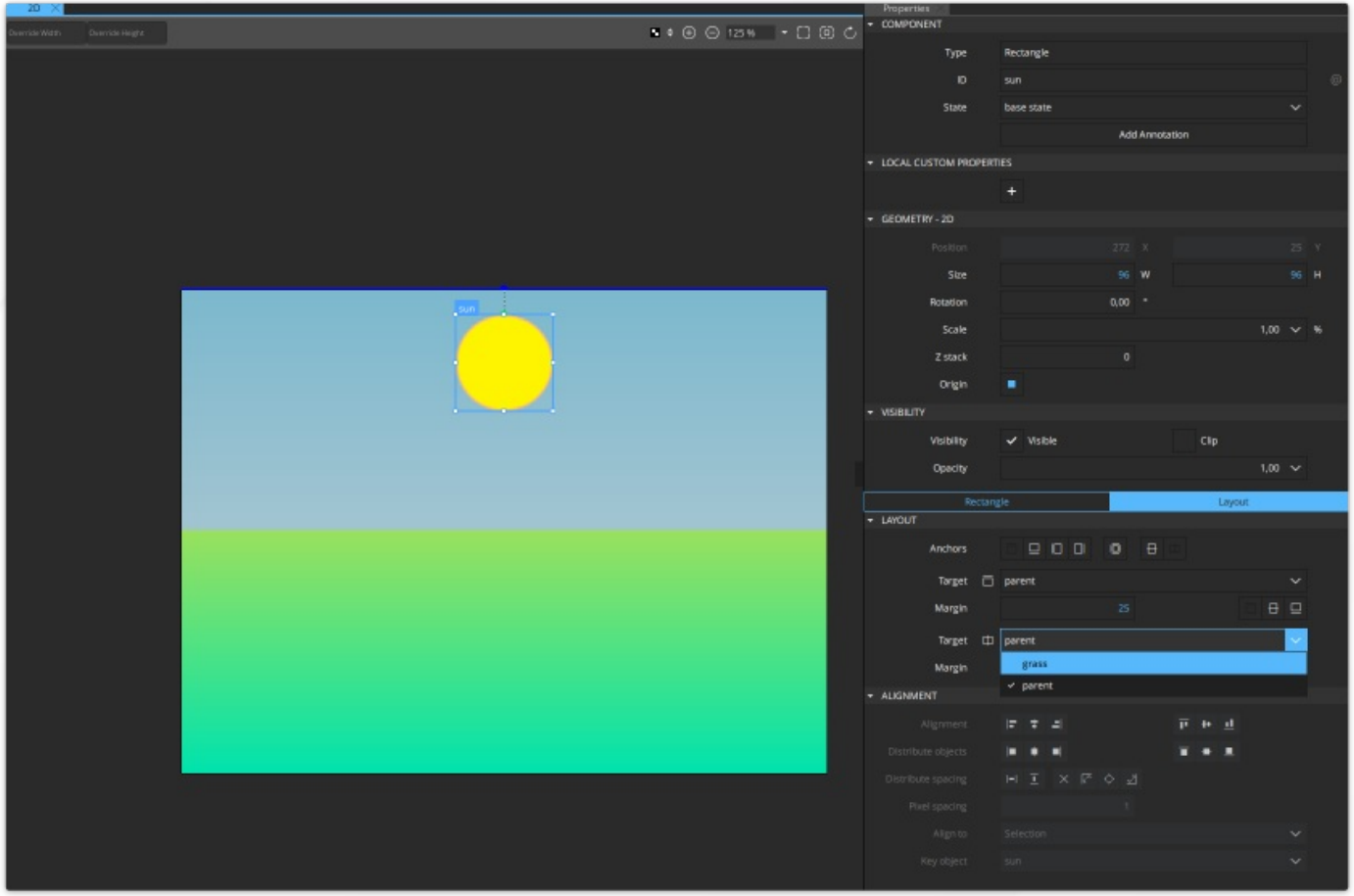
- **Anchor Layout**

- Mechanism to position Items relative to each another
- Edit anchors on properties panel, Layout tab.
- Two kinds of Anchor bindings
  - Anchor lines (left, top etc...).
  - Anchor helpers (margins).

- Line up the edges or central lines of items.
- Anchor lines can only be bound to another compatible anchor line

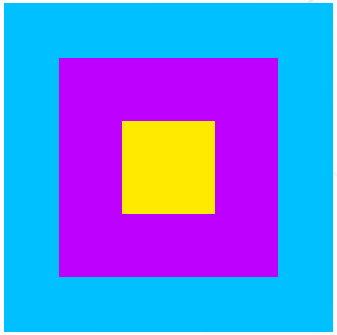


Qt Docs: Layouts In QML



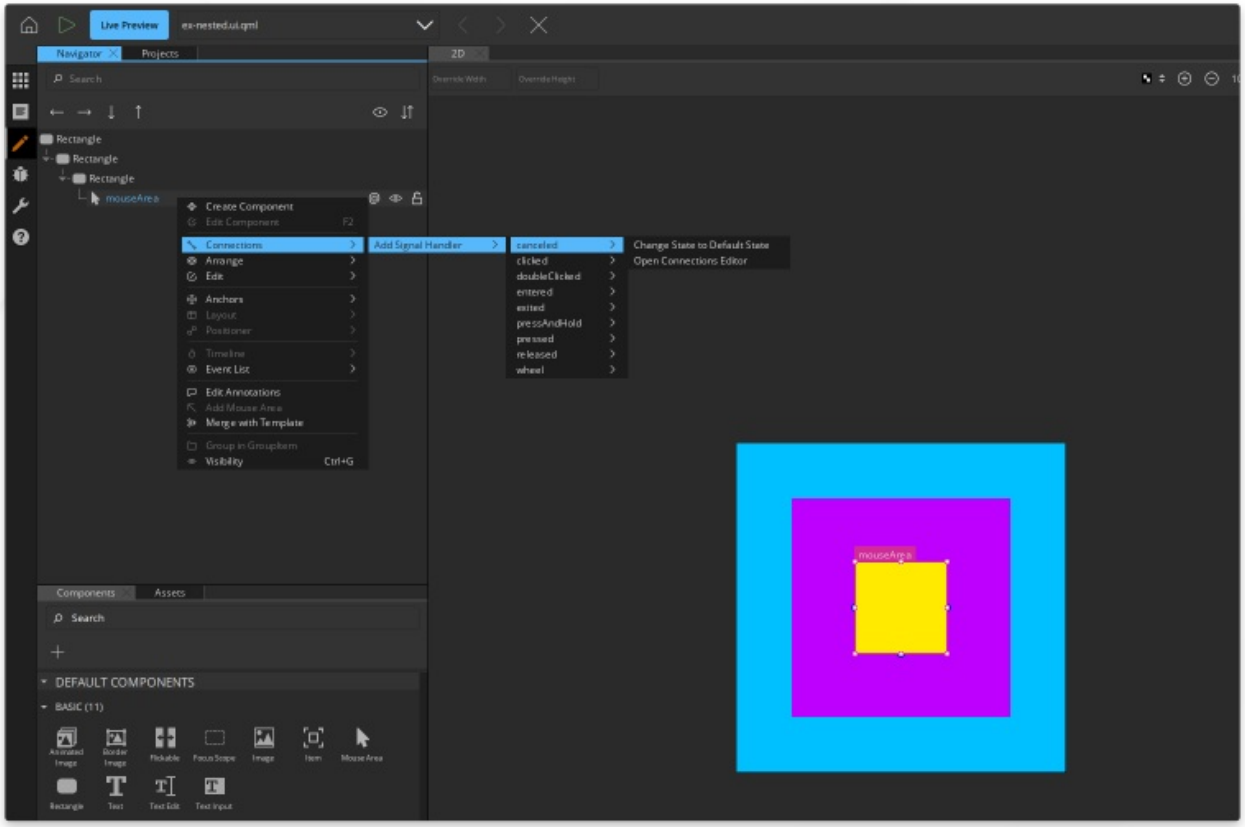
- Constrains the alignments of an item edge relative to another one
- Anchors of other items are referred to directly

```qml
import QtQuick 2.5

Rectangle {
    width: 400
    height: 400
    color: "#00c0ff"
    Rectangle {
        id: pinkSand
        x: 67
        y: 67
        width: 266
        height: 266
        color: "#be00ff"
        Rectangle {
            id: notPink
            x: 77
            y: 77
            width: 112
            height: 112
            color: "#ffea00"
    }
  }
}
```



- Nested Rectangle elements
- Each element positioned relative to its parent

Demo: qtDesignStudio/ex-nested

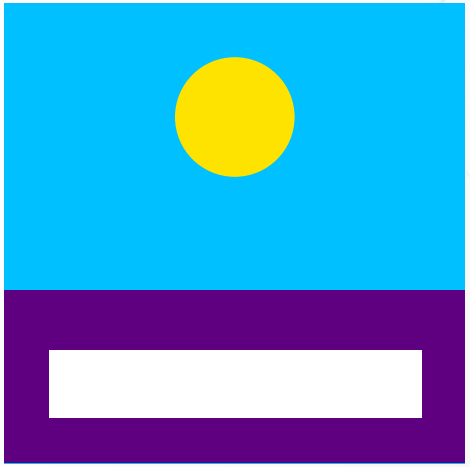


Demo: qtDesignStudio/ex-mouse

Summary

- QML defines user interfaces using elements and properties.
  - Elements are the structures in QML source code.
  - There are visible and invisible elements.
  - Elements can be nested
- Standard elements contain properties and methods.
  - Properties can be changed from their default values.
  - id properties give identities to elements.
- Anchors, positional properties between elements.
  - Can be used to describe position and size.

The image on the right shows two items and two child items inside a 300 x 300 rectangle.



1. Use Anchors for placement

2. Can items overlap?

- Experiment by moving the Sun or Sand rectangles.

3. Can child items be displayed outside their parents?

- Experiment by moving objects around.

Lab: qtDesignStudio/lab-001/lab-001

- Qt Design Studio Application
- **Import from Figma / Photoshop / etc**
  - Figma
  - Importing from PS into QtDS
- Animations in QtDS
- 3D Incorporation
- Introduction to Qt Design Studio
- QML Integration

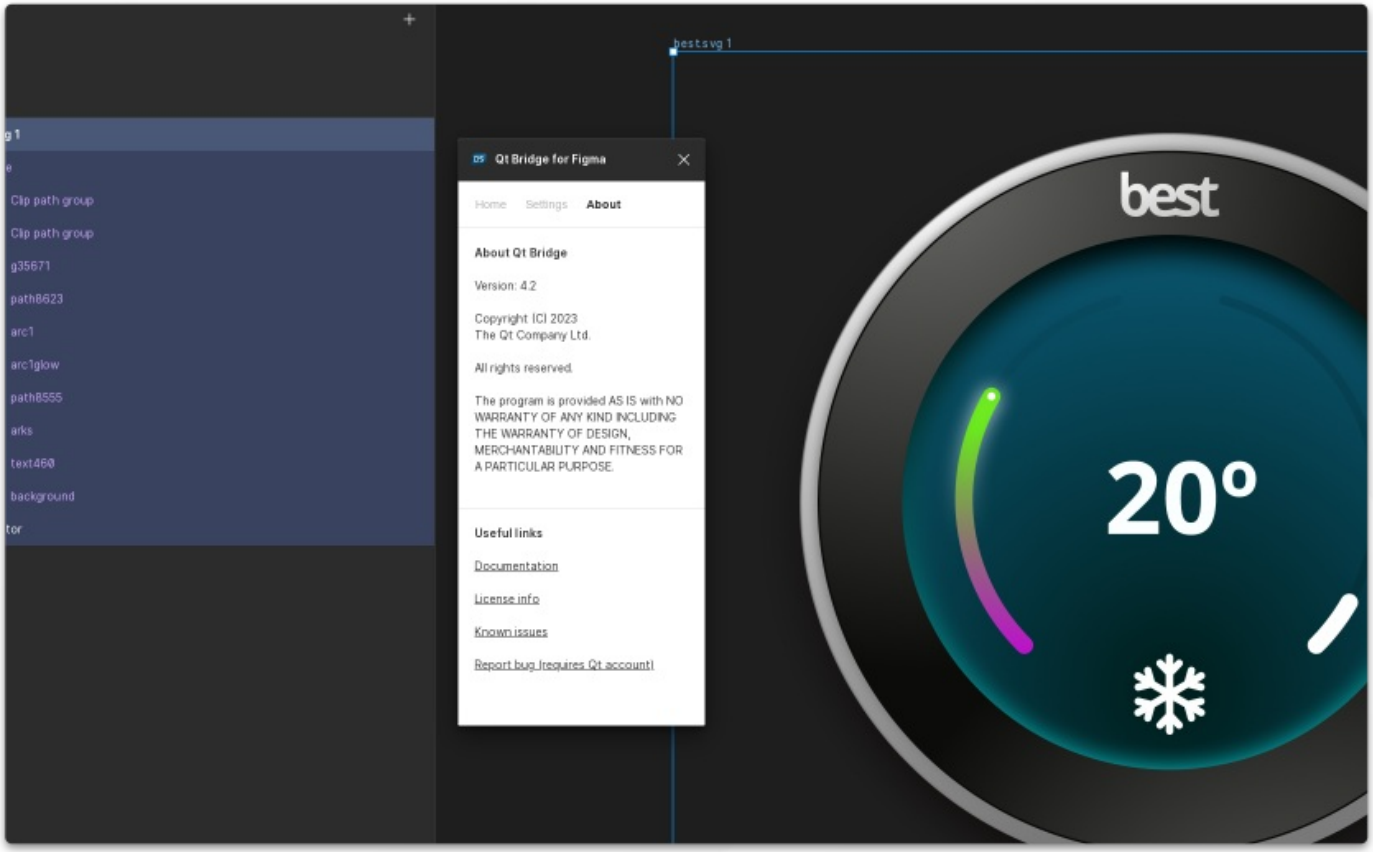
Import from Figma / Photoshop / etc

- **Figma**
  - Qt Figma Plugin
  - Resource Preparation
  - Project Creation & Import in QtDS
- Importing from PS into QtDS

- **Qt Figma Plugin**
- Resource Preparation
- Project Creation & Import in QtDS

A tool to import assets from Figma to QtDS

https://www.figma.com/community/plugin/1167809465162924409/Qt-Bridge-for-Figma


best.svg 1
Clip path group
Clip path group
g35671
path8623
arc1
arc1glow
path8555
arks
text460
background
Qt Bridge for Figma
Home
Settings
About
About Qt Bridge
Version: 4.2
Copyright (C) 2023
The Qt Company Ltd.
All rights reserved.
The program is provided AS IS with NO WARRANTY OF ANY KIND INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
Useful links
Documentation
License info
Known issues
Report bug (requires Qt account)
best
20°

- Qt Figma Plugin
- **Resource Preparation**
- Project Creation & Import in QtDS

You Can Export

- Images
  - Raster (jpg, png)
  - Vector Images (svg)
  - SVG Path Item
- Layers
  - Text layers
  - Shape layers
- Components
  - Component instances
- Frames
- Groups

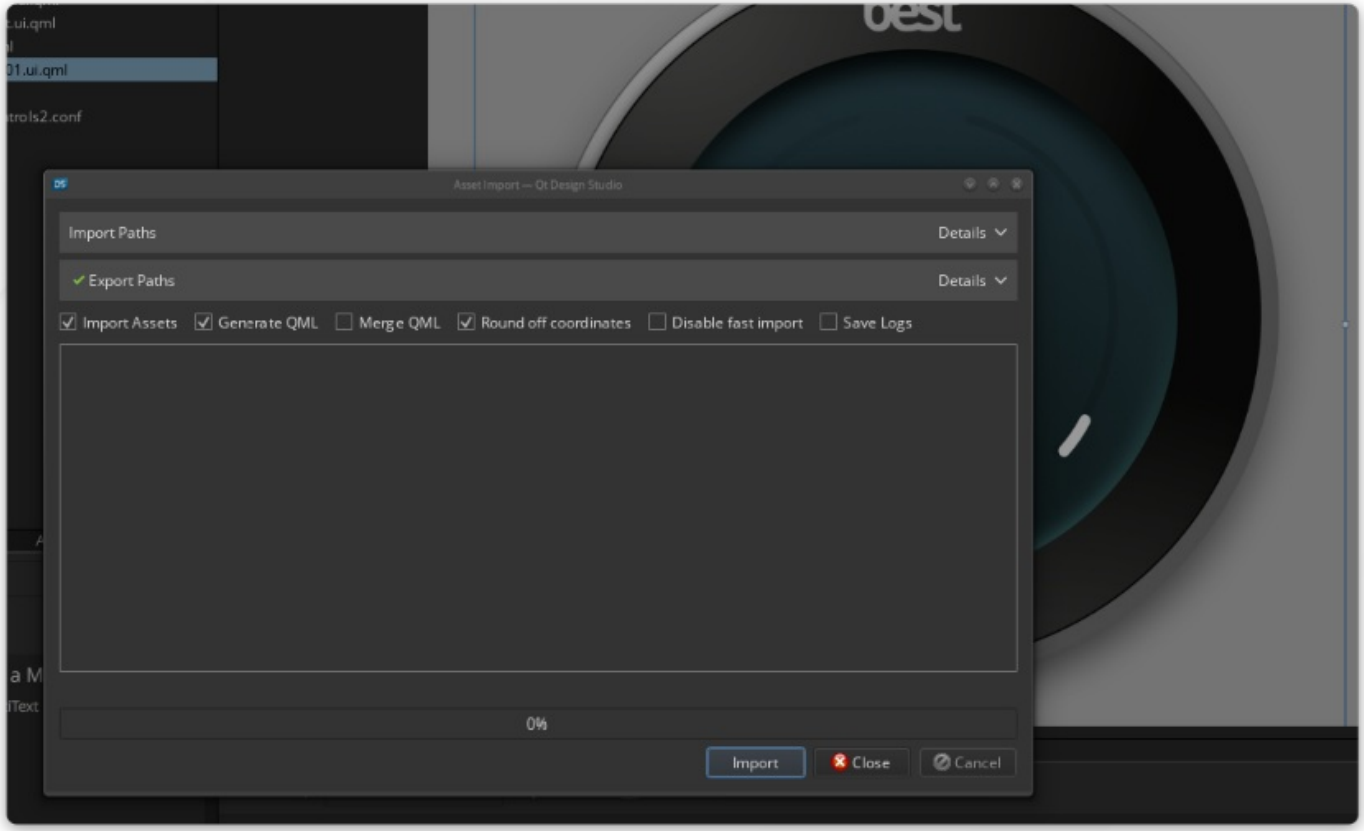
Four options for each Layer/Item

- Component
  - Top level container, (not exactly a component, but more like a page)
- Child
  - Ensures that the element will not be merged with other layers
- Merged
  - Multiple elements will be merged with this option (usually as an image)
- Skipped
  - Will not be exported

- Frames are exported as components of the Rectangle (if supported) type by default.
- Figma Components are exported as QML components, variants are described with states, use them.
- Effects tend to be exported as Rasterized images.
- Use descriptive and unique IDs
- After initial export its more convenient to do incremental exports.

- Effects rasters might show out of place and incorrectly sized
- Font files are not part of the export
- No specific support for multiple sizes
  - SVG comes with is own set of issues (but can be more scalable)

- Qt Figma Plugin
- Resource Preparation
- **Project Creation & Import in QtDS**


Asset Import — Qt Design Studio
Import Paths
Details
Export Paths
Details
Import Assets
Generate QML
Merge QML
Round off coordinates
Disable fast import
Save Logs
0%
Import
Close
Cancel

- Go to Qt Bridge for Figma plugin page.
  - Select 'Try it out'
  - Select your logged in Figma account
  - After the plugin loads, select Run
- Prepare the Design for export.
  - Select Export, to get the .qtbridge file in your local drive
- In Qt Design Studio, drag the file to the 2D, 3D, Assets, or Navigator view in an open project
  - To use the imported Elements remember to import the directory created.

- Export Figma file into QtDS
  - Group layers into logical components
  - Set the correct export options
  - Export assets
- Create new QtDS project
  - Import assets into QtDS

Lab: qtDesignStudio/lab-01/lab-01

- Figma
- **Importing from PS into QtDS**
  - Qt Bridge
  - Resource Preparation
  - Project Creation & Import in QtDS

- **Qt Bridge**
- Resource Preparation
- Project Creation & Import in QtDS

A tool to import assets from Figma-Photoshop-etc to QtDS

Linking Qt Bridge to Photoshop. Step by Step.

1. Download, install and Run ZxpInstaller (https://zxpinstaller.com/)

2. Find the install of Qt Designer studio and locate the .zxp package inside the Photoshop_bridge folder

3. Drag it into the ZxpInstaller window.

4. Close ZxpInstaller and open up Photoshop

5. Open up the preferences editor: Edit > Preferences > General > Plug-ins

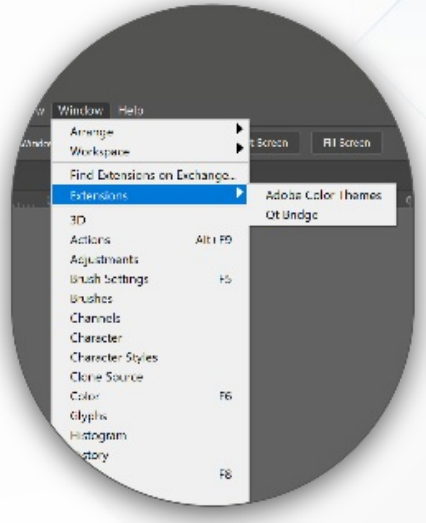
6. Make sure there is a tick in the Enable Remote Connections check box and add a password in the Password field

7. Open up Qt Bridge

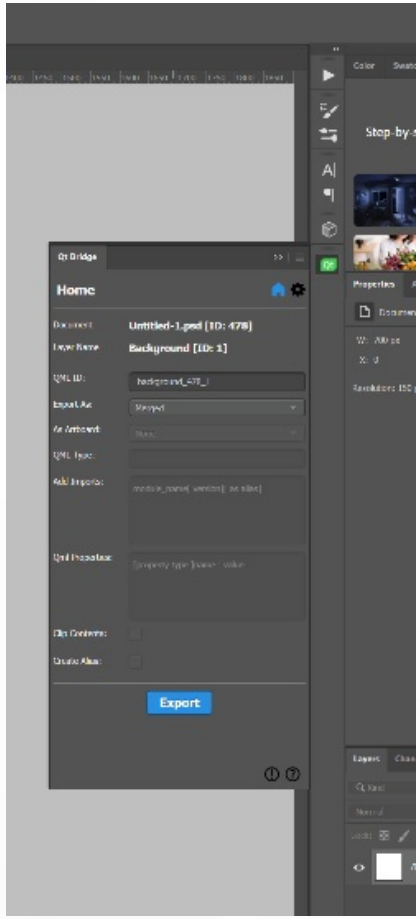
8. Click on the cog at the top right to open up settings

9. Enter the password you created in step 6

10. Press the test connection button to check that everything is working



Where is it?



- Qt Bridge
- **Resource Preparation**
- Project Creation & Import in QtDS

Four options for each layer, group or artboard

- Component
  - Top level container, used as a logical container for other items - usually a group or artboard in Photoshop
- Child
  - Only option for text (other than skipped)
  - Ensures that the layer will not be merged with other layers
- Merged
  - Multiple layers will be merged with this option
- Skipped
  - Will not be exported

Demo: qtDesignStudio/ex01/ex_exporting.psd

- Ensure the layers are in a single top layer folder or artboard, otherwise there may be unexpected results
- A child layer of a skipped group/artboard is not skipped on export unless explicitly set as skipped
- Artwork is clipped outside artboard bounds on export
- Everything you want to work with and export must have a unique QML id
  - Good practice to specify a custom id for each element

Export Results

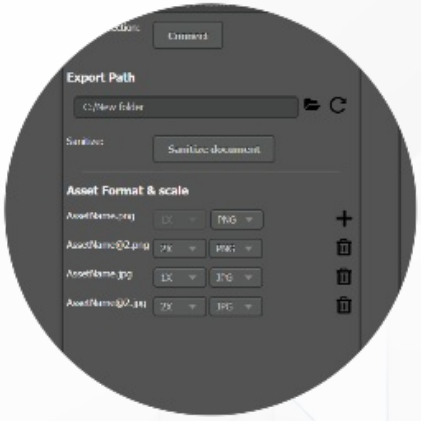
- The export location defaults to a folder in the current save location of the .psd file.
  - This can be overridden in the options
  - Saving and immediately exporting without specifying a location may result in an unexpected location for the exported files
- It is best to check the resulting images before importing into QtDS

Smart objects in Photoshop

- in photoshop these are linked to the same item, however on export they are treated a different items and each instance is exported separately
- a smart object in photoshop consisting of different layers (ie a photoshop smart object) will be exported as a merged object - even if it contains text

Scalability

- When using vector objects imported from Illustrator, it may be useful to export multiple sizes of the images (depending on the size of the target platform)



- Text export does not respect line breaks - a \r is added where qt design studio expects a \n.
- Line height in a Text Area does not get changed if the artboard has been resized in Photoshop
- Text alignment is different on mac/windows - slightly misaligned on windows.
- Reimporting images (via add new resources as there is no other option) deletes animations even on same named objects
- Exporting as jpg
  - the files exported as .jpg but are named .png (this is a bug in version 1.1.1)
  - generated code has links to .png files, will not work in QtDS as the file format is not .png

- Qt Bridge
- Resource Preparation
- **Project Creation & Import in QtDS**

Summary

- Install Qt Bridge.
  - Set artboards.
  - Set correct grouping.
  - Think about scalability.
  - Name your elements.
- Use Add Resources to add an export to QtDS .

- Import photoshop file into QtDS
  - Group layers into logical components
  - Set the correct export options
  - Export assets
  - Create new QtDS project
  - Import assets into QtDS

Lab: qtDesignStudio/lab01/lab

Solution: qtDesignStudio/lab01/solution02

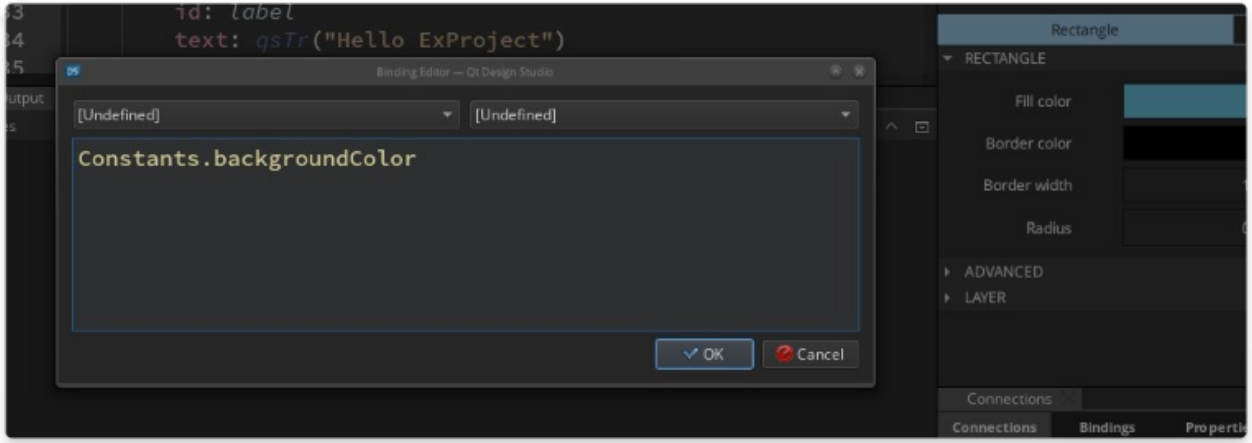
Solution: qtDesignStudio/lab01/solution01

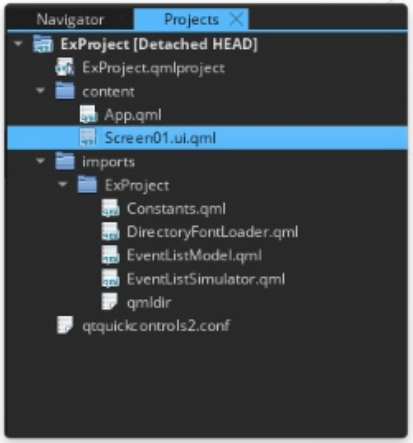
- Qt Design Studio Application
- Import from Figma / Photoshop / etc
- **Animations in QtDS**
  - Preamble
  - Animations
- 3D Incorporation
- Introduction to Qt Design Studio
- QML Integration

- **Preamble**
- Animations

QtDS Tipical Project Structure

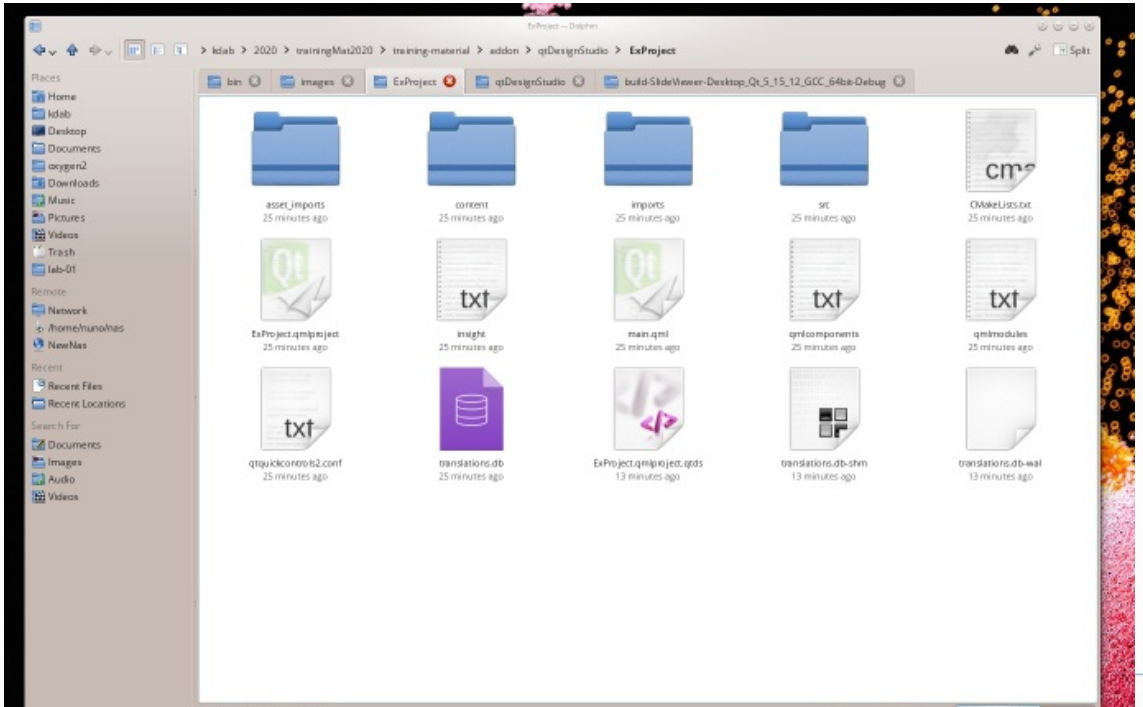
- Project (apparent root folder)
- Content
- Imports
  - Constants





QtDS Tipical Project File Structure

- Project (real root folder)
  - content
    - fonts (use to automatically include font files)
  - Asset_imports (Imported 3D assets and components)
  - Imports
    - ProjectFolderDef (same as imports in project view)
  - src (c++ files, do not touch)

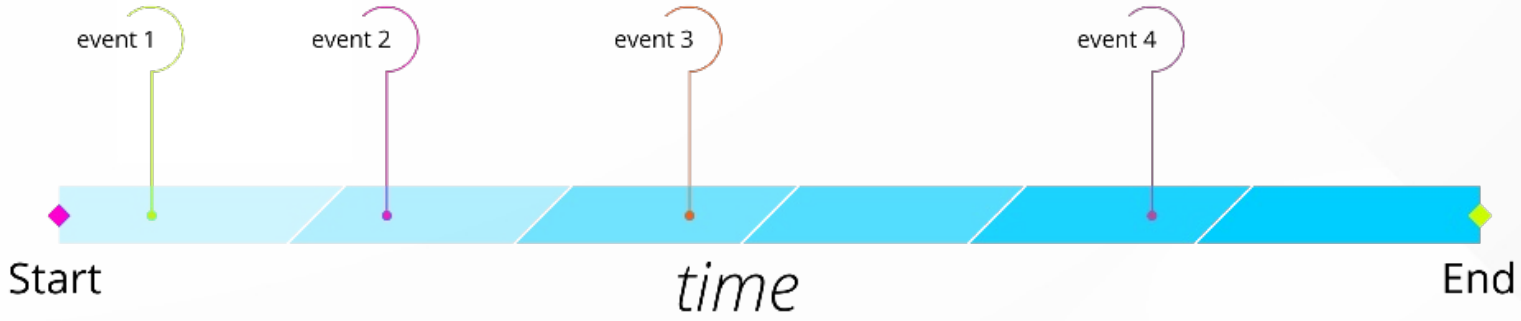


Example...

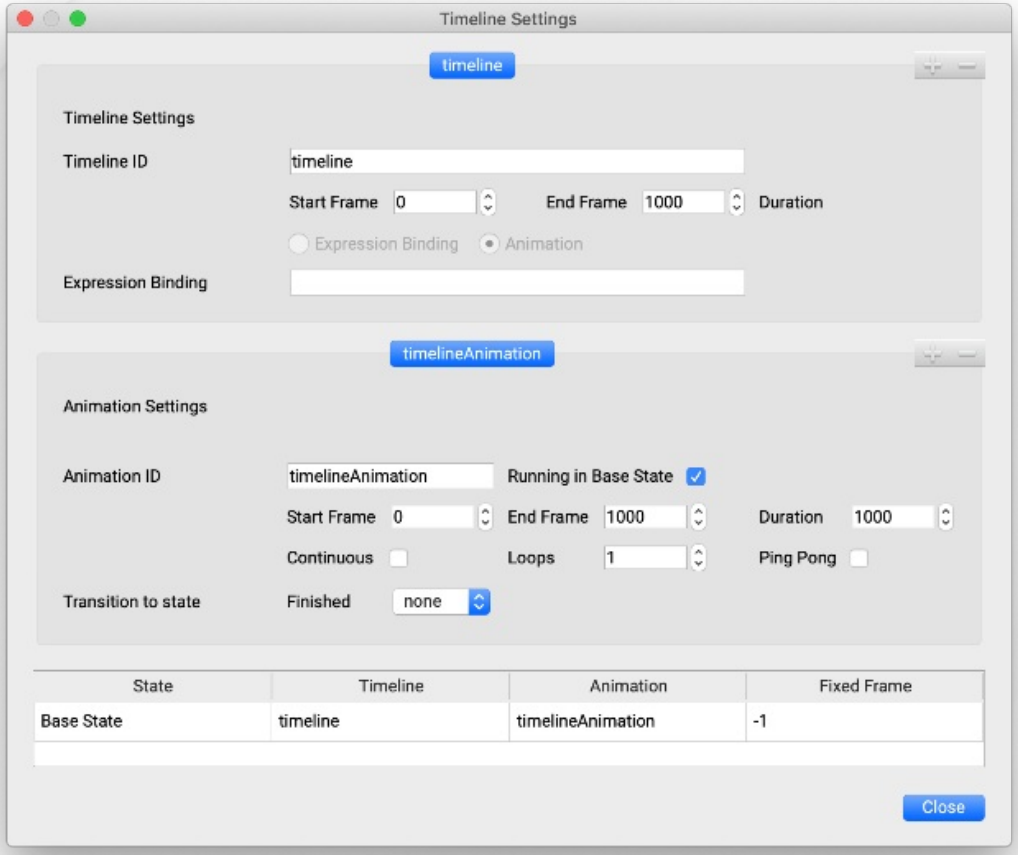
Demo: qtDesignStudio/ex-project

- Preamble
- **Animations**

- Time based change to properties
- Key-frames - recorded point in time



- Create Timeline
  - Create Timeline Animation



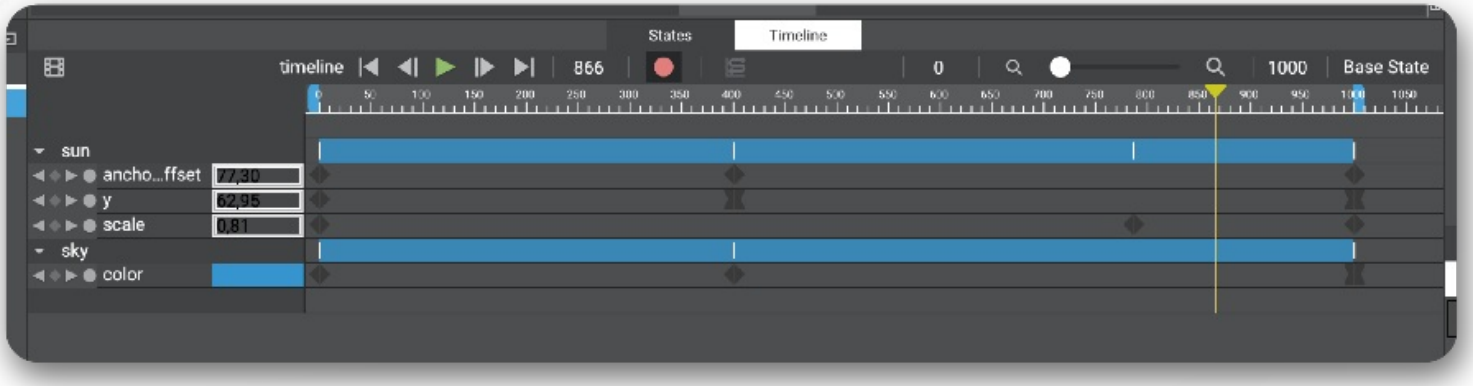
Create a Timeline in QtDS (cont'd)

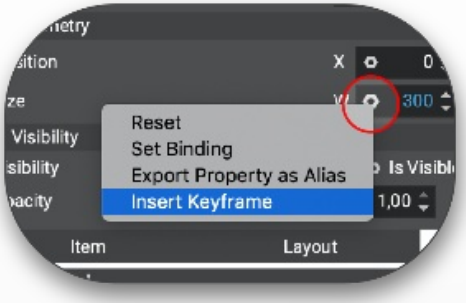
Click plus symbol on Timeline pane - note that a timelineAnimation is created by default for the timeline.

- Start Frame / End Frame
  - these are the available frames for the animation
  - note the duration in time can only be set in the animation panel
- Running in Base State
  - means the animation is running by default
- Start Frame / End Frame
  - these are the available frames for the animation
- Duration
  - the length of the animation in milliseconds
- Continuous / Loops
  - selecting continuous mean that it runs forever (and changes the number of loops to -1)
  - otherwise a definite number of loops can be set
- Ping Pong
  - runs the animation once forwards, and immediately once backwards

Example...

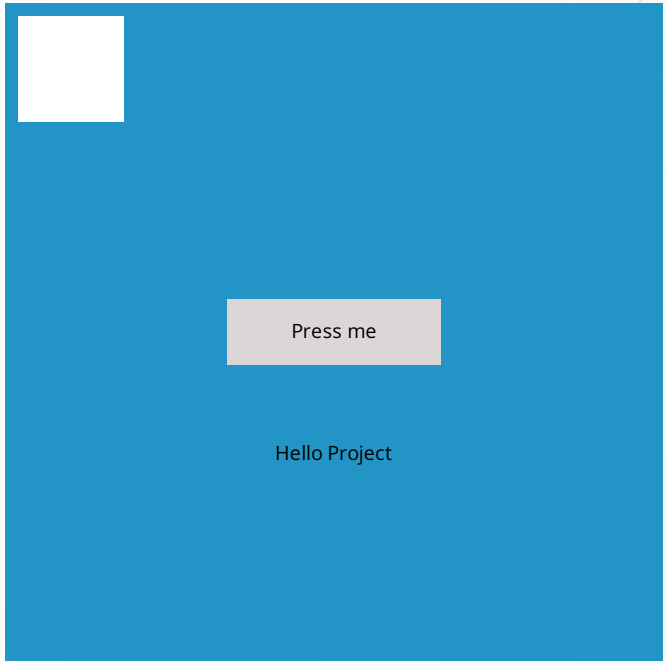
- 2 methods of creating key-frames
  - Use record button
  - Explicitly setting a key-frame on '*nut*' icon





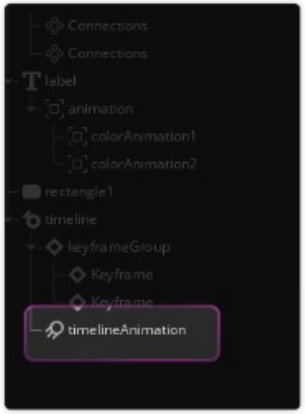
Example continuation...

Demo: qtDesignStudio/ex-Anim



Triggering an Animation. Multiple ways

- set a connection by a trigger event
  - *id*.running = true
  - *id*.start()
  - *id*. .....
- can also set a Binding in the running property
  - not ideal for signals()

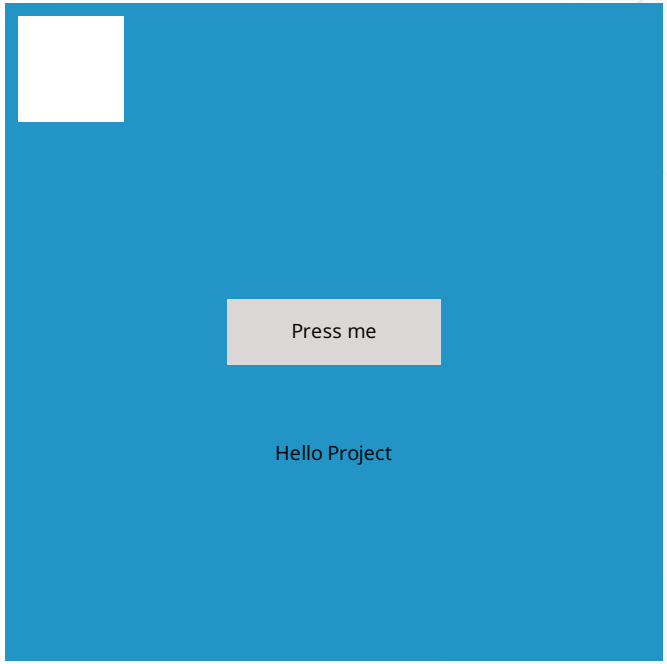


Sequential, Paralel, Number, Color etc Animations

All properties can be Animated

- Grouping Animations
  - Multiple Animations can be set in serial fashion with SequentialAnimation
  - Multiple Animations can be run in parallel with ParallelAnimation
- Property Animations
  - NumberAnimation
  - ColorAnimation
  - RotationAnimation
- Pause Animation We Cover theese Animations further on the QML Deep View Section

Demo: qtDesignStudio/ex-Anim



Summary

- Timelines provide you a method of defining animations
- Composed of frames
  - Each frame has a defined duration
- Key-frames are used to define specified changes in properties at the beginning and end of the transition
  - Easing curves define how these transitions occur in time
- To be able to have multiple independent animations, they must be created on separate components
- Properties can also be Animated by TypePropertyAnimation
  - Create more complex animations Via parallel and Sequential Animations

Animate Lab2

- Animate the arc using the properties *begin* and *end*, looping 6 times
  - *We will cover arc and other shapes more extensively later in this trying.*
- **Extra**
  - *Animate the colour of the arc*
  - *change the temperature text according to position.*

Lab: qtDesignStudio/lab-03/lab-03

- Qt Design Studio Application
- Import from Figma / Photoshop / etc
- Animations in QtDS
- **3D Incorporation**
  - Importing 3D into QtDS
- Introduction to Qt Design Studio
- QML Integration

- **Importing 3D into QtDS**
  - Intro to 3D
  - Exporting from Blender
  - Import in QtDS
  - 3D Summary...

- **Intro to 3D**
- Exporting from Blender
- Import in QtDS
- 3D Summary...

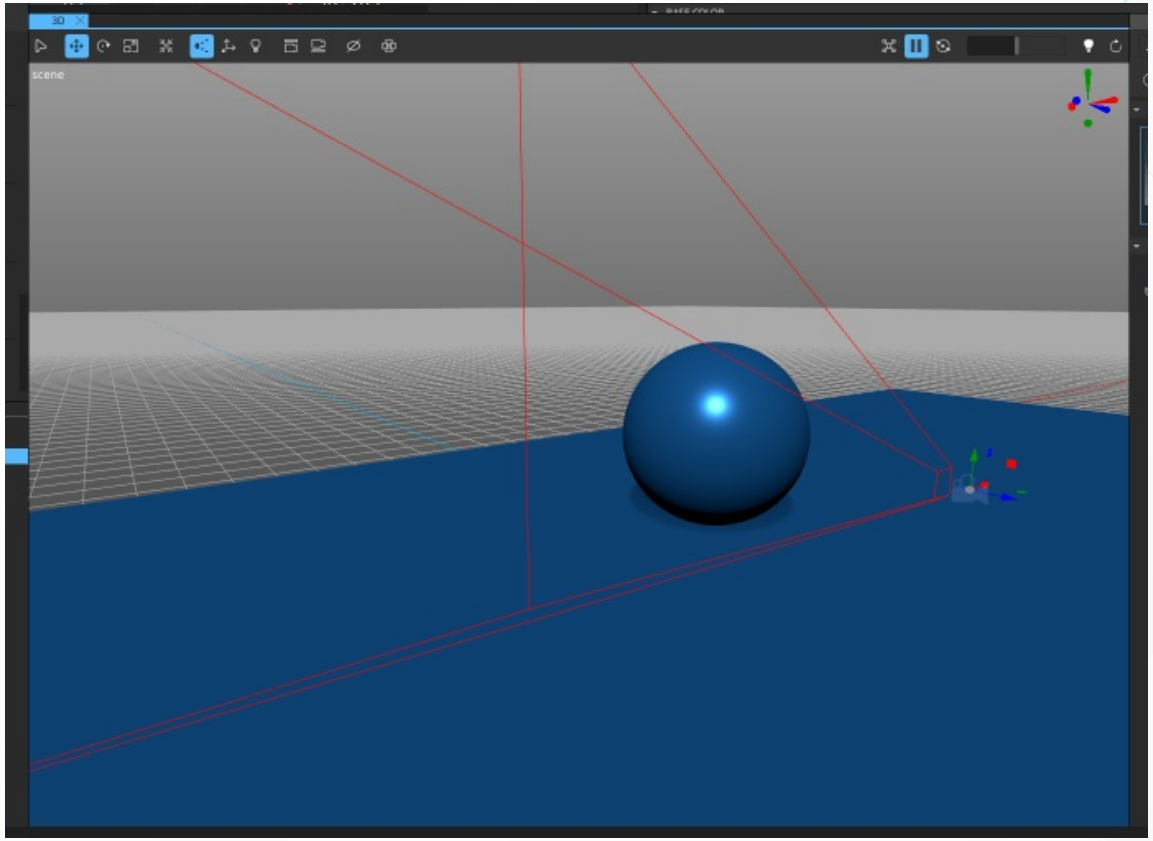
1. Before we start

- import QtQuick 6.5
- import QtQuick3D 6.5

2. Workspace Modes

- 3D Essentials
  - (contains the core work mode for 3D Manipulation)
- 3D Animation
  - (very similar to 2D Animation)
- 3D Advanced
  - (greater focus on material editing)

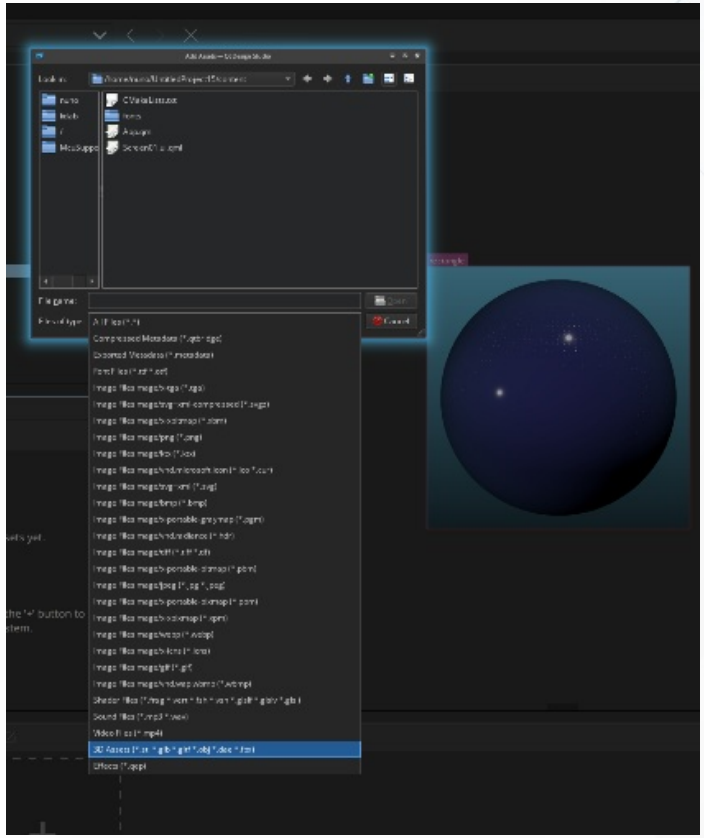
Demo: qtDesignStudio/ex-3D



- Intro to 3D
- **Exporting from Blender**
- Import in QtDS
- 3D Summary...

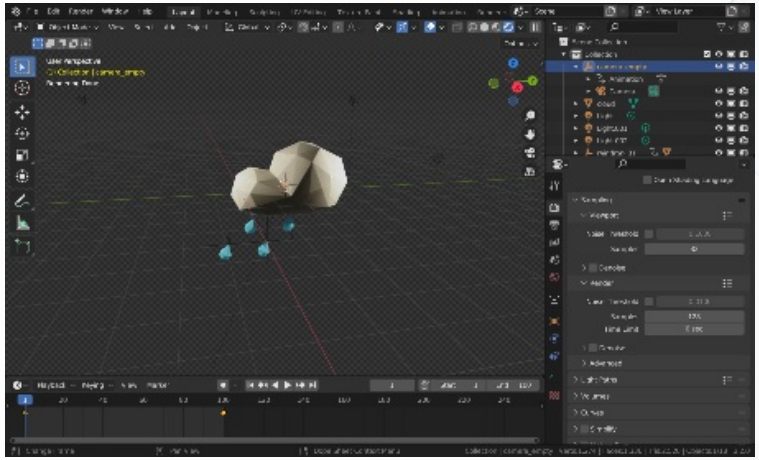
Suports,

- .gltf,
- .obj,
- .dae,
- .fbx,
- .stl,
- .glb.

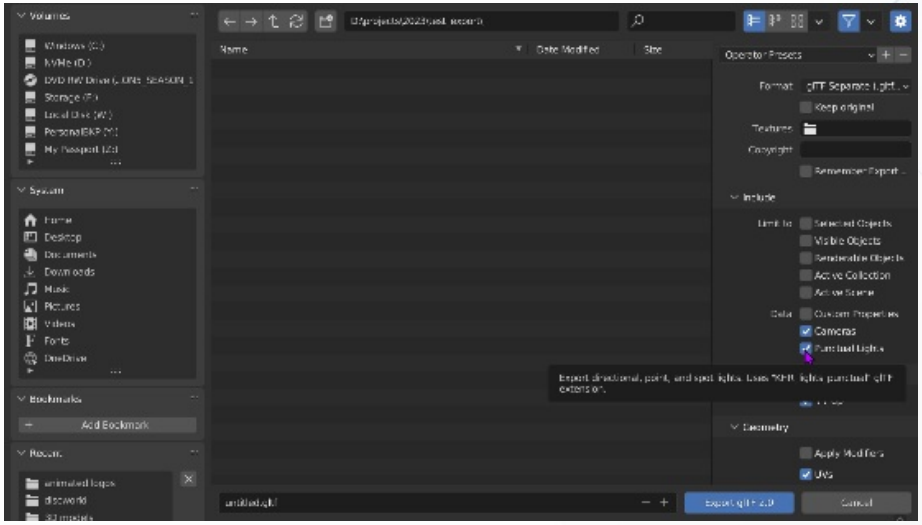


Supported Objects.

- Meshes
- Materials (Principled BSDF) and Shadeless (Unlit)
- Textures
- Cameras
- Punctual lights (point, spot, and directional)
- Animation (key-frame, shape key, and skinning)



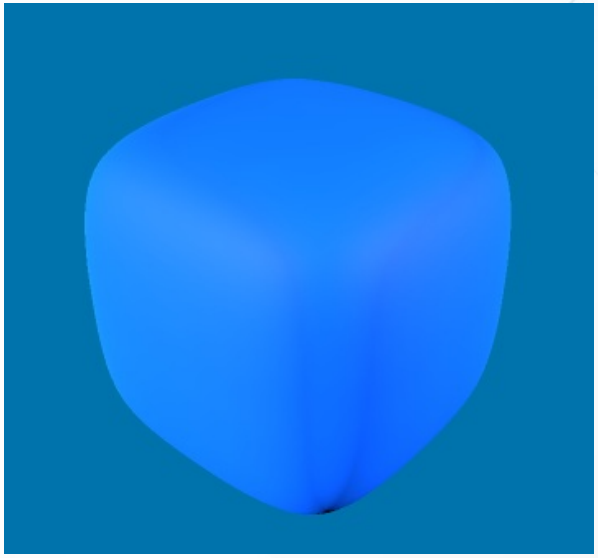
- Attention (not exported)
  - global Lights
  - Node based materials
    - Procedural shaders
  - Paths.
  - Reverse kinematics.
  - etc (https://iconscout.com/blog/export-gltf-files-from-blender)



- Intro to 3D
- Exporting from Blender
- **Import in QtDS**
- 3D Summary...

- Create a View3D Component.
- Set up your materials.
- Set up your Camera, Meshes and Lights.
  - Shadows and Lights are expensive.
  - Consider baking materials reflections & shadows.

Demo: qtDesignStudio/ExSphube



- Intro to 3D
- Exporting from Blender
- Import in QtDS
- **3D Summary...**

- Set up your Camera, Meshes and Lights.
  - Shadows and Lights are expensive.
  - Consider baking materials reflections & shadows.
- When importing 3D objects consider importing a full scene
- No need to specify a camera if one exits in the import
- Animations work just the same way as in 2D.

- Import gltf into QtDS project.
  - Set View3D in correct position
  - Use Imported Scene in View3D Object.
  - Fix Running Animations

Bonus for fading out the raindrops

Lab: qtDesignStudio/lab-04/lab-04

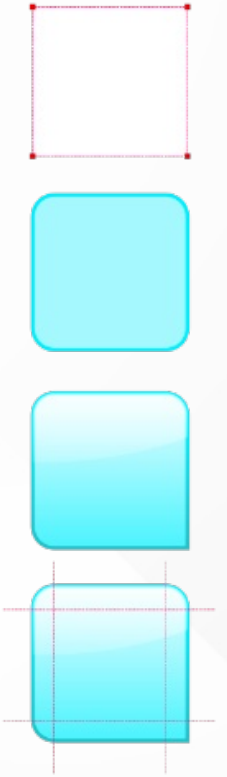
- Qt Design Studio Application
- Import from Figma / Photoshop / etc
- Animations in QtDS
- 3D Incorporation
- **Introduction to Qt Design Studio**
  - Adding Components & User-Interaction
  - States
  - Misc
- QML Integration

- **Adding Components & User-Interaction**
  - Components & User-Interaction
- States
- Misc

- **Components & User-Interaction**
  - Components
  - Interactable Presentation Items
  - Interaction Components

- **Components**
- Interactable Presentation Items
- Interaction Components

- Item
- Rectangle
- Image
  - (multiple scale/fill methods available)
- BorderImage
  - (multiple scale/fill methods available)
- Text
- TextEdit & TextInput



- Containers & Positioners for multiple items
  - Column
  - Row
  - Grid
  - Flow

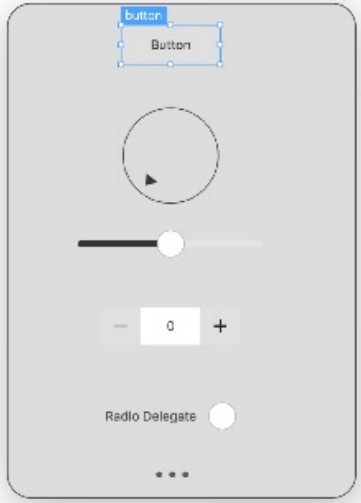
- Border & RectangleItem
  - (Similar to Rectangle item, optional radius per Corner)
- Arc & Pie
- Triangle
- SvgPath
  - (copy paste svgcode into path data property)

- Reuse Components you create
  - Navigator -> mouse option (second mouse button click) in item -> Move Component into Separate File.

- Components
- **Interactable Presentation Items**
- Interaction Components

- Flickable
  - (One single object user flickable, horizontally and/or vertically)
- ListView
  - (A list of flickable elements)
- PathView
  - (A list of flickable elements along a predefined path)
- GridView
  - (A Grid of flickable elements)

- Button
- Dial
- Check Box
- Switch
- .....



- Components
- Interactable Presentation Items
- **Interaction Components**

- MouseArea
  - MultiPointTouchArea
  - PinchArea
    - (we will cover theese more deeply on the code section)

Summary

- Components provide you with many usable premade elements.
  - Some components help you automatically place multiple elements
  - Think about scalability.
  - Name your elements.
- Use Add New Resources to add an exported element to QtDS .

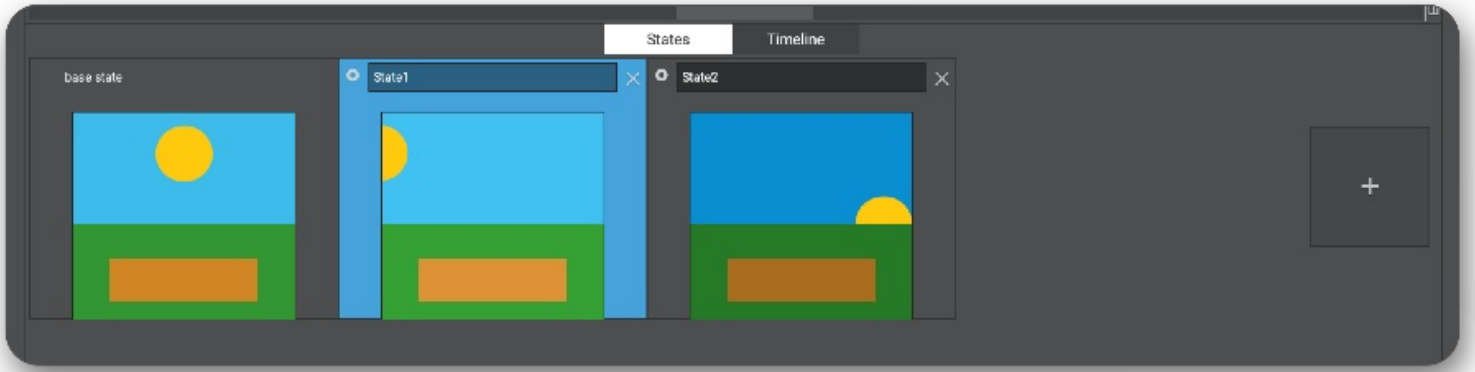
- Create a new component from the snowflake called AcImage
- Create an onClick event on AcImage to change the image to 'images/circles.png'

Lab: qtDesignStudio/lab05/lab-airConControls

- Adding Components & User-Interaction
- **States**
  - States
- Misc

- **States**

- A visual convenience
  - A collection of property changes



Using States - Example

Here we can see a .ui.qml file with 3 states, changes triggered by a mouse event.

Demo: qtDesignStudio/ex03/stateDemo

1. Base state defined (including trigger event)

2. Create second state based on the base state

3. Create a timeline for the second state (ensuring that Running in base state is selected)

4. Key-frame the relevant properties in second state at frame 0

5. Move to the relevant frame in the animation and change/keyframe them accordingly

6. Create a new Connection by clicking the plus symbol
   6.1. Ensure that Target points to trigger object
   6.2. Ensure that the Signal Handler has relevant event selected.
   6.3. Double click on Action and chose '*Change state to State1*'

Summary

- A visual convenience
  - A collection of property changes
- Useful for animation triggering in QtDS scope
- Very useful to bridge the variant concept in Figma

- Replicate the effect seen in the solution on the right using states

Lab: qtDesignStudio/lab06/lab-airConStates

- Adding Components & User-Interaction
- States
- **Misc**
  - Data, Fonts and JavaScript

- **Data, Fonts and JavaScript**

- You can had a font to your project by simply drag and droping it into Assets
  - After it can be aplied to any text in your project
  - Exported projects will use the font system independently.

- The text editor is your friend
  - Auto generated code is not that hard to read and is very helpful to understand the structure of your app.
  - The current nature of QtDS sometimes complicates things that are simpler to achieve via code editing.
- Full power of QML and Qt documentation. Some Examples of easy to do code elements and how to integrate them in QtDS...

- Anything after a ':' is JavaScript
- JavaScript is evaluated whenever an property in the code after ':' line changes.
- Triggers for Javascript can be for example a mouse event or any property change
  - onPropertyChanged()

- Specific fonts can be added to your application
  - Bundle fonts you are not sure to exist in other machines.
- The code doesn't bite (mostly)
  - The JavaScript is evaluated on change

- Qt Design Studio Application
- Import from Figma / Photoshop / etc
- Animations in QtDS
- 3D Incorporation
- Introduction to Qt Design Studio
- **QML Integration**
  - QML Deeper Dive
  - Introduction to Efectcs

- **QML Deeper Dive**
  - Composing User Interfaces
  - Animations
  - User Interaction
  - Components
  - Presenting Data
- Introduction to Efectcs

- **Composing User Interfaces**
  - Image Elements
  - Text Elements
  - Item Transformations
  - Anchor Layout
  - Colors and Gradients
- Animations
- User Interaction
- Components
- Presenting Data

- Images
- Text
  - Displaying text
  - Handling text input
- Item transformations
- Anchors and alignment
  - Allow elements to be placed in an intuitive way
  - Maintain spatial relationships between elements
- Colors and gradients
  - Create appealing UIs

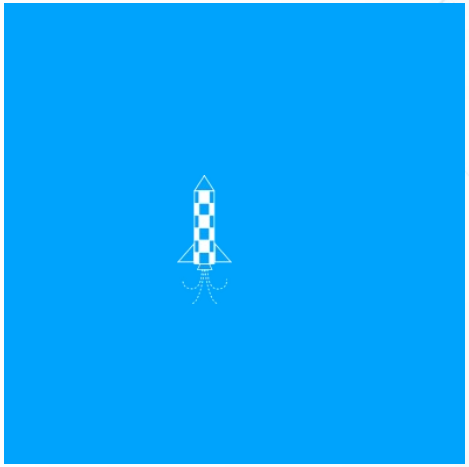
- **Image Elements**
- Text Elements
- Item Transformations
- Anchor Layout
- Colors and Gradients

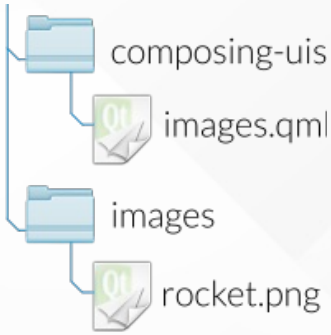
Images

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 400
5      color: "#00a3fc"
6
7      Image {
8          x: 150; y: 150
9          source: "../images/rocket.png"
10     }
11 }
```

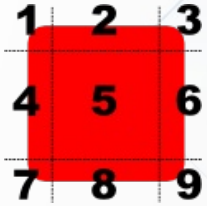
Demo: qml-composing-uis/ex-images

Demo: qml-composing-uis/ex-images-network





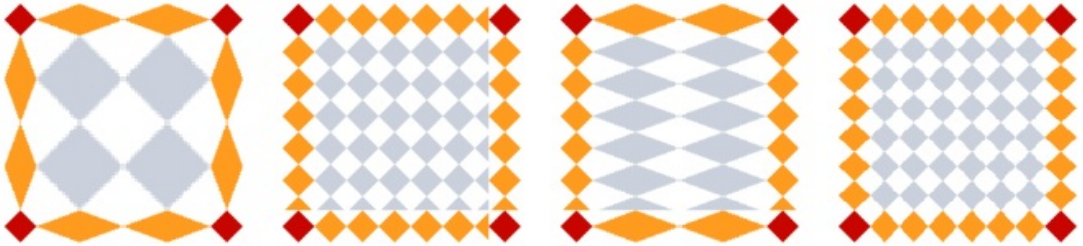
Demo: qml-composing-uis/ex-image-tiling



- Create border using part of an image.
  - Corners (region 1, 3, 7, 9) are not scaled.
  - Horizontal borders (2 and 8) are scaled according to horizontalTileMode.
  - Vertical borders (4 and 6) are scaled according to verticalTileMode.
  - Middle region (5) is scaled according to both modes.
- There are 3 different scale modes:
  - Stretch: Scale the image to fit to the available area.
  - Repeat: Tile the image until there is no more space.
  - Round: Like Repeat, but scales the images down to ensure that the last image is not cropped

Input:

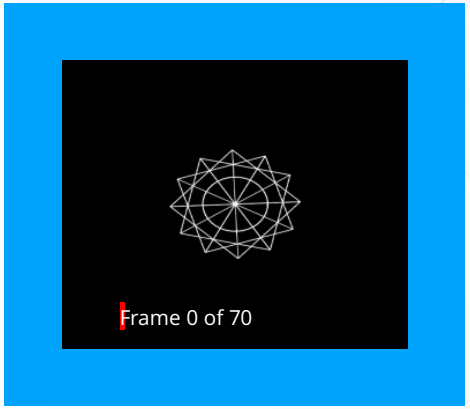
Transformed:



```
1 BorderImage {
2     source: "content/colors.png"
3     border { left: 30; top: 30; right: 30; bottom: 30; }
4     horizontalTileMode: BorderImage.Stretch
5     verticalTileMode: BorderImage.Repeat
6     ...
7 }
```

Demo: qml-composing-uis/ex-image-tiling

```
1  AnimatedImage {
2      id: animation
3
4      x: 50; y: 50
5      width: parent.width-100
6      height: parent.height-100
7
8      source: "../images/image-animated.gif"
9  }
```

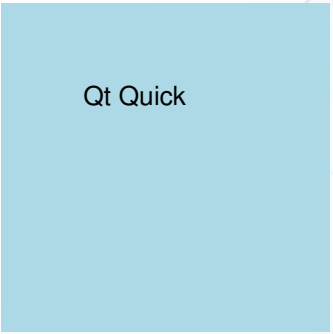


- Inherits from Image
- Additional properties for controlling and monitoring animation playback

Demo: qml-composing-uis/ex-image-animated

- Image Elements
- **Text Elements**
- Item Transformations
- Anchor Layout
- Colors and Gradients

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 400
5      color: "lightblue"
6
7      Text {
8          x: 100; y: 100
9          text: "Qt Quick"
10         font.family: "Helvetica"
11         font.pixelSize: 32
12     }
13 }
```

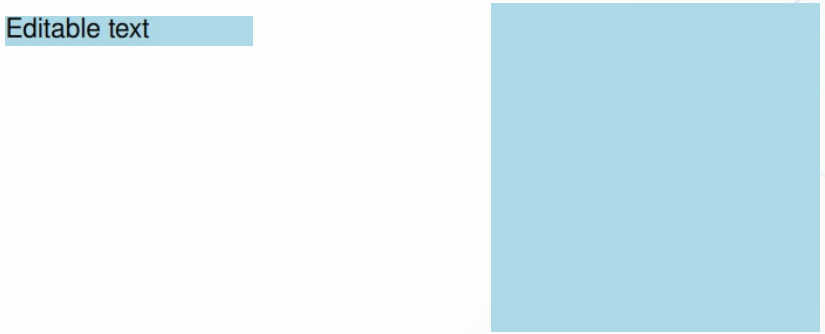


- HTML tags possible in the text:

```
text: "<html><b>Qt Quick</b></html>"
```

Demo: qml-composing-uis/ex-text

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 400
5      color: "lightblue"
6
7      TextInput {
8          x: 50; y: 100; width: 300
9          text: "Editable text"
10         font.family: "Helvetica"; font.pixelSize: 32
11     }
12 }
```



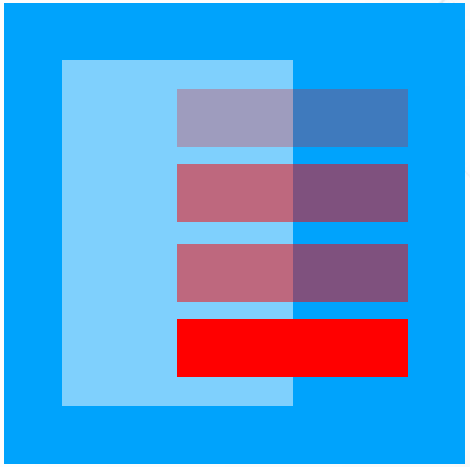
Demo: qml-composing-uis/ex-textinput

- Image Elements
- Text Elements
- **Item Transformations**
- Anchor Layout
- Colors and Gradients

- Can be applied to any Item type
- Like position, the effect is relative to the parent
- Control properties:
  - opacity: values from 0.0 (transparent) to 1.0 (opaque)
  - scale: size multiplication factor
  - rotation: clockwise rotation angle in degrees

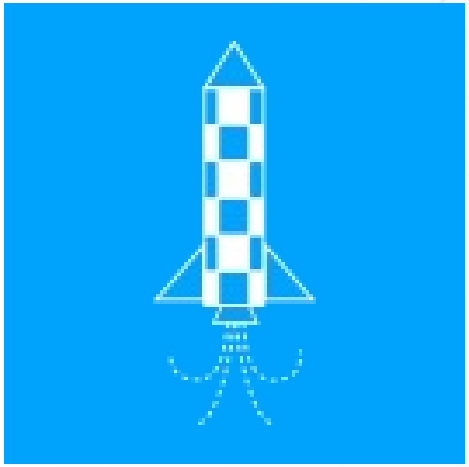
Item Opacity

```
import QtQuick 2.12

Rectangle {
    width: 400; height: 400
    color: "#00a3fc"
    Rectangle {
        x: 50; y: 50; width: 200; height: 300
        color: "white"
        opacity: 0.5

        Rectangle {
            x: 100; y: 25; width: 200; height: 50
            color: "red"
            opacity: 0.5
        }
        Rectangle {
            x: 100; y: 90; width: 200; height: 50
            color: "red"
        }
    }

    Rectangle {
       x: 150; y: 210; width: 200; height: 50
       color: "red"
       opacity: 0.5
    }
    Rectangle {
       x: 150; y: 275; width: 200; height: 50
       color: "red"
    }
}
```



```
1  import QtQuick 2.0
2
3  Rectangle {
4      id: rectangle
5      width: 400; height: 400
6      color: "#00a3fc"
7      Image {
8          id: rocket
9          anchors.centerIn: parent
10         source: "../images/rocket.png"
11         scale: 3.0
12     }
13 }
```

Demo: qml-composing-uis/ex-image-scaling



Item Rotation

```qml
import QtQuick 2.0

Rectangle {
    width: 200; height: 200
    color: "#00a3fc"

    Image {
        x: 50; y: 35
        source: "../images/rocket.png"
        rotation: 45.0
    }
}
```

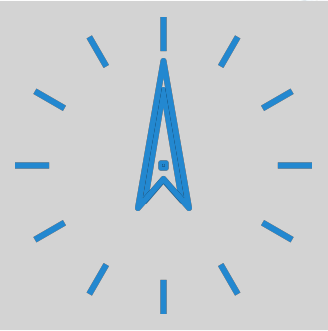
Demo: qml-composing-uis/ex-image-rotation

Item Rotation (cont'd)

```qml
import QtQuick 2.0

Rectangle {
    width: 200; height: 200
    color: "#00a3fc"

    Image {
        x: 50; y: 35
        source: "../images/rocket.png"
        rotation: 45.0
        transformOrigin: Item.Top
    }
}
```

Demo: qml-composing-uis/ex-image-rotation-top

- On the Item Element:
  - transform : list<Transform>
- Transform is one of:
  - Rotation
  - Scale
  - Translate
  - Matrix4x4



```
1  Image {
2      // The 1255 is the anchor point of the arm, measured in the file itself
3      id: largeArm
4      source: "largeArm.png"
5      x: background.width/2 - width/2
6      y: background.height/2 - 1255
7
8      transform: Rotation {
9          origin.x: largeArm.width/2
10         origin.y: 1255
11         angle: 90
12
13     }
14 }
```

Demo: qml-composing-uis/ex-custom-transform

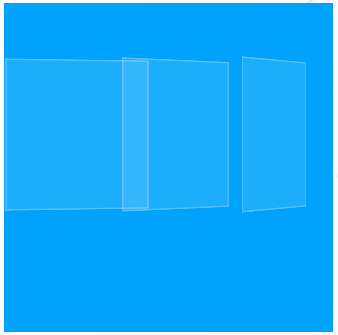
Item X Y Scaling

```qml
import QtQuick 2.4

Rectangle {
    width: 200; height: 200
    color: "#00a3fc"

    Image {
        x: 50; y: 35
        source: "../images/rocket.png"
        mipmap:true
        transform: Scale {
            origin.x: -30
            origin.y: 55
            xScale: 1
            yScale: -1
        }
    }
}
```

Demo: qml-composing-uis/ex-X-Y-scale

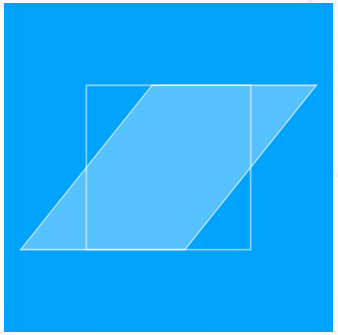
Transform Z Rotation

```
1  import QtQuick 2.4
2
3  Rectangle {
4      width: 200; height: 200
5      color: "#00a3fc"
6      Rectangle {
7          x: 0; y: 35; width:90; height: 90
8          transform: Rotation { origin.x: 45; origin.y: 45
9              axis { x: 0; y: 1; z: 0 } angle: 18 }
10         antialiasing: true
11         color: "#55ffffff"; border.color: "white"}
12     Rectangle {
13         x: 60; y: 35; width:90; height: 90
14         transform: Rotation { origin.x: 45; origin.y: 45
15             axis { x: 0; y: 1; z: 0 } angle: 45 }
16         antialiasing: true
17         color: "#55ffffff";border.color: "white"}
18     Rectangle {
19         x: 120; y: 35; width:90; height: 90
20         transform: Rotation { origin.x: 45; origin.y: 45
21             axis { x: 0; y: 1; z: 0 } angle: 65 }
22         antialiasing: true
23         color: "#55ffffff";border.color: "white"}
24 }
```



Demo: qml-composing-uis/ex-Z-rotation

```
1  import QtQuick 2.4
2
3  Rectangle {
4      width: 200; height: 200
5      color: "#00a3fc"
6
7      Rectangle {
8          x: 50; y: 50; width:100; height: 100
9          color: "#00ffffff"; border.color: "#88ffffff"}
10
11     Rectangle {
12         x: 50; y: 50; width:100; height: 100
13         color: "#55ffffff"; border.color: "#a9ffffff"
14         antialiasing: true
15         transform: Matrix4x4 {
16                matrix: Qt.matrix4x4(1, -.8, 0, 40,
17                                     0, 1, 0, 0,
18                                     0, 0, 1, 0,
19                                     0, 0, 0, 1)}
20     }
21 }
```

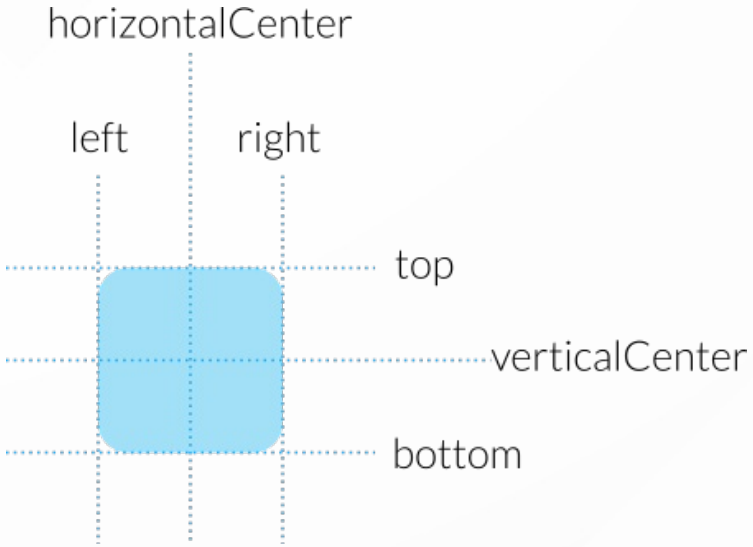


Demo: qml-composing-uis/ex-matrix4x4

- Image Elements
- Text Elements
- Item Transformations
- **Anchor Layout**
- Colors and Gradients

- Mechanism to position Items relative to another
- Anchors are bindings hence dynamically reevaluated
- Two kinds of Anchor bindings
  - Anchor lines (left, top etc...).
  - Anchor helpers (centerIn, fill).

- Line up the edges or central lines of items.
- Anchor lines can only be bound to another compatible anchor line



Qt Docs: Layouts In QML

Anchor Lines (cont'd)

```
1  import QtQuick 2.0
2
3  Rectangle {
4      id: background
5      width: 300; height: 100
6      color: "lightblue"
7
8      Rectangle {
9          color: "green"
10         y: 25
11         height: 50; width: 50
12         anchors.right: background.right
13     }
14 }
```

- Constrains the position of an item relative to another one
- Anchors of other items are referred to directly
  - Use background.right
  - Do not use background.anchors.right

Demo: qml-composing-uis/ex-anchor-layout/anchor-to-anchor.qml

Anchor Lines (cont'd)

```
1  import QtQuick 2.0
2
3  Rectangle {
4      id: background
5      width: 300; height: 100
6      color: "lightblue"
7
8      Rectangle {
9          color: "green"
10         y: 25 //overwritten by the top anchor
11         height: 50; width: 50
12         anchors.right: background.right
13         anchors.top: background.top
14     }
15 }
```

Demo: qml-composing-uis/ex-anchor-layout/anchor-to-anchor2.qml

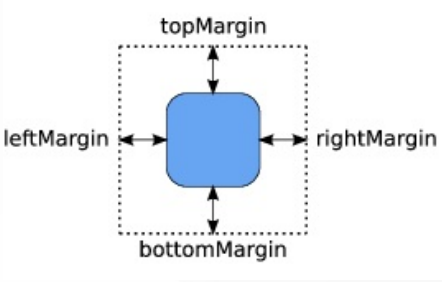
Anchor Lines (cont'd)

```
1  import QtQuick 2.0
2
3  Rectangle {
4      id: background
5      width: 300; height: 100
6      color: "lightblue"
7
8      Rectangle {
9          color: "green"
10         height: 50
11         anchors.top: background.top
12
13         anchors.left: background.left
14         anchors.right: background.right
15     }
16 }
```

- Constrains the position AND the size of an item relative to another one
- Anchor can be used with sizing
  - if width and anchors left / right are both set, anchors override the given width.

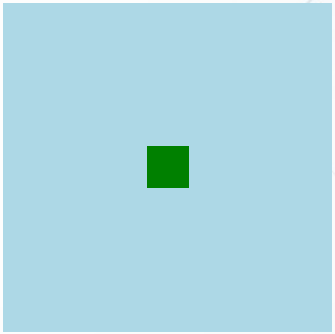
Demo: qml-composing-uis/ex-anchor-layout/anchor-to-anchor-resize.qml

```
1  import QtQuick 2.0
2
3  Rectangle {
4    id: bg
5    width: 400; height: 200
6    color: "lightblue"
7
8    Image { id: book; source: "../images/book.svg"
9            anchors.left: bg.left
10           anchors.leftMargin: bg.width/16
11           anchors.verticalCenter: bg.verticalCenter }
12
13   Text { text: "Writing"; font.pixelSize: 32
14          anchors.left: book.right
15          anchors.leftMargin: 32
16          anchors.baseline: book.verticalCenter }
17 }
```



Demo: qml-composing-uis/ex-anchor-layout/alignment.qml

```qml
import QtQuick 2.0

Rectangle {
    // The parent element
    width: 400; height: 400
    color: "lightblue"

    Rectangle {
        color: "green"
        width: 50; height: 50
        anchors.centerIn: parent
    }
}
```

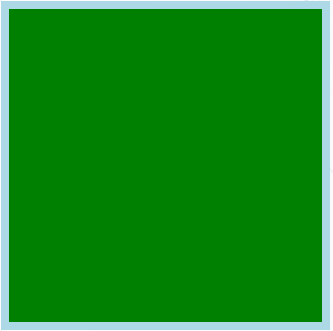


Note:
anchors.centerIn refers to an item id, not an anchor line.

Demo: qml-composing-uis/ex-anchor-layout/anchors-centerin-parent-keyword.qml

Anchor Helpers (cont'd)

```
1  import QtQuick 2.0
2
3  Rectangle {
4      // The parent element
5      width: 400; height: 400
6      color: "lightblue"
7
8      Rectangle {
9          color: "green"
10         anchors.fill: parent
11         anchors.margins: 10
12     }
13 }
```



Demo: qml-composing-uis/ex-anchor-layout/anchors-fill.qml

- Two Anchors helpers
  - centerIn to center an item into another one.
  - fill to make one item as big as another one.
- Anchors helpers are bound to another element id
  - Can refer to a parent id or any named childen of ancestors.
  - Can use the keyword parent to refer to the direct parent of the item.

#NOTES

Anchors not only specify position but also size.

Remember: Either anchor both top and bottom or set a height.

Anchors can only be relative to a parent or siblings.

Careful with *hidden* binding loops.

Demo: qml-composing-uis/ex-binding-loop

- Anchors can only be used with parent and sibling items.
- Anchors work on constraints.
  - Some items need to have well-defined positions and sizes.
  - Items without default sizes should be anchored to fixed or well-defined items.
- Anchors create dependencies on geometries of other items.
  - Create an order in which geometries are calculated.
  - Avoid creating circular dependencies.
    - e.g., parent -> child -> parent
- Margins are only used if the corresponding anchors are used.
  - e.g., leftMargin needs left to be defined.

#NOTES

Identify items with different roles in the user interface.

- Fixed items
  - Make sure these have id properties defined
  - Unless these items can easily be referenced as parent items
- Items that dominate the user interface
  - Make sure these have id properties defined.
- Items that react to size changes of the dominant items
  - Give these anchors that refer to the dominant or fixed items.

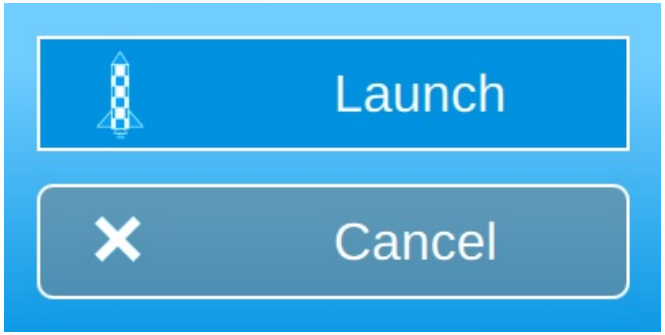
#NOTES

1. When creating an Image, how do you specify the location of the image file?

2. By default, images are rotated about a point inside the image. Where is this point?

3. How do you set the text in a Text element?

Lab - Images, Text and Anchors



- Using the partial solutions as hints, create a user interface similar to the one shown above.
- Use the background image supplied in the common images directory for the background gradient.

Lab: qml-composing-uis/lab-text-images-anchors

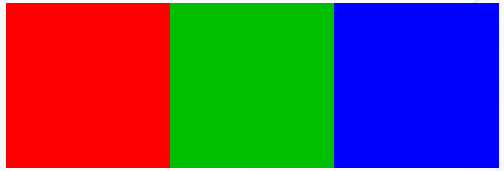
- Image Elements
- Text Elements
- Item Transformations
- Anchor Layout
- **Colors and Gradients**

The colors of elements can be specified in many ways.

- As a named color in a string (using SVG names):
  - "red", "green", "blue", ...
- With color components in a string:
  - alpha, red, green and blue: #<aa><rr><gg><bb> (alpha optional)
  - "#ff000000", "#008000", "#0000ff", ...
- Using a built-in function (red, green, blue, alpha):
  - Qt.rgba(0,0.5,0,1)

Qt Docs: QML Basic Type: color

```
1  import QtQuick 2.0
2
3  Item {
4      width: 300; height: 100
5
6      Rectangle {
7          x: 0; y: 0; width: 100; height: 100; color: "#ff0000"
8      }
9      Rectangle {
10         x: 100; y: 0; width: 100; height: 100
11         color: Qt.rgba(0,0.75,0,1)
12     }
13     Rectangle {
14         x: 200; y: 0; width: 100; height: 100; color: "blue"
15     }
16 }
```



Demo: qml-composing-uis/ex-colors

Define a gradient using the gradient property.

- With a Gradient element as the value
- Containing two or more GradientStop elements, each with
  - A position: a number between 0 (start point) and 1 (end point).
  - A color.
- The start and end points
  - Are on the top and bottom edges of the item.
  - cannot be repositioned.
- Gradients override color definitions.

#NOTES

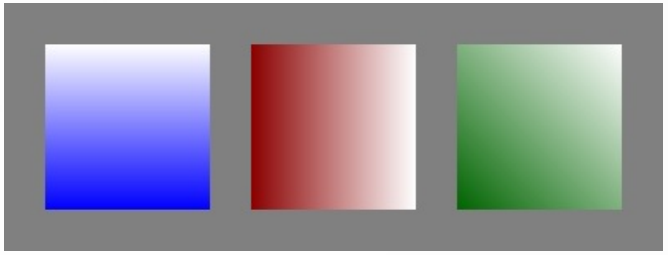
```
1  import QtQuick 2.0
2  Item{width: 400*2; height: 400*2
3  Rectangle{
4      width: 400; height: 400
5      anchors.centerIn: parent
6      color: "red"
7  Rectangle {
8      width: Math.sqrt(400*400+400*400); height: width
9      anchors.centerIn: parent
10     opacity: 0.5
11     gradient: Gradient {
12         GradientStop {
13             position: 0.0; color: "green"
14         }
15         GradientStop {
16             position: 1.0; color: "blue"
17         }
18     }
19     rotation: -45
20 }
21 }
22 }
```

- Note the definition of an element as a property value.

Demo: qml-composing-uis/ex-gradients **Qt Docs: QML Gradient Element Reference**

1. How else can you write the "blue" color?

2. How would you create these items using the gradient property?



3. Describe another way to create these gradients.

Solution: qml-intro/sol-colors-and-gradients/sol-colors-and-gradients.qml

- Composing User Interfaces
- **Animations**
- User Interaction
- Components
- Presenting Data

Can apply animations to user interfaces:

- Understanding of basic concepts
  - Number and property animations
  - Easing curves
- Ability to queue and group animations
  - Sequential and parallel animations
  - Pausing animations
- Knowledge of specialized animations
  - Color and rotation animations

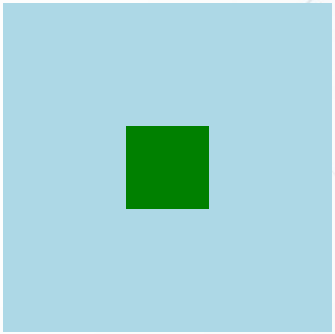
Animations can be applied to any element.

- Animations update properties to cause a visual change.
- All animations are property animations.
- Specialized animation types:
  - NumberAnimation, for changes to numeric properties
  - ColorAnimation, for changes to color properties
  - RotationAnimation, for changes to orientation of items
  - Vector3dAnimation, for motion in 3D space
- Easing curves are used to create variable speed animations.
- Animations are used to create visual effects.

Qt Docs: QML Animation

#NOTES

```qml
import QtQuick 2.0

Rectangle {
    width: 400; height: 400
    color: "lightblue"

    Rectangle {
        y: 150; width: 100; height: 100
        color: "green"
        property int xi: 0
        x: xi
        NumberAnimation on xi {
            from: 0; to: 150
            duration: 10000
            running: true
            easing.bezierCurve: [0.43,0.0025,0.19,1.37,0.33,1.21,0.6,0.775,0.65,1,1,1]
        }
    }
}
```



Demo: qml-animations/ex-number-animation

Number animations change the values of numeric properties.

```
1 NumberAnimation on x {
2     from: 0; to: 150
3     duration: 1000
4 }
```

- Applied directly to properties with the on keyword
- The x property is changed by the NumberAnimation.
  - Starts at 0
  - Ends at 150
  - Takes 1000 milliseconds
- Can also be defined separately

#NOTES

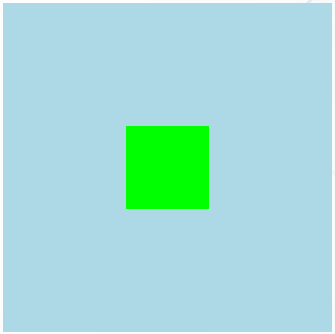
Number Animations as Separate Element

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 400; color: "lightblue"
5
6      Rectangle {
7          id: rect
8          x: 300; y: 300
9          width: 100; height: 100
10         MouseArea {
11             anchors.fill: parent
12             onClicked: ani.running = (ani.running? false: true)
13         }
14
15     }
16
17     NumberAnimation {
18         id: ani
19         target: rect
20         properties: "x,y"
21         from: 0
22         to: 150; duration: 10000
23
24     }
25 }
```


Demo: qml-animations/ex-number-animation2

Color Animation

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 400;
5      color: "lightblue"
6
7      Rectangle {
8          id: rectangle1
9          x: 150; y: 150
10         width: 100
11         height: 100
12         color: Qt.rgba(0,1,0,1)
13     }
14
15     ColorAnimation {
16         target: rectangle1
17         property: "color"
18         from: Qt.rgba(0,1,0,1)
19         to: Qt.rgba(1,1,1,1)
20         duration: 1000
21         running: true
22         loops: -1
23     }
24 }
```



Demo: qml-animations/ex-color-animation

```
1  import QtQuick 2.0
2
3  Item {
4      width: 100; height: 100
5
6      Image {
7          id: ball
8          source: "../images/ball.png"
9          anchors.centerIn: parent
10         smooth: true
11
12         RotationAnimation on rotation {
13             from: 45; to: Math.random()*360
14             direction: RotationAnimation.Counterclockwise
15             duration: 1000
16         }
17     }
18 }
```

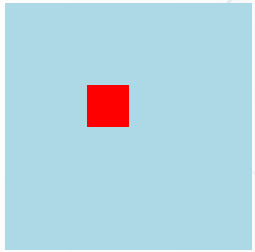
- Counter-clockwise from 45 ° to 315 °
  - Shortest angle of rotation is via 0 °

Demo: qml-animations/ex-rotation-animation

- RotationAnimation describes rotation of items.
- Easier to use than NumberAnimation for the same purpose
- Applied to the rotation property of an element
- Value of direction property controls rotation:
  - RotationAnimation.Clockwise
  - RotationAnimation.Counterclockwise
  - RotationAnimation.Shortest - the direction of least angle between from and to values

#NOTES

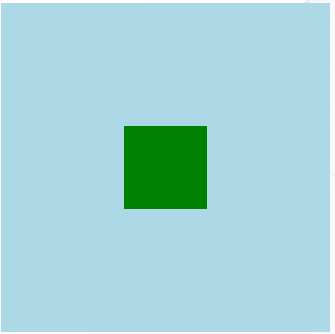
- Behavior allows you to set up an animation whenever a property changes.
  - *not available in .ui.qml files*



```
1 Rectangle {
2     id: rect
3     x: 100; y: 100
4     width: 50; height: 50
5     color: "red"
6
7     Behavior on x { SpringAnimation { spring: 1; damping: 0.2 } }
8     Behavior on y { SpringAnimation { spring: 2; damping: 0.2 } }
9 }
```

Demo: qml-animations/ex-spring-animation

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 400
5      color: "lightblue"
6
7      Rectangle {
8          y: 150; width: 100; height: 100
9          color: "green"
10
11         NumberAnimation on x {
12             from: 0; to: 150; duration: 1000
13             easing.type: Easing.OutExpo
14         }
15     }
16 }
```



Demo: qml-animations/ex-easing-curve

Demo: qml-animations/ex-easing-curve-alltypes

Qt Demo: examples/widgets/animation/easing

Qt Demo: examples/quick/animation

Apply an easing curve to an animation:

```
1 NumberAnimation on x {
2     from: 0; to: 150; duration: 1000
3     easing.type: "OutExpo"
4 }
```

- Sets the easing.type property
- Relates the elapsed time
  - To a value interpolated between the from and to values
  - Using a function for the easing curve
  - In this case, the "OutExpo" curve
- In QtCreator, put the cursor on the easing type and press Ctrl+Alt+Space to show a nice graph of the easing curve.

#NOTES

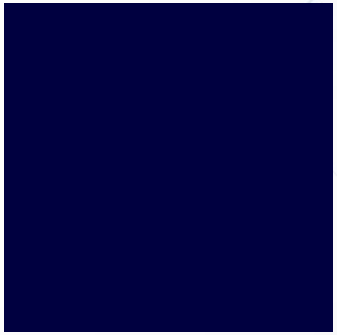
Animations can be performed sequentially and in parallel.

- SequentialAnimation defines a sequence
  - With each child animation run in sequence
- For example:
  - A rescaling animation, followed by
  - An opacity-changing animation
- ParallelAnimation defines a parallel group
  - With all child animations run at the same time
- For example:
  - Simultaneous rescaling and opacity-changing animations

Sequential and parallel animations can be nested.

#NOTES

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 200; height: 200; color: "#000040"
5
6      Image {
7          id: rocket
8          anchors.centerIn: parent
9          source: "../images/rocket.png"
10     }
11
12     SequentialAnimation {
13         NumberAnimation {
14             target: rocket; properties: "scale"
15             from: 1.0; to: 0.5; duration: 1000
16         }
17         NumberAnimation {
18             target: rocket; properties: "opacity"
19             from: 1.0; to: 0.0; duration: 1000
20         }
21         running: true
22     }
23 }
```



Demo: qml-animations/ex-sequential-animation

- Child elements define a two-stage animation:
  - First, the rocket is scaled down
  - Then, it fades out
- SequentialAnimation does not itself have a target.
  - It only groups other animations.

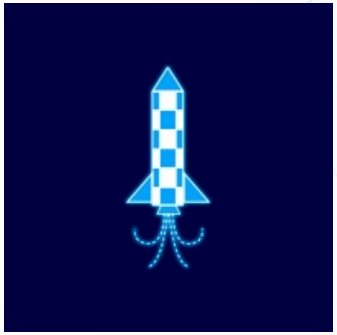
#NOTES

Pausing Between Animations

```
SequentialAnimation {
    NumberAnimation {
        target: rocket; properties: "scale"
        from: 0.0; to: 1.0; duration: 1000
    }
    PauseAnimation {
        duration: 1000
    }
    NumberAnimation {
        target: rocket; properties: "scale"
        from: 1.0; to: 0.0; duration: 1000
    }
    running: true
}
```

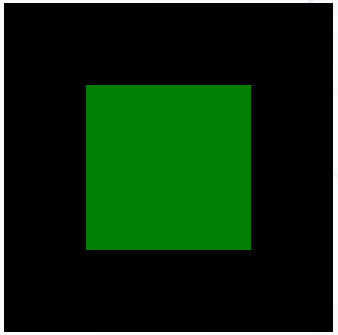
Parallel Animations

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 200; height: 200; color: "#000040"
5
6      Image {
7          id: rocket
8          anchors.centerIn: parent
9          source: "../images/rocket.png"
10     }
11
12     ParallelAnimation {
13         NumberAnimation {
14             target: rocket; properties: "scale"
15             from: 0.5; to: 1.0; duration: 1000
16         }
17         NumberAnimation {
18             target: rocket; properties: "opacity"
19             from: 0.0; to: 1.0; duration: 1000
20         }
21         running: true
22     }
23 }
```



Demo: qml-animations/ex-parallel-animation

```
1  Flipable {
2      id: flipable
3      anchors.centerIn: parent
4      property bool flipped: false
5
6      front: Rectangle {  ~~~ d
7      back: Rectangle {  ~~~ d
8
9      transform: Rotation {
10         axis.x: 1; axis.y: 0; axis.z: 0
11         angle: flipable.flipped ? 180 : 0
12
13         Behavior on angle {
14             NumberAnimation { duration: 500 }
15         }
16     }
17 }
```



- Based on transform, Flipable determines which side to display.

Demo: qml-animations/ex-flipable-animation

To understand the Flipable element, it might be useful to consider its responsibilities:

Provides:

- Showing/hiding the side that is visible, by observing the transformation property.
- **front** and **back** properties to hold the two sides.

Doesn't provide:

- Mouse interaction to execute the flip
- Setting up the transformation to indicate what is shown
- Animation for the flip

#NOTES

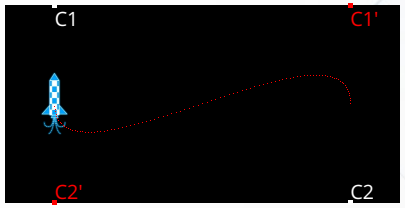
flipped vs. side:
In the code we use a property **flipped** to indicate which side is currently supposed to be shown. At the same time, the flipable element has a property **side** which tells us what is currently visible.

The **side** property is read-only and only tells what, at any given time (e.g during the animation), is currently visible. The **flipped** property, on the other hand, tells us the destination (and is therefore changed at the start of the animation).

So we cannot use **side** instead of **flipped**

#NOTES

```
1  PathAnimation {
2      target: rocket
3      orientation: PathAnimation.TopFirst
4      anchorPoint: Qt.point(rocket.width/2,
5                            rocket.height/2)
6
7      path: Path {
8          startX: 100; startY: window.height/2
9
10         PathCubic {
11             id: part1
12             x: window.width - 100
13             y: window.height/2
14             control1X: 100; control1Y: 0
15             control2X: x; control2Y: window.height
16         }
17
18         PathCubic {  ~~~ d
19     }
20 }
```

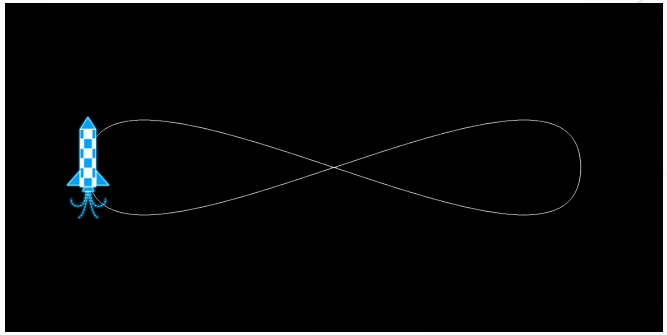


- Path specified using PathLine, PathQuad, PathCubic, PathArc, PathCurve and PathSvg

Demo: qml-animations/ex-path-animation **Qt Docs: PathAnimation**

Path Interpolator

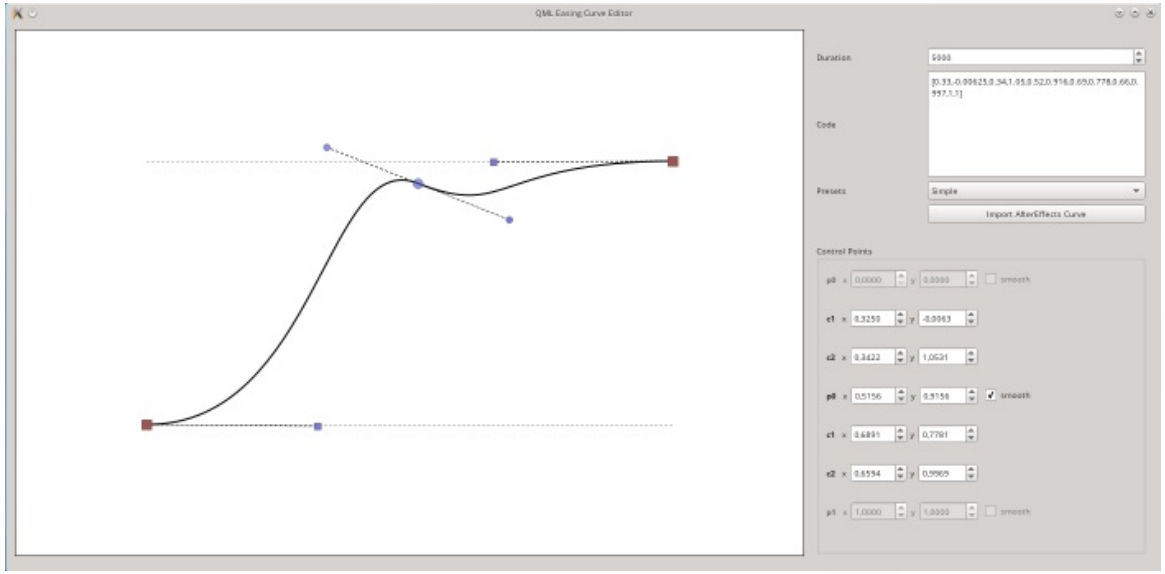
- API similar to PathAnimation
- Requires animating the progress property
- Gives access to x, y and angle of the current point of the path



Demo: qml-animations/ex-pathinterpolator-animation

Qt Docs: PathInterpolator

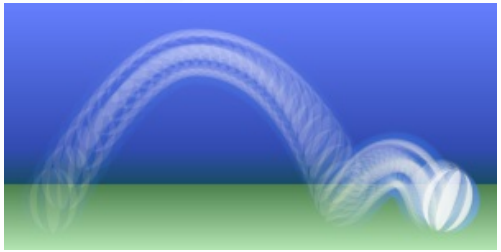
- Custom curve editor available in the bin folder on your Qt Install.



Demo: qml-animations/ex-easing-editor

Overview of Animations

- Type specific Animation
  - ColorAnimation
  - RotationAnimation
  - Vector3dAnimation
  - PathAnimation
  - NumberAnimation
- Structural changes
  - AnchorAnimation
  - ParentAnimation
- Immediate changes
  - PropertyAction
  - ScriptAction
- Animations with build-in easing curves
  - SpringAnimation
  - SmoothedAnimation
- Combining Animations
  - SequentialAnimation
  - ParallelAnimation
  - PauseAnimation
- Other
  - PropertyAnimation
  - PathInterpolator
- Animators (running directly on the scene graph)
  - OpacityAnimator
  - RotationAnimator
  - ScaleAnimator
  - UniformAnimator
  - XAnimator
  - YAnimator



Starting from the first partial solution:

- Make the ball start from the ground and return to the ground.
- Make the ball travel from left to right.
- Add rotation, so the ball completes just over one rotation.
- Reorganize the animations using sequential and parallel animations.
- Make the animation start when the ball is clicked.
- Add decoration (ground and sky).

Lab: qml-animations/lab-animations

- Composing User Interfaces
- Animations
- **User Interaction**
  - Mouse/Touch Input
  - Gestures Support
  - Keyboard Input
- Components
- Presenting Data

Objectives

- Knowledge of ways to receive user input
  - Mouse/touch input
  - Gestures Support
  - Keyboard input
- Awareness of different input processing mechanisms
  - Signal handlers
  - Property bindings

Users interact with Qt Quick user interfaces:

- Mouse movement, clicks and dragging
- Single/Multi Touchpoint gestures
- Keyboard input

- **Mouse/Touch Input**
- Gestures Support
- Keyboard Input

```
1  Rectangle {
2      width: 400; height: 300;
3      color: "lightblue"
4
5      Text {
6          text: "Press me"
7
8          MouseArea {
9              anchors.fill: parent
10             onPressed: parent.color = "green"
11             onReleased: parent.color = "black"
12         }
13     }
14
15     Text {
16         text: "Click me"
17
18         MouseArea {
19             anchors.fill: parent
20             onClicked: parent.font.bold = !parent.font.bold
21         }
22     }
23 }
```

Demo: qml-user-interaction/ex-mouse-pressed-signal

Mouse areas define parts of the screen where cursor input occurs.

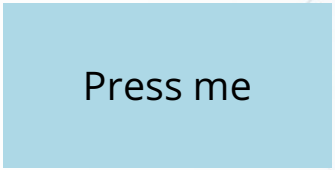
- Placed and resized like ordinary items
  - Using anchors if necessary
- Two ways to monitor mouse input:
  - Handle signals
  - Dynamic property bindings
- Define responses to signals with onPressed and onReleased.
  - By default, only left clicks are handled.
  - Set the acceptedButtons property to change this.

Qt Docs: QML MouseArea Element Reference

#NOTES

Mouse Press and Property Bindings

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 200;
5      color: "lightblue"
6
7      Text {
8          anchors.centerIn: parent
9          text: "Press me"; font.pixelSize: 48
10         color: mouseArea.pressed ? "green" : "black"
11
12         MouseArea {
13             id: mouseArea
14             anchors.fill: parent
15         }
16     }
17 }
```



Demo: qml-user-interaction/ex-mouse-press

- A mouse area only responds to its acceptedButtons.
  - The handlers are not called for other buttons, but
  - any click involving an allowed button is reported.
- With hoverEnabled, set to false.
  - containsMouse can be true, if the mouse area is clicked.

- Mouse events get delivered to item under cursor
- Ignored mouse events get propagated to the item visually below
  - Items ignore mouse events by default
  - enabled MouseAreas accept mouse events by default
- All mouse events until *released* event get delivered to item that accepted *pressed* event
- propagateComposedEvents also propagates ignored composed events (clicked, pressAndHold, ...)

```
1 MouseArea {
2     anchors.fill: parent
3     propagateComposedEvents: true // allow to propagate composed events
4     onPressed: {
5         reactToPress();
6         mouse.accepted = false; // propagate to item below
7     }
8 }
```

Demo: qml-user-interaction/ex-mouse-propagation

Signal Handlers Vs. Property Bindings

- Form two teams.
- Team A: Make a case for using property bindings.
- Team B: Make a case for using signal handlers.

When you assign to a property, its binding will be removed!

Demo: qml-user-interaction/ex-broken-bindings

- Mouse/Touch Input
- **Gestures Support**
- Keyboard Input

Qt Quick user interfaces can handle touch input and gestures.

- Flick
- Swipe
- Pinch
- Tap and hold
- Generic multitouch

```qml
import QtQuick 2.0

Flickable {
    id: flick
    width: 400; height: 400
    contentWidth: 1000
    contentHeight: 1000

    Image {
        width: flick.contentWidth
        height: flick.contentHeight
        source: "../images/rocket.svg"
    }
}
```

Demo: qml-user-interaction/ex-flickable

- The Flickable element provides the behavior.
- The size of the canvas you can pan is controlled with contentWidth and contentHeight.
- Be careful, as parent for elements inside of a Flickable points to the viewport item, not the Flickable itself.
- Also provides methods: flick(), resizeContent() and returnToBounds()

#NOTES

```qml
import QtQuick 2.0

Flickable {
    id: flick
    width: 400; height: 400
    contentWidth: 2000; contentHeight: 2000

    PinchArea {
        anchors.fill: parent
        pinch.target: img
        pinch.maximumScale: 2.0
        pinch.minimumScale: 0.1
        pinch.dragAxis: Pinch.XAndYAxis
    }

    Image {
        id: img
        width: flick.contentWidth
        height: flick.contentHeight
        source: "../images/rocket.svg"
        sourceSize.width: 900
        sourceSize.height: 900
    }
}
```

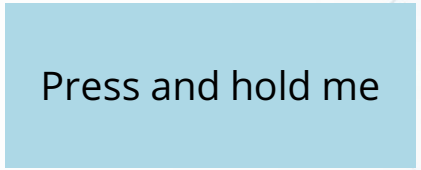
Demo: qml-user-interaction/ex-pinch-target

Demo: qml-user-interaction/ex-pinch

- PinchArea is an invisible item like MouseArea.
- It provides signal handlers to track the gesture: onPinchStarted, onPinchUpdated and onPinchFinished
- All of these signal handlers get a pinch parameter that shows you the current scale, center and angle of the gesture.

#NOTES

```qml
import QtQuick 2.0

Rectangle {
    width: 500; height: 200; color: "lightblue"

    Text {
        anchors.centerIn: parent
        text: "Press and hold me"; font.pixelSize: 48

        property bool isActive: false
        color: isActive ? "green" : "black"

        MouseArea {
            anchors.fill: parent
            onPressAndHold: parent.isActive = !parent.isActive
        }
    }
}
```



Demo: qml-user-interaction/ex-mouse-press-hold-signal

- This feature is directly provided by MouseArea.
- It simply uses the onPressAndHold signal handler.
- This of course means it also works using a mouse.

#NOTES

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 1000; height: 1000; color: "lightblue"
5
6      MultiPointTouchArea {
7          anchors.fill: parent
8          minimumTouchPoints: 1
9          maximumTouchPoints: 3
10
11         touchPoints: [
12             TouchPoint { id: touch1 },
13             TouchPoint { id: touch2 },
14             TouchPoint { id: touch3 }
15         ]
16     }
17
18     Rectangle {
19         x: touch1.x - width/2; y: touch1.y - height/2
20         width: 200; height: 200
21         visible: touch1.pressed
22         color: "cyan"
23     }
24     Rectangle { ~~~ d
25     Rectangle { ~~~ d
26 }
```

Demo: qml-user-interaction/ex-multitouch

- MultiPointTouchArea is used for generic, rich user interaction, like finger painting.
- The minimumTouchPoints and maximumTouchPoints properties control when the area is active and actually tracking the touch points.
- The touchPoints property allows declaring a list of TouchPoint which can be used in property bindings.
- The TouchPoint properties, such as area, pressure and velocity, allow you to track its state.

#NOTES

- Mouse/Touch Input
- Gestures Support
- **Keyboard Input**

Basic keyboard input is handled in two different use cases:

- Accepting text input
  - TextInput and TextEdit
- Navigation between elements
  - Changing the focused element
  - Directional (arrow keys), tab and backtab
- On slide 233, we will see how to handle raw keyboard input.

- UIs with just one TextInput
  - Focus assigned automatically
- More than one TextInput
  - Need to change focus by clicking
- Set the focus property to assign focus.

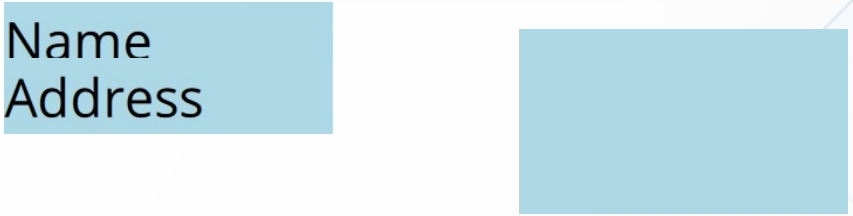
#NOTES

Using TextInputs

```
1  import QtQuick 2.7
2
3  Rectangle {
4      width: 200; height: 112; color: "lightblue"
5
6      TextInput {
7          anchors.left: parent.left; y: 16
8          anchors.right: parent.right
9          text: "Field 1"; font.pixelSize: 32
10         color: activeFocus ? "black" : "gray"
11         focus: true
12         activeFocusOnTab: true
13     }
14     TextInput {
15         anchors.left: parent.left; y: 64
16         anchors.right: parent.right
17         text: "Field 2"; font.pixelSize: 32
18         color: activeFocus ? "black" : "gray"
19         activeFocusOnTab: true
20     }
21 }
```

Demo: qml-user-interaction/ex-text-input

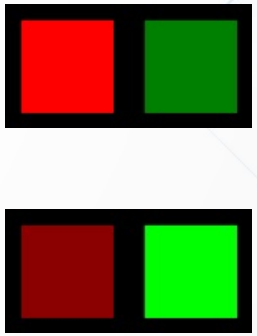
```
1  TextInput {
2      id: nameField
3      anchors.left: parent.left; y: 16
4      anchors.right: parent.right
5      text: "Name"; font.pixelSize: 32
6      focus: true
7      KeyNavigation.tab: addressField
8  }
9
10 TextInput {
11     id: addressField
12     anchors.left: parent.left; y: 64
13     anchors.right: parent.right
14     text: "Address"; font.pixelSize: 32
15     KeyNavigation.tab:nameField
16 }
```



- The nameField item defines KeyNavigation.tab.
  - Pressing **Tab** moves focus to the addressField item.
- The addressField item defines KeyNavigation.backtab.
  - Pressing **Shift+Tab** moves focus to the nameField item.

Demo: qml-user-interaction/ex-tab-navigation

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 600; height: 200; color: "black"
5
6      Rectangle {
7          id: leftRect
8          anchors {
9              verticalCenter: parent.verticalCenter
10             left: parent.left
11             leftMargin: 25
12         }
13         width: 150; height: 150
14         color: activeFocus ? "red" : "darkred"
15         KeyNavigation.right: rightRect
16         focus: true
17     }
18     Rectangle { ~~~ d
19     Rectangle { ~~~ d
20 }
```



Demo: qml-user-interaction/ex-key-navigation

Mouse and cursor input handling:

- MouseArea receives clicks and other events.
- Use anchors to fill objects and make them clickable.
- Respond to user input:
  - Give the area a name and refer to its properties, or
  - Use handlers in the area and change other named items.

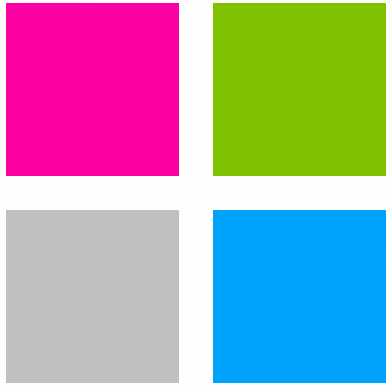
Key handling:

- TextInput and TextEdit provide text entry features.
- Set the focus property to start receiving key input.
- KeyNavigation defines relationships between items.
  - Enables focus to be moved
  - Using cursor keys, tab and backtab
  - Works with non-text-input items

1. Which element is used to receive mouse clicks?

2. Name two ways TextInput can obtain the input focus.

3. How do you define keyboard navigation between items?

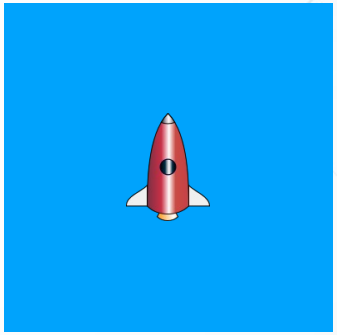


- Using the partial solution as a starting point, create a user interface similar to the one shown above with these features:
  - Items that change color when they have the focus
  - Clicking an item gives it the focus.
  - The current focus can be moved using the cursor keys.

Lab: qml-user-interaction/lab-menu-screen

- Raw key input can be handled by items
  - With predefined handlers for commonly used keys.
  - Full key event information is also available.
- The same focus mechanism is used as for ordinary text input.
  - Enabled by setting the focus property
- Key handling is not an inherited property of items.
  - Enabled using the Keys attached property
- Key events can be forwarded to other objects.
  - Enabled using the Keys.forwardTo attached property
  - Accept a list of objects

```qml
import QtQuick 2.0

Rectangle {
    width: 400; height: 400; color: "#00a3fc"
    focus: true

    Image {
        id: rocket
        anchors.centerIn: parent
        source: "../images/rocket.svg"
    }

    Keys.onLeftPressed:
        rocket.rotation = (rocket.rotation - 10) % 360
    Keys.onRightPressed:
        rocket.rotation = (rocket.rotation + 10) % 360
}
```



Demo: qml-user-interaction/ex-key-press

- Can use predefined handlers for arrow keys:

```
Keys.onLeftPressed:
    rocket.rotation = (rocket.rotation - 10) % 360
Keys.onRightPressed:
    rocket.rotation = (rocket.rotation + 10) % 360
```

- Or inspect events from all key presses:

```
Keys.onPressed: {
    if (event.key === Qt.Key_Left)
        rocket.rotation = (rocket.rotation - 10) % 360;
    else if (event.key === Qt.Key_Right)
        rocket.rotation = (rocket.rotation + 10) % 360;
}
```

Demo: qml-user-interaction/ex-key-press-event

- Keyboard events are propagated to the parent if an item does not accept the event.
- Specialized key handlers like Keys.onLeftPressed accept the event by default.
- Generalized key handlers like Keys.onPressed ignore the event by default.
  - When handling a key, in general the event should be manually accepted.
- Default can be overridden by setting event.accepted to true or false.

Focus Handling: Who receives key events?

- Only one item can have focus set to true, globally.
  - This applies even across multiple QML files!
  - QtQuick will prevent having multiple items that have focus set to true.
- Clicking on a TextInput will set that item's focus to true.

- Composing User Interfaces
- Animations
- User Interaction
- **Components**
- Presenting Data

Two ways to create reusable user interface components:

- Components
  - Defined using the Component item
  - As many as needed, with a file-unique name
  - Used as templates for items
  - Used with models and view
  - Used with generated content
  - Can be instantiated dynamically
- Custom items
  - Defined in separate files
  - One main element per file
  - Used in the same way as standard items
  - Can have an associated version number

Defining a re-usable element inside the current QML file

```
1 Component {
2     id : myTextComponent
3     Text { .... }
4 }
```

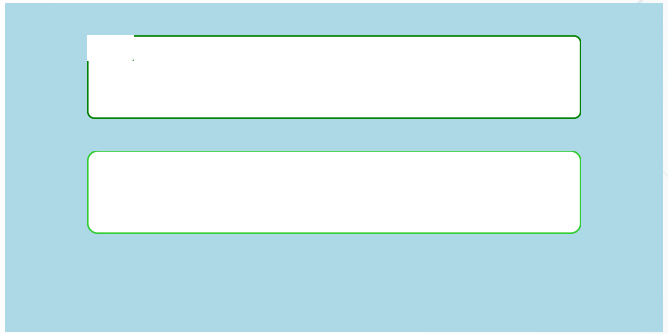
Is not directly instantiated, but can be used with:

- Loader (see ERROR)
- Repeater (see slide 262)
- Views (see slide 275)
- Scripting (see ERROR)

These will create instances of the components child if and when needed.

Defining a Custom Item

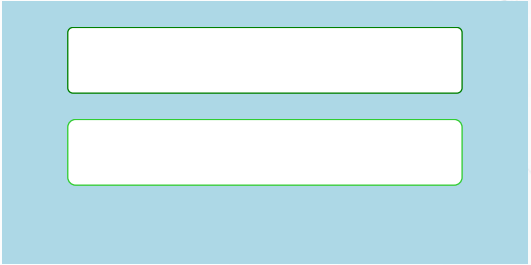
```
1  import QtQuick 2.0
2
3  Rectangle {
4      border.color: "green"
5      color: "white"
6      radius: 4; smooth: true
7      clip: true
8
9      TextInput {
10         anchors.fill: parent
11         anchors.margins: 2
12         text: "Enter text..."
13         color: focus ? "black" : "gray"
14         font.pixelSize: parent.height - 4
15     }
16 }
```



- Stored in file *LineEdit.qml*
- File must start with a capital letter

Using a Custom Item

```
1  import QtQuick 2.12
2
3  Rectangle {
4      width: 400; height: 200
5      color: "lightblue"
6
7
8      NewNewLineEdit {
9          id: newNewLineEdit
10         anchors.horizontalCenter: parent.horizontalCenter
11         anchors.top: lineEdid.bottom
12         anchors.topMargin: 20
13         border.color: "limegreen"
14     }
15
16     LineEdit {
17         id: lineEdid
18         anchors.horizontalCenter: parent.horizontalCenter
19         y: 20
20         width: 300; height: 50
21     }
22 }
```



Demo: qml-modules-components/ex-lineedit

Syntax: **[readonly] property <type> <name> [:<value>]**

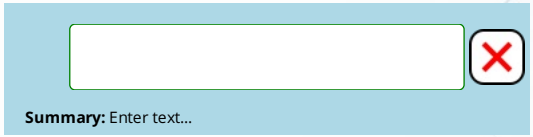
Examples:

```
1 property string product: "Qt Quick"
2 property int count: 123
3 property real slope: 123.456
4 property bool condition: true
5 property url address: "http://qt.io/"
6 readonly property int area: width * height
```

Qt Docs: QML Basic Types

Custom Property Example

```
1  import QtQuick 2.0
2
3  Rectangle {
4      property string text: textInput.text
5
6      border.color: "green"
7      color: "white"
8      radius: 4; smooth: true
9      clip: true
10
11     TextInput {
12         id: textInput
13         anchors.fill: parent
14         anchors.margins: 2
15         text: "Enter text..."
16         color: focus ? "black" : "gray"
17         font.pixelSize: parent.height - 4
18     }
19 }
```

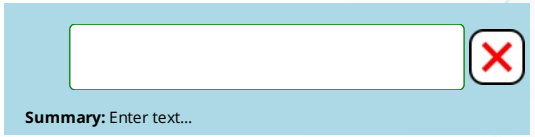


Demo: qml-modules-components/ex-custom-property

- Custom text property *binds to* textInput.text
- Setting the custom property
  - Changes the binding
  - No longer refers to textInput.text
  - See next slide to find out how to solve this.

#NOTES

```qml
import QtQuick 2.0

Rectangle {
    property alias text: textInput.text

    border.color: "green"
    color: "white"
    radius: 4; smooth: true
    clip: true

    TextInput {
        id: textInput
        anchors.fill: parent
        anchors.margins: 2
        text: "Enter text..."
        color: focus ? "black" : "gray"
        font.pixelSize: parent.height - 4
    }
}
```



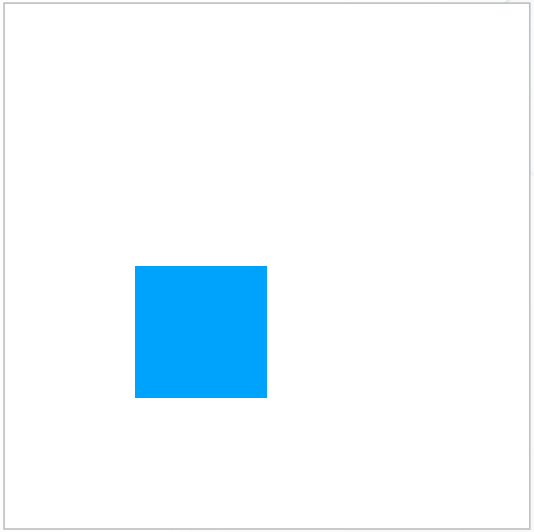
Demo: qml-modules-components/ex-alias-property

- Custom text property *aliases* textInput.text
- Setting the custom property
  - Changes the TextInput's text
- Custom property acts like a proxy

#NOTES

- Properties can be used as local variables.

```
1  Rectangle {
2      id: root
3      width: cellCount * itemSize
4      height: cellCount * itemSize
5      border.color: "silver"
6
7      property int cellCount: 4
8      property int itemSize: 80
9
10     Rectangle {
11         property int __row: 2
12         property int __col: 1
13
14         color: "#00a3fc"
15         x: __col * root.itemSize
16         y: __row * root.itemSize
17         width: root.itemSize
18         height: root.itemSize
19     }
20 }
```



Demo: qml-modules-components/ex-local-variables

Signal syntax: **signal <name>[(<type> <name>, ...)]**

Handler syntax: **on<Name>: <expression>**

Examples of signals and handlers:

```
signal clicked
```

- handled by onClicked

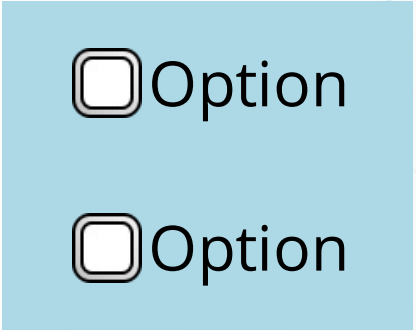
```
signal checked(bool checkValue)
```

- handled by onChecked
- argument passed as checkValue

Demo: qml-modules-components/ex-custom-signal

Defining and Emitting a Custom Signal

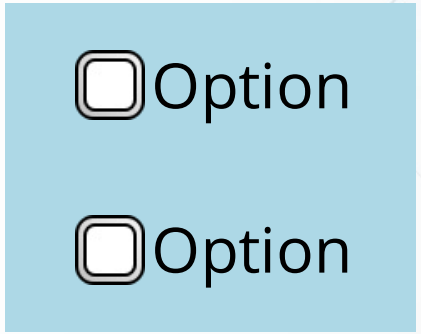
```
1  Item {
2      // define signal
3      signal checked(bool checkValue)
4
5      MouseArea {
6          id: toggleArea
7          anchors.fill: parent
8
9          // define internal state property
10         property bool __checked: false
11
12         onClicked: {
13             __checked = !__checked;
14
15             parent.checked(__checked);  // emit signal with new value
16         }
17     }
18 }
```



- CheckBox item has a checked signal
- Communicates a boolean value called checkValue
- MouseArea's onClicked handler emits the signal by calling it like a function.

#NOTES

```qml
Rectangle {
    CheckBox {
        anchors.centerIn: parent
        onChecked: checkValue ? parent.color = "red"
                              : parent.color = "lightblue"
    }
}

// Note: a signal is not always the best way to go.
// in QML, always prefer a custom property over a signal.
Rectangle {
    // We can now bind to the state
    color: checkBoxBetter.checked ? "red" : "lightblue"

    CheckBoxBetter {
        id: checkBoxBetter
        anchors.centerIn: parent

        // We can now initialize the property explicitly
        checked: false

        // We still have access to a signal handler on the checked property
        onCheckedChanged: console.log("checked=" + checked)
    }
}
```



#NOTES

Receiving a Custom Signal (cont'd)

- checked signal is handled where the item is used
  - By the onChecked handler
  - on* handlers are automatically created for signals
  - Value supplied using name defined in the signal (checkValue)

Demo: qml-modules-components/ex-custom-signal

#NOTES

Implicitly Sized Components

- Some components have an implicit size
  - E.g. Image, Text, ...
  - implicitWidth and implicitHeight determined by content
  - Properties set by component author
  - Acts as a default size hint
- Other components have no implicit size
  - e.g. Rectangle, Item, MouseArea, ...
  - default size is 0
- Implicit size can be overridden with an explicit size
  - By setting width, height or anchors
  - Behavior depends on component
- width, height take the values of implicitWidth, implicitHeight, unless explicitly overridden

Setting the Size of a Component

- width and height should be used top down
  - *An element tells its children the size they should obey*
- implicitWidth and implicitHeight should be used bottom up
  - *A component expresses its wishes for a size*
- **Be aware:** Components with a wrong size will work incorrectly with anchors

Demo: qml-modules-components/ex-button

- For Components
  - Don't specify x, y, z or anchors on the top-level item.
  - Use childrenRect to simplify calculating the size.

```
implicitWidth: leftItem.width + rightItem.anchors.leftMargin
                              + rightItem.width // BAD
implicitWidth: childrenRect.width // GOOD
```

  - In most cases, the size of the component should match the size of its visual appearance.
- General
  - Avoid conflicts between anchors and position/size properties.
  - Don't hardcode the size.

```
implicitHeight: 80 // BAD
implicitHeight: label.implicitHeight // GOOD
```

- Composing User Interfaces
- Animations
- User Interaction
- Components
- **Presenting Data**
  - Arranging Items
  - Simple Data Models
  - Views

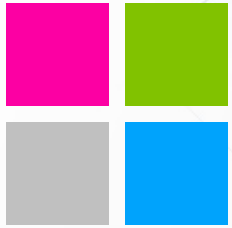
Can manipulate and present data:

- Familiarity with positioners and repeaters
  - Rows, columns, grids, flows
  - Item indexes
- Ability to define and use list models
  - Using pure models with repeaters and delegates
- Ability to use models with views
  - Using list and grid views
  - Decorating views
  - Defining delegates

- **Arranging Items**
- Simple Data Models
- Views

Positioners and repeaters make it easier to work with many items.

- Standard layouts
  - Column
  - Row
  - Grid
  - Flow
- Repeaters create items from a template.
  - For use with positioners
  - Using data from a model
- Combining these makes it easy to lay out a lot of items.



```
1  import QtQuick 2.0
2  Item {
3     width: 300; height: 300
4
5  Grid {
6      id: grid
7      x: 15; y:15
8      width: 300; height: 300
9
10     columns: 2; rows: 2; spacing: 20
11
12     Rectangle { width: 125; height: 125; color: "#fc00a3" }
13     Rectangle { width: 125; height: 125; color: "#81c200" }
14     Rectangle { width: 125; height: 125; color: "silver" }
15     Rectangle { width: 125; height: 125; color: "#00a3fc" }
16
17   }
18 }
```

Demo: qml-presenting-data/ex-grid-rectangles

- Items inside a positioner are automatically arranged.
  - In a 2 by 2 Grid
  - With horizontal/vertical spacing of 20 pixels
- x, y is the position of the first item.
- Like layouts in QtWidgets based UIs

#NOTES

```qml
import QtQuick 2.0
Item {
    width: 400; height: 400

    Grid {
        x: 5; y: 5
        rows: 5; columns: 5; spacing: 10

        Repeater {
            model: 6
            Rectangle {
                width: 70; height: 70
                color: "#00a3fc"

            }
        }
    }
}
```

- Repeater takes data from a model (just a number in this case).
- Creates items from the given delegate component.

Demo: qml-presenting-data/ex-repeater-grid

Demo: qml-presenting-data/ex-repeater-grid-full

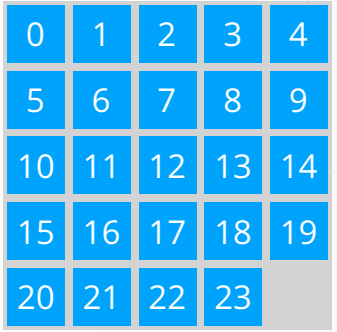
```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 400; color: "black"
5
6      Grid {
7          ...
8          Repeater { ... }
9      }
10 }
```

- The Repeater creates items.
- The Grid arranges them within its parent item.
- The outer Rectangle item provides
  - The space for generated items
  - A local coordinate system

#NOTES

Indexing Items

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 400; color: "lightgrey"
5
6      Grid {
7          x: 5; y: 5
8          rows: 5; columns: 5; spacing: 10
9
10         Repeater {
11             model: 24
12             Rectangle {
13                 width: 70; height: 70
14                 color: "#00a3fc"
15                 Text {
16                     anchors.centerIn: parent
17                     text: model.index
18                     font.pointSize: 30
19                     color: "white"
20                 }
21             }
22         }
23     }
24 }
```

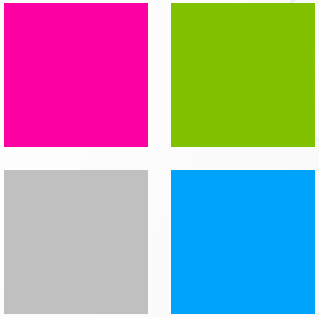


- Repeater attaches model.index to each item it creates

Demo: qml-presenting-data/ex-repeater-grid-index

Using an Array as a Model

```
1  import QtQuick 2.0
2
3  Grid {
4      x: 15; y: 15; width: 300; height: 300
5
6      columns: 2; rows: 2; spacing: 20
7
8      Repeater {
9          model:  ["#fc00a3", "#81c200", "silver", "#00a3fc"]
10         Rectangle {
11             width: 125; height: 125; color: model.modelData
12         }
13     }
14 }
```



Demo: qml-presenting-data/ex-array-model

- Arranging Items
- **Simple Data Models**
- Views

Models and views provide a way to handle data sets.

- Models hold data or items.
- Views display data or items
  - Using delegates.

Pure models provide access to data:

- ListModel

Visual models provide information about how to display data:

- Visual item model: VisualItemModel
  - Contains child items that are supplied to views
- Delegate model: DelegateModel
  - Contains an interface to an underlying model
  - Supplies a delegate for rendering

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 150; height: 200; color: "white"
5
6      ListModel {
7          id: nameModel
8          ListElement { name: "Alice"; age:1 }
9          ListElement { name: "Bob"; age:3 }
10         ListElement { name: "Jane" ; age:5}
11         ListElement { name: "Victor"; age:6 }
12         ListElement { name: "Wendy"; age:1 }
13     }
14
15     Component {
16         id: nameDelegate
17         Text {
18             text: name + " " + age
19             font.pixelSize: 32
20         }
21     }
22
23     Column {
24         anchors.fill: parent
25         Repeater {
26             model: nameModel
27             delegate: nameDelegate
28         }
29     }
30 }
```

Alice 1
Bob 3
Jane 5
Victor 6
Wendy 1

Demo: qml-presenting-data/ex-list-model-repeater

- List models contain simple sequences of elements.
- Each ListElement contains
  - One or more pieces of data
  - Defined using properties
  - *No information* about how to display itself
- ListElement does not have pre-defined properties.
  - All properties are custom properties.

```
1 ListModel {
2     ListElement { ... }
3     ListElement { ... }
4     ...
5 }
```

#NOTES

```
1 ListModel {
2     id: nameModel
3     ListElement { name: "Alice" }
4     ListElement { name: "Bob" }
5     ListElement { name: "Jane" }
6     ListElement { name: "Victor" }
7     ListElement { name: "Wendy" }
8 }
```

Alice 1
Bob 3
Jane 5
Victor 6
Wendy 1

- Define a ListModel
  - With an id so it can be referenced
- Define ListElement child objects
  - Each with a name property
  - The property will be referenced by a delegate.

#NOTES

```
1 Component {
2     id: nameDelegate
3     Text {
4         text: model.name; font.pixelSize: 32
5     }
6 }
```

Alice 1
Bob 3
Jane 5
Victor 6
Wendy 1

- Define a Component to use as a delegate.
  - Describes how the data will be displayed.
- Properties of list elements can be referenced.
  - Use a Text item for each list element.
  - Use the value of the model.name property from each element.
- In the item inside a Component
  - The parent property refers to the view.
  - A ListView attached property can also be used from the root item of the delegate to access the view.

#NOTES

```
1 Column {
2     anchors.fill: parent
3     Repeater {
4         model: nameModel
5         delegate: nameDelegate
6     }
7 }
```

Alice 1
Bob 3
Jane 5
Victor 6
Wendy 1

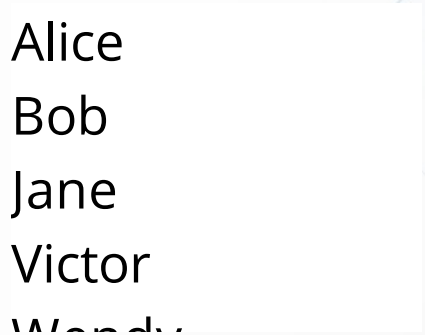
- A Repeater fetches elements from nameModel
  - Using the delegate to display elements as Text items.
- A Column arranges them vertically
  - Using anchors to make room for the items.

#NOTES

- Arranging Items
- Simple Data Models
- **Views**
  - The Path View

- ListView shows a classic list of items
  - With horizontal or vertical placing of items.
- GridView displays items in a grid.
  - Like an file manager's icon view
- PathView displays items on a path.
  - Items follow a path, and the items move instead of the selection.

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 250; height: 200; color: "white"
5
6      ListModel {
7          id: nameModel
8          ListElement { name: "Alice" }
9          ListElement { name: "Bob" }
10         ListElement { name: "Jane" }
11         ListElement { name: "Victor" }
12         ListElement { name: "Wendy" }
13     }
14
15     ListView {
16         anchors.fill: parent
17         model: nameModel
18         delegate: Item {
19             id: topRect
20             height: texti.implicitHeight
21             anchors { left: parent.left; right: parent.right }
22             Text {
23                 anchors.fill: parent
24                 id: texti
25                 text: model.name
26                 color:"black"
27                 font.pixelSize: 32
28
29             }
30         }
31         clip: true
32     }
```



Optional Lab: Add Selection to the ListView

- Observe that the ListView doesn't provide selection out of the box.
- Pretty easy to do with a colored rectangle, though

Lab: qml-presenting-data/ex-list-model-list-view

Solution: qml-presenting-data/ex-list-model-list-view-with-selection

```
ListModel {
    id: nameModel
    ListElement { file: "../images/square.png"
                  name: "shape 1" }
    ListElement { file: "../images/triangle.png"
                  name: "shape 2" }
    ListElement { file: "../images/polygon.png"
                  name: "shape 3" }
    ListElement { file: "../images/circle.png"
                  name: "shape 4" }
}
```

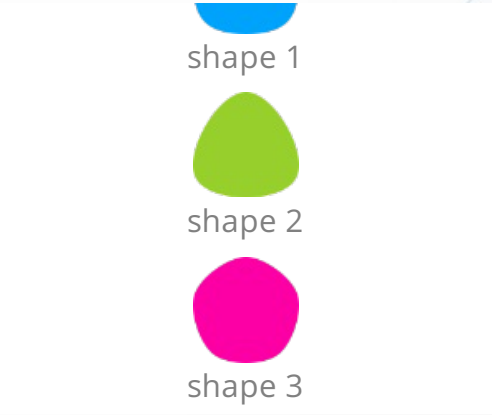
```
Component {
    id: nameDelegate
    Column {
        Image {
            id: delegateImage
            anchors.horizontalCenter: delegateText.horizontalCenter
            source: model.file; width: 64; height: 64; smooth: true
            fillMode: Image.PreserveAspectFit
        }
        Text {
            id: delegateText
            text: model.name; font.pixelSize: 24
        }
    }
}
```

```
GridView {
    anchors.fill: parent
    model: nameModel
    delegate: Column {
        Image {
            id: delegateImage
            anchors.horizontalCenter: delegateText.horizontalCenter
            source: file; width: 64; height: 64; antialiasing: true
            fillMode: Image.PreserveAspectFit
        }
        Text {
            id: delegateText; color: "grey"
            text: name; font.pixelSize: 19
        }
    }
    clip: true
}
```

Demo: qml-presenting-data/ex-list-model-grid-view

- **The Path View**

```
1  PathView {
2      id:foo
3      anchors.fill: parent
4      model: nameModel
5      delegate: nameDelegate
6      focus: true
7
8      path: Path { ~~~ d
9
10     Keys.onLeftPressed: decrementCurrentIndex()
11     Keys.onRightPressed: incrementCurrentIndex()
12 }
```



- Selected item is always at the the same location.
- Animation is already provided by the PathView.

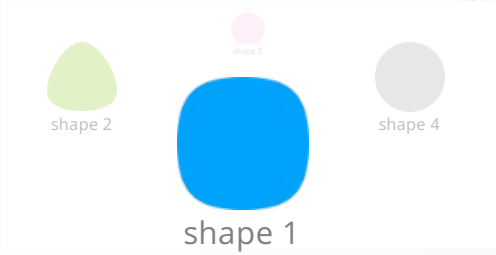
Demo: qml-presenting-data/ex-list-model-path-view

- Specifying attributes:

```
Path {
    startX: 150; startY: 200
    PathAttribute { name: "scale"; value: 1.0 }
    PathAttribute { name: "opacity"; value: 1.0 }

    PathCubic { x: 50; y: 150;
                control1X: 100; control1Y: 200
                control2X: 50; control2Y: 175 }
    PathAttribute { name: "scale"; value: 0.5 }
    PathAttribute { name: "opacity"; value: 0.5 }
    ...
}
```

- Using the attributes:

```
Component {
    id: nameDelegate
    Column {
        opacity: PathView.opacity
        z: PathView.z
        scale: PathView.scale
        Image {  ~~~ d
        Text { ~~~ d
    }
}
```



Demo: qml-presenting-data/ex-path-view-decoration

Review the code and try to answer these questions, by experimenting with the code:

- What makes the spin view show only one full digit?
- Try highlighting the current item, by making its background, say, gray.
- What does this code do:

```
preferredHighlightBegin : 0.5
preferredHighlightEnd : 0.5
```

- Optional: Rewrite SpinLock to use a Repeater and make it possible to specify the amount of digits.
- Optional: Make the property *code* work again.
  Hint: Look at the API for the repeater to get access to the Spins generated.
- Optional: Make the digits start out in a random pattern.

Lab: qml-presenting-data/lab-spinlock

- QML Deeper Dive
- **Introduction to Efectcs**
  - Graphical Effects

- **Graphical Effects**
  - Getting Started
  - Effects List
  - Effects Considerations

Objectives

- What can you do
- Import statement
- Types of effects
- The size
- Speed and memory consumption

- **Getting Started**
- Effects List
- Effects Considerations

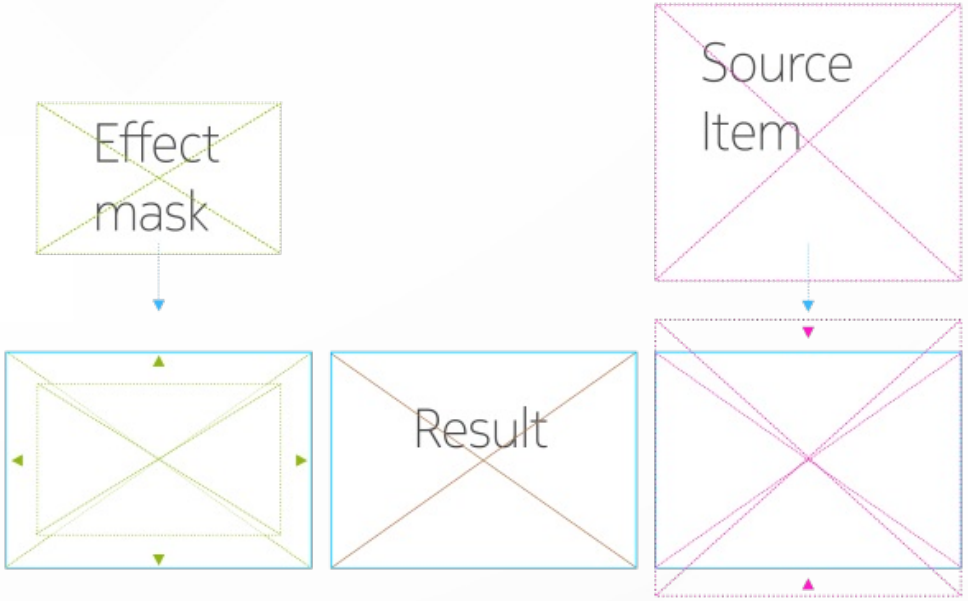
- "import QtGraphicalEffects 1.0"

- Is now
  - "import QtQuick.Studio.Effects"

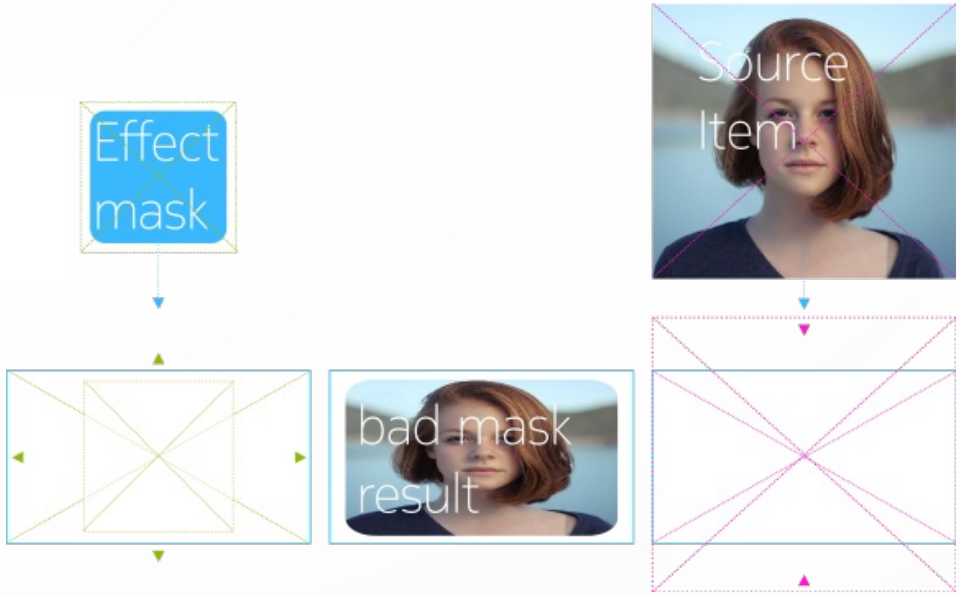
```
import QtQuick 2.0
import QtGraphicalEffects 1.0

Item {
    width: 300; height: 300

    Grid {
        id: grid
        columns: 2; rows: 2; spacing: 20
        anchors.centerIn: parent
        visible: false//<-- slide

        Repeater {
            model:  ["#fc00a3", "#81c200", "silver", "#00a3fc"]
            Rectangle {
                width: 125; height: 125; color: model.modelData
            }
        }
    }

    Image {
        id: displacementmap
        source: "../images/displacement.png"
        anchors.fill: grid
        visible: false
    }

    Displace {
        id: displacedResult
        anchors.fill: grid
        source: grid
        displacementSource: displacementmap
        displacement: 0.0
        antialiasing: true
        smooth: true

        NumberAnimation {
            target: displacedResult
            property: "displacement"
            duration: 12200
            easing.type: Easing.InOutQuad
            to: 0.5
            running: true
        }
    }
```

Demo: qml-designers/ex-intro

Effect anatomy



- An opacity mask
  - Results in a squashed image and squashed corners



Sizes must match example

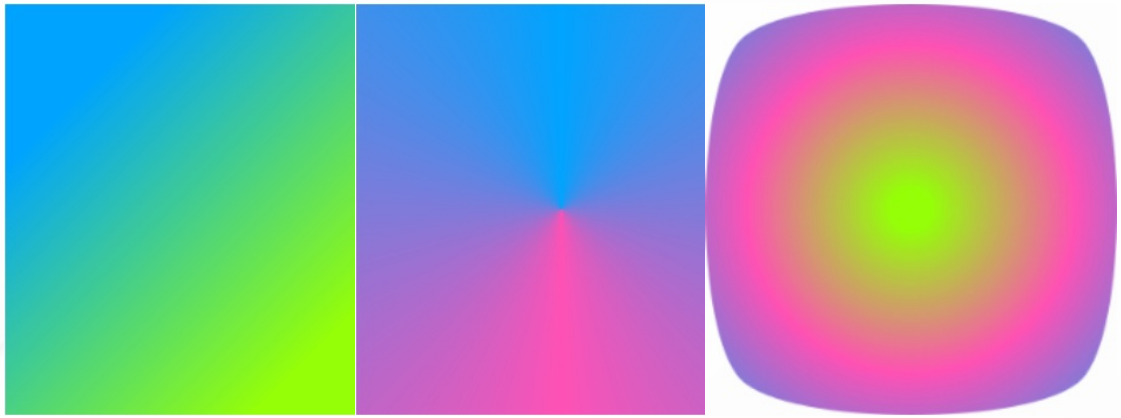
```qml
import QtQuick 2.5
//import QtGraphicalEffects 1.0
import QtQuick.Studio.Effects 1.0
Item {
    width: 300; height: 300
    Image {
        id: photo
        width: 290; height: 290
        fillMode: Image.PreserveAspectCrop
        source: "../images/photo.png"
        visible: false
        anchors.centerIn: parent
    }
    Item {
        id:mask
        anchors.fill: photo
        visible: false
        Rectangle {
            width: 290; height: 290
            radius: 40
            anchors.centerIn: parent }
    }
    OpacityMaskEffect {
        anchors.fill: photo
        source: photo
        maskSource: mask
         }
}
```

Demo: qml-designers/ex-framePhoto

- Getting Started
- **Effects List**
- Effects Considerations

- Blend
  - Blend *Merges two source items by using a blend mode*
- Color
  - BrightnessContrast *Adjusts brightness and contrast*
  - ColorOverlay *Alters the colors of the source item by applying an overlay color*
  - Colorize *Sets the color in the HSL color space*
  - Desaturate *Reduces the saturation of the colors*
  - GammaAdjust *Alters the luminance of the source item*
  - HueSaturation *Alters the source item colors in the HSL color space*
  - LevelAdjust *Adjusts color levels in the RGBA color space*

- Gradient
  - ConicalGradient *Draws a conical gradient*
  - LinearGradient *Draws a linear gradient*
  - RadialGradient *Draws a radial gradient*
- Distortion
  - Displace *Moves the pixels of the source item according to the given displacement map*
- Drop Shadow
  - DropShadow *Generates a soft shadow behind the source item*
  - InnerShadow *Generates a colorized and blurred shadow inside the source*



```
1  Item {
2      width: 900; height: 300
3      LinearGradient {
4                x: 25; y:25; width: 250; height: width
5                start: Qt.point(0, 0)
6                end: Qt.point(250, 250)
7                gradient: Gradient {
8                    GradientStop { position: 0.2; color: "#00a4fe" }
9                    GradientStop { position: 0.8; color: "#95ff08" }
10               }
11         }
```

Demo: qml-designers/ex-gradients

- Blur
  - FastBlur *Applies a fast blur effect to one or more source items*
  - GaussianBlur *Applies a higher quality blur effect*
  - MaskedBlur *Applies a blur effect with a varying intesity*
  - RecursiveBlur *Blurs repeatedly, providing a strong blur effect*
- Motion Blur
  - DirectionalBlur *Applies blur effect to the specified direction*
  - RadialBlur *Applies directional blur in a circular direction around the items center point*
  - ZoomBlur *Applies directional blur effect towards source items center point*

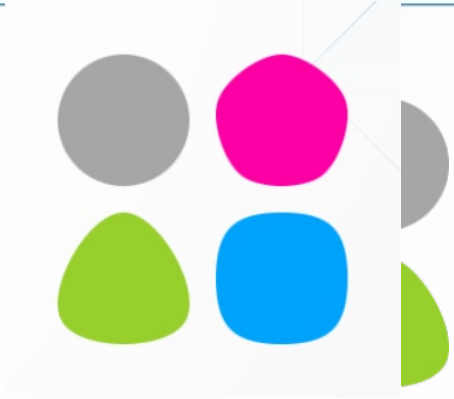
```
Item {
    id: root
    width: 900; height: 300
    property real blurLevel: 16

        GaussianBlur {
            width: sourceImage.width; height: sourceImage.height
            source: sourceImage
            radius: blurLevel; samples: 60 }
        FastBlur {
            width: sourceImage.width; height: sourceImage.height
            source: sourceImage
            radius: blurLevel}
```

Demo: qml-designers/ex-blur

- Glow
  - Glow *Generates a halo like glow around the source item*
  - RectangularGlow *Generates a blurred and colorized rectangle, which gives the impression that the source is glowing*
- Mask
  - OpacityMask *Masks the source item with another item*
  - ThresholdMask *Masks the source item with another item and applies a threshold value*

- Getting Started
- Effects List
- **Effects Considerations**

- ShaderEffectSource.



```
Item {
    width: 600; height: 300
    Item {
        id: original
        width: 300; height: width
        Grid {
            id: grid
            columns: 2; rows: 2; spacing: 20
            anchors.centerIn: parent
            Repeater {
                model:  ["poligon", "circle", "square", "triangle"]
                Image {
                    width: 100; height: 100;//......

    ShaderEffectSource {
        anchors.fill: original; sourceItem: original
        transform: Scale { xScale: -1; origin.x: 300 }
    }
```

Demo: qml-designers/ex-copy

- 32bits per Pixel for the effect size per item
  - Avoid effects loops
  - Fast prototyping, *If you create a long chain of effects consider merging all them on a single shader*
  - Possible to reduce the Mask and other trasformative elements memory footprint *via textureSize of the internal qtgrafical efects ShaderEffectSources*.
  - Consider GPU image Compression methods

Lab: qtDesignStudio/lab-final/lab-final