Introduction au QML et à la création de composants

Capitole du Libre 2018

Presented by Franck Arrecot

Objectifs de cette présentation

- Comprendre la philosophie de QML
- Découvrir la syntaxe QML
- Comprendre les concepts QML
- Appréhender la conception de composants

Introduction au QML

- Qt Quick est un module Qt
- QML langage déclaratif instanciant les objets Qt Quick
- Création rapide d'interface réactive typée moble
- Backend openGL utile dans un contexte embarqué

Hello world en QML : point d'entrée C++

- Une application QML reste un programme C++

```
 1 #include <QGuiApplication>
 2 #include <QQuickView>
 3
 4 int main(int argc, char *argv[])
 5 {
 6     QGuiApplication app(argc, argv);
 7
 8     QQuickView view;
 9     view.setSource(QUrl("main.qml"));
10     view.show();
11
12     return app.exec();
13 }
```

Hello world en QML : point d'entrée QML

- Le fichier main.qml racine de la scène
- Langage déclaratif typé JSON : clé - valeur
- Relation d'imbrication parent / enfant


```
1 import QtQuick 2.9
2
3 Rectangle {
4     width: 400
5     height: 200
6     color: "lightblue"
7 }
```

Exemple d'un main QML simple

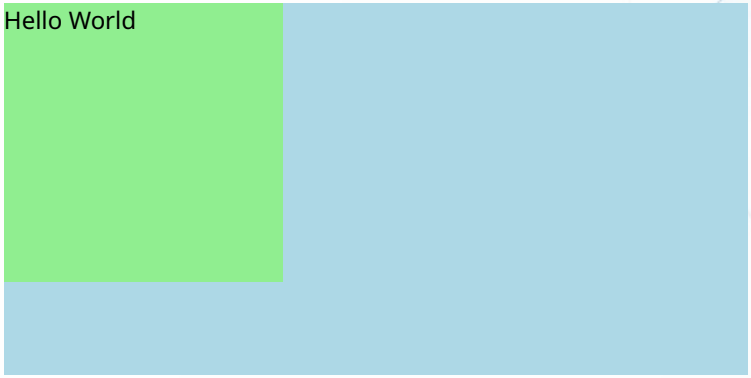
- Concepts QML : Elément, Proprieté, Type, Valeur


```qml
import QtQuick 2.9

Rectangle {
    width: 400
    height: 200
    x: 0; y: 0
    color: "lightblue"
}
```

Exemple d'un main QML avec imbrication

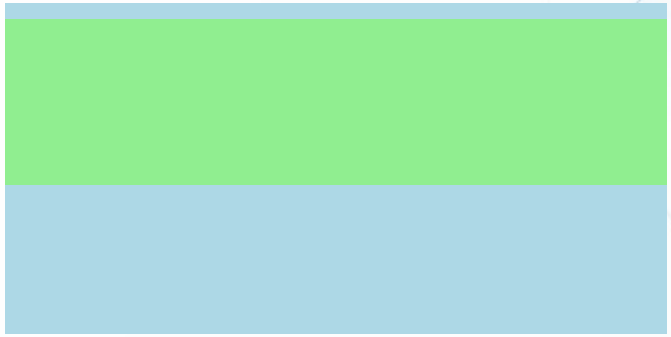
- Relation de parenté
- Propriété spéciale : binding




```
1  import QtQuick 2.9
2
3  Rectangle {
4      width: 400
5      height: 200
6      color: "lightblue"
7
8      Rectangle {
9          width: 150
10         height : width
11         color: "lightgreen"
12
13         Text {
14             text: "Hello World"
15         }
16     }
17 }
```

Exemple d'un main QML avec imbrication (cont'd)

- Proprieté id pour référencer un élément
- Binding inter-éléments



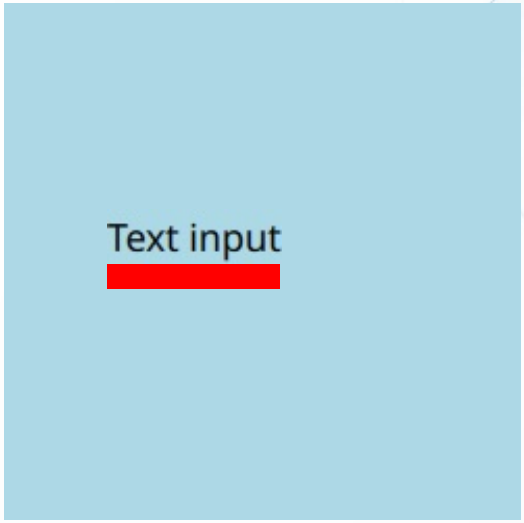
```
 1 import QtQuick 2.9
 2
 3 Rectangle {
 4     id: root
 5     width: 400
 6     height: 200
 7     color: "lightblue"
 8
 9     Rectangle {
10         y: 10
11         height: 100
12         width: root.width
13         color: "lightgreen"
14     }
15 }
```

Utilisation du mot-clé parent

```
1  import QtQuick 2.9
2
3  Rectangle {
4      //id: root
5      width: 400
6      height: 200
7      color: "lightblue"
8
9      Rectangle {
10         y: 10
11         height: 100
12         width: parent.width
13         color: "lightgreen"
14     }
15 }
```

Exemple de réévaluation d'un binding

```
1  import QtQuick 2.9
2
3  Rectangle {
4      width: 250
5      height: 250
6      color: "lightblue"
7
8      TextInput {
9          id: textInput
10         x: 50; y: 100
11         text: "Text input"
12         font.pixelSize: 18
13
14         Rectangle {
15             y: textInput.height
16             width: textInput.width
17             height : 12
18             color: "red"
19         }
20     }
21 }
```



Différents types de propriétés QML

- **Propriétés standards** auxquelles on attribue des valeurs:

```
1 Text {
2     text: "Hello world"
3     height: 50
4 }
```

- **Propriétés d'identité** donnant une nom à un élément:

```
1 Text {
2     id: label
3     text: "Hello world"
4 }
```

Différents types de propriétés QML (cont'd)

- **Propriétés groupées** qui regroupent plusieurs propriétés ayant un lien:

```
1 Text {
2     font.family: "Helvetica"
3     font.pixelSize: 24
4 }
```

- **Propriétés personnalisées** qui peuvent être ajoutées à élément:

```
1 Rectangle {
2     property real mass: 100.0
3 }
4
5 Circle {
6     property real radius: 50.0
7 }
```

- **Propriétés attachées** qui peuvent être attachées sur un élément

Résumé

- Module Qt Quick vs language QML
- Organisation d'un programme QML
- Découverte de la syntaxe et des concepts de bases
- Aspect dynamique du binding
- Notion d'arborescence et de parentée

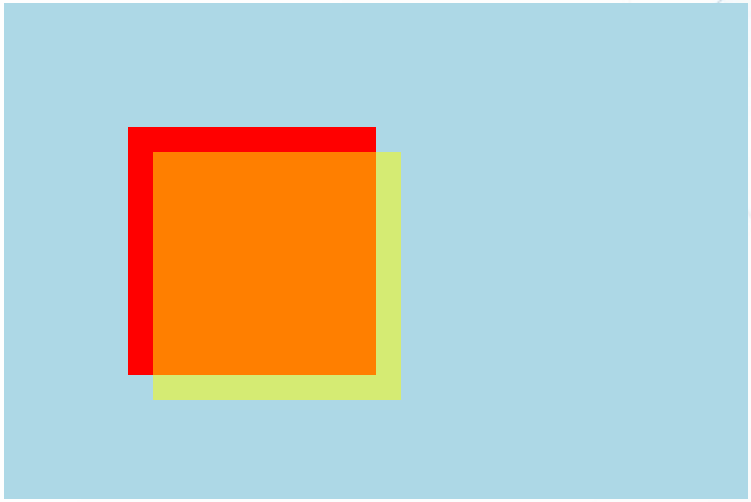
Conséquences de la relation de parentée

- Système de coordonnées
- Priorité d'affichage


```
1  import QtQuick 2.9
2  Rectangle {
3      width: 600
4      height: 400
5      color: "lightblue"
6
7      Rectangle {
8          x: 100
9          y: 100
10         width: 200
11         height : 200
12         color: "red"
13     }
14
15     Rectangle {
16         x: 100
17         y: 100
18         width: 200
19         height : 200
20         color: "yellow"
21     }
22 }
```

Propriété d'opacité

- Systeme de coordonnées
- Priorité d'affichage



```
1  import QtQuick 2.9
2  Rectangle {
3      width: 600
4      height: 400
5      color: "lightblue"
6
7      Rectangle {
8          x: 100
9          y: 100
10         width: 200
11         height : 200
12         color: "red"
13     }
14
15     Rectangle {
16         x: 120
17         y: 120
18         width: 200
19         height : 200
20         color: "yellow"
21         opacity: 0.5
22     }
23 }
```

Clipping en QML

```
1  import QtQuick 2.9
2
3  Rectangle {
4      width: 600
5      height: 400
6      color: "lightblue"
7
8      Rectangle {
9          x: 100
10         y: 100
11         width: 650
12         height : 200
13         color: "yellow"
14         opacity: 0.5
15     }
16 }
```

Propriété clip

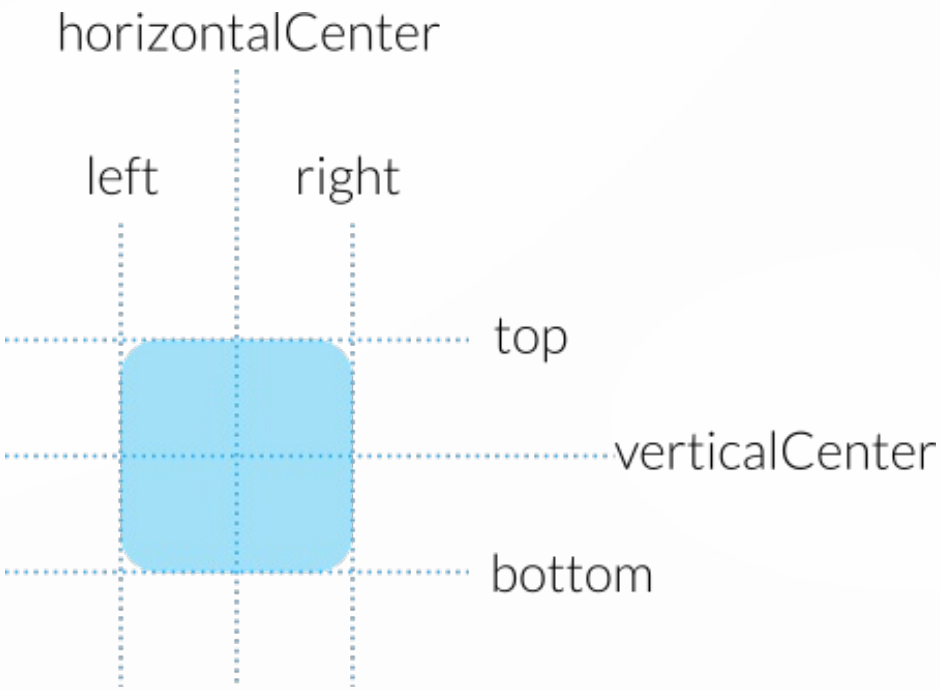
```
1  import QtQuick 2.9
2  Rectangle {
3      width: 600
4      height: 400
5      color: "lightblue"
6      clip: true
7
8      Rectangle {
9          x: 100
10         y: 100
11         width: 650
12         height : 200
13         color: "yellow"
14         opacity: 0.5
15     }
16 }
```

Systeme d'Ancrage

- Mechanisme pour positionner les éléments les uns par rapport aux autres
- Les ancres sont des bindings qui se réévaluent dynamiquement
- Deux types d'ancres
  - Lignes d'ancrage (left, top etc...).
  - Ancrage utilitaire (centerIn, fill).

Ligne d'Ancrage

- Définit des lignes imaginaires sur les bords ou le centre d'un élément
- Une ligne d'ancrage ne peut être alignée qu'avec une autre ligne d'ancrage compatible



Ligne d'Ancrage - exemple

```
 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 }
```

- Contrainte de position d'un élément avec un autre (parent ou frère)
- Référence directe aux lignes d'ancrage :
  - On utilise background.right
  - et pas background.anchors.right

Ligne d'Ancrage - exemple (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 }
```

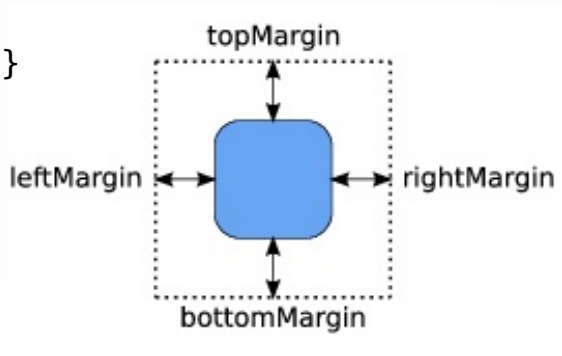
Ligne d'Ancrage - exemple (cont'd)

```
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 }
```

- Contraint la **position** et **la taille** d'un élément par rapport à un autre
- Les ancres peuvent être utilisées en parrallèle de la taille
  - Utiliser width et les ancres left / right donne la priorité aux ancres

Marges

```
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 }
```

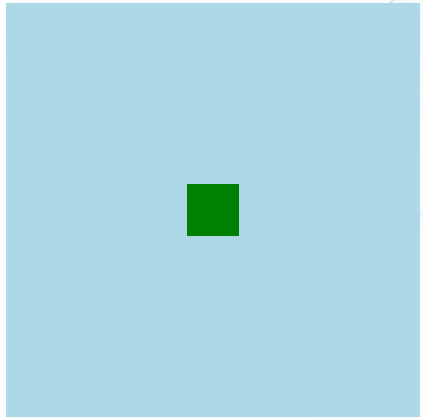


Ancres utilitaires

- Deux ancres utilitaires
  - centerIn pour centrer un élément à l'interieur d'un autre
  - fill pour qu'un élément en remplisse un autre
- Les ancres utilitaires prennent comme valeur l'id d'un élément
  - On ne peut faire référence qu'aux id des éléments parent et frères
  - Possibilité d'utiliser le mot-clé parent

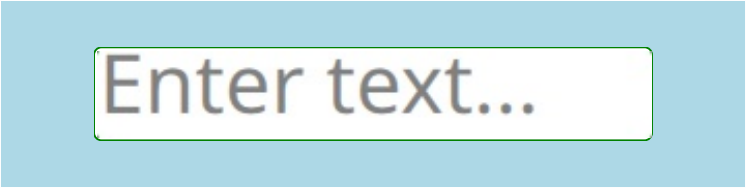
Ancres utilitaires - exemple

```
1  import QtQuick 2.0
2
3  Rectangle {
4      id: rectangle1
5      width: 400; height: 400
6      color: "lightblue"
7
8      Rectangle {
9          color: "green"
10         width: 50; height: 50
11         anchors.centerIn: rectangle1
12     }
13 }
```



Tour d'horizon des éléments Qt Quick

- Assemblage de briques QML de base
- Découverte d'éléments Qt Quick de base

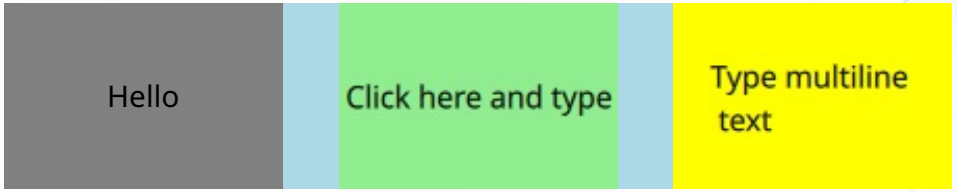


Type de bases

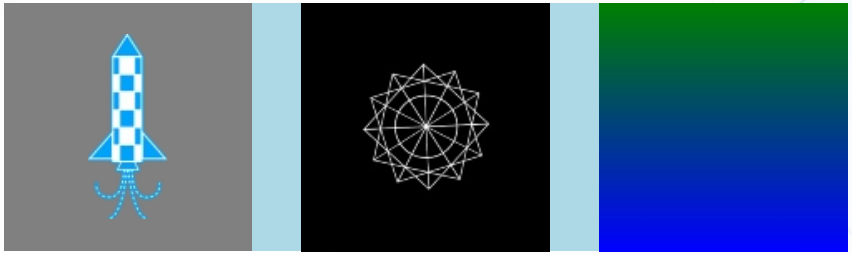
```
1  import QtQuick 2.9
2
3  Rectangle {
4
5      width: 600
6      height: 400
7      color: "lightblue"
8
9      QtObject {
10         id: toto
11         objectName: "toto"
12     }
13
14     Item {
15         x: 0; y:0
16         width: 100; height: 100
17         opacity: 1
18     }
19 }
```

Gestion des champs texte

```
1  import QtQuick 2.9
2
3  Rectangle {
4      width: 500; height: 100;
5      color: "lightblue"
6
7      Text {
8          text: "Hello"
9          font.pixelSize: 16
10     }
11
12     TextInput {
13         text: "Click here and type"
14         font.pixelSize: 16
15     }
16
17     TextEdit {
18         text: "Type multiline \n text"
19         font.pixelSize: 16
20     }
21 }
```



Gestion des images



```
1  Image {
2      source: "images/rocket.png"
3      width: 150; smooth: true
4      fillMode: Image.PreserveAspectFit
5  }
6
7  AnimatedImage {
8      width: 200; height: 200
9      source: "images/image-animated.gif"
10 }
11
12 Rectangle {
13     width: 150; height: 150
14     gradient: Gradient {
15         GradientStop { position: 0.0; color: "green" }
16         GradientStop { position: 1.0; color: "blue" }
17     }
18 }
```

Résumé

- Système de coordonées
- Ordre d'affichage des éléments
- Utilisation des ancres pour placer ses éléments
- Découverte des divers éléments Qt Quick

Gestion des signaux : MouseArea

```
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 }
```

- Signal handler pour réagir à l'émission d'un signal : onSignal
- Paramètre du signal dans la documentation ou la déclaration

Gestion des évènements : MouseArea en binding

```
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 }
```

- Utilisation des propriétés de MouseArea dans un binding
- Evènements instantanés et évènement d'état

Le risque d'écrasement du binding

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 200;
5      color: "lightblue"
6
7      Text {
8          text: "Press Me"
9          color: mouseArea.pressed ? "green" : "black"
10         anchors.centerIn: parent
11         font.pixelSize: 32
12
13         MouseArea {
14             id: mouseArea
15             anchors.fill: parent
16             onDoubleClicked: parent.color = "yellow"
17         }
18     }
19 }
```

- Un binding est une fonction qui se réévalue dynamiquement
- Contexte javascript vs Contexte QML

Syntax: **function <name>([parameter1, ...]) { ... }**

Example:

```
1  Text {
2      id: myItem
3      text: "Hello"
4      function clear() {
5          text = "";
6      }
7  }
8
9  // Dans un contexte javascript
10 myItem.clear()
```

Résumé

- Signaux et gestionnaires de signaux
- Contexte QML et contexte Javascript
- Signaux vs propriétés bindées
- Ecrasement d'un binding

Création et conception de nos propres composants

Création et conception de nos propres composants

- Création de composants dans des fichiers qml séparés
- Comprendre la visiblité et les méchanismes d'exposition
- Concevoir des composants réutilisables et configurables
- Reflexion sur l'API d'un composant QML

Exemple d'élément à réutiliser

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 100;
5      color: "lightblue"
6
7      Rectangle {
8          width: 300
9          height: 50
10         anchors.centerIn: parent
11
12         border.color: "green"
13         color: "white"
14         radius: 4; smooth: true
15         clip: true
16
17         TextInput {
18             anchors.fill: parent
19             anchors.margins: 2
20             text: "Enter text..."
21             color: focus ? "black" : "gray"
22             font.pixelSize: parent.height - 6
23         }
24     }
25 }
```

Instanciation notre élément

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 100
5      color: "lightblue"
6
7      LineEdit {
8          anchors.centerIn: parent
9          width: 300; height: 50
10     }
11 }
```

- Elément réutilisable factorisé dans un fichier dédié : *LineEdit.qml*

Fichier LineEdit.qml

```
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         id: textInput
11         anchors.fill: parent
12         anchors.margins: 2
13         text: "Enter text..."
14         color: focus ? "black" : "gray"
15         font.pixelSize: parent.height - 6
16     }
17 }
```

Notion de visibilité

```qml
import QtQuick 2.9

Rectangle {
    width: 400; height: 100;
    color: "lightblue"

    LineEdit{
        anchors.centerIn: parent
        width: 300; height: 50
    }
}
```

```qml
import QtQuick 2.0

Rectangle {
    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 - 6
    }
}
```

- Première approche de type dérivé en QML
- Proprietés à la racine du composant accessibles
- Notion de configuration via exposition de propriétés internes

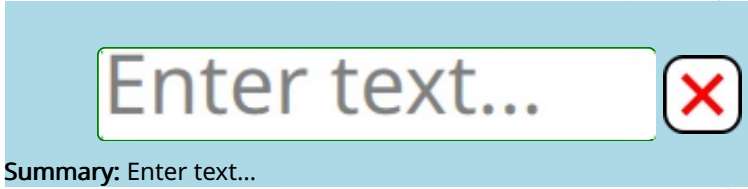
Rappel - Propriétés personnalisées

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

```
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
```

Exposition d'une valeur interne

```
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 - 6
18     }
19 }
```



Utilisation de la propriété exposée

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 100;
5      color: "lightblue"
6
7      LineEdit {
8          id: lineEdit
9          anchors.centerIn: parent
10         width: 300; height: 50
11     }
12
13     Text {
14         anchors.bottom: parent.bottom
15         text: "<b>Summary:</b> " + lineEdit.text
16     }
17 }
```

Modification de la propriété exposée

```
1  import QtQuick 2.0
2
3  Rectangle {
4      width: 400; height: 100;
5      color: "lightblue"
6
7      LineEdit {
8          id: lineEdit
9          anchors.centerIn: parent
10         width: 300; height: 50
11     }
12
13     Text {
14         anchors.bottom: parent.bottom
15         text: "<b>Summary:</b> " + lineEdit.text
16     }
17
18     Image {
19         id: clearButton
20         source: "../images/clear.svg"
21         anchors { right: parent.right; rightMargin: 4
22                   verticalCenter: lineEdit.verticalCenter }
23         opacity: lineEdit.text === "" ? 0.25 : 1.0
24         MouseArea {
25             anchors.fill: parent
26             onClicked: lineEdit.text = ""
27         }
28     }
29 }
```

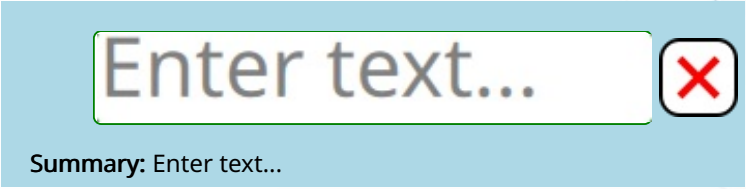
Mot-clé readonly d'une propriété

```
1  import QtQuick 2.0
2
3  Rectangle {
4      readonly 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 - 6
18     }
19 }
```

- Si modifié au runtime, la console affichera une erreur : Type Error...
- Necessité d'un mechanisme d'exposition en écriture

Exposition en lecture et écriture : l'Alias

```
1  import QtQuick 2.0
2
3  Rectangle {
4      property alias 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 - 6
18     }
19 }
```



Créer ses méthodes depuis un composant

Syntax: **function <name>([parameter1, ...]) { ... }**

```
1 Rectangle {
2     id: rect
3     property string text: "Hello"
4     function clear() {
5         text = "";
6         ...
7     }
8 }
```

Appel de la fonction

```
rect.clear()
```

Créer ses propres signaux

Syntaxe du signal : **signal <name>[(<type> <name>, ...)]**

Syntaxe du signal handler :**on<Name>: <expression>**

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

- Signal handler associé : onChecked
- Argument passé sous le nom : checkValue

Créer ses propres signaux - exemple

```
1  import QtQuick 2.9
2
3  Rectangle {
4      id: root
5      signal accepted(string text);
6      readonly property string text: textInput.text
7
8      TextInput {
9          id: textInput
10         anchors.fill: parent
11         anchors.margins: 2
12         text: "Enter text..."
13         color: focus ? "black" : "gray"
14         font.pixelSize: parent.height - 6
15
16         onAccepted: root.accepted(text)
17     }
18 }
```

Utilisation du signal handler - exemple

```
1  import QtQuick 2.9
2
3  Rectangle {
4      width: 400; height: 100;
5      color: "lightblue"
6
7      LineEdit {
8          id: lineEdit
9          anchors.centerIn: parent
10         width: 300; height: 50
11         border.width: 2
12         border.color: "yellow"
13
14         onAccepted: lineEdit.border.color = (text == "") ? "red" : "yellow"
15     }
16
17     Text {
18         anchors.bottom: parent.bottom
19         text: "<b>Summary:</b> " + lineEdit.text
20     }
21 }
```

Signaux autogenérés

```
1  import QtQuick 2.9
2
3  Rectangle {
4      width: 400; height: 100;
5      color: "lightblue"
6
7      LineEdit {
8          id: lineEdit
9          anchors.centerIn: parent
10         width: 300; height: 50
11         border.width: 2
12         border.color: "yellow"
13
14         onTextChanged: lineEdit.border.color = (text == "") ? "red" : "yellow"
15     }
16
17     Text {
18         anchors.bottom: parent.bottom
19         text: "<b>Summary:</b> " + lineEdit.text
20     }
21 }
```

Résumé et conception d'API

- Notion de visibilité
- Notion d'exposition
- Emission de signaux
- Exposer des fonctions

Merci à tous de votre attention,
Si vous avez des questions