## 1. Use properties bindings instead of functions

DO

```
Rectangle {
    id:firstRect
    width: Math.min(otherRect.width,10)
    color: "grey"
}
```

DON'T

```
Rectangle {
  id:firstRect
  color: "grey"
  Component.onCompleted: {
      firstRect.width = (otherRect.width < 10) ?
          otherRect.width : 10;
  }
}
```

## 2. Avoid as much as possible treatment in JavaScript

## 3. Use Loader as soon as necessary

## 4. Use properties or connections to access to external QML objects (in other file)

DO

```
Item {
  id: root
  property real aSize: 10

  Rectangle {
    width: root.aSize
  }
}
```

DON'T

```
Item {
  id: root

  Rectangle {
    width: externalObject.aSize
  }
}
```

## 5. The root item is named «root»

DO

```
Item {
  // This is my root item
  id: root
}
```

DON'T

```
Item {
  // This is my root item
  id: myItem
}
```

## 6. Prefixe external object properties calls by id for readability

DO

```
Item {
  id: parentItem
  property real aSize: 10

  Rectangle {
    width: parentItem.aSize
  }
}
```

DON'T

```
Item {
  id: parentItem
  property real aSize: 10

  Rectangle {
    width: aSize
  }
}
```

## 7. Always use anchors and layout for the items position

DO

```
ListView {
  id: myListView
  anchors {left: leftItem.right}
}
```

DON'T

```
ListView {
  id: myListView
  x: rect1.y + rect1.height
  y: 55
}
```

## 8. Do not use anchors on an item that is an immediate child of a layout

DO

```
RowLayout {
  Rectangle {
    Layout.fillWidth: true
    Layout.minimumWidth: 55
}
```

DON'T

```
RowLayout {
  Rectangle {
    anchors {left: leftItem.right}
}
```

## 9. Use QtObject to declare internal / private properties

## 10. Use states and enum to increase readability and maintainability

DO

```
property bool openned

states: [
    State {
      name: "open"
      when: root.openned
      // states for property changing
    },
      ...
]
transitions: [
    // transitions for animations
]
```

DON'T

```
property bool openned

onOpennedChanged: {
  if(root.openned) {
    // do this
  }
  else {
    // do that
  }
}
```

## 11. Use QML more for UI and let Qt/C++ handle the backend

## 12. Avoid directly manipulating the QML object tree using C++

## 13. Use group properties with the group notation for readability

DO

```
Item {
  id: myItem
  font {
    family: "Arial"
    bold: true
  }
}
```

DON'T

```
Item {
  id: myItem
  font.family: "Arial"
  font.bold: truev
}
```

## 14. Always check if a c++ pointer or javascript object isn't null or undefined

DO

```
Item {
  id: myItem
  myObject: DUtils.isValid(toto.objectCpp) ?
          toto.objectCpp : undefined
}
```

DON'T

```
Item {
  id: myItem
  myObject: toto.objectCpp
  // may trigger qml warning
  }
```

## 15. A role name must always begins by the letter «r» meaning «role»

DO

```
ListView{
  id: myListView
  delgate: Text { text: rText }
  }
```

DON'T

```
ListView{
  id: myListView
  delgate: Text { text: text }
   // will not use the role but the text
   property
  }
```

## 16. Use var type only when necessary

DO

```
property string name
property int size
property MyMenu optionsMenu
```

DON'T

```
property var name
property var size
property var optionsMenu
```

## 17. Performance considerations

**Binding expressions : things to avoid**

- Declaring intermediate JavaScript variables (except for caching purpose)
- Accessing "var" properties
- Calling JavaScript functions
- Constructing closures or defining functions within the binding expression
- Accessing properties outside of the immediate evaluation scope

**Rendering considerations**

- Clipping is disabled by default, and should only be enabled when required.
- If you have elements which are totally covered by other (opaque) elements, it is best to set their "visible" property to false or they will be drawn needlessly.
- Opaque content is generally a lot faster to draw than translucent