# QML Coding Conventions

This document contains the QML coding conventions that we follow in our documentation and examples and recommend that others follow.

## QML Object Declarations

Throughout our documentation and examples, [QML object attributes](https://doc.qt.io/qt-6/qtqml-syntax-objectattributes.html) are always structured in the following order:

* id
* property declarations
* signal declarations
* JavaScript functions
* object properties
* child objects

For better readability, we separate these different parts with an empty line.

For example, a hypothetical *photo* QML object would look like this:

[Rectangle](https://doc.qt.io/qt-6/qml-qtquick-rectangle.html) {

id: photo // id on the first line makes it easy to find an object

property [bool](https://doc.qt.io/qt-6/qml-bool.html) thumbnail: false // property declarations

property alias image: photoImage.source

signal clicked // signal declarations

function doSomething(x) // javascript functions

{

return x + photoImage.width

}

color: "gray" // object properties

x: 20 // try to group related properties together

y: 20

height: 150

width: { // large bindings

if (photoImage.width > 200) {

photoImage.width;

} else {

200;

}

}

states: [

[State](https://doc.qt.io/qt-6/qml-qtquick-state.html) {

name: "selected"

[PropertyChanges](https://doc.qt.io/qt-6/qml-qtquick-propertychanges.html) { target: border; color: "red" }

}

]

transitions: [

[Transition](https://doc.qt.io/qt-6/qml-qtquick-transition.html) {

from: ""

to: "selected"

[ColorAnimation](https://doc.qt.io/qt-6/qml-qtquick-coloranimation.html) { target: border; duration: 200 }

}

]

[Rectangle](https://doc.qt.io/qt-6/qml-qtquick-rectangle.html) { // child objects

id: border

anchors.centerIn: parent; color: "white"

[Image](https://doc.qt.io/qt-6/qml-qtquick-image.html) {

id: photoImage

anchors.centerIn: parent

}

}

}

## Grouped Properties

If using multiple properties from a group of properties, consider using *group notation* instead of *dot notation* if it improves readability.

For example, this:

[Rectangle](https://doc.qt.io/qt-6/qml-qtquick-rectangle.html) {

anchors.left: parent.left; anchors.top: parent.top; anchors.right: parent.right; anchors.leftMargin: 20

}

[Text](https://doc.qt.io/qt-6/qml-qtquick-text.html) {

text: "hello"

font.bold: true; font.italic: true; font.pixelSize: 20; font.capitalization: Font.AllUppercase

}

could be written like this:

[Rectangle](https://doc.qt.io/qt-6/qml-qtquick-rectangle.html) {

anchors { left: parent.left; top: parent.top; right: parent.right; leftMargin: 20 }

}

[Text](https://doc.qt.io/qt-6/qml-qtquick-text.html) {

text: "hello"

[font](https://doc.qt.io/qt-6/qml-font.html) { bold: true; italic: true; pixelSize: 20; capitalization: Font.AllUppercase }

}

## Unqualified access

In order to improve readability and performance always reference properties of parent components by their id explicitly:

[Item](https://doc.qt.io/qt-6/qml-qtquick-item.html) {

id: root

property [int](https://doc.qt.io/qt-6/qml-int.html) rectangleWidth: 50

[Rectangle](https://doc.qt.io/qt-6/qml-qtquick-rectangle.html) {

width: root.rectangleWidth

}

}

## Required properties

When requiring data defined outside the component, make this explicit by using [Required Properties](https://doc.qt.io/qt-6/qml-codingconventions.html" \l "required-properties). Required properties must be set or else the creation of the component will fail. These are preferable to unqualified lookups because they are more performant and allow for both users and tooling to reason about an external property's type. Additionally they remove assumptions that a component otherwise has to make about the environment in which it is created.

## Signal handlers

When handling parameters in signal handlers use functions which name them explicitly:

[MouseArea](https://doc.qt.io/qt-6/qml-qtquick-mousearea.html) {

onClicked: (event) => { console.log(`${event.x},${event.y}`); }

}

## JavaScript Code

If the script is a single expression, we recommend writing it inline:

[Rectangle](https://doc.qt.io/qt-6/qml-qtquick-rectangle.html) { color: "blue"; width: parent.width / 3 }

If the script is only a couple of lines long, we generally use a block:

[Rectangle](https://doc.qt.io/qt-6/qml-qtquick-rectangle.html) {

color: "blue"

width: {

var w = parent.width / 3

console.debug(w)

return w

}

}

If the script is more than a couple of lines long or can be used by different objects, we recommend creating a function and calling it like this:

function calculateWidth(object : Item) : double

{

var w = object.width / 3

// ...

// more javascript code

// ...

console.debug(w)

return w

}

[Rectangle](https://doc.qt.io/qt-6/qml-qtquick-rectangle.html) { color: "blue"; width: calculateWidth(parent) }

Also note that is recommended to add type annotations to your function in order to more easily reason about and refactor your application since both parameter and return types are immediately visible from the function signature.

For long scripts, we will put the functions in their own JavaScript file and import it like this:

import "myscript.js" as Script

[Rectangle](https://doc.qt.io/qt-6/qml-qtquick-rectangle.html) { color: "blue"; width: Script.calculateWidth(parent) }

If the code is longer than one line and hence within a block, we use semicolons to indicate the end of each statement:

[MouseArea](https://doc.qt.io/qt-6/qml-qtquick-mousearea.html) {

anchors.fill: parent

onClicked: {

var scenePos = mapToItem(null, mouseX, mouseY);

console.log("MouseArea was clicked at scene pos " + scenePos);

}

}