# Introduction to QML for Web developers

QML is a **declarative language**, part of Qt Quick, that allows describe how a UI looks and behaves with a JavaScript-based syntax.

13 Mar 2011

Web developers who want approach QML will find a lot of common points between the two worlds: both QML and Web development are based on **declarative programming languages**, and in both worlds **JavaScript is used to specify how the UI behaves**.

This article compares and contrasts QML and Web development technologies such as HTML, and in doing so illustrates techniques for porting between Web and QML.

*Note: this is a work in progress*

# UI Elements

Just like Web developers use HTML elements, QML is based on a **set of UI elements** that allow to build User Interfaces in a declarative manner.

All QML UI elements inherit from Item. Just like an HTML node, a QML element is a rectangle with a given size, accessible through its width and height properties. Also, just like a classical HTML element, a QML element may have one parent and more children elements.

A QML document has a root element, just like the <html> root element for HTML documents, but differently from HTML the QML root element can be any of the available Items.

## Non visual elements

Differently from HTML, QML allows to specify non visual elements as part of the QML document. Examples of non-visual elements are: Timer, Connection, FontLoader. Non-visual elements are useful to add behavior to the QML document without writing JavaScript code.

## Defining new UI Elements

Differently from HTML, QML allows to define custom reusable UI elements (called Components). Components can be defined in two different manners:

- defined in an external .qml file named with the Component's name (for instance, "MyNewComponent.qml"). Each Component is a self-contained QML document, and can be used just like any other QML element by:
  - importing it where it needs to be used
  - using the file name as the Component's name
- defined inline, within the QML document that wants to use it. In this case, Components are defined through the Component element, and are used by the container QML document via its own id.

More information about the writing of new QML components can be found here.

## Quick hints

- QML, apart from the TextEdit and TextInput elements, does not provide support for data entry elements. Anyway, there are various open source projects that provide useful form elements (for instance, checkboxes, comboboxes). One example is Colibri. For more projects, check out the [=272 Qt Quick category] on Nokia Developer Projects.
- The WebView element loads and displays complete HTML pages, in a manner similar to HTML IFrames.

# Arranging UI Elements

HTML allows to position a UI element via various CSS properties, including: top/right/bottom/left, position, float, display and many more.

On the other side, QML offers different ways to position a UI element:

- with an anchor-based method
- using Positioner Items
- using explicit x and y coordinates

Differently from HTML, where nodes are automatically arranged according to other nodes and to their nature (for instance, if "block" or "inline" nodes), QML elements are by default placed one above the other: for this reason, it is usually necessary to use one of the methods described here to arrange elements in an appropriate manner. Also, using one of the described methods excludes the others: for instance, anchoring a node to another and defining x/y coordinates brings to unexpected results.

## Anchor-based Layout

A QML element can be anchored to its siblings or to its parent element thanks to the anchor properties: anchoring an element to another means aligning one or more of its anchor lines to the anchors lines of another element. The available anchors lines are: top, right, bottom, left, horizontalCenter, verticalCenter and baseline.


(C) Copyright Nokia 2013. All rights reserved.

When anchoring an element to another, it is possible to define margins for each anchor (top, right, bottom, left), similarly to the margin-top, margin-right, margin-bottom and margin-left CSS properties. Anchor margins are specified in pixels.

More information about anchors-based layouts can be found here.

## Using Positioner Items

Positioners are special containers that automatically arrange the position and size of their children Items. QML defines four different types of Positioner:

- Column: places children in a column, just as with **HTML block nodes** (placed one under the other)
- Row: places children in a row
- Grid: places childrens in a grid, just like an **HTML Table**
- Flow: places childrens as words of a text, one after the other. Using a Flow Positioner is similar to using the **CSS float property**

More information about Positioners can be found here.

## Using X and Y coordinates

The x and y coordinates are similar to the CSS left and top properties. Differently from what happens with CSS, the QML x and y properties are always relative to the Item's parent: so, there's no inbuilt property that allows to specify an absolute position or a fixed position.

More information about the usage of x and y coordinates can be found here.

# Styling the UI

QML does not offer a CSS-like way of defining the appearance of UI elements. Instead, a set of element properties can be used for this task.

The Item element defines a common set of properties that are shared by all UI elements (as all elements inherit from Item):

- clip: can assume true and false values. If true, its content is clipped to its bounding rectangle, just like with the **overflow: hidden** CSS rule. If false, no clipping is performed, just as with the **overflow: visible** CSS rule. There is no inbuilt method to achieve an effect similar to the **overflow: auto** CSS rule.
- opacity: defines the opacity of the element, assuming values between 0.0 and 1.0. It is equivalent to the **opacity CSS property**.
- scale, rotation, transform: these properties allow to apply scale, rotation and translation transformation to the element

Different elements also define different properties that that are specific to those elements. The most important element and properties are listed below:

- Rectangle
  - color: defines a background color for the Rectangle
  - gradient: defines a Gradient that fills the Rectangle
  - border.color: defines the border's background color
  - border.width: defines the border's width
- Text, TextEdit, TextInput
  - color: defines the text color
  - font.family: defines the font family used for the text
  - font.pointSize and font.pixelSize: allow to define the font size in points and pixels

## Quick hint

- To display an element (for instance, an Image) with a border and/or with a colored background, use a Rectangle containing the desired element

# UI Manipulation

Just like an HTML node, a QML document can be represented as a tree of elements, where each element has zero or more child elements. An element can be directly referred by using its own id. So, instead of writing:

```
var element = document.getElementById("myElementId");
```

QML allows to write:

```
var element = myElementId;
```

**Accessing the children** of a given elements differs depending on the nature of the children: visual children element are accessible through the children property, while non-visual children elements are accessible through the resources property. To access both visual and non-visual children at the same time, the data property can be used.

**The parent of an element** can be accessed through its parent property.

## Dynamic elements creation


(C) Copyright Nokia 2013. All rights reserved.

Elements can be dynamically created in two different ways:

- Using the Qt.createComponent() function: this method behaves in a manner similar to the document.createElement() DOM function, allowing to create an element of a given type. While the DOM function accepts the node name as argument, the Qt.createComponent function accepts the path of the Component that must be loaded
- Using the Qt.createQmlObject() function: this function accepts a string of QML as first argument, builds the related QML structure and appends that to the specified parent node. This approach is then similar to the DOM node's innerHTML property.

**Removing an element** can be performed via its destroy() JavaScript function: **only dynamically created elements can be destroyed**.

More information about dynamic object management can be found here.

# The JavaScript side

Just like with Web development, QML allows to use JavaScript to add advanced behavior to a QML application. JavaScript code can be defined within the QML document, or in external JavaScript files. When importing an external JavaScript file, a qualifier must be used:

```
import "MyLibrary.js" as LibraryName
```

All variables and functions belonging to the imported file will be accessible through the specified qualifier:

```
Item {
    width: LibraryName.myProperty
    onClicked: LibraryName.myFunction()
}
```

## Animations

While in the Web environment animations are usually performed by the periodic update of CSS properties (for instance, x, y, opacity), QML offers a greater support to create complex transitions with plain declarative QML.

Animations in QML can be implemented using several approaches, including:

- PropertyAnimation elements - allow to specify which property values must be updated, the duration and the easing to be applied to the animation
- Behavior elements - allow to specify how a property value should be animated when its value is changed
- State changes together with Transition elements - State elements define the value of a set of properties, allowing their value to be updated at once when the specific State is applied. Transition can be applied to State changes to animate the property value's changes.

## Quick Hints

- There is no window or document object
- No global properties can be defined: all variables and functions must be locally defined, and belong to their local scope
- Due to the different environments, JavaScript frameworks intended for the Web are usually not compatible with QML
- The setTimeout() and setInterval() JavaScript functions are not supported: use Timer elements instead
- The XMLHttpRequest object is supported, and can be used to retrieve data from the network (note that synchronous mode is not supported)

More information about JavaScript usage in QML applications can be found here


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.