Tutorial: Step-by-step to create a timer-based camera

This article explains how to create with QML a camera application taking mutliple-shots using timers. We will explore step-by-step how to start such an application, and how to add features one at a time.

Note: This is an entry in the PureView Imaging Competition 2012Q2

Introduction

Qt Multimedia 1.1 introduced the Camera[1] element in QML, which provides an easy to use entry-point to the world of cell-phone photography. We will see in this article how to use this element and then augment it step-by-step with timer-based features, and how to display the previews taken from the camera.

Initial Setup

To start writing the application, let's create a new project for a Qt Quick Application in the Qt SDK for our platform. The example and source code provided here is for Meego Harmattan and Symbian^3 and Nokia Belle. The only difference in the QML files is at the top of the file, to import the Qt-Components that is appropriate for each. On Symbian, the Camera[1] element requires the UserEnvironment capability. We also need to enable Qt Multimedia Kit, and Qt Components (only required to simplify the creation of the buttons in our examples). Here are the modifications to the .pro file:

```
## Add the UserEnvironment capability to use the Camera element on Symbian
symbian:TARGET.CAPABILITY += UserEnvironment

# If your application uses the Qt Mobility libraries, uncomment the following
# lines and add the respective components to the MOBILITY variable.
CONFIG += mobility
## Add Qt Multimedia libraries
MOBILITY += multimedia

## Add dependency to Symbian components
CONFIG += qt-components
```

Capture One Image

In this first example, we create a simple interface with the Camera[1] element, and we trigger the image capture with a single button. This button calls the function *camera.captureImage()* on the Camera element, which triggers the capture. The Camera element sends the signal ImageCaptured, followed by ImageSaved once the image is written to disk. During testing, some phones (in particular the N950) appear to be having trouble trying to display the Camera[1] element when saving the image to disk. For this reason, we use these 2 signals to hide the element for a short time to let it complete its write.

To simplify the example, we leave most of the settings for the Camera element to its default, which means it will have autofocus, auto-ISO, no zoom... The default settings are quite good for most cases, unless you have something specific in mind. 2 settings are included in the example to show how to use them should you want to. The whiteBalanceMode setting is left to its default, and the Flash is turned off (just so it wouldn't illuminate the whole room every time the app is tested!).

(C) Copyright Nokia 2013. All rights reserved.

Note: If you downloaded the included archive, edit the file main.qml to have it load the Page1 element. The Page1.qml file contains the source for this example.

```
import QtQuick 1.1
import com.nokia.meego 1.1
import QtMultimediaKit 1.1

Page {
   Row {
       anchors.fill: parent
       Item {
           height: parent.height
           width: parent.width - captureButton.width

           Camera {
               id: camera

               anchors.fill: parent
               flashMode: Camera.FlashOff
               whiteBalanceMode: Camera.WhiteBalanceAuto

               onImageCaptured : {
                   console.debug("Captured Image:" +preview);
            // Hide the Camera element while the image is being written to disk
                   camera.visible = false
               }

               onImageSaved: {
                   console.log("Image saved:"+path);
                   // The image has been written, show the element again
                   camera.visible = true
               }

               onCaptureFailed: {
                   console.log("Failed: "+message)
               }
           }
       }

       Button {
           id: captureButton
           height: 100

           text: qsTr("Capture Image")
           onClicked: camera.captureImage()
       }
   }
}
```

Adding a delay

Note: If you downloaded the included archive, edit the file main.qml to have it load the Page2 element. The Page2.qml file contains the source for this example.

In this example, we will add a self-timer, so the photo happens with a configurable delay. To do so, we use the Timer[2] element.

```
    Timer {
        id: delayTimer
        interval: selectDelay.model.get(selectDelay.selectedIndex).value
        onTriggered: {
            camera.captureImage()
        }
    }
```

This Timer[2] is called with *delayTimer.start()* called from the button created in the previous example (see Page2.qml). To create the configuration dialog, we use a SelectionDialog[3] with a few options:

```
    SelectionDialog {
        id: selectDelay
        titleText: "Camera will be triggered after the initial:"
        selectedIndex: 0

        model: ListModel {
            // The value is in microseconds
            ListElement { name: "2 second"; value: 2000 }
            ListElement { name: "5 seconds"; value: 5000 }
            ListElement { name: "10 seconds"; value: 10000 }
            ListElement { name: "Immidiately"; value: 0 }
        }
    }
```

When the "Start Capture" button is pressed, the timer will grab its delay value from the SelectionDialog's selected item, and will wait for that long before triggering the Camera's element capture.

 (C) Copyright Nokia 2013. All rights reserved.

Adding multiple sequential captures

Note: If you downloaded the included archive, edit the file main.qml to have it load the Page3 element. The Page3.qml file contains the source for this example.

In this example, we add another Timer[2] and a count value which will be used to trigger the camera multiple times in a row, with a configurable delay between the shots. For the configuration, we use them same method as for the previous example, a SelectionDialog[3], with a list of acceptable values. Note that, in this example, the frequency value is the delay that the app will wait in-between shots, excluding the capture time.The timer only starts once the previous capture has completed its write to disk. This can easily be changed, but can cause some shots to be missed if the Camera element is not ready yet when we trigger the next capture.

To trigger the next timer, we use the ImageSaved signal that is sent by the Camera element, to re-start the timer.

```
            onImageSaved: {
                    camera.visible = true
                    if (cameraTimer.count < selectNumberShots.model.get(selectNumberShots.selectedIndex).value) {
                        cameraTimer.start()
                    }
            }
```

Adding preview images

Note: If you downloaded the included archive, edit the file main.qml to have it load the Page4 element. The Page4.qml file contains the source for this example.

In this last example, we will add a ListView item which will list all the preview images that have been taken by the application. We create a custom ShowPreview element, which contains the ListView, and its associated ListModel (name previewList). We also use a helper function add(item) which appends the new image to the model. This is called with ''showPreview.add({"path":path})'' when the ImageSaved signal is sent by the Camera element.

```
import QtQuick 1.1

Item {
    function add(item) {
       previewList.append(item)
    }

    ListView {
       anchors.fill: parent
       model: previewList
```


(C) Copyright Nokia 2013. All rights reserved.

```
        delegate: showImage
    }

    Component {
        id: showImage
        Image {
            source: path
            width: 110
            height: 110
        }
    }

    ListModel {
        id: previewList
    }

}
```

Where to go from here

Although these are fairly simple examples, they can easily be augmented with other simple features. Here are a few ideas to continue this project:

- If we would like to take shots as fast as possible, it would be beneficial is lock the Camera settings after the first exposures. For example, keep the focus that the Camera element found in the first image for the ones after. This would reduce the amount of time it takes per capture, and the shots would be closer to each other.
- Exposure bracketing: Use this method to take 3 shots in a row, setting a different Exposure Compensation for each shot. These can then be post-processed into an HDR photo (High Dynamic Range).
- Additional control of settings: Use the QML declarative camera example to see how to control all the Camera settings, and how to augment our example with manual control over those.
- Flash "bracketing": Take 2 sequential shots, one with flash, one without, so they can be compared afterwards, in order to keep the better one.

References

1. ↑ [1.0] [1.1] [1.2] [1.3] QML Camera, Camera element API doc.
2. ↑ [2.0] [2.1] [2.2] QML Timer
3. ↑ [3.0] [3.1] QML SelectionDialog

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.