How to turn your camera flash into a torch on Harmattan using GStreamer

This article explains how to control the N9/N950 camera flash in order to use it as a torch

Introduction

Qt Mobility will have the torch support soon, but in the meanwhile I'm going to show you how to control the camera led using a more low level API: GStreamer. This solution will work even on devices running PR 1.0 software

The implementation: QML part

Since Harmattan supports only QML application I'm going to show you how to make a simple torch application which make use of Harmattan components to switch on and off the camera led. Below you can see the QML part which simply define a button and the torch element. When the button is pressed the torch is switched on or off.

```
import QtQuick 1.1
import com.nokia.meego 1.0
import Torch 1.0

Page {
    Button{
        id: mButton
        anchors.centerIn: parent
        text: "Switch ON"
        onClicked: mTorch.toggle();
    }

    Torch {
      id: mTorch
      onStatusChanged: mButton.text = on ? "Switch OFF" : "Switch ON";
    }
}
```

The implementation: C++ part

The torch element that we have seen in the QML file is actually a C++ QObject-derived class. On the main.cpp file we have registered the class as Torch QML element so that it can be instanced by QML itself.

```
...
#include "torch.h"
#include <QtDeclarative>

Q_DECL_EXPORT int main(int argc, char *argv[])
{
    ...
    qmlRegisterType<torch, 1>("Torch", 1, 0, "Torch");
   ....
}
```

The class which actually manage the flash camera hardware is the following one. It makes use of GStreamer, which is a linux multimedia framework based on pipelines. The minimum pipeline which help us to achieve our target is the following one:

```
gst-launch-0.10 subdevsrc video-torch=1 viewfinder-mode=1
```

```
#include "torch.h"
#include <gst/gst.h>
#include <QDebug>

torch::torch(QObject *parent) :
    QObject(parent), src(0), mStatus(false)
{
    //gst-launch-0.10 subdevsrc video-torch=1 viewfinder-mode=1 ! fakesink

    gst_init(NULL, NULL);
    src = gst_element_factory_make("subdevsrc", "src");

    if (!src)
        return;

    g_object_set(G_OBJECT(src), "video-torch", 1, NULL);
    g_object_set(G_OBJECT(src), "viewfinder-mode", 1, NULL);
    gst_element_set_state(src, GST_STATE_NULL);
}

void torch::start(){
    qDebug() << "START";
    gst_element_set_state(src, GST_STATE_PLAYING);
}

void torch::stop(){
    qDebug() << "STOP";
    gst_element_set_state(src, GST_STATE_NULL);
}

void torch::toggle(){
```


(c) Copyright Nokia 2013. All rights reserved.

```
    qDebug() << "TOGGLE";
    setStatus(!mStatus);
}

bool torch::status(){
    qDebug() << "STATUS";
    return mStatus;
}

void torch::setStatus(bool on){
    qDebug() << "SET STATUS" << on;
    if (on){
        emit statusChanged(true);
        start();
    } else {
        emit statusChanged(false);
        stop();
    }

    mStatus = on;
}
```

The manifest file

Since this API access to camera hw, we need to define an appropriate manifest file which give to our app the correct capabilities.

```
<aegis>
<request policy="add">
<credential name="GRP::video" />
</request>
<for path="/opt/torch/bin/torch" />
<for path="applauncherd-launcher::/usr/bin/applauncherd.bin" id="" />
</aegis>
```


© Copyright Nokia 2013. All rights reserved.

 (c) Copyright Nokia 2013. All rights reserved.