How to avoid QTimer intervals being affected by graphical updates

Submitted by jesper.melin_567 on Tue, 2018-02-20 16:36

This article will present one solution how to avoid QTimers instantiated in C++ (backend) to be affected by graphical changes in QML (frontend).

Whenever QObjects are instantiated in C++ they will be created and belong to the thread which they were created in. If no other threads are created during runtime, then there will only be the Main thread available. The QML code will always be executed in the Main thread; it is therefore recommended to not have CPU-intensive calculations performed in the Main thread since this may affect the graphical performance.

If the graphics require CPU-time due to changes of the layout (loading new content etc.) this will affect any QObjects running in the Main thread. If we would create a QTimer in the Main thread in C++ and assign it a short interval time and then load content in QML, then the interval could be heavily affected, leading to an uneven pace of timeout triggers from the QTimer.

If there is a need for an even pace which is not affected by changes in the GUI (or other QObjects) running in the Main thread, then here is one example on how achieve it.

Use separate threads

The created QTimers will need to run in their own separate threads to avoid being affected by other QObjects or the GUI running in the Main thread.

Any class which inherits from the QObject class may easily be moved to their own thread using the QThread class. To guarantee that QTimers are actually created in the new thread there a few options, but one of the easier ways are by connecting the signal started from QThread to an appropriate slot of the class which should hold the QTimer. Our example will consist of a class named *CyclicWorker* which has a slot called *Initialize* which will create the QTimer.

Note that this is not an complete example, you will need to apply the strategy based on your need.

A class which inherits from the QObject class:

```
class CyclicWorker : public QObject [1]
{
    Q_OBJECT
    public:
     public slots:
```

```
    void Initialize();
     private:
    void pollInput():
    QTimer [2] *m_timer;
}

/**
 * @brief Initialize the timer once the class have been moved to a separate
 */
void CyclicWorker::Initialize()
{
    m_timer = new QTimer [2](this);
    m_timer->setInterval(100);
    m_timer->setTimerType(Qt [3]::PreciseTimer);
    connect(m_timer, &QTimer [2]::timeout,this, &CyclicWorker::
pollInput, Qt [3]::DirectConnection);
}
```

Instanciate the class and move it to a separate thread using these steps:

```
QThread [4] threadHandle;
CyclicWorker m_cyclicWorker;
 QObject [1]::connect(&threadHandle, &QThread [4]::started,
 m_cyclicWorker, &CyclicWorker::Initialise);
m_cyclicWorker->moveToThread(&threadHandle);
threadHandle.start();
```

Once the thread has started it will trigger the *Initialize* slot in *CyclicWorker* which then will create the QTimer and assign in to the member variable *m_timer*. This timer will now execute in the new thread and not in the Main thread, leading to a stable interval betweeen each *timeout*.

Category:
Qt Programming [5]

Source URL: https://support.crosscontrol.com/kb/how-avoid-qtimer-intervals-being-affected-graphical-updates

Links
[1] http://doc.trolltech.com/latest/qobject.html
[2] http://doc.trolltech.com/latest/qtimer.html
[3] http://doc.trolltech.com/latest/qt.html
[4] http://doc.trolltech.com/latest/qthread.html
[5] https://support.crosscontrol.com/kb/qt