Handling Events in Qt

Overview

In this article we will draw a picture how to handle events by using the specialized event handler `timerEvent()`

Basic Idea

We will implement a widget that displays the clock time in the local display format and also the current date in the appropriate format, alternating every ten seconds. The display itself should update every second.

The below two screen shots recorded at an interval of ten seconds

Class Definition

We implement the clock in a class called **ClockWidget**, which we derive from QLCDNumber, a Qt class that provides an imitation of an **LCD** display.

```
#ifndef CLOCKWIDGET_H
#define CLOCKWIDGET_H
#include <QMainWindow>
#include <QLCDNumber>
namespace Ui {
    class ClockWidget;
}
class QTimerEvent;
class ClockWidget : public QLCDNumber
{
    Q_OBJECT
public:
    explicit ClockWidget(QWidget *parent = 0);
    ~ClockWidget();
private:
    Ui::ClockWidget *ui;
protected:
void timerEvent(QTimerEvent *e);
private:
int updateTimer, switchTimer;
bool showClock;
};
#endif // CLOCKWIDGET_H
```

Here we are particularly interested in, besides the constructor, the specialized event handler `timerEvent()`, which will update the clock time. In the **updateTimer** and **switchTimer** member variables we save numbers that serve as identifiers for the timers. The **showClock** status flag determines whether the clock time **(showClock=true)** or the date **(showClock=false)** appears on the widget.

(C) Copyright Nokia 2013. All rights reserved.

Class Implementation

The implementation in `clockwidget.cpp` begins by specifying the form of the display. Usually `QLCDNumber` shows a frame around the digital display. This behavior, inherited from QFrame, is disabled by the `QFrame::NoFrame` frame style. In addition we dissuade the widget from drawing the LCD elements with shadows and a border, by passing on `QLCDNumber::Flat` to the widget's `setSegmentStyle()` method.

```
#include "clockwidget.h"
#include <QtGui>
ClockWidget::ClockWidget(QWidget *parent) :
   QLCDNumber(parent), showClock(true)
{
    setFrameShape(QFrame::NoFrame);
    setSegmentStyle(QLCDNumber::Flat);
    updateTimer = startTimer(1000);
    switchTimer = startTimer(10000);
    QTimerEvent *e = new QTimerEvent(updateTimer);
    QCoreApplication::postEvent(this, e);
}
```

Now we need two timers. Each `QObject` can start a timer using the `startTimer()` method. As an argument `startTimer()` expects the number of seconds that must pass before it triggers a QTimerEvent, which is addressed to the current widget. Each `QTimerEvent` in turn contains an identification number, which is returned by the invocation of `startTimer()` that originates it. We manually send a timer event with the ID of the update-Timer, using the `postEvent()` method of `QCoreApplication`, so that we do not have to wait for a second to elapse before the time appears on the widget's display.

In the `timerEvent()` method we first check whether the pointer to the event really is valid—just to be on the safe side. Next, if the event contains the **switchTimer** ID, this only toggles the **showClock** variable. The actual work awaits in the last conditional statement, which is triggered by an event containing the **updateTimer** ID.

```
void ClockWidget::timerEvent(QTimerEvent *e)
{
if (!e) return;
if (e->timerId() == switchTimer)
showClock = !showClock;
if (e->timerId() == updateTimer) {
if (showClock) {
QTime time = QTime::currentTime();
QString str = time.toString(Qt::LocalDate);
setNumDigits(str.length());
display(str);
} else {
QDate date = QDate::currentDate();
QString str = date.toString(Qt::LocalDate);
setNumDigits(str.length());
display(str);
}
}
}
```

If the widget is supposed to display the time, then we first determine the current time. In Qt, the QTime class is responsible for handling time. The `currentTime()` static method of this provides the current system time in a `QTime` object. This time is converted by `toString()` into a `QString`. `Qt::LocalDate` instructs the method to take into account the country settings (locales)of the user. Finally we must inform the display how many LCD digit positions are required. We deduce this from the string length and display the string with `display()`.

`QLCDNumber` cannot display all alphanumerical characters, we use `setNumDigits()` for that. On the other hand, if **showClock** is set to false, which means that the widget should display just the date, we proceed in the same way with the QDate class, which in Qt is responsible for managing date specifications, and whose API corresponds almost exactly to that of `QTime`.

Source Code

The full source code presented in this article is available here File:ClockWidgetQt.zip

Reference Articles

1. Archived:QTimer example
2. Archived:Creating a Digital Clock QWidget

--somnathbanik 12:49, 5 May 2011 (UTC)

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.