Lomse library. Tutorial 4 for wxWidgets

In previous tutorial we learn hoe to implement score playback. But we didn't include any visual effects during score playback (visual tracking by highlighting notes being played and/or by displaying a vertical tempo line marking beats). In this tutorial we are going to learn how to do this. You can download the full source code for this example from ../../examples/example_4_wxwidgets.cpp.

Table of content

1. Lomse playback: score highlight events

As we learn in previous tutorial, invocation of `ScorePlayer::play()` method will trigger all the playback machinery. As a consequence, Lomse will start playback and will generate two types of events: *sound* events and *score highlight* events. Sound event must always be handled by your application code, but handling *score highlight* events is optional.

When your application handles *score highlight* events it is important to return control to Lomse as soon as possible. This is because, currently, Lomse does not implement an event subsystem with its own thread. Instead Lomse sends events to your application by invoking a callback. This implies that your application code for handling the event is processed by the Lomse thread. As this thread is dealing with sound generation, any delay will cause delays in the music tempo, which will be unacceptable. Therefore, to avoid delays, the suggested way for handling *score highlight* events is to generate an application event and to enqueue it in the application events system.

In your application, processing an *score highlight* event can be as complex as needed for generating any fancy visual feedback. Alternatively, it can be a trivial task, as Lomse provides standard event handlers for creating standard visual effects (coloured notes, tempo line or both). If you would like to use Lomse standard visual effects, your application code should just pass the event to the interactor, and it will take care of all necessary tasks. Finally, your application will receive an *update view* event, and all you have to do is to copy the already updated rendering bitmap onto the window.

As you can see, handling *score highlight* events in your application is a round trip to, finally, delegate its handling on Lomse! But this round trip has an important gain: the code for implementing visual highlight is executed in your application thread instead of in Lomse playback thread. As a consequence, all concurrency problems automatically disappear!

To avoid this burden in your application, probably the best solution will be to implement, in Lomse, an event subsystem with its own thread. But there is a lot of work to do and I have to prioritize the necessities. If you would like to contribute to Lomse project by working on this issue you are welcome. Please post me a message. Thank you.

2. Changes for receiving score highlight events

In this tutorial we will use the code from tutorial 3 and modify it for handling *score highlight* events. First step is to prepare our application to handle these events. For this, we will define a callback method in `MyFrame`. This is the standard procedure for handling events, of any type, sent by Lomse. As explained in tutorial 2 (see section Events sent by Lomse) setting a callback requires defining two methods: an static one (the wrapper method) and the real one that will do the job. Here is our definition:

```
class MyFrame: public wxFrame
{
public:
    ...
    //callback wrappers
    ...
    static void wrapper_lomse_event(void* pThis, SpEventInfo pEvent);

protected:
    ...
    void on_lomse_event(SpEventInfo pEvent);
```

Apart of defining the callback method we have to inform Lomse about its existence!. We do it at Lomse initialization:

```
void MyFrame::initialize_lomse()
{
    ...
    //set required callbacks
    ...
    m_lomse.set_notify_callback(this, wrapper_lomse_event);
}
```

Implementing these methods is straightforward. The wrapper method is just invoking the real one:

```
void MyFrame::wrapper_lomse_event(void* pThis, SpEventInfo pEvent)
{
    static_cast(pThis)->on_lomse_event(pEvent);
}
```

As to the method doing the real work, it has to create an application event and to enqueue it in the application events system, as explained in Lomse playback: score highlight events. Therefore, before coding this method we have to define our own application event.

3. Defining our application events

As we have to deal with Lomse events of type `k_highlight_event`, we will define an equivalent wxWidgets event: `MyScoreHighlightEvent`. It will wrap the Lomse event. As this is standard wxWidgets coding, I will not enter into details. Here is the declaration:

```
//----------------------------------------------------------------------------------------
// MyScoreHighlightEvent
//      An event to signal different actions related to
//      highlighting / unhighlighting notes while they are being played.
//----------------------------------------------------------------------------------------

DECLARE_EVENT_TYPE( MY_EVT_SCORE_HIGHLIGHT_TYPE, -1 )
```

```
class MyScoreHighlightEvent : public wxEvent
{
private:
    SpEventScoreHighlight m_pEvent;   //lomse event

public:
    MyScoreHighlightEvent(SpEventScoreHighlight pEvent, int id = 0)
        : wxEvent(id, MY_EVT_SCORE_HIGHLIGHT_TYPE)
        , m_pEvent(pEvent)
    {
    }

    // copy constructor
    MyScoreHighlightEvent(const MyScoreHighlightEvent& event)
        : wxEvent(event)
        , m_pEvent( event.m_pEvent )
    {
    }

    // clone constructor. Required for sending with wxPostEvent()
    virtual wxEvent *Clone() const { return new MyScoreHighlightEvent(*this); }

    // accessors
    SpEventScoreHighlight get_lomse_event() { return m_pEvent; }
};

typedef void (wxEvtHandler::*ScoreHighlightEventFunction)(MyScoreHighlightEvent&);

#define MY_EVT_SCORE_HIGHLIGHT(fn) \
    DECLARE_EVENT_TABLE_ENTRY( MY_EVT_SCORE_HIGHLIGHT_TYPE, wxID_ANY, -1, \
    (wxObjectEventFunction) (wxEventFunction) (wxCommandEventFunction) (wxNotifyEventFunction) \
    wxStaticCastEvent( ScoreHighlightEventFunction, & fn ), (wxObject *) NULL ),
```

And the definition:

```
DEFINE_EVENT_TYPE( MY_EVT_SCORE_HIGHLIGHT_TYPE )
```

Having defined our event, now we can code the required `MyFrame::on_lomse_event` method. Here is the code:

```
void MyFrame::on_lomse_event(SpEventInfo pEvent)
{
    MyCanvas* pCanvas = get_active_canvas();

    switch (pEvent->get_event_type())
    {
        case k_highlight_event:
        {
            if (pCanvas)
            {
                SpEventScoreHighlight pEv(
                    boost::static_pointer_cast(pEvent) );
                MyScoreHighlightEvent event(pEv);
                ::wxPostEvent(pCanvas, event);
            }
            break;
        }

        default:
            ;
```

```
    }
}
```

As you can see, what we do is to transform the Lomse event into a wxWidgets event, and to inject the wxEvent into the application event handling loop. That's all. Control returns immediately to Lomse, so that Lomse playback thread is not delayed. And our wxWidgets event will be processed when Lomse is idle. But for processing these wxWidgets events we have to define event handler.

4. Changes for handling score highlight events

Our playback events will be processed in `MyCanvas`. For this, first thing to do is to modify our events table, adding a new handler:

```
BEGIN_EVENT_TABLE(MyCanvas, wxWindow)
    ...
    MY_EVT_SCORE_HIGHLIGHT(MyCanvas::on_visual_highlight)
END_EVENT_TABLE()
```

Next, we will define the handler method:

```
class MyCanvas : public wxWindow, public PlayerNoGui
{
    ...
protected:
    //event handlers
    ...
    void on_visual_highlight(MyScoreHighlightEvent& event);
```

As we will implement standard visual effects, the implementation of this method is trivial, as it is just delegating in Lomse:

```
void MyCanvas::on_visual_highlight(MyScoreHighlightEvent& event)
{
    SpEventScoreHighlight pEv = event.get_lomse_event();
    WpInteractor wpInteractor = pEv->get_interactor();
    if (SpInteractor sp = wpInteractor.lock())
        sp->on_visual_highlight(pEv);
}
```

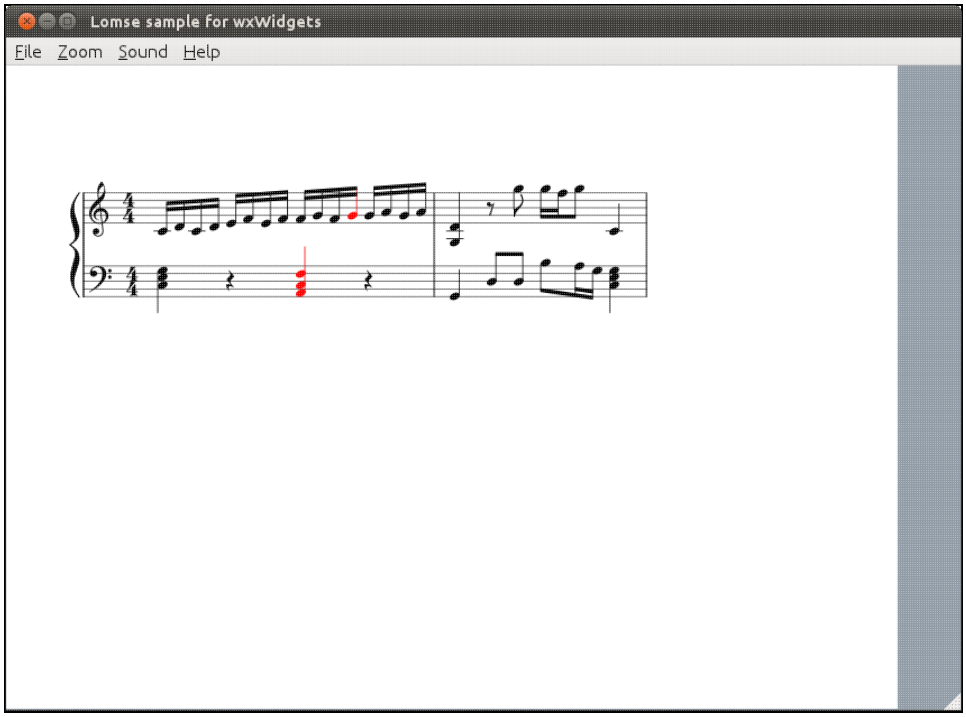
And this is all. Our code is now finished and our application is ready for building and testing. You can download the full source code for this example from ../../examples/example_4_wxwidgets.cpp.

5. Compiling your code and building

At this point, we could compile and tests all these changes. Open a terminal window at enter (adjust paths if necessary):

```
gcc example_4_wxwidgets.cpp ./wxMidi/wxMidi.cpp ./wxMidi/wxMidiDatabase.cpp -o example-4-wx \
    `wx-config --cflags` `pkg-config --cflags liblomse`  -I ./wxMidi/    \
    `pkg-config --libs liblomse` -lstdc++ -lportmidi -lporttime `wx-config --libs`
```

When executing example-4 and playing the score you will hear the music and you will see that notes get coloured in red as they are being played. You should see something as:



6. Conclusions

In this tutorial I have shown the way to implement visual effects during score playback. It also opens the door for creating your own visual effects, as you are not limited by standard visual effects canned in Lomse.

If you would like to contribute with more tutorials or by adapting this tutorial for other platforms, you are welcome!. Join the Lomse list and post me a message.