Subject: Re: need some expert advice to extend the LineEdit
Posted by dolik.rce on Sun, 02 Dec 2012 16:38:20 GMT
View Forum Message <> Reply to Message

OK, so the reason is curiosity/self-education I am OK with that...

The EditorBar as you correctly found is what makes the line numbers work. In theory, it is nothing more than a FrameCtrl in the CodeEditors frame that paints the line numbers and other info. The "friend class" declaration is there only to allow direct access to the private members and thus make the implementation a bit simpler.

The EditorBar in CodeEditor does actually much more then line numbering. It takes alo care of annotations, breakpoints, ifdef tracking etc... To show it to you in simpler form, here is a commented example of it's very stupid cousin

```
#include <CtrlLib/CtrlLib.h>

using namespace Upp;

class MyEditorBar : public FrameLeft<Ctrl> {
public:
 virtual void Paint(Draw& w);
 LineEdit* editor; // we need to know who we work for, so we can
                   // read some necessary info (font, line positions etc.)
public:
 void SetEditor(LineEdit *e)          { editor = e; }

 MyEditorBar() {
  Width(27); // set some default width
             // note that 27 will not work with more than 99 lines ;)
 };
 virtual ~MyEditorBar() {};
};

void MyEditorBar::Paint(Draw& w)
{
 Size sz = GetSize();
 w.DrawRect(0, 0, sz.cx, sz.cy, SColorLtFace); // paint background

 if(!editor)
  return; // can't work without editor

 // read some info from the associated editor
 int fy = editor->GetFontSize().cy;
 Font f = editor->GetFont();
 int line = editor->GetScrollPos().y;
 int linecount = editor->GetLineCount();
```

```
 // iterate over all the visible lines
 int y = 0;
 while(y < sz.cy) {
  if(line == linecount)
   break; // we reached last line

  if(editor->GetCaret().top == y) {
   // current line should be highlighted
   w.DrawRect(0, y, sz.cx, fy, Blend(SColorHighlight(), SColorLtFace(), 200));
   w.DrawText(2, y + 2, IntStr(line+1), f, SColorHighlightText());
  } else {
   //other lines should be rendered normally
   w.DrawRect(0, y, sz.cx, fy, SColorLtFace());
   w.DrawText(2, y + 2, IntStr(line+1), f);
  }

  y += fy;
  line++;
 }
}

class LineEditExtended : public LineEdit {
 typedef LineEditExtended CLASSNAME;
 MyEditorBar bar;
public:
 LineEditExtended(){
  AddFrame(bar); // add the frame
  bar.SetEditor(this);
 }
 virtual bool Key(dword key, int count){
  // refresh every time user presses something (he might add/remove lines or scroll)
  bar.Refresh();
  // and forward the event to parent class
  return LineEdit::Key(key, count);
 }
 virtual Image MouseEvent(int event, Point p, int zdelta, dword keyflags){
  // refresh every time user uses mouse (he might cut/paste lines or scroll)
  bar.Refresh();
  // and forward the event to parent class
  return LineEdit::MouseEvent(event, p, zdelta, keyflags);
 }
};

class MainWindowDlg : public TopWindow {
 typedef MainWindowDlg CLASSNAME;
public:
 MainWindowDlg(){
  Add(edit.SizePos());
```

```
  edit <<= "Some\nsample\ntext...\n";
 }
private:
 LineEditExtended edit;
};

GUI_APP_MAIN
{
    MainWindowDlg().Run();
}
```

It is not complex at all, if you remove all the clutter around Basically just a Paint function and some simple overrides in the LineEditExtended class to keep things synced.

Also, just for your reference, I attach a very simple tool I wrote for myself some time ago. It uses CodeEditor to issue commands to SqlLite database. It is rather stupid, but you can see that using CodeEditor can be very simple To get idea of other capabilities of any class, it is always good to read the public part of its interface, the methods in U++ are usually quite self-explaining.

Honza

File Attachments

1) main.cpp, downloaded 368 times