Subject: What do you think about this approach to making CodeEditor more user extendable?
Posted by cbpporter on Tue, 15 Dec 2015 11:07:19 GMT
View Forum Message <> Reply to Message

Hi Mirek,

In my project I require a powerful code editor and I'm using CodeEditor. While it is a very powerful class, it is not that easy to add features to it without forking it. And this is what I've done.

But I am thinking about how to make it much more extensible.

Currently, there is an enum describing a bunch of hard-coded languages and a few tables with hard-coded positions.

I'm thinking of an approach where CodeEditor has no hard coded support for languages. Instead of a single value describing the whole language, we replace it with a structure having options and an old language option acts as a particular collection of new boolean options.

I'll give two examples on how this could work.

First, a statement like:
if(highlight == HIGHLIGHT_CSS ? *p == '#' : pair == MAKELONG('0', 'x') || pair == MAKELONG('0', 'X'))

would become:
if(highlight.UseCSSHex ? *p == '#' : pair == MAKELONG('0', 'x') || pair == MAKELONG('0', 'X'))

We would register the CSS syntax with this option set to true and for other registered syntaxes to false.

Then in CSyntax I added support for #region tags, like in C#. Rather than checking if it is C# or other languages that support #region, one would check highlight.UseRegions. This option would be checked by EditorBar too in order to draw #region outlines, like for #ifdef. I implemented this using a new hardcoded language, but would like a more flexible solution.

One would register a structure with appropriate options set for a language, together with keywords and upp ids.

How does this approach sound? I already have a forked CodeEditor, so I can create an early minimal prototype if you wish.

Subject: Re: What do you think about this approach to making CodeEditor more

user extendable?
Posted by mirek on Tue, 15 Dec 2015 11:42:11 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Tue, 15 December 2015 12:07Hi Mirek,
Currently, there is an enum describing a bunch of hard-coded languages and a few tables with hard-coded positions.

I am not quite sure this is the completely correct description.

Yes, styling elements (like color of comments) are currently hardcoded. However, CodeEditor is already open w.r.t. language syntax descriptions. You can register your own syntax, with text id, with EditorSyntax::Register. You can then activate this syntax using this text id with Highlight.

Mirek

Subject: Re: What do you think about this approach to making CodeEditor more user extendable?
Posted by cbpporter on Tue, 15 Dec 2015 13:16:01 GMT
View Forum Message <> Reply to Message

Things that I'm trying to fine tune by language:
- highlighting of #ifdefs
- highlighting of @region
- highlighting of nested comments
- highlighting of doxygen like comments
- highlighting things like literal constants a bit differently based on language
- highlighting ids based on other id rules
- highlight a few symbolic meta constants as non-keywords, but as literal constants, like true, for languages where true is not a keyword, but a literal constant and should be highlighted as an int.

And achieve this in a pretty general and fast way.

And generally speaking, the C like language highlighter is able to approximately, often very closely, syntax highlight a given language, but as it currently stands:
- it is not able to properly 100% highlight some inputs. The changes would be trivial to make it 100% but
- would not be compatible with C++.

That's why I proposed a structure with bool options, to make sure that highlighting is fast. What I'd need it to handle, as just one of the examples, is "0.Foo" to be highlighted as a literal int, member selection punctuation, id. And "7.0.Bar" as literal double, member selection punctuation, id. And "7.0f.Foo" as float and so on. For all those pure OOP languages.

Subject: Re: What do you think about this approach to making CodeEditor more user extendable?
Posted by cbpporter on Wed, 16 Dec 2015 13:53:19 GMT
View Forum Message <> Reply to Message

Oh, and one more thing. Is there an HTML export function for CodeEditor? Preferably with both inline and named CSS options. If there isn't, I need to add it.

The output of this export should be the exact text that is in the editor, but with all the font formatting options.

Subject: Re: What do you think about this approach to making CodeEditor more user extendable?
Posted by koldo on Wed, 16 Dec 2015 15:16:35 GMT
View Forum Message <> Reply to Message

Quote:but with all the font formatting options
In addition it could be good to add this when copying to Clipboard and when printing.

Subject: Re: What do you think about this approach to making CodeEditor more user extendable?
Posted by mirek on Thu, 17 Dec 2015 12:29:33 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Wed, 16 December 2015 14:53Oh, and one more thing. Is there an HTML export function for CodeEditor? Preferably with both inline and named CSS options. If there isn't, I need to add it.

The output of this export should be the exact text that is in the editor, but with all the font formatting options.

Not yet, but interesting idea.

IMO, it should be possible to achieve this just using regular CodeEditor/LineEdit interface.

Subject: Re: What do you think about this approach to making CodeEditor more user extendable?
Posted by mirek on Thu, 17 Dec 2015 12:30:30 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Tue, 15 December 2015 14:16Things that I'm trying to fine tune by language:
- highlighting of #ifdefs
- highlighting of @region
- highlighting of nested comments

- highlighting of doxygen like comments
- highlighting things like literal constants a bit differently based on language
- highlighting ids based on other id rules
- highlight a few symbolic meta constants as non-keywords, but as literal constants, like true, for languages where true is not a keyword, but a literal constant and should be highlighted as an int.

And achieve this in a pretty general and fast way.

And generally speaking, the C like language highlighter is able to approximately, often very closely, syntax highlight a given language, but as it currently stands:
- it is not able to properly 100% highlight some inputs. The changes would be trivial to make it 100% but
- would not be compatible with C++.

That's why I proposed a structure with bool options, to make sure that highlighting is fast. What I'd need it to handle, as just one of the examples, is "0.Foo" to be highlighted as a literal int, member selection punctuation, id. And "7.0.Bar" as literal double, member selection punctuation, id. And "7.0f.Foo" as float and so on. For all those pure OOP languages.

OK, so just to make it clear, what you are proposing concerns CSyntax highlighter?

Subject: Re: What do you think about this approach to making CodeEditor more user extendable?
Posted by cbpporter on Thu, 17 Dec 2015 12:52:28 GMT
View Forum Message <> Reply to Message

Yes. Sorry, I should have been more clear. It is about making CSyntax easier to extend for other C like languages. The general Registering system and the other syntax classes are probably fine as they are.

I managed I think to obtain #region #endregion support as a simple patch for the official CodeEditor. I shall post it soon.

Subject: Re: What do you think about this approach to making CodeEditor more user extendable?
Posted by mirek on Thu, 17 Dec 2015 15:15:29 GMT
View Forum Message <> Reply to Message

OK, i have no problem with that, but perhaps it is as easy to just check current language, exactly as is happening now?

Subject: Re: What do you think about this approach to making CodeEditor more user extendable?
Posted by cbpporter on Thu, 17 Dec 2015 15:46:24 GMT

View Forum Message <> Reply to Message

mirek wrote on Thu, 17 December 2015 17:15OK, i have no problem with that, but perhaps it is as easy to just check current language, exactly as is happening now?

I'm doing a merge and testing stuff right now for the patch. And yes, we could probably leave things as they are in CSyntax. But some things are still a bit hard to do with a semi-hardcoded C-Like language list.

I need to add the #region and #endregion tags. So I have changed this code:

```
if(*p == '#' && findarg(highlight, HIGHLIGHT_JAVASCRIPT, HIGHLIGHT_CSS,
HIGHLIGHT_JSON) < 0) {
   static Index<String> macro;
   ONCELOCK {
    static const char *pd[] = {
     "include", "define", "error", "if", "elif", "else", "endif",
     "ifdef", "ifndef", "line", "undef", "pragma",
     // CLR
     "using"
    };
    for(int i = 0; i < __countof(pd); i++)
     macro.Add(pd[i]);
   }
```

I could add them here and that's what I'll do for the patch, but I still think it would be more elegant to have highlight be some sort of structure with fields. findarg(highlight, HIGHLIGHT_JAVASCRIPT, HIGHLIGHT_CSS, HIGHLIGHT_JSON) < 0 could become highlight.HasPreproc. And the macro list would not have to be static, but instead tied to highlight, with each separate highlight having it's own list. This way #region and #using, which are C# mostly, would not show up in other modes.

Subject: Re: What do you think about this approach to making CodeEditor more user extendable?
Posted by cbpporter on Wed, 13 Jan 2016 14:38:55 GMT

View Forum Message <> Reply to Message

Sorry, I didn't have time to touch this up until today.

I have attached a minimal version of #region #endregion support, done with the #if support.

And since you are not a big fan of my proposals for CSyntax, I think that it is best to continue and do what I'm doing right now (fork CSyntax) and maybe in a few months separate it into a new class, i.e CSyntax (unmodified from U++, handling all standard languages) and CMySyntax (handling only my cases). I'll delay this a few months since forking and merging occasionally is less work :). It is just syntax highlighting after all, hardly the most important of tasks.

File Attachments

1) CSyntax.cpp, downloaded 324 times
2) CHighlight.cpp, downloaded 298 times