Subject: Re: 2022(?).2 beta
Posted by Lance on Mon, 19 Dec 2022 22:54:10 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 19 December 2022 17:35Lance wrote on Mon, 19 December 2022 23:14mirek wrote on Mon, 19 December 2022 14:22Lance wrote on Mon, 19 December 2022 19:10And some of the viable options if multi-c++-version support is a necessity (as proposed by Klugier and me):

1. [=] to [&] when necessary. Make local copies of variables that are originally captured by value with undesired modification, and refer only to the copy in the lambda body. A fictitious example:

[&] does not help and the problem is not local copies.

This does not work:

```
struct MyApp : TopWindow {
  Button b;

  MyApp() {
     int j = 12;
     b << [&] { PromptOK(AsString(j)); };
  }
};
```

I see. Reference to local variables invalidated out of function body. So this option is eliminated. We are left with only 2.

3. Disable warning and hope that it will be deprecated for really long time. I bet it will.

See, this whole thing is rather unfortunate. There are 3 options, none of them really good. 2 of these require significant work and a chance of introducing new bugs....

1. Disable warnings option: almost effortless. but like I mentioned in a previous post, when you eventually decide to move to c++20, everybody else need to move with you overnight, or otherwise there will be the same problem of supporting pre- & post-c++20 world simultaneously. I am fine with that but not sure if other people will like it.

2. Klugier's other proposal ([this, a,b,c]). Heavy work, chance of bugs.

3. My proposal. Majority work can be done in 20 minutes. Other examples or packages, etc can be left until a bug is reported (mainly in the case old [=] doesn't involve a `this`, just change back to [=], can be fixed without thinking). I don't expect other subtle bug be introduce because of this

approach. And it should settle down in a time span of months, but involves little work on maintainers/users' part after the initial 20 minutes or so.

In a previous post, I have listed procedures I took. Replace in files, then compiler will tell the case where original [=] doesn't involves `this`, in which cases we change them back to [=]. I fail to see any chance a subtle bug will be introduced as they mean exactly same thing, just added compliance to c++20.

Lance

PS: you probably understand my approach fully. but let me explain it once more.

Context:
we have around 120ish [=] lambda capture in u++ source tree, majority of which should be changed to [=,this] from c++20 onwards. Unfortunately [=,this] is not valid pre-c++20. We want a way to make both worlds happy.

Conditional Macro: one apparent approach is to use a macro that expands to [=] with std = pre-c++20 and [=, this] when std>=c++20. If we do it properly, no bug will be introduced because of this: they are functionally equivalent.

Now suppose we have such a macro named MY_MACRO, which will be expanded to = or =,this, depending on c++ standard used.

Recommended Procedure
1. Do "Replace in Files" for all files under $UPPSRC, replace all occurences of [=] with [MY_MARCO]

2. Open a less involving package to fix the cases where no `this` were captured originally. The one I used is <examples/Color>. F7 to compile it, with -std=c++20, preferably in debug mode to save time. There will be like 3-4 cases where we have wrongfully replaced [=] with [=,this], locate them, change back to [=]. Now the bulk of jobs are done;

3. Open package `theide`, do the same thing as in step 2. We will encounter 2 other cases where we have wrongfully made the replacement. Fix them, theide will compile fine;

4. We can do a buildall on the u++ src tree(I remember we have something like that), then we can fix all such wrongful replacements at once. Or we can leave it until it's be compiled and reported by users.

5. We have a clean uppsrc that's c++-version-smart on existing lambda captures.