Subject: Re: 2022(?).2 beta
Posted by Lance on Mon, 19 Dec 2022 18:10:25 GMT
View Forum Message <> Reply to Message

And 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:

```
void ClassName::FunctionName()
{
    int i = 0;
    auto f = [=]{ this->DoSomething(); ++i; };
    ....
}
```

should be rewrite to

```
void ClassName::FunctionName()
{
    int i = 0;
    int j = i;
    auto f = [&]{ this->DoSomething(); ++j; };
    ....

}
```

pros: concise capture list; no unwanted GLOBAL MACRO;

cons: it's time consuming and error-prone for the change. Majority of affected lambda bodys need to be analysed individually to make copy& refer only to copy manually for affected variables. You get no help from the compiler. If you miss changing one of the reference to old variable name, or miss to change one of the variable you don't want to be modified, a bug arise. And it could be subtle to discover and fix all the errors. And there is no mechanism to enforce the rule, developers/maintainers (mainly both of you atm) has to watch out and be disciplined;

option 2: list individual variable in the capture list.

Pro and cons are quite similar to option 1. Less chance of subtle bugs. Potentially long and tedious capture list.

option 3: MACRO
pros: clean, fast, standard practice. Other u++ users can choose to use the MACRO in their own code to smoothen future transition form c++14 to c++20.

cons: both of you abhor the MACRO that needs to be introduced.  come on guys, if it's just

because you don't like the name, feel free to choose a better one.

Or shall we start a poll-like thing so that more input can be received? I particularly are interested to hear what @Oblivion and @Koldo have to say on this topic.

---