Subject: 2022(?).2 beta
Posted by mirek on Sat, 03 Dec 2022 14:47:32 GMT
View Forum Message <> Reply to Message

I am running out of issues with Asisst/libclang and the end of year is rapidly closing.

Therefore: Let us designate current nightly as the first beta!

Please report any problems here.

Note: For this release, I have decided to drop MacOS version (too little time to fix everything). I will fix U++ for MacOS in 2023 (including ARM support - that is probably the most significant issue now).

Mirek

Subject: Re: 2022(?).2 beta
Posted by Novo on Sun, 04 Dec 2022 16:48:55 GMT
View Forum Message <> Reply to Message

Edited.
Sorry, I missed a part about MacOS.

./umk uppsrc ide CLANG -bus +GUI,X11 won't compile.

Subject: Re: 2022(?).2 beta
Posted by mr_ped on Wed, 07 Dec 2022 23:54:53 GMT
View Forum Message <> Reply to Message

I'm doing this year Advent of Code with U++ (built from source from github 1.12. so beta-testing it), posting my solutions at github (spoilers alert, if you want to try AoC yourself).

So far everything seems to work for me quite well, I have seen only minor issues.

One very minor issue is strange temporary "freeze" of IDE after closing the terminal with finished run of code in certain situations, the detailed setup:
OS: KDE Neon (KDE5 developer's distribution based on Ubuntu 22.04 LTS with KDE desktop)
IDE config - Console binary: /usr/bin/konsole -e
project: U++ Core CLI, using Cout() to display output.
- when I run the project Ctrl+F5, it opens new terminal, outputs results, the terminal shows "<--- Finished, press [ENTER] to close the window --->"
- then I use mouse to select something from the output, Ctrl+Shift+C to copy it to clipboard, press enter to close the terminal
- the terminal does close (so far all of this is OK), but TheIDE is frozen for 5-10 seconds, not showing cursor, or reacting to clicks, etc..

- then it starts reacting again

If I don't do copy to clipboard, it never happens, when I copy something, it seems to usually freeze, although sometimes it stops doing it for following runs of the code, and does it again after restarting IDE.
It may be also some issue with KDE and clipboard thing, as did notice the OS clipboard sometimes losing content when I close the app from which the content was copied, but in this case it seems the text from Konsole survives, just IDE is stuck on something.

Other minor issue is syntax highlight of C++ numeric literals, I think Mirek did implement the apostrophe digit separator few years back, but now it looks like it does think some char string starts there, see attached image.

I haven't used U++ much in recent years, so my usage is quite "trivial", but so far everything works very well, the new clangd parsing with -std=c++20 works too, I will try to refresh the IDE build few times to not fall behind too much, and try to do a bit more stuff with it and report if I see anything more, so far it looks like solid release ahead.

File Attachments

1) num_literal_1.png, downloaded 184 times

Subject: Re: 2022(?).2 beta
Posted by mirek on Fri, 09 Dec 2022 08:56:49 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Thu, 08 December 2022 00:54I'm doing this year Advent of Code with U++ (built from source from github 1.12. so beta-testing it), posting my solutions at github (spoilers alert, if you want to try AoC yourself).

So far everything seems to work for me quite well, I have seen only minor issues.

One very minor issue is strange temporary "freeze" of IDE after closing the terminal with finished run of code in certain situations, the detailed setup:
OS: KDE Neon (KDE5 developer's distribution based on Ubuntu 22.04 LTS with KDE desktop)
IDE config - Console binary: /usr/bin/konsole -e
project: U++ Core CLI, using Cout() to display output.
- when I run the project Ctrl+F5, it opens new terminal, outputs results, the terminal shows "<--- Finished, press [ENTER] to close the window --->"
- then I use mouse to select something from the output, Ctrl+Shift+C to copy it to clipboard, press enter to close the terminal
- the terminal does close (so far all of this is OK), but TheIDE is frozen for 5-10 seconds, not showing cursor, or reacting to clicks, etc..
- then it starts reacting again

This rather feels like gtk / kde interaction error.

X11 clipboard always had problems when the providing application closes before the paste. I guess gtk is trying hard to retrieve data from closed application, then gives up after timeout.

Quote:
It may be also some issue with KDE and clipboard thing, as did notice the OS clipboard sometimes losing content when I close the app from which the content was copied, but in this case it seems the text from Konsole survives, just IDE is stuck on something.

Yep, X11 clipboard....

Quote:
Other minor issue is syntax highlight of C++ numeric literals, I think Mirek did implement the apostrophe digit separator few years back, but now it looks like it does think some char string starts there, see attached image.

Fixed (implemented).

Actually, I did not implemented apostrophe until now. But U++ highlights thousands just fine on its own (try to write 12345678 in theide :)

Mirek

Subject: Re: 2022(?).2 beta
Posted by Tom1 on Sat, 10 Dec 2022 18:17:41 GMT
View Forum Message <> Reply to Message

Hi Mirek,

Please start typing a floating point constant (e.g. 1.) .. and the code completion starts to look for something to add after the point. Can this behavior be removed?

Best regards,

Tom

Subject: Re: 2022(?).2 beta
Posted by mirek on Sat, 10 Dec 2022 22:39:52 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Sat, 10 December 2022 19:17Hi Mirek,

Please start typing a floating point constant (e.g. 1.) .. and the code completion starts to look for something to add after the point. Can this behavior be removed?

Best regards,

Tom

Thanks, hopefully fixed.

Subject: Re: 2022(?).2 beta
Posted by Tom1 on Sun, 11 Dec 2022 18:03:54 GMT
View Forum Message <> Reply to Message

Thanks, Mirek!

Can you fix these warnings for MSBTx64 too:
C:\upp-git\upp.src\uppsrc\Core\Other.h(123): warning C4267: 'argument': conversion from 'size_t' to 'int', possible loss of data
C:\upp-git\upp.src\uppsrc\Core\Other.h(143): warning C4267: 'argument': conversion from 'size_t' to 'int', possible loss of data
C:\upp-git\upp.src\uppsrc\CtrlLib/DropChoice.h(83): warning C4099: 'Upp::PopUpList::Popup': type name first seen using 'struct' now seen using 'class'
C:\upp-git\upp.src\uppsrc\CtrlLib/DropChoice.h(54): note: see declaration of 'Upp::PopUpList::Popup'
C:\upp-git\upp.src\uppsrc\CtrlCore\ImageWin32.cpp(233): warning C4267: 'argument': conversion from 'size_t' to 'int', possible loss of data
C:\upp-git\upp.src\uppsrc\CtrlCore\ImageWin32.cpp(263): warning C4267: 'argument': conversion from 'size_t' to 'int', possible loss of data
C:\upp-git\upp.src\uppsrc\CtrlCore\ImageWin32.cpp(297): warning C4267: 'return': conversion from 'size_t' to 'int', possible loss of data
C:\upp-git\upp.src\uppsrc\CtrlCore\ImageWin32.cpp(323): warning C4267: '+=': conversion from 'size_t' to 'int', possible loss of data
C:\upp-git\upp.src\uppsrc\CtrlCore\CtrlAttr.cpp(137): warning C4244: 'initializing': conversion from 'Upp::int64' to 'Upp::dword', possible loss of data
C:\upp-git\upp.src\uppsrc\CtrlCore\CtrlAttr.cpp(161): warning C4244: 'return': conversion from 'Upp::int64' to 'int', possible loss of data
C:\upp-git\upp.src\uppsrc\CtrlCore\CtrlAttr.cpp(162): warning C4244: 'return': conversion from 'Upp::int64' to 'int', possible loss of data
C:\upp-git\upp.src\uppsrc\CtrlCore\CtrlDraw.cpp(282): warning C4101: 'q': unreferenced local variable
C:\upp-git\upp.src\uppsrc\CtrlCore\Win32Clip.cpp(412): warning C4267: 'argument': conversion from 'size_t' to 'int', possible loss of data
C:\upp-git\upp.src\uppsrc\Core\Mt.cpp(153): warning C4267: 'argument': conversion from 'size_t' to 'unsigned int', possible loss of data
C:\upp-git\upp.src\uppsrc\Core\Stream.cpp(237): warning C4244: 'initializing': conversion from 'Upp::int64' to 'int', possible loss of data

(BTW: I guess it should be 2022.3 or 2023.1.)

Best regards,

Tom

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 11 Dec 2022 18:45:00 GMT
View Forum Message <> Reply to Message

Code formatting is not working (properly). Just formatting <CtrlCore/CtrlCore.h> for a demonstration. All bitfield's are formatted in a strange, unpleasant way. And there are more problems than that. After all, the code is quite old.

In the same spirit that U++ embraces libclang, it
's desirable to move code formatting to utilize LibFormat (part of the llvm-project). It should be quite easy for Mirek with his experience integrating libclang.

Subject: Re: 2022(?).2 beta
Posted by mirek on Sun, 11 Dec 2022 19:02:26 GMT
View Forum Message <> Reply to Message

Lance wrote on Sun, 11 December 2022 19:45Code formatting is not working (properly). Just formatting <CtrlCore/CtrlCore.h> for a demonstration. All bitfield's are formatted in a strange, unpleasant way. And there are more problems than that. After all, the code is quite old.

In the same spirit that U++ embraces libclang, it
's desirable to move code formatting to utilize LibFormat (part of the llvm-project). It should be quite easy for Mirek with his experience integrating libclang.

Do you mean AStyle formattig?

Well, that one I think is broken for years. Code formatting seems to be a simple discipline, probably much easier to do it myself.

LibFormat is probably too hard pill to swallow right now.

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 11 Dec 2022 20:13:32 GMT
View Forum Message <> Reply to Message

ok. We don't really need that much flexibility as offered by LibFormat; just formatting code to u++ preferred style is satisfactory. This way code will look more consistent. The AStyle utility that's currently used is very broken :(

Subject: Re: 2022(?).2 beta
Posted by mirek on Sun, 11 Dec 2022 21:54:52 GMT
View Forum Message <> Reply to Message

Lance wrote on Sun, 11 December 2022 21:13ok. We don't really need that much flexibility as offered by LibFormat; just formatting code to u++ preferred style is satisfactory. This way code will look more consistent. The AStyle utility that's currently used is very broken :(

I can remove it completely for 2022.2, but not replace...

Subject: Re: 2022(?).2 beta
Posted by Klugier on Sun, 11 Dec 2022 22:08:15 GMT
View Forum Message <> Reply to Message

Hello Mirek and Lance,

I think we should remove AStyle completely from the TheIDE. The true thing is that there is no true maintainer of that code and it was not updated for years. However, I think we should do it when we will have alternative.

As a replacement to AStyle we should go with clang-format executable. It is bundle with clang tool-chain. So, if you have clang you have clang-format. In the context of utilizing clang-format, we should create our own style basing on the current U++ source files style and make it default. Moreover, user should be able to provide it's own style for the package by adding/defining .clang-format file. So, we will have customization like we currently have. However, it will be moved from UI to text. We could also add some features like reformat code on save etc.

If I will have more time in the next year, I can look at this topic. It's shouldn't be hard to implement.

Klugier

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 11 Dec 2022 23:47:11 GMT
View Forum Message <> Reply to Message

Hello Klugier:

That will be the fastest and safest route (which is taken by many other code editors). I don't see we lose anything, comparing to calling LibFormat directly. I think it's preferable to writing a home-grown formatter as c++ language is changing, albeit slowly. Shifting the duty of keeping up with new standards to a trustworthy, resource-rich third party could save us a lot of effort in the long run.

BR,
Lance

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 11 Dec 2022 23:52:34 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 11 December 2022 16:54Lance wrote on Sun, 11 December 2022 21:13ok. We don't really need that much flexibility as offered by LibFormat; just formatting code to u++ preferred style is satisfactory. This way code will look more consistent. The AStyle utility that's currently used is very broken :(

I can remove it completely for 2022.2, but not replace...

Yes, time is too tight. It should be removed as it serves no practical purposes in its current state, other than messing up code.

Subject: Re: 2022(?).2 beta
Posted by mirek on Mon, 12 Dec 2022 09:44:49 GMT
View Forum Message <> Reply to Message

Klugier wrote on Sun, 11 December 2022 23:08Hello Mirek and Lance,

I think we should remove AStyle completely from the TheIDE. The true thing is that there is no true maintainer of that code and it was not updated for years. However, I think we should do it when we will have alternative.

As a replacement to AStyle we should go with clang-format executable. It is bundle with clang tool-chain.

Quick check: I do not see it in toolchain we are using in Win32.... :(

Subject: Re: 2022(?).2 beta
Posted by zsolt on Mon, 12 Dec 2022 11:42:54 GMT
View Forum Message <> Reply to Message

I have found an interesting bug.
The new assist complained about a lot of weird bugs in my code.
I'm using some external libraries, so I created a new .bm file and used it for compiling.
But it seemed to me, that Assist isn't working based on that .bm file, so I removed all the other .bm files, and renamed my custom one to CLANG.bm.
Everything seems to be working well now.

Subject: Re: 2022(?).2 beta
Posted by mirek on Mon, 12 Dec 2022 11:59:12 GMT

View Forum Message <> Reply to Message

zsolt wrote on Mon, 12 December 2022 12:42I have found an interesting bug.
The new assist complained about a lot of weird bugs in my code.
I'm using some external libraries, so I created a new .bm file and used it for compiling.
But it seemed to me, that Assist isn't working based on that .bm file, so I removed all the other .bm files, and renamed my custom one to CLANG.bm.
Everything seems to be working well now.

Currently it is always using include paths from CLANG.bm method. There does not seem an easy solution unfortunately as other includes can be incompatible. But I guess adding include paths from current build method at the end of the list should work (will do ASAP).

Subject: Re: 2022(?).2 beta
Posted by zsolt on Mon, 12 Dec 2022 12:24:40 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 12 December 2022 12:59
Currently it is always using include paths from CLANG.bm method. There does not seem an easy solution unfortunately as other includes can be incompatible. But I guess adding include paths from current build method at the end of the list should work (will do ASAP).

This seems to me a good idea.
I can not imagine any reason to use an other compiler, than CLANG. I already converted my projects to use that one. Much better toolchain than anything other. Btw, I use MSYS2 and it's Clang toolchain on Windows. The same feeling, as coding on Linux.
And thanks for the tooltips in IDE, showing when holding the mouse pointer over a symbol while editing the source. Extremely useful.

Subject: Re: 2022(?).2 beta
Posted by mirek on Mon, 12 Dec 2022 21:17:36 GMT
View Forum Message <> Reply to Message

zsolt wrote on Mon, 12 December 2022 12:42I have found an interesting bug.
The new assist complained about a lot of weird bugs in my code.
I'm using some external libraries, so I created a new .bm file and used it for compiling.
But it seemed to me, that Assist isn't working based on that .bm file, so I removed all the other .bm files, and renamed my custom one to CLANG.bm.
Everything seems to be working well now.

I am now adding "real" build method's include paths after CLANG's ones. Hopefully this might fix this issue...

Subject: Re: 2022(?).2 beta

Posted by mirek on Mon, 12 Dec 2022 21:18:13 GMT
View Forum Message <> Reply to Message

Lance wrote on Mon, 12 December 2022 00:52mirek wrote on Sun, 11 December 2022 16:54Lance wrote on Sun, 11 December 2022 21:13ok. We don't really need that much flexibility as offered by LibFormat; just formatting code to u++ preferred style is satisfactory. This way code will look more consistent. The AStyle utility that's currently used is very broken :(

I can remove it completely for 2022.2, but not replace...

Yes, time is too tight. It should be removed as it serves no practical purposes in its current state, other than messing up code.

AStyle removed.

Subject: Re: 2022(?).2 beta
Posted by Lance on Tue, 13 Dec 2022 02:19:03 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 12 December 2022 04:44Klugier wrote on Sun, 11 December 2022 23:08Hello Mirek and Lance,

I think we should remove AStyle completely from the TheIDE. The true thing is that there is no true maintainer of that code and it was not updated for years. However, I think we should do it when we will have alternative.

As a replacement to AStyle we should go with clang-format executable. It is bundle with clang tool-chain.

Quick check: I do not see it in toolchain we are using in Win32.... :(

I need to install some package before I can run clang-format on ubuntu linux.

Subject: Re: 2022(?).2 beta
Posted by Lance on Tue, 13 Dec 2022 02:21:16 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 12 December 2022 16:18Lance wrote on Mon, 12 December 2022 00:52mirek wrote on Sun, 11 December 2022 16:54Lance wrote on Sun, 11 December 2022 21:13ok. We don't really need that much flexibility as offered by LibFormat; just formatting code to u++ preferred style is satisfactory. This way code will look more consistent. The AStyle utility that's currently used is very broken :(

I can remove it completely for 2022.2, but not replace...

Yes, time is too tight. It should be removed as it serves no practical purposes in its current state, other than messing up code.

AStyle removed.

Thanks!

Subject: Re: 2022(?).2 beta
Posted by Tom1 on Tue, 13 Dec 2022 09:25:29 GMT
View Forum Message <> Reply to Message

Hi,

It seems StaticText has changed. Where is:
StaticText::GetFont();
Or how can I get this done now?

Best regards,

Tom

Subject: Re: 2022(?).2 beta
Posted by mirek on Tue, 13 Dec 2022 09:45:34 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Tue, 13 December 2022 10:25Hi,

It seems StaticText has changed. Where is:
StaticText::GetFont();
Or how can I get this done now?

Best regards,

Tom

Sorry, that was gone June during the "sizeof(widget)" campaign. I have now put GetXXX methods to StaticText.

Please check!

Mirek

Subject: Re: 2022(?).2 beta
Posted by Tom1 on Tue, 13 Dec 2022 09:56:47 GMT
View Forum Message <> Reply to Message

Mirek,

Thanks, it works now. (Also thanks for fixing the warnings with MSBTx64.)

Just noticed that the left stripe in code editor no longer shows errors, just recently edited code.
(I'm on Windows if that has any significance...)

Best regards,

Tom

Subject: Re: 2022(?).2 beta
Posted by mirek on Tue, 13 Dec 2022 10:08:22 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Tue, 13 December 2022 10:56Mirek,

Thanks, it works now. (Also thanks for fixing the warnings with MSBTx64.)

Just noticed that the left stripe in code editor no longer shows errors, just recently edited code.
(I'm on Windows if that has any significance...)

Best regards,

Tom

Yes, that is by design. Errors are now shown in the text, while you are typing and also in scrollbar.

Subject: Re: 2022(?).2 beta
Posted by Tom1 on Tue, 13 Dec 2022 10:42:20 GMT
View Forum Message <> Reply to Message

OK, I see. I turned back on the 'Show errors in the current file...' option. It seems less intrusive now. Maybe I will learn to like it.

Unfortunately, it seems to erroneously flag my structure packing pragma:
#pragma pack(push,1)
It complains about "Unterminated #pragma pack(push,...) at end of file". It seems to ignore:
#pragma pack(pop)

following the packed structure(s).

Best regards,

Tom

Subject: Re: 2022(?).2 beta
Posted by mirek on Tue, 13 Dec 2022 10:57:02 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Tue, 13 December 2022 11:42OK, I see. I turned back on the 'Show errors in the current file...' option. It seems less intrusive now. Maybe I will learn to like it.

Unfortunately, it seems to erroneously flag my structure packing pragma:
#pragma pack(push,1)
It complains about "Unterminated #pragma pack(push,...) at end of file". It seems to ignore:
#pragma pack(pop)
following the packed structure(s).

Best regards,

Tom

libclang is not perfect... I guess we just need to take what it gives (which IMO is a lot) and ignore quirks...

Mirek

Subject: Re: 2022(?).2 beta
Posted by mirek on Tue, 13 Dec 2022 10:58:35 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Tue, 13 December 2022 11:42OK, I see. I turned back on the 'Show errors in the current file...' option. It seems less intrusive now. Maybe I will learn to like it.

Unfortunately, it seems to erroneously flag my structure packing pragma:
#pragma pack(push,1)
It complains about "Unterminated #pragma pack(push,...) at end of file". It seems to ignore:
#pragma pack(pop)
following the packed structure(s).

Best regards,

Tom

That said, maybe if it is a warning, you could find some commandline option to switch it off? You can specify that in Setup/Assist commandline... (and if you tell me it, I will add it as default).

Subject: Re: 2022(?).2 beta
Posted by mirek on Tue, 13 Dec 2022 12:21:33 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Tue, 13 December 2022 11:42OK, I see. I turned back on the 'Show errors in the current file...' option. It seems less intrusive now. Maybe I will learn to like it.

Unfortunately, it seems to erroneously flag my structure packing pragma:
#pragma pack(push,1)
It complains about "Unterminated #pragma pack(push,...) at end of file". It seems to ignore:
#pragma pack(pop)
following the packed structure(s).

Best regards,

Tom

-Wno-pragma-pack seems to do the job. It is now default (but you will need to add it to Assist setup. Note that unless you update theide (one more bug fixed), it needs restart).

Thanks for testing. At this time very helpful. I still want to release 2022.2 (not 2023.1) :)

Subject: Re: 2022(?).2 beta
Posted by Tom1 on Tue, 13 Dec 2022 12:30:00 GMT
View Forum Message <> Reply to Message

mirek wrote on Tue, 13 December 2022 14:21Tom1 wrote on Tue, 13 December 2022 11:42OK, I see. I turned back on the 'Show errors in the current file...' option. It seems less intrusive now. Maybe I will learn to like it.

Unfortunately, it seems to erroneously flag my structure packing pragma:
#pragma pack(push,1)
It complains about "Unterminated #pragma pack(push,...) at end of file". It seems to ignore:
#pragma pack(pop)
following the packed structure(s).

Best regards,

Tom

-Wno-pragma-pack seems to do the job. It is now default (but you will need to add it to Assist setup. Note that unless you update theide (one more bug fixed), it needs restart).

Thanks for testing. At this time very helpful. I still want to release 2022.2 (not 2023.1) :)
Thanks Mirek,

It works now! (I could not do it here with -Wno-pragma-pack, but your fix worked.)

Please note though that 2022.2 has been out for quite some time, so this must be 2022.3. (The U++ front page says 2022.2 (rev. 16270)).

BR, Tom

Subject: Re: 2022(?).2 beta
Posted by mirek on Tue, 13 Dec 2022 13:40:46 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Tue, 13 December 2022 13:30
Please note though that 2022.2 has been out for quite some time, so this must be 2022.3. (The U++ front page says 2022.2 (rev. 16270)).

BR, Tom

Good point, thanks! :)

Mirek

Subject: Re: 2022(?).2 beta
Posted by zsolt on Tue, 13 Dec 2022 16:33:15 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 12 December 2022 22:17zsolt wrote on Mon, 12 December 2022 12:42I have found an interesting bug.
The new assist complained about a lot of weird bugs in my code.
I'm using some external libraries, so I created a new .bm file and used it for compiling.
But it seemed to me, that Assist isn't working based on that .bm file, so I removed all the other .bm files, and renamed my custom one to CLANG.bm.
Everything seems to be working well now.

I am now adding "real" build method's include paths after CLANG's ones. Hopefully this might fix this issue...

Seems to be working now, thanks.

Subject: Re: 2022(?).2 beta

Posted by zsolt on Tue, 13 Dec 2022 16:39:21 GMT
View Forum Message <> Reply to Message

An other bug:
Assist goes mad on .icpp files. Try opening Painter/PainterInit.icpp
It is just trying to parse it without success endlessly:

C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp parsed in 15 ms
C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp parser output processed in 0 ms
Failed commandline: C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp -DflagDEBUG
-DflagDEBUG_FULL -DflagMAIN -DflagCLANG -std=c++14 -xc++ -Wno-logical-op-parentheses
C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp parsed in 0 ms
C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp parser output processed in 0 ms
Failed commandline: C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp -DflagDEBUG
-DflagDEBUG_FULL -DflagMAIN -DflagCLANG -std=c++14 -xc++ -Wno-logical-op-parentheses

Subject: Re: 2022(?).2 beta
Posted by mirek on Tue, 13 Dec 2022 16:49:11 GMT
View Forum Message <> Reply to Message

zsolt wrote on Tue, 13 December 2022 17:39An other bug:
Assist goes mad on .icpp files. Try opening Painter/PainterInit.icpp
It is just trying to parse it without success endlessly:

C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp parsed in 15 ms
C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp parser output processed in 0 ms
Failed commandline: C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp -DflagDEBUG
-DflagDEBUG_FULL -DflagMAIN -DflagCLANG -std=c++14 -xc++ -Wno-logical-op-parentheses
C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp parsed in 0 ms
C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp parser output processed in 0 ms
Failed commandline: C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp -DflagDEBUG
-DflagDEBUG_FULL -DflagMAIN -DflagCLANG -std=c++14 -xc++ -Wno-logical-op-parentheses

Accidentally, I have noticed and file into internal TODO already this morning. I might try to fix this later tonight, but I am OK with releasing with this problem.

Subject: Re: 2022(?).2 beta
Posted by zsolt on Tue, 13 Dec 2022 17:17:00 GMT
View Forum Message <> Reply to Message

Thanks.
Sometimes Assist ended up in parsing them endlessly, so I started to rename .icpp files to .cpp. As I can see, they are not needed anymore. I used them for registering translations and unit tests, but now they are working in .cpp files as well.

Subject: Re: 2022(?).2 beta
Posted by mirek on Tue, 13 Dec 2022 17:32:34 GMT
View Forum Message <> Reply to Message

zsolt wrote on Tue, 13 December 2022 18:17Thanks.
Sometimes Assist ended up in parsing them endlessly, so I started to rename .icpp files to .cpp. As I can see, they are not needed anymore. I used them for registering translations and unit tests, but now they are working in .cpp files as well.

Careful here:

The purpose of .icpp was that it never went into .lib file during the build process, always are linked as .obj. This has the effect that it is always linked; files in .lib can be ignored if there are no references to its contents from other files.

In debug, theide builder does not bother creating .libs. So what you do can work in debug and fail in release.

Current U++ really dropped .icpp use (but still supports them when building), replaced with INITIALIZE and INITIALIZER macros (which basically create the reference to initializer by including the file). It is not ideal either, but makes U++ tiny bit more standard.

Subject: Re: 2022(?).2 beta
Posted by zsolt on Tue, 13 Dec 2022 18:01:02 GMT
View Forum Message <> Reply to Message

mirek wrote on Tue, 13 December 2022 18:32zsolt wrote on Tue, 13 December 2022 18:17Thanks.
Sometimes Assist ended up in parsing them endlessly, so I started to rename .icpp files to .cpp. As I can see, they are not needed anymore. I used them for registering translations and unit tests, but now they are working in .cpp files as well.

Careful here:

The purpose of .icpp was that it never went into .lib file during the build process, always are linked as .obj. This has the effect that it is always linked; files in .lib can be ignored if there are no references to its contents from other files.

In debug, theide builder does not bother creating .libs. So what you do can work in debug and fail in release.

Current U++ really dropped .icpp use (but still supports them when building), replaced with INITIALIZE and INITIALIZER macros (which basically create the reference to initializer by including the file). It is not ideal either, but makes U++ tiny bit more standard.

Thanks, I have just checked these macros, but I think, it would be a nightmare to use them for every unit tests I created.
But I turned on Blitz for compiling release builds and it seems to me, that it doesn't generate .libs. Is that true?

Subject: Re: 2022(?).2 beta
Posted by mirek on Tue, 13 Dec 2022 18:12:31 GMT
View Forum Message <> Reply to Message

zsolt wrote on Tue, 13 December 2022 19:01mirek wrote on Tue, 13 December 2022 18:32zsolt wrote on Tue, 13 December 2022 18:17Thanks.
Sometimes Assist ended up in parsing them endlessly, so I started to rename .icpp files to .cpp. As I can see, they are not needed anymore. I used them for registering translations and unit tests, but now they are working in .cpp files as well.

Careful here:

The purpose of .icpp was that it never went into .lib file during the build process, always are linked as .obj. This has the effect that it is always linked; files in .lib can be ignored if there are no references to its contents from other files.

In debug, theide builder does not bother creating .libs. So what you do can work in debug and fail in release.

Current U++ really dropped .icpp use (but still supports them when building), replaced with INITIALIZE and INITIALIZER macros (which basically create the reference to initializer by including the file). It is not ideal either, but makes U++ tiny bit more standard.

Thanks, I have just checked these macros, but I think, it would be a nightmare to use them for every unit tests I created.
But I turned on Blitz for compiling release builds and it seems to me, that it doesn't generate .libs. Is that true?

Yes. But I cannot guarantee it will not change in the future.

Subject: Re: 2022(?).2 beta
Posted by zsolt on Tue, 13 Dec 2022 19:01:50 GMT
View Forum Message <> Reply to Message

OK, I hope, I will notice that :)

Subject: Re: 2022(?).2 beta
Posted by pvictor on Wed, 14 Dec 2022 11:56:48 GMT
View Forum Message <> Reply to Message

Hi,

theide.log gets stuffed with hundreds of identical strings:

11:46:54:638 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:46:54:891 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:46:55:318 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:46:57:764 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:46:57:765 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:47:00:340 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:47:00:342 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:47:03:259 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:47:03:318 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.

Best regards,
Victor

Subject: Re: 2022(?).2 beta
Posted by mirek on Wed, 14 Dec 2022 13:04:02 GMT
View Forum Message <> Reply to Message

pvictor wrote on Wed, 14 December 2022 12:56Hi,

theide.log gets stuffed with hundreds of identical strings:

11:46:54:638 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:46:54:891 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:46:55:318 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:46:57:764 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:46:57:765 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:47:00:340 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:47:00:342 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:47:03:259 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.
11:47:03:318 ERROR MakeBuild::IsAndroidMethod(): Failed to find builder.

Best regards,
Victor

Logs hopefully removed.

Subject: Re: 2022(?).2 beta
Posted by mirek on Wed, 14 Dec 2022 13:05:15 GMT
View Forum Message <> Reply to Message

zsolt wrote on Tue, 13 December 2022 17:39An other bug:
Assist goes mad on .icpp files. Try opening Painter/PainterInit.icpp
It is just trying to parse it without success endlessly:

C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp parsed in 15 ms
C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp parser output processed in 0 ms
Failed commandline: C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp -DflagDEBUG
-DflagDEBUG_FULL -DflagMAIN -DflagCLANG -std=c++14 -xc++ -Wno-logical-op-parentheses
C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp parsed in 0 ms
C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp parser output processed in 0 ms
Failed commandline: C:\Download\Upp\upp\uppsrc\Painter\PainterInit.icpp -DflagDEBUG
-DflagDEBUG_FULL -DflagMAIN -DflagCLANG -std=c++14 -xc++ -Wno-logical-op-parentheses

.icpp assist should be now fixed.

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 18 Dec 2022 04:07:33 GMT
View Forum Message <> Reply to Message

Hi Mirek:

I understand that U++ currently aims to be compliant with c++14. Overall, U++ is very close to c++20 compliant.

IIRC, the only kind of complaints gcc/clang make when building TheIDE with std=c++20 is about caputuring `this` by default is deprecated in c++20 for a lambda. I am not sure if changing affected code to get rid of all such warnings will affect c++14 compliance but it's easy to make both worlds(or maybe all 3 worlds if we want to refer to c++17 and c++20 separately) happy anyways.

With MSVC, it is a different story. It complains in many cases like

```
return some_condition ? SomeString : "Some ASCIIZ String";
```

These, though tedious, are easy to fix. I am no language lawyer, cannot tell which of MSVC and GCC/CLANG is/are correct here. But MSVC changes its behaviour from accepting it in C++14 to rejecting it in C++17 and beyond may tell something. Anyway it's not hard to make all worlds happy by just a little bit more keystrokes.

There are some more errors when compiling TheIDE with MSVC (mine is MSBT 2019 I believe) and std set to C++20. Another one is caused by Upp::Moveable::AssertMovealbe0 or something like that.

I mainly use CLANG now but I feel more assured if my code compiles fine on Ubuntu with GCC & CLANG, and on Windows with MS c++ compiler. I don't know if other users think this kind of check has some value, but it likely will be welcomed if a user can use u++ with more recent standard if he/she wishes(so that he/she can embrace utilities like constexpr-if and concept), and with the compiler he/she choose (one of the 3 major), while the bulk of U++ is in c++14 and be backward compatible.

Correction and some detailed error message:
1. The MSVC I used is MSBT22x64
2. The error message with AssertMoveable0 is like
Quote:
C:\upp\uppsrc\Core\Topt.h (157): error C2100: illegal indirection
C:\upp\uppsrc\Core\Topt.h (172): note: see reference to function template instantiation 'void Upp::AssertMoveable0<T>(T *)' being compiled
with
[
T=double
]ChWin32.cpp

BR,
Lance

Subject: Re: 2022(?).2 beta
Posted by mirek on Sun, 18 Dec 2022 12:26:35 GMT
View Forum Message <> Reply to Message

Lance wrote on Sun, 18 December 2022 05:07Hi Mirek:

I understand that U++ currently aims to be compliant with c++14. Overall, U++ is very close to c++20 compliant.

IIRC, the only kind of complaints gcc/clang make when building TheIDE with std=c++20 is about caputuring `this` by default is deprecated in c++20 for a lambda.

You can have one or another. It looks like

[=, this] {}; // until C++20: Error: this when = is the default

which supercomplicates the stuff everywhere unfortunately. So this change will have to wait a couple of years it seems.

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 18 Dec 2022 13:48:42 GMT
View Forum Message <> Reply to Message

Hi Mirek:

Thank you for your attention to this matter.

The following macro will perfectly pacify both GCC and CLANG.

```
#if __cplusplus > 201703L
#    define CAPTURETHISBYVALUE ,this
#else
#    define CAPTURETHISBYVALUE
#endif
```

And when using it

```
void ColorWindow::Paint(Draw& draw)
{
    auto f = [= CAPTURETHISBYVALUE]{ auto v = GetData(); };
    draw.DrawRect(GetSize(), White());
    auto v = f();
        ...
}
```

It should be easy to add support for MSVC similarly too.

This way, we only care that the U++ library can compiles with std=c++14, std=c++17, std=c++20 or later std. Whether a u++ library user decide to follow the practice so that his/her code is also multiple c++ standards compatible, or simply choose one of the standard to embrace, is not a concern of u++ library developers (like you and Klugier).

The philosophy here is: u++ can choose a stable and well supported c++ standard to embrace, but it should not limit or discourage its users from trying later standard. IMHO, comparing to package-wise c++ standard selection options(it will certainly confuse assist++ if at all doable), this kind of fix in the U++ library level is less painful.

BR,
Lance

PS:

Or probably even easier.

```
#if __cplusplus > 201703L
```

```
#    define CAPBYVALUETHIS =,this
#else
#    define CAPBYVALUETHIS =
#endif
```

Then do a find in files and replace from uppsrc root, it's almost done. I figure there are <=2 occassioins where [=] are not within a member function thus [=,this] is invalid which need be fixed after the find and replace.

Subject: Re: 2022(?).2 beta
Posted by Klugier on Sun, 18 Dec 2022 14:41:51 GMT
View Forum Message <> Reply to Message

Hello Lance,

Quick question, what about replacing [=] with [this]. Does it produce warning with C++20? It compiles fine with C++14 and I think in most cases we can replace it. We are using [=] to capture local variables very really, however this can be overcome by explicit argument capture.

The solution with CAPTURETHISBYVALUE is ugly and very impractical especially for U++ maintainers :)

I agree that we should be compatible with C++20 as much as we can. On the other hand, I also think it is a good moment to change default standard from c++14 to c++17. c++14 is 8/9 years old standard and on most of currently supported system compilers with c++17 support are present. For example I like auto [x, error] = GetTuple() that can not be used in c++14 word. Very useful feature when you want to back-propagate error without using exceptions. I remember, we discussed this transition some time ago, but maybe it is good to discuss it one more time :)

Klugier

Subject: Re: 2022(?).2 beta
Posted by zsolt on Sun, 18 Dec 2022 15:08:51 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 14 December 2022 14:05
.icpp assist should be now fixed.
Thank you!

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 18 Dec 2022 17:50:54 GMT

Klugier wrote on Sun, 18 December 2022 09:41Hello Lance,

Quick question, what about replacing [=] with [this]. Does it produce warning with C++20? It compiles fine with C++14 and I think in most cases we can replace it. We are using [=] to capture local variables very really, however this can be overcome by explicit argument capture.

The solution with CAPTURETHISBYVALUE is ugly and very impractical especially for U++ maintainers :)

I agree that we should be compatible with C++20 as much as we can. On the other hand, I also think it is a good moment to change default standard from c++14 to c++17. c++14 is 8/9 years old standard and on most of currently supported system compilers with c++17 support are present. For example I like auto [x, error] = GetTuple() that can not be used in c++14 word. Very useful feature when you want to back-propagate error without using exceptions. I remember, we discussed this transition some time ago, but maybe it is good to discuss it one more time :)

Klugier

Hi Klugier:

Unfortunately it doesn't work. In a lot of occassions [=] captures more than just [this].

I will try my approach and report the result.

Regards,
Lance

PS:
It works perfectly in GCC/CLANG on Ubuntu.

Here is what I did:
1. Insert the definition of CAP_BY_VALUE_WITH_THIS after #include <utility>

```
#if __cplusplus > 201703L
#    define CAP_BY_VALUE_WITH_THIS = ,this
#else
#    define CAP_BY_VALUE_WITH_THIS =
#endif
```

2. Do a Replace in Files..., replace all occurrences of [=] with CAP_BY_VALUE_WITH_THIS, for all files under the folder uppsrc;

3. Compile some examples, eg. <Examples/Color>, fix all errors due to wrongful replacement in step 2. I believe there are 3-4 such occurrences.

4. Compile TheIDE, 2 more errors pop up. In <ide/Builders/Install.cpp>, replace

```
 auto Fix = [CAP_BY_VALUE_WITH_THIS](const char *s) {
```

with

```
 auto Fix = [=](const char *s) {
```

And in <ide/Designers/HexView.cpp>, Ln 98, replace

```
 RegisterGlobalSerialize("FileHexViewPos", [CAP_BY_VALUE_WITH_THIS](Stream& s) {
```

with

```
 RegisterGlobalSerialize("FileHexViewPos", [=](Stream& s) {
```

and ide is ready to be built. Of course I fix problems as GCC complains to me. As I did nothing in between, any errors that stopped compiling would mean I need to replace [CAP_BY_VALUE_WITH_THIS] with [=] (where there is no `this` at all).

BR,
Lance

Subject: Re: 2022(?).2 beta
Posted by mr_ped on Sun, 18 Dec 2022 18:46:10 GMT
View Forum Message <> Reply to Message

@Mirek: BTW, how about IDE: closing window moves it in Ctrl+Tab order after all opened documents.

I think we did discuss this few years back, and it was like by-design for you? But it keeps irritate me even after those years: I switch by Ctrl+Tab to file which I want to close, Ctrl+W to close it, and now I want to move to some other open file and hit Ctrl+Tab, and closed file is back... I'm just not used to this, and I don't see value in it, feels like wrong UX to me.

I can try to do pull request if you don't mind the change.

BTW you seem to mostly apply pull requests manually, but that way you make the contribution in git changed to your user, not sure if this is intentional, or just the way how you use git.

And one more very minor IDE bug/quirk: when I Ctrl+M to select main package, open one, it re-opens all tabs which were open during working on that package = nice. But current state seems to be saved only when leaving IDE, not when I do another Ctrl+M and switch to other package. That's IMHO unexpected UX, if the state loads upon Ctrl+M, it should also save updated state of open tabs on it?

(sorry if these requests are out of scope for 2022.3, but as I'm lately using IDE daily, I'm pressing the stuff which nags me ;))

Subject: Re: 2022(?).2 beta
Posted by mirek on Sun, 18 Dec 2022 18:53:27 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Sun, 18 December 2022 19:46@Mirek: BTW, how about IDE: closing window moves it in Ctrl+Tab order after all opened documents.

I think we did discuss this few years back, and it was like by-design for you? But it keeps irritate me even after those years: I switch by Ctrl+Tab to file which I want to close, Ctrl+W to close it, and now I want to move to some other open file and hit Ctrl+Tab, and closed file is back... I'm just not used to this, and I don't see value in it, feels like wrong UX to me.

I can try to do pull request if you don't mind the change.

BTW you seem to mostly apply pull requests manually, but that way you make the contribution in git changed to your user, not sure if this is intentional, or just the way how you use git.

And one more very minor IDE bug/quirk: when I Ctrl+M to select main package, open one, it re-opens all tabs which were open during working on that package = nice. But current state seems to be saved only when leaving IDE, not when I do another Ctrl+M and switch to other package. That's IMHO unexpected UX, if the state loads upon Ctrl+M, it should also save updated state of open tabs on it?

(sorry if these requests are out of scope for 2022.3, but as I'm lately using IDE daily, I'm pressing the stuff which nags me ;))

OK, if rc1 fails, I will look into this :)

Subject: Re: 2022(?).2 beta
Posted by mirek on Sun, 18 Dec 2022 18:55:21 GMT
View Forum Message <> Reply to Message

Lance wrote on Sun, 18 December 2022 14:48

```
#if __cplusplus > 201703L
#    define CAPTURETHISBYVALUE ,this
#else
#    define CAPTURETHISBYVALUE
#endif
```

It does not feel nicer than disabling the warning...

Mirek

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 18 Dec 2022 19:08:24 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 18 December 2022 13:55Lance wrote on Sun, 18 December 2022 14:48

```
#if __cplusplus > 201703L
#    define CAPTURETHISBYVALUE ,this
#else
#    define CAPTURETHISBYVALUE
#endif
```

It does not feel nicer than disabling the warning...

Mirek

Hi Mirek:

It doesn't. But this technique is widely used in standard library implementations. It's a necessary evil (if not in this particular situation where you can opt to turn off warnings).

For example, many std::vector member functions are now constexpr modified, what's a better way to provide cross-c++-standard support in this situation? I am not sure if there is one, but the one in practice is a MACRO that will vanish in a lower language/library standard.

BTW, there is no guarantee it will not be promoted to an error in some later days when U++ probably is in c++17. So chance is the problem is merely deferred instead of being solved by disabling warnings on it.

BR,
Lance

Subject: Re: 2022(?).2 beta
Posted by Klugier on Sun, 18 Dec 2022 20:54:34 GMT
View Forum Message <> Reply to Message

Hello Mirek and Lance,

I agree with Lance that someday capture this by [=] might be compilation error instead of warning. Such precedence happened in the past. For example in C++17 std::auto_ptr had been removed. However for capturing this by [=], I do not see that it will be removed in C++23. So, disabling warning might give us 3-4 additional years...

In accordance to table on nextptr article the only valid capture this that works with all standards is capture this as [this] and [&]:

As, in my previous post, we should follow that approach. It won't be easy, but doable for our code base within 1-2 days. Whenever, we need to pass additional variable it should be pass explicitly [this, x] etc.. Alternatively, we can convert [=] to [&] as it is valid too. However, it might caused some unwanted bugs...

Using global macro is not an option to me as I wrote in my previous post.

Klugier

File Attachments

1) Table.png, downloaded 428 times

Subject: Re: 2022(?).2 beta
Posted by zsolt on Sun, 18 Dec 2022 21:33:18 GMT
View Forum Message <> Reply to Message

This is a very good table, thanks.
I think, [=, *this] could be the most easy way to solve the [=] problem. And dropping C++-14 compiler support (if I understand the topic).

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 18 Dec 2022 21:35:30 GMT
View Forum Message <> Reply to Message

Hello Klugier,

I can confirm that all U++ currently uses are [=].

Out of which, some(I expect it to be a total of less than 10) need to remain as [=] prior or beyond c++20; while the majority rest can be changed to [=,this] to make the code compliant to c++20 (which, unfortunately will displease prior c++20 world).

I cannot really tell how many out of the second lot can be replaced by [this] without creating noise when they need more than just `this`, eg, also capturing some local variables, etc.

Do you mean to differentiate from the second lot the ones that actually require some other variables, and list each of them manually so that prior and beyond c++20 worlds will be happy

with their capture lists?

It certainly is doable, but it might be a bit too much effort, IMHO, just for the sake of avoiding an unwanted MACRO.

Otherwise the quickest & dirtiest solution is to change all [=] to [&], with possibly undesired side-effects.

It's totally up to you and Mirek though. We will be happy with a standard-tolerant U++ library however achieved :)

BR,
Lance

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 18 Dec 2022 21:55:30 GMT
View Forum Message <> Reply to Message

zsolt wrote on Sun, 18 December 2022 16:33This is a very good table, thanks.
I think, [=, *this] could be the most easy way to solve the [=] problem. And dropping C++-14 compiler support (if I understand the topic).

Hello Zsolt:

Copying might not be desirable or doable (deleted copy ctor etc). Big no-no. :)

BR,
Lance

Subject: Re: 2022(?).2 beta
Posted by mdelfede on Sun, 18 Dec 2022 22:03:44 GMT
View Forum Message <> Reply to Message

I was the one adding Astyle *many* years ago... and it's true that it has not been updated since years.
But, IMHO, a tool to format code is quite useful. I still use it, even if it's broken on new code.
Ok for removing, but also ok for adding a new one.

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 18 Dec 2022 22:14:34 GMT
View Forum Message <> Reply to Message

mdelfede wrote on Sun, 18 December 2022 17:03I was the one adding Astyle *many* years ago... and it's true that it has not been updated since years.

But, IMHO, a tool to format code is quite useful. I still use it, even if it's broken on new code.
Ok for removing, but also ok for adding a new one.

Hi mdelfede:

Astyle's last activity is probably a few years old. Like I mentioned before, formatting code with bitfields with the old formatting utility is a disaster. It takes a lot of work to bring bitfields in order every time one formats <CtrlCore/CtrlCore.h>. bitfields is something c++ inherited from old c, present since day 1 of the language. It's surprising that this actually happened.

Go with the big guy, here llvm-org/LibFormat, is a more future proof decision.

BR,
Lance

Subject: Re: 2022(?).2 beta
Posted by zsolt on Sun, 18 Dec 2022 22:29:42 GMT
View Forum Message <> Reply to Message

Lance wrote on Sun, 18 December 2022 22:55

Copying might not be desirable or doable (deleted copy ctor etc). Big no-no. :)

BR,
Lance

Yes, you are right. I never used [*this] and didn't know that is gives a copy of the object. Always learning :)

So we have to wait some years.
C++-20 is too new to be a requirement. Ubuntu 20.04 is actively used for example by many people and companies.

Subject: Re: 2022(?).2 beta
Posted by Klugier on Sun, 18 Dec 2022 22:35:27 GMT
View Forum Message <> Reply to Message

Hello,

It looks like that in this thread we are talking about lot of things :) Backing to Mireks attention about lack of clang-format in our toolchaing on Windows. This executable can be downloaded from muttleyxd/clang-tools-static-binaries GitHub repository. On Windows when we will make decision to integrate this tool, we can put it under bin/clang-format directory or attached to

bin/clang/bin/*.

Klugier

Subject: Re: 2022(?).2 beta
Posted by mirek on Sun, 18 Dec 2022 23:15:52 GMT
View Forum Message <> Reply to Message

zsolt wrote on Sun, 18 December 2022 22:33This is a very good table, thanks.
I think, [=, *this] could be the most easy way to solve the [=] problem. And dropping C++-14 compiler support (if I understand the topic).

Except it is not the same thing. Please check the semantics....

Subject: Re: 2022(?).2 beta
Posted by mirek on Sun, 18 Dec 2022 23:18:49 GMT
View Forum Message <> Reply to Message

Klugier wrote on Sun, 18 December 2022 21:54Hello Mirek and Lance,

I agree with Lance that someday capture this by [=] might be compilation error instead of warning. Such precedence happened in the past. For example in C++17 std::auto_ptr had been removed. However for capturing this by [=], I do not see that it will be removed in C++23. So, disabling warning might give us 3-4 additional years...

Yep, exactly. When this is error, we can move on to C++20. Until then, disable warning.

This is not ideal situation. But best that can be done.

That said, I think C++ commitee is a little bit out of touch here...

Mirek

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 18 Dec 2022 23:41:11 GMT
View Forum Message <> Reply to Message

Hello Mirek and Klugier:

The following code is an excerpt from /usr/include/c++/11/bits/stl_vector.h

```
iterator
begin() _GLIBCXX_NOEXCEPT
{ return iterator(this->_M_impl._M_start); }

/**
 *  Returns a read-only (constant) iterator that points to the
 *  first element in the %vector.  Iteration is done in ordinary
 *  element order.
 */
const_iterator
begin() const _GLIBCXX_NOEXCEPT
{ return const_iterator(this->_M_impl._M_start); }

/**
 *  Returns a read/write iterator that points one past the last
 *  element in the %vector.  Iteration is done in ordinary
 *  element order.
 */
iterator
end() _GLIBCXX_NOEXCEPT
{ return iterator(this->_M_impl._M_finish); }
```

Without looking into the definition of _GLIBCXX_NOEXCEPT, most experienced c++ users(, all participants of this thread for sure,) can tell that it will expand to noexcept when the -std version supports it and vanishes otherwise.

It might not be pleasant or pretty, but it certainly works and can be argued as the most reasonable solution in this particular situation.

It's a common problem that libraries with some history need to support different versions; maintaining backward compatibility should not mean stay backward. U++ necessarily has done similar thing for similar purposes, I believe.

Why is it so hard to swallow in this particular case?

BR,
Lance

Subject: Re: 2022(?).2 beta
Posted by Lance on Sun, 18 Dec 2022 23:46:01 GMT
View Forum Message <> Reply to Message

Klugier wrote on Sun, 18 December 2022 17:35Hello,

It looks like that in this thread we are talking about lot of things :) Backing to Mireks attention about lack of clang-format in our toolchaing on Windows. This executable can be downloaded from muttleyxd/clang-tools-static-binaries GitHub repository. On Windows when we will make decision to integrate this tool, we can put it under bin/clang-format directory or attached to bin/clang/bin/*.

Klugier

OK. I was writing my last post before seeing this reply.

Subject: Re: 2022(?).2 beta
Posted by Novo on Mon, 19 Dec 2022 04:36:09 GMT
View Forum Message <> Reply to Message

zsolt wrote on Sun, 18 December 2022 17:29
So we have to wait some years.
C++-20 is too new to be a requirement. Ubuntu 20.04 is actively used for example by many people and companies.

I'm actively using Void linux which is based on Clang 12.
FreeBSD 12 is based on Clang 10.
FreeBSD 13 is based on Clang 13.
32-bit versions of Linux and BSD are quite popular :)

Subject: Re: 2022(?).2 beta
Posted by mirek on Mon, 19 Dec 2022 09:08:56 GMT
View Forum Message <> Reply to Message

Lance wrote on Mon, 19 December 2022 00:41Hello Mirek and Klugier:

The following code is an excerpt from /usr/include/c++/11/bits/stl_vector.h

```
iterator
begin() _GLIBCXX_NOEXCEPT
{ return iterator(this->_M_impl._M_start); }

/**
 *  Returns a read-only (constant) iterator that points to the
 *  first element in the %vector.  Iteration is done in ordinary
 *  element order.
 */
const_iterator
begin() const _GLIBCXX_NOEXCEPT
```

```
{ return const_iterator(this->_M_impl._M_start); }

/**
 *  Returns a read/write iterator that points one past the last
 *  element in the %vector.  Iteration is done in ordinary
 *  element order.
 */
iterator
end() _GLIBCXX_NOEXCEPT
{ return iterator(this->_M_impl._M_finish); }
```

Without looking into the definition of _GLIBCXX_NOEXCEPT, most experienced c++ users(, all participants of this thread for sure,) can tell that it will expand to noexcept when the -std version supports it and vanishes otherwise.

It might not be pleasant or pretty, but it certainly works and can be argued as the most reasonable solution in this particular situation.

It's a common problem that libraries with some history need to support different versions; maintaining backward compatibility should not mean stay backward. U++ necessarily has done similar thing for similar purposes, I believe.

Why is it so hard to swallow in this particular case?

BR,
Lance

Uhm, I guess we are presented with 2 more or less equivalently ugly options here. One of them requires a significant amount of work....

Subject: Re: 2022(?).2 beta
Posted by mirek on Mon, 19 Dec 2022 11:20:18 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Sun, 18 December 2022 19:46@Mirek: BTW, how about IDE: closing window moves it in Ctrl+Tab order after all opened documents.

I think we did discuss this few years back, and it was like by-design for you? But it keeps irritate me even after those years: I switch by Ctrl+Tab to file which I want to close, Ctrl+W to close it, and now I want to move to some other open file and hit Ctrl+Tab, and closed file is back... I'm just not used to this, and I don't see value in it, feels like wrong UX to me.

I can try to do pull request if you don't mind the change.

BTW you seem to mostly apply pull requests manually, but that way you make the contribution in git changed to your user, not sure if this is intentional, or just the way how you use git.

Hopefully fixed.

Subject: Re: 2022(?).2 beta
Posted by Lance on Mon, 19 Dec 2022 17:43:13 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 19 December 2022 04:08Lance wrote on Mon, 19 December 2022 00:41Hello Mirek and Klugier:

The following code is an excerpt from /usr/include/c++/11/bits/stl_vector.h

```
iterator
begin() _GLIBCXX_NOEXCEPT
{ return iterator(this->_M_impl._M_start); }

/**
 *  Returns a read-only (constant) iterator that points to the
 *  first element in the %vector.  Iteration is done in ordinary
 *  element order.
 */
const_iterator
begin() const _GLIBCXX_NOEXCEPT
{ return const_iterator(this->_M_impl._M_start); }

/**
 *  Returns a read/write iterator that points one past the last
 *  element in the %vector.  Iteration is done in ordinary
 *  element order.
 */
iterator
end() _GLIBCXX_NOEXCEPT
{ return iterator(this->_M_impl._M_finish); }
```

Without looking into the definition of _GLIBCXX_NOEXCEPT, most experienced c++ users(, all participants of this thread for sure,) can tell that it will expand to noexcept when the -std version supports it and vanishes otherwise.

It might not be pleasant or pretty, but it certainly works and can be argued as the most reasonable solution in this particular situation.

It's a common problem that libraries with some history need to support different versions;

maintaining backward compatibility should not mean stay backward. U++ necessarily has done similar thing for similar purposes, I believe.

Why is it so hard to swallow in this particular case?

BR,
Lance

Uhm, I guess we are presented with 2 more or less equivalently ugly options here. One of them requires a significant amount of work....

Hello Mirek and Klugier:

Another drawback for the disable-warnings option: Like Novo said, he and many similar-minded people are still using very old systems/compilers; people are different. What if in 4 years' horizon you decide that time has mature for switching to c++20 but there are still a significant number of users who wish to be able to have c++14 as an option?

From the point of smoother user experience, I am not quite sure if it's a good idea that the mainstream version today will become completely unusable the next day because it's upgraded. You likely need to deprecate it and keep it going for a couple of more years so people have time to transit to the new mainstream version/standard.

If this will be the case, we end up still need to be able to support both c++14 and c++20 at the same time for at least a period of time: trouble is deferred instead of solved. And as times goes, more [=] cases will be added to U++ (not in a significant number, but it's a non-decreasing function of time), chance is it will take more time and effort at the postponed switch date.

BR,
Lance

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.

Subject: Re: 2022(?).2 beta
Posted by mirek on Mon, 19 Dec 2022 19:22:52 GMT
View Forum Message <> Reply to Message

Lance 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)); };
  }
};
```

Subject: Re: 2022(?).2 beta
Posted by Lance on Mon, 19 Dec 2022 22:14:58 GMT
View Forum Message <> Reply to Message

mirek 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.

Subject: Re: 2022(?).2 beta
Posted by mirek on Mon, 19 Dec 2022 22:35:31 GMT
View Forum Message <> Reply to Message

Lance 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....

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.

Subject: Re: 2022(?).2 beta
Posted by Novo on Mon, 19 Dec 2022 23:19:33 GMT
View Forum Message <> Reply to Message

Lance wrote on Mon, 19 December 2022 12:43
Like Novo said, he and many similar-minded people are still using very old systems/compilers;
Systems I mentioned are not "very old".
Void Linux is a rolling distro (that means that you get latest versions of everything except of core packages like Clang). This is done for stability. I personally spent quite a lot of time dealing with bugs in compilers (code generators). And because of that I prefer to use a stable and well tested compiler even if it seems to be outdated.
Void Linux is ranked number five by DistroWatch.
Netflix runs on FreeBSD.
Almost all gaming consoles run FreeBSD.

So "very old systems/compilers" aren't that old :)

Subject: Re: 2022(?).2 beta
Posted by Lance on Mon, 19 Dec 2022 23:48:59 GMT
View Forum Message <> Reply to Message

Novo wrote on Mon, 19 December 2022 18:19Lance wrote on Mon, 19 December 2022 12:43
Like Novo said, he and many similar-minded people are still using very old systems/compilers;
Systems I mentioned are not "very old".
Void Linux is a rolling distro (that means that you get latest versions of everything except of core packages like Clang). This is done for stability. I personally spent quite a lot of time dealing with bugs in compilers (code generators). And because of that I prefer to use a stable and well tested compiler even if it seems to be outdated.
Void Linux is ranked number five by DistroWatch.
Netflix runs on FreeBSD.
Almost all gaming consoles run FreeBSD.
So "very old systems/compilers" aren't that old :)

ok, they are not that old.

My point is, say, one day, Mirek decides c++20 is stable enough to switch to, and c++14 support will be abandoned at the same time. Do you expect you undoubtedly have the same judgement on the stability of c++20? And you will be ready to make the move simultaneously? If your answers to both questions are yes and significantly most users also agree, then disable-warning is the surest way to take.