Subject: 2024rc1
Posted by mirek on Sat, 28 Sep 2024 07:16:42 GMT
View Forum Message <> Reply to Message

2024 (rev. 17417) (Sep 2024)

https://sourceforge.net/projects/upp/files/upp/2024rc1/

Core

First release of U++ that requires C++17
Moveable concept redesigned with C++17 features. U++ now allows non-moveable types to be stored in Vector flavor of containers (using Upp::is_upp_guest). PODs are automatically moveable (aka trivially relocatable)
Upp::Tuple now supports structured binding
GetFileTime, GetFileLength, FileExists, DirectoryExists and FileMapping refactored
Stream::GetAll now invokes LoadError on negative count
ValueCache limits setting methods are simplified
Value now directly supports 'float' type
Some iffy code now made more C++ compliant (e.g. always using memcpy for unaligned data)
AsXML had new XML_ESCAPELF
Improved DarkTheme function

plugin/Zip

zip64 support

Draw

UHD image now can serve as source for SD image
New S3 .iml image flag - the images are drawn supersampled 3x, usually without antialiasing, and only downsampled at runtime

Painter

Multithreaded rendering further optimised
New image filtering parameter - so far, rendering image was always with bilinear filtering, new parameter allows other Image filter like Lanczos 3

CtrlCore

Horizontal mouse scroll wheel support
CtrlMapper now provides operator()(Ctrl, T, const T& factor) for simple unit conversions
gtk backend improvements, XWayland mouse cursor bug workaround

CtrlLib

CtrlMapper now provides operator()(Ctrl, T, const T& factor) for simple unit conversions

ide

Icon Designer refactored and optimised, new tools added, S3 flag support added
Alt-M now goes to special scratchpad file of the same type as is current file, this is helpful e.g. for temporary storing and editing parts of .iml images that are then composed to the final image.
Output directory in assembly definition now can be left empty and defaults to reasonable path.
Hexadecimal view is now much faster
Fixed further corner case Assist++ problems
Layout designer text field, used with e.g. Labels, now has Qtf button to edit text with RichEdit
Git file history now goes through renames
Compare with menu now suggests files in Download folder too
Main package configuration dialog improved

plugin upgrades

plugin/sqlite3: 3.46.0
plugin/lzma: 24.6
plugin/zstd: 1.5.6
Core: LZ4 1.9.4
plugin/z: 1.3.1
plugin/png: 1.6.46
plugin/tif: 4.6.0
plugin/jpeg: 9f

Win32

OpenSSL upgraded to3.2.1
Clang compiler upgraded to 18.1.5

Subject: Re: 2024rc1
Posted by Tom1 on Sat, 28 Sep 2024 16:36:50 GMT
View Forum Message <> Reply to Message

Mirek,

Thanks for your hard work!

Best regards,

Tom

Subject: Re: 2024rc1
Posted by JeyCi on Sat, 28 Sep 2024 17:21:46 GMT
View Forum Message <> Reply to Message

what does "Win32" mean in the description ? does it mean that this version of UPP can be installed in windows-10 32x ?

Subject: Re: 2024rc1
Posted by Didier on Sat, 28 Sep 2024 18:50:36 GMT
View Forum Message <> Reply to Message

Thanks Mirek,

Great work (especially for C++17) !!

Subject: Re: 2024rc1
Posted by Tom1 on Sat, 28 Sep 2024 20:55:40 GMT
View Forum Message <> Reply to Message

Hi Mirek,

It seems that Progress::SetPos() 'eats' memory.

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

using namespace Upp;

GUI_APP_MAIN
{
 for(int x=0;x<5;x++){
  Progress progress;
  progress.Create();
  progress.SetTotal(20000);
  for(int i=0;i<20000;i++){
   if(progress.Canceled()) break;
   progress.SetPos(i);
   progress.SetText(Format("MemoryUsedKb %d", MemoryUsedKb()));
  }
  Sleep(2000);
 }
}
```

Best regards,

Tom

Subject: Re: 2024rc1

Posted by mirek on Sun, 29 Sep 2024 18:51:43 GMT
View Forum Message <> Reply to Message

JeyCi wrote on Sat, 28 September 2024 19:21what does "Win32" mean in the description ? does it mean that this version of UPP can be installed in windows-10 32x ?

E.g.

https://stackoverflow.com/questions/61776207/where-does-win32-come-from-when-im-using-windows-64bit

Win32 is the name of API and while it was originally implemented for 32bit CPUs, it is used in 64 bit variant as well.

Funny part is that even library names, like kernel32.dll and user32.dll are still used for 64 bit variants.

Subject: Re: 2024rc1
Posted by mirek on Sun, 29 Sep 2024 18:55:16 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Sat, 28 September 2024 22:55Hi Mirek,

It seems that Progress::SetPos() 'eats' memory.

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

using namespace Upp;

GUI_APP_MAIN
{
 for(int x=0;x<5;x++){
  Progress progress;
  progress.Create();
  progress.SetTotal(20000);
  for(int i=0;i<20000;i++){
   if(progress.Canceled()) break;
   progress.SetPos(i);
   progress.SetText(Format("MemoryUsedKb %d", MemoryUsedKb()));
  }
  Sleep(2000);
 }
}
```

Best regards,

Tom

Teste with CLANG, CLANGx64 and MSBT 64, problem not reproduced. Perhaps needs more instructions to reproduce?

Subject: Re: 2024rc1
Posted by Tom1 on Sun, 29 Sep 2024 20:02:16 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 29 September 2024 21:55
Teste with CLANG, CLANGx64 and MSBT 64, problem not reproduced. Perhaps needs more instructions to reproduce?
Hi Mirek,

Do you mean that your MemoryUsedKb value did not keep climbing through all the five runs???

First, this is not a normal memory leak catched with debugger. This happens on Windows 11 Professional with all compilers: CLANG, CLANGx64, MSBT22, MSBT22x64. The MemoryUsedKb starts out at around 1200 kB on first start, and then keeps gradually rising up to about 20000.. 30000 kB when the fifth run is complete... and more if we let it run longer with higher values of x.

Best regards,

Tom

Subject: Re: 2024rc1
Posted by mirek on Sun, 29 Sep 2024 21:14:09 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Sun, 29 September 2024 22:02mirek wrote on Sun, 29 September 2024 21:55
Teste with CLANG, CLANGx64 and MSBT 64, problem not reproduced. Perhaps needs more instructions to reproduce?
Hi Mirek,

Do you mean that your MemoryUsedKb value did not keep climbing through all the five runs???

First, this is not a normal memory leak catched with debugger. This happens on Windows 11 Professional with all compilers: CLANG, CLANGx64, MSBT22, MSBT22x64. The MemoryUsedKb starts out at around 1200 kB on first start, and then keeps gradually rising up to about 20000.. 30000 kB when the fifth run is complete... and more if we let it run longer with higher values of x.

Best regards,

Tom

Reproduced: It is a problem of dark mode emulation. I was trying with normal mode first...

It is quite obvious - various variants of progress bar are drawn in normal mode, then converted to DarkTheme and the result is cached.

You can adjust maximum size of cache with

```
GUI_APP_MAIN
{
 SetupValueCache(2000, 1);

 for(int x=0;x<5;x++){
```

and it stops increasing the memory.

So I do not think this is a problem nor a bug - it is just using general caching mechanism where it is good to cache results for performance reasons.

Mirek

Subject: Re: 2024rc1
Posted by Lance on Sun, 29 Sep 2024 23:02:50 GMT
View Forum Message <> Reply to Message

Thank you Mirek for the great job!

Subject: Re: 2024rc1
Posted by mirek on Mon, 30 Sep 2024 10:01:50 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 29 September 2024 23:14Tom1 wrote on Sun, 29 September 2024 22:02mirek wrote on Sun, 29 September 2024 21:55
Teste with CLANG, CLANGx64 and MSBT 64, problem not reproduced. Perhaps needs more instructions to reproduce?
Hi Mirek,

Do you mean that your MemoryUsedKb value did not keep climbing through all the five runs???

First, this is not a normal memory leak catched with debugger. This happens on Windows 11 Professional with all compilers: CLANG, CLANGx64, MSBT22, MSBT22x64. The MemoryUsedKb starts out at around 1200 kB on first start, and then keeps gradually rising up to about 20000..

30000 kB when the fifth run is complete... and more if we let it run longer with higher values of x.

Best regards,

Tom

Reproduced: It is a problem of dark mode emulation. I was trying with normal mode first...

It is quite obvious - various variants of progress bar are drawn in normal mode, then converted to DarkTheme and the result is cached.

You can adjust maximum size of cache with

```
GUI_APP_MAIN
{
 SetupValueCache(2000, 1);

 for(int x=0;x<5;x++){
```

and it stops increasing the memory.

So I do not think this is a problem nor a bug - it is just using general caching mechanism where it is good to cache results for performance reasons.

Mirek

Upon further reflectio I decided that caching progress causes is just trashing the cache, so optimised that out (with the advantage that the result is now actually faster in Win32).

The only downside is that now I have to think whether to apply the similar treatment to scrollbar thumbs... :) But probably not.

Mirek

Subject: Re: 2024rc1
Posted by Tom1 on Mon, 30 Sep 2024 10:06:42 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 30 September 2024 13:01
Upon further reflectio I decided that caching progress causes is just trashing the cache, so optimised that out (with the advantage that the result is now actually faster in Win32).

The only downside is that now I have to think whether to apply the similar treatment to scrollbar

thumbs... :) But probably not.

Mirek
Thanks Mirek,

Nice Progress! (Progress behaves very well now.)

Best regards,

Tom

Subject: Re: 2024rc1
Posted by Tom1 on Mon, 30 Sep 2024 13:37:48 GMT
View Forum Message <> Reply to Message

Hi,

I think the "TIMING XpPaint" can be dropped from the release log now.

BR, Tom

Subject: Re: 2024rc1
Posted by Lance on Sat, 12 Oct 2024 15:31:57 GMT
View Forum Message <> Reply to Message

C:\upp\upp.src\uppsrc\CtrlCore\Ctrl.cpp (443): error C2445: result type of conditional expression is ambiguous: types 'Upp::String' and 'const char [5]' can be converted to multiple common types

```
String Name(const Ctrl *ctrl)
{
 return ctrl ? ctrl->Name() : "NULL";
}
```

MSBT has issue with above.

Subject: Re: 2024rc1
Posted by Lance on Sat, 12 Oct 2024 18:20:54 GMT
View Forum Message <> Reply to Message

Code Reformat Issue.

I have been having issues with it for a while. Today I spend a few hours to create a almost mininal

example.

```
#include <CtrlLib/CtrlLib.h>
using namespace Upp;

class S{
 void Set ( Size& sz,
        int x, int y, int z,
        int x1, int y1, int z1
      );

 const S& f(Rect& r, Rect& s)const;

 struct D
 {
  int s(int rc)const
  {
   return rc*2;
  }

  int e(int rc)const
  {
   return rc*1;
  }

  int w(int rc)const
  {
   return rc*3;
  }

  int t()const
  {
   return 4;
  }


  int g(int k)const
  {
   return k;
  }

  void alloc();

  void alloc(int a);

  int v1;
  int v2;
 };
```

```
 D col,row;
};


void S::D::alloc ()
{
}

static void func ( int& x, int& y, int& x1, int& y1, int x2, int y2,
     int x3, int y3, int x4, int y4
      )
{
}


const S& S::f(Rect& r, Rect& s)const
{
 int t, l, row_section_bottom, n;
 r.top = row.g(r.top);
 s.top = t !=0 && r.top<row.v1 ? row.v1 : r.top;

 r.left = col.g(r.left);
 s.left = l != 0 && r.left< col.v1 ? col.v1 : r.left;

 r.bottom = row.g(r.bottom);
 s.bottom = row_section_bottom == 1  && r.bottom > row.v1 + row.v2 ?
  row.v1+row.v2 : r.bottom;

 r.right = col.g(r.right);
 s.right = n == 1  && r.right> col.v1 + col.v2 ?
  col.v1+col.v2 : r.right;
 return *this;
}
```

Add the code as a separate cpp file in a CtrlLib application, with it current, press Ctrl+I to reformat it. The file before reformat compiles fine, not the reformatted one.

Subject: Re: 2024rc1
Posted by Lance on Sun, 13 Oct 2024 04:10:46 GMT
View Forum Message <> Reply to Message

two more minor issues (or maybe non-issues).

1. In dark theme, the Topic++ editor is still in light mode. Would be nice if it can melt with

environment.

2. An unorthodox act will crash theide very badly. Take for example, let's open any package that uses package Draw, for example Examples/Color. Click package Draw at the upper-left part of TheIDE to list files in Draw. In the lower-left part of TheIDE, click the first file, Draw.h. Ctrl+Shift+G to bring out the "go to line... "dialog, enter 45. Let's add a help content for data, in a stupid way.

Right click the dark blue squre beside the line "int64 data;", choose "Insert into topic://Draw/src/Drawing_en-us".

Oops, we just noticed that's the wrong file. We cut the line

int64 data;

from Drawing_en-us.tpp and paste it to the very end of Draw_en-us.tpp. Go back to Draw.h, line 45, right click the dark blue square again, select "copy code reference id".

Switch back to Draw_en-us.tpp, with caret on the newly pasted line, Ctrl-M to bring out the "code reference" dialog. Paste the code reference id we just copied, which should be "Upp::Font::data". Click OK to close the dialog.

So far so good. Click Draw.h tab to bring it current. Oops, an "Invalid memory access!" occurs. If it doesn't, move mouse to over the blue square beside line 45 to show a help content. It happened to me twice, verified.

This error can be fixed in the following way. When restarting theide, you will be prompted to disable Assist features, select "Yes", go to the tpp file, delete thr problematic help line. Then use menu Setup/Settings, on Assist tab, check the first item "Assist libclang parser is enabled..." to reenable Assit++.

Subject: Re: 2024rc1
Posted by mirek on Sun, 13 Oct 2024 05:52:16 GMT
View Forum Message <> Reply to Message

Lance wrote on Sun, 13 October 2024 06:10two more minor issues (or maybe non-issues).

1. In dark theme, the Topic++ editor is still in light mode. Would be nice if it can melt with environment.

Colors are user setting. I do not know whether user wants it or not. Normally, theide chooses the scheme on the first run, but then user is allowed to change it any way he likes, so it would not be very nice to change the scheme at that point.

Well, maybe we could solve that with having 2 user configurations, one for light one for dark? But in the next release..

Subject: Re: 2024rc1
Posted by Oblivion on Sun, 13 Oct 2024 07:53:32 GMT
View Forum Message <> Reply to Message

Hi Mirek,

I think this change should be reverted for the time being, as I tried to explain, it creates more problem than it solves (leaves paths percentage encoded).

Best regards,
Oblivion

Subject: Re: 2024rc1
Posted by Lance on Sun, 13 Oct 2024 11:28:32 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 13 October 2024 01:52Lance wrote on Sun, 13 October 2024 06:10two more minor issues (or maybe non-issues).

1. In dark theme, the Topic++ editor is still in light mode. Would be nice if it can melt with environment.

Colors are user setting. I do not know whether user wants it or not. Normally, theide chooses the scheme on the first run, but then user is allowed to change it any way he likes, so it would not be very nice to change the scheme at that point.

Well, maybe we could solve that with having 2 user configurations, one for light one for dark? But in the next release..

That brings me to another UI suggestion.

Here is how we need to make changes for theme switching, etc.

The one on IDE tab is convenient. 3 options should be well tuned and easy to change:
1. stick with light theme;
2. stick with dark theme;
3. use the theme setting from Host platform.

The one on "Syntax highlighting" tab is not as well-thought. I recommend to change it to the drop choice similar to the one on "IDE" tab. At the moment we can have these entries
1. light theme (stick with light theme, what the "white theme" buttom will do currently);
2. dark theme (stick with dark theme, what the "Dark theme" button will do currently);
3. "Use host theme"(similar to what the similar one on "IDE" tab would do);
4. "Use default colors" (what the Restore default colors button would do).

And a check box below or above the drop choice, saying
[ ] Apply this to IDE and topic++

And do what it promises.

Apply similar UI changes to IDE tabs. So that a user can make desired changes from one of the locations without having to set 3 places for one intention.

BTW, "Use host platform" seems to be a reasonable default for me. If a user choose certain theme for his windows system, chance is he would like the same for TheIDE.

BTW, I don't know how to setting colors for Topic++ up until now. I wouldn't notice I need to change setting in 2 places to make TheIDE looks natural in DarkTheme had Ubuntu not provided the convenient way to switch theme in its recent version.

Of course, these are non-emmergent UI refinement that can be done in later release after more discussion. Anyone who agrees with my suggestion, please vote yes. :lol:

File Attachments

1) tmp.png, downloaded 363 times

Subject: Re: 2024rc1
Posted by mirek on Sun, 13 Oct 2024 11:55:05 GMT
View Forum Message <> Reply to Message

Lance wrote on Sun, 13 October 2024 13:28
1. In dark theme, the Topic++ editor is still in light mode. Would be nice if it can melt with environment.

Another pretty though nut to crack...

What will be the equivalent of choosing the text color in the dark mode?

Not that you are editing text for both modes. We handle, barely, translation of light theme colors to dark theme colors, but is the user, while editing topic++, supposed to select light mode colors (as is now) or dark mode colors that will look different in light mode?

Anyway, all in all, I am postponing this after the release...

Subject: Re: 2024rc1
Posted by mirek on Sun, 13 Oct 2024 11:58:46 GMT
View Forum Message <> Reply to Message

Lance wrote on Sun, 13 October 2024 13:28
The one on "Syntax highlighting" tab is not as well-thought. I recommend to change it to the drop choice similar to the one on "IDE" tab. At the moment we can have these entries
1. light theme (stick with light theme, what the "white theme" buttom will do currently);
2. dark theme (stick with dark theme, what the "Dark theme" button will do currently);
3. "Use host theme"(similar to what the similar one on "IDE" tab would do);
4. "Use default colors" (what the Restore default colors button would do).

This goes into reasonable direction, however I think that the actual choice should be single option:

"User defined colors"

If active, colors are editable (and do not change when mode changes), if not, it is current default colors for dark/light. Maybe current buttons can stay, although only active when colors are editable...

Subject: Re: 2024rc1
Posted by Lance on Sun, 13 Oct 2024 12:20:28 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 13 October 2024 07:55Lance wrote on Sun, 13 October 2024 13:28
1. In dark theme, the Topic++ editor is still in light mode. Would be nice if it can melt with environment.

Another pretty though nut to crack...

What will be the equivalent of choosing the text color in the dark mode?

Not that you are editing text for both modes. We handle, barely, translation of light theme colors to dark theme colors, but is the user, while editing topic++, supposed to select light mode colors (as is now) or dark mode colors that will look different in light mode?

Anyway, all in all, I am postponing this after the release...

I see. I just checked Libre Office. It doesn't respect DarkTheme for its content area. So maybe we shall just accept what we have right now.

Unless u++ users can agree on limiting the color selection to the ones that are theme-defined --

this is doable, but quite involving, and also possibly not what our guys want. Very low priority if it ever will be considered.

Subject: Re: 2024rc1
Posted by Lance on Sun, 13 Oct 2024 12:23:33 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 13 October 2024 07:58Lance wrote on Sun, 13 October 2024 13:28
The one on "Syntax highlighting" tab is not as well-thought. I recommend to change it to the drop choice similar to the one on "IDE" tab. At the moment we can have these entries
1. light theme (stick with light theme, what the "white theme" buttom will do currently);
2. dark theme (stick with dark theme, what the "Dark theme" button will do currently);
3. "Use host theme"(similar to what the similar one on "IDE" tab would do);
4. "Use default colors" (what the Restore default colors button would do).

This goes into reasonable direction, however I think that the actual choice should be single option:

"User defined colors"

If active, colors are editable (and do not change when mode changes), if not, it is current default colors for dark/light. Maybe current buttons can stay, although only active when colors are editable...

Sounds good!

Subject: Re: 2024rc1
Posted by mirek on Mon, 14 Oct 2024 13:36:23 GMT
View Forum Message <> Reply to Message

Oblivion wrote on Sun, 13 October 2024 09:53Hi Mirek,

I think this change should be reverted for the time being, as I tried to explain, it creates more problem than it solves (leaves paths percentage encoded).

Best regards,
Oblivion

Hopefully fixed (added flag to UrlDecode).

Subject: Re: 2024rc1

Lance wrote on Sat, 12 October 2024 20:20Code Reformat Issue.

I have been having issues with it for a while. Today I spend a few hours to create a almost mininal example.

```
#include <CtrlLib/CtrlLib.h>
using namespace Upp;

class S{
 void Set ( Size& sz,
       int x, int y, int z,
       int x1, int y1, int z1
      );

 const S& f(Rect& r, Rect& s)const;

 struct D
 {
  int s(int rc)const
  {
   return rc*2;
  }

  int e(int rc)const
  {
   return rc*1;
  }

  int w(int rc)const
  {
   return rc*3;
  }

  int t()const
  {
   return 4;
  }


  int g(int k)const
  {
   return k;
  }

  void alloc();
```

```
  void alloc(int a);

  int v1;
  int v2;
 };
 D col,row;
};


void S::D::alloc ()
{
}

static void func ( int& x, int& y, int& x1, int& y1, int x2, int y2,
     int x3, int y3, int x4, int y4
      )
{
}


const S& S::f(Rect& r, Rect& s)const
{
 int t, l, row_section_bottom, n;
 r.top = row.g(r.top);
 s.top = t !=0 && r.top<row.v1 ? row.v1 : r.top;

 r.left = col.g(r.left);
 s.left = l != 0 && r.left< col.v1 ? col.v1 : r.left;

 r.bottom = row.g(r.bottom);
 s.bottom = row_section_bottom == 1  && r.bottom > row.v1 + row.v2 ?
  row.v1+row.v2 : r.bottom;

 r.right = col.g(r.right);
 s.right = n == 1  && r.right> col.v1 + col.v2 ?
  col.v1+col.v2 : r.right;
 return *this;
}
```

Add the code as a separate cpp file in a CtrlLib application, with it current, press Ctrl+I to reformat it. The file before reformat compiles fine, not the reformatted one.

Works for me in windows and works in Ubuntu. Unfortunately, this feature is now using clang-format that can be different per distro...

Would be nice to give me a hint which host platform is in use...

Also, if nothing helps, please post reformatted text as well.

Mirek

Subject: Re: 2024rc1
Posted by mirek on Mon, 14 Oct 2024 14:17:17 GMT
View Forum Message <> Reply to Message

Quote:
Oops, we just noticed that's the wrong file. We cut the line

int64 data;

from Drawing_en-us.tpp and paste it to the very end of Draw_en-us.tpp. Go back to Draw.h, line 45, right click the dark blue square again, select "copy code reference id".

Switch back to Draw_en-us.tpp, with caret on the newly pasted line, Ctrl-M to bring out the "code reference" dialog. Paste the code reference id we just copied, which should be "Upp::Font::data". Click OK to close the dialog.

So far so good. Click Draw.h tab to bring it current. Oops, an "Invalid memory access!" occurs. If it doesn't, move mouse to over the blue square beside line 45 to show a help content. It happened to me twice, verified.

This error can be fixed in the following way. When restarting theide, you will be prompted to disable Assist features, select "Yes", go to the tpp file, delete thr problematic help line. Then use menu Setup/Settings, on Assist tab, check the first item "Assist libclang parser is enabled..." to reenable Assit++.

Cannot reproduce.

Maybe if this could be done without using uppsrc sources and tpp, just in single package, maybe you can prepare for me "crashing package"?

Alternative, can you run it in debugger? :)

Mirek

Subject: Re: 2024rc1
Posted by Lance on Mon, 14 Oct 2024 14:23:10 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 14 October 2024 10:13Lance wrote on Sat, 12 October 2024 20:20Code

Reformat Issue.

I have been having issues with it for a while. Today I spend a few hours to create a almost mininal example.

```
#include <CtrlLib/CtrlLib.h>
using namespace Upp;

class S{
 void Set ( Size& sz,
        int x, int y, int z,
        int x1, int y1, int z1
      );

 const S& f(Rect& r, Rect& s)const;

 struct D
 {
  int s(int rc)const
  {
   return rc*2;
  }

  int e(int rc)const
  {
   return rc*1;
  }

  int w(int rc)const
  {
   return rc*3;
  }

  int t()const
  {
   return 4;
  }


  int g(int k)const
  {
   return k;
  }

  void alloc();

  void alloc(int a);
```

```
  int v1;
  int v2;
 };
 D col,row;
};


void S::D::alloc ()
{
}

static void func ( int& x, int& y, int& x1, int& y1, int x2, int y2,
     int x3, int y3, int x4, int y4
      )
{
}


const S& S::f(Rect& r, Rect& s)const
{
 int t, l, row_section_bottom, n;
 r.top = row.g(r.top);
 s.top = t !=0 && r.top<row.v1 ? row.v1 : r.top;

 r.left = col.g(r.left);
 s.left = l != 0 && r.left< col.v1 ? col.v1 : r.left;

 r.bottom = row.g(r.bottom);
 s.bottom = row_section_bottom == 1  && r.bottom > row.v1 + row.v2 ?
  row.v1+row.v2 : r.bottom;

 r.right = col.g(r.right);
 s.right = n == 1  && r.right> col.v1 + col.v2 ?
  col.v1+col.v2 : r.right;
 return *this;
}
```

Add the code as a separate cpp file in a CtrlLib application, with it current, press Ctrl+I to reformat it. The file before reformat compiles fine, not the reformatted one.

Works for me in windows and works in Ubuntu. Unfortunately, this feature is now using clang-format that can be different per distro...

Would be nice to give me a hint which host platform is in use...

Also, if nothing helps, please post reformatted text as well.

Mirek

Operation System: Ubuntu 24.04.1 LTS
GNOME version: 46
Windowing System: Wayland
clang-format --version: Ubuntu clang-format version 18.1.3 (1ubuntu1)

Reformatted output

```
#include <CtrlLib/CtrlLib.h>
using namespace Upp;

class S {
 void Set(Size& sz, int x, int y, int z, int x1, int y1, int z1);

 const S& f(Rect& r, Rect& s) const;

 struct D {
  int s(int rc) const { return rc * 2; }

  int e(int rc) const { return rc * 1; }

  int w(int rc) const { return rc * 3; }

  int t() const { return 4; }

  int g(int k) const { return k; }

  void alloc();

  void alloc(int a);

  int v1;
  int v2;
 };
 D col, row;
};

void S::D::alloc() {}

static void func(int& x, int& y, int& x1, int& y1, int x2, int y2, int x3, int y3, int x4,
                 int y4)
{
}

const S& S::f(Rect& r, Rect& s) const
```

```
{
 int t, l, row_section_bottom, n;
 r.top = row.g(r.top);
 s.top = t != 0 && r.top < row.v1 ? row.v1 : r.top;
 r.left = col.g(r.left);
 s.left = l != 0 && r.left< col.v1 ? col.v1 : r.left;
 s.left = l != 0 && r.left < col.v1 ? col.v1 : r.left;
 r.bottom = row.g(r.bottom);
 s.bottom = row_section_bottom == 1  && r.bottom > row.v1 + row.v2 ?
 s.bottom =
  row_section_bottom == 1 && r.bottom > row.v1 + row.v2 ? row.v1 + row.v2 : r.bottom;
 r.right = col.g(r.right);
 s.right = n == 1  && r.right> col.v1 + col.v2 ?
 s.right = n == 1 && r.right > col.v1 + col.v2 ? col.v1 + col.v2 : r.right;
}
```

Line 46 (?), Line 48 ending (;), LIne 50 (?), Line 51 ending (;) are highlighted by theide(libclang) to indicate grammer errors.

Subject: Re: 2024rc1
Posted by mirek on Mon, 14 Oct 2024 14:42:39 GMT
View Forum Message <> Reply to Message

Lance wrote on Mon, 14 October 2024 16:23mirek wrote on Mon, 14 October 2024 10:13Lance wrote on Sat, 12 October 2024 20:20Code Reformat Issue.

I have been having issues with it for a while. Today I spend a few hours to create a almost mininal example.

```
#include <CtrlLib/CtrlLib.h>
using namespace Upp;

class S{
 void Set ( Size& sz,
        int x, int y, int z,
        int x1, int y1, int z1
      );

 const S& f(Rect& r, Rect& s)const;

 struct D
 {
  int s(int rc)const
  {
   return rc*2;
```

```
  }

  int e(int rc)const
  {
   return rc*1;
  }

  int w(int rc)const
  {
   return rc*3;
  }

  int t()const
  {
   return 4;
  }


  int g(int k)const
  {
   return k;
  }

  void alloc();

  void alloc(int a);

  int v1;
  int v2;
 };
 D col,row;
};


void S::D::alloc ()
{
}

static void func ( int& x, int& y, int& x1, int& y1, int x2, int y2,
     int x3, int y3, int x4, int y4
      )
{
}


const S& S::f(Rect& r, Rect& s)const
{
 int t, l, row_section_bottom, n;
```

```
 r.top = row.g(r.top);
 s.top = t !=0 && r.top<row.v1 ? row.v1 : r.top;

 r.left = col.g(r.left);
 s.left = l != 0 && r.left< col.v1 ? col.v1 : r.left;

 r.bottom = row.g(r.bottom);
 s.bottom = row_section_bottom == 1  && r.bottom > row.v1 + row.v2 ?
  row.v1+row.v2 : r.bottom;

 r.right = col.g(r.right);
 s.right = n == 1  && r.right> col.v1 + col.v2 ?
  col.v1+col.v2 : r.right;
 return *this;
}
```

Add the code as a separate cpp file in a CtrlLib application, with it current, press Ctrl+I to reformat it. The file before reformat compiles fine, not the reformatted one.

Works for me in windows and works in Ubuntu. Unfortunately, this feature is now using clang-format that can be different per distro...

Would be nice to give me a hint which host platform is in use...

Also, if nothing helps, please post reformatted text as well.

Mirek

Operation System: Ubuntu 24.04.1 LTS
GNOME version: 46
Windowing System: Wayland
clang-format --version: Ubuntu clang-format version 18.1.3 (1ubuntu1)

Reformatted output

```
#include <CtrlLib/CtrlLib.h>
using namespace Upp;

class S {
 void Set(Size& sz, int x, int y, int z, int x1, int y1, int z1);

 const S& f(Rect& r, Rect& s) const;

 struct D {
  int s(int rc) const { return rc * 2; }
```

```
  int e(int rc) const { return rc * 1; }

  int w(int rc) const { return rc * 3; }

  int t() const { return 4; }

  int g(int k) const { return k; }

  void alloc();

  void alloc(int a);

  int v1;
  int v2;
 };
 D col, row;
};

void S::D::alloc() {}

static void func(int& x, int& y, int& x1, int& y1, int x2, int y2, int x3, int y3, int x4,
                 int y4)
{
}

const S& S::f(Rect& r, Rect& s) const
{
 int t, l, row_section_bottom, n;
 r.top = row.g(r.top);
 s.top = t != 0 && r.top < row.v1 ? row.v1 : r.top;
 r.left = col.g(r.left);
 s.left = l != 0 && r.left< col.v1 ? col.v1 : r.left;
 s.left = l != 0 && r.left < col.v1 ? col.v1 : r.left;
 r.bottom = row.g(r.bottom);
 s.bottom = row_section_bottom == 1  && r.bottom > row.v1 + row.v2 ?
 s.bottom =
  row_section_bottom == 1 && r.bottom > row.v1 + row.v2 ? row.v1 + row.v2 : r.bottom;
 r.right = col.g(r.right);
 s.right = n == 1  && r.right> col.v1 + col.v2 ?
 s.right = n == 1 && r.right > col.v1 + col.v2 ? col.v1 + col.v2 : r.right;
}
```

Line 46 (?), Line 48 ending (;), LIne 50 (?), Line 51 ending (;) are highlighted by theide(libclang) to indicate grammer errors.

It must be some specific clang format setting that I am unable to reproduce. Can you give me some hints? E.g. screenshot of "format with options" window and/or .clang-format file? (It is listed in that dialog).

Subject: Re: 2024rc1
Posted by Lance on Mon, 14 Oct 2024 15:09:20 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 14 October 2024 10:17
Cannot reproduce.

Maybe if this could be done without using uppsrc sources and tpp, just in single package, maybe you can prepare for me "crashing package"?

Alternative, can you run it in debugger? :)

Mirek

I wasn't able to reproduce it in a new, small project. I tried to abuse logs branch to create a n instance but couldn't.

Could you try to unzip the 3 tpp files that's changed and replace their namesakes in uppsrc/Draw/src.tpp folder?

Then somehow go to Draw/Draw.h, line 45, try to display a topic++ help window for "int64 data".

Thanks!

File Attachments
1) tpps.zip, downloaded 120 times

Subject: Re: 2024rc1
Posted by Lance on Mon, 14 Oct 2024 15:21:39 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 14 October 2024 10:42
It must be some specific clang format setting that I am unable to reproduce. Can you give me some hints? E.g. screenshot of "format with options" window and/or .clang-format file? (It is listed in that dialog).

.clang-format file, current: /home/lance/upp.src/.clang-format

.clang-format file for U++ framework

```
---
BasedOnStyle: LLVM
UseTab: AlignWithSpaces
IndentWidth: 4
TabWidth: 4
ColumnLimit: 96
---
Language: Cpp
AccessModifierOffset: -4
AllowShortFunctionsOnASingleLine: All
AlwaysBreakTemplateDeclarations: true
BreakBeforeBraces: Stroustrup
BreakConstructorInitializers: BeforeComma
CompactNamespaces: true
DerivePointerAlignment: false
IfMacros: ['ONCELOCK']
PointerAlignment: Left
SpaceBeforeParens: Custom
SpaceBeforeParensOptions:
    AfterControlStatements: false
IndentAccessModifiers: false
IndentPPDirectives: None
```

Everything is just as shipped with u++ distribution. I didn't touch any settings.

Subject: Re: 2024rc1
Posted by mirek on Tue, 15 Oct 2024 05:40:21 GMT
View Forum Message <> Reply to Message

Lance wrote on Mon, 14 October 2024 17:09mirek wrote on Mon, 14 October 2024 10:17
Cannot reproduce.

Maybe if this could be done without using uppsrc sources and tpp, just in single package, maybe you can prepare for me "crashing package"?

Alternative, can you run it in debugger? :)

Mirek

I wasn't able to reproduce it in a new, small project. I tried to abuse logs branch to create a n instance but couldn't.

Could you try to unzip the 3 tpp files that's changed and replace their namesakes in uppsrc/Draw/src.tpp folder?

Then somehow go to Draw/Draw.h, line 45, try to display a topic++ help window for "int64 data".

Thanks!

IDK, there is just 1 .tpp file in .zip (Draw_en-us.tpp) and it seems to be unchanged from the master...

Subject: Re: 2024rc1
Posted by mirek on Tue, 15 Oct 2024 11:31:33 GMT
View Forum Message <> Reply to Message

Lance wrote on Mon, 14 October 2024 16:23mirek wrote on Mon, 14 October 2024 10:13Lance wrote on Sat, 12 October 2024 20:20Code Reformat Issue.

I have been having issues with it for a while. Today I spend a few hours to create a almost mininal example.

```
#include <CtrlLib/CtrlLib.h>
using namespace Upp;

class S{
 void Set ( Size& sz,
        int x, int y, int z,
        int x1, int y1, int z1
      );

 const S& f(Rect& r, Rect& s)const;

 struct D
 {
  int s(int rc)const
  {
   return rc*2;
  }

  int e(int rc)const
  {
   return rc*1;
  }

  int w(int rc)const
  {
   return rc*3;
  }

  int t()const
  {
```

```
   return 4;
  }


  int g(int k)const
  {
   return k;
  }

  void alloc();

  void alloc(int a);

  int v1;
  int v2;
 };
 D col,row;
};


void S::D::alloc ()
{
}

static void func ( int& x, int& y, int& x1, int& y1, int x2, int y2,
     int x3, int y3, int x4, int y4
      )
{
}


const S& S::f(Rect& r, Rect& s)const
{
 int t, l, row_section_bottom, n;
 r.top = row.g(r.top);
 s.top = t !=0 && r.top<row.v1 ? row.v1 : r.top;

 r.left = col.g(r.left);
 s.left = l != 0 && r.left< col.v1 ? col.v1 : r.left;

 r.bottom = row.g(r.bottom);
 s.bottom = row_section_bottom == 1  && r.bottom > row.v1 + row.v2 ?
  row.v1+row.v2 : r.bottom;

 r.right = col.g(r.right);
 s.right = n == 1  && r.right> col.v1 + col.v2 ?
  col.v1+col.v2 : r.right;
 return *this;
```

```
}
```

Add the code as a separate cpp file in a CtrlLib application, with it current, press Ctrl+I to reformat it. The file before reformat compiles fine, not the reformatted one.

Works for me in windows and works in Ubuntu. Unfortunately, this feature is now using clang-format that can be different per distro...

Would be nice to give me a hint which host platform is in use...

Also, if nothing helps, please post reformatted text as well.

Mirek

Operation System: Ubuntu 24.04.1 LTS
GNOME version: 46
Windowing System: Wayland
clang-format --version: Ubuntu clang-format version 18.1.3 (1ubuntu1)

Reformatted output

```
#include <CtrlLib/CtrlLib.h>
using namespace Upp;

class S {
 void Set(Size& sz, int x, int y, int z, int x1, int y1, int z1);

 const S& f(Rect& r, Rect& s) const;

 struct D {
  int s(int rc) const { return rc * 2; }

  int e(int rc) const { return rc * 1; }

  int w(int rc) const { return rc * 3; }

  int t() const { return 4; }

  int g(int k) const { return k; }

  void alloc();

  void alloc(int a);

  int v1;
```

```
  int v2;
 };
 D col, row;
};

void S::D::alloc() {}

static void func(int& x, int& y, int& x1, int& y1, int x2, int y2, int x3, int y3, int x4,
                 int y4)
{
}

const S& S::f(Rect& r, Rect& s) const
{
 int t, l, row_section_bottom, n;
 r.top = row.g(r.top);
 s.top = t != 0 && r.top < row.v1 ? row.v1 : r.top;
 r.left = col.g(r.left);
 s.left = l != 0 && r.left< col.v1 ? col.v1 : r.left;
 s.left = l != 0 && r.left < col.v1 ? col.v1 : r.left;
 r.bottom = row.g(r.bottom);
 s.bottom = row_section_bottom == 1  && r.bottom > row.v1 + row.v2 ?
 s.bottom =
  row_section_bottom == 1 && r.bottom > row.v1 + row.v2 ? row.v1 + row.v2 : r.bottom;
 r.right = col.g(r.right);
 s.right = n == 1  && r.right> col.v1 + col.v2 ?
 s.right = n == 1 && r.right > col.v1 + col.v2 ? col.v1 + col.v2 : r.right;
}
```

Line 46 (?), Line 48 ending (;), LIne 50 (?), Line 51 ending (;) are highlighted by theide(libclang) to indicate grammer errors.

OK, after a bit of thinking I have added some more code to logs branch

https://github.com/ultimatepp/ultimatepp/commit/04c14131cc46 99800ca6fa74421858c26cf3eb43

Can you reproduce the problem and send me those files?

Mirek

Subject: Re: 2024rc1
Posted by Lance on Tue, 15 Oct 2024 13:18:26 GMT
View Forum Message <> Reply to Message

mirek wrote on Tue, 15 October 2024 01:40
IDK, there is just 1 .tpp file in .zip (Draw_en-us.tpp) and it seems to be unchanged from the master...

Sorry my bad. Please try this one instead.

File Attachments
1) tpps.zip, downloaded 150 times

Subject: Re: 2024rc1
Posted by Lance on Tue, 15 Oct 2024 22:11:58 GMT
View Forum Message <> Reply to Message

mirek wrote on Tue, 15 October 2024 07:31
OK, after a bit of thinking I have added some more code to logs branch

https://github.com/ultimatepp/ultimatepp/commit/04c14131cc4699800ca6fa74421858c26cf3eb43

Can you reproduce the problem and send me those files?

Mirek

Hello Mirek,

Please see attached.

It appears the one written out is different from what's displaying in theide.

File Attachments
1) reformat.zip, downloaded 141 times

Subject: Re: 2024rc1
Posted by mirek on Wed, 16 Oct 2024 09:30:54 GMT
View Forum Message <> Reply to Message

Lance wrote on Wed, 16 October 2024 00:11mirek wrote on Tue, 15 October 2024 07:31
OK, after a bit of thinking I have added some more code to logs branch

https://github.com/ultimatepp/ultimatepp/commit/04c14131cc4699800ca6fa74421858c26cf3eb43

Can you reproduce the problem and send me those files?

Mirek

Hello Mirek,

Please see attached.

It appears the one written out is different from what's displaying in theide.

Perfect. I believe it is now fixed in the master, please check.

Mirek

Subject: Re: 2024rc1
Posted by Lance on Wed, 16 Oct 2024 13:18:47 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 16 October 2024 05:30
Perfect. I believe it is now fixed in the master, please check.

Mirek

Yes, it is.

Subject: Re: 2024rc1
Posted by mirek on Thu, 17 Oct 2024 08:47:14 GMT
View Forum Message <> Reply to Message

Lance wrote on Tue, 15 October 2024 15:18mirek wrote on Tue, 15 October 2024 01:40
IDK, there is just 1 .tpp file in .zip (Draw_en-us.tpp) and it seems to be unchanged from the master...

Sorry my bad. Please try this one instead.

Reproduced and hopefully fixed in the master... Please confirm.

(Codereference at the very last paragraph of tpp text was the key ingredient I did not reproduce properly based on original bug report.)

Mirek

P.S.: Thanks for your patience, hugely appreciated...

Subject: Re: 2024rc1
Posted by Lance on Thu, 17 Oct 2024 19:01:35 GMT

mirek wrote on Thu, 17 October 2024 04:47
Reproduced and hopefully fixed in the master... Please confirm.

Yes, it works perfectly now!

Quote:
P.S.: Thanks for your patience, hugely appreciated...
It's my pleasure. Thank you for all the efforts!

BTW, now the master branch also need the DEBUGCODE flag to compile (in release mode) because of code like these

```
void CodeEditor::Paint(Draw& w)
{
 DLOG(Format("====at %` ==========", GetSysTime()));
  DDUMP(GetScreenView());
  DDUMP(GetScreenRect());
```

Subject: Re: 2024rc1
Posted by Lance on Mon, 21 Oct 2024 00:11:04 GMT

Lance wrote on Thu, 17 October 2024 15:01
BTW, now the master branch also need the DEBUGCODE flag to compile (in release mode) because of code like these

```
void CodeEditor::Paint(Draw& w)
{
 DLOG(Format("====at %` ==========", GetSysTime()));
  DDUMP(GetScreenView());
  DDUMP(GetScreenRect());
```

That's a misinformation. My local copy got screwed up. I have recreated it. That may mean the subseqent freeze reports might be faulty. I have recompiled logs/ide, and started testing. Sorry for possibly unnecessary frustrations it might have caused.

BTW, I have encounter this code in CtrlLib/EditField.cpp line 151

```
int EditField::GetTextCx(const wchar *txt, int n, bool password, Font fnt) const
{
 if(password)
  return n * font['*'];
 const wchar *s = txt;
```

```
 int x = 0;
 while(n--)
  x += GetCharWidth(*s++);
 return x;
}
```

Is the passed in parameter fnt supposed to be there? Was this function originally intended to be static and using passed in Font to do calculation istead?

Subject: Re: 2024rc1
Posted by Lance on Mon, 21 Oct 2024 00:27:41 GMT
View Forum Message <> Reply to Message

with the u++ moving to c++17,

code like this (Core/Vcont.h line 13)

```
 void Malloc(size_t size) {
  if(std::is_trivially_destructible<T>::value)
   ptr = (T *)MemoryAlloc(size * sizeof(T));
  else {
   void *p = MemoryAlloc(size * sizeof(T) + 16);
   *(size_t *)p = size;
   ptr = (T *)((byte *)p + 16);
  }
 }
```

can benefit from constexpr-if compile time trimming to produce more compact and faster binary (theoretically). I was wondering if u++ is open to such minor, insignificant improvements.

I just happen to encounter these lines :lol:

Subject: Re: 2024rc1
Posted by mirek on Mon, 21 Oct 2024 07:05:24 GMT
View Forum Message <> Reply to Message

Lance wrote on Mon, 21 October 2024 02:27with the u++ moving to c++17,

code like this (Core/Vcont.h line 13)

```
 void Malloc(size_t size) {
  if(std::is_trivially_destructible<T>::value)
   ptr = (T *)MemoryAlloc(size * sizeof(T));
  else {
```

```
   void *p = MemoryAlloc(size * sizeof(T) + 16);
   *(size_t *)p = size;
   ptr = (T *)((byte *)p + 16);
  }
 }
```

can benefit from constexpr-if compile time trimming to produce more compact and faster binary (theoretically).

How? I have noticed that some people tend to constexpr to everything, but I fail to see a reason. If that is supposed to perform the test only in compile time, then 30 years old compiler will do that anyway. But I might be missing something perhaps?

Subject: Re: 2024rc1
Posted by Lance on Mon, 21 Oct 2024 10:55:22 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 21 October 2024 03:05
How? I have noticed that some people tend to constexpr to everything, but I fail to see a reason. If that is supposed to perform the test only in compile time, then 30 years old compiler will do that anyway. But I might be missing something perhaps?

On a second thought, you are right. A reasonably good compiler would perform the optimization anyways. I remeber a couple years ago when I dig into u++ memory allocation utilities, I saw this one

```
template <class T>
void memcpy_t(void *t, const T *s, size_t count)
{
 if((sizeof(T) & 15) == 0)
  memcpy128(t, s, count * (sizeof(T) >> 4));
 else
 if((sizeof(T) & 7) == 0)
  memcpy64(t, s, count * (sizeof(T) >> 3));
 else
 if((sizeof(T) & 3) == 0)
  memcpy32(t, s, count * (sizeof(T) >> 2));
 else
 if((sizeof(T) & 1) == 0)
  memcpy16(t, s, count * (sizeof(T) >> 1));
 else
  memcpy8(t, s, count * sizeof(T));
}
```

Now I know you already counted on the compile time optimization.

Then the question becomes: what constexpr-if has to offer? Maybe speak a programmer's intention more explicitly and thus possible compiler check, like override?

Subject: Re: 2024rc1
Posted by Lance on Mon, 21 Oct 2024 11:01:10 GMT
View Forum Message <> Reply to Message

I take back the hint about code size and efficiency difference. But personally I would use if constexpr () in this and similar cases.

Quote:
For optimization purposes, modern compilers will generally treat non-constexpr if-statements that have constexpr conditionals as if they were constexpr-if-statements. However, they are not required to do so.

A compiler that encounters a non-constexpr if-statement with a constexpr conditional may issue a warning advising you to use if constexpr instead. This will ensure that compile-time evaluation will occur (even if optimizations are disabled).

Subject: Re: 2024rc1
Posted by mirek on Mon, 21 Oct 2024 11:21:50 GMT
View Forum Message <> Reply to Message

Lance wrote on Mon, 21 October 2024 13:01I take back the hint about code size and efficiency difference. But personally I would use if constexpr () in this and similar cases.

Quote:
For optimization purposes, modern compilers will generally treat non-constexpr if-statements that have constexpr conditionals as if they were constexpr-if-statements. However, they are not required to do so.

A compiler that encounters a non-constexpr if-statement with a constexpr conditional may issue a warning advising you to use if constexpr instead. This will ensure that compile-time evaluation will occur (even if optimizations are disabled).

IDK. I am afraid of another mass purge of all U++ sources adding constexpr everywhere for no good reason... Not in this release to be sure.

Subject: Re: 2024rc1
Posted by mirek on Mon, 21 Oct 2024 12:45:52 GMT
View Forum Message <> Reply to Message

Lance wrote on Mon, 21 October 2024 13:01

A compiler that encounters a non-constexpr if-statement with a constexpr conditional may issue a warning advising you to use if constexpr instead. This will ensure that compile-time evaluation will occur (even if optimizations are disabled).

Well, thinking about it, I guess actually the real benefit for me would be something else: compiler issues an error if the expression you marked constexpr is not...

Subject: Re: 2024rc1
Posted by Lance on Mon, 21 Oct 2024 16:12:11 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 21 October 2024 08:45Lance wrote on Mon, 21 October 2024 13:01

A compiler that encounters a non-constexpr if-statement with a constexpr conditional may issue a warning advising you to use if constexpr instead. This will ensure that compile-time evaluation will occur (even if optimizations are disabled).

Well, thinking about it, I guess actually the real benefit for me would be something else: compiler issues an error if the expression you marked constexpr is not...

Agreed. Like "override", make a programmer's intention more explicit, and do, more important than but similar, compiler check when it doesn't going as claimed by the programmer.

Reminds me of a related case, where constexpr seems to be helpful or possibly necessary.

```
union Flags{
 int32 dummy;
 struct{
  byte borderLeft  :3;
  byte borderRight :3;
  byte borderTop   :3;
  byte borderBottom:3;
  byte halign      :2;
  byte valign      :2; //16th bit

  bool faceNotNull     :1;
  bool boldNotNull     :1;
  bool heightNotNull   :1;
```

```
  bool widthNotNull    :1;
  bool underlineNotNull:1;
  bool italicNotNull   :1;
  bool strikeoutNotNull:1; //23rd bit
 };

 constexpr Flags() : dummy(0){ static_assert(sizeof(*this)==sizeof(dummy)); }


 static constexpr int32 FontMask()
 {
  Flags f;
  f.faceNotNull = true;
  f.boldNotNull = true;
  f.heightNotNull = true;
  f.widthNotNull = true;
  f.underlineNotNull = true;
  f.italicNotNull = true;
  f.strikeoutNotNull = true;
  return f.dummy;
 }
};
```

It's a somewhat contrived example. I am not sure ,for int32 FontMask(), if I don't it constexpr, will the code be compiled same as if I do. It's totally possible they do with todays smart and agressive as crazy compiler optimization.
