Subject: U++ state
Posted by andrei_natanael on Sat, 09 Jan 2010 00:34:30 GMT
View Forum Message <> Reply to Message

Hi,
I'm with U++ since December 13, 2007(bytefield) so I think I've gained a bit of experience with it even if I've only done programs which solve my problems, projects for university and for a non-profit organization. I have some reasons for sticking with U++:

U++ use BSD license
statical linkage
have good examples how to do things
it's *almost complete* cross-platform
have good performance
developers dedicated to work on it
friendly community
some good widgets (ArrayCtrl, GridCtrl)
is quite easy to create new widgets
(almost)have a theming engine
it doesn't impose a single work-flow (i.e. widgets should be allocated only on heap or stack)
it's a project where I've learned a lot of new things
keep increasing usage from new developers

But not everything from U++ is as I wanted so I'll explain "negative" sides which keeps me worry about.

Documentation if exists it's sometime outdated
Dynamic linking to U++ libraries is hard
It only support Windows(+CE), Linux & BSD (IMO 2 platforms if we count X11 as platform instead of Linux & BSD), it should support MacOS to be cross-platform
IMO Chameleon is a good design (ChStyle stuff) but data acquisition for it is a bit of mess because it's not providing the same API for different platform i.e. we have XpImage for Windows, GetGTK for gtk+ **
I find hard(or limited)to create an advanced interface without using layouts(sizers) **
Look and feel is incomplete, for example Scrollbars in Windows Vista and 7 have a special behavior (the buttons from heads are highlighted when mouse is over thumb), U++ implementation of menu for gtk+ is using Windows behavior, if there is not enough vertical space it move a part from menu at a side, U++ doesn't disable Scrollbar head button if the thumb is near it(gtk+) and may I continue with many other aspects. **
It doesn't support receiving events like "theme changed" or "DPI changed" from gtk+/gnome (and partly from Windows) so you have to restart your U++ program in order to use new settings
IMO (probably I'm wrong here) U++ choose bad operator = for PICK, it should do what it say "equality" and that means that what is in one side is in other side too, i would use <<= (deep copy?) operator for PICK so you should not have to invent hacks to avoid picking if you didn't want to use it (is that done to have picking for function returned value?)
Even if macros make our work easier(to acquire RAD) i think there are too much macros in a modern framework as U++ and they hide portions of code making it less readable. I'm pro readability even if that means writing 10 chars or more to get it, let's count some macros:

THISBACK, PTEBACK, INITBLOCK, EXITBLOCK, __countof, NTL_MOVEABLE, FN*, ONCELOCK, INTERLOCKED, CH_STYLE, CH_COLOR, GUI_APP_MAIN, CONSOLE_APP_MAIN and all these macros are from developer space not from U++ core developer space which contain more macros which make core unreadable in some portions (i.e. code responsible for IML files, LAY files, DLI), IMO there are nicer solutions to solve problems.

I know that everyone have limited time and I don't expect any change to come from someone but I'm putting these here to know what to work on in future to have a better U++.

** are stuff which I'm saying that I will work on when have time, but always happen to run out of time

Andrei

P.S.: If you have different view on some stuff feel free to post your opinion. I want to hear what others believe about these.

Subject: Re: U++ state
Posted by mdelfede on Sat, 09 Jan 2010 10:24:41 GMT
View Forum Message <> Reply to Message

andrei_natanael wrote on Sat, 09 January 2010 01:34Hi,
.....
[*] IMO (probably I'm wrong here) U++ choose bad operator = for PICK, it should do what it say "equality" and that means that what is in one side is in other side too, i would use <<= (deep copy?) operator for PICK so you should not have to invent hacks to avoid picking if you didn't want to use it (is that done to have picking for function returned value?)

I disagree here, the pick behaviour is one of the strength of UPP, replacing the operator= with another operator would drop its usage to nil.
Imagine a newbie, it'll use the '=' everywhere, so no pick, slow code and would never learn its advantages. Now the worst that can happen is a 'broken pick semantics' exception, he'll be forced to look into manual and learn its usage... or stick back to operator<<= if he don' want to pick.

Quote:
[*] Even if macros make our work easier(to acquire RAD) i think there are too much macros in a modern framework as U++ and they hide portions of code making it less readable. I'm pro readability even if that means writing 10 chars or more to get it, let's count some macros: THISBACK, PTEBACK, INITBLOCK, EXITBLOCK, __countof, NTL_MOVEABLE, FN*, ONCELOCK, INTERLOCKED, CH_STYLE, CH_COLOR, GUI_APP_MAIN, CONSOLE_APP_MAIN and all these macros are from developer space not from U++ core developer space which contain more macros which make core unreadable in some portions (i.e. code responsible for IML files, LAY files, DLI), IMO there are nicer solutions to solve problems.
[/list]

I'm usually also against a strong macro usage, but those in UPP seems to me well placed and

really useful.
They do hyde some implementation details, but usually implementations that don't need to be known in depth.
THISBACK in particular is one of the most clever stuffs I've seen, and much more self-explaining than it's equivalent code.

If you want to see a really ugly, bloated of macro code example, look at OpenCascade CAD library..... No namespaces, macros to do every sort of stuffs that could be much better done with templates, some polymorph classes created by aid of macros and conditional inclusions (sigh).
It has thousands of include's, which make compilation speed a pain, and BLITZ almost unusable because of all above stuffs.

Max

Subject: Re: U++ state
Posted by dolik.rce on Sat, 09 Jan 2010 10:35:17 GMT
View Forum Message <> Reply to Message

Hi Andrei,
Just my quick humble opinions about some of the topics:
andrei_natanael wrote on Sat, 09 January 2010 01:34 IMO (probably I'm wrong here) U++ choose bad operator = for PICK, it should do what it say "equality" and that means that what is in one side is in other side too, i would use <<= (deep copy?) operator for PICK so you should not have to invent hacks to avoid picking if you didn't want to use it (is that done to have picking for function returned value?)A lot of languages (maybe most of them) actually uses the symbol "=" for "assignment" (and "==" for "equality"). Picking is a type of assignment too and I (and U++ devs probably too) think that it's used most of the time, therefore the short operator - it saves time writing and makes the code faster without programmer thinking about it. If you look for example at python, you'll see that it makes shallow copy on "=" and if you need deep copy you need much longer syntax. So U++ is not really that special in this approach. Also the 'length' of "<<=" operator IMHO nicely suggest that it takes longer to perform the operation

andrei_natanael wrote on Sat, 09 January 2010 01:34
Even if macros make our work easier(to acquire RAD) i think there are too much macros in a modern framework as U++ and they hide portions of code making it less readable. I'm pro readability even if that means writing 10 chars or more to get it, let's count some macros: THISBACK, PTEBACK, INITBLOCK, EXITBLOCK, __countof, NTL_MOVEABLE, FN*, ONCELOCK, INTERLOCKED, CH_STYLE, CH_COLOR, GUI_APP_MAIN, CONSOLE_APP_MAIN and all these macros are from developer space not from U++ core developer space which contain more macros which make core unreadable in some portions (i.e. code responsible for IML files, LAY files, DLI), IMO there are nicer solutions to solve problems.Some of those macros hide platform differences (e.g. GUI_APP_MAIN) which is IMO very good reason for macro. Others (e.g. THISBACK) increase a readability of the code a lot, which is also very helpful.

Apart from those two points I mostly agree with you. U++ is a great project and helps me a lot even though it still has few glitches.

Best regards,
Honza

Subject: Re: U++ state
Posted by mirek on Sat, 09 Jan 2010 11:30:40 GMT
View Forum Message <> Reply to Message

andrei_natanael wrote on Fri, 08 January 2010 19:34
[*] IMO (probably I'm wrong here) U++ choose bad operator = for PICK, it should do what it say "equality" and that means that what is in one side is in other side too, i would use <<= (deep copy?) operator for PICK so you should not have to invent hacks to avoid picking if you didn't want to use it (is that done to have picking for function returned value?)

The problem there is you have to use copy constructor in order to make pick work for function returns. No way around it.

I am not 100% happy with it. But that is what C++ gives us.

Maybe some day we could try &&. But there is still that ugly issue with composition rules (&& does not automatically get generated for classes). Maybe we could find a way around.

Quote:
[*] Even if macros make our work easier(to acquire RAD) i think there are too much macros in a modern framework as U++ and they hide portions of code making it less readable. I'm pro readability even if that means writing 10 chars or more to get it, let's count some macros: THISBACK, PTEBACK, INITBLOCK, EXITBLOCK, __countof, NTL_MOVEABLE, FN*, ONCELOCK, INTERLOCKED, CH_STYLE, CH_COLOR, GUI_APP_MAIN, CONSOLE_APP_MAIN and all these

If macro saves time and (first of all) errors, let us use it. Just because sometimes it is argued that (over)using macros in some contexts(!) is bad does not automatically makes them all bad. IMHO.

Quote:
IMO there are nicer solutions to solve problems.

Depends on definition of nicer...

Quote:
I know that everyone have limited time and I don't expect any change to come from someone but I'm putting these here to know what to work on in future to have a better U++.

Sure, you are welcome. I have only commented points where I disagree:)

Subject: Re: U++ state
Posted by mirek on Sat, 09 Jan 2010 11:41:53 GMT
View Forum Message <> Reply to Message

andrei_natanael wrote on Fri, 08 January 2010 19:34
[*] IMO Chameleon is a good design (ChStyle stuff) but data acquisition for it is a bit of mess because it's not providing the same API for different platform i.e. we have XpImage for Windows, GetGTK for gtk+ **
[*] Look and feel is incomplete, for example Scrollbars in Windows Vista and 7 have a special behavior (the buttons from heads are highlighted when mouse is over thumb), U++ implementation of menu for gtk+ is using Windows behavior, if there is not enough vertical space it move a part from menu at a side, U++ doesn't disable Scrollbar head button if the thumb is near it(gtk+) and may I continue with many other aspects. **
[*] It doesn't support receiving events like "theme changed" or "DPI changed" from gtk+/gnome (and partly from Windows) so you have to restart your U++ program in order to use new settings

Well, maybe I should try to refactor relevant parts.

The current messy code is partly because Ch at the beginning looked relatively simple, there was no need for overengineered code (I am now speaking about ChWin32.cpp and ChGtk.cpp) - but I had understimated the issue, things has started getting pretty complicated... And, especially with Gtk, getting the relevant data is sort of black magic...

Other than that, I am afraid that the look&feel will always be a compromise. For me, the benchmark is FireFox and OpenOffice - both are using the same method as U++ (having their implementation and using host platform just for painting). I believe if nobody really complains about FF or OO look&feel, U++ should be OK as well.

In this benchmark, I think we are doing acceptable if not quite well...

(One important step is I should finally switch to Win7 - then I will start noticing differences

Mirek

Subject: Re: U++ state
Posted by andrei_natanael on Sat, 09 Jan 2010 21:52:43 GMT
View Forum Message <> Reply to Message

Hi everyone,
Now things seems more clear.

Mirek wroteOther than that, I am afraid that the look&feel will always be a compromise. For me, the benchmark is FireFox and OpenOffice - both are using the same method as U++ (having their implementation and using host platform just for painting). I believe if nobody really complains about FF or OO look&feel, U++ should be OK as well.
Well, U++ may look like firefox but the behavior is not the same. Just compare U++ with Firefox at these parts:
Scrollbars, Tabs, Menus, Buttons. A little description about differences:
Scrollbars - as i said, gtk+ disable the "heads" of scrollbars when the thumb touch the head and when the button from head is pushed it doesn't have a press offset. Firefox mimic it perfectly, U++ doesn't do that.
Tabs - if they have a close button it usually have a gray cross, in U++ it's a red one (well, the button is red).
Menus - when they don't fit horizontally gtk+ add 2 arrows at top and bottom and when mouse is over one arrow the menu is scrolled. If a menu item is disabled it doesn't have a mouse over effect. U++ ignore these and also ignore usage of icons in menu. You know that gtk+ have an option to disable icons in menus.
Buttons - in gtk+ default buttons have a special look and the focus is not a dashed rectangle (as it's in U++ now), sometime it's a highlighted button (depend on theme), highlighted switch, option, tab, etc.

Well there are many more glitches but i don't want just to sit and talk without fixing them and as is seen that some people can live with that (i can live with that too but i have only best wishes to U++, so that's why I'm not ignoring them), so when i have the necessary time to fix them i will.

About Menus, Firefox still have problems with menus text color with some themes, U++ too.

Regarding theming have you heard about plans for gtk3? Will these affect U++ in some way? I think they will change core theming API to support new features, AFAIK they will introduce css-ed theming support and i thing that will shake U++ a bit.