Subject: Re: What next?
Posted by cbpporter on Sat, 19 Dec 2015 13:40:02 GMT
View Forum Message <> Reply to Message

I saw this thread when it was created, but it took me a few days to gather my thoughts on the subject. Doubly so, since I'm almost in the same boat, but probably where U++ was 10+ years ago: I too am developing a solution for some perceived problems taking the form of a library (no GUI though), an endeavor mostly fueled by dissatisfaction with current popular solutions. I am still in the development phase, but I'm dreading the day I will actually have to promote the project.

Because promotion is very hard, especially in this domain. Really really hard. So much harder than writing code :), which is easy and fun and has deterministic outcome. And especially now, when C++ is no longer as common as it used to be. Everybody is using JS now and we can't even hire competent new people at my workplace. The "if you build it, they will come" approach does not work. Consistently outputting a quality product does not attract people. It may be great at retaining people, but the first exposure most come from somewhere else.

U++ is a great tool with great productivity and generally a good design, but is is also often weird, quirky and some parts of it are downright alien. If you ask me, U++ is equally well designed and well designed to drive off random enthusiasts who just heard of the project and want to give it a quick try. The barrier of entry is too high.

I think that this is mostly due to TheIDE and a lack of structure that puts some of the best parts of U++ to the forefront.

U++ has great value as a GUI framework. But GUI work is relatively rare in this day and age. But it also has huge amounts of value outside of GUI. I love CodeEditor as an example, because it saved me from ritting thousand of lines of code, but you won't attract new people with CodeEditor or ArrayCtrl. You'll do it with Core at first on the non GUI side and with simple GUI samples that show of accessibility. GUI can be complicated and downright threatening looking in some toolkits, but creating a TopWindow (which should be called Window anyway) and plopping down a few Buttons can be inviting.

But I still think Core needs special attention. Core is super valuable but quite inaccessible.

Here is an example: I don't think that there is a seasoned C++ developer out there who really loves std::vector. Most people that I know hate it and iterators with pointer syntax in general. U++ has Vector, which is a lot more pleasant to use and potentially faster. I do not understand why after so many years, a random person can't just pick a random C++ source file, add a #include <upp/Vector> or something, and happily go on and use Vector. No mess, no fuss, no dependencies, no pulling in full packages, no random allocator, no almost proprietary IDEs. Or String. Or VectorMap. People do need hash maps all the time. std::map is so lacking. Or CParser. CParser is pure gold! I don't like pointless hyperbole, but CParser did change a small part of my life for the better! It is ridiculously convenient to have this power in a few lines of code without having to use some lexer lib and grammar. In a language notorious for having zero parsing ability without blowing up some pointers and barely being able to convert an int to a string. Or Socket. Very useful!

The contents of U++ should be filtered and a "core core" should be made very accessible. Not everything that is in the packages though. Just the super useful high quality stuff. Color is nice and I guess it does the job, but it is not a selling point. My Color lib is 40k code of code without conversion functions and I still find it lacking, so a simple Color class is not an attraction like the containers, serialization, Xml support, Unicode and so on.

Now I'm not saying that we no longer use packages or TheIDE. I'm saying the message for new initiates should be "U++ Core is awesome because these classes are great. You can use TheIDE for a modular approach at developing programs, but first let me show you how awesome Vector is using the Linux command line/CMake/Visual Studio. You'll love it and never touch std::vector again! And BTW, Vector is compatible with std, so you can still mix it with other code!".

Neither am I saying we break up Core into 2 separately maintained code bases. There is probably a way to reorganize sources so that stand alone classes can be included with standard C++ paradigms (some thought must be put into how to break up libs for non templates, probably the boost model) one by one and still have Core.h which resides in a U++ package and provides the entire functionality.