Subject: Basic questions about u++
Posted by irtech on Tue, 06 Oct 2009 12:14:24 GMT
View Forum Message <> Reply to Message

Hello,

I'm a MFC programmer, I just heard about u++ a few hours ago and I visited the site but I have some questions that I couldn't find a direct answer to it. Actually I was seriously thinking about moving to Qt. I love c++ and I'm resisting to migrate to .Net and C# not because they are bad but because I don't like to throw away things that I learned so far about c++.

Please forgive me if my questions are very elementary.

1- is u++ considered native or managed code? I mean should we install a framework(as framework name implies) on target machine to be able to run u++?

2- Is it open source and free? I assume it is because of BSD license.

3- How big is community? Which companies, groups support it?

4- Is there a plan to make u++ available on embedded devices like Qt?

5- what are Pros/Cons in comparison to Qt?

6- what are Pros/Cons in comparison to MFC?

Thank you very much
best regards.

Subject: Re: Basic questions about u++
Posted by mr_ped on Tue, 06 Oct 2009 12:41:43 GMT
View Forum Message <> Reply to Message

1. U++ is C++, i.e. normally native code as long as you don't bend some managed compiler to compile U++ libraries.
U++ is supported either for MS C compiler, or G++ (from GNU GCC), both are classic "native" C++ compilers.

2. the core of U++ is BSD license. There are some optional special packages (SDL and maybe something in bazaar directory) under LGPL or maybe even GPL, so watch out when considering those, but as long as you use basic U++ libs, you are in land of BSD.

3. small but dedicated.

4. there are people trying to run U++ on embedded devices, and the main dev team is trying to keep it as portable as possible, i.e. indirectly supporting any embedded effort. But with some new device it's up to the one who needs it to try to port it and suggest patches afterward.

5. Qt is IMHO lot more bigger. Bigger commercial subject behind it, bigger community, bigger (more bloated?) API + code.
U++ is lean and maybe more C++ like (if you like to use full C++, you will like it, if your code is more C like, you will be maybe not that happy). Both are IMHO very powerful tools and good choices. If you are very skilled and a bit adventurous, I think U++ is better fit, as you are more in control. If you like to be one of many, Qt is probably safer choice.

6. U++ is cross platform, performance wise it's better, not directly tied to MS, and masters of U++ should be much more efficient in writing applications then MFC masters. It should be superior to it almost in every aspect. MFC was standard before MS started to bring out new platform/API every second year, thus pushing devs to move and move...

Subject: Re: Basic questions about u++
Posted by mr_ped on Tue, 06 Oct 2009 13:03:04 GMT
View Forum Message <> Reply to Message

One more thing... U++ is very easy to try out. In windows just download some older (2008.1 ?) release with mingw included, install it, and it's all you need to toy around with it (within couple of minutes).

Under linux it's very easy too, especially under Ubuntu you need just to install the .deb and search forum for list of additional packages (if that linux installation was used for SW development, you would probably already have all of them installed anyway).
But under linux it's also quite easy to try out Qt, I'm not sure about windows, there's probably more hassle with Qt.

Subject: Re: Basic questions about u++
Posted by irtech on Tue, 06 Oct 2009 13:30:52 GMT
View Forum Message <> Reply to Message

Thank you very much mr_ped,

While I also like features of compactness and effectiveness about U++, another concern is development time. I've heard Qt has very rich libs nearly as rich as .Net which facilitates developing big applications, especially GUI applications which need a lot of coding.
About being more C++ like, as far as I understood neither have used stdlib but they can make use of boost lib. So I guess they are alike in this regard. I emphasize that was what I understood about U++ and Qt and could be wrong.
But I guess U++ worth further investigation before I make my final decision.

Regards.

Subject: Re: Basic questions about u++
Posted by zsolt on Tue, 06 Oct 2009 13:51:02 GMT
View Forum Message <> Reply to Message

See the Overview document to start.

Subject: Re: Basic questions about u++
Posted by irtech on Wed, 07 Oct 2009 04:19:43 GMT
View Forum Message <> Reply to Message

I actually have read those overview, comparison,... sections.
For example in comparison section there is a sample program that shows U++ far less coding than Qt, but I like to know is this a general rule or just a specific example which is not true in other cases?
I presume volume of coding depends on richness of libraries. Does this suggest that U++ have richer libraries that Qt?
Richness of libs adversely affect dev time. I like to know is it safe to say that U++ dev time is shorter than Qt in most cases?
I especially mean big GUI applications.

Certainly I am not looking for a perfect dev language that has best performance, best libraries, cleanest code,...
I'm ready to compromise but I like to know what I have to give and in return what I will get.

mr-ped said U++ is more C++ like. I've read that U++ people had good reasons not to use stdlib but I like to know having these reasons in mind how U++ is more C++ like than Qt?

Regards.

Subject: Re: Basic questions about u++
Posted by andrei_natanael on Wed, 07 Oct 2009 05:52:47 GMT
View Forum Message <> Reply to Message

irtech wrote on Wed, 07 October 2009 07:19I actually have read those overview, comparison,... sections.
For example in comparison section there is a sample program that shows U++ far less coding than Qt, but I like to know is this a general rule or just a specific example which is not true in other cases?
It is a general rule. To create a qt application you have to follow the same "pattern" always. You have to create an QApplication class in main function, another class derived from QWidget (or QWindow), etc. In U++ you are not forced to do so and you don't have a main Application class.
Quote:
I presume volume of coding depends on richness of libraries. Does this suggest that U++ have richer libraries that Qt?
Richness of libs adversely affect dev time. I like to know is it safe to say that U++ dev time is shorter than Qt in most cases?

I especially mean big GUI applications.
I think U++ doesn't have riches libraries than Qt but are more efficient and overloading of some operators make you typing less. In some parts U++ API is better structured than Qt API and viceversa.
Quote:
Certainly I am not looking for a perfect dev language that has best performance, best libraries, cleanest code,...
I'm ready to compromise but I like to know what I have to give and in return what I will get.

mr-ped said U++ is more C++ like. I've read that U++ people had good reasons not to use stdlib but I like to know having these reasons in mind how U++ is more C++ like than Qt?
Qt has a "custom" C++. They are using moc(Meta Object Compiler) to handle their extensions to C++. If the following code looks like normal C++ code to you, then yes, Qt is more C++ like than U++

```
 class MyClass : public QObject
 {
    Q_OBJECT

 public:
    MyClass(QObject *parent = 0);
    ~MyClass();

 signals:
    void mySignal();

 public slots:
    void mySlot();
 };
```

My C++ doesn't have signals and public slots keywords. So you have to send your source to moc compiler before sending to C++ compiler, else your Qt code is not valid.
Even if U++ doesn't use stdlib(STL)(for performance reasons) it's more C++ like than Qt.

Subject: Re: Basic questions about u++
Posted by mirek on Wed, 07 Oct 2009 06:29:18 GMT
View Forum Message <> Reply to Message

andrei_natanael wrote on Wed, 07 October 2009 01:52Even if U++ doesn't use stdlib(STL)(for performance reasons)

Actually, performance is not the primary concern.

STL makes you wish C++ had garbage collector.
U++ makes you wish Java/C# had destructors.

(But yes, I believe we are still at least 50% faster than anything else, as long as symbolic

processing is concerned).

Subject: Re: Basic questions about u++
Posted by mirek on Wed, 07 Oct 2009 06:46:28 GMT
View Forum Message <> Reply to Message

irtech wrote on Wed, 07 October 2009 00:19
I presume volume of coding depends on richness of libraries. Does this suggest that U++ have richer libraries that Qt?
Richness of libs adversely affect dev time. I like to know is it safe to say that U++ dev time is shorter than Qt in most cases?
I especially mean big GUI applications.

U++ is designed to handle big GUI apps.

Maybe the library is not as rich, OTOH the primary design concern is extensibility. I believe it takes much less effort to develop a brand new widget in U++.

Quote:
I'm ready to compromise but I like to know what I have to give and in return what I will get.

Well, the hardest part is to learn the stuff. U++ is very different style of programming. IME, people were able to start commercial development in U++ after two or three weeks of study (of course, not on guru level, but still be productive).

At the moment, documentation is still imperfect. But tutorials and 'reference' should get you started quickly:

http://www.ultimatepp.org/srcdoc$Core$CoreTutorial$en-us.htm l
http://www.ultimatepp.org/srcdoc$Core$Tutorial$en-us.html
http://www.ultimatepp.org/srcdoc$CtrlLib$Tutorial$en-us.html
http://www.ultimatepp.org/srcdoc$Draw$DrawTutorial$en-us.htm l
http://www.ultimatepp.org/srcdoc$Draw$ImgTutorial$en-us.html
http://www.ultimatepp.org/srcdocSqltutorial$en-us.html

http://www.ultimatepp.org/www$uppweb$examples$en-us.html

To add to other features you get, I would mention total transparency. With theide, there is no fundamental distinction between your code and U++ library code. You navigate (or fix) both in the same way.

What you give is that theide itself might be a pain in the ass, if you are used to other tools. What you get with theide is compilation speed in debug mode, e.g. theide itself full rebuild, including whole U++ libraries, takes about 30s on quadcode machine. For really big GUI apps, this is

priceless... (before we have invented the technology, this app:

http://www.ultimatepp.org/projects$uppweb$webmap$en-us.html

was rebuilding for 50 minutes. Now it is much more manageable 5 minutes...)

Mirek

Subject: Re: Basic questions about u++
Posted by mr_ped on Wed, 07 Oct 2009 07:04:46 GMT
View Forum Message <> Reply to Message

About rich libraries.
U++ is smaller framework, with focus at GUI and core functions. For example U++ does *not* have web browser component (while Qt has Webkit).
If you do a really big GUI app which needs lot of windows, menus, buttons, and other things which are in U++, I think the development will be always faster then with Qt, and the source will be shorter and to many people it will be easier to read and maintain (so you cut costs not only during development, but also during maintenance phase).
I'm not familiar enough with Qt to tell you how many things it has which are missing in U++. Maybe if you are checking Qt, and you do see something what you find very handy and you may need it for your apps, ask here in forum if it is in U++.

About "more C++ like". I mean, the U++ does use C++ language very aggressively. Overloading operators, very good object API with efficient usage of templates and classes inheritance, etc. If you did learn C++ trough C like me and you didn't dwell deep enough into it, the U++ source can be a bit unreadable at first, until you become familiar with the rest of C++ extensions (which are not that commonly used in other sources).
Also I don't feel like STL is part of C++. I know for many professional programmers "knowing C++" means also using STL fluently, but for me it's just external library which can help, but it never fell very natural for me. I often kept doing things without it in pure C++. U++ NTL does make lot more sense to me, and it's much easier for me to use it. Also using STL now after NTL is usually a pain in ass for me. I find NTL to have much better "taste" (but it can be just personal bias).
So U++ is pure C++ source code, which actually does USE lot of C++ features. It goes so far, that for mediocre C++ programmer it can be more difficult to work with U++ then with other "C++" toolkits which keeps their source more C-like and/or use special things like Qt's moc.

Subject: Re: Basic questions about u++
Posted by mirek on Wed, 07 Oct 2009 07:46:43 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Wed, 07 October 2009 03:04
Also I don't feel like STL is part of C++.

Unfortunately, it is a part of C++ standard.

I always felt like Stepanov basically killed the language

Mirek

Subject: Re: Basic questions about u++
Posted by irtech on Wed, 07 Oct 2009 08:04:45 GMT
View Forum Message <> Reply to Message

Ok, thanks now I'm getting a more clear picture.
mr_ped wrote on Wed, 07 October 2009 09:04
If you do a really big GUI app which needs lot of windows, menus, buttons, and other things which are in U++, I think the development will be always faster then with Qt, and the source will be shorter and to many people it will be easier to read and maintain (so you cut costs not only during development, but also during maintenance phase).

So Qt has faster Dev time, lower costs,... . So can we conclude that for commercial developers Qt is more appropriate that U++ ?

Regards.

Subject: Re: Basic questions about u++
Posted by cbpporter on Wed, 07 Oct 2009 08:14:55 GMT
View Forum Message <> Reply to Message

irtech wrote on Wed, 07 October 2009 11:04Ok, thanks now I'm getting a more clear picture.
mr_ped wrote on Wed, 07 October 2009 09:04
If you do a really big GUI app which needs lot of windows, menus, buttons, and other things which are in U++, I think the development will be always faster then with Qt, and the source will be shorter and to many people it will be easier to read and maintain (so you cut costs not only during development, but also during maintenance phase).

So Qt has faster Dev time, lower costs,... . So can we conclude that for commercial developers Qt is more appropriate that U++ ?

Regards.

I believe you have misread the quote "I think the development will be always faster then with Qt". It says that development will be faster with U++.

Subject: Re: Basic questions about u++
Posted by mr_ped on Wed, 07 Oct 2009 08:45:18 GMT
View Forum Message <> Reply to Message

sorry, a very ugly typo... should have been "than".

To be clear: U++ is faster for basic GUI (buttons, menus, text fields, SQL, etc...)

Subject: Re: Basic questions about u++
Posted by unodgs on Wed, 07 Oct 2009 09:54:51 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Wed, 07 October 2009 04:45sorry, a very ugly typo... should have been "than".

To be clear: U++ is faster for basic GUI (buttons, menus, text fields, SQL, etc...)
Actually, I was doing quite complicated gui in my last app (I'll post later few screenshots from it). Creating custom controls in Upp is extremely easy, so all things I missed I created myself in a very short time. Strong poing of Upp is you can mix/combine controls in any way you want. So there is not a problem to create button with grid inside

Subject: Re: Basic questions about u++
Posted by irtech on Wed, 07 Oct 2009 10:11:48 GMT
View Forum Message <> Reply to Message

Ok, thanks.That will be a big + for U++.

About transparency of code which I think is important I also need to investigate about Qt but because of MOC which andrei_natanael pointed out, I doubt Qt has transparent code. sometimes during debug one needs to take a look under hood to figure out what has caused the problem. MFC also has transparent code which is very helpful; while compiler sometimes can't find the error during debug it is a big advantage that reduces debug time significantly.

Another question is about standardization. which group, committee, NGO,.. is responsible for standardization, or community is still too small to feel the need for standardization?

Another Question is about resource management, is this done in compile time or run time(i.e. garbage collection)
I mean does compiler figure out when the owner is destroyed, the attached resources should be freed and [Add] the code automatically to destructor of the object?

Regards.

Subject: Re: Basic questions about u++

Posted by mirek on Wed, 07 Oct 2009 10:23:20 GMT
View Forum Message <> Reply to Message

irtech wrote on Wed, 07 October 2009 06:11

Another Question is about resource management, is this done in compile time or run time(i.e. garbage collection)
I mean does compiler figure out when the owner is destroyed, the attached resources should be freed and [Add] the code automatically to destructor of the object?

I guess that is the main difference of U++. Everything (or almost everything) is somehow directly or indirectly linked to some stack frame. If the stack frame goes out of scope, object is destryoed via destructor.

E.g.

Quote:
```
{
  FileOut out("test");
  ...
} // File gets closed
```

Quote:
```
{
  Array<FileOut> out;
  ...
} // all files get closed
```

Quote:
```
{
  Array<Window> win;
  ...
} // all windows get closed
```

U++ is designed so that structuring code in this way is very simple...

Subject: Re: Basic questions about u++
Posted by mr_ped on Wed, 07 Oct 2009 11:33:11 GMT
View Forum Message <> Reply to Message

standardization - luzr (Mirek) is the (one of the very few) core developer who is also reading forums, he also does decide which patches from community get included into U++. So far it works

very well. (with obvious fixes there's no problem, with more complex updates you either get accepted or you can put them into Bazaar packages where others may check it out and eventually help with improvements)

Resources works like in ordinary C++, no GC (unless you add some C++ GC library on your own, but I wonder why not java then, if you want GC).
As you can see from Mirek's examples, a C++ is not that bad when it comes to resource managements, if you stay within U++ style of doing things. If you use some "new", then it's up to you to "delete" of course. Also using memory leaks detection and valgrind early can help a lot.
Still in C++ you should always know and understand how you work with your resources, the careless programming in C++ never pays off. U++ is just minimizing the hassle.

Subject: Re: Basic questions about u++
Posted by irtech on Wed, 07 Oct 2009 14:18:29 GMT
View Forum Message <> Reply to Message

I'm confused
Mirek said
If the stack frame goes out of scope, object is destryoed via destructor.

Implying it is done automatically.

but mr_ped said
Resources works like in ordinary C++,...If you use some "new", then it's up to you to "delete" of course.

which clearly rules out any compile time resource management.

So if you can see when a resource life time has ended and should be destroyed then why not compiler check it for you automatically?
If such a technology can be implemented I predict Microsoft will throw away its dearly loved .Net to copy the technology in yet another framework just as they quickly copy pasted Java into their .Net framework.

by the way can this forum be accessed via a news server?
It is much more convenient than http interface.

Regards.

Subject: Re: Basic questions about u++
Posted by chickenk on Wed, 07 Oct 2009 15:46:22 GMT
View Forum Message <> Reply to Message

irtech wrote on Wed, 07 October 2009 16:18by the way can this forum be accessed via a news server?
It is much more convenient than http interface.

Personally I use this URL with Google Reader.

Regards,
Lionel

Subject: Re: Basic questions about u++
Posted by mrjt on Wed, 07 Oct 2009 16:14:33 GMT
View Forum Message <> Reply to Message

irtech wrote on Wed, 07 October 2009 15:18I'm confused
Mirek said
If the stack frame goes out of scope, object is destryoed via destructor.

Implying it is done automatically.

but mr_ped said
Resources works like in ordinary C++,...If you use some "new", then it's up to you to "delete" of course.

The idea of U++ is to utilize the standard behaviour of C++ to avoid most of the memory management problems that normally plague C++ code. This is done in two main ways:
- Structuring the library so that every object is 'owned' by something. This means declaring objects either on the stack (as local variables) or as member variables. This allows the compiler to clean up by itself when either the owning object is destroyed or the variable goes out of scope.
- Providing the tools (mainly the NTL) necessary to control dynamic memory in a way that fits with the 'everything is owned' style. This means that although new and delete are used internally in the library you rarely, if ever, need them in your application code.

A simple example:
'ordinary' C++ code:

```
int CalcSomething {
    int *array = new int[256]

    // Do a load of stuff

    delete[] array;
};
```

U++:

```
int CalcSomething {
    Buffer<int> array(256); // Buffer is closest to a C array

    // Do a load of stuff
```

};

By wrapping the new/delete calls in an object (one that has already been thoughly tested) you are able to utilise C++ inbuilt destruction mechanics and avoid the error-prone call to delete. There isn't any 'magic' going on outside of using templates in a clever way

As far as I'm concerned the Upp memory management philosophy is to never use new/delete. There is almost always a better way .

Subject: Re: Basic questions about u++
Posted by irtech on Wed, 07 Oct 2009 16:47:40 GMT
View Forum Message <> Reply to Message

mrjt wrote on Wed, 07 October 2009 18:14
By wrapping the new/delete calls in an object (one that has already been thoughly tested) you are able to utilise C++ inbuilt destruction mechanics and avoid the error-prone call to delete. There isn't any 'magic' going on outside of using templates in a clever way

As far as I'm concerned the Upp memory management philosophy is to never use new/delete. There is almost always a better way .

Ok thanks with your explanation and what I've read in overview now I understood the resource management philosophy of U++!

except one thing !
Ok your example is about a simple int but hw about a big object like a class? Then you certainly need copy constructor which seems to be the motive not to use stdlib. but not having control over copy constructor isn't a drawback by itself?
I mean for example I want to specifically copy value of a sibling object when copy constructor is called or I want to increment a variable inside a class inside copy constructor so I know how many times it has been copied. How should I implement them? I'm not saying it is a routine task but many times you want to do specialized tasks in copy constructor. Then how you do it?

Regards.

Subject: Re: Basic questions about u++
Posted by mrjt on Wed, 07 Oct 2009 17:07:30 GMT
View Forum Message <> Reply to Message

Well, here it is a little more complicated and is difficult to explain.

Basically there is a distinction between moveing and object and a deep copy of the object. This

allows various memory operations to work more efficiently and nice things like returning a Vector<> object from a function. A detailed explanation is here(also follow the link to 'pick semantics'

I'll give you an example tomorrow if you need it, but I've got to run now or I'll miss my train

---

Subject: Re: Basic questions about u++
Posted by mirek on Wed, 07 Oct 2009 17:14:04 GMT
View Forum Message <> Reply to Message

irtech wrote on Wed, 07 October 2009 12:47mrjt wrote on Wed, 07 October 2009 18:14
By wrapping the new/delete calls in an object (one that has already been thoughly tested) you are able to utilise C++ inbuilt destruction mechanics and avoid the error-prone call to delete. There isn't any 'magic' going on outside of using templates in a clever way

As far as I'm concerned the Upp memory management philosophy is to never use new/delete. There is almost always a better way .

Ok thanks with your explanation and what I've read in overview now I understood the resource management philosophy of U++!

except one thing !
Ok your example is about a simple int but hw about a big object like a class? Then you certainly need copy constructor which seems to be the motive not to use stdlib.

Actually, in most cases, you do NOT need copy constructor.

Quote:
I mean for example I want to specifically copy value of a sibling object when copy constructor is called or I want to increment a variable inside a class inside copy constructor so I know how many times it has been copied.

Well, you have 3 kinds of entities:

- "object" (I lack better word) classes like File, Window etc... These usually do not have copy.

- "value" types like int, String, Color etc... These usually have full deep copy semantics

- and, U++ specific "containers". These act as sort of structural glue binding everything together.

And these containers, to workaround possibly missing copy contructors in cointained objects, have so called "pick transfer semantics":

http://www.ultimatepp.org/srcdoc$Core$pick_$en-us.html

which is the most "alien" thing in U++.

In reality, we could probably even live without pick, it is sort of similar in importance to "break" statement in C(++/#)/Java. But it makes live easier.

Mirek

Subject: Re: Basic questions about u++
Posted by mr_ped on Wed, 07 Oct 2009 19:28:53 GMT
View Forum Message <> Reply to Message

The C++ compiler does call destructors when the object goes out of scope. The resources used by that object are not freed magically, you have to write the destructor code. Then it is "magically" called whenever that object goes out of scope.

Probably everything you will use from U++ does already have proper destructor, so you can write it as Mirek posted, you just create new object on heap, and when you go out of scope, you know it was destroyed properly.

In case you use "new" in code (and really can't avoid it, which is usually quite easy with U++ way of doing things), you have either to detect it in destructor and call appropriate "delete" (a good idea which keeps your custom classes to behave same as U++ things) or you have to watch out during that scope and call "delete" by hand whenever it is needed (going out of scope) (quite error prone).

C++ of course calls ctors/dtors automagically (inserting those calls at proper places during compilation) just like you would expect it (well, almost) (see C++ standard), so if you manage your resources this way, you can avoid many common bugs which usually arise from new/delete in function code.
But in the end it's yours (and U++) code responsibility to free everything correctly. If you do it in U++ way, you will very rarely have to think about it, it will work almost "alone".
But still you should understand how it works and why, so you don't allocate for example 3 big memory blocks at the same time (same scope), when you need just 1 at a time and then you can release it and allocate another one (but this is also problem for GC languages, if you are way too much careless, you will degrade performance of your code tenfold, and GC will not save you).
