Subject: U++ talk
Posted by amrein on Mon, 01 Sep 2008 12:43:25 GMT
View Forum Message <> Reply to Message

It would be so cool to have libupp.so.x.y.z and a clean version number before releasing anything into popular Linux distro. It's not a simple work (need to modify TheIDE, ...).
U++ is a monolith. It's so far from "re-use my lib" policy of all other FOSS software and libraries...
The "build all from source" is very interesting for proprietary companies (and for debugging too) but for FOSS... it's another universe. How can I port the entire Gnome 2.x interface to upp if I don't have a lib to do a thin gnome 3.0 for example?
(Yes, Gnome, because those guys are still using C. They don't have a powerful and thin C++ library to replace GTK+. KDE use Qt.)

Subject: Re: Does the provided upp.spec works for you and on which distro?
Posted by cbpporter on Mon, 01 Sep 2008 13:00:39 GMT
View Forum Message <> Reply to Message

amrein wrote on Mon, 01 September 2008 15:43It would be so cool to have libupp.so.x.y.z and a clean version number before releasing anything into popular Linux distro. It's not a simple work (need to modify TheIDE, ...).
U++ is a monolith. It's so far from "re-use my lib" policy of all other FOSS software and libraries...
The "build all from source" is very interesting for proprietary companies (and for debugging too) but for FOSS... it's another universe. How can I port the entire Gnome 2.x interface to upp if I don't have a lib to do a thin gnome 3.0 for example?
(Yes, Gnome, because those guys are still using C. They don't have a powerful and thin C++ library to replace GTK+. KDE use Qt.)
I wouldn't call it a monolith, since it is very modular. And this is the problem: how to build a single lib? Some apps need just some packages, and the most natural would be to have a so for every package. But that's quite some work and not very convenient.

Also, as Mirek said, we do not have a x.y.x versioning system. Our versioning system boils down practically to "year" (not counting dev releases). And I don't think we have the man power to do such versioning, especially that we have to increment a given number anytime we break link compatibility, which is very easy in C++. I honestly don't know what the solution for this problem is, except to link statically.
Or maybe we consider x 2008, and y 1 .

And I don't understand the part about Gnome? Who wants to port Gnome? And they do have GtkMM, but with all the gobject crap that's in the background, I wouldn't call it thin.

Subject: Re: Does the provided upp.spec works for you and on which distro?
Posted by guido on Mon, 01 Sep 2008 21:32:07 GMT
View Forum Message <> Reply to Message

What is the point, when upp is so tiny, just above a megabyte? In contrast, gtkmm alone is 3

megabytes, when you create a semi-static (shared gtk, static C++ wrapper) hello world app! And upp can do a lot more than anemic bare bones gtk to begin with.
Nah, not having to deal with hundreds of bug reports caused by subtle or not so subtle differences between dependend libraries on the various distros, is what attracted me to upp in the first place.

Subject: Re: Does the provided upp.spec works for you and on which distro?
Posted by mirek on Tue, 02 Sep 2008 08:06:59 GMT
View Forum Message <> Reply to Message

amrein wrote on Mon, 01 September 2008 08:43It's so far from "re-use my lib" policy of all other FOSS software and libraries... The "build all from source" is very interesting for proprietary companies (and for debugging too) but for FOSS... it's another universe.

Well, I guess the possible reason is that U++ was developed exactly for development of proprietary software... It is tuned to that case.... This is no "ah, we have some spare time, what about to create another framework" kind of software. It was created to solve something...

OTOH, maybe you should rather think about U++ code in terms of scripting languages. Nobody expects .so for python module. And that interfacing of U++ to existing libraries is very simple and the U++ binaries tend to run faster than anything else is just a bonus

In fact, it is considered that we perhaps might create a "runtime environment" for U++, to make it behave even more like scripting language. It would just use GCC in the process

Mirek

Subject: Re: Does the provided upp.spec works for you and on which distro?
Posted by amrein on Tue, 02 Sep 2008 09:38:03 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Mon, 01 September 2008 15:00
And I don't understand the part about Gnome? Who wants to port Gnome? And they do have GtkMM, but with all the gobject crap that's in the background, I wouldn't call it thin.

What is the memory and disk footprint of 30 applications, from the windows manager to the control panel, from the text edit tool to the word processor, ... when each of them use the same code but don't share it with a dynamic library.

Gnome was an example.

Subject: Re: Does the provided upp.spec works for you and on which distro?

luzr wrote on Tue, 02 September 2008 10:06...
In fact, it is considered that we perhaps might create a "runtime environment" for U++, to make it behave even more like scripting language. It would just use GCC in the process

Mirek

Binding for Python, Perl, Rubby, Tcl, Gambas (http://gambas.sourceforge.net/), Java... Could be done in hours if we had a dynamic library.
It should be possible to create a new package like "libupp" statically linking with all upp, and the output should be a dynamic library libupp.so.0.8.1 instead of an application.

Resume

Since the beginning, I talked about website, doc, dynamic linking, version number, directories clean up, file names policy, class names policy, ... to answer a simple expectation (mine): "How can U++ have a wider audience and become mainstream in FOSS". What amaze me the most now: I'm trying to answer my question, but nobody, except me, see my solutions as interesting suggestions. I have no more idea to submit. The answer I got are completely justified in that context:

Website left menu: Generated from Topic++. Modifications should go there first.

Doc with title and plan: Get it and improve it, if the result is cool, we will use it.

Version number x.y.z (0.year-2000.svntag?): We prefer Year.Release. Use what you want if you work on a dynamic library.

U++ as a dynamic library too: Too much work. U++ and TheIDE API change too often. TheIDE use static linking.

Manage Topic++ doc as doxygen comments in the source: We prefer external Topic++ documentation.

Cpp an .h file names policy (all lower case for better portability): No need for policy, our programmers are used to current file names.

Class names without abbreviations (like Ctrl->Control): No thanks, this is what we are used to.

Directory clean up: Well perhaps TheIDE could be in its own Package/Assembly/Nest. Perhaps U++ directories structure are not clean enough for you but we are comfortable with it and we works on it each days, as many other U++ users

Something easier than "Assembly/Nest/Package": "Assembly/Nest/Unit"?
(...)

I have no more ideas. I won't be able to help as I thought I could. At least, I can help for the rpm build process.

Subject: Re: Does the provided upp.spec works for you and on which distro?
Posted by mirek on Tue, 02 Sep 2008 11:53:13 GMT
View Forum Message <> Reply to Message

amrein wrote on Tue, 02 September 2008 06:45luzr wrote on Tue, 02 September 2008 10:06...
In fact, it is considered that we perhaps might create a "runtime environment" for U++, to make it behave even more like scripting language. It would just use GCC in the process

Mirek

Binding for Python, Perl, Rubby, Tcl, Gambas (http://gambas.sourceforge.net/), Java... Could be done in hours if we had a dynamic library.
It should be possible to create a new package like "libupp" statically linking with all upp, and the output should be a dynamic library libupp.so.0.8.1 instead of an application.

Creating .so is the least problem.

The major problem is HOW MANY.

Surely, you can mix the whole uppsrc into single .so. But in that process, you will have a lot of external dependencies for everybody - and not everybody is using e.g. PGSQL.

BTW, thinking about other language bindings, if we forget for the moment a strong coupling of U++ with C++ (I am not sure how much will be left in other language), maybe the correct solution would be to create thin "fixed API" layer over U++.

Such API would change infrequently, exactly as "FOSS traditional versioning scheme" requires, and linked with UPP using it as "engine code". In that case, you would get exactly what you need.

Quote:
Since the beginning, I talked about website, doc, dynamic linking, version number, directories clean up, file names policy, class names policy, ...

Talk is cheap...

Quote:
What amaze me the most now: I'm trying to answer my question, but nobody, except me, see my solutions as interesting suggestions.

I definitely see the list as interesting, if nothing else, it helps me to see us from another

perspective. In fact, I think it has helped a lot.

But, as newcomer, you cannot expect everything you suggest to be accepted immediately and also if anything IS accepted (like BSD license), to happen in days.

Quote:
Website left menu: Generated from Topic++. Modifications should go there first.

Doc with title and plan: Get it and improve it, if the result is cool, we will use it.

Will you do it?

Frankly, I would like to have better website, but our resources are limited right now...

Quote:
I have no more ideas. I won't be able to help as I thought I could. At least, I can help for the rpm build process.

Thanks for that!

Mirek

Subject: Re: Does the provided upp.spec works for you and on which distro?
Posted by mirek on Tue, 02 Sep 2008 12:17:40 GMT
View Forum Message <> Reply to Message

Well, this was posted by mr_ped while I was splitting the thread, because it was not about "Releasing U++ rpms" anymore...

mr_ped wrote on Tue, 02 September 2008 08:07And you still didn't even hit the "contribute the code, get refused anyway" scenario.

Well, keep contributing with ideas, discussing them, and your target is to make Mirek believe those changes are good and important.

Now imagine what would happen if the gates would be totally open...

In fact, I have hard times to "fix" some old design mistakes made by Tom years ago...

Quote:
(Or "Uno" is your second change to change something in official U++, the third member of core U++ team does not show up at forums AFAIK)

The list is growing. Make James Thomas (mrjt) believe and you have a very strong argument

Mirek

Subject: Re: Does the provided upp.spec works for you and on which distro?
Posted by cbpporter on Tue, 02 Sep 2008 12:19:43 GMT
View Forum Message <> Reply to Message

Don't get thing wrong please. We both appreciate and need help in general, but a lot of you suggestions are related to parts which actually work. It is not as much resistance to change, as it is a lack of interest and man-power to improve something which works quite well rather than focus on the parts which don't work too well.

Regarding creation of .so: I believe we are open to the usage of .so. I don't know who really needs it, but it would definitely be great to have a dll version too. It would also make things easier for those who don't want to use TheIDE. If you're willing to do this, be my guest. I will do my best to help you, but I'm not interested in this on my own, because it would not benefit me: I distribute 2-3 binary systems: one is the installer, and the rest are the application executable. With only 2-3 exe's, code sharing between .so is not that great on memory footprint reduction, and I get the added disadvantage of slower function calls.

But in practice it is not that easy. First of all, we've got all the templates that can't be embedded in the .so. Then there's the versioning, API/ABI issue. We would need a rigorous system to track ABI

in const because it did not change the status of the object. We added const to be correct, but this definitely would have broken compatibility with binary .so because of name mangling.

So I think that dll's are a good idea, but only if you build a version specific to you application and ship it with your application. Getting a general .dll is a lot harder. If you are aware of a solution that would solve such issues, then please tell us.

Quote:Cpp an .h file names policy (all lower case for better portability): No need for policy, our programmers are used to current file names.
I'm sorry, but I consider lower case names as deprecated as I do non-case sensitive paths. Even under Windows I always use proper cased names for files, and also except to get such names. So "Ctrl.h" is not a caseless or lower case name which was written with a capital "C" because Windows is not case-sensitive, it is really called "Ctrl.h" and referring to it as "ctrl.h" is an error IMO. And since all Unixes are case sensitive and Windows is case neutral, I don't think we have any issues left with this scheme. Also, if we change to lower case on Unix, all programs will cease to compile until we manually modify all include clause.

The same applies to using more longer and intuitive names. How can we change the name of a file and except old code to compile. How can we change the name of a class and still except to link with the .so that we are going to produce?

Quote:Something easier than "Assembly/Nest/Package": "Assembly/Nest/Unit"?
Actually I don't care what they are named, but I kind of like their meaning and implication on code layout. A little clean up on interface level and I'm sure things would seem a little cleaner for new people. Also, "assembly" sounds familiar to .Net crowd, but has different semantics, so a change would not be a bad idea.

Doxygen: I'm all for reusing technology which works (i.e. doxygen) and wouldn't mind using it instead of Topic++. Topic may be more intimately tied to A++, but I don't think this mattes that much since docs are not autogenerated.

On the other hand, using doxygen would increase compilation times. I'll go crazy if I use another 0.5 second time per build. I would hate to have to abandon both the language and framework that I have grown to love and switch over to .NET because of trivial issues like compilation speed which should have been fixed decades ago.

And docs are coming along .

On the other hand, I don't see why we should argue about little issues like filenaming, when we need a better C++ parser for example. Code sequences that short out Assist++ are hard to circumvent and they spread like a virus.

Out of all these things, I think that .so should be a priority if you still need it. If we get that process working, adding support for TheIDE to build a dynamic lib and link your project to it would definitely be doable.

Subject: Re: Does the provided upp.spec works for you and on which distro?
Posted by mirek on Tue, 02 Sep 2008 12:22:08 GMT
View Forum Message <> Reply to Message

amrein wrote on Tue, 02 September 2008 05:38cbpporter wrote on Mon, 01 September 2008 15:00
And I don't understand the part about Gnome? Who wants to port Gnome? And they do have GtkMM, but with all the gobject crap that's in the background, I wouldn't call it thin.

What is the memory and disk footprint of 30 applications, from the windows manager to the control panel, from the text edit tool to the word processor, ... when each of them use the same code but don't share it with a dynamic library.

Gnome was an example.

Well

a) we are not creating X11 desktop (yet?

b) you can very easily workaround the problem by providing a single application that does it all (and I bet such complete desktop application would be about 1/10 of those 30 Gnome apps combined size).

c) you can, RIGHT NOW, compile U++ to .so modules that would be shared. But this has package = .so equivalency, not really practical for "classic use".

Mirek

Subject: Re: Does the provided upp.spec works for you and on which distro?
Posted by mirek on Tue, 02 Sep 2008 12:29:08 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Tue, 02 September 2008 08:19
Doxygen: I'm all for reusing technology which works (i.e. doxygen) and wouldn't mind using it instead of Topic++. Topic may be more intimately tied to A++, but I don't think this mattes that much since docs are not autogenerated.

And docs are coming along .

Do not forget one small practical issue. With doxygen style docs, people need write access to trunk code to edit documentation.

Meanwhile, separate .tpp directories do not present any danger to the code. Broken docs is something fixed easily. But I would not want to have deployed broken app because somebody accidentally damaged the code when working on docs.

Means, we can allow much more people to edit docs with T++...

That said, I had and have really hard time deciding this (T++ vs doxygen). This is not a new issue, I was at it since the very beginning.

I see the pros/cons:

Doxygen pros:
- known standard tool that works
- not much work required (except moving doc back from T++)
- very thight integration with code (because it is in the code)

cons:
- needs write access to the code to document it
- hard to put images into docs
- harder to put "explanation passages" (because the document structure is fixed)

T++ pros:
- separate docs with separate write rights
- comfortable wordprocessor with spelling checker

- no problem screenshotting widgets and putting images
- same environment to create code reference docs ("src") and explanation/tutorial docs ("srcdoc")
- possible to add implementation articles

cons:
- "invented here"
- a lot of work programming it (but, we need some of it anyway)

(I might have forgoten something)

Mirek

Mirek

Subject: Re: Does the provided upp.spec works for you and on which distro?
Posted by captainc on Tue, 02 Sep 2008 14:22:43 GMT
View Forum Message <> Reply to Message

I don't think we need full doxygen support, but I think it would be beneficial to have the ability to automatically extract the doxygen-style documentation out of the code to import it into T++. It could extract the documentation along with the function references as T++ does now.
Also, this may entice developers to provide some documentation in code. I don't think we want ALL the documentation in the source files, they will get too big and difficult to read. Code should for the most part be self-documenting, but certain things may need to be explained to avoid confusion when reading the code. Some in-code documentation is very helpful, too much documentation is not.

Subject: Re: U++ talk
Posted by amrein on Tue, 02 Sep 2008 18:47:45 GMT
View Forum Message <> Reply to Message

As long as upp version is not higher than 0.99.999..., the API can broke. That doesn't matter much. A present It could be 0.8.1. No problem with anyone compiling with 2008.1.
2009.0 then? Anyone using static linking will certainly need to rebuild their application too. In FOSS, when a lib is updated to a new release, all distro rebuild their packages if they need it. U++ don't need to care about API stability. Really. There are a lot of libraries modifying their API and some of them broke it for each release. If U++ API is broken from version X to version Y, than it's mean that new U++ version is better. Other FOSS software will just have to follow.

The only problem could be if the U++ team wanted to backport fix to the 2008.1 branch for example. Most distro do it themselft and submit patches (or you can get them from their source packages). They also see by themselves from svn what is a fix and what will break the official

release. They don't get all fix because a perfect product (for them) are not perfect to sell (no need to update to next release).

Quote:I'm sorry, but I consider lower case names as deprecated as I do non-case sensitive paths. Even under Windows I always use proper cased names for files, and also except to get such names. So "Ctrl.h" is not a caseless or lower case name which was written with a capital "C" because Windows is not case-sensitive, it is really called "Ctrl.h" and referring to it as "ctrl.h" is an error IMO. And since all Unixes are case sensitive and Windows is case neutral, I don't think we have any issues left with this scheme. Also, if we change to lower case on Unix, all programs will cease to compile until we manually modify all include clause.

Yes. "#include" clean-up + a strict policy for all new .cpp and .h names.
You can find this policy in Qt, Epoc, Symbian, Gtk, wxWindows, WinCE... Most cross platform libraries use it.
You are used to U++ source code. You don't see it as I do.

Quote:The same applies to using more longer and intuitive names. How can we change the name of a file and except old code to compile. How can we change the name of a class and still except to link with the .so that we are going to produce?

A source clean-up + a strict lower-case name policy for all new .cpp and .h doesn't mean rename internal class. No need to change their names.
The exceptions: class using abbreviations shouldn't use abbreviations. No need to get used to those abbreviations and source code is easier to understand.

Quote:Doxygen: I'm all for reusing technology which works (i.e. doxygen) and wouldn't mind using it instead of Topic++. Topic may be more intimately tied to A++, but I don't think this mattes that much since docs are not autogenerated.

On the other hand, using doxygen would increase compilation times. I'll go crazy if I use another 0.5 second time per build. I would hate to have to abandon both the language and framework that I have grown to love and switch over to .NET because of trivial issues like compilation speed which should have been fixed decades ago.

Doxygen comment are just C++ comment. The C++ parser is optimised to skip them.
Things like dependencies, methods, attributes, class hierarchy, call graph are computed by doxygen.

Note: When you recompile, you don't recompile U++ (BLITZ...). Just your own application. We could say just your modified files.

How to include an image in the final doc: html tag in the doxygen source comment.

How to include a link to another doc or source code(tutorial...): html tag in the doxygen source comment.

How to include a link to load and see the original source code: one option to activate in the

doxygen configuration file.

Is it possible to create an external documentation referring to the html output. Something like we already have on ultimatepp.org? Yes.

Do you know a powerful tools that will extract, insert and manage automatically source code comments for doxygen? No

How can people still add new documentation without touching the source code? Well. You will hate me. An applications like linguist (from Qt) use one tool output, lupdate tool (from Qt too), to extract all translation from .cpp and output a xml file. You can then do the translation job on this file with linguist. It's like TheIDE translation tool but with 2 stand alone applications. They don't reinsert the translation, they load it at run time to save memory (only one language for each xml file because with big application you can have several Mo in one file), but, in a documentation tool like Topic++, this metadata could be validated and reinserted by the documentation project manager.

Then, why still use doxygen: Because doxygen can do a lot of things that Topic++ can't right now. Because if Topic++ become an universal doxygen source code documentation tool it will have a wider audience and will also promote TheIDE and U++.

I include a simple doxygen file to test. Download it in ~/upp, cd into ~/upp then type "doxygen". Doc will be generated in ~/upp/documentation. I used doxywizard (from doxygen package) to create it.
I didn't add "search support" because I guess all people here don't use apache+php on their Pc.

File Attachments

1) Doxyfile, downloaded 328 times

Subject: Re: U++ talk
Posted by mirek on Tue, 02 Sep 2008 19:23:08 GMT
View Forum Message <> Reply to Message

Quote:
A source clean-up + a strict lower-case name policy for all new .cpp and .h doesn't mean rename internal class. No need to change their names.

Well, but you would have to fix all #includes...

Quote:
The exceptions: class using abbreviations shouldn't use abbreviations. No need to get used to those abbreviations and source code is easier to understand.

Not ever. Very long name can make it less understandable.

"Ctrl" is appropriate name for frequently used class.

Also, sometimes there simply is not a proper word to name a thing. That is where those pseudowords are perhaps better than wrong names. At least, so I believe.

Quote:
How to include an image in the final doc: html tag in the doxygen source comment.

Sure, but that is as tedious as it can get....

Mirek

Subject: Re: U++ talk
Posted by mirek on Tue, 02 Sep 2008 19:24:12 GMT
View Forum Message <> Reply to Message

P.S.: Good thing about this Doxygen is that you will likely make me implement those planned T++ improvements sooner

---

Subject: Re: U++ talk
Posted by amrein on Tue, 02 Sep 2008 20:20:59 GMT
View Forum Message <> Reply to Message

luzr wrote on Tue, 02 September 2008 21:23Quote:
A source clean-up + a strict lower-case name policy for all new .cpp and .h doesn't mean rename internal class. No need to change their names.

Well, but you would have to fix all #includes...

I can make a bash script if you want. It's not hard to do.

Quote:Quote:The exceptions: class using abbreviations shouldn't use abbreviations. No need to get used to those abbreviations and source code is easier to understand.

Not ever. Very long name can make it less understandable.

"Ctrl" is appropriate name for frequently used class.

Also, sometimes there simply is not a proper word to name a thing. That is where those

pseudowords are perhaps better than wrong names. At least, so I believe.

Sometimes yes.

Control . You save 3 letters.

FileSel, DocEdit::Fmt, PalCtrl...

With TheIDE with Control+","... do we really need to save those few keystrokes?

Quote:Quote:
How to include an image in the final doc: html tag in the doxygen source comment.

Sure, but that is as tedious as it can get....

Not with Topic++ managing Doxygen doc. Do you see it?

Subject: Re: U++ talk
Posted by cbpporter on Tue, 02 Sep 2008 20:30:02 GMT
View Forum Message <> Reply to Message

Quote:Doxygen comment are just C++ comment. The C++ parser is optimised to skip them.
I've seen that optimization in practice... (sarcasm)

Quote:How can people still add new documentation without touching the source code? Well. You will hate me. An applications like linguist (from Qt) use one tool output, lupdate tool (from Qt too), to extract all translation from .cpp and output a xml file. You can then do the translation job on this file with linguist. It's like TheIDE translation tool but with 2 stand alone applications. They don't reinsert the translation, they load it at run time to save memory (only one language for each xml file because with big application you can have several Mo in one file), but, in a documentation tool like Topic++, this metadata could be validated and reinserted by the documentation project manager.

It's not about creating translation (current system is working great) as contributing documentation. Someone has to write the documentation directly in the source code. That means write access is needed, plus someone has to test that it still compiles.
If user has doesn't have write access, somebody needs to merge the changes for him. Also, after every doc commit, people will have to recompile their sources and run doxygen on them. And if you distribute prebuilt documentation, you basically are back to square one.

But I'm also unhappy with Topic++. I would like to have an autogenerated DB style documentation, with doxygen style tags but only for some logical ordering, like saying that function Split should appear on the documentation page "String manipulation function" and on "Strings".

Running the topic++ toll would generate all pages in qtf, but would not let you edit only specific parts of the file, like implementation notes and parameter descriptions. Also, if a function is removed, it's attached documentation would be hidden, but the information not lost. You could always pull up a screen with dependency graphs.

But Mirek did not like the idea, so now I'm waiting for his idea and when it is ready I'll write some doc pages and hope that other people in the community will have the patience to manually format a consistent looking help content.

Subject: Re: U++ talk
Posted by mirek on Tue, 02 Sep 2008 21:20:24 GMT
View Forum Message <> Reply to Message

Quote:
With TheIDE with Control+","... do we really need to save those few keystrokes?

That is not the issue. Problem is when long names makes your method declaration span over several lines (or similar situations).

Quote:
Not with Topic++ managing Doxygen doc. Do you see it?

Somehow, I do not see what you suggest...

Mirek

Subject: Re: U++ talk
Posted by mirek on Tue, 02 Sep 2008 21:22:29 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Tue, 02 September 2008 16:30
But Mirek did not like the idea, so now I'm waiting for his idea

Only because I have tried it before...

Mirek

Subject: Re: U++ talk
Posted by amrein on Tue, 02 Sep 2008 21:37:52 GMT
View Forum Message <> Reply to Message

Quote:
It's not about creating translation (current system is working great) as contributing documentation. Someone has to write the documentation directly in the source code. That means write access is needed, plus someone has to test that it still compiles.

At present, people use topic++ right? They save the doc in external files. What I said: do exactly the same process. The document manager (not his subordinates) can have a look into the topic++ work (what his subordinate, those without write access to the code, have done) than, if he think that all is ok, he presses a topic++ button and all validated modifications are synchronized back into the source file, ready for doxygen and U++ final release.

Quote:
If user doesn't have write access, somebody needs to merge the changes for him.

Yes, modifications has to be validated. U++ API can break at any time before final release. Two eyes are better than one. You keep total control of the final documentation submitted but your contributors have full access to the topic++ generated documentation.

Quote:
Also, after every doc commit, people will have to recompile their sources and run doxygen on them. And if you distribute prebuilt documentation, you basically are back to square one.

If they don't want to edit the topic++ output, and want the last doxygen output then yes. What prevent them from using topic++ to see the unstable doc? Nothing. How could the topic++ documentation break the source if the integration process is automated by topic++?

Subject: Re: U++ talk
Posted by captainc on Wed, 03 Sep 2008 01:44:54 GMT
View Forum Message <> Reply to Message

Quote:They save the doc in external files. What I said: do exactly the same process. The document manager (not his subordinates) can have a look into the topic++ work (what his subordinate, those without write access to the code, have done) than, if he think that all is ok, he presses a topic++ button and all validated modifications are synchronized back into the source file, ready for doxygen and U++ final release.

This would impose a set structure on the topic++ documentation; it would limit the customization of the documentation by the authors. Also, Doxygen/Javadoc is picky. I don't see how reversing Topic++ documentation to Doxygen would be effective. For example, why would you put a tutorial in code documentation? On the other hand, having Doxygen documentation show in Topic++

would be quite helpful. The entire src documentation section could be automatically generated.

Subject: Re: Does the provided upp.spec works for you and on which distro?
Posted by mirek on Wed, 03 Sep 2008 07:49:42 GMT
View Forum Message <> Reply to Message

amrein wrote on Tue, 02 September 2008 06:45
Cpp an .h file names policy (all lower case for better portability): No need for policy, our programmers are used to current file names.

BTW, out of curiosity, I have decided to check your claim about every FOSS project following lowercase policy to avoid cross-platform problems.

So, here is the list of those that do not (you can probably find more):

- Mozilla Firefox
- fltk
- Fox toolkit

I guess at least Firefox makes your point completely moot

Mirek

Subject: Re: U++ talk
Posted by mirek on Wed, 03 Sep 2008 07:58:27 GMT
View Forum Message <> Reply to Message

amrein wrote on Tue, 02 September 2008 17:37Quote:
It's not about creating translation (current system is working great) as contributing documentation. Someone has to write the documentation directly in the source code. That means write access is needed, plus someone has to test that it still compiles.

At present, people use topic++ right? They save the doc in external files. What I said: do exactly the same process. The document manager (not his subordinates) can have a look into the topic++ work (what his subordinate, those without write access to the code, have done) than, if he think that all is ok, he presses a topic++ button and all validated modifications are synchronized back into the source file, ready for doxygen and U++ final release.

Quote:
If user doesn't have write access, somebody needs to merge the changes for him.

Yes, modifications has to be validated. U++ API can break at any time before final release. Two eyes are better than one. You keep total control of the final documentation submitted but your contributors have full access to the topic++ generated documentation.

OK, now I see where are you heading. Anyway, this is even more complex that what we wanted to do. And, let me say, perhaps a little bit more risky too.

OTOH, it appears that you would like to do some work on .so etc.

Maybe, as part of this process, you can do exactly this.

I mean, it appears that ".so" work will have to somehow take existing uppsrc and via some nice tool transform it to .so (or, rather, something with makefile that produces .so).

The T++ contains enough information (ok, barely enough, but it is manageable) to put the comments back to the code even now. So do it!

Or, maybe, you can export doxygen to separate files, that is even easier.

BTW, to put things into perspective and understand each other, if you are speaking about Doxygen, I was thinking about putting docs into sources, that is why confusion. Exporting Doxygen formatted docs from T++ is IMO completely unrelated topic and perhaps a good idea, if it can be done with relatively low costs.

Mirek

Subject: Re: U++ talk
Posted by cbpporter on Wed, 03 Sep 2008 08:23:00 GMT
View Forum Message <> Reply to Message

Yes, we should go forward and get started on .so. The question is how many .sos should we create.

On possible layout would be: one for Core and other non GUI stuff, one For CtrLib and all control and draw, and one for each SQL dialect. There is no way I as a normal user am installing MySQL, Sqlite3 and the other client so that I can install a .so for U++.
Also, I don't know what to do with NOGTK versions of GUI packages. Linking a NOGTK app with a GTK .so (if possible) would give strange results.

Also, with .so distributions comes the problem of devel packages. Traditionally installed .h files would not work well with out packaging scheme so I think we should skip this one and use the RPM/DEB for full U++ packages to get these files.

Edit: also for MT version of lib.

Subject: Re: U++ talk
Posted by mirek on Wed, 03 Sep 2008 09:10:17 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Wed, 03 September 2008 04:23Yes, we should go forward and get started on .so. The question is how many .sos should we create.

On possible layout would be: one for Core and other non GUI stuff, one For CtrLib and all control and draw

Well, it might not be actual now, but in future, I would like to have "pure" Draw that is not related to X11, to support web applications (which want to draw images, but do not want to have X11 dependency). But that can wait.

Related problem - Image formats. Should it contain all of them? Maybe yes...

Quote:
, and one for each SQL dialect.

And one for SQL as whole and another for SQL GUI....

Or, maybe, generic Sql can be part of 'core.so' (there are no dependencies) and SQL GUI can be part of 'ctrllib.so' ?

Quote:
Also, I don't know what to do with NOGTK versions of GUI packages. Linking a NOGTK app with a GTK .so (if possible) would give strange results.

IMO, that is not a real problem, these are just two libraries.

Quote:
Also, with .so distributions comes the problem of devel packages. Traditionally installed .h files would not work well with out packaging scheme so I think we should skip this one and use the RPM/DEB for full U++ packages to get these files.

Maybe just put 'upp' folder to /usr/include and put all headers there in normal way - I mean, basically, just delete all .cpp files from uppsrc packages, put result into /usr/include/upp and add /usr/include/upp to INCLUDE path.

Quote:
Edit: also for MT version of lib.

Well, I think, MT should be just activated for .so.

Mirek

Subject: Re: U++ talk
Posted by amrein on Wed, 03 Sep 2008 16:52:47 GMT
View Forum Message <> Reply to Message

Quote:
OK, now I see where are you heading. Anyway, this is even more complex that what we wanted to do. And, let me say, perhaps a little bit more risky too.

Yes.
I don't know how to generate complex doxygen doc without including the documentation back into the code.

Quote:
OTOH, it appears that you would like to do some work on .so etc.

Maybe, as part of this process, you can do exactly this.

I mean, it appears that ".so" work will have to somehow take existing uppsrc and via some nice tool transform it to .so (or, rather, something with makefile that produces .so).

The T++ contains enough information (ok, barely enough, but it is manageable) to put the comments back to the code even now. So do it!

Makefile already done, just for testing. I modified the upp-2008.1 Makefile to see how much disk space a lib.so could take.

At present, I'm trying to learn U++. Doc is not enough to be a good U++ programmer. I really need to build a few small applications (statically linked). I'm still a real noob in U++.

When I will feel ready, I will work on TheIDE source code to generate Makefile able to produce dynamic libraries from uppsrc packages and certainly also on U++ missing documentation.

Quote:
Or, maybe, you can export doxygen to separate files, that is even easier.

BTW, to put things into perspective and understand each other, if you are speaking about Doxygen, I was thinking about putting docs into sources, that is why confusion. Exporting Doxygen formatted docs from T++ is IMO completely unrelated topic and perhaps a good idea, if it can be done with relatively low costs.

Mirek

I think you mean Topic++ will still use external files but TheIDE will be able to show them directly side by side in the code editor, right?

Quote:
There is no way I as a normal user am installing MySQL, Sqlite3 and the other client so that I can install a .so for U++.

No need to. Same goes for Qt, wxWindows, Gnomedb, Kde, Gambas, ...

Quote:
On possible layout would be: one for Core and other non GUI stuff, one For CtrLib and all control and draw, and one for each SQL dialect.

The idea in the sent list (other thread) was to have several libs. Each directory in level 3 could be a library:

./source/upp/core
./source/upp/gui/core/x11
./source/upp/gui/core/win
./source/upp/gui/core/osx
./source/upp/gui/core/directfb
./source/upp/gui/library
./source/upp/gui/dialogs/x11
./source/upp/gui/dialogs/win
./source/upp/gui/dialogs/osx
./source/upp/network
./source/upp/database
./source/upp/database/oracle
./source/upp/database/odbc
./source/upp/database/sqlite
./source/upp/database/mysql
./source/upp/database/postgresql
./source/upp/dbus
./source/upp/opengl
./source/upp/xml
./source/upp/sound
./source/upp/video
./source/upp/html
./source/upp/script
./source/upp/richtext
./source/theide/...
./source/unstable/my_unstable_ide/
(...)

Core, Gui, Network, Database, Opengl, XML could be dynamic libraries. Upp could be an all in

one library too.

How to have X11 or directFb back end is a good question. Build several libraries like wxWindows: gui-x11, gui-directfb, gui-gtk? Other solution: gui-core load its screen plugin (x11, directfb, linuxfb, javascript, ...) at runtime.

QtEmbedded does this for Qtopia: if you don't provide a special switch the lib try to connect to the Linux framebuffer, if not it connect to the framebuffer simulator. At compile time, you can remove the ability to connect to different backend and select the main one.

Subject: Re: U++ talk
Posted by mirek on Wed, 03 Sep 2008 20:55:51 GMT
View Forum Message <> Reply to Message

amrein wrote on Wed, 03 September 2008 12:52
The idea in the sent list (other thread) was to have several libs. Each directory in level 3 could be a library:

./source/upp/core
./source/upp/gui/core/x11
./source/upp/gui/core/win
./source/upp/gui/core/osx
./source/upp/gui/core/directfb
./source/upp/gui/library
./source/upp/gui/dialogs/x11
./source/upp/gui/dialogs/win
./source/upp/gui/dialogs/osx
./source/upp/network
./source/upp/database
./source/upp/database/oracle
./source/upp/database/odbc
./source/upp/database/sqlite
./source/upp/database/mysql
./source/upp/database/postgresql
./source/upp/dbus
./source/upp/opengl
./source/upp/xml
./source/upp/sound
./source/upp/video
./source/upp/html
./source/upp/script
./source/upp/richtext
./source/theide/...
./source/unstable/my_unstable_ide/
(...)

Core, Gui, Network, Database, Opengl, XML could be dynamic libraries. Upp could be an all in

one library too.

I suspect this is again another topic. We are not interested in restructuring U++ sources (except separating theide - perhaps).

What we need to decide is what packages should be combined into single .so. IMO, this is mainly decided by dependencies.

Mirek

Subject: Re: U++ talk
Posted by mirek on Wed, 03 Sep 2008 21:00:13 GMT
View Forum Message <> Reply to Message

Quote:
I think you mean Topic++ will still use external files but TheIDE will be able to show them directly side by side in the code editor, right?

Of course, but not only. Maybe, as complete unrelated issue, we can provide doxygen export.

AFAIK, doxygen allows separated documents too. So we can export such files.

Quote:
Quote:
There is no way I as a normal user am installing MySQL, Sqlite3 and the other client so that I can install a .so for U++.

No need to. Same goes for Qt, wxWindows, Gnomedb, Kde, Gambas, ...

What goes for them? I am lost

Mirek

Subject: Re: U++ talk
Posted by cbpporter on Thu, 04 Sep 2008 02:52:55 GMT
View Forum Message <> Reply to Message

This could be a possible list of packages to go into the first .so and also some thoughts regarding them:

- Core
- Crypto (also, add description to Crypt and maybe AES - I still have a textbook AES somewhere if you're interested)
- Esc (Esc is a pretty nice minimal scripting language, perfect to embed in your code, so we could include it. OTOH, a plan for the future: make Esc a strict subset of and existing and established scripting language, so that Esc works in that language, but not the other way around necessarily. There is a myriad of scripting languages for any given purpose, and I don't think we or the world needs a new one. Or maybe replace Esc with something small like Lua, but Lua is not my first choice for what Esc should accomplish)
- Geom (also maybe clean-up and decide future of it)
- Geom\Coords (this package depends on TCore package, which is empty; also same clean-up as with Geom)
- plugin/astyle - maybe (add description)
- plugin/bz2
- plugin/dbf - maybe, I don't know exactly what it is and if it is related to the old database format, in which case we may decide move it to DB .so (also add description)
- plugin/ftp
- plugin/pcre - I didn't even know we had regexp, this should be advertised more (add description)
- plugin/z (also, I think it shouldn't be a requirement for basic Core console app)
- Web and friends (add description, and some parts are in need of some extensions)

This would basically be it. I hope I didn't miss any packages.

The question is what to do about image format packages. Some can be included without Draw, while others can't. The best solution would be to make these plugins work with in-memory images and streams, and registering the encoder/decoder, and Draw stuff being able to work with those images and display them as soon as the required plugin is included. This way it would also work for servers (like the way gdlib is used for this purpose).

We could still squeeze in PNG and maybe some other formats which work without GUI packages right now.

So there would be a first .so with maybe some graphical formats. I see 3 more .sos at least: one for GTK GUI, one for NOGTK GUI (but we do need a new default skin for that, because that look is worst than Gtk 1, which was ugly as hell, and doesn't really fit in with any modern look) and at least a SQL .so.

Also, either build packages for MT, or double the number of .sos.

This is just a proposal though. The important thing is getting them out one by one, and not focus on getting all the .so structures at once. We need some time and get it right, because once you settle on something, it is going to be harder to change it.

Also, some other crazy ideas: make U++ able to run and draw it's own windows when in a framebuffer like environment. This way it could run on the Linux framebuffer, or in a normal WinAPI window, but in DirectX buffers for games. I did some hobby gamedev in the past and there was always the problem of GUI toolkit able to render itself in the DirectX framebuffer. And if this works, why not make it work in a console, for text mode GUI's. We could call it

Subject: Re: U++ talk
Posted by mr_ped on Thu, 04 Sep 2008 07:30:17 GMT
View Forum Message <> Reply to Message

Quote:- plugin/pcre - I didn't even know we had regexp, this should be advertised more (add description)

WOW, I did wish for this one, and I missed it completely when I took a short look at packages.
It should be listed somewhere in features.
Now I see there is reference/RegExp, I can't understand how I missed it. Must have been that "I'm blind" moment.

Quote:plugin/z
I may be wrong, but doesn't the Core depend on the compression a bit with some files? For example .iml data, aren't they compressed?
(I'm asking, I have no idea what's the true)
I know you don't need .iml files for console application, so if your first ".so" is supposed to be like minimal bare Core library with minimal dependencies, and adding "z" later trough extending .so is of no problem, then your point is still valid.

I wonder also how we will combine those .so in final application? How should it work, like having core.so + guigtk.so + sql.so + application, with each ".so" having different functions and working together? I think there may be some funny stuff about making the memory allocator to work together across all the .dll files, but my knowledge is next to none, so if somebody has serious experience/knowledge, I would welcome some explanation how such application should work (the bigger picture).

Subject: Re: U++ talk
Posted by mirek on Thu, 04 Sep 2008 10:45:00 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Thu, 04 September 2008 03:30
Quote:plugin/z
I may be wrong, but doesn't the Core depend on the compression a bit with some files? For example .iml data, aren't they compressed?
(I'm asking, I have no idea what's the true)

It does. zlib has to be mandatory.

Quote:
I wonder also how we will combine those .so in final application? How should it work, like having core.so + guigtk.so + sql.so + application, with each ".so" having different functions and working

together?

Yes. In fact, this is not uncommon - GTK = gobject + gdk + gtk.... (roughly equals to Core, Draw and CtrlLib+CtrlCore).

Sure, for those of us that understand and use U++ way of things, this sounds just like stone-age solution. But .so are for those that do not, I guess

Quote:
I think there may be some funny stuff about making the memory allocator to work together across all the .dll files, but my knowledge is next to none, so if somebody has serious experience/knowledge, I would welcome some explanation how such application should work (the bigger picture).

Well, in the very worst case, we can just compile .so with NOMALLOC.

Anyway, I believe that if core.so is first, it will override new/delete for the rest.

AFAIK, the same applies in Win32. The trick (AFAIK!) is that linker searches .lib files for symbols in the order they are specified on commandline. new/delete are defined in standard c++ library, but that, as default, gets searched last.

Then, when linking, references are directed to the .so/.dll that contains the symbol...

Mirek

Subject: Re: U++ talk
Posted by amrein on Thu, 04 Sep 2008 11:38:25 GMT
View Forum Message <> Reply to Message

Quote:Quote:Quote:
There is no way I as a normal user am installing MySQL, Sqlite3 and the other client so that I can install a .so for U++.

No need to. Same goes for Qt, wxWindows, Gnomedb, Kde, Gambas, ...

What goes for them? I am lost

He said: if I install U++ dynamic libraries, I don't want all database to be installed. I reply "No need to". Other libraries have solutions to solve this.

When you install those dynamic libraries, there is no dependencies with the database. You don't need to install any of them. The database plugins (with database dependencies) are separated

from the main package:

urpmq --list | grep qt3 | sort

libqt3
libqt3-devel
libqt3-mysql
libqt3-odbc
libqt3-psql
libqt3-sqlite
libqt3-static-devel
libqt3support4
qt3-assistant
qt3-common
qt3-doc
qt3-example
qt3-linguist
qt3-tutorial

I mean, there is no rpm/deb dependencies between your application and the database binary as long as you don't link against one of those plugins. Database connections are done at runtime with the plugins and only if those database are available. Your application can list dynamically all available database plugins. You only need headers and libs from MySQL, Sqlite, ... when you build all packages from U++.

Examples:

In Qt3, If your application need ODBC only, you need to install libqt3-odbc and its dependencies (libqt3-odbc needs libqt3 and libunixODBC1).

Gambas goes farer: its plugins doesn't install database dependencies. They try to load those dependencies at runtime and if your program ask for MySQL, the plugin can answer "MySQL lib not available" or "Good, it's there, I load it".

Subject: Re: U++ talk
Posted by amrein on Thu, 04 Sep 2008 11:41:23 GMT
View Forum Message <> Reply to Message

Quote:
Also, some other crazy ideas: make U++ able to run and draw it's own windows when in a framebuffer like environment. This way it could run on the Linux framebuffer, or in a normal WinAPI window, but in DirectX buffers for games. I did some hobby gamedev in the past and there was always the problem of GUI toolkit able to render itself in the DirectX framebuffer. And if this works, why not make it work in a console, for text mode GUI's. We could call it UltimateVision++ .

Or UltimateTUI (text user interface)
If draw use plugins for backend, you can think about plugins like: X11, text/console, pdf, DirectX, SDL, xml ...

Subject: Re: U++ talk
Posted by mirek on Thu, 04 Sep 2008 19:54:52 GMT
View Forum Message <> Reply to Message

amrein wrote on Thu, 04 September 2008 07:38Quote:Quote:Quote:
There is no way I as a normal user am installing MySQL, Sqlite3 and the other client so that I can install a .so for U++.

No need to. Same goes for Qt, wxWindows, Gnomedb, Kde, Gambas, ...

What goes for them? I am lost

He said: if I install U++ dynamic libraries, I don't want all database to be installed. I reply "No need to". Other libraries have solutions to solve this.

When you install those dynamic libraries, there is no dependencies with the database. You don't need to install any of them.

Well, this is consistent with what I think we need....

Mirek

Subject: Re: U++ talk
Posted by mirek on Thu, 04 Sep 2008 19:55:53 GMT
View Forum Message <> Reply to Message

amrein wrote on Thu, 04 September 2008 07:41Quote:
Also, some other crazy ideas: make U++ able to run and draw it's own windows when in a framebuffer like environment. This way it could run on the Linux framebuffer, or in a normal WinAPI window, but in DirectX buffers for games. I did some hobby gamedev in the past and there was always the problem of GUI toolkit able to render itself in the DirectX framebuffer. And if this works, why not make it work in a console, for text mode GUI's. We could call it UltimateVision++ .

Or UltimateTUI (text user interface)
If draw use plugins for backend, you can think about plugins like: X11, text/console, pdf, DirectX, SDL, xml ...

I was thinking about it too, but I am afraid it perhaps is not really possible to make CtrlLib run in textmode. But I might be wrong...

Mirek

Subject: Re: U++ talk
Posted by amrein on Thu, 04 Sep 2008 21:06:48 GMT
View Forum Message <> Reply to Message

Turbo Vision from turbo pascal (or perhaps turbo C++) was doing text widget + events + mouse (msdos). There is a port for Linux: http://tvision.sourceforge.net/.

I also know about ncurses but I don't remember the name of another library with widget that use it. I don't find its name. It has a X11 port too (API compatibility).

Subject: Re: U++ talk
Posted by amrein on Thu, 04 Sep 2008 21:28:17 GMT
View Forum Message <> Reply to Message

Ok. Found. NewT and gNewT: http://www.whoow.org/people/Jean-Marc.Lienher/gnewt/index_en.html

---

Subject: Re: U++ talk
Posted by Mindtraveller on Sun, 07 Sep 2008 21:34:40 GMT
View Forum Message <> Reply to Message

I wonder if it really makes sense - making text version of such a complex library. For me it is hard to imagine typical scenario where text support should be necessary.
99% of modern PC handle graphics - they don`t really need text mode support. Those who don`t (some kind of controllers or may be very special purpose computers) usually don`t have wide support for advanced character work (such as on-screen character attributes support) - so text version of library should be mostly unsupported.
Speaking about old-linux (or BSD) text-style UI... I do really believe that well designed graphics mode file manager will be much easier than Midnight Commander with it`s text mode UI.

The only disadvantage I see is that U++ based apps with static linked library are rather heavy. Much heavier than they would have been with text-only support. But this problem, I beieve, should be solved with dynamic linking and having U++ dynamic libs shipped with *nix distros.

Subject: Re: U++ talk
Posted by Novo on Mon, 08 Sep 2008 03:01:01 GMT

Mindtraveller wrote on Sun, 07 September 2008 17:34I wonder if it really makes sense - making text version of such a complex library. For me it is hard to imagine typical scenario where text support should be necessary.
99% of modern PC handle graphics - they don`t really need text mode support. Those who don`t (some kind of controllers or may be very special purpose computers) usually don`t have wide support for advanced character work (such as on-screen character attributes support) - so text version of library should be mostly unsupported.

Let me not agree with you. If you are working with Linux all the time, you just cannot do anything without text mode. X-Window + SSH are working at acceptable speed only if you have gigabyte network. Even in case of such combination TheIDE is not very fast, I would say it is somewhat slow. It is just not possible to use TheIDE over cable modem. Of curse, there is no such problem if you have a Linux desktop. No network. No SSH.

The only comfortable way for me to work in Linux is to use *vi*. I would use TheIDE just as a text editor, if I could adapt it to understand Makefiles. The build system of TheIDE, which I would like to use at home (in case if I can adapt it to my previous build system), is not that important at work, where I can use distcc against 6 servers with 8 CPU-cores each.

Subject: Re: U++ talk
Posted by mirek on Mon, 08 Sep 2008 06:23:19 GMT

IMO the actual problem of "pixel" gui in text mode is simple - how to represent widgets and dialogs, originally designed for "real" pixels, in environment where 1 pixel == 1 character.

E.g. think about all those Frames in EditFields.

Therefore I am afraid that text mode GUI would require heavy adjustment to everything -> it is perhaps easier to design it from scratch (not to use CtrlLib). OTOH, Ctrl itself has perhaps suitable interface....

Mirek

Subject: Re: U++ talk
Posted by Mindtraveller on Mon, 08 Sep 2008 07:22:00 GMT

Novo, so the only working scenario is a remote development with X+SSH on *nix systems connected via modem? I`d say it is rather unfrequent scenario. Should we really consider such a

problem instead of i.e. developers using better connections? This would really lead to global rewrite of GUI libs.
Besides, I do develop for platforms with neither graphics nor attribute support. Just (ANSI) terminal with minimum available RAM (< 1 Mb). Should I request U++ support for such minimalistic platforms too?

Subject: Re: U++ talk
Posted by mr_ped on Mon, 08 Sep 2008 09:16:52 GMT
View Forum Message <> Reply to Message

BTW, speaking about ".so" version of U++, you may have find these old threads informational:
http://www.ultimatepp.org/forum/index.php?t=msg&th=2700
http://www.ultimatepp.org/forum/index.php?t=msg&th=2735

Subject: Re: Does the provided upp.spec works for you and on which distro?
Posted by mirek on Wed, 10 Sep 2008 17:26:15 GMT
View Forum Message <> Reply to Message

luzr wrote on Tue, 02 September 2008 08:29
Doxygen pros:
- known standard tool that works
- not much work required (except moving doc back from T++)
- very thight integration with code (because it is in the code)

cons:
- needs write access to the code to document it
- hard to put images into docs
- harder to put "explanation passages" (because the document structure is fixed)

T++ pros:
- separate docs with separate write rights
- comfortable wordprocessor with spelling checker
- no problem screenshotting widgets and putting images
- same environment to create code reference docs ("src") and explanation/tutorial docs ("srcdoc")
- possible to add implementation articles

cons:
- "invented here"
- a lot of work programming it (but, we need some of it anyway)

(I might have forgoten something)

Just for the record so the next time I do not have to reinvent this list:

- documenting a set of methods with single description (e.g. overloaded variants of the same thing, like DrawText), T++ is more flexible.

- putting comments into the code would practically disabled possibility to define inline methods within class (see all those simle modifier methods like EditField::Password).

Mirek
