Subject: "All shared" in Windows question
Posted by Mindtraveller on Mon, 13 Oct 2008 22:54:31 GMT
View Forum Message <> Reply to Message

I have a question about "All shared" option in Build options. As far as I understand it means compiling all the included packages into dynamic libraries (*.dll) and loading them dynamically to application executed.
I switched to "all shared" and tried to compile with MSC8 / MSC71 under Windows but TheIDE crashed each time.

Questions.
1) Do I understand "all shared" properly?
2) TheIDE crash is a bug or am I doing something wrong?
3) I`m making U++ based application with U++ based plugins. In order to solve potential memory managers conflicts I need libraries to be shared (this means compilation as dynamic libraries). Will it be enough?

Subject: Re: "All shared" in Windows question
Posted by Mindtraveller on Wed, 15 Oct 2008 14:20:59 GMT
View Forum Message <> Reply to Message

No ideas about "all shared"?

Subject: Re: "All shared" in Windows question
Posted by captainc on Wed, 15 Oct 2008 15:39:35 GMT
View Forum Message <> Reply to Message

Try to diff the output of the makefile generation of a few of the projects. I am trying that now myself to see what is different between the options.

Subject: Re: "All shared" in Windows question
Posted by Oblivion on Wed, 15 Oct 2008 21:05:51 GMT
View Forum Message <> Reply to Message

Quote:
No ideas about "all shared"?

AFAIK it is a posix/linux specific option (at least, for now) and links dynamically the created application to it's "dependencies" (gtk/kde libs etc.)

Subject: Re: "All shared" in Windows question

Posted by Mindtraveller on Thu, 16 Oct 2008 07:59:01 GMT
View Forum Message <> Reply to Message

Yes, it seems like "all shared" is truly a POSIX specific. Still we know that memory manager problem (main app / plugin) is solved by using U++ Core as dynamic library. I searched this forum and came to conclusion that there is no way to make U++ Core DLL-based without applying hacks. This makes classic plugin approach unworkable with U++.

There is one bigger issue with U++ versioning. Let`s imagine we managed to compile U++ libs as DLLs, wrote main application and released some number of U++ plugins for it. Then we distribute them and people start using our application.
After some time passed, we`ve decided to upgrade some plugins and added a pair of new ones. This time, we have U++ with newer version than one in DLLs distributed. And our newly written plugins will most likely crash on user machines where old U++ Core DLLs are resided.

This all means that each DLL should be standalone with U++ libraries statically linked. Yes, it is the hard way, but it is more likely the only way because of problems with U++ core libraries` dynamic linking and rapid U++ growth.

Subject: Re: "All shared" in Windows question
Posted by captainc on Thu, 16 Oct 2008 11:53:37 GMT
View Forum Message <> Reply to Message

Maybe we can take in incremental approach to it, like on a per package basis to see what kind of issues will come up. Maybe some part of U++ must remain statically linked for now, but some parts can be dynamically linked. It may be best to test it out with some easy packages ("pick the low hanging fruit") first.

Subject: Re: "All shared" in Windows question
Posted by cbpporter on Thu, 16 Oct 2008 21:05:00 GMT
View Forum Message <> Reply to Message

I believe there would be two solution two the problem.

1. Very careful monitoring of all breaking changes and make some mechanism for maintaining compatibility. Very difficult, and there will be times when breaking changes are introduced if you don't want to be stuck with the same features and design for years (Gtk is facing this problem right now: they can't do new stuff because of current interface limitations and adding new features would break link compatibility), you'll have to use a new version of the dynamic library.

2. Since releases are so infrequent, have a different dll for each version. Using an inappropriate version should not be permitted. This approach will only be useful if multiple U++ applications run on the same machine, by not replicating the same code in all the applications through static linking. I believe MFC uses this approach because they are not were not able to preserve compatibility.

Subject: Re: "All shared" in Windows question
Posted by mirek on Fri, 17 Oct 2008 13:45:00 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Mon, 13 October 2008 18:54I have a question about "All shared" option in Build options. As far as I understand it means compiling all the included packages into dynamic libraries (*.dll) and loading them dynamically to application executed.
I switched to "all shared" and tried to compile with MSC8 / MSC71 under Windows but TheIDE crashed each time.

Questions.
1) Do I understand "all shared" properly?

Yes. Anyway, it is sort of experimental feature for Win32...

Quote:
2) TheIDE crash is a bug or am I doing something wrong?

A bug (well, rather "failed experiment"

Quote:
3) I`m making U++ based application with U++ based plugins. In order to solve potential memory managers conflicts I need libraries to be shared (this means compilation as dynamic libraries). Will it be enough?

Well, this is a trouble with U++ / Win32. Anyway, why do you suppose to solve memory manager conflicts using .dll?

Mirek

Subject: Re: "All shared" in Windows question
Posted by Mindtraveller on Fri, 17 Oct 2008 15:18:08 GMT
View Forum Message <> Reply to Message

luzr wrote on Fri, 17 October 2008 17:45Anyway, why do you suppose to solve memory manager conflicts using .dll?

Thanks for a good question.
Just looked into command line in verbose mode and found "-MTd" switch selecting static linking of CRT. In theory, if I use dynamic CRT for main app and plugins it will share memory manager between them. And it will solve some low-level memory allocation issues.
Again, even if I manage to compile with "-MDd" switch, it will solve only half of a problem. Other half is equality of U++ Core objects versions, which will be unique in each plugin depending on U++ version it was developed with.

Subject: Re: "All shared" in Windows question
Posted by mr_ped on Fri, 17 Oct 2008 21:54:55 GMT
View Forum Message <> Reply to Message

Quote:Other half is equality of U++ Core objects versions, which will be unique in each plugin depending on U++ version it was developed with.

Just freeze the whole development platform including the tools for the actual application, so whenever you need to add new plugin, you will install the frozen environment from backup, and work there until you are done. Release plugin. Backup whole thing again.

In case you plan to let others to create plugin themselves, you are still in trouble. Then the best way is maybe to stick some of the stable releases for the whole life cycle of application. But in that perspective the u++ does change way too fast, even if it does only 1.5 of stable release per year, or how much of them happen at average.

I don't think freezing ABI between different upp versions is possible (within reasonable costs). Maybe it would be better to distribute whole application and plugins in source form, and let ever user compile with current upp. (just like linux and GNU is used)

Subject: Re: "All shared" in Windows question
Posted by mirek on Sat, 18 Oct 2008 06:58:24 GMT
View Forum Message <> Reply to Message

I was thinking about this issue from time to time and I think the one possible way worth investigating is plugin as separate process.

Of course, clearly depends on what plugins are supposed to do...

Anyway, one interesting advantage of possible plugin system via separate processes is possibility to even run on different machine (And e.g. communicate via TCP/IP or UDP...).

Mirek

Subject: Re: "All shared" in Windows question
Posted by Mindtraveller on Sun, 26 Oct 2008 01:18:54 GMT
View Forum Message <> Reply to Message

The main question here is if Core DLL will solve all the problems with memory management.

Finally I have a pair of little sample plugin apps. Both GUI. First application is linked into DLL and has 2 functions like Start and Finish. It is common U++ GUI project, with main window Execute() called from Start();
Second application is U++ EXE and loads this DLL dynamically with LoadDll__. Loading is successful, but when I call DLL`s Start() function, an exception is thrown. I thought it would not be a problem. In theory these modules have different memory managers and no problems should

appear. What is done wrong here?

Subject: Re: "All shared" in Windows question
Posted by mirek on Sun, 26 Oct 2008 06:57:12 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sat, 25 October 2008 21:18The main question here is if Core DLL will solve all the problems with memory management.

Finally I have a pair of little sample plugin apps. Both GUI. First application is linked into DLL and has 2 functions like Start and Finish. It is common U++ GUI project, with main window Execute() called from Start();
Second application is U++ EXE and loads this DLL dynamically with LoadDll__. Loading is successful, but when I call DLL`s Start() function, an exception is thrown. I thought it would not be a problem. In theory these modules have different memory managers and no problems should appear. What is done wrong here?

There are many many small details than can go wrong....

I think it can be heap, but it can be anything else as well. IMO, even the very simple thing like registering the same window class....

Everything also depends on HOW exactly you link the plugin..

Mirek

Subject: Re: "All shared" in Windows question
Posted by Mindtraveller on Sun, 26 Oct 2008 08:55:23 GMT
View Forum Message <> Reply to Message

```
///////MAIN EXE
typedef void (* PlugFunc)();
struct Plugin
{
 PlugFunc plugStart, plugFinish;
 bool activated;
 DLLHANDLE handle;

 Plugin() :plugStart(0), plugFinish(0), activated(false) {}
 void Start() {plugStart();}
};
Array<Plugin> plugins;
AdvPlugTest::AdvPlugTest()
{
```

```
 ::CtrlLayout(*this);

 plugStart <<= THISBACK(PluginStart);

 FindFile ff(DIR_PLUGINS+"*.dll");
 while (ff)
 {
  plugList.Add(ff.GetName());
  ff.Next();

  Plugin newPlugin;
  String pluginName = DIR_PLUGINS+ff.GetName();
  static const char *const names[3] = {"PlugStart", "PlugFinish",0};
  void *      procs[] = {&newPlugin.plugStart,&newPlugin.plugFinish,0};
  newPlugin.handle = LoadDll__(pluginName, &names[0], &procs[0]);

  plugins.Add(newPlugin);
 }
}
void AdvPlugTest::PluginStart()
{
 int row = plugList.GetClickRow();
 if (row<0 || row>=plugList.GetCount())
  row = 0;

 if (plugins[row].activated)
  return;

 plugins[row].activated = true;
 plugins[row].Start();
}

///////DLL
Plug *plug = 0;
extern "C"
{
 DLLEXPORT void PlugStart();
 DLLEXPORT void PlugFinish();
};
extern "C" void PlugStart()
{
 plug = new Plug();
 plug->Execute();
}

extern "C" void PlugFinish()
{
 delete plug;
```

```
}

Plug::Plug()
{
 ::CtrlLayout(*this, "Plugin window");
}
```

---

Subject: Re: "All shared" in Windows question
Posted by mirek on Fri, 31 Oct 2008 13:31:13 GMT
View Forum Message <> Reply to Message

I wonder what U++ version is this.

CtrlLayout has been moved into namespace long time ago....

Apart from that, I do not see anything particulary wrong. Anyway, devil is in details, especially as far as DLLs are considered...

Also, maybe in fact it would be possible, at least in Win32, to follow some sort of "clean cut" plugin DLL arrangements. That in fact would mean the exact opposite to current "SO" flag: we would avoid makeing any U++ symbols exported, only connecting plugin using narrow interface and regular Win32 libraries.

Of course, that would mean DHCtrl for plugin widgets and separated message loops. And duplicating all library stuff in plugin.

Mirek

---

Subject: Re: "All shared" in Windows question
Posted by cocob on Mon, 17 Nov 2008 11:17:23 GMT
View Forum Message <> Reply to Message

I think it would be nice to secure the output mode, because some combinations doesn't work or crash regulary.

Some examples :

- "all shared" with MSC
- "all shared" with blitz
- ...

and by the way i have again some problems with "all shared" and MINGW like compiling UWord example (problem with plugin/png) and some others with build order.

cocob

---

Subject: Re: "All shared" in Windows question
Posted by cocob on Mon, 17 Nov 2008 11:27:06 GMT
View Forum Message <> Reply to Message

Ok, for the problem with plugin/png i think a dependency to Draw package is missing

cocob

---

Subject: Re: "All shared" in Windows question
Posted by mirek on Mon, 17 Nov 2008 18:42:50 GMT
View Forum Message <> Reply to Message

cocob wrote on Mon, 17 November 2008 06:17I think it would be nice to secure the output mode, because some combinations doesn't work or crash regulary.

Some examples :

- "all shared" with MSC
- "all shared" with blitz
- ...

and by the way i have again some problems with "all shared" and MINGW like compiling UWord example (problem with plugin/png) and some others with build order.

cocob

There is "Lock link mode" Option in Build methods.

Perhaps it is time to start releasing U++ with this mode on....

Mirek

---

Subject: Re: "All shared" in Windows question
Posted by Factor on Thu, 20 Nov 2008 10:11:37 GMT
View Forum Message <> Reply to Message

I was able to compile the upp packages to shared lib with mingw 4.3 on windows (some minor changes needed only). During compilation the following warning message appears:

Info: resolving vtable for Upp::ImageEncoder by linking to __imp___ZTVN3Upp12ImageEncoderE
(auto-importd:/prg/mingw/bin/../lib/gcc/mingw32
 /4.3.0/../../../../mingw32/bin/ld.exe: warning: auto-importing has been activated without
--enable-auto-import specified on the command line.
This should work unless it involves constant data structures referencing symbols from
auto-imported DLLs.)

I've tested a lot of applications in the reference and example directory and all works as excepted.
My question is that how usable this method in a real world application that uses shared libs as plugins?