Subject: Support for plug-in architecture
Posted by slashupp on Sat, 21 Mar 2020 02:21:03 GMT
View Forum Message <> Reply to Message

(linux)

I have an app that will work well with plug-ins, i.e. using custom-controls that are compiled
into dynamic shared libs that can be loaded by the executable without needing to recompile the
app itself.

My current design generates all as one big blob-executable and needs to be re-build for each
small
change or addition of a custom-control, but would be much better using plug-ins.

Since Windows DLL's can be created, why can the linux-equivalent of dynamic shared object (.so)
not
be created? It looks like a simple change to compiler and linker flags that will enable this?

I've hacked the [Setup/Build methods] flags to produce a .so lib, which works with well with
'extern "C"'-functions, but fails to pass an accessable custom-control (which is mangled C++)
back.

Barring out-of-the-box support for plug-in/.so development, is there any other way to implement a
plug-in design using Upp?

Subject: Re: Support for plug-in architecture
Posted by mirek on Sat, 21 Mar 2020 08:54:27 GMT
View Forum Message <> Reply to Message

slashupp wrote on Sat, 21 March 2020 03:21(linux)

I have an app that will work well with plug-ins, i.e. using custom-controls that are compiled
into dynamic shared libs that can be loaded by the executable without needing to recompile the
app itself.

My current design generates all as one big blob-executable and needs to be re-build for each
small
change or addition of a custom-control, but would be much better using plug-ins.

Since Windows DLL's can be created, why can the linux-equivalent of dynamic shared object (.so)
not
be created? It looks like a simple change to compiler and linker flags that will enable this?

I've hacked the [Setup/Build methods] flags to produce a .so lib, which works with well with
'extern "C"'-functions, but fails to pass an accessable custom-control (which is mangled C++)
back.

Barring out-of-the-box support for plug-in/.so development, is there any other way to implement a plug-in design using Upp?

Not at the moment. Somehow this is not much required by typical U++ apps - which quite often are niche engineering applications or custom bussines solutions with tens to hunderds users. Not much need for plugins there.

That said, I am really not sure what is wrong with .so, IMO it should work for mangled C++ names as well.

Anyway, the trouble with .so and C++ is usually the problem that it is very fragile. Swap two virtual methods, add single member variable and the whole thing just explodes... :) When I was thinking about the best approach, I have tended to think about separate processes and RPC communication.

Subject: Re: Support for plug-in architecture
Posted by slashupp on Sat, 21 Mar 2020 12:15:20 GMT
View Forum Message <> Reply to Message

Hi mirek

I attach my attempt at creating .so for plug-ins.

There are two: plug_one that uses 'standard' c++ class, and the second tries to create a Upp-ctrl as plug-in.
The first works fine after a bit of a fiddle, but the second gives invalid access/segfault for reasons I am yet
to find.

I will appreciate if you or anybody checks over my code and hopefully spot the error.

To create the three packages I used two package dirs: 'temp' to hold the test-app and 'libs' to hold the plug-ins.
I then changed for the plugins the following:
In [Build/Output mode..] I set the 'Target file override' to "~/plug_in_libs/libplug_one.so" and "~/plug_in_libs/libmyctrl.so" after checking 'Release' for each plug-in.
The plugins must be compiled in Release mode.
Also for the plugins I changed each separately in [Setup/Build methods..] the 'Release options' by pre-pending -fPIC to it's content (-fPIC -O3 -ffunction-sections -fdata-sections),
and 'Release link options' to (-shared -Wl,--gc-sections,-soname,libmyctrl.so) and (-shared -Wl,--gc-sections,-soname,libplug_one.so)
for the two plug-ins

((POSSIBLE BUG: the [Setup/Build methods..] are not unique per open theide, a change in one gets reflected in others))

Compile the three packages now as you would do normally and the two .so's will be created in the ~/plug_in_libs directory.

Change the constants in 'plug_one.cpp' to reflect your path to the .so's.

File Attachments

1) testplugins.zip, downloaded 242 times

Subject: Re: Support for plug-in architecture
Posted by mirek on Sat, 21 Mar 2020 12:27:25 GMT
View Forum Message <> Reply to Message

Try USEMALLOC.

Subject: Re: Support for plug-in architecture
Posted by slashupp on Tue, 24 Mar 2020 17:20:57 GMT
View Forum Message <> Reply to Message

Tried USEMALLOC and all permutations of the flags and the best
I could find was GUI MT, but with all that, still no-go.

Strange thing is that the plug-in ctrl is correctly painted when I draw
lines around it's position in the parent window ... don't know why

I attach my current attempt, any help to get this working would be
greatly appreciated (I mean the alternative is Qt deargodno!)

Please help me to get this working in Upp.

thx

File Attachments

1) testplugins.zip, downloaded 232 times

Subject: Re: Support for plug-in architecture
Posted by slashupp on Wed, 25 Mar 2020 11:46:02 GMT
View Forum Message <> Reply to Message

Also found that if I call Show() on the plug-in right after AddChild() it shows-up correctly.

I made the plug-in ctrl copy-constructable and some of the custom functions is working now while others still crash.

Subject: Re: Support for plug-in architecture
Posted by mirek on Wed, 25 Mar 2020 11:55:37 GMT
View Forum Message <> Reply to Message

IF ctrlplug is the plugin, then the obvious problem is that you are linking it second set of U++ libraries. With all inlines we have, it is no wonder it crashes.

Try removing all "used" packages (obviously keep includes). I think the plugin should be just single package.

Mirek

Subject: Re: Support for plug-in architecture
Posted by slashupp on Wed, 25 Mar 2020 13:53:57 GMT
View Forum Message <> Reply to Message

Quote:Try removing all "used" packages (obviously keep includes). I think the plugin should be just single package.

I don't understand what you mean or how to do that.

Doesn't compile if I remove the "uses" clause from the .upp-file

Subject: Re: Support for plug-in architecture
Posted by mirek on Wed, 25 Mar 2020 14:35:11 GMT
View Forum Message <> Reply to Message

slashupp wrote on Wed, 25 March 2020 14:53Quote:Try removing all "used" packages (obviously keep includes). I think the plugin should be just single package.

I don't understand what you mean or how to do that.

Doesn't compile if I remove the "uses" clause from the .upp-file

I see :(Well, obviously, current build process might not be quite ready for what you need...

You could try, "all shared" option in build setup, but then you will need fake main package. "all shared" should create all packages as .so (except main) and then you should be able to use the single .so as the plugin.

All that said, I am pretty sure this all leads to nothing. What is the purpose to having this plugin architecture?

Mirek

Subject: Re: Support for plug-in architecture
Posted by slashupp on Wed, 25 Mar 2020 15:52:16 GMT
View Forum Message <> Reply to Message

Quote:You could try, "all shared" option in build setup, but then you will need fake main package. "all shared" should create all packages as .so (except main) and then you should be able to use the single .so as the plugin.

I'll try that, see how far I get.

Quote:What is the purpose to having this plugin architecture?

I'm writing an app to provide many services extending a {base/'kernel'} set of data & functions, with some of these
services mutually exclusive with respect to each other, and also allowing/enabling different other services. To
provide the services as plug-in's rather than a all-in-one-big-blob-app makes more sense and easier to develop
and maintain.

Subject: Re: Support for plug-in architecture
Posted by mirek on Wed, 25 Mar 2020 16:51:54 GMT
View Forum Message <> Reply to Message

slashupp wrote on Wed, 25 March 2020 16:52Quote:You could try, "all shared" option in build setup, but then you will need fake main package. "all shared" should create all packages as .so (except main) and then you should be able to use the single .so as the plugin.

I'll try that, see how far I get.

Quote:What is the purpose to having this plugin architecture?

I'm writing an app to provide many services extending a {base/'kernel'} set of data & functions, with some of these

services mutually exclusive with respect to each other, and also allowing/enabling different other services. To
provide the services as plug-in's rather than a all-in-one-big-blob-app makes more sense and easier to develop
and maintain.

Why do you think it will be easier to develop and maintain? It will be hell even if you will be able to make it work....