Subject: linking problems
Posted by alt3r on Tue, 29 May 2007 15:38:38 GMT
View Forum Message <> Reply to Message

hi all.
I'll try to explain my problem the best i can:
I've installed include and libs for 'open scene graph' (a 3d API development kit) and i have token one of the example files and put
in the 'MyApp' folder in the IDE directory.
But it does not compile.
It is strange since if I compile the example from its folder using makefile it compiles fine.

I hope the following helps...thx

----- osg_game (CONSOLE MAIN GCC DEBUG SHARED DEBUG_FULL LINUX)
main.cpp
osg_game: 1 file(s) built in (0:04.49), 4496 msecs / file, duration = 4500 msecs
Linking...
/home/sacha/upp/out/osg_game/sacha.Console.Debug_full.Main.N oblitz.Shared/main.o: In function `main':
/home/sacha/upp/MyApps/osg_game/main.cpp:21: undefined reference to `osg::ArgumentParser::ArgumentParser(int*, char**)'
/home/sacha/upp/MyApps/osg_game/main.cpp:24: undefined reference to `osg::ArgumentParser::getApplicationName() const'
/home/sacha/upp/MyApps/osg_game/main.cpp:25: undefined reference to `osg::ArgumentParser::getApplicationName() const'
/home/sacha/upp/MyApps/osg_game/main.cpp:26: undefined reference to `osg::ArgumentParser::getApplicationName() const'
/home/sacha/upp/MyApps/osg_game/main.cpp:27: undefined reference to `osg::ApplicationUsage::addCommandLineOption(std::basic_stri ng <char, st
d::char_traits<char>, std::allocator<char> > const&, std::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, std
::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)'
[...]
etc...
[...]
/home/sacha/upp/out/osg_game/sacha.Console.Debug_full.Main.N oblitz.Shared/main.o: In function `Optimizer':
/usr/include/osgUtil/Optimizer:65: undefined reference to `vtable for osgUtil::Optimizer'
/home/sacha/upp/out/osg_game/sacha.Console.Debug_full.Main.N oblitz.Shared/main.o: In function `~Optimizer':
/usr/include/osgUtil/Optimizer:66: undefined reference to `vtable for osgUtil::Optimizer'
collect2: ld returned 1 exit status

There were errors. (0:04.73)

Subject: Re: linking problems
Posted by alt3r on Tue, 29 May 2007 16:10:43 GMT
View Forum Message <> Reply to Message

sorry, the problem is not on compiling but in linking ..the main file compiles with no errors..

Subject: Re: linking problems
Posted by mr_ped on Tue, 29 May 2007 20:20:02 GMT
View Forum Message <> Reply to Message

you must also include libraries into linking process.
I'm still not sure how to do it properly in UPP, but I'm sure somebody else will help. (well, the most proper way is to make package from the library, and add the package to your main package, but I think the first step (converting library to package) may be sometimes quite difficult, if the library has lot of dependencies, etc... I don't have any experience with this).

Subject: Re: linking problems
Posted by alt3r on Wed, 30 May 2007 20:38:18 GMT
View Forum Message <> Reply to Message

thx ped for the answer..!
Lol i think it is a mess..i'm not so experienced as programmer..but i learn quickly.
Anyway ..I have understood that headers compiles the '.o' files..
and further '.o' files are linked to '.so' (libraries) if i'm not wrong..
So I need to create packages in UPP from '.so' files (3D API) and dependencies...
But when I go to create a package, why UPP ask to me for files like .cpp or .h ,and not '.so' or a library folder atlist ?
I'm a bit confused..lol

Subject: Re: linking problems
Posted by mr_ped on Thu, 31 May 2007 09:41:04 GMT
View Forum Message <> Reply to Message

.h and .c/.cpp (and some more extra extensions) are source files containing the source code from programmer.

Than there was in good old days of command line tools a Makefile file, which contained all rules how to compile the source code into final binary files. Upp handles rules in its own way, having all the things hard-wired do GCC or MSC compiler, with options scattered all over the place (Main package configuration, custom build steps, Setup Build methods, and so on...).
Basically it's much less powerful than classic Makefile, but it saves you some hassle as long as you play along Upp rules. (i.e. x86 binaries for win/linux)

The process of compilation does produce .o files (object files, that is (already) binary form of program instructions, but it's not connected to other .o files yet (it's not linked).

And finally to get working product you need to link all .o files together into some .exe or .so (library), .dll (dynamically linked library), and so on...

During the linking you may add to final product already prepared (outside from current source compilation) .o files, libraries (usually .a or .so files, which are basically an archive of .o files).

And during execution of final product there may be dynamic libraries (.dll in windows and .so in linux) loaded and linked during runtime.

(check for example http://www.ibm.com/developerworks/library/l-shobj/ for further description of differences between static and dynamic linking)

--- how does this relate to your problem ---

You want to use different library (open scene graph).

a) So you must include *.h files in your source to let know compiler about API of the library during compilation of .cpp sources into .o object files.

b) you must provide the .a / .so / .o / etc. i.e. binary library during linking so the compiler can produce final executable.

You must firstly know if you want to link statically or dynamically (shared) to that library. If the library does provide .so files, those are used in "shared" linking.
And you must add the path to those files somewhere in the build process. It's either possible (I think) to configure Upp to add external library to linking process (but I don't know how to do it, maybe somebody else will give you exact example). Or you may import the library sources as new package into Upp, set it up properly to compile from source, and create the (open scene graph) .o files inside Upp together with creation of your own .o files. Than the Upp will link that listed package automagically.

So either find out how to add external library, or import the Open scene graph into new package in Upp, and make it compile successfully. (may be lot of fun if it depends on other libraries, which you will have to turn into packages too.)

For example of domesticated libraries check in Upp "All packages" in Select main package, and look in uppsrc for plugin\z, plugin\png, and so on.. All those are external libraries converted into native Upp packages. (and adding them to your Upp application is as simple as doing "Add package" to your own project package, including ".h" in your source, and calling the functions ... nothing more to do then)

I know it's a long post, and still doesn't nail the problem down for you, but at least you may get better insight how the things works inside. And maybe after some toying around with this info your will solve your problem too.