Tutorial 1: Setup Codelite on Linux

[*This is ripped from the README and edited slightly for clarity*]

Orx's core is basically platform-independent. All the platform/OS-dependent features are implemented via plugins. These plugins can be loaded at runtime (hotplug) or they can be embedded at link-time.

If you use the non-embedded versions, you'll have to specify which plugins to use. This is more flexible but also requires additional files (the plugins themselves). The embedded version will store everything in Orx's library, meaning you won't be able to choose which plugin to use at runtime, but in exchange, Orx will be more compact and will also run considerably faster.

From the download page you'll find pre-compiled dynamic embedded binaries for Win32, Linux (x86) and MacOS X (ppc/x86). If you want to use the non-embedded versions (to use your own plugins) or the static ones, you will need to compile Orx yourself from the source. Everything compiles out-of-the-box for the hardware platforms cited above.

The embedded versions currently use:
- GLFW-based plugins for display, joystick, keyboard and mouse.
- OpenAL-based plugin for sound.
- Box2D-based plugin for physics.
- homemade plugin for 2D rendering.

Codelite Linux

Download file is orx-dev-linux-1.2.tar.bz2 (orx-dev-linux-*)

project root directory will be: ~/MyProject/

(For those who don't know, ~ refers to the HOME directory of the current user, so in this tutorial, my files will exist at /home/grey/MyProject/)

grab /bin/, /include/ and /lib/ from inside archive (under <archive>/orx-1.2/dev-linux/)

Please be aware, the files inside the archive are built with an older version of GCC and G++, so they will be incompatible with current releases. (And you are certain to get an error when attempting to link against them in this case.) I am not sure if Iarwain was planning on updating the 1.2 release, if not, you may be required to build all the required files from the SVN yourself.

Now we will set up our workspace in Codelite

Apparently this popup will not allow us to use the tilde (~) symbol to represent our home address, so ~/MyProject/ in my case becomes /home/grey/MyProject/

Next we add our project files

Again, we can't use the tilde symbol... guessing that's a feature which won't be changing.

Next we set up our general options

Our project file is stored at ~/MyProject/project/ so in order to put things in the right place, we want ../bin/ which becomes ~/MyProject/bin/

Same goes for release versions...

Compiler options next

The default project will place a 'main.cpp' file inside the ~/MyProject/project/ folder. We can move this, but to keep things straight forward, we do not remove the current directory include (“.;”) and instead simply add another for our include folder (“../include;”)

Release doesn't require the __orxDEBUG__ preprocessor directive. (That's 2 underscores on the start and end by the way.)

Finally, linker steps

Please be careful to ensure the release version is liborx, not liborxD like the debugging version.