Setting up Visual Studio (win) or Codelite (linux)

Setting up your Development Environment
[*This is ripped from the README and edited slightly for (hopefully) clarity*]

Please note this document needs to be reviewed.

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 Windows (32bit), 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.

Requirements

You will require one of the orx-dev-* files, depending on your IDE/OS from the download page. This will give you a copy of the pre-compiled Orx binary, as a dynamic library, with everything you need to get started, compiled in.

Extract the compress file.

Quick and Dirty SVN + VS2010 Users Start Here!

You will be presented with a number of folders: *bin*, *include* and *lib*.

For now I'm going to create the project in a specific location, feel free to substitute your own in as you see fit.

My project on Windows will be in “C:\MyProject\” and “~/MyProject/” or “/home/grey/MyProject/” on

Linux. Move the three folders previously mentioned, into this directory.

Attention Linux Users! 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. I plan to write up a tutorial for this later.

Making our IDE Project

Just 'click' these blue things for the IDE you're using!

Note, whatever you call your project, will have a folder created by that name (yes, even with the tick turned off!) So in my case, I now have “C:\MyProject\project\project.sln”.

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

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

From this point on, all of our directory structures will use these places as the “base” level. (“C:\MyProject\” and ”~/MyProject/”)

Adding New Files

Next, add a new file to the project:

Attention: Codelite automatically adds a main.cpp file to it's project, so you CAN skip this step if you want!
Those who have not skipped, we're going to -delete- the default main.cpp file and create our own, in

the correct place (~/MyProject/Source/)

Setting Project Properties

Then, it's time to change the project properties:

– fixme moment!

General Options

We set up the general options:

Remember, our 'base' level is here: “C:\MyProject\project\project.sln”, so this will point to “C:\MyProject\bin\”

Remember, our 'base' level is here: “C:\MyProject\project\project.sln”, so this will point to “C:\MyProject\bin\”

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 compiler options:

The preprocessor only needs this addition for debug builds, no release. :) (That's 2 underscores on the start and end by the way.)

Next activate debugging on the debug build:
And deactivate debugging on the release build:
Deactivate optimization on the debug build:

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.)

Linker Options

The linker needs to be configured too:

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

See Also

This article is part of a series!
Series: Grey's tutorials
Prev: "Tutorial 0: Build Orx From Source!"
Next: "Tutorial 2: Stand Alone Application."