Creating your own Orx-based Project using 'init'

Orx supports the creation of your own game projects by providing a very useful script called: `init`

Depending on your operating system, `init` will create a project for Visual Studio, Codelite, Code::Blocks, XCode, and gmake.

This is available in the root of your Orx folder. [1)]

There are two commands:

- `init.sh` for Mac or Linux
- `init.bat` for Windows.

Interactive Mode

cd your way into the orx folder and type the `init` command without any parameters or double click the file from your desktop. You will enter interactive mode:

```
c:\Work\orx>init

== No argument, switching to interactive mode
 * Project name (relative or full path)?
```

You only need to specify a full path or relative path in order to create and name a project. For example, you could enter a direct path like:

`C:\Work\MyGame`

Then the folder above will be created, and `MyGame` will be name of the Solution / Workspace / or Project.

In the same way you could enter a relative path. For example on Linux or Mac:

`~/Documents/MyGame`

After pressing Enter, choose one or more extensions (more on this in the next section).

Getting Help

To see all commandline options:

```
$ ./init --help
== Usage: C:\Work\orx\init.bat name [+/-archive] [+/-c++] [+/-imgui] [+/-
nuklear] [+/-scroll]
```

```
  - name: Project name (relative or full path), required
  - archive: orxArchive support (resources can be stored inside ZIP
files)=[no], optional
  - c++: Create a C++ project instead of C=[yes], optional
  - imgui: Dear ImGui support (https://github.com/ocornut/imgui)=[no],
triggers [+c++], optional
  - nuklear: Nuklear support
(https://github.com/immediate-mode-ui/nuklear)=[no], triggers [+c++],
optional
  - scroll: C++ convenience layer with config-object binding=[no], triggers
[+c++], optional
```

There are additional options to create a Scroll-based or ImGui-based project (or both).

Screen Output

You will see the following as the various projects and solutions are created for many IDEs:

```
c:\Work\orx>init c:\temp\MyGame
[ 12:56:59 ] Initializing [ MyGame ] in [ c:\temp\ ]
[ 12:56:59 ] == Creating files:
  + MyGame\.editorconfig
  + MyGame\build\premake4.lua
  + MyGame\data\config\MyGame.ini
  + MyGame\data\config\MyGamed.ini
  + MyGame\data\config\MyGamep.ini
  + MyGame\data\sound\appear.ogg
  + MyGame\data\texture\logo.png
  + MyGame\src\MyGame.cpp
[ 12:57:01 ] Generating build files for [ windows ]:
  * gmake
Building configurations...
Running action 'gmake'...
Generating windows/gmake/Makefile...
Generating windows/gmake/MyGame.make...
Done.
  * codelite
Building configurations...
Running action 'codelite'...
Generating windows/codelite/MyGame.workspace...
Generating windows/codelite/MyGame.project...
Done.
  * codeblocks
Building configurations...
Running action 'codeblocks'...
Generating windows/codeblocks/MyGame.workspace...
```

```
Generating windows/codeblocks/MyGame.cbp...
Done.
  * vs2013
Building configurations...
Running action 'vs2013'...
Generating windows/vs2013/MyGame.sln...
Generating windows/vs2013/MyGame.vcxproj...
Generating windows/vs2013/MyGame.vcxproj.user...
Generating windows/vs2013/MyGame.vcxproj.filters...
Done.
  * vs2015
Building configurations...
Running action 'vs2015'...
Generating windows/vs2015/MyGame.sln...
Generating windows/vs2015/MyGame.vcxproj...
Generating windows/vs2015/MyGame.vcxproj.user...
Generating windows/vs2015/MyGame.vcxproj.filters...
Done.
  * vs2017
Building configurations...
Running action 'vs2017'...
Generating windows/vs2017/MyGame.sln...
Generating windows/vs2017/MyGame.vcxproj...
Generating windows/vs2017/MyGame.vcxproj.user...
Generating windows/vs2017/MyGame.vcxproj.filters...
Done.
[ 12:57:01 ] Init successful!
```

Output

The `init` command will create a folder `MyGame` in the folder path specified and will give you the following sub-directories:

- build
- data
- src

`build` will contain builds for all the IDEs for your operating system. Just pick the one you want to use.

`data` contains some sample config files, sounds and textures.

`src` contains a basic setup source file.

You don't need to bring in the `include` or `lib` folders from the Orx folder. Your $(ORX) variable will enable your project to see those dependencies from the Orx folder automatically. Therefore your project can be created anywhere, and will still compile fine.

In the same way, you don't need to manually copy over orx*.dll files (or orx*.so) files, as your project is already configured to copy these into the `bin` folder for you on each compile.

That's about it. A lot of work has gone into making this as bullet proof as possible, so you should have no trouble spinning up new projects whenever you need one.

Extensions

You can also create orx/Scroll-based projects for an object oriented way of using Orx with c++.

For making Orx applications with a UI, like applications and editors, you can create Dear ImGui-based projects.

Or a combination of both.

C or C++ projects

This extension allows you to specify if you want a C or C++ based project. However some extensions like `scroll` or `imgui` require +c++ and are incompatible. For example, this will give an error:

```
./init.bat /temp/projecttest +scroll -c++
[ 8:10:49 ] == [ scroll ] triggers [ +c++ ]
== Aborting, the following extensions have been both required and
prohibited: [ c++ ]
```

Troubleshooting

Problem 1: If you receive an error compiling your own game project with something like:

cannot find -lorxd

This is because you either:

1. Did not compile Orx itself first. The Orx library needs to be compiled so that projects created with `init` can make use of Orx's dll(s) that reside there. If they are missing you will get linker errors trying to find orx.dll or orx.so libraries;
2. Did not compile all three required configurations. Compile `Debug`, `Profile` and `Release`.
3. Previously compiled the Orx library as 32-bit, whereas you are trying to compile your own project in 64-bit or vice versa.
4. Previously compiled the Orx library using one compiler whereas you are compiling your project with another. For example: you can't compile your project using the Visual Studio compiler if you compiled the Orx library with mingw.

Problem 2: The dll files at the $ORX location are not being copied into my game's `bin` folder.

Ensure you have compiled all three required Orx library configurations. Compile `Debug`, `Profile` and

`Release`. If the post-event copy step in your game project cannot find all three files at $ORX, then none will be copied over to your project's `bin` folder.

[1)]

This is only available with the git version of Orx. See: Cloning Orx from Github