Read the Docs Template Documentation

Release 1.0

Read the Docs

Feb 16, 2021

Contents

Easing C++ development, inciting code reuse and improving application end-users experience by simplifying software updates.

CHAPTER 1

Getting started

Download nxxm and clone the Get Started repository repository.

CHAPTER 2

nxxm features

To discover all the marvellous features nxxm offers you can take a look on our https://nxxm.github.io/ website.

- **nxxm makes it intuitive to build a C++ project**
  - Requires absolutely no build recipes
  - Builds by conventions with `nxxm` .
- nxxm is a dependency manager for C++ which fetches and compile any C++ project hosted on GitHub.com .
- adds software upgrade support to your apps.
- WebAssembly Ready but not only.

CHAPTER 3

nxxm key principles

3.1 Relaxing & flowing C++

- Code scanning & conventions over build configuration
- 0 setup just coding
  - Just select one environment from our Supported list or specify your own.
  - nxxm.io will download & install the compiler and libraries automatically in an isolated sandbox.

3.2 Every project is a library

In a software project there are 2 kinds of entrypoints :

- Developers entrypoints for code reuse
- End-user entrypoints for application use

Therefore the nxxm tools always prepare out of any project, even if it consists of a single C++ header and implementation file : a library that might be reused and applications that might be shipped.

3.3 Don't pay for what you don't use

This is a core C++ design philosophy, and sadly the world of packages manager obliges you to take more than you need.

By definition a package is a pack of alot of things, and a developer won't need all of them.

nxxm allows you to do a fine-granular selection of your dependencies and pulls in your final application, thanks to modern C++ compilers only the needed bits.

3.4 Opinionated but compromise-ready

While with nxxm you won't need anymore to write build files, you can still customize the parts or all with `CMakeLists.txt.tpl` files as we drive CMake internally.

We don't encourage it though.

CMake is a really robust solution that we cherish, but in our opinion it is a too low-level tool in a modern demanding C++ context.

CHAPTER 4

Contents:

4.1 Builds by Conventions

4.1.1 Why ?

Why did we ever write makefiles ? CMakeLists.txt or configured IDE project settings ? And why can we stop ?

Andrew Koenig once coined FTSE, the Fundamental theorem of software engineering :

> "We can solve any problem by introducing an extra level of indirection."

Actually building application is an extremely complex problems, and the layers are almost infinite : linker, assembler, compiler, frontend, build-system (*e.g.* make), meta-build-system (*e.g.* cmake) to quote only a few.

nxxm leverages all the fabulous work done in these layer to finally make building C++ a simple human task.

4.1.1.1 Because

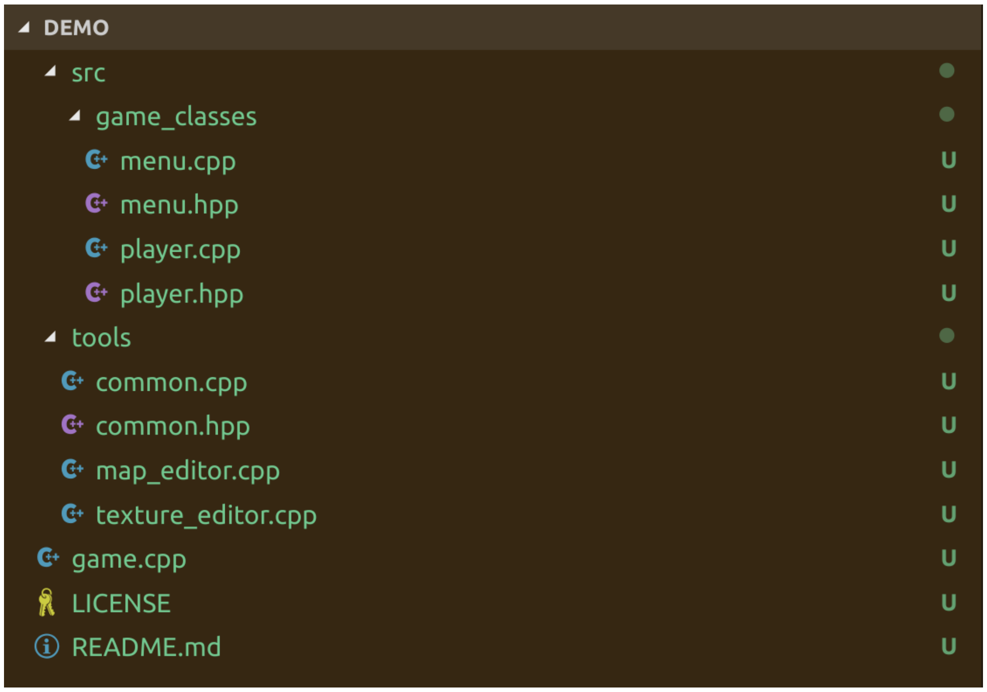
- C++ code expresses enough what is an application entrypoint and what is reusable library code.
- No need to learn a new language to specify how to build.
- We are tired of specifying each single file that should be built.

4.1.2 Convention by example

Take the idea of building a game project. This game project will contain :

1. the game domain with the player characters, game menu and so on.
2. the game app itself
3. tools apps like game maps and texture editor.

The directory could look like this :



```
DEMO
  src                          ●
    game_classes               ●
      menu.cpp                 U
      menu.hpp                 U
      player.cpp               U
      player.hpp               U
  tools                        ●
    common.cpp                 U
    common.hpp                 U
    map_editor.cpp             U
    texture_editor.cpp         U
  game.cpp                     U
  LICENSE                      U
  README.md                    U
```

game.cpp has a main function each, therefore they are **apps** by convention.

The main function may look like :

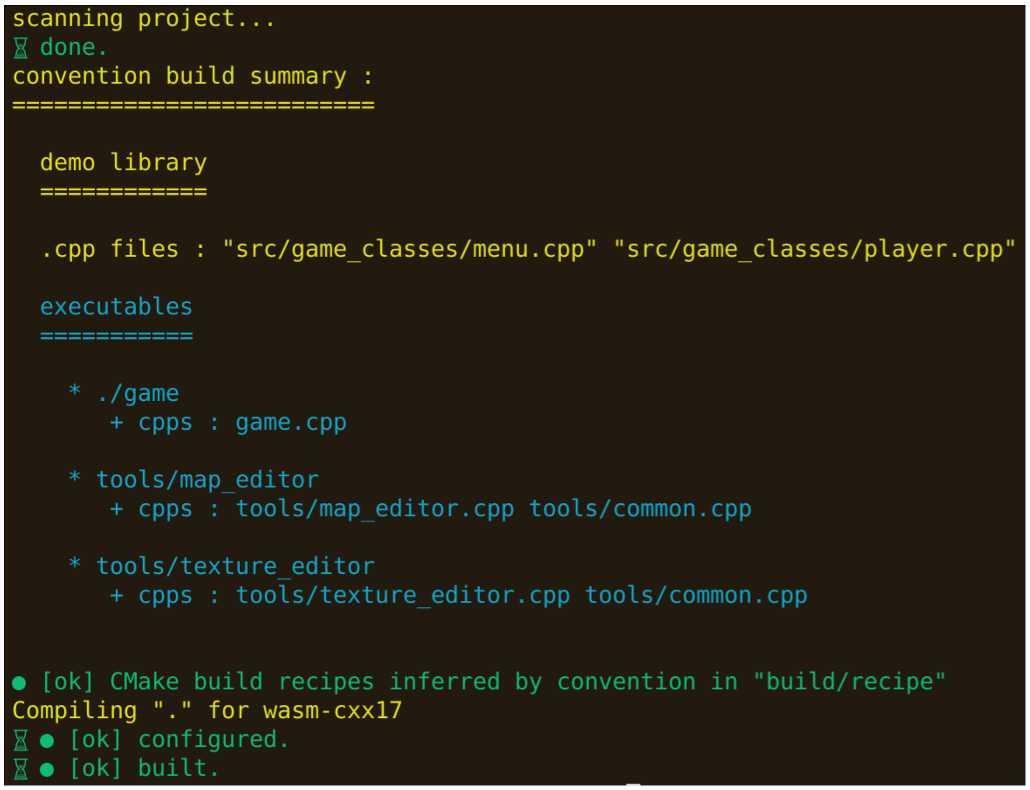
```
#include <iostream>

void main() {

  std::cout << "☺ Welcome into a relaxing C++ experience !" << std::endl;

}
```

The *src/* folder is scanned by nxxm which notices that there are *.hpp* and *.cpp* that don't have any entrypoint, therefore it defaults to the library code (*i.e.* the game domain).

Finally *tools/* isn't quite a good name for library code, and there are `map_editor.cpp` and `texture_editor.cpp` that both have `main()` functions. Which makes them app entrypoint. Therefore the **apps** convention kicks in.

This **apps** convention allows to have supporting file aside, therefore common.cpp is linked to map_editor and texture_editor. The header common.hpp is accessible via `#include "common.hpp"` while the game_classes is exported on the compiler include dirs which makes them usable via `#include <game_classes/*.hpp>`.

nxxm will give the following summary:



```
scanning project...
⌛ done.
convention build summary :
==========================

  demo library
  ============

  .cpp files : "src/game_classes/menu.cpp" "src/game_classes/player.cpp"

  executables
  ===========

    * ./game
       + cpps : game.cpp

    * tools/map_editor
       + cpps : tools/map_editor.cpp tools/common.cpp

    * tools/texture_editor
       + cpps : tools/texture_editor.cpp tools/common.cpp


● [ok] CMake build recipes inferred by convention in "build/recipe"
Compiling "." for wasm-cxx17
⌛ ● [ok] configured.
⌛ ● [ok] built.
```

Resulting in a bin folder like the following:

```
bin
├── game.exe                // game.cpp
├── demo.lib                // game_classes *.cpp
└── tools
    ├── map_editor.exe      // common.cpp, map_editor.cpp
    └── texture_editor.exe  // common.cpp, texture_editor.cpp
```

4.1.3 Conventions Types

First main convention : every project is a library. (*c.f. Every project is a library*).

Out of each git repositories passed to the nxxm program one library is produced.

Hint: All the conventions can be mixed and don't need to be declared beforehand, they are getting detected.

Hint: While often unneeded due to the conventions mentioned below it is possible to specify or reduce the scope of a library within a project with `nxxm -s library-dir` or `{ "s" : "library-dir" }`.

4.1.3.1 splitted libs

```
.
├── include
```

(continues on next page)

(continued from previous page)

```
│   └── header.hpp
└── src
    └── impl.cpp
```

One typical kind of c++ project is when headers and implementation files are splitted in different folders, if it happens to be so nxxm will consider the include/ folder to be the publicly installable headers and src the implementation files constituing the library.

Hint: By default nxxm checks the presence of `include/ inc/ src/ sources/` to infer this convention.

4.1.3.2 samedir libs

```
.
└── src
    ├── header.hpp
    └── impl.cpp
```

Another typical convention C++ programmer uses are the implementation and headers together at the same level of directory hierarchy.

Hint: By default nxxm checks the presence of `src/ sources/` to infer this convention and the absence of main() functions in the files.

4.1.3.3 toplevel libs

These are libraries that don't have any special source folder, their headers are directly rooted at the top of their repositories.

When this is detected the same mechanism as for **samedir libs** applies.

Hint: With this kind of structure it might be required to disambiguate nxxm to tell him which directories are really part of the lib thanks to `nxxm -s library-dir` or `{ "s" : "library-dir" }`.

4.1.3.4 headeronly libs

It is possible to have code which is completely header only while application entrypoints are in .cpp files aside materializing either lib **examples** or a corresponding **app**.

The headers will be put at disposal like in the aforementioned conventions with `#include <>`.

4.1.3.5 apps

any .cpp file with an entrypoint is an app.

For example any file containing a main() function or a macro instantiating a main() function (*e.g.* unit testing frameworks) will be compiled as an application.

Hint: apps are always linked to the project library.

Hint: if others .cpp files are aside in the same or deeper filesystem directory they get linked with the applications in question. Except when those directories are part of the explicitely declared "s"/-s project library dir.

4.1.3.6 test or examples

Same as **apps** convention, however the project will register them within the CMake CTest test driver and calling `nxxm . --test all` will run them all and report result status

This convention kicks-in when files with main() functions in parent folder are named after :

- test
- tests
- example
- examples

4.1.3.7 html

Any .html containing `<script type="text/c++"></script>` in it is compiled as an **app** convention.

4.1.4 Conventions are not enough

It could happen, please contact us so that we can improve nxxm or help you.

You can also tweak the build as explained below, this is however not recommended and goes against our vision. But we don't bite. :)

There is for sure a way for the convention build to work : less is more. Or put differently less CMakeLists is more time for your C++. (^^)

4.1.4.1 Override for one directory convention build

This can be useful for really custom test framework or cases, you can give the hand to your CMake skills by adding in the subdiretories you don't want nxxm to do conventional builds.

Simply add an empty marker file *use-cmake.nxxm* and a valid *CMakeLists.txt*. The build will use CMake for this subpart.

4.1.4.2 Tweak nxxm convention build

We rely on CMake on you can tweak how we interract with it.

We don't recommend it but you can tweak fully or partially the build by adding `CMakeLists.txt.tpl` files in the main or sub directories of your project.

To generate a sample CMakeLists.txt.tpl with the docs embedded of the different variables at your disposal call *nxxm cmaketpl*.

4.2 Dependencies Specification

If your project isn't dependency free then you can consume any GitHub.com repository or CMake Hunter Provided packages.

This simply can be specified in a `.nxxm/deps` which can look like this:

```
{
    "gh-user/repo" : {}

  , "nxxm/gh"      : { "@" : "v0.0.1" }

  , "platform" : [ "Boost::+boost" ]
}
```

This dependency specification (*i.e.* depspec) file tells which libraries your project needs.

Every *key* of the JSON object represent a github URI.

- "gh-user/repo": repository at https://github.com/gh-user/repo
- "nxxm/gh": repository at https://github.com/nxxm/gh
- "platform" however specifies a commonly consumed package that have been tested and integrated on all the toolchains supported by nxxm.

The effect of having such a `.nxxm/deps` file is that on `nxxm .` call the GitHub.com repositories will be fetched, compiled by convention and installed within the `/build/<platform>/sysroot`.

In this specific case :

- "gh-user/repo" default branch (usually master) will be taken and always the latest. Unless `nxxm . -n` is passed which relies then only on it's previously downloaded cache.
- nxxm/gh tag v0.0.1 will be fetched once and always used
- Boost headers distribution will be downloaded once and installed to the build thanks to CMake Hunter.

4.2.1 .nxxm/deps syntax

A JSON object whose **keys** are GitHub URI and values configurations to consume those repositories as C++ libraries dependencies.

4.2.1.1 gh-user/gh-repo

The following attributes are possible to declare how the dependencies should be consumed.

```
{
    "gh-user/repo" : {
        "@" : "<branch/tag/name>"
      , "s" : ["<src-disambiguation>", ...]
      , "x" : ["<exclude dir>", ...]

      , "@:<target>" : "<branch/tag/name>"
      , "s:<target>" : ["<src-disambiguation>", ...]
      , "x:<target>" : ["<exclude dir>", ...]

      , "requires" : { ... }
```

(continues on next page)

(continued from previous page)

```
    }

   "platform[:target-platform]" : ["<dep>::<component>", ...]

}
```

Tip: The attributes are optional. They can all be ommitted.

4.2.1.1.1 @ : tag or branch name

- *ommited* : in this case the default branch is taken is fetched each time for the latest version unless `-n` is passed to nxxm.
- a **tag** is fetched only once, then the version is kept.
- a **branch name** is fetched each time for the latest version unless `-n` is passed to nxxm.
- You can suffix the key it with the target platform to selectively use a dependency in different platforms.

  e.g. `"@:wasm-cxx17" : "v0.0.1"` will select the version v0.0.1 for WebAssembly but not for the native platforms

4.2.1.1.2 s : source dir disambiguation

If a repository has alot of sources directories with uncommon name they can be added to the list of includes or files to link with s.

You can suffix the key it with the target platform to selectively include implementation dir by platform

> e.g. `"s:vs-15-2017-win64-cxx17" : ["src/visual-c"]` will compile with the `src/visual-c` on vs-15-2017 but not on other targets.

4.2.1.1.3 x : directory to completely ignore

Directories that are unneeded to scan. Usually you don't need to specify this. Note that directories starting with a .dot will always be ignored.

You can suffix the key it with the target platform to selectively include implementation dir by platform

> e.g. `"x:wasm-cxx17" : ["src/native-code"]` will compile without the native code directory for the WebAssembly platform.

4.2.1.1.4 requires

The requires is a way to adapt a non nxxm dependency which also has dependencies, there are no limits on the nesting you can use.

It is also really useful to change a transitive dependency, for example if you prefer to use BoringSSL in place of OpenSSL for a libary which would depend on OpenSSL.

4.2.1.1.5 platform[:target-platform]

Tip: For a list of possible platform libraries please refer to *Platform Dependencies Shorthands*.

```
"platform[:target-platform]" : ["<dep>::<component>", ...]
```

It's possible to specify dependencies that we consider platform provided. Meaning they are really common and used accross almost any project, but still needs to be specified.

`:target-platform` can be appended to selectively include dependencies only on certain target platform, hence the key name. The target platform is selected after the *nxxm -t target-platform* parameter.

If there is a `platform` and a `platform::target` both will be used together.

The platform libraries have to be specified as follow :

- "PackageName::+component" if the component is an option of PackageName to be linked but is always shipped with PackageName (*e.g.* header only Boost distribution via "Boost::+boost" is always shipped, we need to declare that we use it.).
- "PackageName::component" if the component is to be linked and needs to be fetched separately. (*e.g.* "Boost::filesystem" is not shipped per-se by Boost it must be declared as to install in sysroot first.").
- "target::native-name" if the component is already installed on such platforms and should be used. (*e.g.* linkign to libdl.so on linux can be specified by `target::dl`)

Tip: For a list of possible platform libraries please refer to *Platform Dependencies Shorthands*.

4.2.1.1.5.1 platform vs GitHub.com

We made the choice to provide the ability to consume well-known C++ libraries via the "platform" library specification.

This makes their usage more common and via a single inclusion without needing to search the exact repository on github.

4.3 GitHub.com Dependencies

Hint: Specify any GitHub.com dependency as *Dependencies Specification*

Use the awesome Github Search Engine to find your C++ dependencies, there are more than 763 000 repositories.

4.4 Platform Dependencies Shorthands

Hint: Specify the platform spec as in *Dependencies Specification*

We rely on the https://docs.hunter.sh/en/latest/ project for the platform dependencies, you can see the version of the platform libraries used by default by `nxxm` in it's Hunter config.

- **accelerate:**
  - `accelerate::accelerate`
- **acf:**
  - `acf acf::acf`
- **aes:**
  - `aes::aes`
- **aglet:**
  - `aglet::aglet`
- AllTheFlopsThreads:
- Android-Apk:
- android_arm64_v8a_system_image_packer:
- Android-ARM64-v8a-System-Image:
- android_arm_eabi_v7a_system_image_packer:
- Android-ARM-EABI-v7a-System-Image:
- android_build_tools_packer:
- Android-Build-Tools:
- android_google_apis_intel_x86_atom_system_image_packer:
- Android-Google-APIs-Intel-x86-Atom-System-Image:
- android_google_apis_packer:
- Android-Google-APIs:
- android_google_repository_packer:
- Android-Google-Repository:
- android_intel_x86_atom_system_image_packer:
- Android-Intel-x86-Atom-System-Image:
- **android_log:**
  - `android_log::android_log`
- android_mips_system_image_packer:
- Android-MIPS-System-Image:
- Android-Modules:
- **android:**
  - `android::android`
- android_sdk_packer:
- android_sdk_platform_packer:
- Android-SDK-Platform:

- android_sdk_platform_tools_packer:
- Android-SDK-Platform-tools:
- Android-SDK:
- android_sdk_tools_packer:
- Android-SDK-Tools:
- android_support_repository_packer:
- Android-Support-Repository:
- **AngelScript:**
  - `boo PUBLIC AngelScript::AngelScript`
- **appkit:**
  - `appkit::appkit`
  - `"-framework AppKit"`
- **ARM_NEON_2_x86_SSE:**
  - `ARM_NEON_2_x86_SSE::ARM_NEON_2_x86_SSE`
- **ArrayFire:**
  - `ArrayFire::af`
  - `ArrayFire::afcpu`
- **assetslibrary:**
  - `assetslibrary::assetslibrary`
  - `"-framework AssetsLibrary"`
- **Assimp:**
  - `Assimp::assimp`
- **Async++:**
  - `Async++::Async++`
- **audiotoolbox:**
  - `audiotoolbox::audiotoolbox`
  - `"-framework AudioToolbox"`
- **audiounit:**
  - `audiounit::audiounit`
  - `"-framework AudioUnit"`
- **autobahn-cpp:**
  - `autobahn-cpp::autobahn-cpp`
- autoutils:
- **Avahi:**
  - `Avahi::common Avahi::client Avahi::compat_libdns_sd`
- **avfoundation:**

  - `avfoundation::avfoundation`
  - `"-framework AVFoundation"`
- **Beast:**
  - `Beast::Beast`
- **benchmark:**
  - `benchmark::benchmark`
- bison:
- **BoostCompute:**
  - `BoostCompute::boost_compute`
- **boost-pba:**
  - `boost-pba::boost-pba`
- **BoostProcess:**
  - `BoostProcess::boost_process`
- **Boost:**
  - `Boost::+boost`
  - `Boost::system Boost::filesystem`
  - `Boost::` followed by any Boost Library name.
- **BoringSSL:**
  - `boo BoringSSL::ssl BoringSSL::crypto`
- **Box2D:**
  - `boo PUBLIC Box2D::Box2D`
- **bullet:**
  -
- **BZip2:**
  - `BZip2::bz2`
- **caffe:**
  - `caffe`
- **CapnProto:**
  - `CapnProto::capnp`
- **carbon:**
  - `carbon::carbon`
  - `"-framework Carbon"`
- **c-ares:**
  - `c-ares::cares`
- Catch:
- catkin:

- cctz:
- **ccv:**
  - `ccv::ccv`
- **cereal:**
  - `cereal::cereal`
- **ceres-solver:**
  - `PRIVATE ceres`
- check_ci_tag:
- **civetweb:**
  - `boo PUBLIC civetweb::c-library`
- Clang:
- ClangToolsExtra:
- CLAPACK:
- clBLAS:
- CLI11:
- **Comet:**
  - `Comet::comet`
- **convertutf:**
  - `convertutf::convertutf`
- **coreaudio:**
  - `coreaudio::coreaudio`
  - `"-framework CoreAudio"`
- **coredata:**
  - `coredata::coredata`
  - `"-framework CoreData"`
- **corefoundation:**
  - `corefoundation::corefoundation`
  - `"-framework CoreFoundation"`
- **coregraphics:**
  - `coregraphics::coregraphics`
  - `"-framework CoreGraphics"`
- **corelocation:**
  - `corelocation::corelocation`
  - `"-framework CoreLocation"`
- **coremedia:**
  - `coremedia::coremedia`

  - `"-framework CoreMedia"`
- **coremotion:**
  - `coremotion::coremotion`
  - `"-framework CoreMotion"`
- **corevideo:**
  - `corevideo::corevideo`
  - `"-framework CoreVideo"`
- CppNetlib:
- **CppNetlibUri:**
  - `network-uri`
- **cpp_redis:**
  - `cpp_redis::cpp_redis`
- **cpr:**
  - `cpr::cpr`
- **crashpad:**
  - ‘‘ crashpad::crashpad_client‘‘
- crashup:
- **crc32c:**
  - `crc32c::crc32c`
- **cryptopp:**
  - `cryptopp-static`
- **CsvParserCPlusPlus:**
  - `CsvParserCPlusPlus::csv_parser_cplusplus`
- ctti:
- **cub:**
  - `cub::cub`
- **CURL:**
  - `CURL::libcurl`
- **cvmatio:**
  - `cvmatio::cvmatio`
- cvsteer:
- cxxopts:
- czmq:
- damageproto:
- date:
- dbus:

- **debug_assert:**
  - `debug_assert_example debug_assert`
- **dest:**
  - `dest::dest`
- dlib:
- dmlc-core:
- **doctest:**
  - `doctest::doctest`
- **double-conversion:**
  - `double-conversion::double-conversion`
- dri2proto:
- dri3proto:
- drishti_assets:
- drishti_faces:
- drishti:
- drm:
- duktape:
- dynalo:
- **egl:**
  - `egl::egl`
- eigen3-nnls:
- **Eigen:**
  - `Eigen::eigen`
- enet:
- EnumGroup:
- **eos:**
  - `eos::eos`
- ethash:
- **Expat:**
  - `${EXPAT_LIBRARIES}`
- **FakeIt:**
  - `FakeIt::FakeIt`
- **farmhash:**
  - `farmhash farmhash::farmhash`
- **fft2d:**
  - `fft2d fft2d::fft2d`

- fixesproto:
- **flatbuffers:**
  - `flatbuffers::flatbuffers`
- **flex:**
  - `main ${FLEX_LIBRARIES}`
  - `main ${BISON_LIBRARIES} ${FLEX_LIBRARIES}`
- **fmt:**
  - `fmt`
- folly:
- :
- **forcefeedback:**
  - `forcefeedback::forcefeedback`
  - `"-framework ForceFeedback"`
- **foundation:**
  - `foundation::foundation`
  - `"-framework Foundation"`
- **freetype:**
  - `freetype::+freetype`
- frugally-deep:
- Fruit:
- FunctionalPlus:
- **gamecontroller:**
  - `gamecontroller::gamecontroller`
  - `"-framework GameController"`
- gauze:
- **gemmlowp:**
  - `gemmlowp gemmlowp::gemmlowp`
- geos:
- getopt:
- **gflags:**
  - `gflags`
- **giflib:**
  - `giflib giflib::giflib`
- **glapi:**
  - `glapi::glapi`
- **glbinding:**

  - `glbinding glbinding::glbinding`
- **gles2:**
  - `gles2::gles2`
- **gles3:**
  - `gles3::gles3`
- **glew:**
  - `boo PUBLIC glew::glew`
- **glfw:**
  - `glfw`
- **glib:**
  - `PkgConfig::glib-2.0`
- **glkit:**
  - `glkit::glkit`
  - `"-framework GLKit"`
- **glm:**
  - `PRIVATE glm`
- **globjects:**
  - `globjects::globjects`
- **glog:**
  - `glog::glog`
  - `glog`
- glproto:
- glslang:
- GPUImage:
- **gRPC:**
  - `gRPC::grpc`
- **GSL:**
  - `GSL::gsl`
- **gst_plugins_bad:**
  - `PkgConfig::gstreamer-bad-video-1.0`
- **gst_plugins_base:**
  - `PkgConfig::gstreamer-video-1.0`
- gst_plugins_good:
- gst_plugins_ugly:
- **gstreamer:**
  - `PkgConfig::gstreamer-1.0`

- **GTest:**
  - `GTest::+gtest_main) # GTest::gtest will be linked automatically`
  - `GTest::+gtest`
  - `GMock::+gmock_main`
- **gumbo:**
  - `gumbo::gumbo`
- h3:
- **half:**
  - `half::half`
- harfbuzz:
- **hdf5:**
  - `hdf5`
- **highwayhash:**
  - `highwayhash highwayhash::highwayhash`
- http-parser:
- ice:
- ICU:
- **IF97:**
  - `IF97 IF97::IF97`
- Igloo:
- **imageio:**
  - `imageio::imageio`
  - `"-framework ImageIO"`
- imgui:
- **imshow:**
  - `imshow::imshow`
- **inja:**
  - `inja inja::inja`
- inputproto:
- intltool:
- **intsizeof:**
  - `PUBLIC intsizeof::intsizeof`
- **iokit:**
  - `iokit::iokit`
  - `"-framework IOKit"`
- ios_sim:

- ippicv:
- **irrXML:**
    - `irrXML::irrXML`
- jaegertracing:
- jansson:
- jasper:
- **jo_jpeg:**
    - `jo_jpeg::jo_jpeg`
- Jpeg:
- **jsoncpp:**
    - `jsoncpp_lib_static`
- **JsonSpirit:**
    - `json`
- kbproto:
- **kNet:**
    - `boo PUBLIC kNet::kNet`
- **LAPACK:**
    - `blas lapack`
- lcms:
- Leathers:
- Leptonica:
- **leveldb:**
    - `leveldb::leveldb`
- **LibCDS:**
    - `LibCDS::cds) # Use cds-s for static library`
- **libcpuid:**
    - `boo PUBLIC libcpuid::libcpuid`
- Libcxxabi:
- Libcxx:
- libdaemon:
- **libdill:**
    - `libdill libdill::dill`
- **Libevent:**
    - `Libevent::event_core`
- libevhtp:
- **libffi:**

  - `PkgConfig::libffi`
- libjson-rpc-cpp:
- **libmill:**
  - `libmill libmill::mill_s`
- libogg:
- **libpcre:**
  - `PkgConfig::libpcre`
- librtmp:
- libscrypt:
- **libsodium:**
  - `libsodium::libsodium`
- Libssh2:
- libunibreak:
- **libuv:**
  - `libuv::uv`
- libxml2:
- **libyuv:**
  - `PUBLIC libyuv::yuv`
- LLVMCompilerRT:
- LLVM:
- **lmdb:**
  - `lmdb liblmdb::lmdb`
- **lmdbxx:**
  - `lmdbxx::lmdbxx`
- **log4cplus:**
  - `log4cplus::log4cplus`
- Lua:
- **lz4:**
  - `boo PUBLIC lz4::lz4`
- **lzma:**
  - `lzma::lzma`
- **md5:**
  - `boo PUBLIC md5::md5`
- **metal:**
  - `metal::metal`
  - `"-framework Metal"`

Microsoft.* GSL: * mini_chromium: * minizip:

- `minizip::minizip`
- mng:
- **mobilecoreservices:**
    - `mobilecoreservices::mobilecoreservices`
    - `"-framework MobileCoreServices"`
- **mojoshader:**
    - `boo PUBLIC mojoshader::mojoshader`
- **mongoose:**
    - `mongoose mongoose::mongoose`
- mpark_variant:
- **msgpack:**
    - `msgpack::msgpack`
- mtplz:
- **MySQL-client:**
    - `"MySQL::libmysql"`
    - `"MySQL::client"`
- nanoflann:
- NASM:
- ncnn:
- **nlohmann_json:**
    - `nlohmann_json::+nlohmann_json`
- **nsync:**
    - `nsync::nsync`
- odb-boost:
- odb-compiler:
- odb-mysql:
- **odb-pgsql:**
    - `odb::pgsql`
- odb:
- odb-sqlite:
- **ogles_gpgpu:**
    - `ogles_gpgpu::ogles_gpgpu`
- oniguruma:
- onmt:
- **OpenAL:**

  - `OpenAL::+OpenAL`
- **OpenBLAS:**
  - `OpenBLAS::OpenBLAS`
- **OpenCL-cpp:**
  - `PRIVATE OpenCL-cpp::OpenCL-cpp`
- **OpenCL:**
  - `PRIVATE OpenCL::OpenCL`
- OpenCV-Extra:
- **OpenCV:**
  - `PRIVATE ${OpenCV_LIBS}`
- **openddlparser:**
  - `openddlparser::openddl_parser`
- **opengles:**
  - `opengles::opengles`
  - `"-framework OpenGLES"`
- OpenNMTTokenizer:
- **OpenSSL:**
  - `OpenSSL::+SSL OpenSSL::+Crypto`
- **opentracing-cpp:**
  - `OpenTracing::opentracing`
  - `OpenTracing::opentracing-static`
- **osmesa:**
  - `osmesa::osmesa`
- pcg:
- pciaccess:
- PhysUnits:
- PNG:
- **PocoCpp:**
  - `Poco::Foundation`
- **poly2tri:**
  - `poly2tri::poly2tri`
- **polyclipping:**
  - `polyclipping::polyclipping`
- **PostgreSQL:**
  - `PostgreSQL::libpq`
- presentproto:

- PROJ4:
- protobuf-c:
- **Protobuf:**
  - `protobuf::libprotobuf`
- pthread-stubs:
- **pugixml:**
  - `boo PUBLIC pugixml`
- **pybind11:**
  - `pybind11::pybind11 pybind11::embed pybind11::module`
- QtAndroidCMake:
- QtCMakeExtra:
- QtQmlManager:
- Qt:
- **quartzcore:**
  - `quartzcore::quartzcore`
  - `"-framework QuartzCore"`
- **rabbitmq-c:**
  - `rabbitmq-c::rabbitmq-static`
- rabit:
- randrproto:
- range-v3:
- **RapidJSON:**
  - `RapidJSON::rapidjson`
- **RapidXML:**
  - `RapidXML::RapidXML`
- **re2:**
  - `RE2::re2`
- **recastnavigation:**
  -
- renderproto:
- rocksdb:
- ros_comm_msgs:
- ros_common_msgs:
- ros_console_bridge:
- roscpp_core:
- ros_environment:

- ros_gencpp:
- ros_geneus:
- ros_genlisp:
- ros_genmsg:
- ros_gennodejs:
- ros_genpy:
- ros_message_generation:
- ros_message_runtime:
- rospack:
- ros:
- ros_std_msgs:
- **SDL2:**
  - `SDL2::SDL2`
- **SDL_image:**
  - `main`
- **SDL_mixer:**
  - `SDL_mixer::SDL_mixer`
- **SDL_ttf:**
  - `SDL_ttf::SDL_ttf`
- **sds:**
  - `sds::sds`
- sm:
- Snappy:
- Sober:
- sources_for_android_sdk_packer:
- Sources-for-Android-SDK:
- sparsehash:
- **spdlog:**
  - `spdlog::spdlog`
- sqlite3:
- **sse2neon:**
  - `sse2neon::sse2neon`
- **stanhull:**
  - `boo PUBLIC stanhull::stanhull`
- **state_machine:**
  - ‘‘ sm state_machine‘‘

- **stb:**
  - `boo PUBLIC stb::stb`
- stdext-path:
- **stormlib:**
  - `stormlib::stormlib`
- sugar:
- **SuiteSparse:**
  - `SuiteSparse::cholmod`
- **szip:**
  - `szip::szip`
- **tacopie:**
  - `tacopie::tacopie`
- tclap:
- tcl:
- Tesseract:
- **thread-pool-cpp:**
  - `thread-pool-cpp::thread-pool-cpp`
- **thrift:**
  - `PUBLIC`
- **TIFF:**
  - `TIFF::libtiff`
- **tinydir:**
  - `tinydir::tinydir`
- tinyxml2:
- toluapp:
- tomcrypt:
- tommath:
- **type_safe:**
  - `type_safe_example type_safe`
- **uikit:**
  - `uikit::uikit`
  - `"-framework UIKit"`
- **Urho3D:**
  - `boo PUBLIC Urho3D::Urho3D`
- **util_linux:**
  -

- vorbis:
- VulkanMemoryAllocator:
- Washer:
- **WDC:**
  - `WDC::libwdc`
- WebKit:
- WebP:
- **websocketpp:**
  - `websocketpp::websocketpp`
- WinSparkle:
- **WTL:**
  - `WTL::WTL`
- **wxWidgets:**
  - `${wxWidgets_LIBRARIES}`
- x11:
- x264:
- xau:
- xcb-proto:
- xcb:
- xcursor:
- xdamage:
- xextproto:
- xext:
- xf86vidmodeproto:
- xfixes:
- **xgboost:**
  - `xgboost::xgboost`
- xineramaproto:
- xinerama:
- xi:
- xorg-macros:
- xproto:
- xrandr:
- xrender:
- xshmfence:
- xtrans:

- xxf86vm:
- **yaml-cpp:**
  - `yaml-cpp::yaml-cpp`
- **ZeroMQ:**
  - ``ZeroMQ::libzmq) ``
- **ZLIB:**
  - `ZLIB::zlib`
- **ZMQPP:**
  - `ZMQPP::zmqpp`
- **zookeeper:**
  - `zookeeper::zookeeper_mt`
  - `# zookeeper::zookeeper_st) # if you want the single-threaded lib instead`

4.5 Upgrade your users

Test it live: https://github.com/nxxm/example-upgrd-app

4.5.1 Installing upgrd

nxxm keeps your deployments up-to-date for you, add upgrd to *.nxxm/deps*:

```
{
    "nxxm/upgrd" : { "@" : "v0.0.3" }
}
```

Add to your app main function the following:

```
#include <upgrd/upgrd.hxx>

int main(int argc, char** argv) {

  // Download Releases out of GitHub Release Page Assets
  upgrd::manager up{
    "github-account",
    "your-github-repo",
    "v0.0.1",
    argc,
    argv,
    std::cout
  };
  up.propose_upgrade_when_needed();

  return 0;
}
```

Relies on GitHub Releases to distribute always the newest version to your users.

4.5.2 Publishing Releases

Look at an example: https://github.com/nxxm/example-upgrd-app/releases

- Add a zip file with your binary to a GitHub Release. Simply add in the zip name respective to each platform : * *windows* * *linux* * *macOS*
- Add a SHA1 sum in the body of the release for each archive : *archive-name.zip:SHA1XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*

4.6 Passing -D defines

Pass them simply via `nxxm . -DSOME_OPTION=1 -DOTHER_OPTION=OK`.

Hint: The defines will transitively be passed to all your dependencies build as well.

4.7 Compile options

We rely on the CMake project. nxxm abstracts it away, but you might need to tweak the compilation flags.

We encourage you to put all your compile options within your target toolchain files. We deliver many target toolchain files pre-configured. You can also add your own in `.nxxxm/<distro>/polly/`.

Nevertheless if you add compile options, they will be used for all projects in the build tree as we add them in the context of the current project to the sysroot toolchain file.

4.7.1 .nxxm/opts[.target-platform]

If the file is named `.nxxm/opts` it is always used, to that if their are `.nxxm.target-platform` file the options are used only in the case the platfom is selected via `nxxm . -t target-platform` it is cumulative to the the main opts.

The opts file have to contain valid CMake Syntax. For example to pass #defines or compile options this way simply add:

```
add_compile_options ( -fmath-errno -Wextra )
add_compile_definitions( DEFINE_TO_PASS_WITHOUT_D_BEFORE=1 )
```

4.8 GitHub.com & Github Enterprise Authentication

4.8.1 Create a .nxxm/.auth file

Hint: On Windows it's in *C:\.nxxm\.auth*

Hint: On other platforms in *${HOME}/.nxxm/.auth*

If you have modified the environment variable ($env:NXXM_HOME_DIR), the authentication token must be in the *$NXXM_HOME_DIR/.nxxm/.auth* .

The file is a JSON Array of credentials to setup on one entry GitHub.com credentials and on mutliple for Github Enterprise :

```
[
  {
    "auth_info" : {
      "user" : "<GitHub.com username>",
      "pass" : "<GitHub.com password or Personal Access Token (if you have 2FA on)>"
    }
  },
  {
    "endpoint" : "https://github.your-enterprise.com",

    "auth_info" : {
      "user" : "<username>",
      "pass" : "<GitHub.com password or Personal Access Token (if you have 2FA on)>"
    }
  }

]
```

To create a personal access token, please refer to the ‘GitHub documentation <https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/> ‘_ .

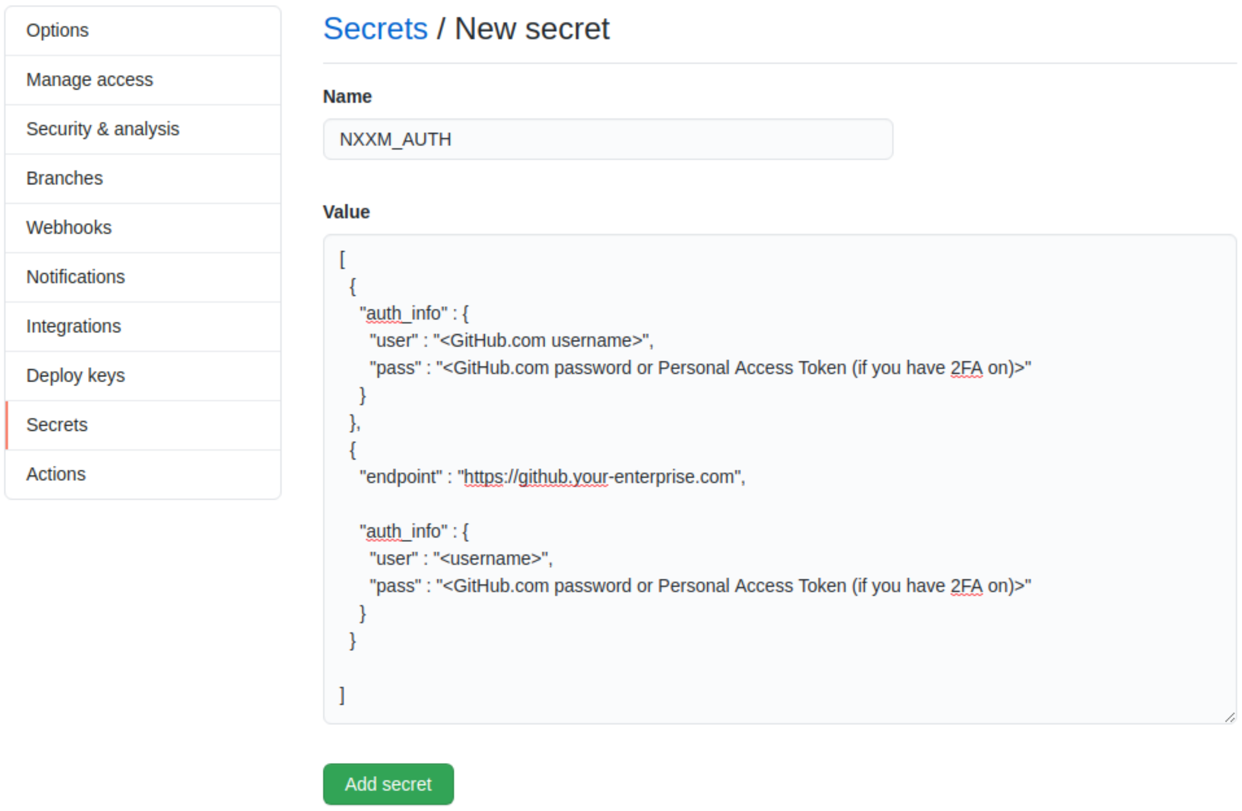
4.9 Continous integration with GitHub actions

With `nxxm ci` you can generate a workflow that will build and test your project in github actions.

4.9.1 Create a Github secret

In order for nxxm to be able to pull from your private Github repositories, it needs the credentials you created for authentication.

After creating it you can copy its content and paste it in a Github secret with the name of NXXM_AUTH.



NXXM_AUTH

```
[
 {
   "auth_info" : {
     "user" : "<GitHub.com username>",
     "pass" : "<GitHub.com password or Personal Access Token (if you have 2FA on)>"
   }
 },
 {
   "endpoint" : "https://github.your-enterprise.com",

   "auth_info" : {
     "user" : "<username>",
     "pass" : "<GitHub.com password or Personal Access Token (if you have 2FA on)>"
   }
 }

]
```

4.10 Environment variables

4.10.1 NXXM_HOME_DIR (dependencies & tools cache)

When using nxxm for the first time, the software will create a new directory *.nxxm*

- On Windows it's in *C:\.nxxm*
- On other platforms in *${HOME}/.nxxm/*

In this directory nxxm will install dependencies, toolchain files and tools for your environments.

But you may not have permission to write to this part of the disk. Or that you run nxxm from another disk.

To solve this problem you can tell nxxm to install it's tools and dependencies at the location pointed by : *NXXM_HOME_DIR*.

For example on windows powershell you can specify *$env:NXXM_HOME_DIR = "D:\a\.nxxm"*

4.10.2 NXXM_DISTRO_JSON

nxxm uses a json file which contains the required tools used by nxxm to build projects, as cmake or make. These tools are automatically downloaded and installed by nxxm at runtime before running projects build.

To allow nxxm usage flexibility, nxxm is ready to use the environment variable *NXXM_DISTRO_JSON* which may point to an absolute or relative file path, or even an HTTP(s) url which point to your own distribution json file.

The original json file can be found at https://github.com/nxxm/distro/blob/master/v0.0.12/distro.json.

Below some examples of what you can set as *NXXM_DISTRO_JSON*:

- NXXM_DISTRO_JSON = "~/projects/nxxm/distro.json"
- NXXM_DISTRO_JSON = "/home/user/projects/nxxm/distro.json"
- NXXM_DISTRO_JSON = "https://company.com/nxxm/distro.json"

If *NXXM_DISTRO_JSON* contains an absolute or relative file path, it will be used directly by nxxm.

If *NXXM_DISTRO_JSON* contains an HTTP(s) url, nxxm will handle the file download and the environment variable *NXXM_DISTRO_JSON_SHA1* has to be defined too (see *NXXM_DISTRO_JSON_SHA1*).

4.10.3 NXXM_DISTRO_JSON_SHA1

As nxxm handles the environment variable *NXXM_DISTRO_JSON*, if it contains an HTTP(s) url, nxxm will check the SHA1 got from the environment variable *NXXM_DISTRO_JSON_SHA1* to know if it needs to download and override the existing json file.

For example:

- NXXM_DISTRO_JSON = "https://company/nxxm/distro.json"
- NXXM_DISTRO_JSON_SHA1 = "4eb777d088ea949709e9ea97bbc8c389a63856e2"