Mesonbuild

A fundamental introduction to one of the simplest C++ build systems.

Alex Schwartz

Credentials

- **Worked with Mesonbuild out of college for 3.5 years**
  - Never before worked with CMake
- **Project used Meson for building/assembling all sw**
  - Web, C++, Golang, etc
  - Meson: External commands
- **Enjoyed my C++ life**

Why Mesonbuild?

- **Can Meson do better?**
  - Must be simple to use

Why Mesonbuild?

- **Can Meson do better?**
  - Must do the right thing by default
    - “Most builds are done by developers working on the code. Therefore the defaults must be tailored towards that use case. As an example the system shall build objects without optimization and with debug information.”

Why Mesonbuild?

- **Can Meson do better?**
  - Must enforce established best practices
    - “**There really is no reason to compile source code without the equivalent of -Wall**. **So enable it by default**. A different kind of best practice is the total separation of source and build directories. **All build artifacts must be stored in the build directory.** Writing stray files in the source directory is not permitted under any circumstances.”
    - Meson: warning_level

Why Mesonbuild?

- **Can Meson do better?**
  - Must have native support for platforms that are in common use
    - “A lot of free software projects can be used on non-free platforms such as Windows or OSX. The system must provide **native support for the tools of choice on those platforms**. In practice this means native support for Visual Studio and XCode. **Having said IDEs invoke external builder binaries does not count as native support**.”

Why Mesonbuild?

- **Can Meson do better?**
  - Must be fast
    - “Running the configuration step on a moderate sized project must not take more than five seconds. Running the compile command on a fully up to date tree of 1000 source files must not take more than 0.1 seconds.”

Why Mesonbuild?

- **Can Meson do better?**
  - Must provide easy to use support for modern sw development features
    - “An example is **precompiled headers**. Currently no free software build system provides native support for them. Other examples could include easy **integration of Valgrind and unit tests, test coverage reporting** and so on.”

Why Mesonbuild?

- **Can Meson do better?**
  - Must allow override of default values
    - “Sometimes you just have to compile files with only given compiler flags and no others, or install files in weird places. The system must allow the user to do this if he really wants to.”

Why Mesonbuild?

- **Meson is opinionated**
  - Q: “Why doesn't meson have user defined functions/macros?” (5)
  - A: “The tl;dr answer to this is that **meson's design is focused on solving specific problems** rather than providing a general purpose language to write complex code solutions in build files. **Build systems should be quick to write and quick to understand,** functions muddle this simplicity.”

Why Mesonbuild?

- **Simpler to read/write than CMake**
  - “I’ve been working with CMake for 20 years and I still get struggle to work with it.” ~ David, coworker

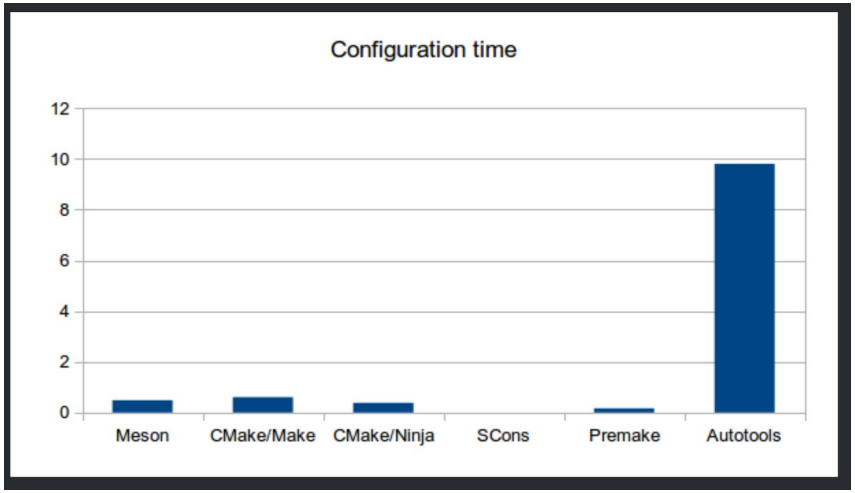
.

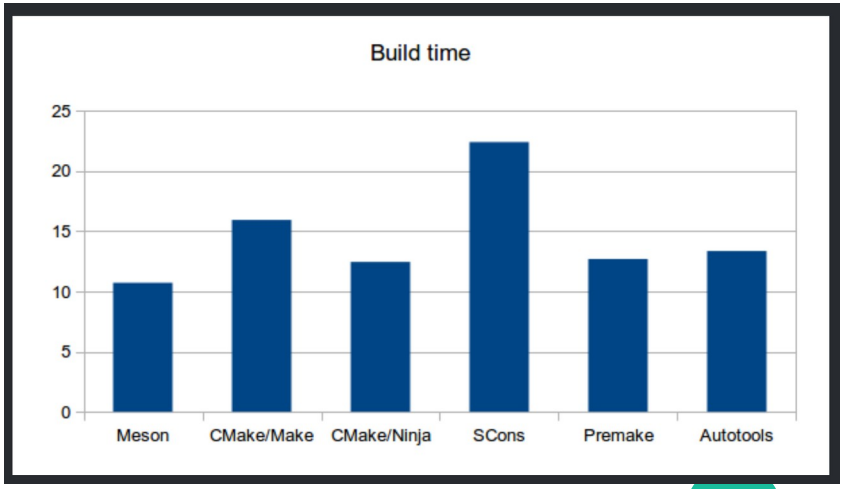
Why Mesonbuild?

- **Simpler to read/write than CMake**
  - “There are a lot of technical reasons to argue for or against using any build system. However, why **I'm convinced Meson is a great choice for both SciPy and other projects** is: **Meson is a pleasure to use**. It "feels right". That's a rare thing to say about a build system - I've certainly not heard anyone say this about numpy.distutils or setuptools. Nor about CMake. ... **Meson is** even better than Waf and is **well-maintained**, so we can finally have a single nice build system.” ~ **Ralf Gommers, director of Quantsight Labs** (4)

Why Mesonbuild?

- **User friendly and fast** (1)
  - Written in Python (2)
- **At least as fast as CMake/Ninja** (3)





Going from 0 to a C++ project

- **Bill of materials**
  - A build container (link)
    - Meson (PyPi)
    - Ninja (PyPi)
    - Containers are not the point of the talk
  - A repo (link)
    - Top level meson.build

Top level Mesonbuild

- **Project declaration (link)**
- **Add global / project arguments (link)**
- **Declare dependencies (link)**
  - OS system dependencies (i.e. gl, threads, built-from-source sw)
  - Subprojects (more on that later)
- **Declare subdirectories (link)**

Subprojects

- **“subproject” folder needed**
- **Create a .wrap file** (4)
  - Catalog of packages
- **Add content to (usually) top level meson.build**

Subprojects

meson.build | nlohmann_json.wrap

build > subprojects > nlohmann_json.wrap

```
[wrap-file]
directory = nlohmann_json-3.11.3
lead_directory_missing = true
source_url = https://github.com/nlohmann/json/releases/download/v3.11.3/include.zip
source_filename = nlohmann_json-3.11.3.zip
source_hash = a22461d13119ac5c78f205d3df1db13403e58ce1bb1794edc9313677313f4a9d
source_fallback_url = https://github.com/mesonbuild/wrapdb/releases/download/nlohmann_json_3.11.3-1/nlohmann_json-3.11.3.zip
wrapdb_version = 3.11.3-1

[provide]
nlohmann_json = nlohmann_json_dep
```

```
# nlohmann_json
nlohmann_json_proj:subproject = subproject(
    subproject_name:'nlohmann_json',
    default_options: [
        'default_library=static',
        'warning_level=0',
        'werror=false',
    ],
)
nlohmann_json_dep:any|dep|dict(any)|list(any) = nlohmann_json_proj.get_variable(var_name:'nlohmann_json_dep')
dep_list:list(any|dep|dict(...)) += nlohmann_json_dep
```

Currently Meson has four kinds of wraps:

- wrap-file
- wrap-git
- wrap-hg
- wrap-svn

Subprojects

- **Meson supports mixing build systems** (5)
  - “The tl/dr version is that while we do provide some functionality for this use case, **it only works for simple cases**. Anything more complex cannot be made reliable and trying to do that would burden Meson developers with an effectively infinite maintenance burden.”

Subprojects

- **Example: microbench**

build > subprojects > nanobench.wrap

```
[wrap-git]
url = https://github.com/martinus/nanobench.git
revision = head
depth = 1
```

```
# nanobench

cmake:cmake_module = import(module_name:'cmake')
cmake_opts:cmake_subprojectoptions = cmake.subproject_options()

if meson.current_build_dir().contains(fragment:'debug')
  cmake_opts.add_cmake_defines(defines:{
                               'CMAKE_POSITION_INDEPENDENT_CODE': true,
                               'CMAKE_BUILD_TYPE': 'Debug'})
else
  cmake_opts.add_cmake_defines(defines:{
                               'CMAKE_POSITION_INDEPENDENT_CODE': true,
                               'CMAKE_BUILD_TYPE': 'Release'})
endif

cmake_opts.append_compile_args(language:'cpp', arg…:'-w') # silence warnings
nanobench_proj:cmake_subproject = cmake.subproject(subproject_name:'nanobench', options: cmake_opts)
nanobench_dep:dep = nanobench_proj.dependency(tgt:'nanobench')
```

Really, really fast builds

- **Example: Muon, Meson's C99 implementation**
  - Repo (link)
  - A 100% compatible Meson project
  - "muon setup …"
  - "muon samu …" (Samurai is Ninja compatible)
- **Use case: use Muon to prebuild C++ binaries**
  - Consume prebuilt binaries as a wrap

Future research

- Meson cross and native files (“machine files”)
- Precompiled headers
- Reproducible builds
- Unity builds


Questions?