Buildyard and CMake

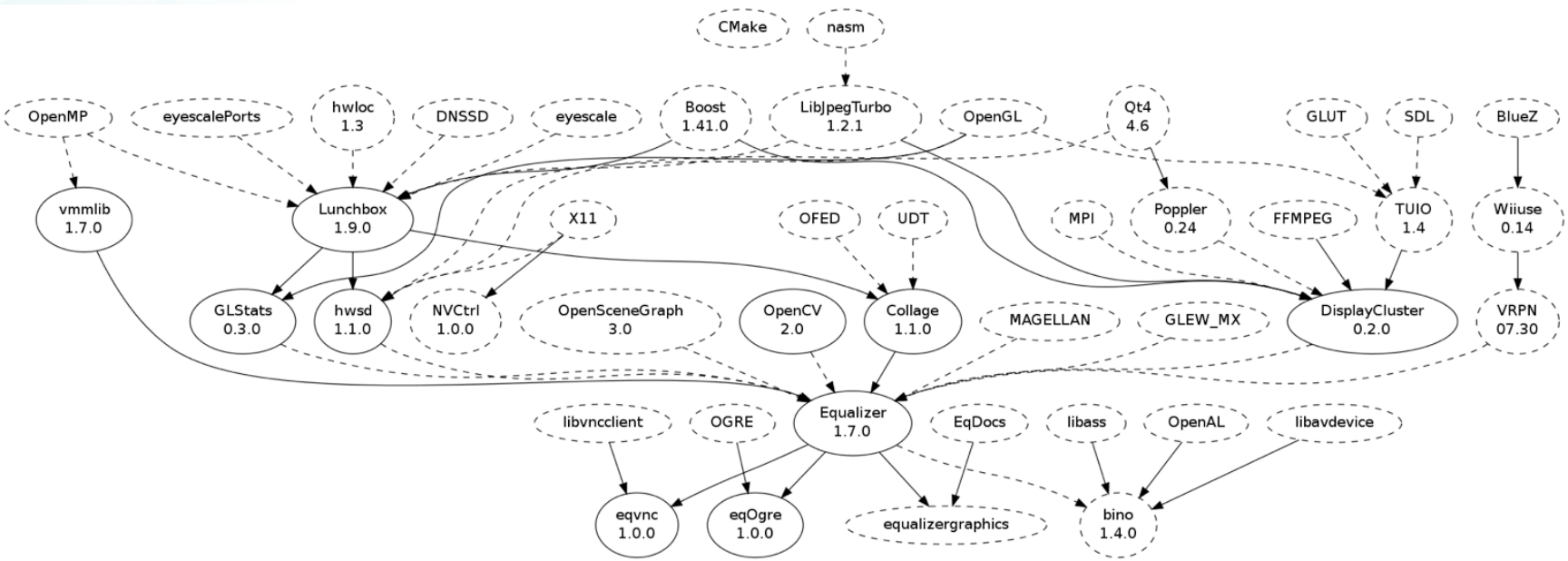
BBP Standard for C++ Development

What is Buildyard?

- CMake-based build environment
- Facilitates build of multiple projects with dependencies
- Uses installed packages, svn or git source repositories
- Extensible through modular configurations

Why?

- Build setup of modular software is painful
- Automate!
- Solved for packages, but not for development



How?

- Get Buildyard:

```
> git clone https://github.com/Eyescale/Buildyard.git
```

- Get a configuration folder:

```
> cd Buildyard
> git clone https://github.com/BlueBrain/config.git config.bluebrain
```

- Configure and install system packages:

```
> make apt-get       # Ubuntu
> make port-get      # Mac OS X, uses MacPorts
```

- Configure and build a project:

```
> make dash -j 9
```

- Work on a project:

```
> cd src/dash; vi …;  make -j 9
```

Give me more!

- Update Buildyard and configurations:

```
> make update
```

- Show the results of the last configuration:

```
> make info
```

- Reuse dependencies for project:

```
include(FindPackages) in src/Project/CMakeLists.txt
```

  - CMake/FindPackages.cmake is BY-generated

- Use an autoconf-based project:

```
set(LIBJPEGTURBO_AUTOCONF ON) in config/LibJpegTurbo.cmake
```

Give me more!

- Use a github user fork:

  `set(DASH_USER_URL https://github.com/eile/dash.git) in config.local/forks.cmake`

  – remote ”origin” points to eile, “root” to original

File System Layout

- Build/, Release/, Coverage/: Build directories where generated files end up
- Build/[Project]: Per-project build directory
- Build/install: Installed project artefacts
- src/: All project sources
- src/[Project]: Per-project source directory

Show me the Magic!

- Uses standard ExternalProject.cmake
  - Chains projects together
  - Chains download->configure->build->install for each project
  - Make <project> takes long
    - Used only to bootstrap
    - “make” in src/project does only build project
- Configured using config.<org> folders

The Magic: config Folders

- One config folder:

```
> ls config.bluebrain/
dash.cmake codash.cmake  depends.txt  Livre.cmake  README.md
```

- Depends.txt declares dependent configs:

```
config.eyescale https://github.com/Eyescale/config.git master
```

  - Buildyard clones and parses these recursively

- Per-project configuration, e.g., dash:

```
set(DASH_PACKAGE_VERSION 1.1.0)
set(DASH_REPO_URL https://github.com/BlueBrain/dash.git)
set(DASH_DEPENDS bluebrain REQUIRED Lunchbox Boost)
set(DASH_BOOST_COMPONENTS serialization)
set(DASH_DEB_DEPENDS libboost-serialization-dev)
```

The Magic: project configs

- PACKAGE_VERSION: minimum needed
- REPO_URL: Source repository
- REPO_TAG: repo revision, default master
- DEPENDS: Dependencies
  - Can be system packages
  - Source is used as fallback, if configured
  - Missing REQUIRED dependencies will cause project to not be configured

The Magic: project configs

- BOOST_COMPONENTS: optional components for a dependency
  - Used for finding dependency
  - Forwarded to project source
- DEB_DEPENDS: used for apt-get target
  - Used to configure Travis CI
- PORT_DEPENDS: used for port-get target

CMake

- Consistent project setup
- git repository included as subdirectory
  - Uses GitExternal.cmake from CMake
  - Simple .gitexternals file
- Common.cmake does most
  - Settings: system, compiler, git, cmake, ...
  - Targets: git, GNU modules, …
  - Functions: common_library, common_application, update_file, ...

CMake

- CommonCTest.cmake
  - Unit tests (ctest)
  - Code coverage (lcov)
  - Static analysis (cppcheck)
- DoyxgenRule.cmake
  - Build, install, run doxygen
  - Copy to common documentation repository
    - Published on bluebrain.github.io

CMake

- See Readme for up-to-date documentation
- Hello.git to get started
  - Uses CMake git external
  - Documents common practice for C++
  - http://bluebrain.github.io/Hello-1.0/index.html