Pragmatic CMake

BRET BROWN

Pragmatic CMake

How to Avoid Headaches with Simple CMake

Bloomberg

Engineering

CppCon 2025
September 17, 2025

Bret Brown
Developer Experience

TechAtBloomberg.com

CMake Headaches!

- CMake has a lot going on
- Reference docs aren’t tutorials
- What are “the good parts”?

Pragmatic CMake

- Guidelines learned from scaling CMake
  - Engineers: thousands @ all experience levels
  - Projects: tens of thousands
- Focus: Developer experience
  - Featureful
  - Portable
  - Teachable
  - Interoperable

Roadmap

- Resources
- Commanding CMake
- Principles
- `CMakeLists.txt` Walkthrough

Watch for ✨

These are New or Underused Features!!

TechAtBloomberg.com

Bloomberg

Engineering

Watch for 🔍

These are all snippets from beman.exemplar, line numbers included.

Paths are relative to the root of the repo.

🔍 bemanproject/exemplar `./README.md`

```
10 `beman.exemplar` is a minimal C++ library conforming to [The Beman
Standard](https://github.com/bemanproject/beman/blob/main/docs/BEMAN_STANDARD.md).
11 This can be used as a template for those intending to write Beman libraries.
12 It may also find use as a minimal and modern  C++ project structure.
```

Full reference: beman.exemplar `2.2.1` *release @* `69b712dc0024ff563673696ac41f857707a33b5a`

The Beman Project

- “Tomorrow’s C++ Standard Libraries Today”
  - Website: https://bemanproject.org
  - Discourse: https://discourse.bemanproject.org
  - GitHub: https://github.com/bemanproject/beman
- Collection of engineers and libraries
- For library developers
  - Provide a clear path toward Standardization
- For the C++ community
  - Foster usage by shipping easy-to-use libraries

TechAtBloomberg.com

Bloomberg

Engineering

`beman.exemplar`

- https://github.com/bemanproject/exemplar

- `beman.exemplar` is a C++ library
  - See also: `std::identity` from `<functional>`
  - Conforms to The Beman Standard

- An example of a minimal and modern project structure
- Literally a project (cookiecutter) template
- Industry standard tools and technology

beman.exemplar repo

Resources

TechAtBloomberg.com

Bloomberg

Engineering

Recommended!

- CMake Discourse
  - https://discourse.cmake.org
  - Sponsor: Kitware
- *Professional CMake*
  - https://crascit.com/professional-cmake
  - Author: Craig Scott (crascit)
- #cmake on CppLang Slack
  - https://cpplang.slack.com
  - Sponsor: C++ Alliance
- Official CMake repository issues
  - https://gitlab.kitware.com/cmake/cmake/-/issues
  - Sponsor: Kitware

Gist with Links for Reference

Not Recommended (for CMake tips)

- [cmake] on Stack Overflow
  - https://stackoverflow.com/questions/tagged/cmake
  - Unfortunately, quite out of date!
- LLMs I have tried so far
  - Lack of new guides to train against?
  - Caveat: LLMs get smarter constantly

Commanding CMake

TechAtBloomberg.com

Bloomberg

Engineering

CMake Workflow Overview

- Provision
- Configure / Resolve
- Build / Compile
- Test
- Install / Package

Commanding CMake: Provisioning

TechAtBloomberg.com

Bloomberg

Engineering

Provisioning Your Environment

Off-machine I/O:

- Prepare development environment
- Select production environment
- Your source code + dependencies

Driven by requirements of ultimate executable(s).

Recommended: Select a package manager

Often: Managing Deps Isn’t Bad

Examples:

- Use a known-good container image
- Install `libgtest-dev` in your Ubuntu environment
- macOS, plus `conan install binary_dir`

Commanding CMake:
Configuring

TechAtBloomberg.com

Bloomberg

Engineering

Configure a Binary Directory

Build Directory → Same thing

```
cmake -B binary_dir -S source_dir
```

- ✨Makes a directory
- ✨No need to change directory

Configure: Extra Options

```
cmake \
    -B binary_dir -S source_dir \
    -DBUILD_TESTING=ON
    --fresh
```

- [Optional] Provide extra options
  - See docs or `cmake --help`
  - `-D` options common for key/value options
  - ✨`--fresh`: Reconfigure without deleting `binary_dir`

Configure: Generator

The underlying implementation of CMake

```
cmake \
    -B binary_dir -S source_dir \
    -G Ninja
```

- Default: Often `Unix Makefiles`
- Prefer: `Ninja`, `Xcode`, `Visual Studio <version>`
- ✨Set `CMAKE_GENERATOR` as an environment variable
- See also: `Ninja Multi-Config` generator

Configure: Build Type

Configuration Type → Same Thing

```
cmake \
    -B binary_dir -S source_dir \
    -DCMAKE_BUILD_TYPE=Debug
```

- Default: Compiler defaults
- CMake-provided: `Debug`, `Release`, `RelWithDebInfo`, `MinSizeRel`
- ✨Set `CMAKE_BUILD_TYPE` as an environment variable
- See also: `CMAKE_CONFIGURATION_TYPES` for Ninja Multi-Config

Configure: Toolchain Files

Toolchain: Suite of tools that builds your program; includes a C++ compiler

CMake Toolchain file: Describes the toolchain selected for your build

```
cmake \
    -B binary_dir -S source_dir \
    -DCMAKE_TOOLCHAIN_FILE=some/path/some_file.cmake
```

- Used by: Conan, vcpkg, etc.
- Ideal place for all ABI-affecting settings.
- ✨Set `CMAKE_TOOLCHAIN_FILE` as an environment variable

Configure: Extra Flags

To fiddle with flags specifically:

```
cmake \
    -B binary_dir -S source_dir \
    -DCMAKE_CXX_FLAGS=”-fdiagnostics-add-output=sarif”
```

Prefer more specific mechanisms and injected settings when available.

Commanding CMake:
Building

TechAtBloomberg.com

Bloomberg

Engineering

Build Your Project

```
cmake --build binary_dir \
    [--target all]
```

- Builds a target
- `all` is the default target
- See also: `EXCLUDE_FROM_ALL` target property
  - For instance: integration tests or microbenchmarks

Build: Specific Targets

To select a specific build target:

```
cmake --build binary_dir \
    --target test
```

Built-in targets: `all`, `clean`, `test`, `install`

✨Now: `codegen` target

Build: See Your Commands

See all of your compile commands as you build:

```
cmake --build binary_dir \
    --verbose
```

See also: `-DCMAKE_VERBOSE_MAKEFILE=ON`

Commanding CMake:
Testing

TechAtBloomberg.com

Bloomberg

Engineering

Test Your Project

```
ctest --test-dir binary_dir
```

Or

```
cmake --build binary_dir --target test
```

- Runs registered tests
- See docs for features like:
  verbosity, filtering, parallelism, test dependencies

Test: Build Your Tests

By default, running tests ***does not*** build tests:

```
Could not find executable
beman.exemplar.tests.identity_NOT_BUILT
```

Build your tests first!

Commanding CMake:
Shipping

TechAtBloomberg.com

Bloomberg

Engineering

Ship Your Project

CMake provides a built-in `install` target

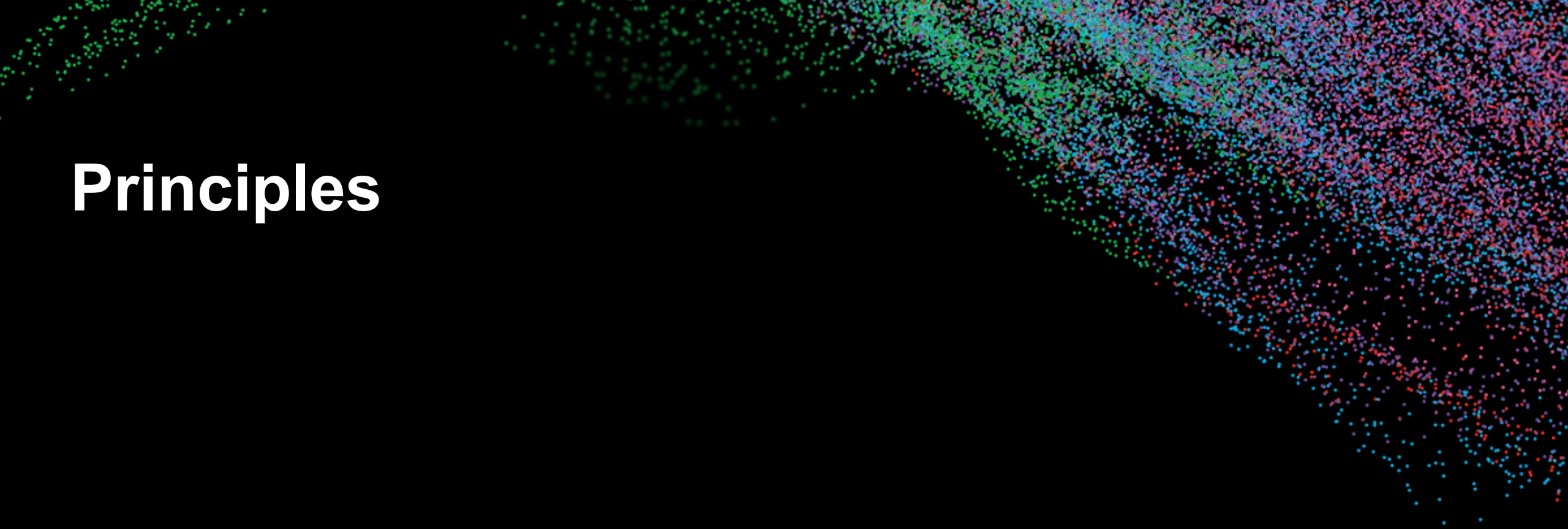
Use it for packaging

Install Your Build

Build first, then:

```
DESTDIR=put/it/here \
  cmake --install binary_dir \
  --prefix /opt/beman
```

Principles



TechAtBloomberg.com

Bloomberg

Engineering

Write Less CMake

- Look at the slide number
- How much CMake code have we looked at?
- That is not an accident

Write Less CMake: Keep it Simple

Discouraged in `CMakeLists.txt`

- ⚠️ Interop logic
- ⚠️ Algorithms
- ⚠️ Loops

Minimize these too:

- ⚠️ Variables
- ⚠️ Conditionals

Instead:

- 🧠 Describe your project declaratively

Write Less CMake: Day-to-Day CMake Editing

Mostly listing activities. Add or remove:

- Library dependencies
- Source and header files
- Tests

Note: `CMakeLists.txt`

Don’t Break `cmake` Workflows

Consider it a bug when:

- Breaking the workflows described here or in CMake docs
- Setting `CMAKE_*` variable in `CMakeLists.txt`
  - `CMAKE_BUILD_TYPE`
  - `CMAKE_CXX_FLAGS`
  - `CMAKE_CXX_STANDARD`

Don’t Break `cmake` Workflows: CTest

- Make sure the `test` target always passes
- Register zero tests if you must
  - Passing 0/0 tests is useful information!
- Add options to enable extra tests

🔍beman.exemplar `./CMakeLists.txt`

```
21 option(
22     BEMAN_EXEMPLAR_BUILD_EXAMPLES
23     "Enable building examples[...]"
24     ${PROJECT_IS_TOP_LEVEL}
25 )
```

Sensitive Compile Settings in `CMakeLists.txt`

- The Beman Standard: [CMAKE.PASSIVE_PROJECTS]
- Including the C++ Standard
  - Projects often have ABI dependencies on the C++ standard setting
  - Examples: Abseil, Boost, Beman, tl::expected, fmt, and projects that use them
- Also
  - C++ ABI flags
  - Architecture tuning flags
  - Sizes of fundamental types

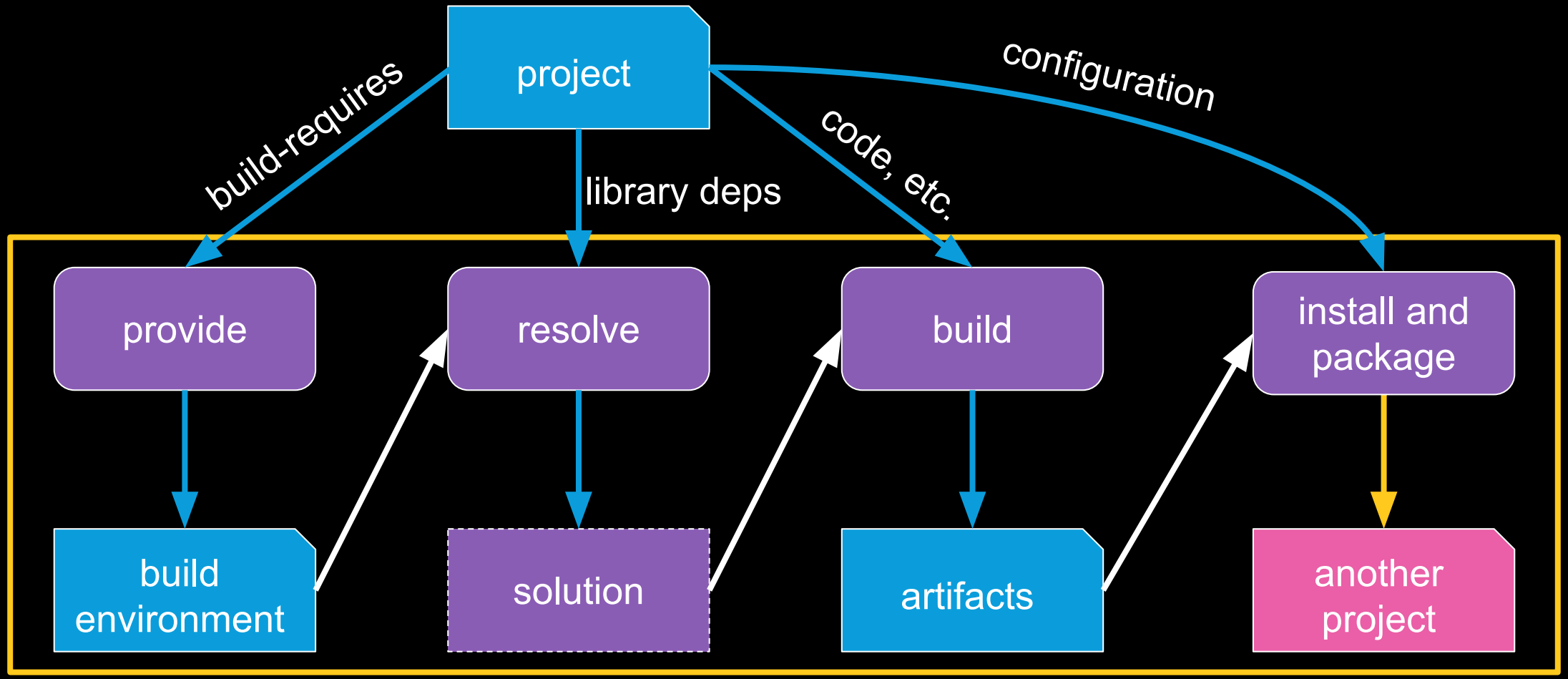
✓ Fail fast and clearly if a problem is detected

The Beman Standard

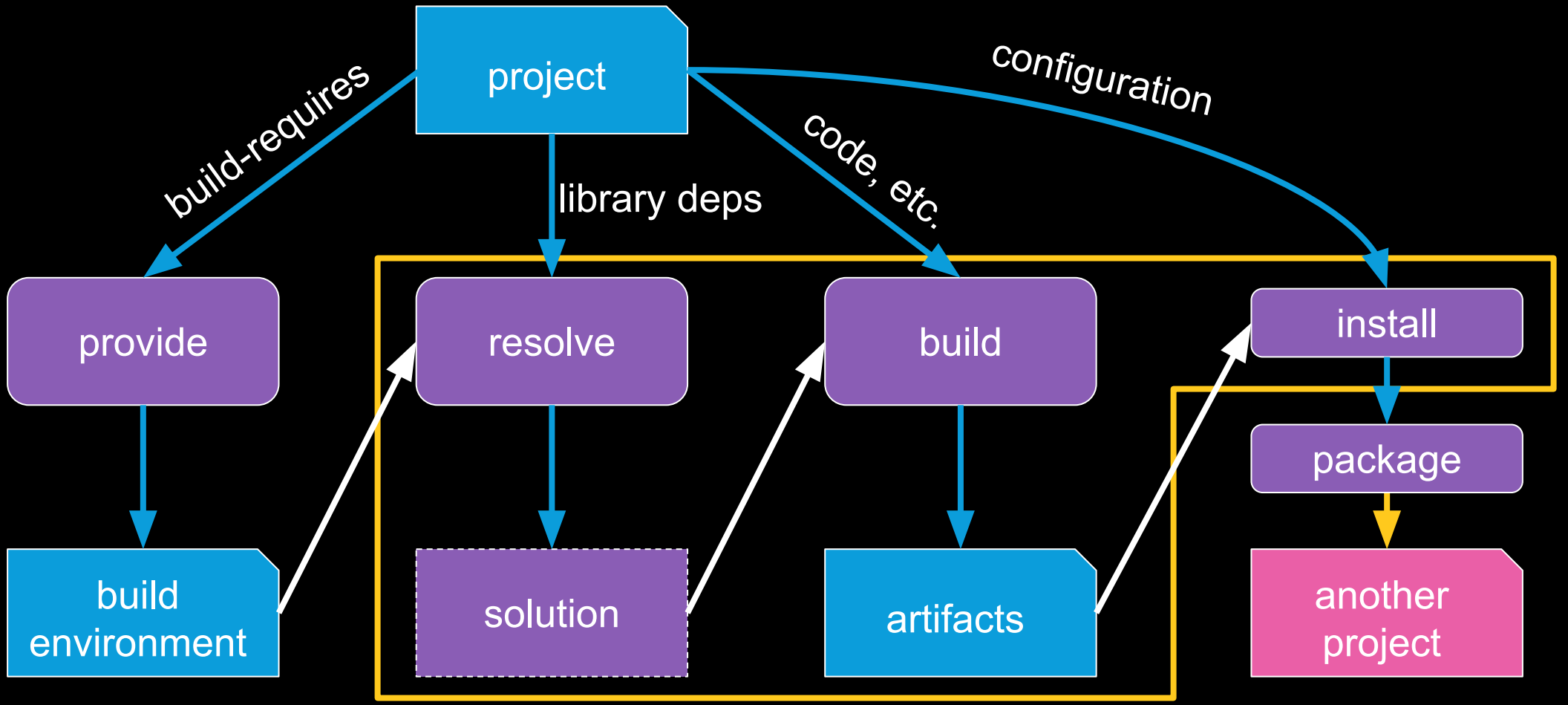
Bloomberg

Engineering

Note: You *can* make CMake Do This



Sweet Spot: CMake Is Best At This



TechAtBloomberg.com

Bloomberg

Engineering

Example: Beman Exemplar

Example: Beman Exemplar
Names

Names for Provisioning

Name	Found In	Value
git URL	Various	`github.com/bemanproject/exemplar`
vcpkg port	`vcpkg.json`	`beman-exemplar`
CMake FetchContent	`FetchContent_Declare(...)`	`beman.exemplar`

Names for Resolution

Name	Found In	Value
CMake package	`find_package(...)`	`beman.exemplar`
CMake FetchContent	`FetchContent_Declare(...)`	`beman.exemplar`
CMake target	`target_link_libraries(...)`	`beman::exemplar`
CPS component	`"requires"` entry	`beman:exemplar`

Names for Building

Name	Found In	Value
`#include` target	C++ code	`<beman/exemplar/identity.hpp>`
C++ typename	C++ code	`beman::exemplar::identity`

`CMakeLists.txt` Walkthrough

`CMakeLists.txt` Walkthrough

Top-level `CMakeLists.txt`

TechAtBloomberg.com

Bloomberg

Engineering

Essential Command: `cmake_minimum_required`

Make a `CMakeLists.txt` in the root of your repo.

🔍 bemanproject/exemplar `./CMakeLists.txt`

```
3 cmake_minimum_required(VERSION 3.25)
```

- Keep this bumped!

Essential Command: `project`

🔍bemanproject/exemplar `./CMakeLists.txt`

```
 5 project(
 6   beman.exemplar # CMake Project Name[...]
 7   # [...]
 8   DESCRIPTION "A Beman library exemplar"
 9   LANGUAGES CXX
10   VERSION 2.2.1
11 )
```

Don’t skip this statement! CMake gets very confused

Essential Command: Enable Testing

!! *Always* enable the test target in your **top-level** `CMakeLists.txt` !!

🔍bemanproject/exemplar `./CMakeLists.txt`

```
27 include(CTest)
```

Alternatively:

- `enable_testing()`

Essential Command: `add_subdirectory`

Put targets in organized subdirs
with their own `CMakeLists.txt`.

🔍bemanproject/exemplar `./CMakeLists.txt`

```
32 add_subdirectory(src/beman/exemplar)
```

Essential Command: `add_subdirectory`

Conditionally enable “expensive” or non-portable portions of your project.

🔍bemanproject/exemplar `./CMakeLists.txt`

```
35 if(BEMAN_EXEMPLAR_BUILD_EXAMPLES)
36   add_subdirectory(examples)
37 endif()
```

`CMakeLists.txt` Walkthrough
The beman.exemplar library

TechAtBloomberg.com

Bloomberg

Engineering

Define a library: `add_library`

🔍bemanproject/exemplar `./src/beman/exemplar/CMakeLists.txt`

```
3 add_library(beman.exemplar)
4 add_library(beman::exemplar ALIAS beman.exemplar)
```

`beman.exemplar` ➡ Use inside this subdirectory
`beman::exemplar` ➡ Use everywhere else

Avoid collisions! Use “globally” unique names!

Define a Library: `target_sources`

🔍bemanproject/exemplar `./src/beman/exemplar/CMakeLists.txt`

```
6 target_sources(beman.exemplar PRIVATE identity.cpp)
```

- Adds source files to your targets
- `PRIVATE` best for typical use cases
  - 💡Means: Nothing inherits the need to build this source file

Define a Library: Header File Sets

🔍bemanproject/exemplar ./src/beman/exemplar/CMakeLists.txt

```
 8 target_sources(
 9   beman.exemplar
10   PUBLIC
11     FILE_SET HEADERS
12     BASE_DIRS ${CMAKE_CURRENT_SOURCE_DIR}/../../../include
13     FILES
14 {CMAKE_CURRENT_SOURCE_DIR}/../../../include/beman/exemplar/identity.hpp
15 )
```

- ✨Let CMake know your headers belong to your library!

Define a Library: ✨Header File Sets

Header File Sets

- Trivially installed as part of your library
- Implicitly wired up as include directories
- Headers can be validated directly

Define a Library: Verify Your Headers

🔍bemanproject/exemplar `./src/beman/exemplar/CMakeLists.txt`

```
17 set_target_properties(beman.exemplar
     PROPERTIES VERIFY_INTERFACE_HEADER_SETS ON
   )
18
```

- ✨Creates an `all_verify_interface_header_sets` target
- ✨Adds an entry for `beman/exemplar/identity.hpp.cxx` to `compile_commands.json`

`CMakeLists.txt` Walkthrough
Exporting beman.exemplar

TechAtBloomberg.com

Bloomberg

Engineering

How Does Beman Install Libraries?

🔍 bemanproject/exemplar `./src/beman/exemplar/CMakeLists.txt`

```
19 find_package(BemanInstallLibrary REQUIRED)
20 beman_install_library(beman.exemplar)
```

This `find_package` defines `beman_install_library`

Bret Brown

Modern CMake Modules

Modern CMake Modules

Bloomberg

Engineering

CppCon 2021
October 27, 2021

Bret Brown
Software Engineer

TechAtBloomberg.com

What does `beman_install_library` do?

Ensures the following interfaces are provided correctly to consumers:

Name	Found In	Value
CMake package	`find_package(...)`	`beman.exemplar`
CMake target	`target_link_libraries(...)`	`beman::exemplar`
`#include` target	C++ code	`<beman/exemplar/identity.hpp>`

Export a library: Install the Library

```
install(
    TARGETS beman.exemplar
    EXPORT beman.exemplar
    FILE_SET HEADERS
)
```

- Installs built `beman.exemplar` binary file
- Installs `beman/exemplar/identity.hpp`
- ✨`FILE_SET HEADERS` lets you skip header install steps

Export a library: Simple CMake Support

```
include(GNUInstallDirs)
install(
    EXPORT beman.exemplar
    DESTINATION "${CMAKE_INSTALL_LIBDIR}/cmake"
    NAMESPACE beman::
    FILE beman.exemplar-config.cmake
    # Experimental!!!!
    EXPORT_PACKAGE_DEPENDENCIES
)
```

- Adds basic support for `find_package(beman.exemplar)`
- `beman::` matches `beman::exemplar` library alias

Export a library: Matching the Beman Aliases

- Needed to support `beman::exemplar` instead of `beman::beman.exemplar`

```
set_target_properties(
  beman.exemplar
  PROPERTIES
      EXPORT_NAME exemplar ⬅️
)
```

C++ now 2025

Post-Modern Cmake

From 3.0 to 4.0

Vito Gamberini

C++ now 2025
Aspen, Colorado

CppNow.org

Video Sponsorship Provided By

Bloomberg
Engineering

think-cell

Post-Modern Cmake
From 3.0 to 4.0

Vito Gamberini

`CMakeLists.txt` Walkthrough
Defining a (Test) Executable

Bloomberg

Engineering

Pull in a Test Dependency

🔍bemanproject/exemplar `./tests/beman/exemplar/CMakeLists.txt`

```
3 find_package(GTest REQUIRED)
```

- Pulls googletest in as an imported target: `GTest::gtest`

Make an Executable

🔍bemanproject/exemplar ./tests/beman/exemplar/CMakeLists.txt

```
 5 add_executable(beman.exemplar.tests.identity)
 6 target_sources(beman.exemplar.tests.identity
       PRIVATE identity.test.cpp)
 7 target_link_libraries(
 8     beman.exemplar.tests.identity
 9     PRIVATE beman::exemplar GTest::gtest GTest::gtest_main
10 )
```

- Declare the target with `add_executable`
- Associate source files with `target_sources`
- (As appropriate) Associate headers with header file sets
- Declare library dependencies with `target_link_libraries`

`CMakeLists.txt` Walkthrough
Defining a Test

Add a Test

🔍bemanproject/exemplar `./tests/beman/exemplar/CMakeLists.txt`

```
11 include(GoogleTest)
12 gtest_discover_tests(beman.exemplar.tests.identity)
```

- Wires up GoogleTest drivers to the `test` target
  - Remember: `include(CTest)` required to define `test`
- Makes a unique test target per `TEST()` declaration
  - Allows `ctest` and/or IDEs to filter, randomize, etc.

Add a Test (Alternative)

```
add_test(
  NAME beman.exemplar.tests.identity
  COMMAND beman.exemplar.tests.identity
)
```

- Wires up an executable to the `test` target
- Works for any subprocess call

Takeaways

- Optimize for simple, declarative `CMakeLists.txt`
  - A handful of statements per library, executable, test
  - If you wouldn’t do it across 12 projects, don’t do it once
- Use features of the `cmake` command more, CMake code less
- Defer to tools with more context
  - CMake presets
  - CI tooling
  - Even a top-level Makefile
  - Package managers

Thank you!

https://TechAtBloomberg.com/cplusplus

https://www.bloomberg.com/careers

Contact me:
mail@bretbrownjr.com
https://github.com/bretbrownjr
https://x.com/bretbrownjr
https://mastodon.social/@bretbrownjr
https://linkedin.com/in/bretbrownjr
https://reddit.com/user/bretbrownjr
Bret Brown @ https://cpplang.slack.com

Bloomberg

Engineering

TechAtBloomberg.com

✨Bonus: CPS Support

- Using these recommendations future-proofs your project
- Expect implicit support when CMake moves to JSON-based exports
- Pragmatic CMake projects should expect cleaner support
- See *CPS in CMake* @ C++Now 2025 by Bill Hoffman

✨Bonus: CPS Support

What a standard C++ library package might look like

```
[...]
├── include/beman/exemplar/identity.hpp
└── lib
    ├── libbeman.exemplar.a
    ├── cps
    │   └── beman
    │       ├── beman-exemplar.cps
    │       └── beman-exemplar@debug.cps
    └── debug
        └── libbeman.exemplar.a
```

Bonus: CPS Files

beman-exemplar.cps

```
{
  "components": {
    "exemplar":
        { "type": "archive" }
  },
  "cps_path":
"@prefix@/lib/cps/beman",
  "cps_version": "0.13.0",
  "name": "beman.exemplar",
  "version": "2.2.1"
}
```

beman-exemplar@debug.cps

```
{
  "components": {
    "exemplar": {
      "link_languages": [ "cpp" ],
      "location":
"@prefix@/lib/debug/libbeman.exemplar.a"
    }
  },
  "configuration": "Debug",
  "name": "beman"
}
```