MRB 5 and cetbuildtools 8

Chris Green, FNAL

LArSoft Coordination Meeting, 2021-09-21

Recap

Quick recap:

- Developing a long term replacement for our current UPS-based ecosystem with wide applicability across HEP.
- Spack / cetmodules / SpackDev / BuildCache vs UPS & ssibuildshims / cetbuildtools / MRB / SciSoft.
- Current issue: need to find a way to allow experiments to continue development of UPS packages while transitioning to Spack.

Solution 1: cetmodules 1

- Not capable of producing UPS packages, nor of finding UPS dependencies.
- Not backward-compatible with cetbuildtools 7: code forks necessary.
- Not compatible with MRB.

Solution 2: cetmodules 2

- Capable of reading `product_deps` files and both using and producing UPS packages.
- Leverages “modern” CMake paradigms.
- Fine as a final destination, but lots of deprecation warnings.

Solution 2a: cetbuildtools 8 / MRB 5

- cetbuildtools 8 is a wrapper around cetmodules 2 with increased compatibility and fewer deprecation warnings.
- Generally fewer changes needed.
- Can transition to cetmodules 2 and modern CMake paradigms at a more leisurely pace.
- MRB 5 capable of managing cetbuildtools 8 and cetmodules 2-based packages in the same development set.

New features of MRB 5 *vs* MRB 4

- Improved setup time.
- Improved management of in-tree dependencies and include paths.
- `buildtool -[cp]` (`mrb b -[cp]`) are fully functional in MRB 5 development environments with “Ninja” and “UNIX Makefiles” generators —`mrb z` and `mrb mp` respectively are supported, but no longer necessary.
- `mrb uc` is executed by `mrbsetenv` to refresh `CMakeLists.txt`.
- Detection of incorrect/out-of-date package subdirectory order at build time.
- Non architecture-dependent (*e.g.* cetmodules) or non-qualified packages (*e.g.* TRACE) can be developed alongside qualified packages in an MRB 5 development area.

Migrating from MRB 4 -> MRB 5

1. Make a new development area with `mrb newDev`.
2. Move/copy/link sources into `srcs/`.
3. Refresh `CMakeLists.txt` with `mrb uc`.
4. Update `cetbuildtools` version in all packages with `mrb uv`.
5. Setup up products with `mrbsetenv`.
6. Build (subject to cetbuildtools 7 -> cetbuildtools 8 migration).

New features of cetbuildtools 8 *vs* cetbuildtools 7

- More comprehensive generated CMake config files.
- Removal of reliance on UPS-set environment variables.
- (Enhanced) CMake command `find_package()` supersedes `find_ups_product()`, although the latter will still work.
- CMake targets (*e.g.* Boost::regex) preferred to environment variables.
- Link library dependence on targets confers include location, and can be used to specify transitive dependencies.

New features of cetbuildtools 8 *vs* cetbuildtools 7

- Per-package build configuration is handled with *project variables*—enhanced CMake variables with properties:
  - Can be propagated to dependencies via CMake Config files and `find_package()`.
  - Can be initialized different ways with a deterministic priority: initial value (command line/cache), declaration-time default, code-based initial value, global override, package override.
  - Once declared, every package gets its own value.
  - Can be used to represent path fragments (subdirectories), which are then expanded as needed (*e.g.* for file installation).
  - Can be visible in the cache for configuration by `ccmake`, or hidden (“advanced”).

New features of cetbuildtools 8 *vs* cetbuildtools 7

- `CMAKE_MODULE_PATH` additions now handled by `find_package()` and CMake config files.
- Automatic target generation now supported by library / plugin generation functions, *etc.*
- Vastly improved dependency management via targets and CMake-based transitivity directives (PRIVATE, INTERFACE, PUBLIC) will lead eventually to faster build times and few recompilations when something changes.
- Improved plugin building facilities to allow avoidance of One Definition Rule violations via separate implementation and (non-linkable) plugin registration libraries.

Migrating from cetbuildtools 7 -> cetbuildtools 8

- Mandatory for correct behavior:
  - `buildtool` must be used in place of raw invocation of `cmake <top-level-CMakeLists.txt-dir>` as most propagation of build configuration from `product_deps` -> CMake is handled by `buildtool`
  - `[cet_]find_library(...[PATHS|HINTS] $ENV{<name>})` must be changed to `[cet_]find_library(...[PATHS|HINTS] ENV <name>)`, along with:
  - Explicit mention of `${<LIBRARY_VAR>}` as set by `[cet_]find_library()` must be replaced by `<LIBRARY_VAR>` (*e.g.*) `${LIBTORCH}` -> `LIBTORCH` to allow for relocatable propagation of transitive dependencies.

Migrating from cetbuildtools 7 -> cetbuildtools 8

- Recommended to avoid expensive mitigations:
  - `find_package(cetbuildtools [<min-version>])` should be *before* `project()`
  - Project name and version should be specified to CMake, and version specification should be removed from `product_deps` to provide a single point of maintenance.
  - Manual manipulation of `CMAKE_MODULE_PATH` should be replaced by `find_package()` (dependencies) and `cet_cmake_module_directories()`.
  - Removal of all use of `${product}/${version}` from manual `install()` commands.

Notes and caveats

- Addition of some `find_package()` commands may be necessary to cover direct use of dependencies previously brought in by a transitive dependency no longer required.
- *“Is upgrading to cetbuildtools 8 sufficient to be able to build with Spack?”*

  No.

  “OK, then: is migrating direct to cetmodules 2 sufficient?”

  No.

  “OK, then: why bother?”

  Once a package has been migrated to use cetbuildtools 8, the changes necessary to upgrade a package from cetbuildtools 8 -> cetmodules 2 -> “Spack-ready” can be done progressively without compromising the ability to use MRB for development, to use or produce UPS packages, or to be developed with MRB alongside packages at an earlier stage of Spack readiness.