TriBITS Modernization Update

Roscoe A. Bartlett
Department 1424
Software Engineering and Research

October 27, 2022
Trilinos Users Group Meeting, Developers Day
TriBITS and CMake Background

TriBITS History
- **2007**: A partial initial CMake build system for Trilinos started by Tim Shead & Danny Dunlavy.
- **2008**: Ross takes over CMake build system and creates package-based architecture and wrappers for raw CMake.
- **2011**: TriBITS system factored out of Trilinos into independent git repo to support larger, more complex CASL VERA project.
- **2014**: Primary TriBITS development is complete and TriBITS is put out on GitHub.

CMake Developments: (Source: Professional CMake: 10th edition)
- **2014**: CMake 3.1: First usable target-centric modern CMake.
- **2016**: CMake 3.7: More realistic for using modern target-centric CMake.
- **2018** (Mar): CMake 3.11: Modern target-centric dependency management for aggregate projects well supported. (a.k.a. Modern CMake)

Trilinos CMake Minimum versions:
- **2008**: CMake 2.6
- **2011**: CMake 2.7
- **2014**: CMake 2.8.11
- **2018**: CMake 3.10
- **2021**: CMake 3.17.0
- **2022**: CMake 3.22.0?

TriBITS implemented a scalable architecture for CMake projects 6 years before than was possible with raw CMake in CMake 3.1 and 8 years before it was really well supported in CMake 3.7. But, TriBITS is now standing in the way of adopting some modern CMake features.
Modern CMake: Accelerated Adoption and Developments

• There has been significant growth in CMake adoption, maturation and feature development in recent years. (CMake is now most popular build system for C++ code in the world)

• Many features/workarounds added to TriBITS in early years have been resolved in native CMake.

• Many now-redundant TriBITS features are inconsistent and/or inferior to native CMake solutions and idioms. Examples:
  • Target-centric builds (compiler options, link options, include dirs., etc.)
  • Fortran/C name mangling (FortranCInterface.cmake)
  • Standard install locations (GNUInstallDirs.cmake)
  • RPATH Handling
  • Handling of deprecated code (GenerateExportHeader.cmake)
  • ...

• However, areas where (nearly) everyone seems to agree native CMake is lacking where a (reduced) TriBITS provides value:
  • Package architecture for CMake projects (e.g. VTK Modules)
  • Helper functions for defining and managing tests (e.g. MPI, allocating tests to GPUs, limiting tests based on MPI ranks and threads, etc.)
What is Modern CMake?

CMake library target objects contain full usage requirements, example:

```cmake
add_library(<libname> ...) # Internally built library or IMPORTED library
target_compile_definitions(<libname> PUBLIC COMPILe_DEFINE=1)
target_compile_features(<libname> PUBLIC cxx_std_17)
target_compile_options(<libname> PUBLIC -O2 PRIVATE -O5)
target_include_directories(<libname> PUBLIC /base/dir/pub PRIVATE /base/dir/priv)
target_link_directories(<libname> ...)
target_link_options(<libname> -mkl)
```

and propagate dependencies using `target_link_libraries()`:

```cmake
target_link_libraries( <downstreamExecOrLib>
    [PRIVATE|PUBLIC|INTERFACE] <upstreamLib> )
```

**<Package>Config.cmake**: Each CMake “Package” installs a package config file that defines IMPORTED targets and pulls in all upstream dependencies automatically:

```cmake
find_dependency(<upstreamPackage>) # Pulls in upstream dependencies!
add_library(<Package>::<libname> IMPORTED)
...
```

Downstream CMake projects pull in these external packages using `find_package(<externalPackage>)`
Componentized CMake-based Projects Approaches

CMake, CTest, and CDash are great, but raw usage does not scale very well to large projects and multiple repositories and teams!

• **Multiple CMake projects:**
  • Manual builds and linking through `<Package>Config.cmake` files
  • **CMake ExternalProject**: Provided as standard CMake module (raw CMake)
  • **CApp**: Lightweight CMake package manager by Dan Ibanez (raw CMake and git)
  • **Google Catkin**: Used for the Google Robotics Operating System (ROS) project (requires Python)
  • **Spack**: Source builds/package manager used in ECP project and E4S (requires Python)
  • Likely many others as well …

• **Single CMake project:**
  • Kitware **VTK Modules**:
  • TriBITS:
    + Support multiple repos
    + Core functionality depends only on CMake 3.17+

**TriBITS Goal** => Develop CMake packages that allow building in single CMake projects or in separate CMake projects in arbitrary sets depending on need.
Refactoring TriBITS CMake Build System to Modern CMake

Goals for updated Trilinos (TriBITS) build system\(^\ddagger\):

- Allow packages to use raw CMake to define targets for libraries, executables, etc. according to the proposed standard (e.g. provide `<Package>::<lib>` and `<Package>::all_libs`)
- Use `tribits_add_test()`, `tribits_add_advanced_test()` and even `tribits_add_executable_and_test()` to define tests.
- Use TriBITS external package/TPL system to find external packages (i.e. combine requirements from all enabled packages and call `find_package()` just once per each external package/TPL).
- TriBITS refactoring should allow existing packages to keep working without out modification.
- The decision to use `tribits_add_library()` and `tribits_add_executable()` and other optional TriBITS convenience functions and can be made on a package-by-package basis.

\(^\ddagger\) See [TriBITS #342](https://github.com/trilinos/TriBITS/issues/342)

Constraints/Requirements:

- **Not break existing CMakeLists.txt files** in existing TriBITS projects including Trilinos, Drekar, Charon2, etc. [Successful]
- **Not break existing user Trilinos and other configure scripts.** [Successful]
- Allow trimming down TriBITS and switching to native CMake in each TriBITS project to occur incrementally. [Successful (so far)]
- Allow refactoring of existing Trilinos packages to use raw CMake targets and build independently from Trilinos to occur incrementally. [Not started yet]
Generalized Handling of External and Internal Packages

Refactoring of TriBITS to modern CMake targets to deal with internal and external packages consistently [COMPLETE]
- `<Package>::<lib>`: Single (library) target (Self-contained modern CMake target which contains include directories, compiler options, link options, etc.), from:
  - Standard library target for **internal (TriBITS) packages** built within the CMake project, or
  - IMPORTED target for **external packages defined with `<Package>Config.cmake` files**, or
  - IMPORTED target generated from a legacy TriBITS TPL specification.
- `<Package>::all_libs`: INTERFACE (IMPORTED) library target for all libraries for internal or external `<Package>`
  - From internal packages, or from external `<Package>Config.cmake` file, or from generated from a TriBITS TPL specification

Refactoring of TriBITS dependency logic to deal with internal and external packages consistently: [ALMOST COMPLETE]
- Treat internal packages as external packages and visa versa

Driving Use Cases:
- Allow an existing TriBITS project to be built and installed in smaller CMake projects. Examples:
  - **Build and install Kokkos, Kokkos-Kernel, and SEACAS as independent CMake projects** and pull them in as KokkosConfig.cmake, KokkosKernelsConfig.cmake, and SEACAS<Subpackage>Config.cmake files and build the rest of Trilinos.
  - **Build and install Tpetra and Belos as independent CMake projects** pulling in pre-installed Kokkos and KokkosKernels.
- **Allow any TriBITS package to be pulled out and built as an independent CMake project** building against pre-installed upstream packages as external packages.
Trilinos Build/Install flexibility with updated TriBITS

- Kokkos
- Kokkos-Kernels
- Zoltan
- SEACAS
- Tpetra
- Belos
- Remainder of Trilinos: MueLu, Panzer, NOX, etc...

Spack

Internal SNL Customers
Trilinos Developers
ECP Customers
Finding external packages in CMake

```
find_package(<Package> [<version>] [MODULE|CONFIG] [COMPONENTS <c1> <c2> …] … )
```

- Finds (uses) either `Find<Package>.cmake` find module or `<Package>Config.cmake` package config file!
- Sets `<Package>_FOUND=TRUE` if found

```
find_package(<Package> MODULE …)
```

- Use a `Find<Package>.cmake` find module found in `CMAKE_MODULE_PATH`
- Does **not** set `<Package>_DIR`!

```
find_package(<Package> CONFIG …)
```

- On output, sets `<Package>_DIR !=""`
- On input, if `<Package>_DIR !=""` and package at `${<Package>_DIR}` package does not satisfy usage requirements, CMake will start find from scratch! (see discussion in [CMake Issue #23685](https://cmake.org/issue/23685))

**NOTE:** The older `Find<Package>.cmake` package find modules are only used as last resort (and are being phased out as much as possible).
CMake Packages and the Package Ecosystem Issues

1) No standard name for target for “all the library targets for <Package>”, examples:
   - Boost::boost => Only include dirs
   - HDF5::hdf5 => C libraries ; HDF5::HDF5 => All libraries (and changes with different HDF5 versions)
   - netCDF::netcdf => All libraries

2) No uniform support for IMPORTED targets and find_dependency() on upstream dependent packages, examples:
   - Official find module FindBullet.cmake in CMake 3.25 does not yet support IMPORTED targets
   - Latest netCDFConfig.cmake file does not call find_dependency(HDF5) (see Trilinos GitHub PR #11175)

3) Finding inconsistent upstream packages (see discussion in CMake Issue #23685), examples:
   - SomePackage versions 3 and 5 installed: First find_package(SomePackage 3...6) => 5, Second find_package(SomePackage 2...4) => 3  (But installed version 3 works for both!)

4) Non-scalable find_dependency() calls and package components (see discussion in CMake Issue #23685), example:
   - Using standard CMake approach results in find_package(Trilinos) taking upwards of 30 minutes!

These are fundamental problems with the CMake Package Ecosystem!

Existing solutions to these problems?
   - => Spack solves the problem of finding inconsistent upstream packages (#3 above)
TriBITS Solution to CMake Packages and the Package Ecosystem Issues

1) No standard name for target for “all the library targets for <Package>”:
   • => New standard INTERFACE target `<Package>::all_libs` for all external packages/TPLs and internal packages

2) No uniform support for IMPORTED targets and `find_dependency()` on upstream dependent packages
   • => TriBITS TPL dependencies and TriBITS-generated `<tplName>Config.cmake` files provide automatic namespaced IMPORTED targets and `find_dependency()` calls (e.g. fixes usage of broken `netCDFConfig.cmake` file)

3) Finding inconsistent upstream packages
   • => Aggregate usage requirements up-front and call `find_package(<upstreamTPL> ...) once` with consistent usage requirements that satisfy all downstream TPLs and TriBITS packages

4) Non-scalable `find_dependency()` calls and package components
   • => Finer-grained `<SubPackage>Config.cmake` files, with no COMPONENTS
   • => Don’t call `find_package()` with COMPONENTS argument so can use guard with:

```cmake
if (NOT TARGET <upstreamPackage>::all_libs)
  find_dependency(<upstreamPackage>)
endif()
```
Challenge: Provide standard self-contained modern CMake targets `<tplName>::all_libs` for all external packages/TPLs specified in different ways:

1. Legacy TriBITS TPLs: List of include directories, libraries, link options, etc. `TPL_<tplName>_INCLUDE_DIRS` and `TPL_<tplName>_LIBRARIES` variables?
   
   => **Solution:** Automatically handled by refactored TriBITS

2. Pre-installed upstream TriBITS packages?
   
   => **Solution:** Automatically handled by refactored TriBITS

3. Using `find_package(<tplName>)` to find external standard (or non-standard) `Find<tplName>.cmake` module or `<tplName>Config.cmake` file provided by an external package/TPL?
   
   => **Solution:** Create custom `FindTPL<tplName>.cmake` files that call `find_package(<tplName>)` and construct self-contained `<tplName>::all_libs` target.

**NOTE:** The need to create custom `FindTPL<tplName>.cmake` files where (partial) modern CMake is used with `Find<tplName>.cmake` find modules or `<tplName>Config.cmake` package config files to provide IMPORTED targets is where a majority of work of developers will be expended in transitioning to modern CMake 😊
Challenge: Support existing TriBITS TPL specifications through:

- `D <tplName>_INCLUDE_DIRS="<Idir1>;<Idir2>;..."`
- `D <tplName>_LIBRARY_NAMES="<name1>;<name2>;..."`
- `D <tplName>_LIBRARY_DIRS="<Ldir1>;<Ldir2>;..."`

(which are resolved using find_() calls) or explicitly through:

- `D TPL_<tplName>_INCLUDE_DIRS="<Idir1>;<Idir2>;..."
- `D TPL_<tplName>_LIBRARIES="/full/path/to/lib<libname1>.so;-L<dir2>;-l<libname2>;<libname3>;..."

and create `<tplName>Config.cmake` package config files with modern CMake IMPORTED library targets and linked targets with upstream external packages/TPLs. These files are installed and loaded from the build directory:

```
<buildDir>/external_packages/<tplName>/<tplName>Config.cmake
```

and install directory under:

```
<installDir>/lib/external_packages/<tplName>/<tplName>Config.cmake
```

- **NOTE:** Arbitrary link options can be translated into IMPORTED library targets **but can’t maintain the needed ordering of the link line.** Example: `-Wl,-Bstatic -l<libname>` cannot be handled!

- **No known breakages to any existing Trilinos configure scripts!**
Generated <tplName>Config.cmake files for Legacy TPLs

Legacy TPL configure arguments:
-D TPL_SomeTpl_INCLUDE_DIRS="/some/path/to/include/a" \
-D TPL_SomeTpl_LIBRARIES="-llib2; -L/some/explicit/path2; -lmkl; -llib1; -L/some/explicit/path1"

Generated SomeTplConfig.cmake file:

```cpp
if (TARGET SomeTpl::all_libs)
    return()
endif()

add_library(SomeTpl::lib1 IMPORTED INTERFACE)
set_target_properties(SomeTpl::lib1 PROPERTIES IMPORTED_LIBNAME "lib1")

add_library(SomeTpl::lib2 IMPORTED INTERFACE)
set_target_properties(SomeTpl::lib2 PROPERTIES IMPORTED_LIBNAME "lib2")
target_link_libraries(SomeTpl::lib2 INTERFACE SomeTpl::some-other-option)

Continued ...
```

... Continued

```cpp
add_library(SomeTpl::all_libs INTERFACE IMPORTED)
target_link_libraries(SomeTpl::all_libs INTERFACE SomeTpl::lib1 INTERFACE SomeTpl::some-other-option INTERFACE SomeTpl::lib2 )
target_include_directories(SomeTpl::all_libs SYSTEM INTERFACE "/some/path/to/include/a")
target_link_options(SomeTpl::all_libs INTERFACE "-L/some/explicit/path2" INTERFACE "-mkl" INTERFACE "-L/some/explicit/path1")
```
Creating FindTPL<tplName>.cmake using `find_package()` with IMPORTED targets

```cmake
find_package(<externalPkg> REQUIRED)
tribits_extpkg_create_imported_all_libs_target_and_config_file(
    <tplName>
    INNER_FIND_PACKAGE_NAME <externalPkg>
    IMPORTED_TARGETS_FOR_ALL_LIBS <importedTarget0> <importedTarget1> ...
)
```

Creating FindTPL<tplName>.cmake using `find_package()` without IMPORTED targets

```cmake
find_package(<externalPkg> REQUIRED)
set(TPL_<tplName>_INCLUDE_DIRS ${<externalPkg>_INCLUDE_DIRS} CACHE PATH "...")
set(TPL_<tplName>_LIBRARIES ${<externalPkg>_LIBRARIES} CACHE FILEPATH "...")
set(TPL_<tplName>_LIBRARY_DIRS ${<externalPkg>_LIBRARY_DIRS} CACHE PATH "...")
tribits_tpl_find_include_dirs_and_libraries( <tplName>
    REQUIRED_HEADERS neverFindThisHeader
    REQUIRED_LIBS_NAMES neverFindThisLib
)
```

Creating a FindTPL<tplName>.cmake module without `find_package()`

```cmake
tribits_tpl_find_include_dirs_and_libraries( <tplName>
    REQUIRED_HEADERS <header0> <header1> ...
    REQUIRED_LIBS_NAMES <libname0> <libname1> ...
    MUST_FIND_ALL_LIBS )
```
Define TPL dependencies file:

```c
<tplDefsDir>/
...
FindTPL<tplName>.cmake
  FindTPL<tplName>Dependencies.cmake
...
```

Example: `FindTPLLLAPACKDependencies.cmake`:

```c
tribits_extpkg_define_dependencies( LAPACK
  DEPENDENCIES  BLAS )
```

NOTES:
- IMPORTED targets in `LAPACKConfig.cmake` are linked against `BLAS::all_libs`
- Currently, to preserve backwards compatibility, enabling `TPL_ENABLE_<downstreamTPL>=ON` does **not automatically enable** dependent `TPL_ENABLE_<upstreamTPL>=ON`
- Future, we should make setting `TPL_ENABLE_<downstreamTPL>=ON` automatically trigger `TPL_ENABLE_<upstreamTPL>=ON`
Generated \texttt{<tplName>Config.cmake} file with dependencies

Legacy TPL configure arguments:
- \texttt{-D TPL\_SomeTpl\_INCLUDE\_DIRS="/some/path/to/include/a"}
- \texttt{-D TPL\_SomeTpl\_LIBRARIES="-l\lib2;-L/some/path2;-l\lib1;-L/some/explicit/path1"}

### Generated \texttt{SomeTplConfig.cmake} file:

```cpp
if (TARGET SomeTpl::all libs)
  return()
endif()

if (NOT TARGET UpstreamTpl::all_libs)
  set(UpstreamTpl\_DIR "<...>/../UpstreamTpl")
  find_dependency(UpstreamTpl REQUIRED CONFIG)
endif()

add_library(SomeTpl::lib1 IMPORTED INTERFACE)
set_target_properties(SomeTpl::lib1 PROPERTIES IMPORTED\_LIBNAME "lib1")
target_link_libraries(SomeTpl::lib1 INTERFACE UpstreamTpl::all_libs)

add_library(SomeTpl::lib2 IMPORTED INTERFACE)
set_target_properties(SomeTpl::lib2 PROPERTIES IMPORTED\_LIBNAME "lib2")
target_link_libraries(SomeTpl::lib2 INTERFACE SomeTpl::lib1)

add_library(SomeTpl::all_libs INTERFACE IMPORTED)
target_link_libraries(SomeTpl::all_libs INTERFACE SomeTpl::lib1 INTERFACE SomeTpl::lib2)
target_include_directories(SomeTpl::all_libs SYSTEM INTERFACE "/some/path/to/include/a")
target_link_options(SomeTpl::all_libs INTERFACE "-L/some/path2" INTERFACE "-L/some/path1")
```

### Continued ...

```cpp
Continued...
```

... Continued

```cpp
add_library(SomeTpl::lib2 IMPORTED INTERFACE IMPORTED)
set_target_properties(SomeTpl::lib2 PROPERTIES IMPORTED\_LIBNAME "lib2")
target_link_libraries(SomeTpl::lib2 INTERFACE SomeTpl::lib1)

add_library(SomeTpl::all_libs INTERFACE IMPORTED)
target_link_libraries(SomeTpl::all_libs INTERFACE SomeTpl::lib1 INTERFACE SomeTpl::lib2)
target_include_directories(SomeTpl::all_libs SYSTEM INTERFACE "/some/path/to/include/a")
target_link_options(SomeTpl::all_libs INTERFACE "-L/some/path2" INTERFACE "-L/some/path1")
```
How TriBITS Modernization Impacts CMake Customers

Documentation:
- **TriBITS Build Reference Guide Documentation:**
  - 8.6 Using the installed software in downstream CMake projects
  - 8.7 Using packages from the build tree in downstream CMake projects
- Example projects:
  - [TribitsOldSimpleExampleApp](#) (works with old and new TriBITS)
  - [TribitsSimpleExampleApp](#)
  - [TribitsExampleApp](#)

From [TribitsSimpleExampleApp/CMakeLists.txt](#):

```cmake
find_package(TribitsExProj REQUIRED COMPONENTS SimpleCxx MixedLang WithSubpackages)
...
add_executable(app app.cpp)
target_link_libraries(app PRIVATE TribitsExProj::all_selected_libs)
```

Or, link to `<packageName>::all_libs` for external packages/TPLs and TriBITS packages!

Also, could use individual `find_package(SimpleCxx)`, `find_package(MixedLang)`, `find_package(WithSubpackages)` calls to avoid scalability problems with downstream CMake projects!
Keeping and breaking backwards compatibility

• Avoid breaking hundreds (or thousands) existing Trilinos configure scripts across the world
  ⇒ Maintained near perfect backward compatibility!

• Avoid needing to refactor thousands of existing TriBITS project CMakeLists.txt files
  ⇒ Maintained near perfect backwards compatibility! (only invalid TriBITS usage was an issue)

• Avoid changes to downstream CMakelake projects pulling in installed Trilinos
  ⇒ Changed from “-I <include-dir>” to “-isystem <include-dir>” (required by CMake)
    ⇒ Changes order of searching include directories (broke SPARC build)
  ⇒ Trilinos_LIBRARY no longer contains raw library names (broke Albany)
  ⇒ Non-namespaced library targets are deprecated (broke Albany initially)
  ⇒ Trilinos_TPL_INCLUDE_DIRS is now empty (broke SPARC)
  ⇒ Other examples ...
Summary: Current Status and Next Steps

- Refactor to internal usage of modern CMake targets and for treating internal and external packages uniformly
  - Clean linking against `<Package>::<libname>` and `<Package>::all_libs` for internal and external packages (and strip out old TriBITS logic) [COMPLETE]
  - Uniform dependency handling and treatment external packages/TPLs and internal packages (including between external packages) [NEARLY COMPLETE]

Next:
- Building and installing upstream selected packages independently:
  - Prebuild and install Kokkos and KokkosKernels and build remaining Trilinos package against these
  - Prebuild and install SEACAS (against pre-installed Kokkos and Zoltan) and build remaining Trilinos packages against these.
- TriBITS Meta packages:
  - ShyLU: Where `Trilinos_ENABLE_ShyLU=[ON|OFF]` and `ShyLU_ENABLE_TESTS=[ON|OFF]` behaves like it is a package and `ShyLU_Node` and `ShyLU_DD` are its subpackages

To keep track of progress:
- TriBITS Refactor Kanban Board (Project Board #2)
- EPIC: TriBITS Modernization Plan (TriBITS #367)
- Bi-weekly meeting TriBITS Modernization Meetings
- Selected SEMS Review meetings
Questions and Comments?