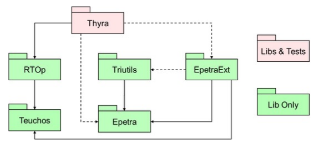
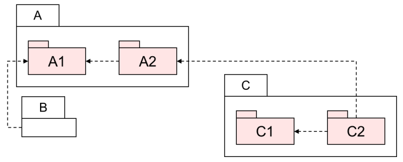
Modern TriBITS





Roscoe A. Bartlett
Department 1424
Software Engineering and Research

November 2, 2023

Trilinos Users Group Meeting, Developers Day

What is Modern CMake?

CMake **library target objects contain full usage requirements**, example:

```
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 usage required and dependencies using `target_link_libraries()`:**

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

What is a Modern CMake External Package?

<Package>Config.cmake: *Package config file* defines IMPORTED targets and pulls in all upstream dependencies automatically:

```
find_dependency(<upstreamPackage> REQUIRED)  # Pulls in upstream dependencies!
add_library(<Package>::<libname> IMPORTED [SHARED|STATIC])
…
```

Downstream CMake projects pull in these external packages using find_package(<externalPackage>)

Example Minimal Raw Modern CMake Package

<pakageDir>/CMakeLists.txt

```
cmake_minimum_required(
  VERSION 3.23.0 FATAL_ERROR)
project(Package1 LANGUAGES C CXX)
include(GNUInstallDirs)
find_package(Tpl1 CONFIG REQUIRED)
add_subdirectory(src)
if (Package1_ENABLE_TESTS)
  include(CTest)
  add_subdirectory(test)
endif()
```

<pakageDir>/src/CMakeLists.txt

```
add_library(package1 Package1.hpp Package1.cpp)
target_include_directories(package1
  PUBLIC $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}>)
target_link_libraries(package1 PRIVATE tpl1::tpl1 )

add_executable(package1-prg Package1_Prg.cpp)
target_link_libraries(package1-prg PRIVATE package1)
```

<pakageDir>/test/CMakeLists.txt

```
add_test(NAME Package1_Prg
  COMMAND package1-prg)
set_tests_properties(Package1_Prg
  PROPERTIES PASS_REGULAR_EXPRESSION
  "Package1 Deps: tpl1")
```

Limitations (i.e. NOT “Professional CMake” compliant):

- Does **not** create namespaced targets (e.g. Package1::package1)
- Does **not** install libraries, header files, or executables
- Does **not** install a <Package>Config.cmake file

Example Minimal Raw Modern CMake External Package

`<installDir>/lib/cmake/Tpl2/Tpl2Config.cmake`

```
if (TARGET Tpl2::tpl2a)
  return()
endif()

find_dependency(Tpl1 REQURIED)

add_library(Tpl2::tpl2a IMPORTED SHARED)
set_target_properties(Tpl2::tpl2a PROPERTIES
  IMPORTED_LOCATION "<installDir>lib/libtpl2a.so")
target_include_directories(Tpl2::tpl2b SYSTEM
  INTERFACE "/<installDir>/include")
target_link_libraries(Tpl2::tpl2a
  INTERFACE $<LINK_ONLY:Tpl1::tpl1> )

add_library(Tpl2::tpl2b IMPORTED SHARED)
set_target_properties(Tpl2::tpl2b PROPERTIES
  IMPORTED_LOCATION "<installDir>lib/libtpl2b.so")
target_include_directories(Tpl2::tpl2b SYSTEM
  INTERFACE "/<installDir>/include")
target_link_libraries(Tpl2::tpl2b
  INTERFACE Tpl2::tpl2a)
```

Consistent with “Professional CMake”:

- Pulls in upstream dependencies (i.e. Tpl1)
- Defines namespaced IMPORTED targets
- IMPORTED targets CMake code can be created by CMake project automatically
- Non-CMake projects can manually create and install these files

A modern CMake project must write two CMake programs!

1. Containing CMakeLists.txt files to configure, build, test, and install the package
2. An installed <Package>Config.cmake file that downstream CMake projects run to access the installed package

Refactored TriBITS CMake Build System to Modern CMake

Goals for initial Trilinos (TriBITS) build system refactorʕ**: [COMPLETE]**

- **Allow packages to use raw CMake to define targets** for libraries, executables, using modern CMake and (e.g. provide <Package>::<lib> and <Package>::all_libs).
- **Use TriBITS functionality to define tests** using tribits_add_test(), tribits_add_advanced_test() and even tribits_add_executable_and_test() .
- **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 to define targets and other optional functionality can be made on a package-by-package basis** (e.g. tribits_add_library() and tribits_add_executable()).

ʕ See TriBITS #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 refactoring of existing Trilinos packages to use raw CMake targets and build independently from Trilinos to occur **incrementally**. **[Successful]**
- Allow trimming down TriBITS and switching to native CMake in each TriBITS project to occur as desired **incrementally**. **[Successful (so far)]**

How are existing TriBITS packages using Modern CMake?

Example TriBITS CMakeLists.txt file

```
include_directories(
  ${CMAKE_CURRENT_SOURCE_DIR})

tribits_add_library(package1
  HEADERS   Package1.hpp
  SOURCES   Package1.cpp)
```

Trilinos CMake build system was upgraded to use Modern CMake without touching:

- 1776 CMakeLists.txt files
- 229 tribits_add_library() calls
- 630 tribits_add_executable() calls
- 1393 tribits_add_test() calls
- 284 tribits_add_advanced_test() calls
- 2206 tribits_add_executable_and_test() calls

What tribits_add_library() is doing under the covers?

```
get_directory_property(includeDirsCurrent
  INCLUDE_DIRECTORIES)

add_library(Package1_package1
  Package1.hpp Package1.cpp)
target_include_directories(Package1_package1
  PUBLIC $<BUILD_INTERFACE:${includeDirsCurrent}>)
set_target_properties(Package1_package1 PROPERTIES
  EXPORT_NAME package1)
target_link_libraries(Package1_package1
  PUBLIC Tpl1::all_libs )

add_library(Package1::package1 ALIAS Package1_package1)

install(TARGETS Package1_package1
  EXPORT ${PACKAGE_NAME}
  INCLUDES DESTINATION ${CMAKE_INSTALL_INCLUDEDIR} )

install(FILES Package1.hpp
  DESTINATION ${CMAKE_INSTALL_INCLUDEDIR} )
```

* Consistent with Modern CMake Advocated in:

"Professional CMake", by Craig Scott

Handling of External Packages/TPLs

Finding external packages in raw 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 **or** <Package>_CONFIG vars!

find_package(<Package> CONFIG …)

- On output, sets <Package>_DIR != “” **and** <Package>_CONFIG != “”
- On input, if <Package>_DIR != “” and package at ${<Package>_DIR} does not satisfy usage requirements, CMake will start find from scratch! (see discussion in CMake 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 by the CMake community).

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
- Recent netCDFConfig.cmake file 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!)

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: Modern CMake with External Packages/TPLs

Challenge: Create TriBITS-Compliant External Package <tplName>config.cmake files for every external packages/TPLs no matter how they are defined:

1. **Legacy TriBITS TPLs**: List of include directories, libraries, link options, etc. TPL_<tplName>_INCLUDE_DIRS and TPL_<tplName>_LIBRARIES variables:

   => Automatically handled by refactored TriBITS through legacy FindTPL<tplName>.cmake files
2. **Using find_package(<externalPkg>) to find other external packages**: Find<tplName>.cmake module or <tplName>Config.cmake file with or without modern CMake IMPORTED targets:

   => Create custom FindTPL<tplName>.cmake files that call find_package(<tplName>) and construct self-contained <tplName>::all_libs target.
3. **Pre-installed upstream TriBITS-compliant packages**

   => Automatically handled by refactored TriBITS

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 inreally transitioning to modern CMake ☹

TriBITS Generated <tplName>Config.cmake and <Package>Config.cmake files

Build Directory:

```
<buildDir>/
  external_packages/
    <tpl1>/
      <tpl1>Config.cmake
    <tpl2>/
      <tpl2>Config.cmake
    …
  cmake_packages/
    <package1>/
      <package1>Config.cmake
    <package2>/
      <package2>Config.cmake
    …
  packages/
```

Generated <tplName>Config.cmake files are included by <packageName>config.cmake files to provide <tplName>::all_libs targets. **They are not meant to be found by find_package(<tplName>) calls!**

Can use built packages without installing with:
  -D CMAKE_PREFIX_PATH=<buildDir>/cmake_packages

Install Directory:

```
<installDir>/
    lib[64]/
      external_packages/
        <tpl1>/
          <tpl1>Config.cmake
        <tpl2>/
          <tpl2>Config.cmake
        …
      cmake/
        <package1>/
          <package1>Config.cmake
        <package2>/
          <package2>Config.cmake
        …
```

Installed <tplName>Config.cmake files are included by <packageName>config.cmake files to provide <tplName>::all_libs targets. **They are not meant to be found by find_package(<tplName>) calls!**

Using installed packages:
  -D CMAKE_PREFIX_PATH=<installDir>

Generated <tplName>Config.cmake files for TriBITS 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"
```

TriBITS-Generated SomeTplConfig.cmake file:

```
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

```
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"
  )
```

TriBITS External Package/TPL Dependencies

Define TPL dependencies file:

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

Example: `FindTPLLAPACKDependencies.cmake:`

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

NOTES:

- **Dependencies needed to have the libraries listed on the link line in the correct order!**
- IMPORTED targets in LAPACKConfig.cmake are linked against BLAS::all_libs
- Currently, to preserve backwards compatibility, enabling TPL_ENABLE_<dowstreamTPL>=ON **does not automatically enable** dependent TPL_ENABLE_<upstreamTPL>=ON
- Future, support optional and required upstream TPL dependencies? (**Break backward compatibility!**)

Generated <tplName>Config.cmake file for TriBITS Legacy TPL with dependencies

Legacy TPL configure arguments:

```
-D TPL_SomeTpl_INCLUDE_DIRS="/some/path/to/include/a" \
-D TPL_SomeTpl_LIBRARIES="-llib2;-L/some/path2;-llib1;-L/some/explicit/path1" \
```

TriBITS-Generated <tplName>Config.cmake file:

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

if (NOT TARGET UpstreamTpl::all_libs)
  include("<…>/../UpstreamTpl/UpstreamTplConfig.cmake")
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)
```

Continued …

… Continued

```
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_lib
  SYSTEM INTERFACE "/some/path/to/include/a")
target_link_options(SomeTpl::all_libs
  INTERFACE "-L/some/path2"
  INTERFACE "-L/some/path1")
```

Generated <tplName>Config.cmake files using find_package() with modern CMake IMPORTED targets

FindTPLTpl2.cmake:

```
find_package(Tpl2 REQUIRED)
tribits_extpkg_create_imported_all_libs_target_and_config_file(Tpl2
  INNER_FIND_PACKAGE_NAME  Tpl2
  IMPORTED_TARGETS_FOR_ALL_LIBS tpl2::tpl2a tpl2::tpl2b)
```

FindTPLTpl2Dependencies.cmake:

```
tribits_extpkg_define_dependencies(
  Tpl2
  DEPENDENCIES  Tpl1)
```

TriBITS-Generated Tpl2Config.cmake wrapper file:

```
# <comments …>

# Guard against multiple inclusion
if (TARGET Tpl2::all_libs)
  return()
endif()

if (NOT TARGET Tpl1::all_libs)
  include(
    "${CMAKE_CURRENT_LIST_DIR}/../Tpl1/Tpl1Config.cmake")
endif()

include(CMakeFindDependencyMacro)

set(Tpl2_DIR "<tpl2InstallDir>/lib/cmake/Tpl2")
find_dependency(Tpl2)
```

Continued …

Continued …

```
add_library(Tpl2::all_libs INTERFACE IMPORTED)
target_link_libraries(Tpl2::all_libs
  INTERFACE tpl2::tpl2a
  INTERFACE tpl2::tpl2b
  )
target_link_libraries(Tpl2::all_libs
  INTERFACE $<LINK_ONLY:Tpl1::all_libs>  # i.e. PRIVATE
  )

# Standard TriBITS-compliant external package variables
set(Tpl2_IS_TRIBITS_COMPLIANT TRUE)
set(Tpl2_TRIBITS_COMPLIANT_PACKAGE_CONFIG_FILE
  "${CMAKE_CURRENT_LIST_FILE}")
set(Tpl2_TRIBITS_COMPLIANT_PACKAGE_CONFIG_FILE_DIR
  "${CMAKE_CURRENT_LIST_DIR}")
```

TriBITS Uniform Handling of Internal and External Packages

TriBITS Uniform Treatment of Internal and External Packages

Any internally defined TriBITS Package <Pkg> can be pre-build/installed and pulled in with:

```
-D TPL_ENABLE_<Pkg>=ON \
-D CMAKE_PREFIX_PATH=<pkgInstallDir> \
```

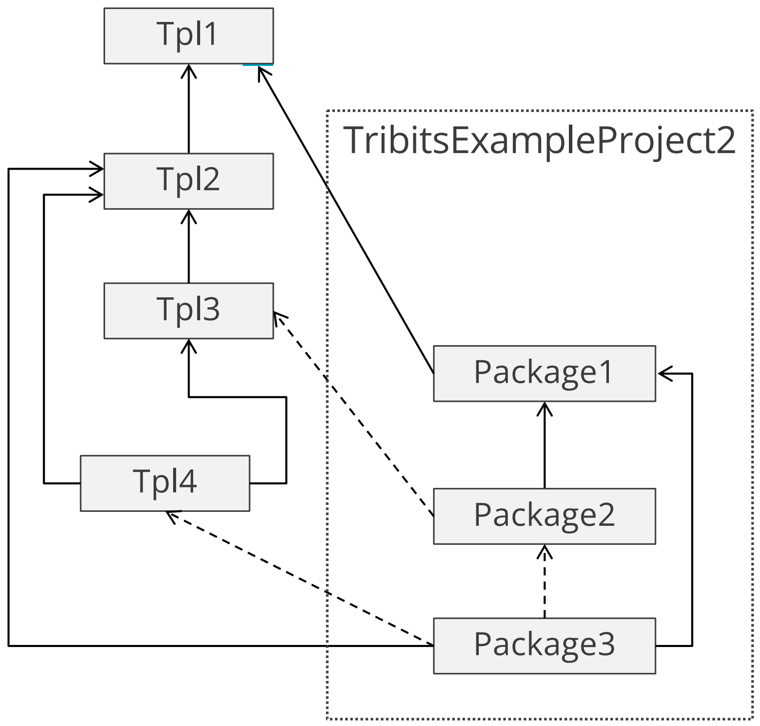
Has the following effect:

- The package <Pkg> is **enabled in the dependency logic** just as of -D <Project>_ENABLE_<Pkg>=ON was set.
- The package <Pkg> is **treated an TriBITS-compliant external package** and the **internal CMakeLists.txt file is slipped** and instead find_package(<Pkg> CONFIG REQUIRED) is called.
- Every package **<UpstreamPkg>** upstream from <Pkg> is **also an external package**.

Finding the External Packages/TPLs is done in two loops:

1. In reverse order, loop over enabled TriBITS-compliant external packages and call find_package(…).
2. In forward order, look over remaining enabled TriBITS external packages and use FindTPL<tplName>.cmake module to find <tplName>.

TribitsExampleProject2: Pre-build/install packages example



CMake Configure Input:

```
  -DTPL_ENABLE_Package2=ON \
  -DCMAKE_PREFIX_PATH="<pkg2InstallDir>;<tpl4InstallDir>" \
  -DTribitsExProj2_ENABLE_ALL_PACKAGES=ON \
```

CMake Configure Output:

```
Adjust the set of internal and external packages:

-- Treating internal package Package2 as EXTERNAL because
TPL_ENABLE_Package2=ON
-- Treating internal package Package1 as EXTERNAL because
downstream package Package2 being treated as EXTERNAL
-- NOTE: Tpl3 is directly upstream from a TriBITS-compliant
external package Package2
-- NOTE: Tpl2 is indirectly upstream from a TriBITS-compliant
external package
-- NOTE: Tpl1 is indirectly upstream from a TriBITS-compliant
external package

<…>

Final set of enabled packages:  Package3 1

Final set of enabled external packages/TPLs:  Tpl1 Tpl2 Tpl3
Tpl4 Package1 Package2 6
```

TribitsExampleProject2: Pre-build/install packages example

CMake Configure Output (Continued)

```
Getting information for all enabled TriBITS-compliant or upstream external packages/TPLs in
reverse order ...

Processing enabled external package/TPL: Package2 (enabled explicitly, disable with <…>)
-- Calling find_package(Package2) for TriBITS-compliant external package
-- Found Package2_DIR= '<pkg2InstallDir>/lib/cmake/Package2'
Processing enabled external package/TPL: Package1 (enabled explicitly, disable with <…>)
-- The external package/TPL Package1 was defined by a downstream TriBITS-compliant external
package already processed
Processing enabled external package/TPL: Tpl3 (enabled explicitly, <…>)
-- The external package/TPL Tpl3 was defined by a downstream TriBITS-compliant external
package already processed
Processing enabled external package/TPL: Tpl2 (enabled explicitly, disable with <…>)
-- The external package/TPL Tpl2 was defined by a downstream TriBITS-compliant <…>
Processing enabled external package/TPL: Tpl1 (enabled explicitly, disable with <…>)
-- The external package/TPL Tpl1 was defined by a downstream TriBITS-compliant <…>

Getting information for all remaining enabled external packages/TPLs ...

Processing enabled external package/TPL: Tpl4 (enabled explicitly, disable <…>)
<…>
```

TribitsExampleProject2: Pre-build/install packages example

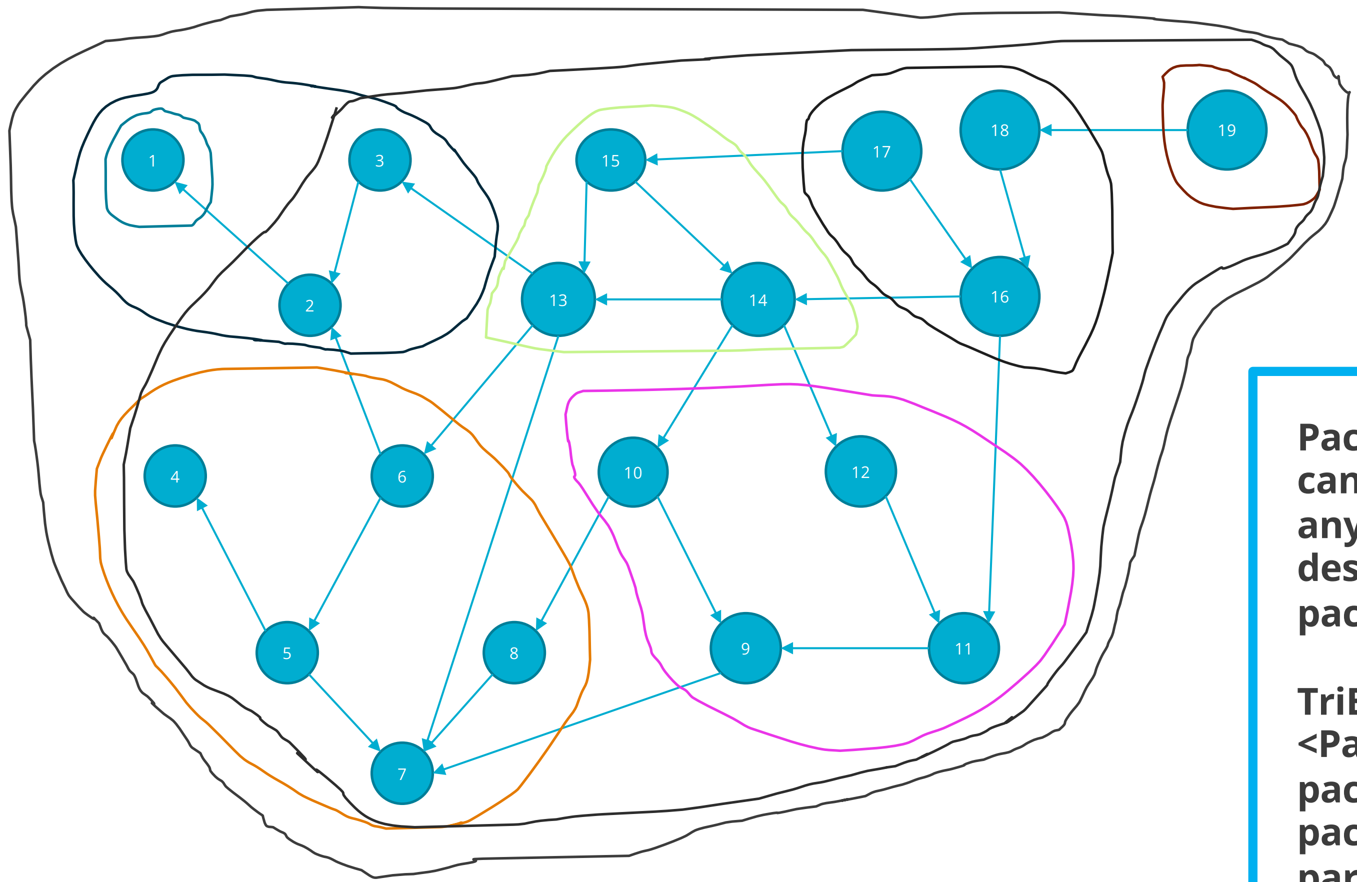
CMake Configure Output (Continued)

```
<…>

Configuring individual enabled TribitsExProj2 packages ...

Processing enabled top-level package: Package3 (Libs, Tests, Examples)

<…>

-- Configuring done
-- Generating done
```

Important Points:

- Only CMakeLists.txt file or Package3 is processed! (**Not** for Package1 or Package2)
- Calling find_package() for TriBITS-compliant external packages/TPLs in reverse order:
  - => Avoids finding inconsistent packages (e.g. a different Package1 than being used by Package2)
  - => Allows pulling an indirect <Package>Config.cmake file that can't be found in the current CMAKE_PREFIX_PATH (e.g. no search path for Package1Config.cmake)

Updated TriBITS: Flexible subgraph builds/installs



Example TriBITS Project Package Dependency Graph

Build/Install As:

1 CMake Project (fastest)
2 CMake Projects
6 CMake Projects
19 CMake Projects (slowest)

Package dependency graph can be build/installed with any subgraph partitioning desired (Including one-package-at-a-time).

TriBITS generates a <Package>Config.cmake package config file for each package independent of partitioning!

Build Trilinos against pre-installed Kokkos

Implementation in Kokkos, and Spack (beyond core TriBITS refactorings):

- **Removed subpackages** from the TriBITS build of Kokkos under Trilinos:
  - ⇒ Touched many Trilinos Packages
- **Extended native non-TriBITS Kokkos CMake build system**:
  - ⇒ Added some missing Kokkos_XYZ variables to installed KokkosConfig.cmake file
  - ⇒ Added Kokkos::all_libs target to KokkosConfig.cmake file
- **Updated Spack trilinos/package.py file**:
  - ⇒ Added dependency on Spack 'kokkos' package (with a complex set of constraints)
  - ⇒ Added -D TPL_ENABLE_Kokkos=ON to Trilinos CMake configure input

Impact on Customers on updated Spack Trilinos package:

- 'Kokkos' is no longer a COMPONENT of Trilinos!
  - find_package(Trilinos COMPONENTS Kokkos Tpetra …) => **Error: Kokkos not part of Trilinos!**
    - **Solution 1**: find_package(Trilinos COMPONENTS Tpetra …) ; find_package(Kokkos)
    - **Solution 2**: find_package(Kokkos); find_package(Tpetra); …

Splitting up Trilinos into multiple Spack package installs?

Options to break Trilinos into multiple Spack packages?

- **Option 1**: **Pull out Spack packages only as needed** (Current approach): E.g.:
  - Kokos, KokkosKernels, Zoltan, SEACAS, Trilinos
- **Option 2**: **Create Meta-Packages for Trilinos**: E.g.:
  - Kokkos, KokkosKernels, Zoltan, SEACAS, TrilinosTools, TrilinosDataStructures, TrilinosLinearSolvers, TrilinosNonlinearSolvers, TrilinosDiscretizations, …
- **Option 3**: **A Spack package for every Trilinos package**: E.g.:
  - Kokkos, Teuchos, RTOp, Tpetra, … ROL
- **Option 4**: **Create 'trilinos-dev' Spack package** to drive development in addition to above options

Impact on Trilinos Developers?

- The more Spack packages there are, the harder and slower Trilinos testing development will be (if using Spack to generate build environments)

Impact on Customers on new Spack Trilinos packages?

- Switch from find_package(Trilinos COMPONENTS Kokkos Tpetra MueLU … Piro) to:

  => find_package(Kokkos) ; find_package(Tpetra) ; find_package(MueLU) … find_package(Piro)
- Actually: It is better to call them in reverse package dependency order:

  => find_package(Piro) … find_package(MueLU) ; find_package(Tpetra); find_package(Kokkos)

Using Raw CMake for TriBITS-Compliant Internal and External Packages

Requirements for TriBITS-Compliant Packages

- Provides the (INTERFACE) target <Package>::all_libs which provides all usage requirements for the libraries of <Package> through the target properties:
- INTERFACE_LINK_LIBRARIES, INTERFACE_INCLUDE_DIRECTORIES, INTERFACE_COMPILE_OPTIONS, INTERFACE_COMPILE_DEFINITIONS, INTERFACE_LINK_OPTIONS, and any other INTERFACE_XXX or IMPORTED_XXX target property needed to correctly use the libraries for package <Package>.
- Provides namespaced variables <Package>_ENABLE_<UpstreamPackage> set to TRUE or FALSE for all of the upstream required and optional dependencies for the package <Package>.
- [Optional] Provides namespaced variables of the form <Package>_<SOME_INFO> for any other information about the configuration of package <Package> that may need to be known by a downstream TriBITS package.
- [Optional] Provides any (namespaced by <package>_ or <Package>_) CMake macros or functions that downstream CMake packages may need to use the upstream package <Package>.
- **[Optional] All of the upstream dependencies (listed in the INTERFACE_LINK_LIBRARIES property recursively) are also TriBITS-compliant packages**

Documentation link: TriBITS-Compliant Packages

Requirements for TriBITS-Compliant Internal Packages

- **All of the requirements for a TriBITS-Compliant Package**.
- At the end of configuration and generation, writes out a **TriBITS-Compliant External Package file** <Package>Config.cmake and supporting files **under the build directory** <buildDir>/cmake_packages/<Package>/ allowing the built (but not installed) package to be used by downstream CMake packages/projects.
- Provides an install target to **create a TriBITS-Compliant External Package file** <Package>Config.cmake and supporting files **under the install directory** <installDir>/lib/cmake/<Package>/ allowing the installed package to be used by downstream CMake packages/projects.
- **[Optional] All of the upstream dependencies (recursively) are also TriBITS-compliant packages.**

If a TriBITS package provides any CTest tests, then it must also satisfy the following requirements:

- Test names must be prefixed with the package name <Package>_.
- Tests should only be added if the variable <Package>_ENABLE_TESTS is true.
- Examples (that run as CTest tests) should only be added if the variable <Package>_ENABLE_EXAMPLES is true.
- The PROCESSORS test property and other test properties must be set in a way consistent with tribits_add_test() so as to run in parallel with other tests and not overwhelm the computing resources on the machine.
- The test <fullTestName> must not be added if the cache variable <fullTestName>_DISABLE is set to TRUE or if the cache variable <fullTestName>_SET_DISABLED_AND_MSG is set to non-empty (and the message string should be printed to STDOUT).

Documentation link: TriBITS-Compliant Internal Packages

Requirements for TriBITS-Compliant External Packages

- All of the requirements for a **TriBITS-Compliant Package**.
- Defined by an **installed <Package>Config.cmake file** that provides IMPORTED targets and set() statements for all of the needed variables.
- Provides CMake variables:
    - **<Package>_CONFIG** or **<Package>_TRIBITS_COMPLIANT_PACKAGE_CONFIG_FILE**: Points to the file <Package>Config.cmake (i.e. ${CMAKE_CURRENT_LIST_FILE})
    - **<Package>_DIR** or **<Package>_TRIBITS_COMPLIANT_PACKAGE_CONFIG_FILE_DIR:** Points to the base directory for <Package>Config.cmake (i.e. ${CMAKE_CURRENT_LIST_DIR})
- **[Optional] All of the upstream dependencies (recursively) are also provided as TriBITS-compliant external packages** with <UpstreamPackage>Config.cmake files (see above) and all of the targets and variables for a TriBITS-compliant external package are defined when the <Package>Config.cmake file is included (or pulled in with find_package() or find_dependency()).

Documentation link: TriBITS-Compliant External Packages

TriBITS-Compliant Packages Using Raw CMake HowTos

TriBITS Users Guide (see tribits.org)

- 10 Howtos:
    - ...
    - 10.10 How to implement a TriBITS-compliant internal package using raw CMake
    - 10.11 How to implement a TriBITS-compliant external package using raw CMake
    - 10.12 How to use TriBITS testing support in non-TriBITS project
    - ...

Snapshotted Trilinos packages that also maintain their own native CMake build system should consider using only (TriBITS-compliant) raw CMake, except for defining tests with tribits_add_test() when building under TriBITS project:

E.g.: Kokkos, KokkosKernels, STK, ...

NOTE: SEACAS uses TriBITS natively

TriBITS vs. Raw CMake TriBITS-Compliant CMake Package

package1/CMakeLists.tribits.cmake

```
tribits_package(Package1)
add_subdirectory(src)
tribits_add_test_directories(test)
tribits_package_postprocess()
```

package1/CMakeLists.raw.cmake

```
cmake_minimum_required(VERSION 3.23.0 FATAL_ERROR)

if (COMMAND tribits_package)
  message("Configuring raw CMake package Package1")
else()
  message("Configuring raw CMake project Package1")
endif()

# Standard project-level stuff
project(Package1 LANGUAGES C CXX)
include(GNUInstallDirs)
find_package(Tpl1 CONFIG REQUIRED)
add_subdirectory(src)
if (Package1_ENABLE_TESTS)
  include(CTest)
  include("cmake/raw/EnableTribitsTestSupport.cmake")
  add_subdirectory(test)
endif()

# Stuff that TriBITS does automatically
include("cmake/raw/DefineAllLibsTarget.cmake")
include("cmake/raw/GeneratePackageConfigFileForBuildDir.cmake")
include("cmake/raw/GeneratePackageConfigFileForInstallDir.cmake")
```

TriBITS vs. Raw CMake TriBITS-Compliant CMake Package

**package1/
  src/CMakeLists.tribits.cmake**

```
tribits_include_directories(
  ${CMAKE_CURRENT_SOURCE_DIR})
tribits_add_library(package1
  HEADERS  Package1.hpp
  SOURCES  Package1.cpp)
tribits_add_executable(package1-prg
  NOEXEPREFIX  NOEXESUFFIX
  SOURCES  Package1_Prg.cpp
  INSTALLABLE )
```

**package1/
  src/CMakeLists.raw.cmake**

```
# Create and install library 'package1'
add_library(Package1_package1 Package1.hpp Package1.cpp)
target_include_directories(Package1_package1
  PUBLIC $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}>)
target_link_libraries(Package1_package1
  PRIVATE tpl1::tpl1 )
set_target_properties(Package1_package1 PROPERTIES
  EXPORT_NAME package1)
add_library(Package1::package1 ALIAS Package1_package1)
install(TARGETS Package1_package1
  EXPORT ${PROJECT_NAME}
  INCLUDES DESTINATION ${CMAKE_INSTALL_INCLUDEDIR} )
install(
  FILES Package1.hpp
  DESTINATION ${CMAKE_INSTALL_INCLUDEDIR} )

# Create and install executable 'package1-prg'
add_executable(package1-prg Package1_Prg.cpp)
target_link_libraries(package1-prg PRIVATE Package1::package1)
install(
  TARGETS package1-prg
  EXPORT ${PROJECT_NAME}
  INCLUDES DESTINATION ${CMAKE_INSTALL_INCLUDEDIR} )
```

Extra Code in Raw CMake TriBITS-Compliant CMake Package

package1/cmake/raw/DefineAllLibsTarget.cmake

```
# Generate the all_libs target(s)
add_library(Package1_all_libs INTERFACE)
set_target_properties(Package1_all_libs
  PROPERTIES EXPORT_NAME all_libs)
target_link_libraries(Package1_all_libs
  INTERFACE Package1_package1)
install(TARGETS Package1_all_libs
  EXPORT ${PROJECT_NAME}
  COMPONENT ${PROJECT_NAME}
  INCLUDES DESTINATION
    ${CMAKE_INSTALL_INCLUDEDIR} )
add_library(Package1::all_libs ALIAS
  Package1_all_libs)
```

package1/cmake/raw/Package1Config.cmake.in

```
set(Tpl1_DIR "@Tpl1_DIR@")
find_package(Tpl1 CONFIG REQUIRED)
include("${CMAKE_CURRENT_LIST_DIR}/Package1ConfigTargets.cmake")
```

package1/cmake/raw/GeneratePackageConfigFileForBuildDir.cmake

```
if (COMMAND tribits_package)
  # Generate Package1Config.cmake file for the build tree (for internal
  # TriBITS-compliant package)
  set(packageBuildDirCMakePackagesDir
    "${${CMAKE_PROJECT_NAME}_BINARY_DIR}/cmake_packages/${PROJECT_NAME}")
  export(EXPORT ${PROJECT_NAME}
    NAMESPACE ${PROJECT_NAME}::
    FILE
      "${packageBuildDirCMakePackagesDir}/${PROJECT_NAME}ConfigTargets.cmake"
    )
  configure_file(
    "${CMAKE_CURRENT_LIST_DIR}/Package1Config.cmake.in"
    "${packageBuildDirCMakePackagesDir}/${PROJECT_NAME}/Package1Config.cmake"
    @ONLY )
endif()
```

Consistent with Modern CMake Advocated in:

“Professional CMake”

by Craig Scott

Extra Code in Raw CMake TriBITS-Compliant CMake Package

package1/cmake/raw/
GeneratePackageConfigFileForInstallDir.cmake

```
# Generate and install the Package1Config.cmake file for the install tree
# (needed for both internal and external TriBITS package)
set(pkgConfigInstallDir "${CMAKE_INSTALL_LIBDIR}/cmake/${PROJECT_NAME}")
install(EXPORT ${PROJECT_NAME}
  DESTINATION "${pkgConfigInstallDir}"
  NAMESPACE ${PROJECT_NAME}::
  FILE ${PROJECT_NAME}ConfigTargets.cmake )
configure_file(
  "${CMAKE_CURRENT_SOURCE_DIR}/cmake/raw/Package1Config.cmake.in"
  "${CMAKE_CURRENT_BINARY_DIR}/CMakeFiles/Package1Config.install.cmake"
  @ONLY )
install(
  FILES "${CMAKE_CURRENT_BINARY_DIR}/CMakeFiles/Package1Config.install.cmake"
  RENAME "Package1Config.cmake"
  DESTINATION "${pkgConfigInstallDir}" )
```

Future Work and Summary

Future TriBITS/Trilinos CMake Modernization Work?

DANGER! Risk of shifting significant complexity from TriBITS to Trilinos packages and Trilinos developers!

TriBITS Publications:

- TriBITS overview SAND technical report (FY24 Q1)
- TriBITS overview journal article (JOSS?)

Address a few lingering issues with updated TriBITS: E.g.:

- Relocatable installations of installed \`<Package>Config.cmake\` files?
- Public/private package dependencies, optional and required intra-external package/TPL dependencies?

Refactor to use CMake features overlapping with TriBITS (see TriBITS #411): E.g.:

- Use the standard CMake FortranCInterface.cmake module to handle Fortran/C name mangling.
- Consider switching to using find_package(MPI) (using standard CMake FindMPI.cmake module).

Refactor to remove TriBITS features and simplify TriBITS (see TriBITS #569): E.g.:

- Switch to explicit library linking (more explicit, avoid over linking)
- Remove support for subpackages (lot of added complexity)

Refactor FindTPL<tplName>.cmake files to use find_package(<ExternalPkg>) and remove support for Legacy TriBITS TPLs

- **This is where the most work lies and the biggest breaks to backward comparibility!**

Refactor downstream CMake projects for changes in how Trilinos packages are installed: E.g.:

- Stop using find_package(Trilinos)! => Instead, use find_package(Kokkos), find_package(Tpetra), …

Summary

- **Modern TriBITS:**
  - Uses Modern CMake internally (strips out a lot of older complex TriBITS code)
  - Allows pre-building/installing Trilinos packages in any subgraph sets desired
  - Allows usage of raw CMake to create TriBITS-compliant internal and external packages
  - Usage of find_package(<ExternalPkg>) to pull in external packages using modern CMake IMPORTED targets.
- **Realized impact so far:**
  - Significant simplifications in the implementation of TriBITS
  - Trilinos can use pre-installed native Kokkos (Updated Spack Trilinos package)
  - (Almost) no breakage in backward compatibility for Trilinos developers or customers
- **Future plans:**
  - TriBITS Publications (FY24 Q1)
  - Address a few lingering issues with updated TriBITS
  - Refactor to use CMake features overlapping with TriBITS (see TriBITS #411)
  - Refactor to remove TriBITS features and simplify TriBITS (see TriBITS #569)
  - Refactor FindTPL<tplName>.cmake files to use find_package(<ExternalPkg>) and remove support for Legacy TriBITS TPLs
  - Refactor downstream CMake projects for changes in how Trilinos packages are installed