TriBITS Developers Guide

Author: Roscoe A. Bartlett (bartlettra@ornl.gov)

Contents

1 Introduction

This document describes the usage of the TriBITS (Tribal Build, Integration, Test System) to develop software projects. An initial overview of TriBITS is provided in the TriBITS Overview document which contains the big picture and provides a high-level road map to what TriBITS provides. This particular document, however, describes the details on how to use the TriBITS system to create a CMake build system for a set of compiled software packages.

TriBITS is a fairly extensive framework that is build on CMake/CTest/CPack/CDash which in of itself is a very extensive system of software and tools. The most

important thing to remember is that software project that use TriBITS are really just CMake projects. TriBITS makes no attent to hide that either from the TriBITS project developers or from the users that need to configure and build the software. Therefore, to make effective usage of TriBITS, one must learn the basics of CMake. In particular, CMake is a Turning complete programming lanauge with local and global variables (with strange scoping rules), macros, functions, targets, commands, and other features. One needs to understand how to define and use variables, macros, functions in CMake. One needs to know how to debug CMakeLists.txt files and CMake code in general (i.e. using `MESSAGE()` print statements). One needs to understand how CMake defines and uses targets for various qualities like libraries, executables, etc. Without this basic understanding of CMake, one will have trouble resolving problems when they might occur.

2 Structure of a TriBITS Project

???

3 TriBITS Global Project Settings

TriBITS defines a number of global project-level settings that can be set by the user and can have their default determined by each individual TriBITS project. If a given TriBITS project does not define its own default, a reasonble default is set by the TriBITS system automatically.

ToDo: Document what parameters influence the entire TriBITS project, what parameters can have project-specific defaults, etc.

4 References

5 Appendix

5.1 History of TriBITS

TriBITS started development in November 2007 as a set of helper macros to provide a CMake build system for a small subset of packages in Trilinos. The initial goal was to just to support a native Windows build (using Visual C++) to compile and install these few Trilinos packages on Windows for usage by another project (the Sandia Titan project which included VTK). At that time, Trilinos was using a highly customized autotools build system. Initially, this CMake system was just a set of macros to streamline creating executables and tests. Some of the conventions started in that early effort (e.g. naming conventions of variables and macros where functions use upper case like old FORTRAN and variables are mixed case) were continued in later efforts and are reflected in the current. Then, stating in early 2008, a more detailed evaluation was performed to see if Trilinos should stitch over to CMake as the default (and soon only)

[SCALE] http://scale.ornl.gov/

supported build and test system (see "Why CMake?" in TriBITS Overview). This lead to the initial implementation of a scale-able package-based architecture (PackageArch) for the Trilinos CMake project in late 2008. This Trilinos CMake PackageArch system evolved over the next few years with development in the system slowing into 2010. This Trilinos CMake build system was then adopted as the build infrastructure for the CASL VERA effort in 2011 where CASL VERA packages were treated as add-on Trilinos packages (see Section ???). Over the next year, there was significant development of the system to support larger multi-repo projects in support of CASL VERA. That lead to the decision to formally generalize the Trilinos CMake PackageArch build system outside of Trilinos and the name TriBITS was formally adopted in November 2011. Work to refactor the Trilinos CMake system into a general reusable stand-alone CMake-based build system started in October 2011 and an initial implementation was complete in December 2011 when it was used for the CASL VERA build system. In early 2012, the ORNL CASL-related projects Denovo and SCALE ([SCALE]) adopted TriBITS as their native development build systems. Shortly after TriBITS was adopted the native build system for the the CASL-related University of Michigan code MPACT. In addition to being used in CASL, all of these codes also had a significant life outside of CASL. Because they used the same TriBITS build system, it proved relatively easy to keep these various codes integrated together in the CASL VERA code meta-build. At the same time, TriBITS well served the independent development teams and non-CASL projects independent from CASL VERA. Since the initial extraction of TriBITS from Trilinos, the TriBITS system was further extended and refined, driven by CASL VERA development and expansion. Independently, an early version of TriBITS from 2012 was adopted by the LiveV projectfootnote{https://github.com/lifev/cmake} which was forked and extended independently.

5.2 TriBITS Macros and Functions

The following subsection give detailed documentation for the macros and functions that make up the core TriBITS system. These are what are used by TriBITS project developers. These are listed in approximately the order they will be encounted in a project or packages CMakeLists.txt or other files.

5.2.1 TRIBITS_DEFINE_REPOSITORY_PACKAGES_DIRS_CLASSIFICATIONS()

Define the set of packages for a given TriBIT repo. This macro is typically called from inside of a PackagesList.cmake file for a given TriBITS repo.

Usage:

```
TRIBITS_DEFINE_REPOSITORY_PACKAGES_DIRS_CLASSIFICATIONS(
  <pkg0>  <pkg0_dir>  <pkg0_classifications>
  ...
  <pkgnm1>  <pkgnm1_dir>  <pkgnm1_classifications>
  )
```

This macro sets up a 2D array of NumPackages by NumColumns listing out the packages for a TriBITS repository. Each row (with 3 entries) specifies a package which contains the three columns:

- **PACKAGE**: The name of the TriBITS package. This name must be unique across all other TriBITS packages in this or any other TriBITS repo that might be combined into a single TriBITS project meta-build. The name should be a valid identifier (e.g. matches the regex `[a-zA-Z_][a-zA-Z0-9_]*`).

- **DIR**: The relative directory for the package. This is relative to the TriBITS repository base directory. Under this directory will be a package-specific 'cmake/' directory with file 'cmake/Dependencies.cmake' and a base-level CMakeLists.txt file. The entire contents of the package including all of the source code and all of the tests should be contained under this directory. The TriBITS testing infrastructure relies on the mapping of changed files to these base directories when deciding what packages are modified and need to be retested (along with downstream packages).

- **CLASSIFICATION**: Gives the testing group PT, ST, EX and the maturity level EP, RS, PG, PM, GRS, GPG, GPM, UM. These are seprated by a coma with no space in between such as "RS,PT" for a "Research Stable", "Primary Tested" package. No spaces are allowed so that CMake treats this a one field in the array. The maturity level can be left off in which case it is assumed to be UM for "Unspecified Maturity".

  NOTE: This macro just sets the varaible ${REPOSITORY_NAME}_PACKAGES_AND_DIRS_ANI in the current scope. The advantages of using this macro instead of directly setting this varible include:

  - Asserts that REPOSITORY_NAME is defined and set
  - Avoids having to hard-code the assumed repository name ${REPOSITORY_NAME}. This provides more flexibility for how other TriBITS project name a given TriBITS repo (i.e. the name of repo subdirs).
  - Avoid mispelling the name of the varible ${REPOSITORY_NAME}_PACKAGES_AND_DIRS_ If you misspell the name of the macro, it is an immediate error in CMake.

5.2.2 TRIBITS_DEFINE_REPOSITORY_TPLS_FINDMODS_CLASSIFICATIONS()

Define the list of TPLs, find modules, and classifications for a given TriBITS repository. This macro is typically called from inside of a TPLsList.cmake fil for a given TriBITS repo.

Usage:

```
TRIBITS_DEFINE_REPOSITORY_TPLS_FINDMODS_CLASSIFICATIONS(
  <tpl0_name>   <tpl0_findmod>  <tpl0_classification>
  ...
  <tplnm1_name>   <tplnm1_findmod>  <tplnm1_classification>
```

This macro sets up a 2D array of NumTPLS by NumColumns listing out the TPLs for a TriBITS repository. Each row (with 3 entries) specifies a package which contains the three columns:

- **TPL**: The name of the TriBITS TPL <TPL_NAME>. This name must be unique across all other TriBITS TPLs in this or any other TriBITS

repo that might be combined into a single TriBITS project meta-build. However, a TPL can be redefined (see below). The name should be a valid identifier (e.g. matches the regex `[a-zA-Z_][a-zA-Z0-9_]*`).

- **FINDMOD**: The relative directory for the find module, usually with the name FindTPL<TPL_NAME>.cmake. This is relative to the repository base directory. If just the base path for the find module is given, ending with "/" (e.g. "cmake/tpls/") then the find module will be assumed to be under that this directory with the standard name (e.g. "cmake/tpls/FindTPL<TPL_NAME>.cmake"). A standard way to write a FindTPL<TPL_NAME>.cmake module is to use the function TRIBITS_TPL_DECLARE_LIBRARIES().

- **CLASSIFICATION**: Gives the testing group PT, ST, EX and the maturity level EP, RS, PG, PM, GRS, GPG, GPM, UM. These are seprated by a coma with no space in between such as "RS,PT" for a "Research Stable", "Primary Tested" package. No spaces are allowed so that CMake treats this a one field in the array. The maturity level can be left off in which case it is assumed to be UM for "Unspecified Maturity".

NOTE: A TPL defined in a upstream repo can listed again, which allows redefining the find module that is used to specificy the TPL. This allows downstream repos to add additional requirements on a given TPL. However, the downstream repo's find module file must find the TPL components that are fully compatible with the upstream's find module.

NOTE: This macro just sets the varaible ${REPOSITORY_NAME}_TPLS_FINDMODS_CLASSIFICA in the current scope. The advantages of using this macro instead of directly setting this varible include:

- Asserts that REPOSITORY_NAME is defined and set

- Avoids having to hard-code the assumed repository name ${REPOSITORY_NAME}. This provides more flexibility for how other TriBITS project name a given TriBITS repo (i.e. the name of repo subdirs).

- Avoid mispelling the name of the varible ${REPOSITORY_NAME}_. If you misspell the name of the macro, it is an immediate error in CMake.

5.2.3 TRIBITS_DEFINE_PACKAGE_DEPENDENCIES()

Define the dependenices for a given TriBITS SE package (i.e. a top-level package or a subpackage).

Usage:

```
TRIBITS_DEFINE_PACKAGE_DEPENDENCIES(
   [LIB_REQUIRED_PACKAGES <pkg1> <pkg2> ...]
   [LIB_OPTIONAL_PACKAGES <pkg1> <pkg2> ...]
   [TEST_REQUIRED_PACKAGES <pkg1> <pkg2> ...]
   [TEST_OPTIONAL_PACKAGES <pkg1> <pkg2> ...]
   [LIB_REQUIRED_TPLS <tpl1> <tpl2> ...]
   [LIB_OPTIONAL_TPLS <tpl1> <tpl2> ...]
   [TEST_REQUIRED_TPLS <tpl1> <tpl2> ...]
   [TEST_OPTIONAL_TPLS <tpl1> <tpl2> ...]
   [REGRESSION_EMAIL_LIST  <regression-email-address>
```

```
[SUBPACKAGES_DIRS_CLASSIFICATIONS_OPTREQS
  <spkg1_name>  <spkg1_dir>  <spkg1_classifications>  <spkg1_optreq>
  <spkg2_name>  <spkg2_dir>  <spkg2_classifications>  <spkg2_optreq>
  ...
  ]
)
```

Every argument in this macro is optional. The arguments that apply a package itself are:

- **LIB_REQUIRED_PACKAGES:** List of upstream packages that must be enabled in order to build and use the libraries (or capabilities) in this package.
- **LIB_OPTIONAL_PACKAGES:** List of additional optional upstream packages that can be used in this package if enabled. These upstream packages need not be enabled in order to use this package but not enabling one or more of these optional upstream packages will result in diminished capabilities of this package.
- **TEST_REQUIRED_PACKAGES:** List of additional upstream packages that must be enabled in order to build and/or run the tests and/or examples in this packages. If any of these upstream packages is not enabled, then there will be no tests or examples defined or run for this package.
- **TEST_OPTIONAL_PACKAGES:** List of additional optional upstream packages that can be used by the tests in this package. These upstream packages need not be enabled in order to run basic tests for this package. Typically, extra tests that depend on optional test packages involve integration testing of some type.
- **LIB_REQUIRED_TPLS:** List of upstream TPLs that must be enabled in order to build and use the libraries (or capabilities) in this package.
- **LIB_OPTIONAL_TPLS:** List of additional optional upstream TPLs that can be used in this package if enabled. These upstream TPLs need not be enabled in order to use this package but not enabling one or more of these optional upstream TPLs will result in diminished capabilities of this package.
- **TEST_REQUIRED_TPLS:** List of additional upstream TPLs that must be enabled in order to build and/or run the tests and/or examples in this packages. If any of these upstream TPLs is not enabled, then there will be no tests or examples defined or run for this package.
- **TEST_OPTIONAL_TPLS:** List of additional optional upstream TPLs that can be used by the tests in this package. These upstream TPLs need not be enabled in order to run basic tests for this package. Typically, extra tests that depend on optional test TPLs involve integration testing of some type.

Only direct package dependenices need to be listed. Indirect package dependencies are automatically handled. For example, if this SE package directly depends on PKG2 which depends on PKG1 (but this SE package does not directly depend on anything in PKG1) then this package only needs to list a dependency on PKG2, not PKG1. The dependnecy on PKG1 will be taken care of automatically by the TriBITS dependency tracking system.

However, currently, all TPL dependendies must be listed, even the indirect ones. This is a requirement that will be dropped in the future.

The packages listed in LIB_REQUIRED_PACKAGES are implicitly also dependenices in TEST_REQUIRED_PACKAGES. Likewise LIB_OPTIONAL_PACKAGES are implicitly also dependenices in TEST_OPTIONAL_PACKAGES. Same goes for TPL dependencies.

The dependencies within a single list do not need to be listed in any order. For example if PKG2 depends on PKG1, and this given SE package depends on both, one can list "LIB_REQUIRED_PACKAGES PKG2 PKG1" or "LIB_REQUIRED_PACKAGES PKG1 PKG2". Likewise the listing of TPLs order is not important.

If some upstream packages are allowed to be missing, this can be specified by calling the macro TRIBITS_ALLOW_MISSING_EXTERNAL_PACKAGES().

A top-level package can also have subpackages. In this case, the following varible must be set:

- **SUBPACKAGES_DIRS_CLASSIFICATIONS_OPTREQS:** 2D array with rows listing the subpackages and the columns:
  - **SUBPACKAGE:** The name of the subpackage <spkg_name>. The full SE package name is "${PARENT_PACKAGE_NAME}<spkg_name>". The full SE package name is what is used in listing dependenices in other SE packages.
  - **DIRS:** The subdirectory <spkg_dir> relative to the parent package's base directory. All of the contents of the subpackage should be under this subdirectory. This is assumed by the TriBITS testing support software when mapping modified files to SE packages that need to be tested.
  - **CLASSIFICATIONS***: The test group PT, ST, EX and the maturity level EP, RS, PG, PM, GRS, GPG, GPM, and UM, separated by a coma ',' with no spaces in between (e.g. "PT,GPM"). These have exactly the name meaning as for full packages (see TRIBITS_DEFINE_REPOSITORY_PACKAGE
  - **OPTREQ:** Determines if the outer parent package has an OPTIONAL or REQUIRED dependence on this subpackage.

Other variables that this macro handles:

- **REGRESSION_EMAIL_LIST:** The email list that is used to send CDash error messages. If this is missing, then the email list that CDash errors go to is determined by other means (see ???).

NOTE: All this macro really does is to just define the variables:

- LIB_REQUIRED_DEP_PACKAGES
- LIB_OPTIONAL_DEP_PACKAGES

- TEST_REQUIRED_DEP_PACKAGES
- TEST_OPTIONAL_DEP_PACKAGES
- LIB_REQUIRED_DEP_TPLS
- LIB_OPTIONAL_DEP_TPLS
- TEST_REQUIRED_DEP_TPLS
- TEST_OPTIONAL_DEP_TPLS
- REGRESSION_EMAIL_LIST
- SUBPACKAGES_DIRS_CLASSIFICATIONS_OPTREQS

which are then read by the TriBITS cmake code to build the package dependency graph. The advantage of using this macro instead of just directly setting the varibles is that you only need to list the dependencies you have. Otherwise, you need to set all of these varibles, even those that are empty. This is a error checking property of the TriBITS system to avoid misspelling the names of these variables.

5.2.4 TRIBITS_ALLOW_MISSING_EXTERNAL_PACKAGES()

Macro used in Dependencies.cmake files to allow some upstream dependent packages to be missing.

Usage:

```
TRIBITS_ALLOW_MISSING_EXTERNAL_PACKAGES(<pack_1> <pack_2> ...)
```

If the missing upstream SE package <pack_i> is optional, then the effect will be to simply ignore the missing package and remove it from the dependency list. However, if the missing upstream SE package <pack_i> is required, then in addition to ignoring the missing package, the current SE (sub)package will also ee hard disabled, i.e. ${PROJECT_NAME}_ENABLE_{CURRENT_PACKAGE}=OFF.

This function is typically used in packages in external TriBITS repos that are depend on other packages in other exteral TriBITS repos that might be missing.

NOTE: Using this function effectively turns off error checking for misspelled package names so it is important to only use it when it absolutely is needed.

5.2.5 TRIBITS_TPL_DECLARE_LIBRARIES()

Function that sets up cache variables for users to specify where to find a TPL's headers and libraries. This function is typically called inside of a file `FindTPL<tpl_name>.cmake` file.

Usage:

```
TRIBITS_TPL_DECLARE_LIBRARIES(
  <tpl_name>
  [REQUIRED_HEADERS <header1> <header2> ...]
  [MUST_FIND_ALL_HEADERS]
  [REQUIRED_LIBS_NAMES <libname1> <libname2> ...]
```

```
[MUST_FIND_ALL_LIBS]
[NO_PRINT_ENABLE_SUCCESS_FAIL]
)
```

This function can set up a with header files and/or libraries.
The input arguments to this function are:

- `<tpl_name>`: Name of the TPL that is listed in a TPLsList.cmake file. Below, this is referted to as the local CMake variable `TPL_NAME`.
- `REQUIRED_HEADERS`: List of header files that are searched for the TPL using `FIND_PATH()`.
- `MUST_FIND_ALL_HEADERS`: If set, then all of the header files listed in REQUIRED_HEADERS must be found in order for TPL_${TPL_NAME}_INCLUDE_DIRS to be defined.
- `REQUIRED_LIBS_NAMES`: List of libraries that are searched for when looked for the TPLs libraries with FIND_LIBRARY(...).
- `MUST_FIND_ALL_LIBS`: If set, then all of the library files listed in REQUIRED_LIBS_NAMES must be found or the TPL is considered not found!
- `NO_PRINT_ENABLE_SUCCESS_FAIL`**: If set, then the final success/fail** will not be printed

The following cache variables, if set, will be used by that this function:

- `${TPL_NAME}_INCLUDE_DIRS:PATH`: List of paths to search first for header files defined in `REQUIRED_HEADERS`.
- `${TPL_NAME}_INCLUDE_NAMES:STIRNG`: List of include names to be looked for instead of what is specified in REQUIRED_HEADERS.
- `${TPL_NAME}_LIBRARY_DIRS:PATH`: The list of directories to search first for libraies defined in REQUIRED_LIBS_NAMES.
- `${TPL_NAME}_LIBRARY_NAMES:STIRNG`: List of library names to be looked for instead of what is specified in REQUIRED_LIBS_NAMES.

This function sets global varibles to return state so it can be called from anywhere in the call stack. The following cache variables defined that are intended for the user to set and/or use:

- `TPL_${TPL_NAME}_INCLUDE_DIRS`: A list of common-separated full directory paths that contain the TPLs headers. If this varible is set before calling this function, then no headers are searched for and this variable will be assumed to have the correct list of header paths.
- `TPL_${TPL_NAME}_LIBRARIES`: A list of commons-seprated full library names (output from FIND_LIBRARY(...)) for all of the libraries found for the TPL. IF this varible is set before calling this function, no libraries are searched for and this varaible will be assumed to have the correct list of libraries to link to.

5.2.6 TRIBITS_PACKAGE()

Macro called at the very beginning of a ${PROJECT_NAME} package's top-level CMakeLists.txt file.

Usage:

```
TRIBITS_PACKAGE(
  <packageName>
  [ENABLE_SHADOWING_WARNINGS]
  [DISABLE_STRONG_WARNINGS]
  [CLEANED]
  [DISABLE_CIRCULAR_REF_DETECTION_FAILURE]
  )
```

See TRIBITS_PACKAGE_DECL() for the documentation for the arguments and TRIBITS_PACKAGE_DECL() and TRIBITS_PACKAGE() for a description of the arguments and the side-effects (and varibles set) of calling this macro.

5.2.7 TRIBITS_PACKAGE_DECL()

Macro called at the very beginning of a ${PROJECT_NAME} package's top-level CMakeLists.txt file when a packages has subpackages.

If the package does not have subpackages, just call TRIBITS_PACKAGE() which calls this macro.

Usage:

```
TRIBITS_PACKAGE_DECL(
  <packageName>
  [ENABLE_SHADOWING_WARNINGS]
  [DISABLE_STRONG_WARNINGS]
  [CLEANED]
  [DISABLE_CIRCULAR_REF_DETECTION_FAILURE]
  )
```

The arguments are:

- `<packageName>`: Gives the name of the Package, mostly just for checking and documentation purposes. This much match the name of the package provided in the PackagesLists.cmake or it is an error.
- `ENABLE_SHADOWING_WARNINGS`:If specified, then shadowing warnings will be turned on for supported platforms/compilers. The default is for shadowing warnings to be turned off. Note that this can be overridden globally by setting the cache variable ${PROJECT_NAME}_ENABLE_SHADOWING_WARNINGS.
- `DISABLE_STRONG_WARNINGS`: If specified, then all strong warnings will be turned off, if they are not already turned off by global cache variables. Strong warnings are turned on by default in development mode.
- `CLEANED`:If specified, then warnings will be promoted to errors for all defined warnings.

- DISABLE_CIRCULAR_REF_DETECTION_FAILURE: If specified, then the standard grep looking for RCPNode circular references that causes tests to fail will be disabled. Note that if these warnings are being produced then it means that the test is leaking memory and user like may also be leaking memory.

There are several side-effects of calling this macro:

- The package's list of targets varibles aree initialized to emtpy.
- The local varibles PACKAGE_SOURCE_DIR and PACKAGE_BINARY_DIR are set for this package's use in its CMakeLists.txt files.
- Package-specific compiler options are set up in package-scoped (i.e., the package's subdir and its subdirs) in CMAKE_<LANG>_FLAG.
- This packages's cmake subdir is added to CMAKE_MODULE_PATH so that the package's try-compile modules can be read in with just a raw INCLUDE() leaving off the full path and the *.cmake extension.

5.2.8 TRIBITS_PACKAGE_DEF()

Macro called after subpackages are processed in order to handle the libraries, tests, and examples of the final package.

Usage:

```
TRIBITS_PACKAGE_DEF()
```

If the package does not have subpackages, just call TRIBITS_PACKAGE() which calls this macro.

This macro has several side effects:

- The varible PACKAGE_NAME is set in the local scope for usage by the package's CMakeLists.txt files.
- The intra-package dependency varibles (i.e. list of include directoires, list of libraries, etc.) are initialized to emtpy.

5.2.9 TRIBITS_PROCESS_SUBPACKAGES()

Macro that processes subpackages for packages that have them. This is called in the parent packages top-level CMakeLists.txt file.

Usage:

```
TRIBITS_PROCESS_SUBPACKAGES()
```

Must be called after TRIBITS_PACKAGE_DECL() but before TRIBITS_PACKAGE_DEF().

5.2.10 TRIBITS_ADD_TEST_DIRECTORIES()

Macro called to add a set of test directories for an SE package.

Usage:

```
TRIBITS_ADD_TEST_DIRECTORIES(<dir1> <dir2> ...)
```

This macro only needs to be called from the top most CMakeList.txt file for which all subdirectories are all "tests".

This macro can be called several times within a package and it will have the right effect.

Currently, really all it does macro does is to call `ADD_SUBDIRECTORY(<diri>)` if `${PACKAGE_NAME}_ENABLE_TESTS` or `${PARENT_PACKAGE_NAME}_ENABLE_TESTS` are true. However, this macro may be extended in the futgure in order to modify behavior related to adding tests and examples in a uniform way..

5.2.11 TRIBITS_ADD_EXAMPLE_DIRECTORIES()

Macro called to conditionally add a set of example directories for an SE package.

Usage:

```
TRIBITS_ADD_EXAMPLE_DIRECTORIES(<dir1> <dir2> ...)
```

This macro only needs to be called from the top most CMakeList.txt file for which all subdirectories are all "examples".

This macro can be called several times within a package and it will have the right effect.

Currently, really all it does macro does is to call `ADD_SUBDIRECTORY(<diri>)` if `${PACKAGE_NAME}_ENABLE_EXAMPLES` or `${PARENT_PACKAGE_NAME}_ENABLE_EXAMPLES` are true. However, this macro may be extended in the futgure in order to modify behavior related to adding tests and examples in a uniform way..

5.2.12 TRIBITS_SET_ST_FOR_DEV_MODE()

Function that allows packages to easily make a feature `ST` for development builds and `PT` for release builds by default.

Usage:

```
TRIBITS_SET_ST_FOR_DEV_MODE(<outputVar>)
```

`${<outputVar>}` is set to `ON` or `OFF` based on the configure state. In development mode it will be set to `ON` only if `ST` code is enabled, otherwise it is set to `OFF`. In release mode it is always set to `ON`. This allows some sections of a TriBITS package to be considered `ST` for development mode reducing testing time which includes only `PT` code., while still having important functionality available to users by default in a release.

5.2.13 TRIBITS_PACKAGE_POSTPROCESS()

Macro called at the very end of a package's top-level CMakeLists.txt file. This macro performs some critical post-processing activities before downstream packages are processed.

Usage:

```
TRIBITS_PACKAGE_POSTPROCESS()
```

NOTE: It is unfortunate that a packages's CMakeLists.txt file must call this macro but limitations of the CMake language make it necessary to do so.