bcm Documentation

Release 0.1

Paul Fultz II

Sep 27, 2017

Contents

Paul Fultz II

CHAPTER 1

Motivation

This provides cmake modules that can be re-used by boost and other dependencies. It provides modules to reduce the boilerplate for installing, versioning, setting up package config, and creating tests.

CHAPTER 2

Usage

The modules can be installed using standard cmake install:

```
mkdir build
cd build
cmake ..
cmake --build . --target install
```

Once installed, the modules can be used by using `find_package` and then including the appropriate module:

```
find_package(BCM)
include(BCMPackage)
```

CHAPTER 3

Modules

BCMInstallTargets

bcm_install_targets

This installs the targets specified. The directories will be installed according to GNUInstallDirs. It will also install a corresponding cmake package config(which can be found with `find_package`) to link against the library targets.

TARGETS `<target-name>...`

The name of the targets to install.

INCLUDE `<directory>...`

Include directories to be installed. It also makes the include directory available for targets to be installed.

EXPORT

This specifies an export file. By default, the export file will be named `${PROJECT_NAME}-targets`.

BCMPackage

bcm_find_package

This works the same as cmake's `find_package` except in addition, it will keep track of the each call to `bcm_find_package` in the project. Functions like `bcm_package` and `bcm_boost_package` will include these dependencies automatically in cmake's package config that gets generated.

bcm_package

This setups a non-boost package in cmake. This is similar to `bcm_boost_package` but with extra flexibility as it does not have boost specific conventions. This function creates a library target that will be installed and exported. In

addition, the target will be setup to auto-link for the tests.

<package-name>

The name of the package. This is both the name of the library target and the cmake package config that can be used with `find_package(<package-name>)`.

VERSION `<version>`

Sets the version of the package.

VERSION_HEADER `<header>`

This will parse the version from a header file. It will parse the macros defined in the header as `<prefix>_<package-name>_VERSION_MAJOR`, `<prefix>_<package-name>_VERSION_MINOR`, and `<prefix>_<package-name>_VERSION_PATCH`. The `prefix` by default is the package name in all caps, but can be set using the `VERSION_PREFIX` option.

VERSION_PREFIX `<prefix>`

This sets the prefix that macros used to define the version will use.

SOURCES `<source-files>...`

The source files to build the library.

INCLUDE `<directory>...`

This sets the include directories for the package. Each include directory will be installed.

NAMESPACE `<namespace>`

This is the namespace to be added to the exported targets.

bcm_boost_package

This setups a boost package in cmake. It creates a library target that will be installed. The target will be exported and it will be in the `boost::` namespace. In addition, the target will be setup to auto-link for the tests.

<package-name>

The name of the boost package. The corresponding cmake package config can be used with `find_package(boost_<package-name>)`.

VERSION `<version>`

Sets the version of the package.

VERSION_HEADER `<header>`

This will parse the version from a header file. It will parse the macros defined in the header as `BOOST_<package-name>_VERSION_MAJOR`, `BOOST_<package-name>_VERSION_MINOR`, and `BOOST_<package-name>_VERSION_PATCH`.

SOURCES `<source-files>...`

The source files to build the library.

DEPENDS `<boost-dependencies>...`

This specifies internal boost dependencies, that is, dependencies on other boost libraries. The libraries should not be prefixed with `boost_` nor `boost::`.

BCMPackageConfigHelpers

This provides helpers functions for creating config files that can be included by other projects to find and use a package.

bcm_configure_package_config_file

This works similiar to `configure_package_config_file` from cmake, except it provides in addition a `find_dependency` function.

<input> <output>

The `<input>` and `<output>` arguments are the input and output file, the same way as in `configure_file()`.

NO_SET_AND_CHECK_MACRO

Don't genereate the `set_and_check` macro.

NO_CHECK_REQUIRED_COMPONENTS_MACRO

Don't generate `_CHECK_REQUIRED_COMPONENTS` macro.

INSTALL_DESTINATION `<path>`

The `<path>` given to `INSTALL_DESTINATION` must be the destination where the cmake config file will be installed to.

PATH_VARS `<var>...`

The variables `<var>...` given as `PATH_VARS` are the variables which contain install destinations. For each of them the macro will create a helper variable `PACKAGE_<var...>`. These helper variables can be used in the cmake config file for setting the installed location. They are calculated by `bcm_configure_package_config_file` so that they are always relative to the installed location of the package. This works both for relative and also for absolute locations.

bcm_auto_export

This generates a simple cmake config file that includes the exported targets.

EXPORT

This specifies an export file. By default, the export file will be named `${PROJECT_NAME}-targets`.

DEPENDS `PACKAGE <package-name>...`

This will search for these dependent packages in the cmake package config that is generated.

NAMESPACE `<namespace>`

This is the namespace to add to the targets that are exported.

COMPATIBILITY `<compatibility>`

This uses the version compatibility specified by cmake version config.

NAME `<name>`

This is the name to use for the package config file. By default, this uses the project name, but this parameter can override it.

TARGETS `<target>...`

The generated config will set `<package>_LIBRARIES` to the list of targets listed.

BCMPkgConfig

bcm_generate_pkgconfig_file

This will generate a simple pkgconfig file.

NAME <name>

This is the name of the pkgconfig module.

LIB_DIR <directory>

This is the directory where the library is linked to. This defaults to `${CMAKE_INSTALL_LIBDIR}`.

INCLUDE_DIR <directory>

This is the include directory where the header file are installed. This defaults to `${CMAKE_INSTALL_INCLUDEDIR}`.

DESCRIPTION <text>

A description about the library.

TARGETS <targets>...

The library targets to link.

CFLAGS <flags>...

Additionaly, compiler flags.

LIBS <library flags>...

Additional libraries to be linked.

REQUIRES <packages>...

List of other pkgconfig packages that this module depends on.

bcm_auto_pkgconfig

This will auto generate pkgconfig from a given target. All the compiler and linker flags come from the target.

<target>

The first parameter is target that will be used to generate the pkgconfig file.

NAME <name>

This is the name of the pkgconfig module. By default, this will use the project name.

REQUIRES <packages>...

List of other pkgconfig packages that this module depends on.

BCMProperties

This module defines several properties that can be used to control language features in C++.

CXX_EXCEPTIONS

This property can be used to enable or disable C++ exceptions. This can be applied at global, directory or target scope. At global scope this defaults to On.

CXX_RTTI

This property can be used to enable or disable C++ runtime type information. This can be applied at global, directory or target scope. At global scope this defaults to On.

CXX_STATIC_RUNTIME

This property can be used to enable or disable linking against the static C++ runtime. This can be applied at global, directory or target scope. At global scope this defaults to Off.

BCMSetupVersion

bcm_setup_version

This sets up the project version by setting these version variables:

```
PROJECT_VERSION, ${PROJECT_NAME}_VERSION
PROJECT_VERSION_MAJOR, ${PROJECT_NAME}_VERSION_MAJOR
PROJECT_VERSION_MINOR, ${PROJECT_NAME}_VERSION_MINOR
PROJECT_VERSION_PATCH, ${PROJECT_NAME}_VERSION_PATCH
```

VERSION <major>.<minor>.<patch>

This is the version to be set.

GENERATE_HEADER <header-name>

This is a header which will be generated with defines for the version number.

PREFIX <identifier>

By default, the upper case of the project name is used as a prefix for the version macros that are defined in the generated header: ${PREFIX}_VERSION_MAJOR, ${PREFIX}_VERSION_MINOR, ${PREFIX}_VERSION_PATCH, and ${PREFIX}_VERSION. The PREFIX option allows overriding the prefix name used for the macros.

PARSE_HEADER <header-name>

Rather than set a version and generate a header, this will parse a header with macros that define the version, and then use those values to set the version for the project.

BCMTest

bcm_mark_as_test

This marks the target as a test, so it will be built with the tests target. If BUILD_TESTING is set to off then the target will not be built as part of the all target.

bcm_test_link_libraries

This sets libraries that the tests will link against by default.

bcm_test

This setups a test. By default, a test will be built and executed.

SOURCES `<source-files>...`

Source files to be compiled for the test.

CONTENT `<content>`

This a string that will be used to create a test to be compiled and/or ran.

NAME `<name>`

Name of the test.

COMPILE_ONLY

This just compiles the test instead of running it. As such, a `main` function is not required.

WILL_FAIL

Specifies that the test will fail.

NO_TEST_LIBS

This won't link in the libraries specified by `bcm_test_link_libraries`

bcm_test_header

This creates a test to test the include of a header.

NAME `<name>`

Name of the test.

HEADER `<header-file>`

The header to include.

STATIC

Rather than just test the include, using `STATIC` option will test the include across translation units. This helps check for incorrect include guards and duplicate symbols.

NO_TEST_LIBS

This won't link in the libraries specified by `bcm_test_link_libraries`

Index