ROCm CMake Build Tools

Release 0.14.0

Advanced Micro Devices, Inc.

Mar 27, 2026

INSTALLATION

ROCm CMake build tools (also known as "rocm-cmake") is a collection of functions that unify and simplify the CMake code of ROCm components, as well as ensuring consistency across these different components. ROCm CMake build tools are primarily used when building a library, and as such are not runtime dependencies for any generated libraries, packages, or executables.

The build tools can be included into a CMake project by running:

```
find_package(ROCmCMakeBuildTools)
# or
find_package(ROCM) # deprecated, but included for backwards compatibility
```

Once the tools have been included in this manner, individual files may be included by running `include(<file_name>)`. The file names, the functions, variables, and macros accessible using each file are described in this documentation.

You can access the build tools on the ROCm CMake GitHub repository. For a complete listing of the features of ROCm CMake refer to contents.

To contribute to the documentation, refer to Contributing to ROCm.

You can find licensing information on the Licensing page.

CHAPTER

ONE

ROCMINSTALLTARGETS

1.1 Commands

rocm_install

```
rocm_install(TARGETS <target>... [<arg>...])
rocm_install(<FILES | PROGRAMS> <file>... [<arg>...])
rocm_install(DIRECTORY <dir>... [<arg>...])
rocm_install(SCRIPT <file> [<arg>...])
rocm_install(CODE <code> [<arg>...])
rocm_install(EXPORT <export-name> [<arg>...])
```

Wraps installers to install to the correct component (devel or runtime) unless COMPONENT is specified. The TARGETS signature wraps `rocm_install_targets`, all other signatures wrap `install`.

rocm_install_targets

```
rocm_install_targets(
    TARGETS <target>...
    [PREFIX <path>]
    [EXPORT <export-file>]
    [INCLUDE <directory>...]
    [COMPONENT <component>]
)
```

Install targets into the appropriate directory. Unless COMPONENT is specified, libraries will be installed to the base package and namelinked in the `devel` package, and everything else will be installed to the `devel` package.

rocm_export_targets

```
rocm_export_targets(
    [NAMESPACE <namespace>]
    [EXPORT <export>]
    [INCLUDE <cmake-file>...]
    [NAME <name>]
    [COMPATIBILITY <compatibility>]
    [PREFIX <prefix>]
    [TARGETS <targets>...]
    [DEPENDS [PACKAGE <package-name>]...]
    [STATIC_DEPENDS [PACKAGE <package-name>]...]
)
```

Export the installed targets so they can be consumed with `find_package`.

CHAPTER

TWO

ROCMINSTALLSYMLINKS

2.1 Commands

rocm_install_symlink_subdir

```
rocm_install_symlink_subdir(<subdir> [<component>])
```

Install symlinks which point into the `subdir` directory to the component *component*, or to the runtime and development components otherwise.

CHAPTER

THREE

ROCMHEADERWRAPPER

3.1 Commands

rocm_wrap_header_file

```
rocm_wrap_header_file(
    [HEADERS] <header-file> [<header-file>...]
    [HEADER_LOCATION <header-location>]
    [INCLUDE_LOCATION <include-location>]
    [GUARDS <guard>...]
    [WRAPPER_LOCATIONS <wrapper-location>...]
    [OUTPUT_LOCATIONS <output-location>...]
    [ORIGINAL_FILES <original-file>...]
)
```

Create a C/C++ wrapper file for each specified header file. The wrapper is simply a C/C++ header that emits a deprecation warning before including its corresponding header. The warning can be turned off by defining `ROCM_NO_WRAPPER_HEADER_WARNING` when using the wrapper header files. There is an additional configure-time CMake variable `ROCM_HEADER_WRAPPER_WERROR`, which is used to set the default value for the compile-time C macro of the same name (with CMake truthy values setting a default of true/1). If the compile-time macro is true, then deprecation errors will be emitted instead of warnings.

Any relative header or wrapper locations are relative to `${CPACK_PACKAGING_INSTALL_PREFIX}` if it is set, or to `${CMAKE_INSTALL_PREFIX}` otherwise (i.e. the install directory). Any relative output locations are relative to `${PROJECT_BINARY_DIR}` (i.e. the build directory)

Each <header-file> is presumed to be installed to `${CMAKE_INSTALL_PREFIX}/`<header-location>`/`<header-file>. If it is not specified, <header-location> defaults to `include/${CMAKE_PROJECT_NAME}` (e.g. `include/rocblas`).

The <include-location> parameter specifies the presumed compiler include directory, to correctly calculate suggested include directives.

3.1.1 Guards

A guard item consists of the guard string <guard>, a wrapper location <wrapper-location>, and an output location <output-location>. You may specify any number of guard strings, wrapper locations and output locations. Guard items will be created from the arguments, and if necessary the following defaults will be used: * Default <guard>: `WRAPPER` * Default <wrapper-location>: `${CMAKE_PROJECT_NAME}/include` * Default <output-location>: The associated <wrapper-location> If no guard items are specified, one will be created using all of the default values. Each guard item will create a wrapper file at `${PROJECT_BINARY_DIR}/`<output-location>`/`<header-file> for each header file. This wrapper file will have the include guard `ROCM_`<guard>`_`<item-path>, where <item-path> is <header-file> with each `/` and `.` replaced with `_`. It assumes that it will be installed to

${CMAKE_INSTALL_PREFIX}/<wrapper-location>/<header-file>, and will include a relative path that is correct if it is installed to that location.

For example, suppose the project name is rocexample and consider the following:

```
rocm_wrap_header_file(
    foo/bar.h
    HEADER_LOCATION include/rocexample
    GUARDS
        EXAMPLE
        EXAMPLE_INC
    WRAPPER_LOCATIONS
        rocexample # EXAMPLE
    OUTPUT_LOCATIONS
        wrapper/rocexample # EXAMPLE
)
```

This will create two wrapper files.

The first wrapper file will be created at ${PROJECT_BINARY_DIR}/wrapper/rocexample/foo/bar.h. Its include guard will be ROCM_EXAMPLE_FOO_BAR_H, and it will include the file ../../include/rocexample/foo/bar.h (which is the correct file when this wrapper is installed at ${CMAKE_INSTALL_PREFIX}/rocexample/foo/bar.h).

The second wrapper file uses the default locations, so it will be created at ${PROJECT_BINARY_DIR}/rocexample/include/foo/bar.h. Its include guard will be ROCM_EXAMPLE_INC_FOO_BAR_H, and it will include the file ../../../include/rocexample/foo/bar.h (which is the correct file when this wrapper is installed at ${CMAKE_INSTALL_PREFIX}/rocexample/include/foo/bar.h)

If the name of the wrapper file being generated is the same as the name of any <original-file>, the contents of that <original-file> will be added to the wrapper file inside a #if 0 block. This has no effect on the code of the header, but it does allow projects which search for specific strings inside a header file to function correctly.

rocm_wrap_header_dir

```
rocm_wrap_header_dir(
    <include-directory>
    [HEADER_LOCATION <header-location>]
    [GUARDS <guard>...]
    [WRAPPER_LOCATIONS <wrapper-location>...]
    [OUTPUT_LOCATIONS <output-location>...]
    [PATTERNS <pattern>...]
    [ORIGINAL_FILES <original-file>...]
)
```

Create a C/C++ wrapper file for each header file in the given directory (or any subdirectory) matching at least one pattern.

Each file in the specified directory which matches a pattern will have a wrapper file created for it. The <header-file> used in each call to rocm_wrap_header_file is the path to the header file relative to <include-directory>.

CHAPTER

FOUR

ROCMCREATEPACKAGE

4.1 Commands

rocm_create_package

```
rocm_create_package(
    NAME <name>
    [DESCRIPTION <description>]
    [SECTION <section>]
    [MAINTAINER <maintainer>]
    [LDCONFIG_DIR <lib-directory>]
    [PREFIX <path>]
    [LDCONFIG]
    [HEADER_ONLY]
)
```

Sets up CPack packaging, including installing the license file to the correct location and component. If the license file is not specified, also attempts to locate a LICENSE, LICENSE.md, or LICENSE.txt file in *CMAKE_SOURCE_DIR*.

If the CPACK_GENERATOR has not been specified then if ROCM_PKGTYPE is set then use that to set it, otherwise probe the system to see what programs are available.

rocm_package_add_rpm_dependencies

```
rocm_package_add_rpm_dependencies(
    [QUIET]
    [COMPONENT <component>]
    [DEPENDS <dependencies>...]
    [SHARED_DEPENDS <dependencies>...]
    [STATIC_DEPENDS <dependencies>...]
)
```

Sets up the RPM package for a component to depend on some other packages, possibly versioned. The dependencies should be listed in RPM format (e.g. *foo > 0.3* or *bar = 1.0*). Will warn if the package has already been created, unless QUIET is specified.

rocm_package_add_deb_dependencies

```
rocm_package_add_deb_dependencies(
    [QUIET]
    [COMPONENT <component>]
    [DEPENDS <dependencies>...]
```

(continues on next page)

(continued from previous page)

```
    [SHARED_DEPENDS <dependencies>...]
    [STATIC_DEPENDS <dependencies>...]
)
```

Sets up the deb package for a component to depend on some other packages, possibly versioned. The dependencies may be listed in RPM format (e.g. *foo > 0.3* or *bar = 1.0*), and will automatically be converted to the DEB formatting. Will warn if the package has already been created, unless QUIET is specified.

rocm_package_add_dependencies

```
rocm_package_add_dependencies(
    [QUIET]
    [COMPONENT <component>]
    [DEPENDS <dependencies>...]
    [SHARED_DEPENDS <dependencies>...]
    [STATIC_DEPENDS <dependencies>...]
)
```

A convenience function wrapping both `rocm_package_add_rpm_dependencies` and `rocm_package_add_deb_dependencies`. The dependencies must be listed in RPM format (e.g. `foo > 0.3` or `bar = 1.0`), and will automatically be converted to the DEB formatting. Will warn if the package has already been created, unless QUIET is specified.

rocm_package_setup_component

```
rocm_package_setup_component(
    <component-name>
    [PACKAGE_NAME <package-name>]
    [LIBRARY_NAME <library-name>]
    [PARENT <parent>]
    [DEPENDS
        [COMMON <common-dependencies>...]
        [RPM <rpm-dependencies>...]
        [DEB <deb-dependencies>...]
        [COMPONENT <component-dependencies>...]
    ]
)
```

Set up a component for packaging. Sets the package name, and adds dependencies as appropriate. Component dependencies are where the package generated by component A depends on the package generated by component B. In order to ensure that these dependencies correctly use the generated package names, these dependencies are not fully added until the packages are generated.

CHAPTER

FIVE

ROCMCLIENTS

5.1 Commands

rocm_package_setup_client_component

```
rocm_package_setup_client_component(
    <component>
    [PACKAGE_NAME <package-name>]
    [LIBRARY_NAME <library-name>]
    [DEPENDS
        [COMMON <common-dependencies>...]
        [RPM <rpm-dependencies>...]
        [DEB <deb-dependencies>...]
        [COMPONENT <component-dependencies>...]
    ]
)
```

Setup a client component for packaging. See `rocm_package_setup_component`.

CHAPTER

SIX

ROCMPACKAGECONFIGHELPERS

6.1 Commands

rocm_configure_package_config_file

```
rocm_configure_package_config_file(<input> <output>
    INSTALL_DESTINATION <path>
    [PATH_VARS <var>...]
    [PREFIX <path>]
)
```

Configure the config file used by `find_package`.

CHAPTER

SEVEN

ROCMCHECKTARGETIDS

7.1 Commands

rocm_check_target_ids

```
rocm_check_target_ids(<output-variable>
    TARGETS <target-id>...
)
```

Returns the subset of HIP target-ids supported by the current CXX compiler.

CHAPTER

EIGHT

ROCMSETUPVERSION

8.1 Commands

rocm_get_version

```
rocm_get_version(<output-variable>
    [VERSION <version>]
    [DIRECTORY <path>]
)
```

Get the version of directory using git tags if possible.

rocm_setup_version

```
rocm_setup_version(
    VERSION <version>
    [NO_GIT_TAG_VERSION]
    [PARENT <commit>]
)
```

Setup the version for the project. This will try to use git tag to set the version if possible unless `NO_GIT_TAG_VERSION` is passed. The `PARENT` argument can be used to set the commit to start the count of number of commits to the current revision.

CHAPTER

NINE

ROCMANALYZERS

This creates an `analyze` target which can run all analysis for a project.

9.1 Commands

rocm_mark_as_analyzer

```
rocm_mark_as_analyzer(<target>)
```

Marks a target to be included with the `analyze` target.

9.2 Variables

ROCM_ENABLE_GH_ANNOTATIONS

Set this variable to `ON` so that analyzers will emit diagnostics in a format that GitHub can use to annotate pull requests.

CHAPTER

TEN

ROCMCLANGTIDY

10.1 Commands

rocm_enable_clang_tidy

```
rocm_enable_clang_tidy(
    [CHECKS <tidy-checks>...]
    [ERRORS <tidy-checks>...]
    [EXTRA_ARGS <args>...]
    [CLANG_ARGS <args>...]
    [HEADER_FILTER <filter>]
    [ALL]
    [ANALYZE_TEMPORARY_DTORS]
    [ENABLE_ALPHA_CHECKS]
)
```

Enable checks for clang tidy.

rocm_clang_tidy_check

```
rocm_clang_tidy_check(TARGET)
```

Check the sources from target with clang tidy.

10.2 Variables

CLANG_TIDY_EXE

This cached variable can be used to set which `clang-tidy` executable to use. By default it will search for `clang-tidy` on the system while preferring the `clang-tidy` found in the same directory as `CMAKE_CXX_COMPILER`.

CLANG_TIDY_CACHE

This is location of the `clang-tidy` cache. By default, this is stored in the build directory under `tidy-cache`.

CLANG_TIDY_CACHE_SIZE

This sets the size of the cache. If set to `0` it will disable the cache. By default, it will cache 10 runs when using the Makefile generators. When using other generators it is disabled since it is not supported.

CLANG_TIDY_DEPEND_ON_TARGET

When set to `On` it will build the target first before running `clang-tidy`. By default, this is set to `On`.

CLANG_TIDY_USE_COLOR

When set to `On` then `clang-tidy` will output diagnostics in color. By default, this is set to `On`. It is disabled if `ROCM_ENABLE_GH_ANNOTATIONS` is enabled.

10.3 Fixits

All fixits are saved in the build directory under `fixits/`. All fixits found during analysis can be applied by running `clang-apply-replacements fixits/`.

CHAPTER

ELEVEN

ROCMCPPCHECK

11.1 Commands

rocm_enable_cppcheck

```
rocm_enable_cppcheck(
    [CHECKS <checks>...]
    [SUPPRESS <suppresions>...]
    [DEFINE <defines>...]
    [UNDEFINE <undefines>...]
    [INCLUDE <include-paths>...]
    [SOURCES <sources>...]
    [ADDONS <addons>...]
    [RULE_FILE <path-to-rule-file>]
    [FORCE]
    [INCONCLUSIVE]
)
```

Enable checks for cppcheck.

11.2 Variables

CPPCHECK_EXE

This cached variable can be used to set which `cppcheck` executable to use. By default it will search for `cppcheck` on the system.

CPPCHECK_BUILD_DIR

Sets the path to use for cppcheck's build directory where it caches the analysis. By default, this is set the `cppcheck-build` under cmake's build directory.

CHAPTER

TWELVE

ROCMTEST

This adds a `check` target to build and run the tests using CTest. A `tests` target can be used to just build the tests. All the tests are then packaged in the test component.

12.1 Commands

rocm_enable_test_package

```
rocm_enable_test_package(<name>)
```

Enable the test package. This must be called before `rocm_create_package` in the same directory. The `<name>` specifies the name of the directory to install the tests to.

rocm_add_test

```
rocm_add_test(NAME <name> COMMAND <command> [<arg>...]
    [CONFIGURATIONS <config>...]
    [COMMAND_EXPAND_LISTS]
)
```

Add test command to CTest and to test package.

rocm_add_test_executable

```
rocm_add_test_executable(<name> <sources>...)
```

Adds an executable to be built and ran for tests. The executable will link in the dependencies specified with `rocm_test_link_libraries` or `rocm_test_include_directories`. It will also be installed with the test component. The name of the test will be the same as the name of the executable.

rocm_test_header

```
rocm_test_header(<test-name> <header-include>)
```

This will add a test for checking a header can be included standalone and that there is no ODR issues in the header.

rocm_test_headers

```
rocm_test_headers(
    PREFIX <prefix>
    HEADERS <header-paths>
    DEPENDS <targets>...
)
```

This will test multiple headers at once. The PREFIX will specifiy any prefix to the included file needed. THe HEADERS is a list of headers to test for. This can also include globbing. For each file, the PREFIX and base name will be used for the include. The DEPENDS can list targets to link in for the test.

rocm_install_test

```
rocm_install_test(
    [TARGETS <targets>...]
    [FILES <files>...]
    [DESTINATION <path>]
)
```

Install the target or file into the test directory. The DESTINATION can be specified for FILES but is relative to the test installation directory.

rocm_mark_as_test

```
rocm_mark_as_test(<targets>...)
```

This will include the target as part of the tests target.

rocm_link_test_dependencies

```
rocm_link_test_dependencies(<targets>...)
```

This will add test dependencies specified with rocm_test_link_libraries or rocm_test_include_directories.

rocm_test_link_libraries

```
rocm_test_link_libraries(<targets>...)
```

Targets to link to test executables.

12.2 Variables

CTEST_PARALLEL_LEVEL

The parallel level used for check target to run the tests. The default is the number of cores.

CTEST_TIMEOUT

The timeout used for check target to run the tests. The default is 5000 seconds.

ROCM_TEST_GDB

Use gdb to printout a stacktrace when a test fails. This is either set to On or Off.

CHAPTER

THIRTEEN

ROCMDOCS

This creates a doc target which can run all documentation generation for the project.

13.1 Commands

rocm_mark_as_doc

```
rocm_mark_as_doc(<target>)
```

Marks a target to be included with the doc target.

rocm_clean_doc_output

```
rocm_clean_doc_output(DIR)
```

Output directory of documentation that should be removed when calling `make clean`.

CHAPTER

FOURTEEN

ROCMDOXYGENDOC

14.1 Commands

rocm_add_doxygen_doc

```
rocm_add_doxygen_doc(
    [DEPENDS <doc-targets>]
    [<doxygen-setting>...]
)
```

This will generate a doxygen file and then create a `doxygen` target that will run doxygen. Doxygen settings can be passed directly to the `rocm_add_doxygen_doc`. Settings for doxygen can be found here.

14.2 Variables

DOXYGEN_EXECUTABLE

This cached variable can be used to set which `doxygen` executable to use. By default it will search for `doxygen` on the system.

DOT_EXECUTABLE

This cached variable can be used to set which `dot` executable to use. By default it will search for `dot` on the system. If `dot` is found then its path will be set in the generated doxygen file.

CHAPTER

FIFTEEN

ROCMSPHINXDOC

15.1 Commands

rocm_add_sphinx_doc

```
rocm_add_sphinx_doc(
    SRC_DIR
    BUILDER <sphinx-builder>
    [OUTPUT_DIR <output-directory>]
    [CONFIG_DIR <config-directory>]
    [DEPENDS <doc-targets>...]
    [VARS <sphinx-variables>...]
    [TEMPLATE_VARS <sphinx-variables>...]
    [USE_DOXYGEN]
)
```

This will create a `sphinx-${BUILDER}` doc-type target which will generate documentation using sphinx. The `SRC_DIR` should be the directory that contains the `conf.py` file.

The options are:

OUTPUT_DIR
: The directory where build output will be written. It takes its default, from `ROCM_CMAKE_DOCS_DIR` if set, otherwise defaults to `sphinx/${BUILDER}`. Relative paths are interpreted relative to `CMAKE_CURRENT_BINARY_DIR`.

CONFIG_DIR
: The directory where `conf.py` will be searched. It sets the `-c` argument of `sphinx-build` if set, otherwise defaults to `${SRC_DIR}` as per the underlying tool. Relative paths are interpreted relative to `CMAKE_CURRENT_SOURCE_DIR`.

DEPENDS
: Sets up target-level dependencies between `sphinx-${BUILDER}` and the user-provided list of targets.

VARS
: List of configuration values passed to Sphinx. List items will be passed as command-line args by prepending `-D` to each item.

TEMPLATE_VARS
: List of HTML template values passed to Sphinx. List items will be passed as command-line args by prepending `-A` to each item.

USE_DOXYGEN
: Flag denoting the use of Doxygen in the fashion rocm-docs-core expects it.

15.2 Variables

SPHINX_EXECUTABLE

This cached variable can be used to set which `sphinx-build` executable to use. By default it will search for `sphinx-build` on the system. The `SPHINX_DIR` environmental variable is taken as a `HINT` while searching.

DOXYGEN_EXECUTABLE

This cached variable can be used to set which `doxygen` executable to use. By default it will search for `doxygen` on the system. The `DOXYGEN_DIR` environmental variable is taken as a `HINT` while searching.

SPHINX_${BUILDER}_DIR

This is the directory where the documentation will be built. By default, it will use the `OUTPUT_DIR` passed to `rocm_add_sphinx_doc` otherwise it will be set to `sphinx/${BUILDER}` directory in the cmake build directory.

CHAPTER

SIXTEEN

ROCMUTILITIES

16.1 Commands

rocm_join_if_set

```
rocm_join_if_set(<glue> <inout_var> [<input>...])
```

Join all the <input> arguments together using the <glue> string. If <inout_var> names a variable with a set value, join that string at the beginning, also using the <glue> string, and always store the result in <inout_var>.

rocm_defer

```
rocm_defer(<command>)
```

Call <command> at the end of configure.

rocm_find_program_version

```
rocm_find_program_version(
    <PROGRAM>
    [QUIET] [REQUIRED]
    [GREATER <version>]
    [GREATER_EQUAL <version>]
    [LESS <version>]
    [LESS_EQUAL <version>]
    [EQUAL <version>]
    [OUTPUT_VARIABLE <out-var>]
)
```

Determine the presence and installed version of a program that accepts the `--version` option. Optionally check the version using any of the comparison operators (each comparison operator may only be specified once). If `out-var` is not specified, it defaults to <PROGRAM>_VERSION. If the program is found, <out-var> is set to the version detected. If that version satisfies all version constraints, the variable <out-var>_OK is set to TRUE, otherwise it is set to FALSE.

CHAPTER

SEVENTEEN

LICENSE

MIT License

Copyright (c) 2017 Advanced Micro Devices, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

INDEX

S

V