NAME

cmake-policies - CMake Policies Reference

INTRODUCTION

Policies in CMake are used to preserve backward compatible behavior across multiple releases. When a new policy is introduced, newer CMake versions will begin to warn about the backward compatible behavior. It is possible to disable the warning by explicitly requesting the OLD, or backward compatible behavior using the **cmake_policy()** command. It is also possible to request **NEW**, or non-backward compatible behavior for a policy, also avoiding the warning. Each policy can also be set to either **NEW** or **OLD** behavior explicitly on the command line with the **CMAKE_POLICY_DEFAULT_CMP**<NNNN> variable.

The **cmake_minimum_required()** command does more than report an error if a too-old version of CMake is used to build a project. It also sets all policies introduced in that CMake version or earlier to **NEW** behavior. To manage policies without increasing the minimum required CMake version, the **if(POLICY)** command may be used:

```
if(POLICY CMP0990)
cmake_policy(SET CMP0990 NEW)
endif()
```

This has the effect of using the **NEW** behavior with newer CMake releases which users may be using and not issuing a compatibility warning.

The setting of a policy is confined in some cases to not propagate to the parent scope. For example, if the files read by the **include()** command or the **find_package()** command contain a use of **cmake_policy()**, that policy setting will not affect the caller by default. Both commands accept an optional **NO_POLICY_SCOPE** keyword to control this behavior.

The **CMAKE_MINIMUM_REQUIRED_VERSION** variable may also be used to determine whether to report an error on use of deprecated macros or functions.

ALL POLICIES

CMP0000

A minimum required CMake version must be specified.

CMake requires that projects specify the version of CMake to which they have been written. This policy has been put in place so users trying to build the project may be told when they need to update their CMake. Specifying a version also helps the project build with CMake versions newer than that specified. Use the cmake_minimum_required command at the top of your main CMakeLists.txt file:

```
cmake_minimum_required(VERSION <major>.<minor>)
```

where <major>.<minor> is the version of CMake you want to support (such as 2.6). The command will ensure that at least the given version of CMake is running and help newer versions be compatible with the project. See documentation of cmake_minimum_required for details.

Note that the command invocation must appear in the CMakeLists.txt file itself; a call in an included file is not sufficient. However, the cmake_policy command may be called to set policy CMP0000 to OLD or NEW behavior explicitly. The OLD behavior is to silently ignore the missing invocation. The NEW behavior is to issue an error instead of a warning. An included file may set CMP0000 explicitly to affect how this policy is enforced for the main CMakeLists.txt file.

This policy was introduced in CMake version 2.6.0.

CMP0001

CMAKE_BACKWARDS_COMPATIBILITY should no longer be used.

The OLD behavior is to check CMAKE_BACKWARDS_COMPATIBILITY and present it to the user. The NEW behavior is to ignore CMAKE_BACKWARDS_COMPATIBILITY completely.

In CMake 2.4 and below the variable CMAKE_BACKWARDS_COMPATIBILITY was used to request compatibility with earlier versions of CMake. In CMake 2.6 and above all compatibility issues are handled by policies and the cmake_policy command. However, CMake must still check CMAKE_BACKWARDS_COMPATIBILITY for projects written for CMake 2.4 and below.

This policy was introduced in CMake version 2.6.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0002

Logical target names must be globally unique.

Targets names created with add_executable, add_library, or add_custom_target are logical build target names. Logical target names must be globally unique because:

```
- Unique names may be referenced unambiguously both in CMake
code and on make tool command lines.
- Logical names are used by Xcode and VS IDE generators
to produce meaningful project names for the targets.
```

The logical name of executable and library targets does not have to correspond to the physical file names built. Consider using the OUTPUT_NAME target property to create two targets with the same physical name while keeping logical names distinct. Custom targets must simply have globally unique names (unless one uses the global property ALLOW_DUPLICATE_CUSTOM_TARGETS with a Makefiles generator).

This policy was introduced in CMake version 2.6.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0003

Libraries linked via full path no longer produce linker search paths.

This policy affects how libraries whose full paths are NOT known are found at link time, but was created due to a change in how CMake deals with libraries whose full paths are known. Consider the code

```
target_link_libraries(myexe /path/to/libA.so)
```

CMake 2.4 and below implemented linking to libraries whose full paths are known by splitting them on the link line into separate components consisting of the linker search path and the library name. The example code might have produced something like

```
... -L/path/to -lA ...
```

in order to link to library A. An analysis was performed to order multiple link directories such that the linker would find library A in the desired location, but there are cases in which this does not work. CMake versions 2.6 and above use the more reliable approach of passing the full path to libraries directly to the linker in most cases. The example code now produces something like

```
... /path/to/libA.so ....
```

Unfortunately this change can break code like

```
target_link_libraries(myexe /path/to/libA.so B)
```

where B is meant to find /path/to/libB.so. This code is wrong because the user is asking the linker to find library B but has not provided a linker search path (which may be added with the link_directories command). However, with the old linking implementation the code would work accidentally because the linker search path added for library A allowed library B to be found.

In order to support projects depending on linker search paths added by linking to libraries with known full paths, the OLD behavior for this policy will add the linker search paths even though they are not needed for their own libraries. When this policy is set to OLD, CMake will produce a link line such as

```
... -L/path/to /path/to/libA.so -lB ...
```

which will allow library B to be found as it was previously. When this policy is set to NEW, CMake will produce a link line such as

```
... /path/to/libA.so -lB ...
```

which more accurately matches what the project specified.

The setting for this policy used when generating the link line is that in effect when the target is created by an add_executable or add_library command. For the example described above, the code

```
cmake_policy(SET CMP0003 OLD) # or cmake_policy(VERSION 2.4)
add_executable(myexe myexe.c)
target_link_libraries(myexe /path/to/libA.so B)
```

will work and suppress the warning for this policy. It may also be updated to work with the corrected linking approach:

```
cmake_policy(SET CMP0003 NEW) # or cmake_policy(VERSION 2.6)
link_directories(/path/to) # needed to find library B
add_executable(myexe myexe.c)
target_link_libraries(myexe /path/to/libA.so B)
```

Even better, library B may be specified with a full path:

```
add_executable(myexe myexe.c)
target_link_libraries(myexe /path/to/libA.so /path/to/libB.so)
```

When all items on the link line have known paths CMake does not check this policy so it has no effect.

Note that the warning for this policy will be issued for at most one target. This avoids flooding users with messages for every target when setting the policy once will probably fix all targets.

This policy was introduced in CMake version 2.6.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0004

Libraries linked may not have leading or trailing whitespace.

CMake versions 2.4 and below silently removed leading and trailing whitespace from libraries linked with code like

```
target_link_libraries(myexe " A ")
```

This could lead to subtle errors in user projects.

The OLD behavior for this policy is to silently remove leading and trailing whitespace. The NEW behavior for this policy is to diagnose the existence of such whitespace as an error. The setting for this policy used when checking the library names is that in effect when the target is created by an add_executable or add_library command.

This policy was introduced in CMake version 2.6.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0005

Preprocessor definition values are now escaped automatically.

This policy determines whether or not CMake should generate escaped preprocessor definition values added via add_definitions. CMake versions 2.4 and below assumed that only trivial values would be given for macros in add_definitions calls. It did not attempt to escape non-trivial values such as string literals in generated build rules. CMake versions 2.6 and above support escaping of most values, but cannot assume the user has not added escapes already in an attempt to work around limitations in earlier versions.

The OLD behavior for this policy is to place definition values given to add_definitions directly in the generated build rules without attempting to escape anything. The NEW behavior for this policy is to generate correct escapes for all native build tools automatically. See documentation of the COMPILE_DEFINITIONS target property for limitations of the escaping implementation.

This policy was introduced in CMake version 2.6.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0006

Installing MACOSX_BUNDLE targets requires a BUNDLE DESTINATION.

This policy determines whether the install(TARGETS) command must be given a BUNDLE DESTINATION when asked to install a target with the MACOSX_BUNDLE property set. CMake 2.4 and below did not distinguish application bundles from normal executables when installing targets. CMake 2.6 provides a BUNDLE option to the install(TARGETS) command that specifies rules specific to application bundles on the Mac. Projects should use this option when installing a target with the MACOSX_BUNDLE property set.

The OLD behavior for this policy is to fall back to the RUNTIME DESTINATION if a BUNDLE DESTINATION is not given. The NEW behavior for this policy is to produce an error if a bundle target is installed without a BUNDLE DESTINATION.

This policy was introduced in CMake version 2.6.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0007

list command no longer ignores empty elements.

This policy determines whether the list command will ignore empty elements in the list. CMake 2.4 and below list commands ignored all empty elements in the list. For example, a;b;;c would have length 3 and not 4. The OLD behavior for this policy is to ignore empty list elements. The NEW behavior for this policy is to correctly count empty elements in a list.

This policy was introduced in CMake version 2.6.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0008

Libraries linked by full-path must have a valid library file name.

In CMake 2.4 and below it is possible to write code like

```
target_link_libraries(myexe /full/path/to/somelib)
```

where somelib is supposed to be a valid library file name such as libsomelib.a or somelib.lib. For Makefile generators this produces an error at build time because the dependency on the full path cannot be found. For VS IDE and Xcode generators this used to work by accident because CMake would always split off the library directory and ask the linker to search for the library by name (-lsomelib or somelib.lib). Despite the failure with Makefiles, some projects have code like this and build only with VS and/or Xcode. This version of CMake prefers to pass the full path directly to the native build tool, which will fail in this case because it does not name a valid library file.

This policy determines what to do with full paths that do not appear to name a valid library file.

The OLD behavior for this policy is to split the library name from the path and ask the linker to search for it. The NEW behavior for this policy is to trust the given path and pass it directly to the native build tool unchanged.

This policy was introduced in CMake version 2.6.1. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0009

FILE GLOB_RECURSE calls should not follow symlinks by default.

In CMake 2.6.1 and below, FILE GLOB_RECURSE calls would follow through symlinks, sometimes coming up with unexpectedly large result sets because of symlinks to top level directories that contain hundreds of thousands of files.

This policy determines whether or not to follow symlinks encountered during a FILE GLOB_RECURSE call. The OLD behavior for this policy is to follow the symlinks. The NEW behavior for this policy is not to follow the symlinks by default, but only if FOLLOW_SYMLINKS is given as an additional argument to the FILE command.

This policy was introduced in CMake version 2.6.2. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0010

Bad variable reference syntax is an error.

In CMake 2.6.2 and below, incorrect variable reference syntax such as a missing close-brace (${FOO) was reported but did not stop processing of CMake code. This policy determines whether a bad variable reference is an error. The OLD behavior for this policy is to warn about the error, leave the string untouched, and continue. The NEW behavior for this policy is to report an error.

This policy was introduced in CMake version 2.6.3. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0011

Included scripts do automatic cmake_policy PUSH and POP.

In CMake 2.6.2 and below, CMake Policy settings in scripts loaded by the include() and find_package() commands would affect the includer. Explicit invocations of cmake_policy(PUSH) and cmake_policy(POP) were required to isolate policy changes and protect the includer. While some scripts intend to affect the policies of their includer, most do not. In CMake 2.6.3 and above, include() and find_package() by default PUSH and POP an entry on the policy stack around an included script, but provide a NO_POLICY_SCOPE option to disable it. This policy determines whether or not to imply NO_POLICY_SCOPE for compatibility. The OLD behavior for this policy is to imply NO_POLICY_SCOPE for include() and find_package() commands. The NEW behavior for this policy is to allow the commands to do their default cmake_policy PUSH and POP.

This policy was introduced in CMake version 2.6.3. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0012

if() recognizes numbers and boolean constants.

In CMake versions 2.6.4 and lower the if() command implicitly dereferenced arguments corresponding to variables, even those named like numbers or boolean constants, except for 0 and 1. Numbers and boolean constants such as true, false, yes, no, on, off, y, n, notfound, ignore (all case insensitive) were recognized in some cases but not all. For example, the code if(TRUE) might have evaluated as false. Numbers such as 2 were recognized only in boolean expressions like if(NOT 2) (leading to false) but not as a single-argument like if(2) (also leading to false). Later versions of CMake prefer to treat numbers and boolean constants literally, so they should not be used as variable names.

The OLD behavior for this policy is to implicitly dereference variables named like numbers and boolean constants. The NEW behavior for this policy is to recognize numbers and boolean constants without dereferencing variables with such names.

This policy was introduced in CMake version 2.8.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0013

Duplicate binary directories are not allowed.

CMake 2.6.3 and below silently permitted add_subdirectory() calls to create the same binary directory multiple times. During build system generation files would be written and then overwritten in the build tree and could lead to strange behavior. CMake 2.6.4 and above explicitly detect duplicate binary directories. CMake 2.6.4 always considers this case an error. In CMake 2.8.0 and above this policy determines whether or not the case is an error. The OLD behavior for this policy is to allow duplicate binary directories. The NEW behavior for this policy is to disallow duplicate binary directories with an error.

This policy was introduced in CMake version 2.8.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0014

Input directories must have CMakeLists.txt.

CMake versions before 2.8 silently ignored missing CMakeLists.txt files in directories referenced by add_subdirectory() or subdirs(), treating them as if present but empty. In CMake 2.8.0 and above this policy determines whether or not the case is an error. The OLD behavior for this policy is to silently ignore the problem. The NEW behavior for this policy is to report an error.

This policy was introduced in CMake version 2.8.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0015

link_directories() treats paths relative to the source dir.

In CMake 2.8.0 and lower the link_directories() command passed relative paths unchanged to the linker. In CMake 2.8.1 and above the link_directories() command prefers to interpret relative paths with respect to CMAKE_CURRENT_SOURCE_DIR, which is consistent with include_directories() and other commands. The OLD behavior for this policy is to use relative paths verbatim in the linker command. The NEW behavior for this policy is to convert relative paths to absolute paths by appending the relative path to CMAKE_CURRENT_SOURCE_DIR.

This policy was introduced in CMake version 2.8.1. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0016

target_link_libraries() reports error if its only argument is not a target.

In CMake 2.8.2 and lower the target_link_libraries() command silently ignored if it was called with only one argument, and this argument wasnt a valid target. In CMake 2.8.3 and above it reports an error in this case.

This policy was introduced in CMake version 2.8.3. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0017

Prefer files from the CMake module directory when including from there.

Starting with CMake 2.8.4, if a cmake-module shipped with CMake (i.e. located in the CMake module directory) calls include() or find_package(), the files located in the CMake module directory are preferred over the files in CMAKE_MODULE_PATH. This makes sure that the modules belonging to CMake always get those files included which they expect, and against which they were developed and tested. In all other cases, the files found in CMAKE_MODULE_PATH still take precedence over the ones in the CMake module directory. The OLD behavior is to always prefer files from CMAKE_MODULE_PATH over files from the CMake modules directory.

This policy was introduced in CMake version 2.8.4. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0018

Ignore CMAKE_SHARED_LIBRARY_<Lang>_FLAGS variable.

CMake 2.8.8 and lower compiled sources in SHARED and MODULE libraries using the value of the undocumented CMAKE_SHARED_LIBRARY_<Lang>_FLAGS platform variable. The variable contained platform-specific flags needed to compile objects for shared libraries. Typically it included a flag such as -fPIC for position independent code but also included other flags needed on certain platforms. CMake 2.8.9 and higher prefer instead to use the POSITION_INDEPENDENT_CODE target property to determine what targets should be position independent, and new undocumented platform variables to select flags while ignoring CMAKE_SHARED_LIBRARY_<Lang>_FLAGS completely.

The default for either approach produces identical compilation flags, but if a project modifies CMAKE_SHARED_LIBRARY_<Lang>_FLAGS from its original value this policy determines which approach to use.

The OLD behavior for this policy is to ignore the POSITION_INDEPENDENT_CODE property for all targets and use the modified value of CMAKE_SHARED_LIBRARY_<Lang>_FLAGS for SHARED and MODULE libraries.

The NEW behavior for this policy is to ignore CMAKE_SHARED_LIBRARY_<Lang>_FLAGS whether it is modified or not and honor the POSITION_INDEPENDENT_CODE target property.

This policy was introduced in CMake version 2.8.9. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0019

Do not re-expand variables in include and link information.

CMake 2.8.10 and lower re-evaluated values given to the include_directories, link_directories, and link_libraries commands to expand any leftover variable references at the end of the configuration step. This was for strict compatibility with VERY early CMake versions because all variable references are now normally evaluated during CMake language processing. CMake 2.8.11 and higher prefer to skip the extra evaluation.

The OLD behavior for this policy is to re-evaluate the values for strict compatibility. The NEW behavior for this policy is to leave the values untouched.

This policy was introduced in CMake version 2.8.11. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0020

Automatically link Qt executables to qtmain target on Windows.

CMake 2.8.10 and lower required users of Qt to always specify a link dependency to the qtmain.lib static library manually on Windows. CMake 2.8.11 gained the ability to evaluate generator expressions while determining the link dependencies from IMPORTED targets. This allows CMake itself to automatically link executables which link to Qt to the qtmain.lib library when using IMPORTED Qt targets. For applications already linking to qtmain.lib, this should have little impact. For applications which supply their own alternative WinMain implementation and for applications which use the QAxServer library, this automatic linking will need to be disabled as per the documentation.

The OLD behavior for this policy is not to link executables to qtmain.lib automatically when they link to the QtCore IMPORTEDtarget. The NEW behavior for this policy is to link executables to qtmain.lib automatically when they link to QtCore IMPORTED target.

This policy was introduced in CMake version 2.8.11. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0021

Fatal error on relative paths in INCLUDE_DIRECTORIES target property.

CMake 2.8.10.2 and lower allowed the INCLUDE_DIRECTORIES target property to contain relative paths. The base path for such relative entries is not well defined. CMake 2.8.12 issues a FATAL_ERROR if the INCLUDE_DIRECTORIES property contains a relative path.

The OLD behavior for this policy is not to warn about relative paths in the INCLUDE_DIRECTORIES target property. The NEW behavior for this policy is to issue a FATAL_ERROR if INCLUDE_DIRECTORIES contains a relative path.

This policy was introduced in CMake version 2.8.12. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0022

INTERFACE_LINK_LIBRARIES defines the link interface.

CMake 2.8.11 constructed the link interface of a target from properties matching **(IMPORTED_)?LINK_INTERFACE_LIBRARIES(_<CONFIG>)?**. The modern way to specify config-sensitive content is to use generator expressions and the **IMPORTED_** prefix makes uniform processing of the link interface with generator expressions impossible. The INTERFACE_LINK_LIBRARIES target property was introduced as a replacement in CMake 2.8.12. This new property is named consistently with the INTERFACE_COMPILE_DEFINITIONS, INTERFACE_INCLUDE_DIRECTORIES and INTERFACE_COMPILE_OPTIONS properties. For in-build targets, CMake will use the INTERFACE_LINK_LIBRARIES property as the source of the link interface only if policy CMP0022 is NEW. When exporting a target which has this policy set to NEW, only the INTERFACE_LINK_LIBRARIES property will be processed and generated for the IMPORTED target by default. A new option to the install(EXPORT) and export commands allows export of the old-style properties for compatibility with downstream users of CMake versions older than 2.8.12. The target_link_libraries command will no longer populate the properties matching LINK_INTERFACE_LIBRARIES(_<CONFIG>)? if this policy is NEW.

Warning-free future-compatible code which works with CMake 2.8.7 onwards can be written by using the **LINK_PRIVATE** and **LINK_PUBLIC** keywords of **target_link_libraries()**.

The OLD behavior for this policy is to ignore the INTERFACE_LINK_LIBRARIES property for in-build targets. The NEW behavior for this policy is to use the INTERFACE_LINK_LIBRARIES property for in-build targets, and ignore the old properties matching **(IMPORTED_)?LINK_INTERFACE_LIBRARIES(_<CONFIG>)?**.

This policy was introduced in CMake version 2.8.12. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0023

Plain and keyword target_link_libraries signatures cannot be mixed.

CMake 2.8.12 introduced the target_link_libraries signature using the PUBLIC, PRIVATE, and INTERFACE keywords to generalize the LINK_PUBLIC and LINK_PRIVATE keywords introduced in CMake 2.8.7. Use of signatures with any of these keywords sets the link interface of a target explicitly, even if empty. This produces confusing behavior when used in combination with the historical behavior of the plain target_link_libraries signature. For example, consider the code:

```
target_link_libraries(mylib A)
target_link_libraries(mylib PRIVATE B)
```

After the first line the link interface has not been set explicitly so CMake would use the link implementation, A, as the link interface. However, the second line sets the link interface to empty. In order to avoid this subtle behavior CMake now prefers to disallow mixing the plain and keyword signatures of target_link_libraries for a single target.

The OLD behavior for this policy is to allow keyword and plain target_link_libraries signatures to be mixed. The NEW behavior for this policy is to not to allow mixing of the keyword and plain signatures.

This policy was introduced in CMake version 2.8.12. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0024

Disallow include export result.

CMake 2.8.12 and lower allowed use of the include() command with the result of the export() command. This relies on the assumption that the export() command has an immediate effect at configure-time during a cmake run. Certain properties of targets are not fully determined until later at generate-time, such as the link language and complete list of link libraries. Future refactoring will change the effect of the export() command to be executed at generate-time. Use ALIAS targets instead in cases where the goal is to refer to targets by another name.

The OLD behavior for this policy is to allow including the result of an export() command. The NEW behavior for this policy is not to allow including the result of an export() command.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0025

Compiler id for Apple Clang is now **AppleClang**.

CMake 3.0 and above recognize that Apple Clang is a different compiler than upstream Clang and that they have different version numbers. CMake now prefers to present this to projects by setting the **CMAKE_<LANG>_COMPILER_ID** variable to **AppleClang** instead of **Clang**. However, existing projects may assume the compiler id for Apple Clang is just **Clang** as it was in CMake versions prior to 3.0. Therefore this policy determines for Apple Clang which compiler id to report in the **CMAKE_<LANG>_COMPILER_ID** variable after language **<LANG>** is enabled by the **project()** or **enable_language()** command. The policy must be set prior to the invocation of either command.

The OLD behavior for this policy is to use compiler id **Clang**. The NEW behavior for this policy is to use compiler id **AppleClang**.

This policy was introduced in CMake version 3.0. Use the **cmake_policy()** command to set this policy to OLD or NEW explicitly. Unlike most policies, CMake version 3.0.2 does *not* warn by default when this policy is not set and simply uses OLD behavior. See documentation of the **CMAKE_POLICY_WARNING_CMP0025** variable to control the warning.

CMP0026

Disallow use of the LOCATION target property.

CMake 2.8.12 and lower allowed reading the LOCATION target property (and configuration-specific variants) to determine the eventual location of build targets. This relies on the assumption that all necessary information is available at configure-time to determine the final location and filename of the target. However, this property is not fully determined until later at generate-time. At generate time, the $<TARGET_FILE> generator expression can be used to determine the eventual LOCATION of a target output.

Code which reads the LOCATION target property can be ported to use the $<TARGET_FILE> generator expression together with the file(GENERATE) subcommand to generate a file containing the target location.

The OLD behavior for this policy is to allow reading the LOCATION properties from build-targets. The NEW behavior for this policy is to not to allow reading the LOCATION properties from build-targets.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0027

Conditionally linked imported targets with missing include directories.

CMake 2.8.11 introduced introduced the concept of INTERFACE_INCLUDE_DIRECTORIES, and a check at cmake time that the entries in the INTERFACE_INCLUDE_DIRECTORIES of an IMPORTED target actually exist. CMake 2.8.11 also introduced generator expression support in the target_link_libraries command. However, if an imported target is linked as a result of a generator expression evaluation, the entries in the INTERFACE_INCLUDE_DIRECTORIES of that target were not checked for existence as they should be.

The OLD behavior of this policy is to report a warning if an entry in the INTERFACE_INCLUDE_DIRECTORIES of a generator-expression conditionally linked IMPORTED target does not exist.

The NEW behavior of this policy is to report an error if an entry in the INTERFACE_INCLUDE_DIRECTORIES of a generator-expression conditionally linked IMPORTED target does not exist.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0028

Double colon in target name means ALIAS or IMPORTED target.

CMake 2.8.12 and lower allowed the use of targets and files with double colons in target_link_libraries, with some buildsystem generators.

The use of double-colons is a common pattern used to namespace IMPORTED targets and ALIAS targets. When computing the link dependencies of a target, the name of each dependency could either be a target, or a file on disk. Previously, if a target was not found with a matching name, the name was considered to refer to a file on disk. This can lead to confusing error messages if there is a typo in what should be a target name.

The OLD behavior for this policy is to search for targets, then files on disk, even if the search term contains double-colons. The NEW behavior for this policy is to issue a FATAL_ERROR if a link dependency contains double-colons but is not an IMPORTED target or an ALIAS target.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0029

The **subdir_depends()** command should not be called.

The implementation of this command has been empty since December 2001 but was kept in CMake for compatibility for a long time.

CMake >= 3.0 prefer that this command never be called. The OLD behavior for this policy is to allow the command to be called. The NEW behavior for this policy is to issue a FATAL_ERROR when the command is called.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0030

The **use_mangled_mesa()** command should not be called.

This command was created in September 2001 to support VTK before modern CMake language and custom command capabilities. VTK has not used it in years.

CMake >= 3.0 prefer that this command never be called. The OLD behavior for this policy is to allow the command to be called. The NEW behavior for this policy is to issue a FATAL_ERROR when the command is called.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0031

The **load_command()** command should not be called.

This command was added in August 2002 to allow projects to add arbitrary commands implemented in C or C++. However, it does not work when the toolchain in use does not match the ABI of the CMake process. It has been mostly superseded by the **macro()** and **function()** commands.

CMake >= 3.0 prefer that this command never be called. The OLD behavior for this policy is to allow the command to be called. The NEW behavior for this policy is to issue a FATAL_ERROR when the command is called.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0032

The **output_required_files()** command should not be called.

This command was added in June 2001 to expose the then-current CMake implicit dependency scanner. CMakes real implicit dependency scanner has evolved since then but is not exposed through this command. The scanning capabilities of this command are very limited and this functionality is better achieved through dedicated outside tools.

CMake >= 3.0 prefer that this command never be called. The OLD behavior for this policy is to allow the command to be called. The NEW behavior for this policy is to issue a FATAL_ERROR when the command is called.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0033

The **export_library_dependencies()** command should not be called.

This command was added in January 2003 to export **<tgt>_LIB_DEPENDS** internal CMake cache entries to a file for installation with a project. This was used at the time to allow transitive link dependencies to work for applications outside of the original build tree of a project. The functionality has been superseded by the **export()** and **install(EXPORT)** commands.

CMake >= 3.0 prefer that this command never be called. The OLD behavior for this policy is to allow the command to be called. The NEW behavior for this policy is to issue a FATAL_ERROR when the command is called.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0034

The **utility_source()** command should not be called.

This command was introduced in March 2001 to help build executables used to generate other files. This approach has long been replaced by **add_executable()** combined with **add_custom_command()**.

CMake >= 3.0 prefer that this command never be called. The OLD behavior for this policy is to allow the command to be called. The NEW behavior for this policy is to issue a FATAL_ERROR when the command is called.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0035

The **variable_requires()** command should not be called.

This command was introduced in November 2001 to perform some conditional logic. It has long been replaced by the **if()** command.

CMake >= 3.0 prefer that this command never be called. The OLD behavior for this policy is to allow the command to be called. The NEW behavior for this policy is to issue a FATAL_ERROR when the command is called.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0036

The **build_name()** command should not be called.

This command was added in May 2001 to compute a name for the current operating system and compiler combination. The command has long been documented as discouraged and replaced by the **CMAKE_SYSTEM** and **CMAKE_<LANG>_COMPILER** variables.

CMake >= 3.0 prefer that this command never be called. The OLD behavior for this policy is to allow the command to be called. The NEW behavior for this policy is to issue a FATAL_ERROR when the command is called.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0037

Target names should not be reserved and should match a validity pattern.

CMake 2.8.12 and lower allowed creating targets using **add_library()**, **add_executable()** and **add_custom_target()** with unrestricted choice for the target name. Newer cmake features such as **cmake-generator-expressions(7)** and some diagnostics expect target names to match a restricted pattern.

Target names may contain upper and lower case letters, numbers, the underscore character (_), dot(.), plus(+) and minus(-). As a special case, ALIAS targets and IMPORTED targets may contain two consequtive colons.

Target names reserved by one or more CMake generators are not allowed. Among others these include all, help and test.

The OLD behavior for this policy is to allow creating targets with reserved names or which do not match the validity pattern. The NEW behavior for this policy is to report an error if an add_* command is used with an invalid target name.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0038

Targets may not link directly to themselves.

CMake 2.8.12 and lower allowed a build target to link to itself directly with a **target_link_libraries()** call. This is an indicator of a bug in user code.

The OLD behavior for this policy is to ignore targets which list themselves in their own link implementation. The NEW behavior for this policy is to report an error if a target attempts to link to itself.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0039

Utility targets may not have link dependencies.

CMake 2.8.12 and lower allowed using utility targets in the left hand side position of the **target_link_libraries()** command. This is an indicator of a bug in user code.

The OLD behavior for this policy is to ignore attempts to set the link libraries of utility targets. The NEW behavior for this policy is to report an error if an attempt is made to set the link libraries of a utility target.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0040

The target in the TARGET signature of add_custom_command() must exist.

CMake 2.8.12 and lower silently ignored a custom command created with the TARGET signature of **add_custom_command()** if the target is unknown.

The OLD behavior for this policy is to ignore custom commands for unknown targets. The NEW behavior for this policy is to report an error if the target referenced in **add_custom_command()** is unknown.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0041

Error on relative include with generator expression.

Diagnostics in CMake 2.8.12 and lower silently ignored an entry in the **INTERFACE_INCLUDE_DIRECTORIES** of a target if it contained a generator expression at any position.

The path entries in that target property should not be relative. High-level API should ensure that by adding either a source directory or a install directory prefix, as appropriate.

As an additional diagnostic, the **INTERFACE_INCLUDE_DIRECTORIES** generated on an **IMPORTED** target for the install location should not contain paths in the source directory or the build directory.

The OLD behavior for this policy is to ignore relative path entries if they contain a generator expression. The NEW behavior for this policy is to report an error if a generator expression appears in another location and the path is relative.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0042

MACOSX_RPATH is enabled by default.

CMake 2.8.12 and newer has support for using **@rpath** in a targets install name. This was enabled by setting the target property **MACOSX_RPATH**. The **@rpath** in an install name is a more flexible and powerful mechanism than **@executable_path** or **@loader_path** for locating shared libraries.

CMake 3.0 and later prefer this property to be ON by default. Projects wanting **@rpath** in a targets install name may remove any setting of the **INSTALL_NAME_DIR** and **CMAKE_INSTALL_NAME_DIR** variables.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0043

Ignore COMPILE_DEFINITIONS_<Config> properties

CMake 2.8.12 and lower allowed setting the **COMPILE_DEFINITIONS_<CONFIG>** target property and **COMPILE_DEFINITIONS_<CONFIG>** directory property to apply configuration-specific compile definitions.

Since CMake 2.8.10, the **COMPILE_DEFINITIONS** property has supported **generator expressions** for setting configuration-dependent content. The continued existence of the suffixed variables is redundant, and causes a maintenance burden. Population of the **COMPILE_DEFINITIONS_DEBUG** property may be replaced with a population of **COMPILE_DEFINITIONS** directly or via **target_compile_definitions()**:

```
# Old Interfaces:
set_property(TARGET tgt APPEND PROPERTY
COMPILE_DEFINITIONS_DEBUG DEBUG_MODE
)
set_property(DIRECTORY APPEND PROPERTY
COMPILE_DEFINITIONS_DEBUG DIR_DEBUG_MODE
)

# New Interfaces:
```

```
set_property(TARGET tgt APPEND PROPERTY
COMPILE_DEFINITIONS $<$<CONFIG:Debug>:DEBUG_MODE>
)
target_compile_definitions(tgt PRIVATE $<$<CONFIG:Debug>:DEBUG_MODE>)
set_property(DIRECTORY APPEND PROPERTY
COMPILE_DEFINITIONS $<$<CONFIG:Debug>:DIR_DEBUG_MODE>
)
```

The OLD behavior for this policy is to consume the content of the suffixed **COMPILE_DEFINITIONS_<CONFIG>** target property when generating the compilation command. The NEW behavior for this policy is to ignore the content of the **COMPILE_DEFINITIONS_<CONFIG>** target property .

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0044

Case sensitive **<LANG>_COMPILER_ID** generator expressions

CMake 2.8.12 introduced the **<LANG>_COMPILER_ID generator expressions** to allow comparison of the **CMAKE_<LANG>_COMPILER_ID** with a test value. The possible valid values are lowercase, but the comparison with the test value was performed case-insensitively.

The OLD behavior for this policy is to perform a case-insensitive comparison with the value in the **<LANG>_COMPILER_ID** expression. The NEW behavior for this policy is to perform a case-sensitive comparison with the value in the **<LANG>_COMPILER_ID** expression.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0045

Error on non-existent target in get_target_property.

In CMake 2.8.12 and lower, the **get_target_property()** command accepted a non-existent target argument without issuing any error or warning. The result variable is set to a **-NOTFOUND** value.

The OLD behavior for this policy is to issue no warning and set the result variable to a **-NOTFOUND** value. The NEW behavior for this policy is to issue a **FATAL_ERROR** if the command is called with a non-existent target.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0046

Error on non-existent dependency in add_dependencies.

CMake 2.8.12 and lower silently ignored non-existent dependencies listed in the **add_dependencies()** command.

The OLD behavior for this policy is to silently ignore non-existent dependencies. The NEW behavior for this policy is to report an error if non-existent dependencies are listed in the **add_dependencies()** command.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0047

Use **QCC** compiler id for the qcc drivers on QNX.

CMake 3.0 and above recognize that the QNX qcc compiler driver is different from the GNU compiler. CMake now prefers to present this to projects by setting the **CMAKE_<LANG>_COMPILER_ID** variable to **QCC** instead of **GNU**. However, existing projects may assume the compiler id for QNX qcc is just **GNU** as it was in CMake versions prior to 3.0. Therefore this policy determines for QNX qcc which compiler id to report in the **CMAKE_<LANG>_COMPILER_ID** variable after language **<LANG>** is enabled by the **project()** or **enable_language()** command. The policy must be set prior to the invocation of either command.

The OLD behavior for this policy is to use the **GNU** compiler id for the qcc and QCC compiler drivers. The NEW behavior for this policy is to use the **QCC** compiler id for those drivers.

This policy was introduced in CMake version 3.0. Use the **cmake_policy()** command to set this policy to OLD or NEW explicitly. Unlike most policies, CMake version 3.0.2 does *not* warn by default when this policy is not set and simply uses OLD behavior. See documentation of the **CMAKE_POLICY_WARNING_CMP0047** variable to control the warning.

CMP0048

The **project()** command manages VERSION variables.

CMake version 3.0 introduced the **VERSION** option of the **project()** command to specify a project version as well as the name. In order to keep **PROJECT_VERSION** and related variables consistent with variable **PROJECT_NAME** it is necessary to set the VERSION variables to the empty string when no **VERSION** is given to **project()**. However, this can change behavior for existing projects that set VERSION variables themselves since **project()** may now clear them. This policy controls the behavior for compatibility with such projects.

The OLD behavior for this policy is to leave VERSION variables untouched. The NEW behavior for this policy is to set VERSION as documented by the **project()** command.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0049

Do not expand variables in target source entries.

CMake 2.8.12 and lower performed and extra layer of variable expansion when evaluating source file names:

```
set(a_source foo.c)
add_executable(foo \${a_source})
```

This was undocumented behavior.

The OLD behavior for this policy is to expand such variables when processing the target sources. The NEW behavior for this policy is to issue an error if such variables need to be expanded.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

CMP0050

Disallow add_custom_command SOURCE signatures.

CMake 2.8.12 and lower allowed a signature for **add_custom_command()** which specified an input to a command. This was undocumented behavior. Modern use of CMake associates custom commands with their output, rather than their input.

The OLD behavior for this policy is to allow the use of **add_custom_command()** SOURCE signatures. The NEW behavior for this policy is to issue an error if such a signature is used.

This policy was introduced in CMake version 3.0. CMake version 3.0.2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.

COPYRIGHT