Appendix F – CMake 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 2.8.0-rc2 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 2.8.0-rc2 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 2.8.0-rc2 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 2.8.0-rc2 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 2.8.0-rc2 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 2.8.0-rc2 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 2.8.0-rc2 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 2.8.0-rc2 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 2.8.0-rc2 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 2.8.0-rc2 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 2.8.0-rc2 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: The if() command can recognize named boolean constants.

In CMake versions prior to 2.6.5 the only boolean constants were 0 and 1. Other 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. In later versions of cmake these values are treated as boolean constants more consistently and should not be used as variable names. The OLD behavior for this policy is to allow variables to have names such as true and to dereference them. The NEW behavior for this policy is to treat strings like true as a boolean constant.

This policy was introduced in CMake version 2.6.5. CMake version 2.8.0-rc2 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 2.6.5 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 2.6.5. CMake version 2.8.0-rc2 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 2.8.0-rc2 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: The set() CACHE mode and option() command make the cache value visible.

In CMake 2.6 and below the CACHE mode of the set() command and the option() command did not expose the value from the named cache entry if it was already set both in the cache and as a local variable. This led to subtle differences between first and later configurations because a conflicting local variable would be overridden only when the cache value was first created. For example, the code

```
set(x 1)
set(before ${x})
set(x 2 CACHE STRING "X")
set(after ${x})
message(STATUS "${before},${after}")
```

would print "1,2" on the first run and "1,1" on future runs.

CMake 2.8.0 and above prefer to expose the cache value in all cases by removing the local variable definition, but this changes behavior in subtle cases when the local variable has a different value than that exposed from the cache. The example above will always print "1,2".

This policy determines whether the commands should always expose the cache value. The OLD behavior for this policy is to leave conflicting local variable values untouched and hide the true cache value. The NEW behavior for this policy is to always expose the cache value.

This policy was introduced in CMake version 2.8.0. CMake version 2.8.0-rc2 warns when the policy is not set and uses OLD behavior. Use the cmake_policy command to set it to OLD or NEW explicitly.