3.4. Version and policies

Like any other piece of software CMake evolves, effectively introducing new features and deprecating dangerous or confusing behavior.

There are two entities that help you to manage difference between old and new versions of CMake:

- Command cmake_minimum_required: for checking what minimum version of CMake user should have to run your configuration
- CMake policies: for fine tuning newly introduced behavior

If you just want to experiment without worrying about backward compatibility, policies, warnings, etc. just set **first line** of CMakeLists.txt to `cmake_minimum_required(VERSION a.b.c)` where `a.b.c` is a current version of CMake you're using:

```
> cmake --version
cmake version 3.5.2

> cat CMakeLists.txt
cmake_minimum_required(VERSION 3.5.2)
```

3.4.1. cmake_minimum_required

CMake documentation

- cmake_minimum_required

What version to put into this command is mostly an executive decision. You need to know:

- what version is installed on users hosts?
- is it appropriate to ask them to install newer version?
- what features do they need?
- do you need to be backward compatible for one users and have fresh features for another?

The last case will fit most of them but will harder to maintain for developer and probably will require automatic testing system with good coverage.

See also

- CMake versions for Hunter

For example the code with version `2.8` as a minimum one and with `3.0` features will look like:

```
cmake_minimum_required(VERSION 2.8)

if(NOT CMAKE_VERSION VERSION_LESS "3.0") # means 'NOT version < 3.0', i.e. 'version >= 3.0'
  # Code with 3.0 features
endif()
```

Command `cmake_minimum_required` **must be the first** command in your CMakeLists.txt. If you're planning to support several versions of CMake then you need to put the smallest one in `cmake_minimum_required` and call it in the first line of CMakeLists.txt.

Even if some commands look harmless at the first glance it may be not so in fact, e.g. `project` is the place where a lot of checks happens and where toolchain is loaded. If you run this example on `Cygwin` platform:

```
project(foo) # BAD CODE! You should check version first!
cmake_minimum_required(VERSION 3.0)

message("Using CMake version ${CMAKE_VERSION}")
```

CMake will think that you're running code with old policies and warns you:

```
[minimum-required-example]> cmake -Hbad -B_builds/bad
-- The C compiler identification is GNU 4.9.3
-- The CXX compiler identification is GNU 4.9.3
CMake Warning at /.../share/cmake-3.3.1/Modules/Platform/CYGWIN.cmake:15 (message):
  CMake no longer defines WIN32 on Cygwin!

  (1) If you are just trying to build this project, ignore this warning or
  quiet it by setting CMAKE_LEGACY_CYGWIN_WIN32=0 in your environment or in
  the CMake cache.  If later configuration or build errors occur then this
  project may have been written under the assumption that Cygwin is WIN32.
  In that case, set CMAKE_LEGACY_CYGWIN_WIN32=1 instead.

  (2) If you are developing this project, add the line

    set(CMAKE_LEGACY_CYGWIN_WIN32 0) # Remove when CMake >= 2.8.4 is required

  at the top of your top-level CMakeLists.txt file or set the minimum
  required version of CMake to 2.8.4 or higher.  Then teach your project to
  build on Cygwin without WIN32.
Call Stack (most recent call first):
  /.../share/cmake-3.3.1/Modules/CMakeSystemSpecificInformation.cmake:36 (include)
  CMakeLists.txt:1 (project)
...
-- Detecting CXX compile features - done
Using CMake version 3.3.1
...
```

Fixed version:

```
cmake_minimum_required(VERSION 3.0)
project(foo)

message("Using CMake version ${CMAKE_VERSION}")
```

with no warnings:

```
[minimum-required-example]> cmake -Hgood -B_builds/good
-- The C compiler identification is GNU 4.9.3
-- The CXX compiler identification is GNU 4.9.3
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++.exe
-- Check for working CXX compiler: /usr/bin/c++.exe -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
Using CMake version 3.3.1
-- Configuring done
-- Generating done
-- Build files have been written to: /.../minimum-required-example/_builds/good
```

Examples on GitHub

- Repository
- Latest ZIP

3.4.2. CMake policies

CMake documentation

- CMake policies

When new version of CMake released there may be a list of policies describing cases when behavior changed comparing to the previous CMake version.

Let's see how it works on practice. In CMake `3.0` policy CMP0038 was introduced. Before version `3.0` user can have target linked to itself, which make no sense and definitely **is a bug**:

```
cmake_minimum_required(VERSION 2.8)
project(foo)

add_library(foo foo.cpp)

target_link_libraries(foo foo) # BAD CODE! Make no sense
```

Examples on GitHub

- Repository
- Latest ZIP

Works fine for CMake before `3.0`:

```
[policy-examples]> cmake --version
cmake version 2.8.12.2
[policy-examples]> rm -rf _builds
[policy-examples]> cmake -Hbug-2.8 -B_builds
-- The C compiler identification is GNU 4.8.4
-- The CXX compiler identification is GNU 4.8.4
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Configuring done
-- Generating done
-- Build files have been written to: /.../policy-examples/_builds
```

For CMake version `>= 3.0` warning will be reported:

```
[policy-examples]> cmake --version
cmake version 3.5.2
[policy-examples]> rm -rf _builds
[policy-examples]> cmake -Hbug-2.8 -B_builds
...
-- Configuring done
CMake Warning (dev) at CMakeLists.txt:4 (add_library):
  Policy CMP0038 is not set: Targets may not link directly to themselves.
  Run "cmake --help-policy CMP0038" for policy details.  Use the cmake_policy
  command to set the policy and suppress this warning.

  Target "foo" links to itself.
This warning is for project developers.  Use -Wno-dev to suppress it.

-- Generating done
-- Build files have been written to: /.../policy-examples/_builds
```

Assume you want to drop the support of old version and more to some new `3.0` features. When you set `cmake_minimum_required(VERSION 3.0)`

```
--- /examples/policy-examples/bug-2.8/CMakeLists.txt
+++ /examples/policy-examples/set-3.0/CMakeLists.txt
@@ -1,4 +1,4 @@
-cmake_minimum_required(VERSION 2.8)
+cmake_minimum_required(VERSION 3.0)
 project(foo)

 add_library(foo foo.cpp)
```

warning turns into error:

```
[policy-examples]> rm -rf _builds
[policy-examples]> cmake -Hset-3.0 -B_builds
-- The C compiler identification is GNU 4.8.4
-- The CXX compiler identification is GNU 4.8.4
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
CMake Error at CMakeLists.txt:4 (add_library):
  Target "foo" links to itself.


-- Generating done
-- Build files have been written to: /.../policy-examples/_builds
[policy-examples]> echo $?
1
```

Two cases will be shown below. In first case we want to keep support of old version (`2.8` for now) so it will work with both `CMake 2.8` and `CMake 3.0+`. In second case we decide to drop support of old version and move to `CMake 3.0+`. We'll see how it will affect policies. It will be shown in the end that in fact without **using new features** from `CMake 3.0` it doesn't make sense to change `cmake_minimum_required`.

3.4.2.1. Keep using old

Our project works fine with `CMake 2.8` however `CMake 3.0+` emits warning. We don't want to fix the error now but want only to suppress warning and explain to CMake that it should behaves like `CMake 2.8`.

> **Note**
>
> This approach described in documentation:

```
It is possible to disable the warning by explicitly requesting the OLD, or
backward compatible behavior using the cmake_policy() command
```

Let's add cmake_policy:

```
cmake_minimum_required(VERSION 2.8)
project(foo)

cmake_policy(SET CMP0038 OLD)

add_library(foo foo.cpp)

target_link_libraries(foo foo) # BAD CODE! Make no sense
```

Looks good for `CMake 3.0+`:

```
[policy-examples]> cmake --version
cmake version 3.5.2
[policy-examples]> rm -rf _builds
[policy-examples]> cmake -Hunknown-2.8 -B_builds
-- The C compiler identification is GNU 4.8.4
-- The CXX compiler identification is GNU 4.8.4
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
```

Are we done? No, `CMP0038` is introduced since `CMake 3.0` so `CMake 2.8` have no idea what this policy is about:

```
> cmake --version
cmake version 2.8.12.2
> rm -rf _builds
> cmake -Hunknown-2.8 -B_builds
-- The C compiler identification is GNU 4.8.4
-- The CXX compiler identification is GNU 4.8.4
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
CMake Error at CMakeLists.txt:4 (cmake_policy):
  Policy "CMP0038" is not known to this version of CMake.

-- Configuring incomplete, errors occurred!
```

We should protect new code with `if(POLICY CMP0038)` condition:

```
cmake_minimum_required(VERSION 2.8)
project(foo)

if(POLICY CMP0038)
  # Policy CMP0038 introduced since CMake 3.0 so if we want to be compatible
  # with 2.8 (see cmake_minimum_required) we should put 'cmake_policy' under
  # condition.
  cmake_policy(SET CMP0038 OLD)
endif()

add_library(foo foo.cpp)

target_link_libraries(foo foo) # BAD CODE! Make no sense
```

Of course you should find the time, apply real fix and remove policy logic since it will not be needed anymore:

```
--- /examples/policy-examples/suppress-2.8/CMakeLists.txt
+++ /examples/policy-examples/fix-2.8/CMakeLists.txt
@@ -1,13 +1,4 @@
 cmake_minimum_required(VERSION 2.8)
 project(foo)

-if(POLICY CMP0038)
-  # Policy CMP0038 introduced since CMake 3.0 so if we want to be compatible
-  # with 2.8 (see cmake_minimum_required) we should put 'cmake_policy' under
-  # condition.
-  cmake_policy(SET CMP0038 OLD)
-endif()
-
 add_library(foo foo.cpp)
-
-target_link_libraries(foo foo) # BAD CODE! Make no sense
```

Final version:

```
cmake_minimum_required(VERSION 2.8)
project(foo)

add_library(foo foo.cpp)
```

3.4.2.2. Moving to new version

With `cmake_minimum_required` updated to `3.0` warning turns into error. To suppress error without doing real fix (temporary solution) you can add `cmake_policy` directive:

```
cmake_minimum_required(VERSION 3.0)
project(foo)

cmake_policy(SET CMP0038 OLD)

add_library(foo foo.cpp)

target_link_libraries(foo foo) # BAD CODE! Make no sense
```

Note

We don't need to protect `cmake_policy` with `if(POLICY)` condition since `cmake_minimum_required(VERSION 3.0)` guarantee us that we are using `CMake 3.0+`.

Policy can be removed after real fix applied:

```
--- /examples/policy-examples/suppress-3.0/CMakeLists.txt
+++ /examples/policy-examples/fix-3.0/CMakeLists.txt
@@ -1,8 +1,4 @@
 cmake_minimum_required(VERSION 3.0)
 project(foo)

-cmake_policy(SET CMP0038 OLD)
-
 add_library(foo foo.cpp)
-
-target_link_libraries(foo foo) # BAD CODE! Make no sense
```

Final version:

```
cmake_minimum_required(VERSION 3.0)
project(foo)

add_library(foo foo.cpp)
```

You may notice that final version for both cases differs only in `cmake_minimum_required`:

```
--- /examples/policy-examples/fix-2.8/CMakeLists.txt
+++ /examples/policy-examples/fix-3.0/CMakeLists.txt
@@ -1,4 +1,4 @@
-cmake_minimum_required(VERSION 2.8)
+cmake_minimum_required(VERSION 3.0)
 project(foo)

 add_library(foo foo.cpp)
```

It means that there is no much sense in changing `cmake_minimum_required` without using any **new features**.

3.4.3. Summary

- Policies can be used to control CMake **behavior**
- Policies can be used to suppress warnings/errors
- `cmake_minimum_required` describe **features** you use in CMake code
- For backward compatibility new features can be protected with `if(CMAKE_VERSION ...)` directive