cmake-generator-expressions(7)

Contents

Introduction

Generator expressions are evaluated during build system generation to produce information specific to each build configuration.

Generator expressions are allowed in the context of many target properties, such as `LINK_LIBRARIES`, `INCLUDE_DIRECTORIES`, `COMPILE_DEFINITIONS` and others. They may also be used when using commands to populate those properties, such as `target_link_libraries()`, `target_include_directories()`, `target_compile_definitions()` and others.

This means that they enable conditional linking, conditional definitions used when compiling, and conditional include directories and more. The conditions may be based on the build configuration, target properties, platform information or any other queryable information.

Logical Expressions

Logical expressions are used to create conditional output. The basic expressions are the `0` and `1` expressions. Because other logical expressions evaluate to either `0` or `1`, they can be composed to create conditional output:

```
$<$<CONFIG:Debug>:DEBUG_MODE>
```

expands to `DEBUG_MODE` when the `Debug` configuration is used, and otherwise expands to nothing.

Available logical expressions are:

`$<BOOL:...>`
: `1` if the `...` is true, else `0`

`$<AND:?[,?]...>`
: `1` if all `?` are `1`, else `0`

  The `?` must always be either `0` or `1` in boolean expressions.

`$<OR:?[,?]...>`
: `0` if all `?` are `0`, else `1`

`$<NOT:?>`
: `0` if `?` is `1`, else `1`

`$<STREQUAL:a,b>`
: `1` if `a` is STREQUAL `b`, else `0`

`$<EQUAL:a,b>`
: `1` if `a` is EQUAL `b` in a numeric comparison, else `0`

`$<CONFIG:cfg>`
: `1` if config is `cfg`, else `0`. This is a case-insensitive comparison. The mapping in **`MAP_IMPORTED_CONFIG_<CONFIG>`** is also considered by this expression when it is evaluated on a property on an **`IMPORTED`** target.

`$<PLATFORM_ID:comp>`
: `1` if the CMake-id of the platform matches `comp`, otherwise `0`.

`$<C_COMPILER_ID:comp>`
: `1` if the CMake-id of the C compiler matches `comp`, otherwise `0`.

`$<CXX_COMPILER_ID:comp>`
: `1` if the CMake-id of the CXX compiler matches `comp`, otherwise `0`.

`$<VERSION_GREATER:v1,v2>`
: `1` if `v1` is a version greater than `v2`, else `0`.

`$<VERSION_LESS:v1,v2>`
: `1` if `v1` is a version less than `v2`, else `0`.

`$<VERSION_EQUAL:v1,v2>`
: `1` if `v1` is the same version as `v2`, else `0`.

`$<C_COMPILER_VERSION:ver>`
: `1` if the version of the C compiler matches `ver`, otherwise `0`.

`$<CXX_COMPILER_VERSION:ver>`
: `1` if the version of the CXX compiler matches `ver`, otherwise `0`.

`$<TARGET_POLICY:pol>`
: `1` if the policy `pol` was NEW when the ‘head’ target was created, else `0`. If the policy was not set, the warning message for the policy will be emitted. This generator expression only works for a subset of policies.

`$<COMPILE_FEATURES:feature[,feature]...>`
: `1` if all of the `feature` features are available for the ‘head’ target, and `0` otherwise. If this expression is used while evaluating the link implementation of a target and if any dependency transitively increases the required **`C_STANDARD`** or **`CXX_STANDARD`** for the ‘head’ target, an error is reported. See the **`cmake-compile-features(7)`** manual for information on compile features and a list of supported compilers.

`$<COMPILE_LANGUAGE:lang>`
: `1` when the language used for compilation unit matches `lang`, otherwise `0`. This expression used to specify compile options for source files of a particular language in a target. For example, to specify the use of the `-fno-exceptions` compile option

(compiler id checks elided):

```
add_executable(myapp main.cpp foo.c bar.cpp)
target_compile_options(myapp
  PRIVATE $<$<COMPILE_LANGUAGE:CXX>:-fno-exceptions>
)
```

This generator expression has limited use because it is not possible to use it with the Visual Studio generators. Portable buildsystems would not use this expression, and would create separate libraries for each source file language instead:

```
add_library(myapp_c foo.c)
add_library(myapp_cxx foo.c)
target_compile_options(myapp_cxx PUBLIC -fno-exceptions)
add_executable(myapp main.cpp)
target_link_libraries(myapp myapp_c myapp_cxx)
```

The `Makefile` and `Ninja` based generators can also use this expression to specify compile-language specific compile definitions and include directories:

```
add_executable(myapp main.cpp foo.c bar.cpp)
target_compile_definitions(myapp
  PRIVATE $<$<COMPILE_LANGUAGE:CXX>:COMPILING_CXX>
)
target_include_directories(myapp
  PRIVATE $<$<COMPILE_LANGUAGE:CXX>:/opt/foo/cxx_headers>
)
```

Informational Expressions

These expressions expand to some information. The information may be used directly, eg:

```
include_directories(/usr/include/$<CXX_COMPILER_ID>/)
```

expands to `/usr/include/GNU/` or `/usr/include/Clang/` etc, depending on the Id of the compiler.

These expressions may also may be combined with logical expressions:

```
$<$<VERSION_LESS:$<CXX_COMPILER_VERSION>,4.2.0>:OLD_COMPILER>
```

expands to `OLD_COMPILER` if the **`CMAKE_CXX_COMPILER_VERSION`** is less than 4.2.0.

Available informational expressions are:

`$<CONFIGURATION>`
: Configuration name. Deprecated. Use `CONFIG` instead.

`$<CONFIG>`
: Configuration name

`$<PLATFORM_ID>`

The CMake-id of the platform. See also the `CMAKE_SYSTEM_NAME` variable.

`$<C_COMPILER_ID>`
: The CMake-id of the C compiler used. See also the `CMAKE_<LANG>_COMPILER_ID` variable.

`$<CXX_COMPILER_ID>`
: The CMake-id of the CXX compiler used. See also the `CMAKE_<LANG>_COMPILER_ID` variable.

`$<C_COMPILER_VERSION>`
: The version of the C compiler used. See also the `CMAKE_<LANG>_COMPILER_VERSION` variable.

`$<CXX_COMPILER_VERSION>`
: The version of the CXX compiler used. See also the `CMAKE_<LANG>_COMPILER_VERSION` variable.

`$<TARGET_FILE:tgt>`
: Full path to main file (.exe, .so.1.2, .a) where `tgt` is the name of a target.

`$<TARGET_FILE_NAME:tgt>`
: Name of main file (.exe, .so.1.2, .a).

`$<TARGET_FILE_DIR:tgt>`
: Directory of main file (.exe, .so.1.2, .a).

`$<TARGET_LINKER_FILE:tgt>`
: File used to link (.a, .lib, .so) where `tgt` is the name of a target.

`$<TARGET_LINKER_FILE_NAME:tgt>`
: Name of file used to link (.a, .lib, .so).

`$<TARGET_LINKER_FILE_DIR:tgt>`
: Directory of file used to link (.a, .lib, .so).

`$<TARGET_SONAME_FILE:tgt>`
: File with soname (.so.3) where `tgt` is the name of a target.

`$<TARGET_SONAME_FILE_NAME:tgt>`
: Name of file with soname (.so.3).

`$<TARGET_SONAME_FILE_DIR:tgt>`
: Directory of with soname (.so.3).

`$<TARGET_PDB_FILE:tgt>`
: Full path to the linker generated program database file (.pdb) where `tgt` is the name of a target.

  See also the `PDB_NAME` and `PDB_OUTPUT_DIRECTORY` target properties and their configuration specific variants `PDB_NAME_<CONFIG>` and `PDB_OUTPUT_DIRECTORY_<CONFIG>`.

`$<TARGET_PDB_FILE_NAME:tgt>`
: Name of the linker generated program database file (.pdb).

`$<TARGET_PDB_FILE_DIR:tgt>`
: Directory of the linker generated program database file (.pdb).

`$<TARGET_PROPERTY:tgt,prop>`

Value of the property `prop` on the target `tgt`.

Note that `tgt` is not added as a dependency of the target this expression is evaluated on.

`$<TARGET_PROPERTY:prop>`
Value of the property `prop` on the target on which the generator expression is evaluated.

`$<INSTALL_PREFIX>`
Content of the install prefix when the target is exported via **`install(EXPORT)`** and empty otherwise.

`$<COMPILE_LANGUAGE>`
The compile language of source files when evaluating compile options. See the unary version for notes about portability of this generator expression.

Output Expressions

These expressions generate output, in some cases depending on an input. These expressions may be combined with other expressions for information or logical comparison:

```
-I$<JOIN:$<TARGET_PROPERTY:INCLUDE_DIRECTORIES>, -I>
```

generates a string of the entries in the **`INCLUDE_DIRECTORIES`** target property with each entry preceeded by `-I`. Note that a more-complete use in this situation would require first checking if the INCLUDE_DIRECTORIES property is non-empty:

```
$<$<BOOL:${prop}>:-I$<JOIN:${prop}, -I>>
```

where `${prop}` refers to a helper variable:

```
set(prop "$<TARGET_PROPERTY:INCLUDE_DIRECTORIES>")
```

Available output expressions are:

`$<0:...>`
Empty string (ignores `...`)

`$<1:...>`
Content of `...`

`$<JOIN:list,...>`
Joins the list with the content of `...`

`$<ANGLE-R>`
A literal `>`. Used to compare strings which contain a `>` for example.

`$<COMMA>`
A literal `,`. Used to compare strings which contain a `,` for example.

`$<SEMICOLON>`

A literal `;`. Used to prevent list expansion on an argument with `;`.

`$<TARGET_NAME:...>`
: Marks `...` as being the name of a target. This is required if exporting targets to multiple dependent export sets. The `...` must be a literal name of a target- it may not contain generator expressions.

`$<LINK_ONLY:...>`
: Content of `...` except when evaluated in a link interface while propagating *Transitive Usage Requirements*, in which case it is the empty string. Intended for use only in an **`INTERFACE_LINK_LIBRARIES`** target property, perhaps via the **`target_link_libraries()`** command, to specify private link dependencies without other usage requirements.

`$<INSTALL_INTERFACE:...>`
: Content of `...` when the property is exported using **`install(EXPORT)`**, and empty otherwise.

`$<BUILD_INTERFACE:...>`
: Content of `...` when the property is exported using **`export()`**, or when the target is used by another target in the same buildsystem. Expands to the empty string otherwise.

`$<LOWER_CASE:...>`
: Content of `...` converted to lower case.

`$<UPPER_CASE:...>`
: Content of `...` converted to upper case.

`$<MAKE_C_IDENTIFIER:...>`
: Content of `...` converted to a C identifier.

`$<TARGET_OBJECTS:objLib>`
: List of objects resulting from build of `objLib`. `objLib` must be an object of type `OBJECT_LIBRARY`. This expression may only be used in the sources of **`add_library()`** and **`add_executable()`** commands.