NAME

cmake-generator-expressions − CMake Generator Expressions

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.

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

Generator expressions have the form **$<...>**. To avoid confusion, this page deviates from most of the CMake documentation in that it omits angular brackets **<...>** around placeholders like **condition**, **string**, **target**, among others.

Generator expressions can be nested, as shown in most of the examples below.

BOOLEAN GENERATOR EXPRESSIONS

Boolean expressions evaluate to either **0** or **1**. They are typically used to construct the condition in a *conditional generator expression*.

Available boolean expressions are:

Logical Operators

$<BOOL:string>

Converts **string** to **0** or **1**. Evaluates to **0** if any of the following is true:

- **string** is empty,
- **string** is a case−insensitive equal of **0**, **FALSE**, **OFF**, **N**, **NO**, **IGNORE**, or **NOTFOUND**, or
- **string** ends in the suffix **−NOTFOUND** (case−sensitive).

Otherwise evaluates to **1**.

$<AND:conditions>

where **conditions** is a comma−separated list of boolean expressions. Evaluates to **1** if all conditions are **1**. Otherwise evaluates to **0**.

$<OR:conditions>

where **conditions** is a comma−separated list of boolean expressions. Evaluates to **1** if at least one of the conditions is **1**. Otherwise evaluates to **0**.

$<NOT:condition>

0 if **condition** is **1**, else **1**.

String Comparisons

$<STREQUAL:string1,string2>

1 if **string1** and **string2** are equal, else **0**. The comparison is case−sensitive. For a case−insensitive comparison, combine with a *string transforming generator expression*,

```
$<STREQUAL:$<UPPER_CASE:${foo}>,"BAR"> # "1" if ${foo} is any of "BAR", "
```

$<EQUAL:value1,value2>

1 if **value1** and **value2** are numerically equal, else **0**.

$<IN_LIST:string,list>
New in version 3.12.

1 if **string** is member of the semicolon−separated **list**, else **0**. Uses case−sensitive comparisons.

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

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

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

$<VERSION_LESS_EQUAL:v1,v2>
New in version 3.7.

1 if **v1** is a version less than or equal to **v2**, else **0**.

$<VERSION_GREATER_EQUAL:v1,v2>
New in version 3.7.

1 if **v1** is a version greater than or equal to **v2**, else **0**.

Variable Queries

$<TARGET_EXISTS:target>
New in version 3.12.

1 if **target** exists, else **0**.

$<CONFIG:cfgs>
1 if config is any one of the entries in comma−separated list **cfgs**, 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:platform_ids>
where **platform_ids** is a comma−separated list. **1** if the CMake's platform id matches any one of the entries in **platform_ids**, otherwise **0**. See also the**CMAKE_SYSTEM_N AME** variable.

$<C_COMPILER_ID:compiler_ids>
where **compiler_ids** is a comma−separated list. **1** if the CMake's compiler id of the C compiler matches any one of the entries in**compiler_ids**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<CXX_COMPILER_ID:compiler_ids>
where **compiler_ids** is a comma−separated list. **1** if the CMake's compiler id of the CXX compiler matches any one of the entries in **compiler_ids**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<CUDA_COMPILER_ID:compiler_ids>
New in version 3.15.

where **compiler_ids** is a comma−separated list. **1** if the CMake's compiler id of the CUDA compiler matches any one of the entries in **compiler_ids**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<OBJC_COMPILER_ID:compiler_ids>
New in version 3.16.

where **compiler_ids** is a comma−separated list. **1** if the CMake's compiler id of the Objective−C compiler matches any one of the entries in **compiler_ids**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<OBJCXX_COMPILER_ID:compiler_ids>
New in version 3.16.

where **compiler_ids** is a comma−separated list. **1** if the CMake's compiler id of the Objective−C++ compiler matches any one of the entries in **compiler_ids**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<Fortran_COMPILER_ID:compiler_ids>
where **compiler_ids** is a comma−separated list. **1** if the CMake's compiler id of the Fortran compiler matches any one of the entries in **compiler_ids**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<HIP_COMPILER_ID:compiler_ids>
where **compiler_ids** is a comma−separated list. **1** if the CMake's compiler id of the HIP compiler matches any one of the entries in**compiler_ids**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<ISPC_COMPILER_ID:compiler_ids>
New in version 3.19.

where **compiler_ids** is a comma−separated list. **1** if the CMake's compiler id of the ISPC compiler matches any one of the entries in **compiler_ids**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<C_COMPILER_VERSION:version>
1 if the version of the C compiler matches **version**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<CXX_COMPILER_VERSION:version>
1 if the version of the CXX compiler matches **version**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<CUDA_COMPILER_VERSION:version>
New in version 3.15.

1 if the version of the CXX compiler matches **version**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<OBJC_COMPILER_VERSION:version>
New in version 3.16.

1 if the version of the OBJC compiler matches **version**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<OBJCXX_COMPILER_VERSION:version>
New in version 3.16.

1 if the version of the OBJCXX compiler matches **version**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<Fortran_COMPILER_VERSION:version>
1 if the version of the Fortran compiler matches **version**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<HIP_COMPILER_VERSION:version>
1 if the version of the HIP compiler matches **version**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<ISPC_COMPILER_VERSION:version>
New in version 3.19.

1 if the version of the ISPC compiler matches **version**, otherwise **0**. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

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

$<COMPILE_FEATURES:features>
New in version 3.1.

where **features** is a comma−spearated list. Evaluates to **1** if all of the **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_LANG_AND_ID:language,compiler_ids>
New in version 3.15.

1 when the language used for compilation unit matches **language** and the CMake's compiler id of the language compiler matches any one of the entries in **compiler_ids**, otherwise **0**. This expression is a short form for the combination of **$<COMPILE_LANGUAGE:language>** and **$<LANG_COMPILER_ID:compiler_ids>**. This expression may be used to specify compile options, compile definitions, and include directories for source files of a particular language and compiler combination in a target. For example:

```
add_executable(myapp main.cpp foo.c bar.cpp zot.cu)
target_compile_definitions(myapp
  PRIVATE $<$<COMPILE_LANG_AND_ID:CXX,AppleClang,Clang>:COMPILING_CXX_WI
          $<$<COMPILE_LANG_AND_ID:CXX,Intel>:COMPILING_CXX_WITH_INTEL>
          $<$<COMPILE_LANG_AND_ID:C,Clang>:COMPILING_C_WITH_CLANG>
)
```

This specifies the use of different compile definitions based on both the compiler id and compilation language. This example will have a **COMPILING_CXX_WITH_CLANG** compile definition when Clang is the CXX compiler, and **COMPILING_CXX_WITH_INTEL** when Intel is the CXX compiler. Likewise when the C compiler is Clang it will only see the **COMPILING_C_WITH_CLANG** definition.

Without the **COMPILE_LANG_AND_ID** generator expression the same logic would be

expressed as:

```
target_compile_definitions(myapp
  PRIVATE $<$<AND:$<COMPILE_LANGUAGE:CXX>,$<CXX_COMPILER_ID:AppleClang,Cl
          $<$<AND:$<COMPILE_LANGUAGE:CXX>,$<CXX_COMPILER_ID:Intel>>:COMPI
          $<$<AND:$<COMPILE_LANGUAGE:C>,$<C_COMPILER_ID:Clang>>:COMPILING
)
```

$<COMPILE_LANGUAGE:languages>
New in version 3.3.

1 when the language used for compilation unit matches any of the entries in **languages**, otherwise **0**. This expression may be used to specify compile options, compile definitions, and include directories for source files of a particular language in a target. For example:

```
add_executable(myapp main.cpp foo.c bar.cpp zot.cu)
target_compile_options(myapp
  PRIVATE $<$<COMPILE_LANGUAGE:CXX>:-fno-exceptions>
)
target_compile_definitions(myapp
  PRIVATE $<$<COMPILE_LANGUAGE:CXX>:COMPILING_CXX>
          $<$<COMPILE_LANGUAGE:CUDA>:COMPILING_CUDA>
)
target_include_directories(myapp
  PRIVATE $<$<COMPILE_LANGUAGE:CXX,CUDA>:/opt/foo/headers>
)
```

This specifies the use of the **-fno-exceptions** compile option, **COMPILING_CXX** compile definition, and **cxx_headers** include directory for C++ only (compiler id checks elided). It also specifies a **COMPILING_CUDA** compile definition for CUDA.

Note that with Visual Studio Generators and **Xcode** there is no way to represent target-wide compile definitions or include directories separately for **C** and **CXX** languages. Also, with Visual Studio Generators there is no way to represent target-wide flags separately for **C** and **CXX** languages. Under these generators, expressions for both C and C++ sources will be evaluated using **CXX** if there are any C++ sources and otherwise using **C**. A workaround is to create separate libraries for each source file language instead:

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

$<LINK_LANG_AND_ID:language,compiler_ids>
New in version 3.18.

1 when the language used for link step matches **language** and the CMake's compiler id of the language linker matches any one of the entries in **compiler_ids**, otherwise **0**. This expression is a short form for the combination of **$<LINK_LANGUAGE:language>** and **$<LANG_COMPILER_ID:compiler_ids>**. This expression may be used to specify link libraries, link options, link directories and link dependencies of a particular language and linker combination in a target. For example:

```
add_library(libC_Clang ...)
add_library(libCXX_Clang ...)
add_library(libC_Intel ...)
add_library(libCXX_Intel ...)

add_executable(myapp main.c)
if (CXX_CONFIG)
  target_sources(myapp PRIVATE file.cxx)
endif()
target_link_libraries(myapp
  PRIVATE $<$<LINK_LANG_AND_ID:CXX,Clang,AppleClang>:libCXX_Clang>
          $<$<LINK_LANG_AND_ID:C,Clang,AppleClang>:libC_Clang>
          $<$<LINK_LANG_AND_ID:CXX,Intel>:libCXX_Intel>
          $<$<LINK_LANG_AND_ID:C,Intel>:libC_Intel>)
```

This specifies the use of different link libraries based on both the compiler id and link language. This example will have target **libCXX_Clang** as link dependency when **Clang** or **AppleClang** is the **CXX** linker, and **libCXX_Intel** when **Intel** is the **CXX** linker. Likewise when the **C** linker is **Clang** or **AppleClang**, target **libC_Clang** will be added as link dependency and **libC_Intel** when **Intel** is the **C** linker.

See *the note related to* **$<LINK_LANGUAGE:language>** for constraints about the usage of this generator expression.

$<LINK_LANGUAGE:languages>
New in version 3.18.

1 when the language used for link step matches any of the entries in **languages**, otherwise **0**. This expression may be used to specify link libraries, link options, link directories and link dependencies of a particular language in a target. For example:

```
add_library(api_C ...)
add_library(api_CXX ...)
add_library(api INTERFACE)
target_link_options(api INTERFACE $<$<LINK_LANGUAGE:C>:-opt_c>
                                  $<$<LINK_LANGUAGE:CXX>:-opt_cxx>)
target_link_libraries(api INTERFACE $<$<LINK_LANGUAGE:C>:api_C>
                                    $<$<LINK_LANGUAGE:CXX>:api_CXX>)

add_executable(myapp1 main.c)
target_link_options(myapp1 PRIVATE api)

add_executable(myapp2 main.cpp)
target_link_options(myapp2 PRIVATE api)
```

This specifies to use the **api** target for linking targets **myapp1** and **myapp2**. In practice, **myapp1** will link with target **api_C** and option **-opt_c** because it will use **C** as link language. And **myapp2** will link with **api_CXX** and option **-opt_cxx** because **CXX** will be the link language.

NOTE:
To determine the link language of a target, it is required to collect, transitively, all the targets which will be linked to it. So, for link libraries properties, a double evaluation will be done. During the first evaluation, **$<LINK_LANGUAGE:..>** expressions will always return **0**. The link language computed after this first pass will be used to do the second pass. To avoid

inconsistency, it is required that the second pass do not change the link language. Moreover, to avoid unexpected side−effects, it is required to specify complete entities as part of the **$<LINK_LANGUAGE:..>** expression. For example:

```
add_library(lib STATIC file.cxx)
add_library(libother STATIC file.c)

# bad usage
add_executable(myapp1 main.c)
target_link_libraries(myapp1 PRIVATE lib$<$<LINK_LANGUAGE:C>:other>)

# correct usage
add_executable(myapp2 main.c)
target_link_libraries(myapp2 PRIVATE $<$<LINK_LANGUAGE:C>:libother>)
```

In this example, for **myapp1**, the first pass will, unexpectedly, determine that the link language is **CXX** because the evaluation of the generator expression will be an empty string so **myapp1** will depends on target **lib** which is **C++**. On the contrary, for **myapp2**, the first evaluation will give **C** as link language, so the second pass will correctly add target **libother** as link dependency.

$<DEVICE_LINK:list>
New in version 3.18.

Returns the list if it is the device link step, an empty list otherwise. The device link step is controlled by **CUDA_SEPARABLE_COMPILATION** and **CUDA_RESOLVE_DEVICE_SYMBOLS** properties and policy **CMP0105**. This expression can only be used to specify link options.

$<HOST_LINK:list>
New in version 3.18.

Returns the list if it is the normal link step, an empty list otherwise. This expression is mainly useful when a device link step is also involved (see **$<DEVICE_LINK:list>** generator expression). This expression can only be used to specify link options.

STRING-VALUED GENERATOR EXPRESSIONS

These expressions expand to some string. For example,

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

expands to **/usr/include/GNU/** or **/usr/include/Clang/** etc, depending on the compiler identifier.

String−valued expressions may also be combined with other expressions. Here an example for a string−valued expression within a boolean expressions within a conditional expression:

```
$<$<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.

And here two nested string−valued expressions:

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

generates a string of the entries in the **INCLUDE_DIRECTORIES** target property with each entry

preceded by **−I**.

Expanding on the previous example, if one first wants to check if the **INCLUDE_DIRECTORIES** property is non−empty, then it is advisable to introduce a helper variable to keep the code readable:

```
set(prop "$<TARGET_PROPERTY:INCLUDE_DIRECTORIES>") # helper variable
$<$<BOOL:${prop}>:-I$<JOIN:${prop}, -I>>
```

The following string−valued generator expressions are available:

Escaped Characters

String literals to escape the special meaning a character would otherwise have:

$<ANGLE−R>
A literal >. Used for example to compare strings that contain a >.

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

$<SEMICOLON>
A literal **;**. Used to prevent list expansion on an argument with **;**.

Conditional Expressions

Conditional generator expressions depend on a boolean condition that must be **0** or **1**.

$<condition:true_string>
Evaluates to **true_string** if **condition** is **1**. Otherwise evaluates to the empty string.

$<IF:condition,true_string,false_string>
New in version 3.8.

Evaluates to **true_string** if **condition** is **1**. Otherwise evaluates to **false_string**.

Typically, the **condition** is a *boolean generator expression*. For instance,

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

expands to **DEBUG_MODE** when the **Debug** configuration is used, and otherwise expands to the empty string.

String Transformations

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

$<REMOVE_DUPLICATES:list>
New in version 3.15.

Removes duplicated items in the given **list**.

$<FILTER:list,INCLUDE|EXCLUDE,regex>
New in version 3.15.

Includes or removes items from **list** that match the regular expression **regex**.

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

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

$<GENEX_EVAL:expr>
New in version 3.12.

Content of **expr** evaluated as a generator expression in the current context. This enables consumption of generator expressions whose evaluation results itself in generator expressions.

$<TARGET_GENEX_EVAL:tgt,expr>
New in version 3.12.

Content of **expr** evaluated as a generator expression in the context of **tgt** target. This enables consumption of custom target properties that themselves contain generator expressions.

Having the capability to evaluate generator expressions is very useful when you want to manage custom properties supporting generator expressions. For example:

```
add_library(foo ...)

set_property(TARGET foo PROPERTY
  CUSTOM_KEYS $<$<CONFIG:DEBUG>:FOO_EXTRA_THINGS>
)

add_custom_target(printFooKeys
  COMMAND ${CMAKE_COMMAND} -E echo $<TARGET_PROPERTY:foo,CUSTOM_KEYS>
)
```

This naive implementation of the **printFooKeys** custom command is wrong because **CUSTOM_KEYS** target property is not evaluated and the content is passed as is (i.e. **$<$<CONFIG:DEBUG>:FOO_EXTRA_THINGS>**).

To have the expected result (i.e. **FOO_EXTRA_THINGS** if config is **Debug**), it is required to evaluate the output of **$<TARGET_PROPERTY:foo,CUSTOM_KEYS>**:

```
add_custom_target(printFooKeys
  COMMAND ${CMAKE_COMMAND} -E
    echo $<TARGET_GENEX_EVAL:foo,$<TARGET_PROPERTY:foo,CUSTOM_KEYS>>
)
```

Variable Queries

$<CONFIG>
Configuration name.

$<CONFIGURATION>
Configuration name. Deprecated since CMake 3.0. Use **CONFIG** instead.

$<PLATFORM_ID>
The current system's CMake platform id. See also the **CMAKE_SYSTEM_NAME** variable.

$<C_COMPILER_ID>
The CMake's compiler id of the C compiler used. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<CXX_COMPILER_ID>
The CMake's compiler id of the CXX compiler used. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<CUDA_COMPILER_ID>
The CMake's compiler id of the CUDA compiler used. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<OBJC_COMPILER_ID>
New in version 3.16.

The CMake's compiler id of the OBJC compiler used. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<OBJCXX_COMPILER_ID>
New in version 3.16.

The CMake's compiler id of the OBJCXX compiler used. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<Fortran_COMPILER_ID>
The CMake's compiler id of the Fortran compiler used. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<HIP_COMPILER_ID>
The CMake's compiler id of the HIP compiler used. See also the **CMAKE_<LANG>_COMPILER_ID** variable.

$<ISPC_COMPILER_ID>
New in version 3.19.

The CMake's compiler id of the ISPC 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.

$<CUDA_COMPILER_VERSION>
The version of the CUDA compiler used. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<OBJC_COMPILER_VERSION>
New in version 3.16.

The version of the OBJC compiler used. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<OBJCXX_COMPILER_VERSION>
New in version 3.16.

The version of the OBJCXX compiler used. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<Fortran_COMPILER_VERSION>
The version of the Fortran compiler used. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<HIP_COMPILER_VERSION>
The version of the HIP compiler used. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<ISPC_COMPILER_VERSION>
New in version 3.19.

The version of the ISPC compiler used. See also the **CMAKE_<LANG>_COMPILER_VERSION** variable.

$<COMPILE_LANGUAGE>
New in version 3.3.

The compile language of source files when evaluating compile options. See *the related boolean expression* **$<COMPILE_LANGUAGE:language>** for notes about the portability of this generator expression.

$<LINK_LANGUAGE>
New in version 3.18.

The link language of target when evaluating link options. See *the related boolean expression* **$<LINK_LANGUAGE:language>** for notes about the portability of this generator expression.

NOTE:
This generator expression is not supported by the link libraries properties to avoid side−effects due to the double evaluation of these properties.

Target−Dependent Queries
These queries refer to a target **tgt**. This can be any runtime artifact, namely:

- an executable target created by **add_executable()**
- a shared library target (**.so**, **.dll** but not their **.lib** import library) created by **add_library()**
- a static library target created by **add_library()**

In the following, "the **tgt** filename" means the name of the **tgt** binary file. This has to be distinguished from "the target name", which is just the string **tgt**.

$<TARGET_NAME_IF_EXISTS:tgt>
New in version 3.12.

The target name **tgt** if the target exists, an empty string otherwise.

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

$<TARGET_FILE:tgt>
Full path to the **tgt** binary file.

$<TARGET_FILE_BASE_NAME:tgt>
New in version 3.15.

Base name of **tgt**, i.e. **$<TARGET_FILE_NAME:tgt>** without prefix and suffix. For example, if the **tgt** filename is **libbase.so**, the base name is **base**.

See also the **OUTPUT_NAME**, **ARCHIVE_OUTPUT_NAME**, **LIBRARY_OUTPUT_NAME** and **RUNTIME_OUTPUT_NAME** target properties and their configuration specific variants **OUTPUT_NAME_<CONFIG>**, **ARCHIVE_OUTPUT_NAME_<CONFIG>**, **LIBRARY_OUTPUT_NAME_<CONFIG>** and **RUNTIME_OUTPUT_NAME_<CONFIG>**.

The **<CONFIG>_POSTFIX** and **DEBUG_POSTFIX** target properties can also be considered.

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

$<TARGET_FILE_PREFIX:tgt>
New in version 3.15.

Prefix of the **tgt** filename (such as **lib**).

See also the **PREFIX** target property.

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

$<TARGET_FILE_SUFFIX:tgt>
New in version 3.15.

Suffix of the **tgt** filename (extension such as **.so** or **.exe**).

See also the **SUFFIX** target property.

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

$<TARGET_FILE_NAME:tgt>
The **tgt** filename.

Note that **tgt** is not added as a dependency of the target this expression is evaluated on (see policy **CMP0112**).

$<TARGET_FILE_DIR:tgt>
Directory of the **tgt** binary file.

Note that **tgt** is not added as a dependency of the target this expression is evaluated on (see policy **CMP0112**).

$<TARGET_LINKER_FILE:tgt>
File used when linking to the **tgt** target. This will usually be the library that**tgt** represents (**.a**, **.lib**, **.so**), but for a shared library on DLL platforms, it would be the **.lib** import library associated with the DLL.

$<TARGET_LINKER_FILE_BASE_NAME:tgt>
New in version 3.15.

Base name of file used to link the target **tgt**, i.e. **$<TARGET_LINKER_FILE_NAME:tgt>** without prefix and suffix. For example, if target file name is **libbase.a**, the base name is **base**.

See also the **OUTPUT_NAME**, **ARCHIVE_OUTPUT_NAME**, and **LIBRARY_OUTPUT_NAME** target properties and their configuration specific variants

OUTPUT_NAME_<CONFIG>, **ARCHIVE_OUTPUT_NAME_<CONFIG>** and **LIBRARY_OUTPUT_NAME_<CONFIG>**.

The **<CONFIG>_POSTFIX** and **DEBUG_POSTFIX** target properties can also be considered.

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

$<TARGET_LINKER_FILE_PREFIX:tgt>
New in version 3.15.

Prefix of file used to link target **tgt**.

See also the **PREFIX** and **IMPORT_PREFIX** target properties.

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

$<TARGET_LINKER_FILE_SUFFIX:tgt>
New in version 3.15.

Suffix of file used to link where **tgt** is the name of a target.

The suffix corresponds to the file extension (such as ".so" or ".lib").

See also the **SUFFIX** and **IMPORT_SUFFIX** target properties.

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

$<TARGET_LINKER_FILE_NAME:tgt>
Name of file used to link target **tgt**.

Note that **tgt** is not added as a dependency of the target this expression is evaluated on (see policy **CMP0112**).

$<TARGET_LINKER_FILE_DIR:tgt>
Directory of file used to link target **tgt**.

Note that **tgt** is not added as a dependency of the target this expression is evaluated on (see policy **CMP0112**).

$<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**).

Note that **tgt** is not added as a dependency of the target this expression is evaluated on (see policy **CMP0112**).

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

Note that **tgt** is not added as a dependency of the target this expression is evaluated on (see policy **CMP0112**).

$<TARGET_PDB_FILE:tgt>
New in version 3.1.

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_BASE_NAME:tgt>
New in version 3.15.

Base name of the linker generated program database file (.pdb) where **tgt** is the name of a target.

The base name corresponds to the target PDB file name (see **$<TARGET_PDB_FILE_NAME:tgt>**) without prefix and suffix. For example, if target file name is **base.pdb**, the base name is **base**.

See also the **PDB_NAME** target property and its configuration specific variant **PDB_NAME_<CONFIG>**.

The **<CONFIG>_POSTFIX** and **DEBUG_POSTFIX** target properties can also be considered.

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

$<TARGET_PDB_FILE_NAME:tgt>
New in version 3.1.

Name of the linker generated program database file (.pdb).

Note that **tgt** is not added as a dependency of the target this expression is evaluated on (see policy **CMP0112**).

$<TARGET_PDB_FILE_DIR:tgt>
New in version 3.1.

Directory of the linker generated program database file (.pdb).

Note that **tgt** is not added as a dependency of the target this expression is evaluated on (see policy **CMP0112**).

$<TARGET_BUNDLE_DIR:tgt>
New in version 3.9.

Full path to the bundle directory (**my.app**, **my.framework**, or **my.bundle**) where **tgt** is the name of a target.

Note that **tgt** is not added as a dependency of the target this expression is evaluated on (see policy **CMP0112**).

$<TARGET_BUNDLE_CONTENT_DIR:tgt>
New in version 3.9.

Full path to the bundle content directory where **tgt** is the name of a target. For the macOS SDK it leads to **my.app/Contents**, **my.framework**, or **my.bundle/Contents**. For all other SDKs (e.g.

iOS) it leads to **my.app**, **my.framework**, or **my.bundle** due to the flat bundle structure.

Note that **tgt** is not added as a dependency of the target this expression is evaluated on (see policy **CMP0112**).

$<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 for which the expression is being evaluated. Note that for generator expressions in Target Usage Requirements this is the consuming target rather than the target specifying the requirement.

$<TARGET_RUNTIME_DLLS:tgt>
New in version 3.21.

List of DLLs that the target depends on at runtime. This is determined by the locations of all the **SHARED** and **MODULE** targets in the target's transitive dependencies. Using this generator expression on targets other than executables, **SHARED** libraries, and **MODULE** libraries is an error. On non−DLL platforms, it evaluates to an empty string.

This generator expression can be used to copy all of the DLLs that a target depends on into its output directory in a **POST_BUILD** custom command. For example:

```
find_package(foo CONFIG REQUIRED) # package generated by install(EXPORT)

add_executable(exe main.c)
target_link_libraries(exe PRIVATE foo::foo foo::bar)
add_custom_command(TARGET exe POST_BUILD
  COMMAND ${CMAKE_COMMAND} −E copy $<TARGET_RUNTIME_DLLS:exe> $<TARGET_FI
  COMMAND_EXPAND_LISTS
  )
```

NOTE:
Imported Targets are supported only if they know the location of their **.dll** files. An imported **SHARED** or **MODULE** library must have **IMPORTED_LOCATION** set to its **.dll** file. See the add_library imported libraries section for details. Many Find Modules produce imported targets with the **UNKNOWN** type and therefore will be ignored.

$<INSTALL_PREFIX>
Content of the install prefix when the target is exported via **install(EXPORT)**, or when evaluated in the **INSTALL_NAME_DIR** property or the **INSTALL_NAME_DIR** argument of **install(RUNTIME_DEPENDENCY_SET)**, and empty otherwise.

Output−Related Expressions

$<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:...>
New in version 3.1.

Content of **...** except when evaluated in a link interface while propagating Target 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.

$<MAKE_C_IDENTIFIER:...>
Content of **...** converted to a C identifier. The conversion follows the same behavior as **string(MAKE_C_IDENTIFIER)**.

$<TARGET_OBJECTS:objLib>
New in version 3.1.

List of objects resulting from build of **objLib**.

$<SHELL_PATH:...>
New in version 3.4.

Content of **...** converted to shell path style. For example, slashes are converted to backslashes in Windows shells and drive letters are converted to posix paths in MSYS shells. The **...** must be an absolute path.

New in version 3.14: The **...** may be a semicolon−separated list of paths, in which case each path is converted individually and a result list is generated using the shell path separator (**:** on POSIX and **;** on Windows). Be sure to enclose the argument containing this genex in double quotes in CMake source code so that **;** does not split arguments.

$<OUTPUT_CONFIG:...>
New in version 3.20.

Only valid in **add_custom_command()** and **add_custom_target()** as the outer−most generator expression in an argument. With the **Ninja Multi−Config** generator, generator expressions in **...** are evaluated using the custom command's "output config". With other generators, the content of **...** is evaluated normally.

$<COMMAND_CONFIG:...>
New in version 3.20.

Only valid in **add_custom_command()** and **add_custom_target()** as the outer−most generator expression in an argument. With the **Ninja Multi−Config** generator, generator expressions in **...** are evaluated using the custom command's "command config". With other generators, the content of **...** is evaluated normally.

DEBUGGING

Since generator expressions are evaluated during generation of the buildsystem, and not during processing of **CMakeLists.txt** files, it is not possible to inspect their result with the **message()** command.

One possible way to generate debug messages is to add a custom target,

```
add_custom_target(genexdebug COMMAND ${CMAKE_COMMAND} −E echo "$<...>")
```

The shell command **make genexdebug** (invoked after execution of **cmake**) would then print the result of **$<...>**.

Another way is to write debug messages to a file:

```
file(GENERATE OUTPUT filename CONTENT "$<...>")
```

COPYRIGHT