Feature names are case-sensitive and may only contain letters, numbers

and underscores. Feature names defined in all uppercase are reserved for

CMake's own built-in features (see `Predefined Features`\_ further below).

Some aspects of feature behavior can be defined by the

:variable:`CMAKE\_<LANG>\_LINK\_LIBRARY\_<FEATURE>\_ATTRIBUTES` and

:variable:`CMAKE\_LINK\_LIBRARY\_<FEATURE>\_ATTRIBUTES` variables.

Feature Definitions

^^^^^^^^^^^^^^^^^^^

A library feature definition is a list that contains one or three elements:

::

[<PREFIX>] <LIBRARY\_EXPRESSION> [<SUFFIX>]

When ``<PREFIX>`` and ``<SUFFIX>`` are specified, they precede and follow

respectively the whole list of libraries specified in the

:genex:`LINK\_LIBRARY` expression, not each library item individually.

There is no guarantee that the list of specified libraries will be kept

grouped together though, so the ``<PREFIX>`` and ``<SUFFIX>`` may appear

more than once if the library list is reorganized by CMake to satisfy other

constraints. This means constructs like ``--start-group`` and ``--end-group``,

as supported by the GNU ``ld`` linker, cannot be used in this way. The

:genex:`LINK\_GROUP` generator expression should be used instead for such

constructs.

``<LIBRARY\_EXPRESSION>`` is used to specify the pattern for constructing the

corresponding fragment on the linker command line for each library.

The following placeholders can be used in the expression:

\* ``<LIBRARY>`` is expanded to the full path to the library for CMake targets,

or to a platform-specific value based on the item otherwise (the same as

``<LINK\_ITEM>`` on Windows, or the library base name for other platforms).

\* ``<LINK\_ITEM>`` is expanded to how the library would normally be linked on

the linker command line.

\* ``<LIB\_ITEM>`` is expanded to the full path to the library for CMake targets,

or the item itself exactly as specified in the ``<LIBRARY\_EXPRESSION>``

otherwise.

In addition to the above, it is possible to have one pattern for paths

(CMake targets and external libraries specified with file paths) and another

for other items specified by name only. The ``PATH{}`` and ``NAME{}`` wrappers

can be used to provide the expansion for those two cases, respectively.

When wrappers are used, both must be present. For example:

.. code-block:: cmake

set(CMAKE\_LINK\_LIBRARY\_USING\_weak\_library

"PATH{-weak\_library <LIBRARY>}NAME{LINKER:-weak-l<LIB\_ITEM>}"

)

For all three elements of this variable (``<PREFIX>``, ``<LIBRARY\_EXPRESSION>``,

and ``<SUFFIX>``), the ``LINKER:`` prefix can be used.

.. include:: ../command/LINK\_OPTIONS\_LINKER.txt

:start-line: 3

Examples

^^^^^^^^

Loading a whole static library

""""""""""""""""""""""""""""""

A common need is to prevent the linker from discarding any symbols from a

static library. Different linkers use different syntax for achieving this.

The following example shows how this may be implemented for some linkers.

Note that this is for illustration purposes only. Projects should use the

built-in ``WHOLE\_ARCHIVE`` feature instead (see `Predefined Features`\_), which

provides a more complete and more robust implementation of this functionality.

.. code-block:: cmake

set(CMAKE\_C\_LINK\_LIBRARY\_USING\_load\_archive\_SUPPORTED TRUE)

if(CMAKE\_C\_COMPILER\_ID STREQUAL "AppleClang")

set(CMAKE\_C\_LINK\_LIBRARY\_USING\_load\_archive "-force\_load <LIB\_ITEM>")

elseif(CMAKE\_C\_COMPILER\_ID STREQUAL "GNU" AND CMAKE\_SYSTEM\_NAME STREQUAL "Linux")

set(CMAKE\_C\_LINK\_LIBRARY\_USING\_load\_archive

"LINKER:--push-state,--whole-archive"

"<LINK\_ITEM>"

"LINKER:--pop-state"

)

elseif(CMAKE\_C\_COMPILER\_ID STREQUAL "MSVC")

set(CMAKE\_C\_LINK\_LIBRARY\_USING\_load\_archive "/WHOLEARCHIVE:<LIBRARY>")

else()

# feature not yet supported for the other environments

set(CMAKE\_C\_LINK\_LIBRARY\_USING\_load\_archive\_SUPPORTED FALSE)

endif()

add\_library(lib1 STATIC ...)

add\_library(lib2 SHARED ...)

if(CMAKE\_C\_LINK\_LIBRARY\_USING\_load\_archive\_SUPPORTED)

# The -force\_load Apple linker option requires a file name

set(external\_lib

"$<IF:$<LINK\_LANG\_AND\_ID:C,AppleClang>,libexternal.a,external>"

)

target\_link\_libraries(lib2 PRIVATE

"$<LINK\_LIBRARY:load\_archive,lib1,${external\_lib}>"

)

else()

target\_link\_libraries(lib2 PRIVATE lib1 external)

endif()

CMake will generate the following link expressions:

\* ``AppleClang``: ``-force\_load /path/to/lib1.a -force\_load libexternal.a``

\* ``GNU``: ``-Wl,--push-state,--whole-archive /path/to/lib1.a -lexternal -Wl,--pop-state``

\* ``MSVC``: ``/WHOLEARCHIVE:/path/to/lib1.lib /WHOLEARCHIVE:external.lib``

Linking a library as weak

"""""""""""""""""""""""""

On macOS, it is possible to link a library in weak mode (the library and all

references are marked as weak imports). Different flags must be used for a

library specified by file path compared to one specified by name.

This constraint can be solved using ``PATH{}`` and ``NAME{}`` wrappers.

Again, the following example shows how this may be implemented for some

linkers, but it is for illustration purposes only. Projects should use the

built-in ``WEAK\_FRAMEWORK`` or ``WEAK\_LIBRARY`` features instead (see

`Predefined Features`\_), which provide more complete and more robust

implementations of this functionality.

.. code-block:: cmake

if (CMAKE\_C\_COMPILER\_ID STREQUAL "AppleClang")

set(CMAKE\_LINK\_LIBRARY\_USING\_weak\_library

"PATH{-weak\_library <LIBRARY>}NAME{LINKER:-weak-l<LIB\_ITEM>}"

)

set(CMAKE\_LINK\_LIBRARY\_USING\_weak\_library\_SUPPORTED TRUE)

endif()

add\_library(lib SHARED ...)

add\_executable(main ...)

if(CMAKE\_LINK\_LIBRARY\_USING\_weak\_library\_SUPPORTED)

target\_link\_libraries(main PRIVATE "$<LINK\_LIBRARY:weak\_library,lib,external>")

else()

target\_link\_libraries(main PRIVATE lib external)

endif()

CMake will generate the following linker command line fragment when linking

``main`` using the ``AppleClang`` toolchain:

``-weak\_library /path/to/lib -Xlinker -weak-lexternal``.

Predefined Features

^^^^^^^^^^^^^^^^^^^

The following built-in library features are pre-defined by CMake:

.. include:: LINK\_LIBRARY\_PREDEFINED\_FEATURES.txt