# CMake/Tutorials/Exporting and Importing Targets

From KitwarePublic
< CMake | Tutorials

## Contents

CMake 2.6 introduced support for exporting targets from one project and importing them into another. The main feature allowing this functionality is the notion of an IMPORTED target. Here we present imported targets and then show how CMake files may be generated by a project to export its targets for use by other projects.

# Importing Targets

Imported targets are used to convert files outside the project on disk into logical targets inside a CMake project. They are created using the IMPORTED option to add_executable and add_library commands. No build files are generated for imported targets. They are useful simply for convenient, flexible reference to outside executables and libraries.

Consider the following example which creates and uses an IMPORTED executable target:

```
add_executable(generator IMPORTED)                                    # 1
set_property(TARGET generator PROPERTY IMPORTED_LOCATION "/path/to/some_generator")   # 2

set(GENERATED_SRC ${CMAKE_CURRENT_BINARY_DIR}/generated.c)
add_custom_command(
  OUTPUT ${GENERATED_SRC}
  COMMAND generator ${GENERATED_SRC}                                  # 3
  )

add_executable(myexe src1.c src2.c ${GENERATED_SRC})
```

Line #1 creates a new CMake target called "generator". Line #2 tells CMake the location of the file on disk to import. Line #3 references the target in a custom command. The generated build system will contain a command line such as

```
/path/to/some_generator /project/binary/dir/generated.c
```

in the rule to generate the source file.

Libraries may also be used through imported targets:

```
add_library(foo STATIC IMPORTED)
set_property(TARGET foo PROPERTY IMPORTED_LOCATION /path/to/libfoo.a)
add_executable(myexe src1.c src2.c)
target_link_libraries(myexe foo)
```

The generated build system will contain a command line such as

```
... -o myexe /path/to/libfoo.a ...
```

in the rule to link myexe. On Windows a .dll and its .lib import library may be imported together:

```
add_library(bar SHARED IMPORTED)
set_property(TARGET bar PROPERTY IMPORTED_LOCATION c:/path/to/bar.dll)
set_property(TARGET bar PROPERTY IMPORTED_IMPLIB c:/path/to/bar.lib)
add_executable(myexe src1.c src2.c)
target_link_libraries(myexe bar)
```

The generated build system will contain a command line such as

```
... -o myexe.exe c:/path/to/bar.lib ...
```

in the rule to link myexe. A library with multiple configurations may be imported with a single target:

```
add_library(foo STATIC IMPORTED)
set_property(TARGET foo PROPERTY IMPORTED_LOCATION_RELEASE c:/path/to/foo.lib)
```

```
set_property(TARGET foo PROPERTY IMPORTED_LOCATION_DEBUG   c:/path/to/foo_d.lib)
add_executable(myexe src1.c src2.c)
target_link_libraries(myexe foo)
```

The generated build system will link myexe to foo.lib when it is built in the release configuration and foo_d.lib when built in the debug configuration.

In some cases, such as when using the result of a find_library command to create an imported target, the type of the library (static or shared) is not known. As of CMake 2.6.2 one may specify this explicitly:

```
find_library(FOO_LIBRARY NAMES foo)
add_library(foo UNKNOWN IMPORTED)
set_property(TARGET foo PROPERTY IMPORTED_LOCATION "${FOO_LIBRARY}")
add_executable(myexe src1.c src2.c)
target_link_libraries(myexe foo)
```

In this case CMake does not know the library type, so it just puts the library on the link line as-is. Therefore on Windows there is no special treatment for a shared library. The runtime library (foo.dll) need not be found. The import library (foo.lib) is specified by the IMPORTED_LOCATION property, not the IMPORTED_IMPLIB property.

# Exporting Targets

Imported targets on their own are useful, but they still require the project that imports targets to know the locations of the target files on disk. The real power of this feature is when the project providing the target files also provides a file to help import them.

## Exporting from an Installation Tree

The install(TARGETS) and install(EXPORT) commands work together to install a target and a file to help import it. For example, the code

```
add_executable(generator generator.c)
install(TARGETS generator DESTINATION lib/myproj/generators EXPORT myproj-targets)
install(EXPORT myproj-targets DESTINATION lib/myproj)
```

installs the files

```
<prefix>/lib/myproj/generators/generator
<prefix>/lib/myproj/myproj-targets.cmake
```

The myproj-targets.cmake file contains code such as

```
add_executable(generator IMPORTED)
set_property(
  TARGET generator
  PROPERTY IMPORTED_LOCATION ${PREFIX}/lib/myproj/generators/generator
  )
```

(in practice the ${PREFIX} is computed relative to the file location automatically). An outside project may now contain code such as

```
include(${PREFIX}/lib/myproj/myproj-targets.cmake)        # 1
set(GENERATED_SRC ${CMAKE_CURRENT_BINARY_DIR}/generated.c)
add_custom_command(
  OUTPUT ${GENERATED_SRC}
  COMMAND generator ${GENERATED_SRC}              # 2
  )
add_executable(myexe src1.c src2.c ${GENERATED_SRC})
```

Line #1 loads the target import script. The script may import any number of targets. Their locations are computed relative to the script location so the install tree may be moved around. The importing project need only find the import script (see the packaging section of this document for how this might be done). Line #2 references the generator executable in a custom command. The resulting build system will run the executable from its installed location.

Libraries may also be exported and imported:

```
add_library(foo STATIC foo1.c)
install(TARGETS foo DESTINATION lib EXPORT myproj-targets)
install(EXPORT myproj-targets DESTINATION lib/myproj)
```

This installs the library and an import file referencing it. Outside projects may simply write

```
include(${PREFIX}/lib/myproj/myproj-targets.cmake)
add_executable(myexe src1.c)
```

```
target_link_libraries(myexe foo)
```

and the executable will be linked to the library foo exported and installed by the original project.

Any number of target installations may be associated with the same export name. The export names are considered global so any directory may contribute a target installation. Only one call to the install(EXPORT) command is needed to install an import file that references all targets. Both of the examples above may be combined into a single export, even if they are in different subdirectories of the project:

```
# A/CMakeLists.txt
add_executable(generator generator.c)
install(TARGETS generator DESTINATION lib/myproj/generators EXPORT myproj-targets)

# B/CMakeLists.txt
add_library(foo STATIC foo1.c)
install(TARGETS foo DESTINATION lib EXPORT myproj-targets)


# Top CMakeLists.txt
add_subdirectory(A)
add_subdirectory(B)
install(EXPORT myproj-targets DESTINATION lib/myproj)
```

# Exporting from a Build Tree

Typically projects are built and installed before being used by an outside project. However in some cases it is desirable to export targets directly from a build tree. The targets may then be used by an outside project that references the build tree with no installation involved.

The export command is used to generate a file exporting targets from a project build tree. For example, the code

```
add_executable(generator generator.c)
export(TARGETS generator FILE myproj-exports.cmake)
```

will create a file in the project build tree called myproj-exports.cmake that contains code such as

```
add_executable(generator IMPORTED)
set_property(TARGET generator PROPERTY IMPORTED_LOCATION "/path/to/build/tree/generator")
```

This file may be loaded by an outside project that is aware of the project build tree in order to use the executable to generate a source file.

An example application of this feature is for building a generator executable on a host platform when cross compiling. The project containing the generator executable may be built on the host platform and then the project that is being cross-compiled for another platform may load it.

Retrieved from "https://cmake.org/Wiki/index.php?title=CMake/Tutorials/Exporting_and_Importing_Targets&oldid=41001"

---

- This page was last modified on 21 June 2011, at 14:32.
- Content is available under Attribution2.5 unless otherwise noted.