

**ACCU
2023**

CMAKE
A CASE STUDY

HANS VREDEVELD

About me

- Senior consultant at CGI Nederland B.V.
- Experienced C++ software engineer
- Experience with CMake through practical application
 - Converting legacy build to CMake
 - Not a CMake expert

The Case

Build a C++ application that uses Oracle embedded SQL with CMake

- \approx 80 files with embedded SQL
- Files spread out over several build targets
- No support for Oracle in CMake
- No support for CMake in Oracle
- Must be usable by colleagues without CMake knowledge

C++ with Embedded SQL

```
1 void my_func() {
2     EXEC SQL BEGIN DECLARE SECTION;
3     char name[NAME_SIZE]; int age;
4     EXEC SQL END DECLARE SECTION;
5
6     // C++ code
7
8     EXEC SQL EXECUTE
9     BEGIN
10         SELECT p.age INTO :age FROM person p WHERE p.name = :name;
11     END;
12     END-EXEC;
13
14     std::cout << "name = " << name ", age = " << age << '\n';
15 }
```

The Environment

- Red Hat Enterprise Linux 7
- GCC 10.3
- CMake 3.18
- Oracle 12

Manual Build

Manual Build

1. From C++ with embedded SQL generate pure C++
2. Compile generated C++ file
3. Link object files

Manual Build

1. From C++ with embedded SQL generate pure C++

```
1 proc iname=my_source.pcpp oname=my_generated_source.cpp
2     code=cpp
3     cpp_suffix=cpp
4     userid=user/password@database
5     option=value
6     include=${ORACLE_HOME}/rdbms/public
7     include=${ORACLE_HOME}/precomp/public
8     include=${GCC_DIR}/include/c++/10.3.0
9     include=${GCC_DIR}/include/c++/10.3.0/backward
10    include=other_gcc_dirs
11    include=project_dir
12    include=other_include_dirs
```


Manual Build

1. From C++ with embedded SQL generate pure C++

```
1 proc iname=my_source.pcpp oname=my_generated_source.cpp
2     code=cpp
3     cpp_suffix=cpp
4     userid=user/password@database
5     option=value
6     include=${ORACLE_HOME}/rdbms/public
7     include=${ORACLE_HOME}/precomp/public
8     include=${GCC_DIR}/include/c++/10.3.0
9     include=${GCC_DIR}/include/c++/10.3.0/backward
10    include=other_gcc_dirs
11    include=project_dir
12    include=other_include_dirs
```

Manual Build

1. From C++ with embedded SQL generate pure C++

```
1 proc iname=my_source.pcpp oname=my_generated_source.cpp
2     code=cpp
3     cpp_suffix=cpp
4     userid=user/password@database
5     option=value
6     include=${ORACLE_HOME}/rdbms/public
7     include=${ORACLE_HOME}/precomp/public
8     include=${GCC_DIR}/include/c++/10.3.0
9     include=${GCC_DIR}/include/c++/10.3.0/backward
10    include=other_gcc_dirs
11    include=project_dir
12    include=other_include_dirs
```

Manual Build

1. From C++ with embedded SQL generate pure C++

```
1 proc iname=my_source.pcpp oname=my_generated_source.cpp
2     code=cpp
3     cpp_suffix=cpp
4     userid=user/password@database
5     option=value
6     include=${ORACLE_HOME}/rdbms/public
7     include=${ORACLE_HOME}/precomp/public
8     include=${GCC_DIR}/include/c++/10.3.0
9     include=${GCC_DIR}/include/c++/10.3.0/backward
10    include=other_gcc_dirs
11    include=project_dir
12    include=other_include_dirs
```

Manual Build

1. From C++ with embedded SQL generate pure C++

```
1 proc iname=my_source.pcpp oname=my_generated_source.cpp
2     code=cpp
3     cpp_suffix=cpp
4     userid=user/password@database
5     option=value
6     include=${ORACLE_HOME}/rdbms/public
7     include=${ORACLE_HOME}/precomp/public
8     include=${GCC_DIR}/include/c++/10.3.0
9     include=${GCC_DIR}/include/c++/10.3.0/backward
10    include=other_gcc_dirs
11    include=project_dir
12    include=other_include_dirs
```

Manual Build

2. Compile generated C++ file

```
1 c++ -c my_generated_source.cpp -o my_object.o
2     -Iproject_dir
3     -Iother_include_dirs
4     -isystem ${ORACLE_HOME}/rdbms/public
5     -isystem ${ORACLE_HOME}/precomp/public
6     -Wall -Wextra -Wpedantic -Wother-warnings
7     -Wno-missing-field-initializers
8     -Wno-shadow
9     -Wno-useless-cast
10    -Wno-write-strings
```

Manual Build

2. Compile generated C++ file

```
1 c++ -c my_generated_source.cpp -o my_object.o
2     -Iproject_dir
3     -Iother_include_dirs
4     -isystem ${ORACLE_HOME}/rdbms/public
5     -isystem ${ORACLE_HOME}/precomp/public
6     -Wall -Wextra -Wpedantic -Wother-warnings
7     -Wno-missing-field-initializers
8     -Wno-shadow
9     -Wno-useless-cast
10    -Wno-write-strings
```

Manual Build

3. Link object files

```
1 c++ -o my_executable
2   my_object.o
3   other_object.o
4   -Wl, -rpath, ${ORACLE_HOME}/lib:
5   ${ORACLE_HOME}/lib/libclntsh.so
6   other_lib1.a
7   other_lib2.so
```

Quick Overview of CMake

Workflow

1. Configure build

```
1 cmake -S /source/dir -B /build/dir
```

1. Configure build (parse configuration files)
2. Generate build configuration

2. Build project

```
1 cmake --build /build/dir
```

3. Test build / install or package artifacts

CMake Configuration

- File `CMakeLists.txt`
- Command `include(/path/to/file)`
 - Load and run a CMake file
- Command `add_subdirectory(sub_dir)`
 - Add a subdirectory to the project
 - Looks for a `CMakeLists.txt` in the directory

Creating Build Targets

- Create an executable target

```
1 add_executable(my_exe  
2   source1.cpp source2.cpp)
```

- Create a library target

```
1 add_library(my_lib  
2   STATIC|SHARED|...  
3   source1.cpp source2.cpp)
```

Working with Targets

- Add source files to a target

```
1 target_sources(my_target
2     INTERFACE|PUBLIC|PRIVATE
3     sourceA.cpp sourceB.cpp)
```

- Add include directories to a target

```
1 target_include_directories(my_target
2     INTERFACE|PUBLIC|PRIVATE
3     /dir/1 dir/2)
```

- Add compile options to a target

```
1 target_compile_options(my_target
2     INTERFACE|PUBLIC|PRIVATE
3     -option1 -option2)
```

Working with Targets

- Add libraries or linker flags to target

```
1 target_link_libraries(my_target
2   INTERFACE|PUBLIC|PRIVATE
3   another_target
4   /path/to/some.so
5   some_lib
6   -flag)
```

- For `another_target`: Also takes care of include directories, compiler flags, etc.

Working with targets

Scope keywords in `target_*` commands:

- **PRIVATE**
Item needed for building target itself
- **PUBLIC**
Item needed for building target and depending targets
- **INTERFACE**
Item needed for building depending targets only

Working with Variables

- Setting a normal variable

```
1 set(MY_VAR "Some value with ${OTHER_VAR}")
```

- Setting a cache entry (stored for use in later CMake invocations)

```
1 set(CACHE_VAR "Default value"  
2     CACHE  
3     BOOL|STRING|...  
4     "Documentation for CACHE_VAR")
```

User can set when configuring project:

```
1 cmake -DCACHE_VAR:STRING="value" \  
2     -S /source/dir -B /build/dir
```

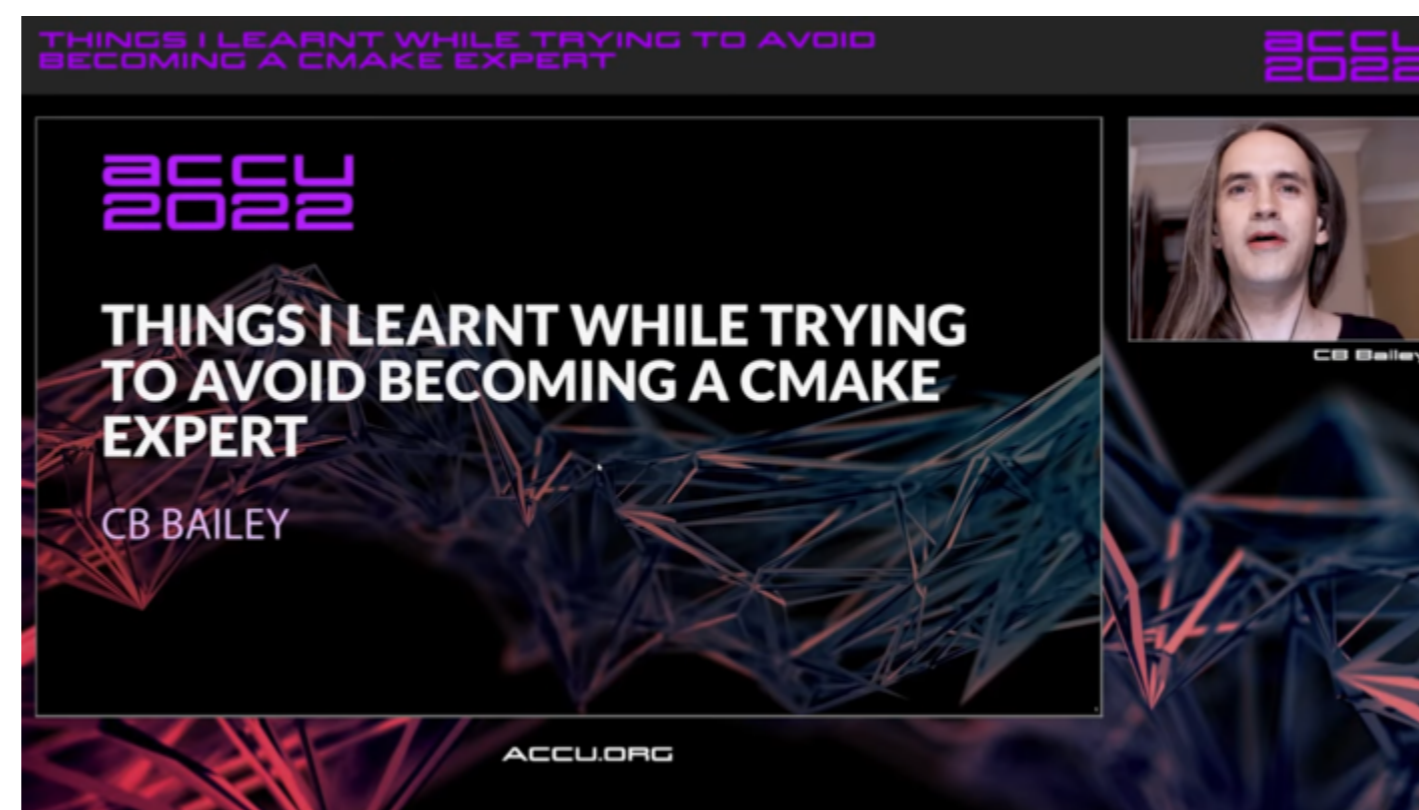
Finding Modules

- Searching for `external_module` with version `1.2.3` or later

```
1 find_package(external_module 1.2.3 REQUIRED)
```

- `REQUIRED` makes CMake configuration fail when module not found

Introduction to CMake



<https://www.youtube.com/watch?v=852VSXFAD00>

Finding Oracle

Goal

```
1 find_package(Oracle 12.0.0 REQUIRED)
2
3 ...
4
5 target_link_libraries(my_target PRIVATE Oracle::Oracle)
```

Module File

For `find_package(Oracle 12.0.0 REQUIRED)` to work

- CMake must know where to find the module

```
1 list(APPEND CMAKE_MODULE_PATH
2        ${CMAKE_CURRENT_SOURCE_DIR}/cmake/Modules)
```

- Module file must be called `FindOracle.cmake`

Oracle Location

- Module needs to know where Oracle client is installed
- Oracle needs to know where it is installed too
 - Uses environment variable `ORACLE_HOME`

Oracle Location

- Module needs to know where Oracle client is installed
- Oracle needs to know where it is installed too
 - Uses environment variable ORACLE_HOME

In FindOracle.cmake:

```
1 if(NOT Oracle_DIR)
2   if(EXISTS $ENV{ORACLE_HOME})
3     set(Oracle_DIR $ENV{ORACLE_HOME}
4         CACHE PATH "Root directory of the Oracle installation")
5   endif()
6 endif()
```

Oracle Include Path

```
1 find_path(Oracle_INCLUDE_DIR
2   NAMES oratypes.h
3   HINTS ${Oracle_DIR}
4   PATH_SUFFIXES rdbms/public precomp/public
5   DOC "Include path for Oracle headers"
6   NO_DEFAULT_PATH)
7 mark_as_advanced(Oracle_INCLUDE_DIR)
```

Search for directory containing `oratypes.h`

- Store result in `Oracle_INCLUDE_DIR`
- `Oracle_INCLUDE_DIR` is not shown in GUIs
 - unless 'show advanced' is on

Oracle Include Path

```
1 find_path(Oracle_INCLUDE_DIR
2   NAMES oratypes.h
3   HINTS ${Oracle_DIR}
4   PATH_SUFFIXES rdbms/public precomp/public
5   DOC "Include path for Oracle headers"
6   NO_DEFAULT_PATH)
7 mark_as_advanced(Oracle_INCLUDE_DIR)
```

Search in:

- `${Oracle_DIR}`
- `${Oracle_DIR}/rdbms/public`
- `${Oracle_DIR}/precomp/public`
- and no default locations

Oracle Include Path

```
1 find_path(Oracle_INCLUDE_DIR
2   NAMES oratypes.h
3   HINTS ${Oracle_DIR}
4   PATH_SUFFIXES rdbms/public precomp/public
5   DOC "Include path for Oracle headers"
6   NO_DEFAULT_PATH)
7 mark_as_advanced(Oracle_INCLUDE_DIR)
```

Finds location of Oracle RDBMS headers

- Store in Oracle_INCLUDE_DIR

Oracle Include Path

```
1 find_path(Oracle_PRECOMP_INCLUDE_DIR
2   NAMES sqlca.h
3   HINTS ${Oracle_DIR}
4   PATH_SUFFIXES rdbms/public precomp/public
5   DOC "Include path for Oracle precompile headers"
6   NO_DEFAULT_PATH)
7 mark_as_advanced(Oracle_PRECOMP_INCLUDE_DIR)
```

Finds location of Oracle precompile headers

- Store in Oracle_PRECOMP_INCLUDE_DIR

Oracle Library

```
1 find_library(Oracle_LIBRARY
2   NAMES libclntsh clntsh
3   HINTS ${Oracle_DIR}
4   PATH_SUFFIXES lib
5   DOC "Path to the Oracle client library"
6   NO_DEFAULT_PATH)
7 mark_as_advanced(Oracle_LIBRARY)
```

- Search for one of:
 - libclntsh, libclntsh.so, libclntsh.a
 - clntsh, clntsh.so, clntsh.a
- Search in `${Oracle_DIR}`, `${Oracle_DIR}/lib`

Oracle Client Programs

```
1 find_program(Oracle_SQLPLUS
2   NAMES sqlplus
3   HINTS ${Oracle_DIR}
4   PATH_SUFFIXES bin
5   DOC "Path to the Oracle SQL*Plus command line SQL program"
6   NO_DEFAULT_PATH)
7 mark_as_advanced(Oracle_SQLPLUS)
```

- Search for sqlplus (SQL command line interface)
- Search in \${Oracle_DIR}, \${Oracle_DIR}/bin

Oracle Client Programs

```
1 find_program(Oracle_PROC
2   NAMES proc
3   HINTS ${Oracle_DIR}
4   PATH_SUFFIXES bin
5   DOC "Path to the Oracle Pro*C compiler program"
6   NO_DEFAULT_PATH)
7 mark_as_advanced(Oracle_PROC)
```

- Search for proc (Oracle precompiler)
- Search in `${Oracle_DIR}`, `${Oracle_DIR}/bin`

Oracle Version

Oracle SQL*Plus can tell the version

```
1 $ sqlplus -V
2
3 SQL*Plus: Release 12.2.0.1.0 Production
4
5 $ _
```

Oracle Version

```
1 $ sqlplus -V
2
3 SQL*Plus: Release 12.2.0.1.0 Production
4
5 $ _
```

From CMake:

```
1 if(Oracle_SQLPLUS)
2   execute_process(
3     COMMAND ${Oracle_SQLPLUS} -V
4     OUTPUT_VARIABLE Oracle_version_output
5     ERROR_VARIABLE Oracle_version_error
6     RESULT_VARIABLE Oracle_version_result
7     OUTPUT_STRIP_TRAILING_WHITESPACE)
8 endif()
```

Oracle Version

```
1 if (Oracle_SQLPLUS)
2   execute_process (
3     COMMAND ${Oracle_SQLPLUS} -V
4     OUTPUT_VARIABLE Oracle_version_output
5     ERROR_VARIABLE Oracle_version_error
6     RESULT_VARIABLE Oracle_version_result
7     OUTPUT_STRIP_TRAILING_WHITESPACE)
8 endif ()
```

- Oracle_version_output: stdout output
- Oracle_version_error: stderr output
- Oracle_version_result: exit status

Oracle Version

If `Oracle_version_result` \neq 0, command failed

```
1 if(NOT ${Oracle_version_result} EQUAL 0)
2   message(SEND_ERROR
3     "Command \"${Oracle_SQLPLUS} -V\" failed "
4     "with output:\n${Oracle_version_error}")
5 endif()
```

Oracle Version

Oracle_version_output has output:

```
1  
2 SQL*Plus: Release 12.2.0.1.0 Production
```

Oracle Version

```
1
2 SQL*Plus: Release 12.2.0.1.0 Production
```

Extract version:

```
1 if("${Oracle_version_output}" MATCHES
2     "[\t\r\n]*SQL\\*Plus: Release ([^ ]+) .*")
3     set(Oracle_VERSION "${CMAKE_MATCH_1}")
4 endif()
```

Putting it together

We have set the following variables

```
1 Oracle_INCLUDE_DIR
2 Oracle_PRECOMP_INCLUDE_DIR
3 Oracle_LIBRARY
4 Oracle_PROC
5 Oracle_SQLPLUS
6 Oracle_VERSION
```

but don't know if everything is found correctly

Putting it together

Check if

- required variables are set
- version matches with requested version

```
1 find_package(PackageHandleStandardArgs)
2 find_package_handle_standard_args(Oracle
3   REQUIRED_VARS
4     Oracle_INCLUDE_DIR
5     Oracle_PRECOMP_INCLUDE_DIR
6     Oracle_LIBRARY
7     Oracle_PROC
8     Oracle_SQLPLUS
9   VERSION_VAR
10  Oracle_VERSION)
```

Putting it together

```
1 find_package(PackageHandleStandardArgs)
2 find_package_handle_standard_args(Oracle
3   REQUIRED_VARS
4     Oracle_INCLUDE_DIR
5     Oracle_PRECOMP_INCLUDE_DIR
6     Oracle_LIBRARY
7     Oracle_PROC
8     Oracle_SQLPLUS
9   VERSION_VAR
10   Oracle_VERSION)
```

- Sets variable `Oracle_FOUND`
- Aborts configuration when `REQUIRED` option to `find_package` is set

Putting it together

Create target Oracle::Oracle

```
1 if(Oracle_FOUND)
2   set(Oracle_INCLUDE_DIRS
3     ${Oracle_INCLUDE_DIR}
4     ${Oracle_PRECOMP_INCLUDE_DIR})
5   set(Oracle_LIBRARIES ${Oracle_LIBRARY})
6
7   if(NOT TARGET Oracle::Oracle)
8     add_library(Oracle::Oracle UNKNOWN IMPORTED)
9     set_target_properties(Oracle::Oracle PROPERTIES
10      IMPORTED_LOCATION "${Oracle_LIBRARIES}"
11      INTERFACE_INCLUDE_DIRECTORIES
12      "${Oracle_INCLUDE_DIRS}")
13   endif()
14 endif()
```

Putting it together

Create target Oracle::Oracle

```
1 if(Oracle_FOUND)
2   set(Oracle_INCLUDE_DIRS
3     ${Oracle_INCLUDE_DIR}
4     ${Oracle_PRECOMP_INCLUDE_DIR})
5   set(Oracle_LIBRARIES ${Oracle_LIBRARY})
6
7   if(NOT TARGET Oracle::Oracle)
8     add_library(Oracle::Oracle UNKNOWN IMPORTED)
9     set_target_properties(Oracle::Oracle PROPERTIES
10      IMPORTED_LOCATION "${Oracle_LIBRARIES}"
11      INTERFACE_INCLUDE_DIRECTORIES
12      "${Oracle_INCLUDE_DIRS}")
13   endif()
14 endif()
```


Oracle Found

Now we can use the Oracle library

```
find_package(Oracle 12.0.0 REQUIRED)
```

```
...
```

```
target_link_libraries(my_lib PUBLIC Oracle::Oracle)
```

```
target_link_libraries(my_exe PRIVATE Oracle::Oracle)
```

Default Include Dirs

Default Include Dirs

- Pro*C needs to be able to find standard headers
- Compiler has their location internalised

Default Include Dirs

- Pro*C needs to be able to find standard headers
- Compiler has their location internalised
- Need to specify location to Pro*C explicitly
- GCC can report the include search path

Default Include Dirs

```
1 $ g++ -xc++ -E -v /dev/null
2 ... <output truncated>
3 #include "... " search starts here:
4 #include <...> search starts here:
5 /usr/include/c++/10
6 /usr/include/c++/10/x86_64-pc-linux-linux
7 /usr/include/c++/10/backward
8 /usr/local/include
9 /usr/include
10 End of search list.
11 ... <output truncated>
12 $ _
```

- Runs preprocessor only (-E)
- Prints what it does on stderr (-v)

Default Include Dirs

We'll create a function to add the default include dirs to a target:

```
compiler_default_include_dirs(<target>)
```

Using the Function

In FindOracle.cmake:

```
1 if(NOT TARGET compiler_internals)
2   include(${CMAKE_CURRENT_LIST_DIR}/CompilerInternals.cmake)
3   add_library(compiler_internals INTERFACE)
4   compiler_default_include_dirs(compiler_internals)
5 endif()
```

- Include CompilerInternals.cmake
- Create interface library compiler_internals
- Add default include dirs to interface library

Defining the Function

In `CompilerInternals.cmake`:

```
1 function(compiler_default_include_dirs target)
2   if(CMAKE_CXX_COMPILER_ID STREQUAL "GNU")
3     _gnu_default_include_dirs(${target})
4   else()
5     message(AUTHOR_WARNING
6       "Default include directories not set for "
7       "'${CMAKE_CXX_COMPILER_ID}' compiler.")
8   endif()
9 endfunction()
```


Defining the Function

In `CompilerInternals.cmake`:

```
1 function(compiler_default_include_dirs target)
2   if(CMAKE_CXX_COMPILER_ID STREQUAL "GNU")
3     _gnu_default_include_dirs(${target})
4   else()
5     message(AUTHOR_WARNING
6       "Default include directories not set for "
7       "'${CMAKE_CXX_COMPILER_ID}' compiler.")
8   endif()
9 endfunction()
```

`_gnu_default_include_dirs` does the actual work for GCC

Defining the Function

`_gnu_default_include_dirs` executes `g++ -xc++ -E -v /dev/null`:

```
1 function(_gnu_default_include_dirs target)
2   execute_process(
3     COMMAND ${CMAKE_CXX_COMPILER} -xc++ -E -v /dev/null
4     OUTPUT_QUIET
5     ERROR_VARIABLE compiler_output)
6
7   ...
8 endfunction()
```

Defining the Function

Transform output into list of lines

```
1 function(_gnu_default_include_dirs target)
2     execute_process (
3         COMMAND ${CMAKE_CXX_COMPILER} -xc++ -E -v /dev/null
4         OUTPUT_QUIET
5         ERROR_VARIABLE compiler_output)
6     string(REGEX REPLACE "\n" ";" output_list
7           "${compiler_output}")
8
9     ...
10 endfunction()
```

Intermezzo: Lists

- A list in CMake is a group of ';' separated strings
- `set(var a b c)` creates the list `a;b;c`
- `set(var "a b c")` creates the string `a b c`
- More list manipulation with `list` command

Defining the Function

Find first line with an include path

```
1 function(_gnu_default_include_dirs target)
2   execute_process (
3     COMMAND ${CMAKE_CXX_COMPILER} -xc++ -E -v /dev/null
4     OUTPUT_QUIET
5     ERROR_VARIABLE compiler_output)
6   string(REGEX REPLACE "\n" ";" output_list
7     "${compiler_output}")
8
9   list(FIND output_list
10     "#include <...> search starts here:"
11     include_dirs_begin)
12   math(EXPR include_dirs_begin "${include_dirs_begin} + 1")
13   ...
14 endfunction()
```

Defining the Function

Find first line with an include path

```
1 function(_gnu_default_include_dirs target)
2   execute_process (
3     COMMAND ${CMAKE_CXX_COMPILER} -xc++ -E -v /dev/null
4     OUTPUT_QUIET
5     ERROR_VARIABLE compiler_output)
6   string(REGEX REPLACE "\n" ";" output_list
7     "${compiler_output}")
8
9   list(FIND output_list
10     "#include <...> search starts here:"
11     include_dirs_begin)
12   math(EXPR include_dirs_begin "${include_dirs_begin} + 1")
13   ...
14 endfunction()
```

Defining the Function

Calculate number of include paths

```
5     ERROR_VARIABLE compiler_output)
6     string(REGEX REPLACE "\n" ";" output_list
7           "${compiler_output}")
8
9     list(FIND output_list
10          "#include <...> search starts here:"
11          include_dirs_begin)
12     math(EXPR include_dirs_begin "${include_dirs_begin} + 1")
13
14     list(FIND output_list "End of search list."
15          include_dirs_end)
16     math(EXPR include_dirs_length
17           "${include_dirs_end} - ${include_dirs_begin}")
18     ...
19 endfunction()
```

Defining the Function

Extract include paths from list

```
10     "#include <...> search starts here:"
11     include_dirs_begin)
12     math(EXPR include_dirs_begin "${include_dirs_begin} + 1")
13
14     list(FIND output_list "End of search list."
15          include_dirs_end)
16     math(EXPR include_dirs_length
17          "${include_dirs_end} - ${include_dirs_begin}")
18
19     list(SUBLIST output_list
20          ${include_dirs_begin} ${include_dirs_length}
21          include_dirs)
22     list(TRANSFORM include_dirs STRIP)
23     ...
24 endfunction()
```


Defining the Function

Add include paths to target

```
12  math(EXPR include_dirs_begin "${include_dirs_begin} + 1")
13
14  list(FIND output_list "End of search list."
15       include_dirs_end)
16  math(EXPR include_dirs_length
17       "${include_dirs_end} - ${include_dirs_begin}")
18
19  list(SUBLIST output_list
20       ${include_dirs_begin} ${include_dirs_length}
21       include_dirs)
22  list(TRANSFORM include_dirs STRIP)
23
24  target_include_directories(${target} SYSTEM
25       INTERFACE ${include_dirs})
26 endfunction()
```

The Precompiler

The Precompiler

Refresher on the Pro*C invocation:

```
1 proc iname=my_source.pcpp oname=my_generated_source.cpp
2     code=cpp
3     cpp_suffix=cpp
4     userid=user/password@database
5     option=value
6     include=${ORACLE_HOME}/rdbms/public
7     include=${ORACLE_HOME}/precomp/public
8     include=${GCC_DIR}/include/c++/10.3.0
9     include=${GCC_DIR}/include/c++/10.3.0/backward
10    include=other_gcc_dirs
11    include=project_dir
12    include=other_include_dirs
```

The Precompiler

Refresher on the Pro*C invocation:

```
1 proc iname=input.pcpp oname=output.cpp
2   option=value ... # more options
3   include=path ... # more include directories
```

Pro*C needs:

- an input file
- an output file
- 0 or more options
- 0 or more include paths

The Compiler

Compare this to compiler invocation:

```
1 c++ -c example.cpp -o example.o
2   -Wall ... # more options
3   -I/usr/include ... # more include directories
```

Compiler needs:

- an input file
- an output file
- 0 or more options
- 0 or more include paths

The Compiler

CMake generates compiler command with information supplied by:

- `target_sources` for input files
- internal heuristics for output files
- `target_compile_options` for options
- `target_include_directories` for include paths

The Precompiler

Similar to this we will:

- create `target_oracle_sources` for input files
- use internal heuristics for output files
- create `target_oracle_options` for options

We can use `target_include_directories` for include paths

Target Properties

Target Properties

- CMake commands for working with targets manipulate properties of the target
- These properties come in pairs
 - One for the target itself
 - One for dependent targets

Target Properties

Command	Property for Target	Property for Dependants
target_compile_definitions	COMPILE_DEFINITIONS	INTERFACE_COMPILE_DEFINITIONS
target_compile_features	COMPILE_FEATURES	INTERFACE_COMPILE_FEATURES
target_compile_options	COMPILE_OPTIONS	INTERFACE_COMPILE_OPTIONS
target_include_directories	INCLUDE_DIRECTORIES	INTERFACE_INCLUDE_DIRECTORIES
target_link_directories	LINK_DIRECTORIES	INTERFACE_LINK_DIRECTORIES
target_link_libraries	LINK_LIBRARIES	INTERFACE_LINK_LIBRARIES
target_link_options	LINK_OPTIONS	INTERFACE_LINK_OPTIONS
target_precompile_headers	PRECOMPILE_HEADERS	INTERFACE_PRECOMPILE_HEADERS
target_sources	SOURCES	INTERFACE_SOURCES

Target Properties

- Keywords in `target_<name>` decide to which property things are added
 - `PRIVATE` adds to `<name>`
 - `INTERFACE` adds to `INTERFACE_<name>`
 - `PUBLIC` adds to both `<name>` and `INTERFACE_<name>`

Target Properties

- Calling a target command multiple times appends to the properties
- When generating the build configuration, those properties are used to create build commands

Target Properties

Target properties can be manipulated directly

- `get_target_property`
 - gets a single property
- `set_target_properties`
 - sets multiple properties

Precompiler Options

Precompiler Options

Mimicking

```
1 target_compile_options(target
2     INTERFACE option1=value1 ...
3     PUBLIC option2=value2 ...
4     PRIVATE option3=value3 ...)
```

we create

```
1 target_oracle_options(target
2     INTERFACE option1=value1 ...
3     PUBLIC option2=value2 ...
4     PRIVATE option3=value3 ...)
```

Precompiler Options

```
1 function(target_oracle_options target)
2     ...
3 endfunction()
```


Intermezzo: Functions

```
1 function(my_func first_arg second_arg)
```

defines a function named `my_func`

- having 2 formal parameters; and
- having 0 or more optional parameters

Intermezzo: Functions

```
1 function(my_func first_arg second_arg)
```

- Function names are case insensitive
- Function arguments are case sensitive
- Not supplying a formal argument is an error
- Formal arguments can be referenced by name
 - e.g. `${first_arg}`

Intermezzo: Functions

```
1 function(my_func first_arg second_arg)
```

- Variable ARGV has the number of arguments
- Variable ARGV# references the #'s argument
 - First argument is `${ARGV0}`
- Variable ARGV has a list of all arguments
- Variable ARGN has a list of all optional arguments
- Function name is never included

Precompiler Options

```
1 function(target_oracle_options target)
2     ...
3 endfunction()
```

Precompiler Options

```
1 function(target_oracle_options target)
2     set(options "")
3     set(one_value_args "")
4     set(multi_value_args INTERFACE PUBLIC PRIVATE)
5     cmake_parse_arguments(OPTS "${options}" "${one_value_args}"
6         "${multi_value_args}" ${ARGN})
7
8     ...
9 endfunction()
```

Precompiler Options

```
1 set(multi_value_args INTERFACE PUBLIC PRIVATE)
2 cmake_parse_arguments(OPTS "${options}" "${one_value_args}"
3   "${multi_value_args}" ${ARGN})
```

Creates and sets variables:

- OPTS_INTERFACE
- OPTS_PUBLIC
- OPTS_PRIVATE
- OPTS_UNPARSED_ARGUMENTS
- OPTS_KEYWORDS_MISSING_VALUES

Precompiler Options

```
1 function(target_oracle_options target)
2     set(options "")
3     set(one_value_args "")
4     set(multi_value_args INTERFACE PUBLIC PRIVATE)
5     cmake_parse_arguments(OPTS "${options}" "${one_value_args}"
6         "${multi_value_args}" ${ARGN})
7
8     if(OPTS_UNPARSED_ARGUMENTS)
9         message(FATAL_ERROR
10             "target_oracle_options: for target ${target}: "
11             "unknown arguments \"${OPTS_UNPARSED_ARGUMENTS}\"")
12     endif()
13
14     get_target_property(oracle_options ${target}
15         ORACLE_OPTIONS)
```

Precompiler Options

```
2  set(options "")
3  set(one_value_args "")
4  set(multi_value_args INTERFACE PUBLIC PRIVATE)
5  cmake_parse_arguments(OPTS "${options}" "${one_value_args}"
6     "${multi_value_args}" ${ARGN})
7
8  if(OPTS_UNPARSED_ARGUMENTS)
9     message(FATAL_ERROR
10        "target_oracle_options: for target ${target}: "
11        "unknown arguments \"${OPTS_UNPARSED_ARGUMENTS}\"")
12 endif()
13
14 get_target_property(oracle_options ${target}
15    ORACLE_OPTIONS)
16 get_target_property(interface_oracle_options ${target}
17    INTERFACE_ORACLE_OPTIONS)
```


Precompiler Options

```
8  if(OPTS_UNPARSED_ARGUMENTS)
9      message(FATAL_ERROR
10         "target_oracle_options: for target ${target}: "
11         "unknown arguments \"${OPTS_UNPARSED_ARGUMENTS}\"")
12  endif()
13
14  get_target_property(oracle_options ${target}
15     ORACLE_OPTIONS)
16  get_target_property(interface_oracle_options ${target}
17     INTERFACE_ORACLE_OPTIONS)
18
19  if(NOT oracle_options)
20      set(oracle_options "")
21  endif()
22  if(NOT interface_oracle_options)
23      set(interface_oracle_options "")
```

Precompiler Options

```
14  get_target_property(oracle_options ${target}
15      ORACLE_OPTIONS)
16  get_target_property(interface_oracle_options ${target}
17      INTERFACE_ORACLE_OPTIONS)
18
19  if(NOT oracle_options)
20      set(oracle_options "")
21  endif()
22  if(NOT interface_oracle_options)
23      set(interface_oracle_options "")
24  endif()
25
26  list(APPEND oracle_options ${OPTS_PRIVATE})
27  list(APPEND oracle_options ${OPTS_PUBLIC})
28  list(APPEND interface_oracle_options ${OPTS_PUBLIC})
29  list(APPEND interface_oracle_options ${OPTS_INTERFACE})
```

Precompiler Options

```
20     set(oracle_options "")
21 endif()
22 if(NOT interface_oracle_options)
23     set(interface_oracle_options "")
24 endif()
25
26 list(APPEND oracle_options ${OPTS_PRIVATE})
27 list(APPEND oracle_options ${OPTS_PUBLIC})
28 list(APPEND interface_oracle_options ${OPTS_PUBLIC})
29 list(APPEND interface_oracle_options ${OPTS_INTERFACE})
30
31 set_target_properties(${target} PROPERTIES
32     ORACLE_OPTIONS "${oracle_options}"
33     INTERFACE_ORACLE_OPTIONS "${interface_oracle_options}")
34 endfunction()
```

Precompiler Options

```
20     set(oracle_options "")
21 endif()
22 if(NOT interface_oracle_options)
23     set(interface_oracle_options "")
24 endif()
25
26 list(APPEND oracle_options ${OPTS_PRIVATE})
27 list(APPEND oracle_options ${OPTS_PUBLIC})
28 list(APPEND interface_oracle_options ${OPTS_PUBLIC})
29 list(APPEND interface_oracle_options ${OPTS_INTERFACE})
30
31 set_target_properties(${target} PROPERTIES
32     ORACLE_OPTIONS "${oracle_options}"
33     INTERFACE_ORACLE_OPTIONS "${interface_oracle_options}")
34 endfunction()
```

Precompiler Sources

Precompiler Sources

Mimicking

```
1 target_sources(target
2   INTERFACE source1.cpp ...
3   PUBLIC source2.cpp ...
4   PRIVATE source3.cpp ...)
```

we create

```
1 target_oracle_sources(target
2   INTERFACE source1.pcpp ...
3   PUBLIC source2.pcpp ...
4   PRIVATE source3.pcpp ...)
```

Precompiler Sources

```
function(target_oracle_sources target)
    ...
endfunction()
```

Precompiler Sources

```
function(target_oracle_sources target)
  set(options "")
  set(one_value_args "")
  set(multi_value_args INTERFACE PUBLIC PRIVATE)
  cmake_parse_arguments(SRCS "${options}" "${one_value_args}"
    "${multi_value_args}" ${ARGN})

  ...
endfunction()
```


Precompiler Sources

```
1 set(multi_value_args INTERFACE PUBLIC PRIVATE)
2 cmake_parse_arguments(SRCS "${options}" "${one_value_args}"
3   "${multi_value_args}" ${ARGN})
```

Creates and sets variables:

- SRCS_INTERFACE
- SRCS_PUBLIC
- SRCS_PRIVATE
- SRCS_UNPARSED_ARGUMENTS
- SRCS_KEYWORDS_MISSING_VALUES

Precompiler Sources

```
1 function(target_oracle_sources target)
2     set(options "")
3     set(one_value_args "")
4     set(multi_value_args INTERFACE PUBLIC PRIVATE)
5     cmake_parse_arguments(SRCS "${options}" "${one_value_args}"
6         "${multi_value_args}" ${ARGN})
7
8     if(SRCS_UNPARSED_ARGUMENTS)
9         message(FATAL_ERROR
10             "target_oracle_sources: for target ${target}: "
11             "unknown arguments \"${SRCS_UNPARSED_ARGUMENTS}\"")
12     endif()
13
14     target_link_libraries(${target} PRIVATE compiler_internals)
15
16     # ... (more code) ...
```

Precompiler Sources

```
3  set(one_value_args "")
4  set(multi_value_args INTERFACE PUBLIC PRIVATE)
5  cmake_parse_arguments(SRCS "${options}" "${one_value_args}"
6     "${multi_value_args}" ${ARGN})
7
8  if(SRCS_UNPARSED_ARGUMENTS)
9     message(FATAL_ERROR
10        "target_oracle_sources: for target ${target}: "
11        "unknown arguments \"${SRCS_UNPARSED_ARGUMENTS}\"")
12 endif()
13
14 target_link_libraries(${target} PRIVATE compiler_internals)
15
16 file(MAKE_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR}/proc)
17 ...
```

Precompiler Sources

```
4  set(multi_value_args INTERFACE PUBLIC PRIVATE)
5  cmake_parse_arguments(SRCS "${options}" "${one_value_args}"
6     "${multi_value_args}" ${ARGN})
7
8  if(SRCS_UNPARSED_ARGUMENTS)
9     message(FATAL_ERROR
10        "target_oracle_sources: for target ${target}: "
11        "unknown arguments \"${SRCS_UNPARSED_ARGUMENTS}\"")
12  endif()
13
14  target_link_libraries(${target} PRIVATE compiler_internals)
15
16  file(MAKE_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR}/proc)
17  ...
18  endfunction()
```

Precompiler Sources

```
4  set(multi_value_args INTERFACE PUBLIC PRIVATE)
5  cmake_parse_arguments(SRCS "${options}" "${one_value_args}"
6     "${multi_value_args}" ${ARGN})
7
8  if(SRCS_UNPARSED_ARGUMENTS)
9     message(FATAL_ERROR
10        "target_oracle_sources: for target ${target}: "
11        "unknown arguments \"${SRCS_UNPARSED_ARGUMENTS}\"")
12  endif()
13
14  target_link_libraries(${target} PRIVATE compiler_internals)
15
16  file(MAKE_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR}/proc)
17  ...
18  endfunction()
```

Precompiler Sources

- Variables `SRCS_INTERFACE`, `SRCS_PUBLIC` and `SRCS_PRIVATE` contain lists of source files

Precompiler Sources

- Variables `SRCS_INTERFACE`, `SRCS_PUBLIC` and `SRCS_PRIVATE` contain lists of source files

```
1 function(target_oracle_sources target)
2     ...
3
4     file(MAKE_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR}/proc)
5     foreach(file IN LISTS SRCS_INTERFACE)
6         ... # Process a single file
7     endforeach()
8     # Repeat for SRCS_PUBLIC and SRCS_PRIVATE
9 endfunction()
```

Precompiler Sources

- Variables `SRCS_INTERFACE`, `SRCS_PUBLIC` and `SRCS_PRIVATE` contain lists of source files
 - Variable `multi_value_args` contains list of scopes: `INTERFACE;PUBLIC;PRIVATE`

```
1 function(target_oracle_sources target)
2     ...
3
4     file(MAKE_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR}/proc)
5     foreach(file IN LISTS SRCS_INTERFACE)
6         ... # Process a single file
7     endforeach()
8     # Repeat for SRCS_PUBLIC and SRCS_PRIVATE
9 endfunction()
```


Precompiler Sources

- Variables `SRCS_INTERFACE`, `SRCS_PUBLIC` and `SRCS_PRIVATE` contain lists of source files
 - Variable `multi_value_args` contains list of scopes: `INTERFACE`; `PUBLIC`; `PRIVATE`

```
1 function(target_oracle_sources target)
2     ...
3
4     file(MAKE_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR}/proc)
5     foreach(scope IN LISTS multi_value_args)
6         foreach(file IN LISTS SRCS_${scope})
7             ... # Process a single file
8         endforeach()
9     endforeach()
10 endfunction()
```

Precompiler Sources

Inner loop:

```
1 foreach(file IN LISTS SRCS_${scope})
2   get_filename_component(input_file ${file} REALPATH)
3   get_filename_component(filename ${file} NAME)
4   set(output_file
5     ${CMAKE_CURRENT_BINARY_DIR}/proc/${filename}.cpp)
6   file(RELATIVE_PATH rel_output_file
7     "${CMAKE_BINARY_DIR}" "${output_file}")
8
9   add_custom_command(...) # Creates ${output_file}
10
11   target_sources(${target} ${scope} ${output_file})
12
13   ...
14 endforeach()
```

Precompiler Sources

Inner loop:

```
1 foreach(file IN LISTS SRCS_${scope})
2   get_filename_component(input_file ${file} REALPATH)
3   get_filename_component(filename ${file} NAME)
4   set(output_file
5     ${CMAKE_CURRENT_BINARY_DIR}/proc/${filename}.cpp)
6   file(RELATIVE_PATH rel_output_file
7     "${CMAKE_BINARY_DIR}" "${output_file}")
8
9   add_custom_command(...) # Creates ${output_file}
10
11  target_sources(${target} ${scope} ${output_file})
12
13  ...
14 endforeach()
```

Precompiler Sources

Inner loop:

```
1 foreach(file IN LISTS SRCS_${scope})
2   get_filename_component(input_file ${file} REALPATH)
3   get_filename_component(filename ${file} NAME)
4   set(output_file
5     ${CMAKE_CURRENT_BINARY_DIR}/proc/${filename}.cpp)
6   file(RELATIVE_PATH rel_output_file
7     "${CMAKE_BINARY_DIR}" "${output_file}")
8
9   add_custom_command(...) # Creates ${output_file}
10
11   target_sources(${target} ${scope} ${output_file})
12
13   ...
14 endforeach()
```

Precompiler Sources

Inner loop:

```
1 foreach(file IN LISTS SRCS_${scope})
2   get_filename_component(input_file ${file} REALPATH)
3   get_filename_component(filename ${file} NAME)
4   set(output_file
5     ${CMAKE_CURRENT_BINARY_DIR}/proc/${filename}.cpp)
6   file(RELATIVE_PATH rel_output_file
7     "${CMAKE_BINARY_DIR}" "${output_file}")
8
9   add_custom_command(...) # Creates ${output_file}
10
11   target_sources(${target} ${scope} ${output_file})
12
13   ...
14 endforeach()
```

Precompiler Sources

Inner loop:

```
1 foreach(file IN LISTS SRCS_${scope})
2   get_filename_component(input_file ${file} REALPATH)
3   get_filename_component(filename ${file} NAME)
4   set(output_file
5     ${CMAKE_CURRENT_BINARY_DIR}/proc/${filename}.cpp)
6   file(RELATIVE_PATH rel_output_file
7     "${CMAKE_BINARY_DIR}" "${output_file}")
8
9   add_custom_command(...) # Creates ${output_file}
10
11   target_sources(${target} ${scope} ${output_file})
12
13   ...
14 endforeach()
```

Precompiler Sources

Inner loop:

```
1 foreach(file IN LISTS SRCS_${scope})
2   get_filename_component(input_file ${file} REALPATH)
3   get_filename_component(filename ${file} NAME)
4   set(output_file
5     ${CMAKE_CURRENT_BINARY_DIR}/proc/${filename}.cpp)
6   file(RELATIVE_PATH rel_output_file
7     "${CMAKE_BINARY_DIR}" "${output_file}")
8
9   add_custom_command(...) # Creates ${output_file}
10
11   target_sources(${target} ${scope} ${output_file})
12
13   ...
14 endforeach()
```

Precompiler Sources

Inner loop:

```
1 foreach(file IN LISTS SRCS_${scope})
2   get_filename_component(input_file ${file} REALPATH)
3   get_filename_component(filename ${file} NAME)
4   set(output_file
5     ${CMAKE_CURRENT_BINARY_DIR}/proc/${filename}.cpp)
6   file(RELATIVE_PATH rel_output_file
7     "${CMAKE_BINARY_DIR}" "${output_file}")
8
9   add_custom_command(...) # Creates ${output_file}
10
11  target_sources(${target} ${scope} ${output_file})
12
13  ...
14 endforeach()
```


Precompiler Sources

Inner loop:

```
3  get_filename_component(filename ${file} NAME)
4  set(output_file
5     ${CMAKE_CURRENT_BINARY_DIR}/proc/${filename}.cpp)
6  file(RELATIVE_PATH rel_output_file
7     "${CMAKE_BINARY_DIR}" "${output_file}")
8
9  add_custom_command(...) # Creates ${output_file}
10
11 target_sources(${target} ${scope} ${output_file})
12
13 set_source_files_properties(${output_file}
14     PROPERTIES
15     COMPILE_FLAGS "-Wno-missing-field-initializers
16     -Wno-shadow -Wno-useless-cast -Wno-write-strings")
17 endforeach()
```

Precompiler Sources

```
add_custom_command(  
  OUTPUT ${output_file}  
  COMMAND  
    ${Oracle_PROC} iname=${input_file} oname=${output_file} ...  
    # Options  
    # Include directories  
  MAIN_DEPENDENCY ${input_file}  
  COMMENT "Creating Pro*C/C++ output ${rel_output_file}")
```

Precompiler Sources

```
add_custom_command(  
  OUTPUT ${output_file}  
  COMMAND  
    ${Oracle_PROC} iname=${input_file} oname=${output_file} ...  
    # Options  
    # Include directories  
  MAIN_DEPENDENCY ${input_file}  
  COMMENT "Creating Pro*C/C++ output ${rel_output_file}")
```

Specifying dependencies of custom command:

- **DEPENDS**: list of dependencies (files, targets)
- **MAIN_DEPENDENCY**: primary input source file

Intermezzo: Property Use

- With `get_target_property` we get the property as it is at that moment of configuring build

E.g. Value of `my_var` in

```
1 set_target_properties(my_target PROPERTIES CONFERENCE "ACCU")  
2 get_target_property(my_var my_target CONFERENCE)
```

is different from value in

```
1 get_target_property(my_var my_target CONFERENCE)  
2 set_target_properties(my_target PROPERTIES CONFERENCE "ACCU")
```

Intermezzo: Property Use

- We want to be able to do

```
1 target_oracle_options(my_target PRIVATE conference=ACCU)
2 target_oracle_sources(my_target PRIVATE source.pcpp)
3 target_oracle_options(my_target
4     PRIVATE location=Bristol)
```

- Need value of target property after configuring build
 - that is, when generating the build
- This is what generator expressions are for

Intermezzo: Generator Expressions

- Are evaluated when generating build
- Can be used in many places
 - But not everywhere
- Have the form $\$(\dots)$
- Cannot be split across multiple lines

Intermezzo: Generator Expressions

- Support
 - Boolean logic
 - Working with information from targets
 - Working with general build information
 - Utility expressions

Intermezzo: Generator Expressions

- Generator expression for target property

```
1 $<TARGET_PROPERTY:target,property>
```

- For most properties, just gets the property content
- For `INCLUDE_DIRECTORIES` (and some other) target property concatenates:
 - The target property itself
 - The interface target property of dependents

Precompiler Sources

```
1 add_custom_command(  
2   OUTPUT ${output_file}  
3   COMMAND  
4     ${Oracle_PROC} iname=${input_file} oname=${output_file} ...  
5     # Options  
6     # Include directories  
7   MAIN_DEPENDENCY ${input_file}  
8   COMMENT "Creating Pro*C/C++ output ${rel_output_file}")
```

Precompiler Sources

```
1 add_custom_command(  
2   OUTPUT ${output_file}  
3   COMMAND  
4     ${Oracle_PROC} iname=${input_file} oname=${output_file}  
5     "$<TARGET_PROPERTY:${target},ORACLE_OPTIONS>" ...  
6     # Include directories  
7   MAIN_DEPENDENCY ${input_file}  
8   COMMAND_EXPAND_LISTS  
9   VERBATIM  
10  COMMENT "Creating Pro*C/C++ output ${rel_output_file}")
```

Precompiler Sources

```
1 add_custom_command(  
2   OUTPUT ${output_file}  
3   COMMAND  
4     ${Oracle_PROC} iname=${input_file} oname=${output_file}  
5     "$<TARGET_PROPERTY:${target},ORACLE_OPTIONS>"  
6     "${includes}"  
7   MAIN_DEPENDENCY ${input_file}  
8   COMMAND_EXPAND_LISTS  
9   VERBATIM  
10  COMMENT "Creating Pro*C/C++ output ${rel_output_file}")
```

Precompiler Sources

```
1 set(includes ...)
2
3 add_custom_command(
4     OUTPUT ${output_file}
5     COMMAND
6         ${Oracle_PROC} iname=${input_file} oname=${output_file}
7         "$<TARGET_PROPERTY:${target},ORACLE_OPTIONS>"
8         "${includes}"
9     MAIN_DEPENDENCY ${input_file}
10    COMMAND_EXPAND_LISTS
11    VERBATIM
12    COMMENT "Creating Pro*C/C++ output ${rel_output_file}")
```

Precompiler Sources

```
1  
2  
3  
4  
5 set(includes ...)  
6  
7  
8 add_custom_command(...)
```

Precompiler Sources

```
1 set(include_dirs
2     "$<TARGET_PROPERTY:${target},INCLUDE_DIRECTORIES>")
3
4
5 set(includes ...)
6
7
8 add_custom_command(...)
```

Precompiler Sources

```
1 set(include_dirs
2     "$<TARGET_PROPERTY:${target},INCLUDE_DIRECTORIES>")
3 set(deduplicated_dirs "$<REMOVE_DUPLICATES:${include_dirs}>")
4
5 set(includes ...)
6
7
8 add_custom_command(...)
```

Precompiler Sources

```
1 set(include_dirs
2     "$<TARGET_PROPERTY:${target},INCLUDE_DIRECTORIES>")
3 set(deduplicated_dirs "$<REMOVE_DUPLICATES:${include_dirs}>")
4 set(filtered_dirs "$<FILTER:${deduplicated_dirs},EXCLUDE,^$>")
5 set(includes ...)
6
7
8 add_custom_command(...)
```


Precompiler Sources

```
1 set(include_dirs
2     "$<TARGET_PROPERTY:${target},INCLUDE_DIRECTORIES>")
3 set(deduplicated_dirs "$<REMOVE_DUPLICATES:${include_dirs}>")
4 set(filtered_dirs "$<FILTER:${deduplicated_dirs},EXCLUDE,^$>")
5 set(includes
6     "include=$<JOIN:${filtered_dirs},$<SEMICOLON>include=>")
7
8 add_custom_command(...)
```

Precompiler Sources

```
1 set(includes
2     "include=${JOIN:${filtered_dirs},${<SEMICOLON>include=>}")
```

converts

```
1 /path/to/dir/1;/path/to/dir/2;/path/3
```

into

```
1 include=/path/to/dir/1;include=/path/to/dir/2;include=/path/3
```

Working Build!

```
build $ cmake --build .
[ 2%] Automatic UIC for target my_lib
[ 2%] Built target my_lib_autogen
[ 36%] Built target my_lib
[ 55%] Built target logger
[ 57%] Creating Pro*C/C++ output src/my_exe/proc/DbCommunicationError.pcpp.cpp
Pro*C/C++: Release 12.2.0.1.0 - Production on Sun Feb 26 19:27:01 2023
Copyright (c) 1982, 2017, Oracle and/or its affiliates. All rights reserved.
System default option values taken from: /u01/app/oracle/product/12.2.0/client_1/precomp/admin/pcscfg.cfg
[ 59%] Creating Pro*C/C++ output src/my_exe/proc/DbCommunication.pcpp.cpp
Pro*C/C++: Release 12.2.0.1.0 - Production on Sun Feb 26 19:27:01 2023
Copyright (c) 1982, 2017, Oracle and/or its affiliates. All rights reserved.
System default option values taken from: /u01/app/oracle/product/12.2.0/client_1/precomp/admin/pcscfg.cfg
[ 61%] Automatic UIC for target my_exe
[ 61%] Built target my_exe_autogen
Scanning dependencies of target my_exe
[ 63%] Building CXX object src/my_exe/CMakeFiles/my_exe.dir/proc/DbCommunication.pcpp.cpp.o
[ 65%] Building CXX object src/my_exe/CMakeFiles/my_exe.dir/proc/DbCommunicationError.pcpp.cpp.o
[ 68%] Linking CXX executable my_exe
[100%] Built target my_exe
build $
```

Improvement: Silencing Pro*C

Build Output

- Pro*C always outputs some uninteresting text
- Interesting output can be difficult to spot
 - By Pro*C
 - By other tools
- More effort needed to check build logs

Build Output

Wanted output:

```
build $ cmake --build .
[ 2%] Automatic UIC for target my_lib
[ 2%] Built target my_lib_autogen
[ 36%] Built target my_lib
[ 55%] Built target logger
[ 57%] Creating Pro*C/C++ output src/my_exe/proc/DbCommunicationError.pcpp.cpp
[ 59%] Creating Pro*C/C++ output src/my_exe/proc/DbCommunication.pcpp.cpp
[ 61%] Automatic UIC for target my_exe
[ 61%] Built target my_exe_autogen
Scanning dependencies of target my_exe
[ 63%] Building CXX object src/my_exe/CMakeFiles/my_exe.dir/proc/DbCommunication.pcpp.cpp.o
[ 65%] Building CXX object src/my_exe/CMakeFiles/my_exe.dir/proc/DbCommunicationError.pcpp.cpp.o
[ 68%] Linking CXX executable my_exe
[100%] Built target my_exe
build $
```

- Makes anomalies (warnings) stand out better
- Still want the former output when errors occur

External Commands

Two ways to execute external command

- `execute_process`
- `add_custom_command`

External Commands

	execute_process	add_custom_command	Wanted
When	build configuration	actual build	actual build
Suppress output	✓	✗	✓ ¹
Capture output	✓	✗	✓ ¹

¹ Normally suppress output, but capture when error occurs

External Commands

Solution:

- Use `add_custom_command` to call `execute_process`
- Encapsulate in new `add_quiet_custom_command`

External Command Usage

Replace

```
1 function(target_oracle_sources target)
2     ...
3     add_custom_command(
4         OUTPUT ${output_file}
5         COMMAND
6             ${Oracle_PROC} iname=${input_file}
7                 oname=${output_file}
8                 "$<TARGET_PROPERTY:${target},ORACLE_OPTIONS>"
9                 "${includes}"
10        MAIN_DEPENDENCY ${input_file}
11        COMMAND_EXPAND_LISTS
12        VERBATIM
13        COMMENT "Creating Pro*C/C++ output ${rel_output_file}")
14     ...
15 endfunction()
```

External Command Usage

By

```
1 function(target_oracle_sources target)
2     ...
3     add_quiet_custom_command(
4         OUTPUT ${output_file}
5         COMMAND
6             ${Oracle_PROC} iname=${input_file}
7             oname=${output_file}
8             "$<TARGET_PROPERTY:${target},ORACLE_OPTIONS>"
9             "${includes}"
10        WORKING_DIR ${CMAKE_CURRENT_BINARY_DIR}
11        MAIN_DEPENDENCY ${input_file}
12        COMMENT "Creating Pro*C/C++ output ${rel_output_file}")
13    ...
14 endfunction()
```

External Command Usage

By

```
1 include(${CMAKE_CURRENT_LIST_DIR}/Execute.cmake)
2 function(target_oracle_sources target)
3     ...
4     add_quiet_custom_command(
5         OUTPUT ${output_file}
6         COMMAND
7             ${Oracle_PROC} iname=${input_file}
8             oname=${output_file}
9             "$<TARGET_PROPERTY:${target},ORACLE_OPTIONS>"
10            "${includes}"
11            WORKING_DIR ${CMAKE_CURRENT_BINARY_DIR}
12            MAIN_DEPENDENCY ${input_file}
13            COMMENT "Creating Pro*C/C++ output ${rel_output_file}")
14     ...
15 endfunction()
```

External Command Usage

File `Execute.cmake` will contain:

- Function `add_quiet_custom_command`
- All other code for the quiet execution of external commands

External Command Usage

File `Execute.cmake` will contain:

- Function `add_quiet_custom_command`
- All other code for the quiet execution of external commands

`Execute.cmake` will be both:

- An include file
- A CMake script

Execute.cmake

```
1 set(EXECUTE_SCRIPT ${CMAKE_CURRENT_LIST_FILE})
2
3 function(add_quiet_custom_command)
4     ... # call add_custom_command
5 endfunction()
6
7 if(EXECUTE_COMMAND STREQUAL "OUTPUT_SUPPRESSION")
8     ... # call execute_process
9 endif()
```

Quiet Custom Command

```
1 function(add_quiet_custom_command)
2   set(options "")
3   set(one_value_args COMMENT MAIN_DEPENDENCY WORKING_DIR)
4   set(multi_value_args COMMAND DEPENDS OUTPUT)
5   cmake_parse_arguments(ARG "${options}" "${one_value_args}"
6     "${multi_value_args}" ${ARGN})
7
8   if(ARG_UNPARSED_ARGUMENTS)
9     message(FATAL_ERROR "add_quiet_custom_command: Unknown
10       arguments \"${ARG_UNPARSED_ARGUMENTS}\"")
11   endif()
12
13   foreach(argument
14     COMMAND MAIN_DEPENDENCY OUTPUT WORKING_DIR)
15     if(NOT ARG_${argument})
```


Quiet Custom Command

```
9     message(FATAL_ERROR "add_quiet_custom_command: Unknown
10         arguments \"${ARG_UNPARSED_ARGUMENTS}\"")
11 endif()
12
13 foreach(argument
14     COMMAND MAIN_DEPENDENCY OUTPUT WORKING_DIR)
15     if(NOT ARG_${argument})
16         message(FATAL_ERROR "add_quiet_custom_command:
17             Missing required argument \"${argument}\"")
18     endif()
19 endforeach()
20
21 foreach(opt_arg DEPENDS COMMENT)
22     if(ARG_${opt_arg})
23         set(${opt_arg}_LINE ${opt_arg} "${ARG_${opt_arg}}")
```

Quiet Custom Command

```
16     message(FATAL_ERROR "add_quiet_custom_command:  
17         Missing required argument \"${argument}\"")  
18     endif()  
19 endforeach()  
20  
21 foreach(opt_arg DEPENDS COMMENT)  
22     if(ARG_${opt_arg})  
23         set(${opt_arg}_LINE ${opt_arg} "${ARG_${opt_arg}}")  
24     else()  
25         set(${opt_arg}_LINE)  
26     endif()  
27 endforeach()  
28  
29 ... # call add_custom_command  
30 endfunction()
```

Variable Substitution

```
1 set(${opt_arg}_LINE ${opt_arg} "${ARG_${opt_arg}}")
```

Variable Substitution

```
1 set(opt_arg DEPENDS)
2 set(ARG_DEPENDS "A dependency")
3
4 set(${opt_arg}_LINE ${opt_arg} "${ARG_${opt_arg}}")
```

Variable Substitution

```
1 set(opt_arg DEPENDS)
2 set(ARG_DEPENDS "A dependency")
3
4 set(DEPENDS_LINE DEPENDS "${ARG_DEPENDS}")
```

Variable Substitution

```
1 set(opt_arg DEPENDS)
2 set(ARG_DEPENDS "A dependency")
3
4 set(DEPENDS_LINE DEPENDS "A dependency")
```

Quiet Custom Command

```
16     message(FATAL_ERROR "add_quiet_custom_command:  
17     Missing required argument \"${argument}\"")  
18     endif()  
19 endforeach()  
20  
21 foreach(opt_arg DEPENDS COMMENT)  
22     if(ARG_${opt_arg})  
23         set(${opt_arg}_LINE ${opt_arg} "${ARG_${opt_arg}}")  
24     else()  
25         set(${opt_arg}_LINE)  
26     endif()  
27 endforeach()  
28  
29 ... # call add_custom_command  
30 endfunction()
```

Quiet Custom Command

```
16     message(FATAL_ERROR "add_quiet_custom_command:  
17         Missing required argument \"${argument}\"")  
18     endif()  
19 endforeach()  
20  
21 foreach(opt_arg DEPENDS COMMENT)  
22     if(ARG_${opt_arg})  
23         set(${opt_arg}_LINE ${opt_arg} "${ARG_${opt_arg}}")  
24     else()  
25         set(${opt_arg}_LINE)  
26     endif()  
27 endforeach()  
28  
29 ... # call add_custom_command  
30 endfunction()
```


Quiet Custom Command

```
26     endif ()
27 endforeach ()
28
29 add_custom_command(
30     OUTPUT ${ARG_OUTPUT}
31     COMMAND
32         ${CMAKE_COMMAND}
33         -DEXECUTE_COMMAND="OUTPUT_SUPPRESSION"
34         -DWORKING_DIR=${ARG_WORKING_DIR}
35         -DCOMMAND="${ARG_COMMAND} "
36         -P ${EXECUTE_SCRIPT}
37     MAIN_DEPENDENCY ${ARG_MAIN_DEPENDENCY}
38     COMMAND_EXPAND_LISTS
39     ${DEPENDS_LINE} ${COMMENT_LINE} )
40 endfunction ()
```

The Script

```
1 if(EXECUTE_COMMAND STREQUAL "OUTPUT_SUPPRESSION")
2     separate_arguments(command_as_list UNIX_COMMAND
3         "${COMMAND}")
4     execute_process(
5         COMMAND ${command_as_list}
6         WORKING_DIRECTORY "${WORKING_DIR}"
7         RESULT_VARIABLE execute_result
8         OUTPUT_VARIABLE execute_output
9         ERROR_VARIABLE execute_output)
10
11     if(NOT ${execute_result} EQUAL 0)
12         message(FATAL_ERROR
13             "Execution of '${COMMAND}' failed with result"
14             "${execute_result}\n"
15             "${execute_output}\n")
16     endif()
```

The Script

```
1 if(EXECUTE_COMMAND STREQUAL "OUTPUT_SUPPRESSION")
2     separate_arguments(command_as_list UNIX_COMMAND
3         "${COMMAND}")
4     execute_process(
5         COMMAND ${command_as_list}
6         WORKING_DIRECTORY "${WORKING_DIR}"
7         RESULT_VARIABLE execute_result
8         OUTPUT_VARIABLE execute_output
9         ERROR_VARIABLE execute_output)
10
11     if(NOT ${execute_result} EQUAL 0)
12         message(FATAL_ERROR
13             "Execution of '${COMMAND}' failed with result"
14             "${execute_result}\n"
15             "${execute_output}\n")
16     endif()
```

The Script

```
3     "${COMMAND}")
4 execute_process (
5     COMMAND ${command_as_list}
6     WORKING_DIRECTORY "${WORKING_DIR}"
7     RESULT_VARIABLE execute_result
8     OUTPUT_VARIABLE execute_output
9     ERROR_VARIABLE execute_output)
10
11 if(NOT ${execute_result} EQUAL 0)
12     message(FATAL_ERROR
13         "Execution of '${COMMAND}' failed with result"
14         "${execute_result}\n"
15         "${execute_output}\n")
16 endif()
17 endif()
```

Done

We have a "silent" build!

```
build $ cmake --build .
[ 2%] Automatic UIC for target my_lib
[ 2%] Built target my_lib_autogen
[ 36%] Built target my_lib
[ 55%] Built target logger
[ 57%] Creating Pro*C/C++ output src/my_exe/proc/DbCommunicationError.pcpp.cpp
[ 59%] Creating Pro*C/C++ output src/my_exe/proc/DbCommunication.pcpp.cpp
[ 61%] Automatic UIC for target my_exe
[ 61%] Built target my_exe_autogen
Scanning dependencies of target my_exe
[ 63%] Building CXX object src/my_exe/CMakeFiles/my_exe.dir/proc/DbCommunication.pcpp.cpp.o
[ 65%] Building CXX object src/my_exe/CMakeFiles/my_exe.dir/proc/DbCommunicationError.pcpp.cpp.o
[ 68%] Linking CXX executable my_exe
[100%] Built target my_exe
build $
```

References:

- Craig Scott: "Professional CMake: A Practical Guide"
- CMake Reference Documentation:
<https://cmake.org/cmake/help/latest/index.html>

Thank you

- E-mail: c++@closingbrace.nl
- Linkedin: <https://www.linkedin.com/in/hans-vredeveld-88510b49/>

References:

- Craig Scott: "Professional CMake: A Practical Guide"
- CMake Reference Documentation:
<https://cmake.org/cmake/help/latest/index.html>

