Hacking modern CMake

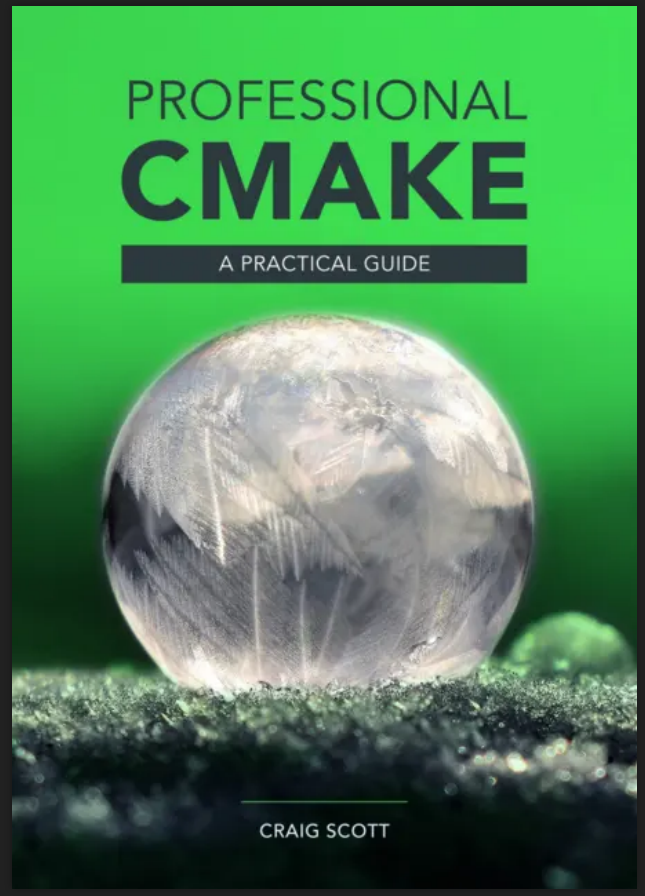
Alexander Voronkov, Senior Software Developer, Align Technology

Agenda

- A few theoretical topics
- Generating files on different stages
- Separate debug symbols
- CMake and PCH
- Resource management with Ninja build system
- Dynamic library specific stuff

The book

https://crascit.com/professional-cmake/



CMake is a code

- Treat CMake language files as an application that builds your project
- Develop CMake sources like any other application
- Make the convenient interfaces

Stages of CMake project workflow

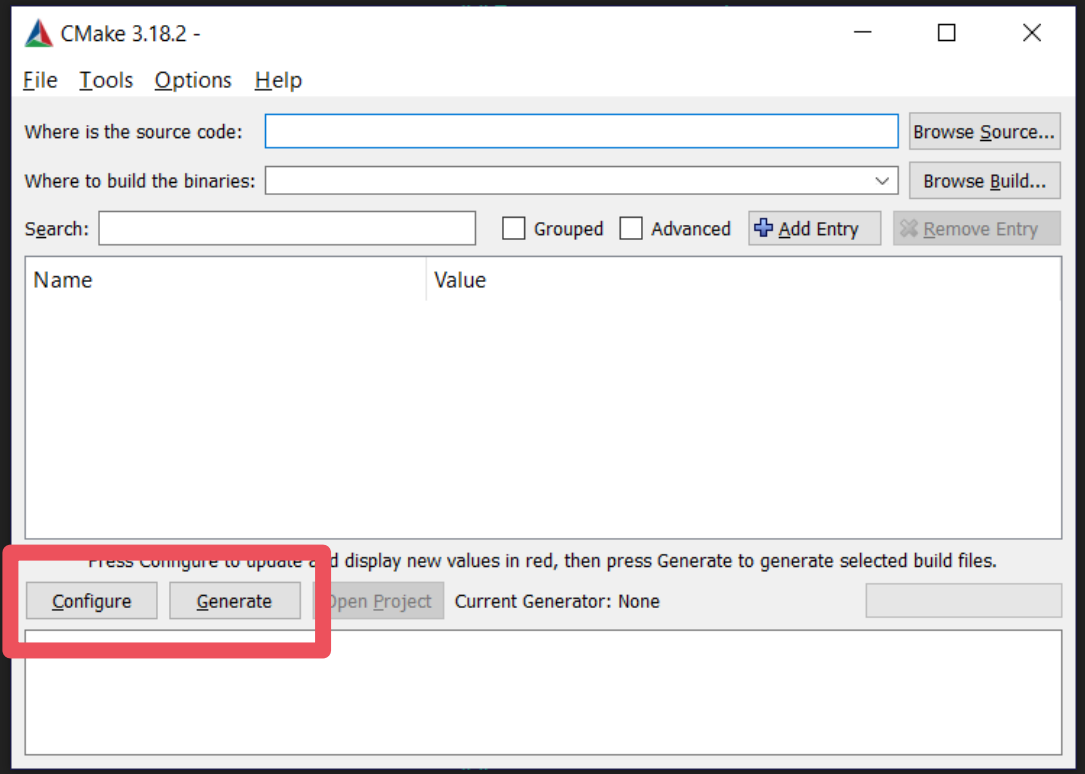
Stages

- Configuration
- Generation
- Build
- Running tests
- Installation
- Packaging

Configuration & Generation by CLI

```
cmake -G Ninja ~/MyProjectSrc
```

Configuration & Generation in the UI



Configuration & Generation

Configuration

- Execution of scripts
  - Creation of targets
  - Filling in the properties
  - The CMake cache is populated
  - *Generator expressions aren't expanded*

Generation

- Generation of underlying build system files
  - Generator expressions are expanded
  - Calculation of the final property values
  - Custom generation-step actions are performed
  - The selected generator is executed

Generator expressions

Generator expressions

- Special strings $<SOMETHING>
- May by nested
- Can be used only in specific properties context
- The count of the allowed properties is increased on each CMake release

Generator expressions types

- Boolean – may be used only as a condition
- String-valued – may be used to produce the final value

Conditionals

- $<condition:true_string>
- $<IF:condition, true_string, false_string>

Property values

- Property values one of the main generator expression value sources
- Target property
  - `$<TARGET_PROPERTY:prop>`
  - `$<TARGET_PROPERTY:target, prop>`
- The property value itself may be a generator expression, if so it must be explicitly evaluated
  - `$<GENEX_EVAL:$<TARGET_PROPERTY:target1, prop>>`
  - `$<TARGET_GENEX_EVAL:target2, $<TARGET_PROPERTY:target1, prop>>`

Value processing

- String case transform `$<LOWER_CASE:string>` , `$<UPPER_CASE:string>`
- List transform
  - `$<JOIN:list,string>`
  - `$<REMOVE_DUPLICATES:list>`
  - `$<FILTER:list,INCLUDE,regex>`
  - `$<FILTER:list,EXCLUDE,regex>`
  - Sorting is not implemented yet ☹

Escaping

- $<ANGLE-R>
- $<COMMA>
- $<SEMICOLON>
- Don't forget to escape variable values
  - `string(REPLACE ">" "$<ANGLE-R>" var1 "${var1}")`
  - `string(REPLACE ";" "$<SEMICOLON>" var1 "${var1}")`
  - `string(REPLACE "," "$<COMMA>" var1 "${var1}")`

Generated files

Generated files during configuration step

- Without dependencies creation – executed only on configuration but each time:
  - `file(WRITE <filename> <content>)`
  - `file(APPEND <filename> <content>)`
  - Since CMake 3.18
    `file(CONFIGURE OUTPUT <output-file> CONTENT <content> )`
- With dependencies creation – input change triggers reconfiguration
  - `configure_file(<input-file> <output-file>)`

Autotools-like config.h

- The template file – config.h.in
- Find modules – used by find_package
- Other checks (see cmake-modules documentation, modules with ‘Check*’ names)
- Variables – template parameters

CMakeLists.txt

```
project(MySample LANGUAGES CXX VERSION 0.1.1)

find_package(BZip2)
find_package(ZLIB)
find_package(LibLZMA)
find_package(SQLite3)

set(HAVE_BZIP2 ${BZip2_FOUND})
set(HAVE_ZLIB ${ZLIB_FOUND})
set(HAVE_LZMA ${LibLZMA_FOUND})
set(HAVE_SQLITE ${SQLite3_FOUND})

include(CheckCXXSymbolExists)
check_cxx_symbol_exists(MD5Init md5.h HAVE_MD5)
check_cxx_symbol_exists(chmod io.h HAVE_CHMOD)

configure_file(config.h.in config.h @ONLY)
```

config.h.in

```
#define PACKAGE_NAME "@CMAKE_PROJECT_NAME@"
#define PACKAGE_VERSION "@CMAKE_PROJECT_VERSION@"
#define PACKAGE_STRING \
        "@CMAKE_PROJECT_NAME@ @CMAKE_PROJECT_VERSION@"
#cmakedefine HAVE_BZIP2
#cmakedefine HAVE_ZLIB
#cmakedefine HAVE_LZMA 1
#cmakedefine HAVE_SQLITE 1
#cmakedefine01 HAVE_MD5
#cmakedefine01 HAVE_CHMOD
```

←TRUE

←TRUE

←TRUE

config.h

```
#define PACKAGE_NAME "MySample"
#define PACKAGE_VERSION "0.1.1"
#define PACKAGE_STRING \
        "MySample 0.1.1"
/* #undef HAVE_BZIP2 */
#define HAVE_ZLIB
/* #undef HAVE_LZMA */
#define HAVE_SQLITE 1
#define HAVE_MD5 0
#define HAVE_CHMOD 1
```

CMakeLists.txt

```
project(MySample LANGUAGES CXX VERSION 0

find_package(BZip2)
find_package(ZLIB)
find_package(LibLZMA)
find_package(SQLite3)

set(HAVE_BZIP2 ${BZip2_FOUND})
set(HAVE_ZLIB ${ZLIB_FOUND})
set(HAVE_LZMA ${LibLZMA_FOUND})
set(HAVE_SQLITE ${SQLite3_FOUND})

include(CheckCXXSymbolExists)
check_cxx_symbol_exists(MD5Init md5.h HA
check_cxx_symbol_exists(chmod io.h HAVE_

configure_file(config.h.in config.h @ONL
```

Don’t overdo the checks

- Fast configuration is very important
- Know your platforms
- Set the configuration variables according to the platform
- Use toolchain files and include configuration variables there

Generated files at generation step

Generated files at generation step

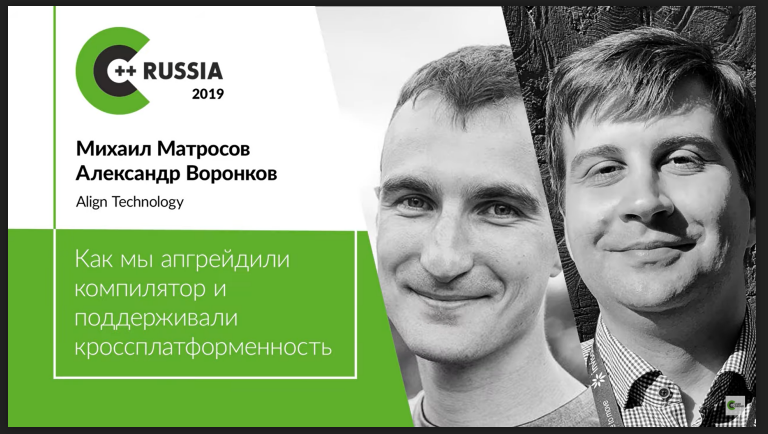
- `file(GENERATE OUTPUT <output filename> CONTENT <content>)`
- `file(GENERATE OUTPUT <output filename> INPUT <input filename>)`
- Both output filename and content may include generator expressions
- May include generator expression – condition
  `file(GENERATE … CONDITION $<$<CONFIG>:Release>)`
- Since CMake 3.19 the target for context of expressions evaluation may be set
  `file(GENERATE … TARGET MyLib)`
- The files are not created until generation phase
- Variables are not expanded, only generator expressions

The fake target's properties hack

- The template can't use the variables substitution on the generation step
- Solution:
  - Create the UNKNOWN IMPORTED GLOBAL target
  - Collect the data needed into its properties
  - Consume the properties by the generator expressions on the generation phase

Example – generation of Conan file

- We are using Conan package manager
- For each target we are using a helper function to add the required Conan packages
- This helper adds used Conan packages information to the global ;-list cache variable and updates our fake target properties
- The Conan file that will copy Conan-supplied libraries during install stage is created on the generation stage



https://youtu.be/PGOvl0dH8IM

conan_support.cmake

```
function(update_conan_interface)
    if(NOT TARGET ConanInterface::Data)
        add_library(ConanInterface::Data UNKNOWN IMPORTED GLOBAL)
    endif()
    set_target_properties(ConanInterface::Data PROPERTIES
        CONAN_INSTALLABLE_PACKAGES "${CONAN_INSTALLABLE_PACKAGES}"
        CONAN_USED_REF_LIST "${CONAN_USED_REF_LIST}"
        )
endfunction()
```

install_conanfile.py.in

```
class GeneratedInstallConanfile(ConanFile):
    requires = tuple(
        "$<TARGET_PROPERTY:ConanInterface::Data,CONAN_USED_REF_LIST>".split(";"))
    …
    def imports(self):
        conan_installable_packages = (
            "$<TARGET_PROPERTY:ConanInterface::Data,CONAN_INSTALLABLE_PACKAGES>")
```

Debug symbols

Debug symbols on Windows: PDB

- The debug information for DLL/Exe is stored in the separate PDB files
- In general PDB files are uploaded to the symbol server
- CMake controls PDB location by the properties PDB_OUTPUT_DIRECTORY and PDB_OUTPUT_DIRECTORY_<config>
- Since CMake 3.12 generator expressions are supported, so prefer PDB_OUTPUT_DIRECTORY
- The properties may be initialized globally by CMAKE_PDB_OUTPUT_DIRECTORY variable

```
install(DIRECTORY "${CMAKE_PDB_OUTPUT_DIRECTORY}/" DESTINATION Debug
    COMPONENT DebugInfo FILES_MATCHING PATTERN "*.pdb" )
```

Separate debug symbols on Linux

- The debug information is embedded to built binary
- May be stripped to make the binary without debug symbols
- May be copied to another ELF, that may contain only debug symbols
- Modern compilers produce build-id information to identify the built binary and its symbols
- To perform build-id symbols lookup the debug information shall be placed to path `<debug info dir>/.build-id/ab/cdef1234.debug`

target_utils.cmake https://github.com/holybrake/cpprussia2020piter

```
set(MY_MODULE_DIR "${CMAKE_CURRENT_LIST_DIR}")
function(configure_install_for_target target install_dir)
    install(TARGETS ${target}
        RUNTIME DESTINATION ${install_dir} COMPONENT Binaries
        LIBRARY DESTINATION ${install_dir} COMPONENT Binaries)
    if(CMAKE_STRIP)
        find_program(READELF_EXECUTABLE readelf)
        if(READELF_EXECUTABLE)
            set(_debug_info_script_prefix ${CMAKE_CURRENT_BINARY_DIR}/${target}.debug_info)
            configure_file(${MY_MODULE_DIR}/debug_info.cmake.in
                ${_debug_info_script_prefix}.in @ONLY)
            file(GENERATE OUTPUT ${_debug_info_script_prefix}.$<CONFIG>.cmake
                INPUT ${_debug_info_script_prefix}.in)
            install(SCRIPT ${_debug_info_script_prefix}.\${CMAKE_INSTALL_CONFIG_NAME}.cmake
                    COMPONENT DebugInfo)
        else()
            message(WARNING "readelf executable not found")
        endif()
    endif()
endfunction()
```

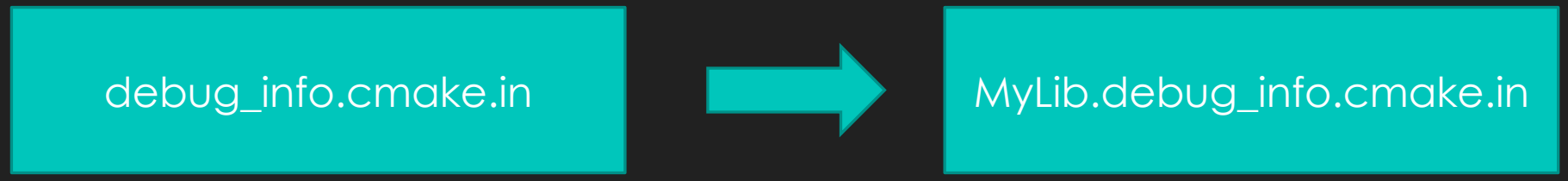
debug_info.cmake.in

```
if(CMAKE_INSTALL_DO_STRIP)
    execute_process(COMMAND "@READELF_EXECUTABLE@“
                    -W -n "$<TARGET_FILE:@target@>"
                    RESULT_VARIABLE ret  OUTPUT_VARIABLE readelf_out)
    if(ret)
        message(FATAL_ERROR "readelf run failed")
    endif()
    string(REPLACE "\n" ";" readelf_out ${readelf_out})
    set(build_id_dir)
    set(build_id_filename)
    foreach(line IN LISTS readelf_out)
        if(line MATCHES "NT_GNU_BUILD_ID.*Build ID: ([0-9a-f][0-9a-f])([0-9a-f]+)$")
            set(build_id_dir ${CMAKE_MATCH_1})
            set(build_id_filename ${CMAKE_MATCH_2})
            break()
        endif()
    endforeach()
    set(out_dir $ENV{DESTDIR}${CMAKE_INSTALL_PREFIX}/debug/.build-id/${build_id_dir})
    file(MAKE_DIRECTORY ${out_dir})
    execute_process(COMMAND "@CMAKE_STRIP@"
        --only-keep-debug "$<TARGET_FILE:@target@>"
        -o "${out_dir}/${build_id_filename}.debug")
endif()
```

Configuring install of binaries

```
install(TARGETS ${target}
    RUNTIME DESTINATION ${install_dir} COMPONENT Binaries
    LIBRARY DESTINATION ${install_dir} COMPONENT Binaries)
```

Preparing intermediate template




```
set(_debug_info_script_prefix ${CMAKE_CURRENT_BINARY_DIR}/${target}.debug_info)
configure_file(${MY_MODULE_DIR}/debug_info.cmake.in
    ${_debug_info_script_prefix}.in @ONLY)
```

debug_info.cmake.in

```
if(CMAKE_INSTALL_DO_STRIP)
    execute_process(COMMAND "@READELF_EXECUTABLE@"
                    -W -n "$<TARGET_FILE:@target@>"
                    RESULT_VARIABLE ret  OUTPUT_VARIABLE readelf_out)
    if(ret)
        message(FATAL_ERROR "readelf run failed")
    endif()
    string(REPLACE "\n" ";" readelf_out ${readelf_out})
    set(build_id_dir)
    set(build_id_filename)
    foreach(line IN LISTS readelf_out)
        if(line MATCHES "NT_GNU_BUILD_ID.*Build ID: ([0-9a-f][0-9a-f])([0-9a-f]+)$")
            set(build_id_dir ${CMAKE_MATCH_1})
            set(build_id_filename ${CMAKE_MATCH_2})
            break()
        endif()
    endforeach()
    set(out_dir $ENV{DESTDIR}${CMAKE_INSTALL_PREFIX}/debug/.build-id/${build_id_dir})
    file(MAKE_DIRECTORY ${out_dir})
    execute_process(COMMAND "@CMAKE_STRIP@"
        --only-keep-debug "$<TARGET_FILE:@target@>"
        -o "${out_dir}/${build_id_filename}.debug")
endif()
```

MyLib.debug_info.cmake.in

```
if(CMAKE_INSTALL_DO_STRIP)
    execute_process(COMMAND "/opt/rh/devtoolset-7/root/usr/bin/readelf"
                    -W -n "$<TARGET_FILE:MyLib>"
                    RESULT_VARIABLE ret  OUTPUT_VARIABLE readelf_out)
    if(ret)
        message(FATAL_ERROR "readelf run failed")
    endif()
    string(REPLACE "\n" ";" readelf_out ${readelf_out})
    set(build_id_dir)
    set(build_id_filename)
    foreach(line IN LISTS readelf_out)
        if(line MATCHES "NT_GNU_BUILD_ID.*Build ID: ([0-9a-f][0-9a-f])([0-9a-f]+)$")
            set(build_id_dir ${CMAKE_MATCH_1})
            set(build_id_filename ${CMAKE_MATCH_2})
            break()
        endif()
    endforeach()
    set(out_dir $ENV{DESTDIR}${CMAKE_INSTALL_PREFIX}/debug/.build-id/${build_id_dir})
    file(MAKE_DIRECTORY ${out_dir})
    execute_process(COMMAND "/opt/rh/devtoolset-7/root/usr/bin/strip"
        --only-keep-debug "$<TARGET_FILE:MyLib>"
        -o "${out_dir}/${build_id_filename}.debug")
endif()
```

Creating the script during generation step

MyLib.debug_info.cmake.in

MyLib.debug_info.Release.cmake

```
file(GENERATE OUTPUT ${_debug_info_script_prefix}.$<CONFIG>.cmake
     INPUT ${_debug_info_script_prefix}.in)
```

MyLib.debug_info.cmake.in

```
if(CMAKE_INSTALL_DO_STRIP)
    execute_process(COMMAND "/opt/rh/devtoolset-7/root/usr/bin/readelf"
                    -W -n "$<TARGET_FILE:MyLib>"
                    RESULT_VARIABLE ret  OUTPUT_VARIABLE readelf_out)
    if(ret)
        message(FATAL_ERROR "readelf run failed")
    endif()
    string(REPLACE "\n" ";" readelf_out ${readelf_out})
    set(build_id_dir)
    set(build_id_filename)
    foreach(line IN LISTS readelf_out)
        if(line MATCHES "NT_GNU_BUILD_ID.*Build ID: ([0-9a-f][0-9a-f])([0-9a-f]+)$")
            set(build_id_dir ${CMAKE_MATCH_1})
            set(build_id_filename ${CMAKE_MATCH_2})
            break()
        endif()
    endforeach()
    set(out_dir $ENV{DESTDIR}${CMAKE_INSTALL_PREFIX}/debug/.build-id/${build_id_dir})
    file(MAKE_DIRECTORY ${out_dir})
    execute_process(COMMAND "/opt/rh/devtoolset-7/root/usr/bin/strip"
        --only-keep-debug "$<TARGET_FILE:MyLib>"
        -o "${out_dir}/${build_id_filename}.debug")
endif()
```

MyLib.debug_info.Release.cmake

```
if(CMAKE_INSTALL_DO_STRIP)
    execute_process(COMMAND "/opt/rh/devtoolset-7/root/usr/bin/readelf"
                    -W -n "/home/build/build/Output.Release/lib/libMyLib.so"
                    RESULT_VARIABLE ret  OUTPUT_VARIABLE readelf_out)
    if(ret)
        message(FATAL_ERROR "readelf run failed")
    endif()
    string(REPLACE "\n" ";" readelf_out ${readelf_out})
    set(build_id_dir)
    set(build_id_filename)
    foreach(line IN LISTS readelf_out)
        if(line MATCHES "NT_GNU_BUILD_ID.*Build ID: ([0-9a-f][0-9a-f])([0-9a-f]+)$")
            set(build_id_dir ${CMAKE_MATCH_1})
            set(build_id_filename ${CMAKE_MATCH_2})
            break()
        endif()
    endforeach()
    set(out_dir $ENV{DESTDIR}${CMAKE_INSTALL_PREFIX}/debug/.build-id/${build_id_dir})
    file(MAKE_DIRECTORY ${out_dir})
    execute_process(COMMAND "/opt/rh/devtoolset-7/root/usr/bin/strip"
        --only-keep-debug "/home/build/build/Output.Release/lib/libMyLib.so"
        -o "${out_dir}/${build_id_filename}.debug")
endif()
```

```
install(SCRIPT ${_debug_info_script_prefix}.\${CMAKE_INSTALL_CONFIG_NAME}.cmake
        COMPONENT DebugInfo)
```

cmake_install.cmake

```
if("x${CMAKE_INSTALL_COMPONENT}x" STREQUAL "xDebugInfox" OR NOT CMAKE_INSTALL_COMPONENT)
  include("/home/build/build/MyLib/MyLib.debug_info.${CMAKE_INSTALL_CONFIG_NAME}.cmake")
endif()
```

Read build-id

```
execute_process(COMMAND "/opt/rh/devtoolset-7/root/usr/bin/readelf"
                -W -n "/home/build/Output.Release/lib/libMyLib.so"
                RESULT_VARIABLE ret  OUTPUT_VARIABLE readelf_out)
if(ret)
    message(FATAL_ERROR "readelf run failed")
endif()
```

```
Displaying notes found in: .note.gnu.build-id
  Owner                Data size       Description
  GNU                  0x00000014      NT_GNU_BUILD_ID (unique build ID bitstring)        Build ID: 12e4c332022a01793dbdb30a9134a426f88261af
```

Parse and split build-id

```
string(REPLACE "\n" ";" readelf_out ${readelf_out})
set(build_id_dir)
set(build_id_filename)
foreach(line IN LISTS readelf_out)
    if(line MATCHES "NT_GNU_BUILD_ID.*Build ID: ([0-9a-f][0-9a-f])([0-9a-f]+)$")
        set(build_id_dir ${CMAKE_MATCH_1})
        set(build_id_filename ${CMAKE_MATCH_2})
        break()
    endif()
endforeach()
```

Save debug info

```
set(out_dir $ENV{DESTDIR}${CMAKE_INSTALL_PREFIX}/debug/.build-id/${build_id_dir})
file(MAKE_DIRECTORY ${out_dir})
execute_process(COMMAND "/opt/rh/devtoolset-7/root/usr/bin/strip"
    --only-keep-debug "/home/build/build/Output.Release/lib/libMyLib.so"
    -o "${out_dir}/${build_id_filename}.debug")
```

CPack

The common approach

- Group you `install()` commands by components
- Configure all necessary CPACK_* variable before including CPack module
- At the end of main CMakeLists.txt
  ```
  include(CPack)
  ```
- After build and test run CPack
  ```
  cpack –C <configuration>
  ```
  or
  ```
  cpack -G <generator> –C <configuration>
  ```

Simple archive

```
set(CPACK_PACKAGE_VENDOR "ACME Inc.")
set(CPACK_PACKAGE_DESCRIPTION "${CMAKE_PROJECT_NAME} Package")

if(UNIX)
    set(CPACK_GENERATOR TGZ)
    set(CPACK_STRIP_FILES ON)
else()
    set(CPACK_GENERATOR ZIP)
endif()
include(CPack)
```

Components

```
install(TARGETS ${target}
    RUNTIME DESTINATION ${install_dir} COMPONENT Binaries
    LIBRARY DESTINATION ${install_dir} COMPONENT Binaries)
```

```
install(SCRIPT ${_debug_info_script_prefix}.\${CMAKE_INSTALL_CONFIG_NAME}.cmake
        COMPONENT DebugInfo)
```

Multiple archives

```
set(CPACK_PACKAGE_VENDOR "ACME Inc.")
set(CPACK_PACKAGE_DESCRIPTION "${CMAKE_PROJECT_NAME} Package")

set(CPACK_ARCHIVE_COMPONENT_INSTALL ON)

if(UNIX)
    set(CPACK_GENERATOR TGZ)
    set(CPACK_STRIP_FILES ON)
else()
    set(CPACK_GENERATOR ZIP)
endif()
include(CPack)
```

Precompiled Headers (PCH)

PCH workflow

PCH out-of-the box since CMake 3.16

- `target_precompile_headers(<target>  <INTERFACE|PUBLIC|PRIVATE> [header1]...)`
- Fills in PRECOMPILE_HEADERS and/or INTERFACE_PRECOMPILE_HEADERS properties
- Properties are used to generate precompiled header source
- One target may reuse PCH artifact of another, but compiler flags should coincide
  `target_precompile_headers(<target> REUSE_FROM <other_target>)`
- Generator expressions may be used for these properties

Juggling the resources
Ninja & Job Pools

Limiting the concurrent jobs count

- Ninja has total jobs count limit like Make: `ninja -j <count>`
- Job pool – limits max count of the concurrent jobs belonging to the same pool

CMake support for Ninja job pools

- Global property JOB_POOLS list of key-value pairs describing the pools
- Target properties JOB_POOL_COMPILE and JOB_POOL_LINK
- Custom commands/target option JOB_POOL (since CMake 3.15)

Memory limited pools setup example

```
function(setup_job_pool name mem_per_task)
    math(EXPR res "${mem_avail} / ${mem_per_task}")
    if(res LESS 1)
        set(res 1)
    endif()
    message(STATUS "Job pool ${name}:\t  max job count: ${res}")
    set_property(GLOBAL APPEND PROPERTY JOB_POOLS ${name}=${res})
endfunction()
```

```
function(setup_job_pools)
    cmake_host_system_information(RESULT mem_avail QUERY AVAILABLE_PHYSICAL_MEMORY)
    message(STATUS "Availiable physical memory: ${mem_avail} MB")
    if(mem_avail LESS 2000)
        message(FATAL_ERROR "Insufficient free memory, at least 2.5GiB needed")
    endif()

    setup_job_pool(light_tasks 500)
    setup_job_pool(medium_tasks 800)
    setup_job_pool(heavy_tasks 1350)
    set(CMAKE_JOB_POOL_COMPILE light_tasks PARENT_SCOPE)
    set(CMAKE_JOB_POOL_LINK medium_tasks PARENT_SCOPE)
endfunction()


…
setup_job_pools()
…
set_property(TARGET MyLib PROPERTY JOB_POOL_COMPILE medium_tasks)
set_property(TARGET MyLib PROPERTY JOB_POOL_LINK heavy_tasks)
```

Dynamic library-specific stuff

Symbols visibility problem

- Windows hides symbols in the dynamic library by default
- Linux, macOS exports all symbols in the dynamic library by default

To hide or not to hide

- To hide, pro:
  - The library has an explicit public interface
  - Faster linking
  - Mitigate the possible ODR violations if the symbols are the same in different modules
- To hide, contra:
  - The internal symbols aren't available for unit tests
  - The developer has to create an explicit public interface
  - *Possible exception RTTI issues for old GCC*

Export all library symbols on Windows

- Turn on for specific target
  ```
  set_property(TARGET <target> PROPERTY WINDOWS_EXPORT_ALL_SYMBOLS ON)
  ```
- Turn on for all shared libraries in the project by default
  ```
  set(CMAKE_WINDOWS_EXPORT_ALL_SYMBOLS ON)
  ```

Hide symbols by default for GCC/Clang

- Turn on for specific target

```
set_target_properties(<target> PROPERTIES CXX_VISIBILITY_PRESET hidden
                                          C_VISIBILITY_PRESET hidden
                                          VISIBILITY_INLINES_HIDDEN ON)
```

- Turn on for all targets by default

```
set(CMAKE_CXX_VISIBILITY_PRESET hidden)
set(CMAKE_C_VISIBILITY_PRESET hidden)
set(CMAKE_VISIBILITY_INLINES_HIDDEN ON)
```

Import/Export macros

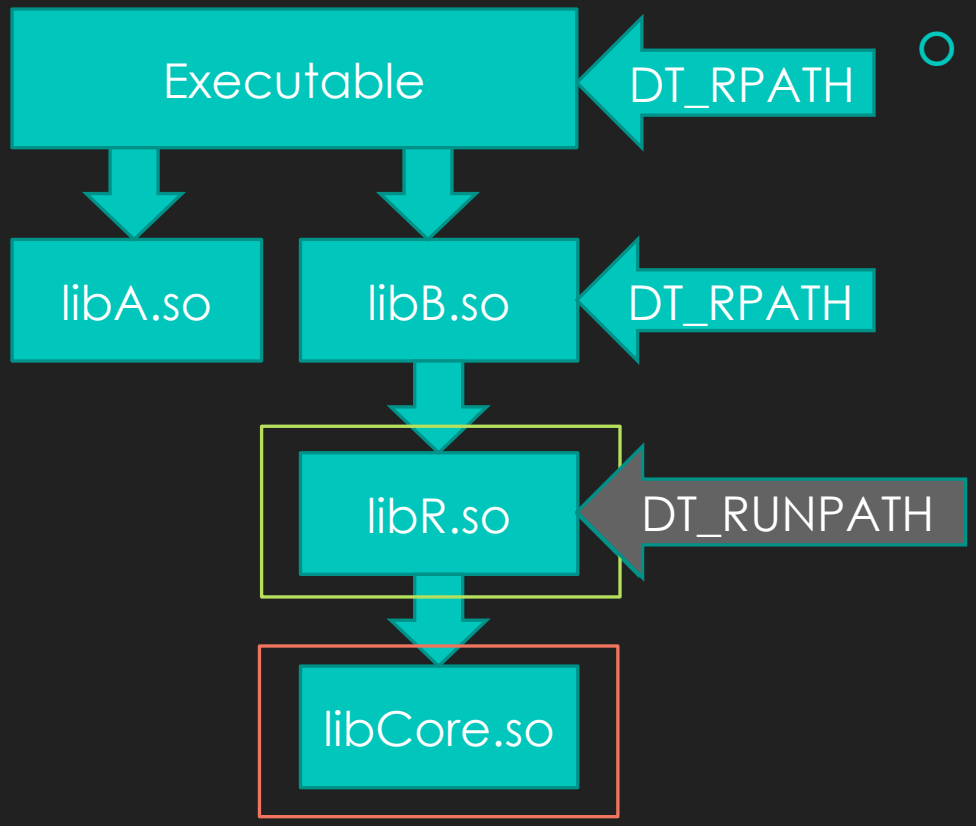
- The preprocessor definition <target>_EXPORTS is created by default for each shared library target
- Automatic import/export header generation

```
include(GenerateExportHeader)
generate_export_header(MyLib)
set_property(TARGET MyLib PROPERTY PUBLIC_HEADER
             ${CMAKE_CURRENT_BINARY_DIR}/MyLib_export.h APPEND)
target_include_directories(MyLib PUBLIC
             $<BUILD_INTERFACE:${CMAKE_CURRENT_BINARY_DIR}>)
```

Linux/ELF runtime search path

- Linux library search paths may be hardcoded into the ELF binary
- The paths are contained in the .dynamic section attribute
- The old-style attribute DT_RPATH had been used a long time
- The new DT_RUNPATH is superseded it but having a different behavior

Linux: dynamic libraries search



- Old-style DT_RPATH search order:
  - DT_RPATH of the current ELF
  - DT_RPATH of each 'parent'
  - LD_LIBRARY_PATH
  - /etc/ld.so.cache
  - /lib
  - /usr/lib
- New DT_RUNPATH search order
  - LD_LIBRARY_PATH
  - DT_RUNPATH of the current ELF
  - /etc/ld.so.cache
  - /lib
  - /usr/lib

Selecting RPATH or RUNPATH

- DT_RPATH:
  `target_link_options(<target> PRIVATE -Wl,--disable-new-dtags)`
- DT_RUNPATH:
  `target_link_options(<target> PRIVATE -Wl,--enable-new-dtags)`
- Keep in mind that DT_RPATH is deprecated

RPATH support properties

- BUILD_RPATH – The value of RPATH after the build stage
- INSTALL_RPATH – The value of RPATH after the install stage
- BUILD_WITH_INSTALL_RPATH – use INSTALL_RPATH value for the build
- BUILD_RPATH_USE_ORIGIN – Use relative RPATH for build (since CMake 3.14)
- SKIP_BUILD_RPATH – don't write RPATH after the build
- INSTALL_REMOVE_ENVIRONMENT_RPATH – remove additional toolset-depended RPATH
- INSTALL_RPATH_USE_LINK_PATH – append the paths used for linking to RPATH

RPATH support variables

- The same names as the properties but with CMAKE_ prefix to initialize the corresponding properties globally
- CMAKE_SKIP_RPATH – turn off any RPATH handling
- CMAKE_SKIP_INSTALL_RPATH – clear RPATH value at install stage

Relative RPATH/RUNPATH

- Use $ORIGIN as the current ELF path substitution
- Avoid the form ${ORIGIN} – the proper escaping has been fixed only in 3.16
- Universal simple solution if the project uses FHS inside the install folder

```
set(CMAKE_INSTALL_RPATH [[$ORIGIN]] [[$ORIGIN/../lib]])
```

Library paths: macOS

- Similar to Linux but *install_name* Mach-O header is used for dependency search
- To allow relative dependency path *install_name* shall contain *@rpath* substitution
- Only executable has *rpath*
- *rpath* may contain *@loader_path* and *@executable_path* substitutions

RPATH/INSTALL_NAME support properties

- INSTALL_NAME_DIR – set the folder part of *install_name* to this value during library install
- BUILD_WITH_INSTALL_NAME_DIR – set the folder part of *install_name* to the value of INSTALL_NAME_DIR property during the build stage
- MACOSX_RPATH – if TRUE (by default) than default value of the folder part of *install_name* is set to the value *@rpath/*

Simple relative RPATH solution

```
set(CMAKE_INSTALL_RPATH [[@executable_path]] [[@executable_path/../lib]])


set(rpath_origin [[$ORIGIN]])
if(APPLE)
   set(rpath_origin [[@executable_path]])
endif()
set(CMAKE_INSTALL_RPATH ${rpath_origin} ${rpath_origin}/../lib)
```

Thank you!

Bonus

CMake language project entities

- Commands
- Comments
- Variables
- Cache variables
- Modules
- Directories
- Functions
- Macros
- Policies
- Targets
- Properties
  - Properties of Global Scope
  - Properties on Directories
  - Properties on Targets
  - Properties on Tests
  - Properties on Source Files
  - Properties on Cache Entries
  - Properties on Installed Files

Substitutions

- CMake Variable ${MY_VARIABLE_NAME}
- Environment variable $ENV{MY_VARIABLE_NAME}
- CMake Cache variable $CACHE{MY_VARIABLE_NAME}
  - Syntax exists since CMake 2.6.3 RC 7 but documented only since CMake 3.13
  - Documentation amendment since CMake 3.14 for '`if`' command

There is no automatic evaluation for environment or cache Variable References. Their values must be referenced as `$ENV{<name>}` or `$CACHE{<name>}` wherever the above-documented condition syntax accepts `<variable|string>`.

- Generator expressions $<SOMETHING>
- Template placeholder @MY_VARIABLE_NAME@

Custom targets, custom commands

The difference

- Custom command has output files,
  custom target has no outputs
- Custom command has only file-level dependencies,
  custom target has file-level and target-level dependencies
- Custom target is executed always,
  custom command is executed if and only if its outputs required to build another target

A few tricks

- The files shall be produced during the build, but only if are outdated
- The custom command requires some cross-platform filesystem manipulation
- The custom command requires some cross-platform script
- One custom command produces files that are used by different targets, so underlying build system has multiple copies of this custom command
- Use custom command to produce files and custom target that depends on them
- Use cmake command line tool `${CMAKE_COMMAND} -E <command>`
- Use cmake-language script `${CMAKE_COMMAND} -P <command>`
- If a file that is consumed by only one target exists among the produced ones, it may be the only one in OUTPUT parameter, and other ones should be moved to BYPRODUCTS parameter. In other case the intermediate custom target shall be created.

- The custom command output has a generator expression
- Generator expressions aren't supported in output parameters (probably will be implemented in 3.20), but if this expression is `$<CONFIG>` the workaround exists

```
get_property(is_multi_config GLOBAL PROPERTY GENERATOR_IS_MULTI_CONFIG)
if(is_multi_config)
    set(conf_subst "${CMAKE_CFG_INTDIR}")
else()
    set(conf_subst "${CMAKE_BUILD_TYPE}")
endif()
string(REPLACE "$<CONFIG>" "${conf_subst}" output_files "${output_files}")
```

- The alternate solution – fake output file and undeclared real outputs