CMake Tutoriál

Tomáš Milet

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole
imilet@fit.vutbr.cz

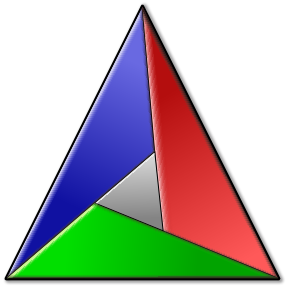
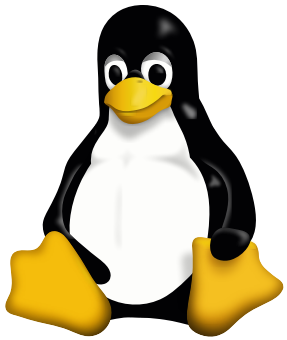
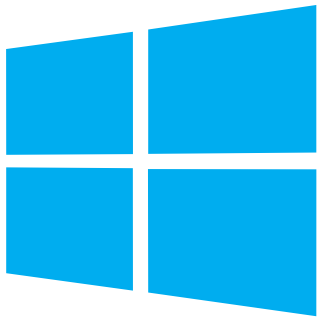
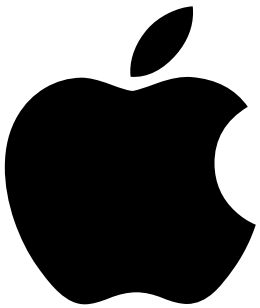
27. října 2018

Úvod

Zdroje informací

- https://cmake.org/cmake/help/v3.13/
- https://www.youtube.com/watch?v=bsXLMQ6WgIk
- https://pabloariasal.github.io/2018/02/19/its-time-to-do-cmake-right/

- Multiplatformní
- Usnadňuje práci (závislosti, nastavení, ...)
- Standardizuje sestavování
- Testy, balíčky, uživatelské skripty, ...
- Out of source building, ...

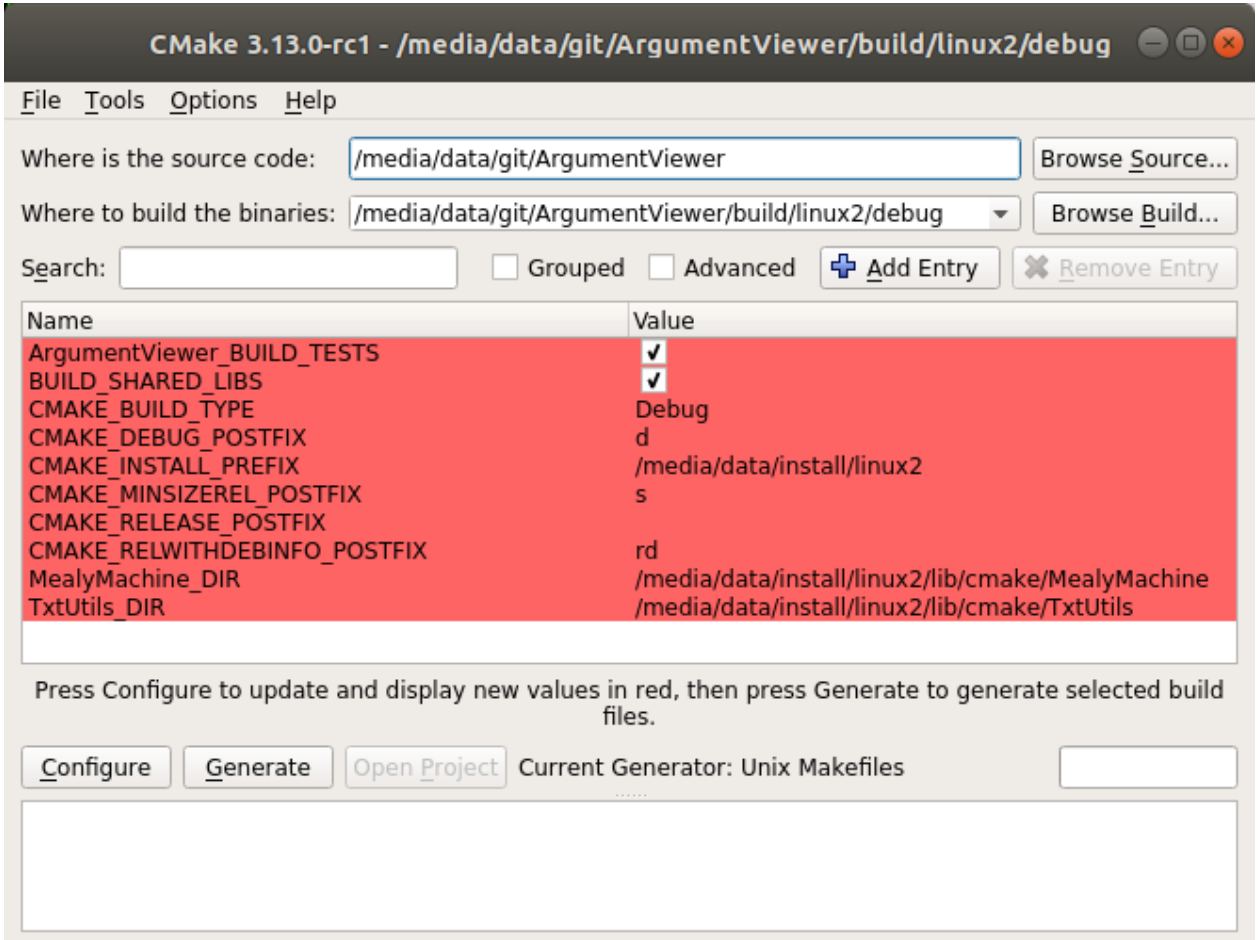
Jak nainstalovat pod Linuxem?

1. Odinstalovat všechny starší verze (najít třeba pomocí $ **which** cmake)
2. Stáhnout nejnovější binárni tar.gr verzi z `https://cmake.org/download/`
3. Překopírovat obsah rozbaleného adresáře do /usr

```
$ wget https://cmake.org/files/v3.13/cmake-3.13.0-rc1-Linux-x86_64.tar.gz
$ tar xf cmake-3.13.0-rc1-Linux-x86_64.tar.gz
$ cd cmake-3.13.0-rc1-Linux-x86_64
$ sudo cp -r * /usr
$ cmake --version
cmake version 3.13.0-rc1
```

Jak použít?

1. cmake-gui / ccmake
2. Nastavit místo, kde se sestavuje a kde je složka s CMakeLists.txt
3. Configure
4. Vyřešit závislosti - například nastavit SDL2_DIR na složku obsahující SDL2Config.cmake
5. Generate - vygenerovat makefile / sln file
6. make / pustit MSVS



Základní návod

- Složka obsahující CMakeLists.txt je složka, kterou je možné sestavit
- Hierarchie složek
- CMake sám o sobě nesestavuje aplikace/knihovny
- CMake slouží pro generování makefile / sln souborů
- CMake obsahuje aplikace **cmake cmake-gui ccmake ctest cpack**
- **cmake** - rovnou vygeneruje makefile (parametery můžou ovlivnit generování
- **cmake-gui** / **ccmake** - uživatelské nastavní parametrů a následné vygenerování makefile
- **ctest** - pro spouštění testů (unit tests, ...)
- **cpack** - pro vytvoření balíčků (*.deb, ...)

Prázdný CMake

CMakeLists.txt:

```
# minimum version, always use the newest cmake
cmake_minimum_required(VERSION 3.13.0)
```

HelloWorld CMake

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.13.0)

# this command will add application HelloWorld to build
add_executable(HelloWorld main.cpp)
```

main.cpp:

```
#include<iostream>

int main(int,char*[]){
  std::cout << "Hello world!" << std::endl;
  return 0;
}
```

CMake umožňuje out-of-source building - vytváření *.o souborů mimo *.cpp soubory.

```
$ ls
CMakeLists.txt main.cpp
$ mkdir build
$ cd build
$ cmake ..
$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  Makefile
$ make
$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  HelloWorld  M
```

Include Directories

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.12.2)

add_executable(HelloWorld src/main.cpp include/header.h)

target_include_directories(HelloWorld PUBLIC include)
```

src/main.cpp:

```
#include<iostream>
#include<header.h>

int main(int,char*[]){
  std::cout << "header: " << header_function() << std::endl;
  return 0;
}
```

include/header.h:

```
#pragma once

inline int header_function(){
  return 10;
}
```

Compile Definitions

Compile Definitions

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.12.2)

# this command will add application HelloWorld to build list
add_executable(HelloWorld main.cpp)

target_compile_definitions(HelloWorld PUBLIC PARROT=32)
```

src/main.cpp:

```
#include<iostream>

int main(int,char*[]){
  std::cout << "PARROT value: " << PARROT << std::endl;
  return 0;
}
```

Sestavení a instalace knihovny třetí strany

https://github.com/dormon/Vars

```
$ tree
.
|
|-- CMakeLists.txt
|-- main.cpp
|-- Vars      # extern library
    |-- CMakeLists.txt
    |-- README.md
    |-- src
    |   |-- Vars
    |       |-- Fwd.h
    |       |-- Resource.cpp
    |       |-- Resource.h
    |       |-- Vars.cpp
    |       |-- Vars.h
    |       |-- VarsImpl.cpp
    |       |-- VarsImpl.h
    |-- tests
        |-- catch.hpp
        |-- CMakeLists.txt
        |-- tests.cpp
```

```
$ mkdir varsBuild
$ mkdir install
$ # vygenerování makefile
$ # -H složka s CMakeLists.txt
$ # -B složka pro build
$ # -D nastavení parameteru (proměnné)
$ # CMAKE_INSTALL_PREFIX místo pro instalaci knihovny
$ cmake -HVars -BvarsBuild/ -DCMAKE_INSTALL_PREFIX=install/
$ # spuštění překladu a instalace
$ # --build cesta k build složce
$ # --target název cíle, který se bude sestavovat
$ # pokud cíl závisí na jiných, nejprve se sestaví ty
$ cmake --build varsBuild/ --target install
```

Po instalaci vypadá složka s nainstalovanou knihovnou takto:

```
$ tree install/
|-- include
|   |-- Vars
|       |-- Fwd.h
|       |-- Resource.h
|       |-- vars_export.h
|       |-- Vars.h
|-- lib
    |-- cmake
    |   |-- Vars
    |       |-- VarsConfig.cmake
    |       |-- VarsConfigVersion.cmake
    |       |-- VarsTargets.cmake
    |       |-- VarsTargets-noconfig.cmake
    |-- libVars.a
```

Využití nainstalované knihovny

Využití nainstalované knihovny

main.cpp:

```
#include<Vars/Vars.h>

int main(int,char*[]){
  vars::Vars vars;
  vars.addBool("AmIABadPerson?",true);
  return 0;
}
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.13.0)
#search for Vars library
find_package(Vars CONFIG REQUIRED)

add_executable(app main.cpp)
#app needs Vars library for building and linking
#Vars library provides Vars::Vars target
#Vars::Vars target contain every information about
#library (includes, paths to *.so/*.dll *.lib/*.a, defs...
target_link_libraries(app PUBLIC Vars::Vars)
```

```
$ ls
CMakeLists.txt  install  main.cpp  Vars  varsBuild
$ mkdir build
$ # vygenerování makefile pro aplikaci
$ # využívající knihovnu Vars
$ # Vars_DIR je proměnná, která by měla ukazovat na
$ # složku, obsahující soubor VarsConfig.cmake
$ cmake -H. -Bbuild/ -DVars_DIR=install/lib/cmake/Vars
$ # sestavení aplikace
$ cmake --build build/
```

CMake pro knihovny

Co požadujeme od CMake pro knihovny?

- Knihovnu by mělo jít snadno sestavit.
- Knihovna by měla jít nainstalovat.
- Knihovnu by mělo jít sestavit staticy/dynamicky.
- Knihovna by měla propagovat závislosti.
- Knihovna by měla vyřešit obtížné závislosti a nepropagovat problémy dál.
- Knihovna by měla nastavovat správně cesty k *.lib a *.h
- Knihovnu by mělo jít snadno vestavět do většího projektu
- Knihovna by neměla předpokládat, že je top most projekt.
- Knihovna by měla mít správně verze.
- Knihovna by měla mít politiku pro kompatiblitu.
- ...

Druhy knihoven

Druhy knihoven:

- dynamické knihovny, sdílené knihovny *.dll *.so
- statické knihovny, import knihovny *.lib *.a
- header only knihovny *.h
- object knihovny *.obj, *.o
- plugin/module *.dll *.so
- ...

Kde se knihovny hledají, když se pustí aplikace:

- Windows - v místě, kde byla aplikace spuštěna, systémové adresáře, adresáře v proměnné PATH
- Linux - zjištění pomocí ldconfig, soubor /etc/ld.so.conf, RPATH, **ne ve working directory!**

Jak zjistit závislosti knihoven/aplikací:

- Windows - dependency walker
- Linux - **ldd lib.so**, jak zjistit RPATH - **objdump** -x lib.so | **grep** -i "runpath"

Druhy knihoven II

Dělení podle druhu kompilace CMAKE_BUILD_TYPE:

- RELEASE knihovny
- DEBUG knihovny
- RELWITHDEBINFO
- MINSIZEREL

Dělení podle toho, jestli ji buildíme my nebo ji od někud dostaneme

- IMPORTED - externí knihovna
- SHARED - *.dll+*.lib / *.so
- STATIC - *.lib / *.a
- OBJECT - *.obj / *.o
- INTERFACE - *.h
- MODULE - *.dll / *.so
- ALIAS - přejmenování

Statická knihovna

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.13.0)

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_INCLUDE_CURRENT_DIR_IN_INTERFACE ON)

add_library(engine STATIC engine.h engine.cpp)
```

engine.h:

```
#pragma once

int add(int a,int b);
```

engine.cpp:

```
#include<engine.h>

int add(int a,int b){
  return a+b;
}
```

Sdílená knihovna

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.13.0)

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_INCLUDE_CURRENT_DIR_IN_INTERFACE ON)

add_library(engine SHARED engine.h engine.cpp)

#we need to generated *_export.h file that contains __declspec ...
include(GenerateExportHeader)
generate_export_header(engine EXPORT_FILE_NAME engine_export.h)
```

engine.h:

```
#pragma once

#include<engine_export.h>

ENGINE_EXPORT int add(int a,int b);
```

engine.cpp:

```
#include<engine.h>

int add(int a,int b){
  return a+b;
}
```

Header only knihovna

Header only knihovna

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.13.0)

add_library(engine INTERFACE)
```

engine.h:

```
#pragma once

inline int add(int a,int b){
  return a+b;
}
```

```
cmake_minimum_required(VERSION 3.13.0)

project(
  MealyMachine
  VERSION 1.0.2
  )

#set these variables to *.cpp, *.c, ..., *.h, *.hpp, ...
set(SOURCES
  src/${PROJECT_NAME}/MealyMachine.cpp
  )
set(PRIVATE_INCLUDES )
set(PUBLIC_INCLUDES
  src/${PROJECT_NAME}/Fwd.h
  src/${PROJECT_NAME}/MapTransitionChooser.h
  src/${PROJECT_NAME}/MealyMachine.h
  src/${PROJECT_NAME}/TransitionChooser.h
  src/${PROJECT_NAME}/Exception.h
  )
set(INTERFACE_INCLUDES )

#list of libraries to find
#it should contain list of "lists"
#format:
#set(ExternLibraries
#  A\\ B\\ C\\ D
#  E\\ F\\ G
#  )
#It will be converted into:
#find_package(A B C D)
#find_package(E F G)
#If version is specified, it has to be the second parameter (B)
set(ExternPrivateLibraries )
set(ExternPublicLibraries )
set(ExternInterfaceLibraries )

#set these variables to targets
set(PrivateTargets )
set(PublicTargets )
set(InterfaceTargets )

#set these libraries to variables that are provided by libraries that does not support configs
set(PrivateIncludeVariables )
set(PublicIncludeVariables )
set(InterfaceIncludeVariables )
set(PrivateReleaseLibraryVariables )
set(PublicReleaseLibraryVariables )
set(InterfaceReleaseLibraryVariables )
set(PrivateDebugLibraryVariables )
set(PublicDebugLibraryVariables )
set(InterfaceDebugLibraryVariables )

SET(CMAKE_CXX_STANDARD 14)

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_INCLUDE_CURRENT_DIR_IN_INTERFACE ON)

option(${PROJECT_NAME}_BUILD_TESTS "toggle building of unit tests")
if(${PROJECT_NAME}_BUILD_TESTS)
  enable_testing()
  add_subdirectory(tests)
  add_test(NAME baseTest COMMAND tests)
endif()
```

Dobrá knihovna? Část 2.

```
if("${SOURCES}" STREQUAL "")
  set(HeaderOnly TRUE)
else()
  set(HeaderOnly FALSE)
endif()

#find private dependencies
foreach(lib ${ExternPrivateLibraries})
  list(GET lib 0 libName)
  if(NOT TARGET ${libName})
    find_package(${lib})
  endif()
endforeach()

#find public dependencies
foreach(lib ${ExternPublicLibraries})
  list(GET lib 0 libName)
  if(NOT TARGET ${libName})
    find_package(${lib})
  endif()
endforeach()


if(NOT ${HeaderOnly})
  option(BUILD_SHARED_LIBS "build this library as shared")
endif()

SET(CMAKE_DEBUG_POSTFIX          "d"  CACHE STRING "add a postfix, usually d on windows"    )
SET(CMAKE_RELEASE_POSTFIX        ""   CACHE STRING "add a postfix, usually empty on windows")
SET(CMAKE_RELWITHDEBINFO_POSTFIX "rd" CACHE STRING "add a postfix, usually empty on windows")
SET(CMAKE_MINSIZEREL_POSTFIX     "s"  CACHE STRING "add a postfix, usually empty on windows")

if(${HeaderOnly})
  add_library(${PROJECT_NAME} INTERFACE)
else()
  add_library(${PROJECT_NAME} ${SOURCES} ${PRIVATE_INCLUDES} ${PUBLIC_INCLUDES} ${INTERFACE_INCLUDES})
endif()

add_library(${PROJECT_NAME}::${PROJECT_NAME} ALIAS ${PROJECT_NAME})

include(GNUInstallDirs)

if(${HeaderOnly})
  target_include_directories(${PROJECT_NAME} INTERFACE $<INSTALL_INTERFACE:${CMAKE_INSTALL_INCLUDEDIR}>)
  target_include_directories(${PROJECT_NAME} INTERFACE $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/src>)
else()
  target_include_directories(${PROJECT_NAME} PUBLIC $<INSTALL_INTERFACE:${CMAKE_INSTALL_INCLUDEDIR}>)
  target_include_directories(${PROJECT_NAME} PUBLIC $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/src>)
endif()

foreach(inc ${PrivateIncludeVariables})
  target_include_directories(${PROJECT_NAME} PRIVATE ${${inc}})
endforeach()

foreach(inc ${PublicIncludeVariables})
  #this has to be also private because we are exporting this include manually
  target_include_directories(${PROJECT_NAME} PRIVATE ${${inc}})
endforeach()

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_INCLUDE_CURRENT_DIR_IN_INTERFACE ON)

if(${HeaderOnly})
  target_link_libraries(${PROJECT_NAME}
    INTERFACE ${InterfaceTargets}
  )
else()
  target_link_libraries(${PROJECT_NAME}
    PUBLIC    ${PublicTargets}
    PRIVATE   ${PrivateTargets}
    INTERFACE ${InterfaceTargets}
  )
endif()

if("${CMAKE_BUILD_TYPE}" STREQUAL "RELEASE")
  foreach(lib ${PrivateReleaseLibraryVariables})
    target_link_libraries(${PROJECT_NAME} PRIVATE ${${lib}})
  endforeach()
  foreach(lib ${PublicReleaseLibraryVariables})
    #this has to be also PRIVATE, we are exporting manually...
    target_link_libraries(${PROJECT_NAME} PRIVATE ${${lib}})
  endforeach()
else()
    foreach(lib ${PrivateDebugLibraryVariables})
      target_link_libraries(${PROJECT_NAME} PRIVATE ${${lib}})
    endforeach()
    foreach(lib ${PublicDebugLibraryVariables})
      #this has to be also PRIVATE, we are exporting manually...
      target_link_libraries(${PROJECT_NAME} PRIVATE ${${lib}})
    endforeach()
endif()
```

Dobrá knihovna? Část 3.

```
set(PROJECT_NAME_LOWER)
string(TOLOWER ${PROJECT_NAME} PROJECT_NAME_LOWER)

if(NOT ${HeaderOnly})
  include(GenerateExportHeader)
  generate_export_header(${PROJECT_NAME} EXPORT_FILE_NAME ${PROJECT_NAME}/${PROJECT_NAME_LOWER}_export.h)
  set_property(TARGET ${PROJECT_NAME} PROPERTY VERSION ${PROJECT_VERSION})
  set_property(TARGET ${PROJECT_NAME} PROPERTY SOVERSION ${PROJECT_VERSION_MAJOR})
endif()

set_property(TARGET ${PROJECT_NAME} PROPERTY INTERFACE_${PROJECT_NAME}_MAJOR_VERSION ${PROJECT_VERSION_MAJOR})
set_property(TARGET ${PROJECT_NAME} APPEND PROPERTY COMPATIBLE_INTERFACE_STRING ${PROJECT_NAME}_MAJOR_VERSION)

install(TARGETS ${PROJECT_NAME} EXPORT ${PROJECT_NAME}Targets
  LIBRARY  DESTINATION ${CMAKE_INSTALL_LIBDIR}
  ARCHIVE  DESTINATION ${CMAKE_INSTALL_LIBDIR}
  RUNTIME  DESTINATION ${CMAKE_INSTALL_BINDIR}
  INCLUDES DESTINATION ${CMAKE_INSTALL_INCLUDEDIR}
  )

#install header files
if(NOT ${HeaderOnly})
  install(
    FILES       ${CMAKE_CURRENT_BINARY_DIR}/${PROJECT_NAME}/${PROJECT_NAME_LOWER}_export.h
    DESTINATION ${CMAKE_INSTALL_INCLUDEDIR}/${PROJECT_NAME}
    COMPONENT   Devel
    )
endif()

install(
  FILES       ${PUBLIC_INCLUDES} ${INTERFACE_INCLUDES}
  DESTINATION ${CMAKE_INSTALL_INCLUDEDIR}/${PROJECT_NAME}
  COMPONENT   Devel
  )

#create *ConfigVersion.cmake config file
include(CMakePackageConfigHelpers)
set(ConfigDir ${CMAKE_CURRENT_BINARY_DIR}/${PROJECT_NAME})
write_basic_package_version_file(
  ${ConfigDir}/${PROJECT_NAME}ConfigVersion.cmake
  VERSION ${${PROJECT_NAME}_VERSION}
  COMPATIBILITY SameMajorVersion
  )

#create *Targets.cmake config file
export(EXPORT ${PROJECT_NAME}Targets
  FILE      ${ConfigDir}/${PROJECT_NAME}Targets.cmake
  NAMESPACE ${PROJECT_NAME}::
  )

#create content of *Config.cmake config file
string(CONCAT ConfigContent
  "include(CMakeFindDependencyMacro)\n"
  )

foreach(lib ${ExternPublicLibraries} ${ExternInterfaceLibraries})
  list(GET lib 0 libName)
  list(LENGTH lib len)
  if(${len} LESS 2)
    string(CONCAT ConfigContent
      ${ConfigContent}
      "find_dependency(" ${libName} ")\n"
      )
  else()
    list(GET lib 1 libVersion)
    #try to match the version
    string(REGEX MATCH "^[0-9]+(\\.[0-9]+)$" matchedVersion ${libVersion})
    if("${matchedVersion}" STREQUAL "")
      string(CONCAT ConfigContent
        ${ConfigContent}
        "find_dependency(" ${libName} ")\n"
        )
    else()
      string(CONCAT ConfigContent
        ${ConfigContent}
        "find_dependency(" ${libName} " " ${libVersion} ")\n"
        )
    endif()
  endif()
endforeach()


string(CONCAT ConfigContent
  ${ConfigContent}
  "include(${" "${CMAKE_CURRENT_LIST_DIR}/${PROJECT_NAME}Targets.cmake)\n"
  )

#create *Config.cmake config file
file(WRITE ${ConfigDir}/${PROJECT_NAME}Config.cmake ${ConfigContent})

#install configs
set(ConfigPackageLocation lib/cmake/${PROJECT_NAME})
install(
  FILES
  ${ConfigDir}/${PROJECT_NAME}Config.cmake
  ${ConfigDir}/${PROJECT_NAME}ConfigVersion.cmake
  DESTINATION ${ConfigPackageLocation}
  COMPONENT   Devel
  )

install(EXPORT ${PROJECT_NAME}Targets
  FILE        ${PROJECT_NAME}Targets.cmake
  NAMESPACE   ${PROJECT_NAME}::
  DESTINATION ${ConfigPackageLocation}
  )
```

Dobrá knihovna? Část 4.

```
if((NOT "${PublicIncludeVariables}" STREQUAL "") OR (NOT "${InterfaceIncludeVariables}" STREQUAL ""))

  string(CONCAT FIX
    "\n"
    "# We need to fix libraries that do not provide configs but variables (like GLEW_INCLUDE_DIR GLEW_LIBRARY_RELEASE)\n"
    "# We do it by adding these variables to INTERFACE_INCLUDE_DIRECTORIES and INTERFACE_LINK_LIBRARIES\n"
    "get_target_property(includes " ${PROJECT_NAME} "::" ${PROJECT_NAME} " INTERFACE_INCLUDE_DIRECTORIES)\n"
    "get_target_property(libs " ${PROJECT_NAME} "::" ${PROJECT_NAME} " INTERFACE_LINK_LIBRARIES)\n"
    )


  foreach(inc ${PublicIncludeVariables})
    string(CONCAT FIX "${FIX}"
      "string(CONCAT includes \\\"\\$" "{includes}\\\" \\\"\;\\$" "{${inc}}\\\")\n"

      )
  endforeach()

  foreach(inc ${InterfaceIncludeVariables})
    string(CONCAT FIX "${FIX}"
      "string(CONCAT includes \\\"\\$" "{includes}\\\" \\\"\;\\$" "{${inc}}\\\")\n"

      )
  endforeach()

  foreach(lib ${PublicReleaseLibraryVariables})
    string(CONCAT FIX "${FIX}"
      "string(CONCAT libs \\\"\\$" "{libs}\\\" \\\"\\$<\\$<CONFIG:Release>:\;\\$" "{${lib}}>\\\")\n"

      )
  endforeach()

  foreach(lib ${InterfaceReleaseLibraryVariables})
    string(CONCAT FIX "${FIX}"
      "string(CONCAT libs \\\"\\$" "{libs}\\\" \\\"\\$<\\$<CONFIG:Release>:\;\\$" "{${lib}}>\\\")\n"

      )
  endforeach()

  foreach(lib ${PublicDebugLibraryVariables})
    string(CONCAT FIX "${FIX}"
      "string(CONCAT libs \\\"\\$" "{libs}\\\" \\\"\\$<\\$<CONFIG:Debug>:\;\\$" "{${lib}}>\\\")\n"

      )
  endforeach()

  foreach(lib ${InterfaceDebugLibraryVariables})
    string(CONCAT FIX "${FIX}"
      "string(CONCAT libs \\\"\\$" "{libs}\\\" \\\"\\$<\\$<CONFIG:Debug>:\;\\$" "{${lib}}>\\\")\n"

      )
  endforeach()

  string(CONCAT FIX "${FIX}"
    "set_target_properties(" ${PROJECT_NAME} "::" ${PROJECT_NAME} " PROPERTIES INTERFACE_INCLUDE_DIRECTORIES \\\"\\$" "{includes}\\\" INTERFACE_LINK_LIBRARIES \\\"\\$" "{libs}\\\")\n"

    )

  string(CONCAT installCode
    "file(APPEND ${CMAKE_INSTALL_PREFIX}/${ConfigPackageLocation}/${PROJECT_NAME}Targets.cmake \"${FIX}\")\n"
    )

  install(CODE ${installCode})

endif()
```

Jak řešit závislosti, které mají špatný nebo žádný cmake?

Špatné knihovny:

- špatné knihovny poskytují pouze proměnné nikoliv TARGET
- špatné knihovny neposkytují instaslaci
- špatné knihovny mají špatný install script
- špatné knihovny nemají cmake

Jak řešit?

- Dodat autorům opravený cmake
- Napsat / najít Find*.cmake script, který vytvoří target za knihovnu
- Distribuovat Find*.cmake script pro závislosti se svojí knihovnou
- Vytvořit si ručně IMPORTED TARGET
- ...

Překopírování DLL do buildu

```
function(getSharedLibraries out lib cfg)
  #message("getSharedLibraries(${lib} ${cfg})")

  if(TARGET ${lib})
    get_target_property(target_type ${lib} TYPE)
    if(${target_type} STREQUAL "INTERFACE_LIBRARY")
      #message("${lib} JE ${target_type}")
      return()
    endif()
  else()
    #message("${lib} NENI TARGET")
    return()
  endif()


  get_target_property(dll ${lib} IMPORTED_LOCATION_${cfg})
  list(APPEND dlls ${dll})

  get_target_property(interfaceLibs ${lib} INTERFACE_LINK_LIBRARIES)
  if(NOT "${interfaceLibs}" STREQUAL "interfaceLibs-NOTFOUND")
    foreach(interfaceLib ${interfaceLibs})
      getSharedLibraries(ilibs ${interfaceLib} ${cfg})
      list(APPEND dlls ${ilibs})
    endforeach()
  endif()
  list(REMOVE_DUPLICATES dlls)
  set(${out} ${dlls} PARENT_SCOPE)
endfunction()


function(getAllSharedLibraries allLibraries app cfg)
  get_target_property(libs ${app} LINK_LIBRARIES)
  foreach(lib ${libs})
    getSharedLibraries(libList ${lib} ${cfg})
    #message("${lib} ###### ${libList}")
    list(APPEND allLibs ${libList})
  endforeach()
  list(REMOVE_DUPLICATES allLibs)
  set(${allLibraries} ${allLibs} PARENT_SCOPE)
endfunction()

getAllSharedLibraries(allDebugSharedLibraries ${PROJECT_NAME} DEBUG)
getAllSharedLibraries(allReleaseSharedLibraries ${PROJECT_NAME} RELEASE)

#message("SharedDebug  : ${allDebugSharedLibraries}")
#message("SharedRelease: ${allReleaseSharedLibraries}")

if(MSVC)
  foreach(lib ${allDebugSharedLibraries})
    file(COPY ${lib} DESTINATION ${CMAKE_CURRENT_BINARY_DIR}/Debug)
  endforeach()
  foreach(lib ${allReleaseSharedLibraries})
    file(COPY ${lib} DESTINATION ${CMAKE_CURRENT_BINARY_DIR}/Release)
  endforeach()
endif()

install(TARGETS ${PROJECT_NAME} RUNTIME DESTINATION .)
install(FILES ${allDebugSharedLibraries} DESTINATION .)
```

Stažení knihovny třetí strany

```
cmake_minimum_required(VERSION 3.13.0)

include(FetchContent)

FetchContent_Declare(MealyMachine     GIT_REPOSITORY https://github.com/dormon/MealyMachine.git    )
FetchContent_Declare(TxtUtils         GIT_REPOSITORY https://github.com/dormon/TxtUtils.git        )
FetchContent_Declare(ArgumentViewer   GIT_REPOSITORY https://github.com/dormon/ArgumentViewer.git )

FetchContent_GetProperties(MealyMachine)
if(NOT MealyMachine_POPULATED)
  FetchContent_Populate(MealyMachine)
  add_subdirectory(${mealymachine_SOURCE_DIR} ${mealymachine_BINARY_DIR})
endif()

FetchContent_GetProperties(TxtUtils)
if(NOT TxtUtils_POPULATED)
  FetchContent_Populate(TxtUtils)
  add_subdirectory(${txtutils_SOURCE_DIR} ${txtutils_BINARY_DIR})
endif()

FetchContent_GetProperties(ArgumentViewer)
if(NOT ArgumentViewer_POPULATED)
  FetchContent_Populate(ArgumentViewer)
  add_subdirectory(${argumentviewer_SOURCE_DIR} ${argumentviewer_BINARY_DIR})
endif()

add_executable(as main.cpp)
target_link_libraries(as ArgumentViewer::ArgumentViewer)
```

Thank you for your attention! Questions?