

CMAKE MODERNE

-•~ *ET* ~--

LA PLACE DES PACKAGE MANAGER

GUILLAUME RACICOT



- Druide Informatique
 - Antidote Web
-
- Auteur et contributeur à des librairies C++
 - Engin de jeu maison et auteur de subgine-pkg

PRESENTATIONLISTS.TXT

- CMake: Quoi et Pourquoi
- CMake Moderne en Survol
- Dépendances Externes
- La Place des Package Managers
- Des outils pour la gestion des paquets + preuve de concept

C'EST QUOI DÉJÀ CMAKE?

- Pas un build system
- Un meta build system!

→ Generateur de build system

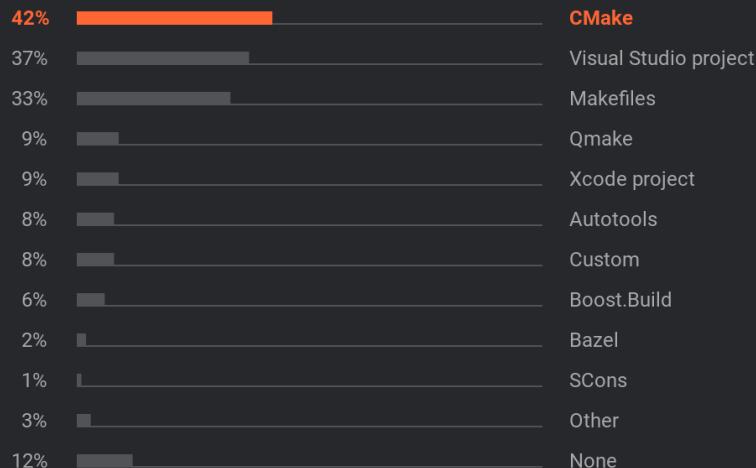


Logo de CMake

- Peut générer des makefiles, solution visual studio, projet xcode, ninja, nmake et autres!

RECONNU PAR L'INDUSTRIE

Which project models or build systems do you regularly use?



Last year CMake beat Visual Studio project to become the most popular project model / build system used for C++ development.

Its share has since added 5 percentage points and reached 42%.

C++ 2019 - The state of Developer Ecosystem in 2019 Infographic
<https://www.jetbrains.com/lp/devecosystem-2019/cpp>

LES IDE QUI SUPPORTENT CMAKE

- Visual Studio (≥ 2017)
- QtCreator
- KDevelop
- CLion
- VSCode (extensions)
- ... Surement d'autres!

CMAKE MODERNE

EN SURVOL

CMAKE MODERNE

- Configuration de projet basé sur les *target* ou *cible*
- Les propriétés des cible détermine comment elle sera compilée
- Les liens entre les cibles affecteront les propriétés afin de rendre la relation possible

EXEMPLE DE CIBLE

```
# Compilé en executable
add_executable(target1 a.cpp b.cpp)

# Compilé en librairie statique ou dynamique
# selon les argument passé a CMake
add_library(target2 a.cpp b.cpp)

# Compilé en librairie statique ou dynamique (fixe)
add_library(target3 <STATIC|SHARED> a.cpp b.cpp)

# Librairie purement logique. Représente seulement
# une gamme de prérequis afin d'être utilisée
add_library(target4 INTERFACE)
```

EXEMPLE DE CIBLE

```
# Pas de librairie finale sert à compiler
# les fichiers et "link" dans d'autre target
add_library(target5 OBJECT a.cpp b.cpp)

# Librairie dynamique qui ne sera jamais liée à d'autre cible
# conçu pour représenter un plugin qui sera ouvert dynamiquement
add_library(target6 MODULE a.cpp b.cpp)

# Cible dont la façon de compilé et les résultat sont personnalisé
add_custom_target(target7 COMMAND ... BYPRODUCTS ... DEPENDS ...)

# Donne un nouveau nom à une cible
add_library(target8 ALIAS target7)
```

COMMENT LIER DES CIBLE ENSEMBLE?

REPRÉSENTER DES LIENS

```
src/
└── CMakeLists.txt
exec/
└── CMakeLists.txt
    └── main.cpp

allo/
└── CMakeLists.txt
    ├── allo.cpp
    └── include/
        └── allo.h
```

REPRÉSENTER DES LIENS

```
src/
└── CMakeLists.txt
exec/
└── CMakeLists.txt
    └── main.cpp

allo/
└── CMakeLists.txt
    └── allo.cpp
        └── include/
            └── allo.h
```

```
add_subdirectory(exec)
add_subdirectory(allo)
```

REPRÉSENTER DES LIENS

```
src/
└── CMakeLists.txt
exec/
└── CMakeLists.txt
    └── main.cpp
allo/
└── CMakeLists.txt
    ├── allo.cpp
    └── include/
        └── allo.h
```

```
add_subdirectory(exec)
add_subdirectory(allo)
```

```
allo.h
```

```
auto allo_message() -> char const*
```

```
allo.cpp
```

```
#include "allo.h"
```

```
auto allo_message() -> char const* {
    return "Allo CMake!";
}
```

REPRÉSENTER DES LIENS

```
src/
└── CMakeLists.txt
exec/
└── CMakeLists.txt
    └── main.cpp
allo/
└── CMakeLists.txt
    ├── allo.cpp
    └── include/
        └── allo.h
```

```
add_subdirectory(exec)
add_subdirectory(allo)
```

```
allo.h
```

```
auto allo_message() -> char const*
```

```
allo.cpp
```

```
#include "allo.h"
```

```
auto allo_message() -> char const* {
    return "Allo CMake!";
}
```

```
main.cpp
```

```
#include "allo.h"
```

```
auto main() -> int {
    puts(allo_message());
}
```

REPRÉSENTER DES LIENS

```
src/
└── CMakeLists.txt
exec/
└── CMakeLists.txt
    └── main.cpp
allo/
└── CMakeLists.txt
    ├── allo.cpp
    └── include/
        └── allo.h
```

```
add_subdirectory(exec)
add_subdirectory(allo)
```

```
allo.h
```

```
auto allo_message() -> char const*
```

```
allo.cpp
```

```
#include "allo.h"
```

```
auto allo_message() -> char const* {
    return "Allo CMake!";
}
```

```
main.cpp
```

```
#include "allo.h"
    ^---- comment trouver allo.h?
auto main() -> int {
    puts(allo_message());
}
```

REPRÉSENTER DES LIENS

ANCIENNE MÉTHODE: PROPRIÉTÉS SUR LES RÉPERTOIRES

```
allo/CMakeLists.txt
```

```
add_library(allo allo.cpp)
include_directories("include")
```

```
exec/CMakeLists.txt
```

```
add_executable(exec main.cpp)
include_directories("../allo/include")
target_link_libraries(exec allo)
```

REPRÉSENTER DES LIENS

ANCIENNE MÉTHODE: PROPRIÉTÉS SUR LES RÉPERTOIRES

```
allo/CMakeLists.txt
```

```
add_library(allo allo.cpp)
include_directories("include")
```

```
exec/CMakeLists.txt
```

```
add_executable(exec main.cpp)
include_directories("../allo/include")
target_link_libraries(exec allo)
```

NOUVELLE MÉTHODE: PROPRIÉTÉS SUR LES CIBLES

```
allo/CMakeLists.txt
```

```
add_library(allo allo.cpp)
target_include_directories(allo PRIVATE "include")
target_include_directories(allo INTERFACE "include")
```

```
exec/CMakeLists.txt
```

```
add_executable(exec main.cpp)
target_link_libraries(exec PRIVATE allo)
```

REPRÉSENTER DES LIENS

ANCIENNE MÉTHODE: PROPRIÉTÉS SUR LES RÉPERTOIRES

```
allo/CMakeLists.txt
```

```
add_library(allo allo.cpp)
include_directories("include")
```

```
exec/CMakeLists.txt
```

```
add_executable(exec main.cpp)
include_directories("../allo/include")
target_link_libraries(exec allo)
```

NOUVELLE MÉTHODE: PROPRIÉTÉS SUR LES CIBLES

```
allo/CMakeLists.txt
```

```
add_library(allo allo.cpp)
target_include_directories(allo PUBLIC "include")
```

```
exec/CMakeLists.txt
```

```
add_executable(exec main.cpp)
target_link_libraries(exec PRIVATE allo)
```

REPRÉSENTER DES LIENS

Commandes sur les cibles

- target_link_libraries
- target_compile_definitions
- target_compile_options
- target_compile_features
- target_include_directories
- target_link_options

REPRÉSENTER DES LIENS

Autres propriétés

(avec `set_target_properties`)

- `POSITION_INDEPENDENT_CODE`
- `PUBLIC_HEADER`
- `RUNTIME_OUTPUT_DIRECTORY`
- `RUNTIME_OUTPUT_NAME`
- `STATIC_LIBRARY_OPTIONS`
- ... Et beaucoup d'autres!

Documentation: [CMake Properties](#)

REPRÉSENTER DES LIENS

Autres propriétés

(avec `set_target_properties`)

- `POSITION_INDEPENDENT_CODE`
- `PUBLIC_HEADER`
- `RUNTIME_OUTPUT_DIRECTORY`
- `RUNTIME_OUTPUT_NAME`
- `STATIC_LIBRARY_OPTIONS`
- ... Et beaucoup d'autres!

Documentation: [CMake Properties](#)

```
set_target_properties(cible PROPERTIES
    PROPERTY1 value1
    PROPERTY2 value2
    ...
)
```

REPRÉSENTER DES LIENS

Commandes sur les repertoires

- add_compile_options
- include_directories
- link_directories
- link_libraries



DÉPENDANCES EXTERNES

DÉPENDANCES EXTERNES

- Mettre les bons include directories
- Lier les cibles au bonnes librairies
- Mettre les bonnes définitions au compilateur
- Utiliser le bon standard...
- ... d'autre ...

DÉPENDANCES EXTERNES

- Utiliser une cible importée!
- Déclarer une cible importée et définire ses propriétés d'interface

DÉPENDANCES EXTERNES

- Utiliser une cible importée!
- Déclarer une cible importée et définire ses propriétés d'interface

```
add_library(cpplocate STATIC IMPORTED)
set_target_properties(cpplocate PROPERTIES
    IMPORTED_LOCATION /path/vers/le/binaire.a
    INTERFACE_INCLUDE_DIRECTORIES /path/vers/les/entetes/
    INTERFACE_COMPILE_DEFINITIONS MACRO1 MACRO2=valeur
    ...
)
```

DÉPENDANCES EXTERNES

- Utiliser une cible importée!
- Déclarer une cible importée et définire ses propriétés d'interface

```
add_library(cpplocate STATIC IMPORTED)
set_target_properties(cpplocate PROPERTIES
    IMPORTED_LOCATION /path/vers/le/binaire.a
    INTERFACE_INCLUDE_DIRECTORIES /path/vers/les/entetes/
    INTERFACE_COMPILE_DEFINITIONS MACRO1 MACRO2=valeur
    ...
)

target_link_libraries(cible PUBLIC cpplocate)
```

DÉPENDANCES EXTERNES

- Utiliser une cible importée!
- Déclarer une cible importée et définire ses propriétés d'interface

```
add_library(cpplocate STATIC IMPORTED)
set_target_properties(cpplocate PROPERTIES
    IMPORTED_LOCATION /path/vers/le/binaire.a
    INTERFACE_INCLUDE_DIRECTORIES /path/vers/les/entetes/
    INTERFACE_COMPILE_DEFINITIONS MACRO1 MACRO2=valeur
    ...
)

target_link_libraries(cible PUBLIC cpplocate)
```

???

DÉPENDANCES EXTERNES

`find_package`: Trouver un projet externe

DÉPENDANCES EXTERNES

`find_package`: Trouver un projet externe

```
find_package (cpplocate REQUIRED)
```

```
target_link_libraries (mon-executable PUBLIC cpplocate::cpplocate)
```

DÉPENDANCES EXTERNES

Peut trouver des paquets...

- ... installé sur le poste
- ... installé dans un répertoire local
- ... installé dans un sous dossier du projet
- ... pas installé, seulement compilé

COMMENT `find_package` TROUVE LES PROJETS EXTERNES?

Utilise un prefix path

- `/usr/`
- `/usr/local/`
- `C:\Program Files\`

```
-DCMAKE_PREFIX_PATH=/mon/prefix/
```

```
list(APPEND CMAKE_PREFIX_PATH "un/autre/prefix")
```

Recherche `<nom-du-paquet>-config.cmake`

COMMENT `find_package` TROUVE LES PROJETS EXTERNES?

- `<prefix>/<name>*/(lib/<arch>| lib* | share)/cmake/<name>*/`
- `<prefix>/<name>*/(lib/<arch>| lib* | share)/<name>*/`
- `<prefix>/<name>*/(lib/<arch>| lib* | share)/<name>*/(cmake | CMake)/`

```
/mon/prefix/glm/lib/share/cmake/glm/glm-config.cmake
```

```
/mon/prefix/glm/share/cmake/glm/glm-config.cmake
```

```
/mon/prefix/glm/share/glm/cmake/glm-config.cmake
```

```
/mon/prefix/glm/lib/glm/glm-config.cmake
```

COMMENT `find_package` TROUVE LES PROJETS EXTERNES?

Chemins spécifique a unix

- `<prefix>/(lib/<arch>|lib*|share)/cmake/<name>*/`
- `<prefix>/(lib/<arch>|lib*|share)/<name>*/`
- `<prefix>/(lib/<arch>|lib*|share)/<name>*/(cmake|CMake)/`

```
/mon/prefix/lib/share/cmake/glm/glm-config.cmake
```

```
/mon/prefix/share/cmake/glm/glm-config.cmake
```

```
/mon/prefix/share/glm/cmake/glm-config.cmake
```

```
/mon/prefix/lib/glm/glm-config.cmake
```

COMMENT `find_package` TROUVE LES PROJETS EXTERNES?

Chemins spécifique a Windows

- <prefix>/
- <prefix>/(cmake|CMake)/
- <prefix>/<name>*/
- <prefix>/<name>*/(cmake|CMake)/

```
/mon/prefix/glm-config.cmake
```

```
/mon/prefix/cmake/glm-config.cmake
```

```
/mon/prefix/glm/glm-config.cmake
```

```
/mon/prefix/glm/cmake/glm-config.cmake
```

COMMENT `find_package` TROUVE LES PROJETS EXTERNES?

Spécifier des versions

COMMENT `find_package` TROUVE LES PROJETS EXTERNES?

Spécifier des versions

```
find_package(nlohmann_json 2.1)
```

COMMENT `find_package` TROUVE LES PROJETS EXTERNES?

Spécifier des versions

```
find_package(nlohmann_json 2.1)
```

```
find_package(nlohmann_json 3.5)
```

COMMENT `find_package` TROUVE LES PROJETS EXTERNES?

Spécifier des versions

```
find_package(nlohmann_json 2.1)
```

```
find_package(nlohmann_json 3.5)
```

```
find_package(nlohmann_json 3.7 EXACT)
```

COMMENT `find_package` TROUVE LES PROJETS EXTERNES?

Spécifier des versions

```
find_package(nlohmann_json 2.1)
```

```
find_package(nlohmann_json 3.5)
```

```
find_package(nlohmann_json 3.7 EXACT)
```

La notion de compatibilité est déterminée par le paquet même

COMMENT `find_package` TROUVE LES PROJETS EXTERNES?

Spécifier des versions

```
find_package(nlohmann_json 2.1)
```

```
find_package(nlohmann_json 3.5)
```

```
find_package(nlohmann_json 3.7 EXACT)
```

La notion de compatibilité est déterminée par le paquet même
`<paquet>-config-version.cmake`

Une myriade d'options afin d'ajuster le processus de trouver les paquets!

https://cmake.org/cmake/help/latest/command/find_package.html

Avec un prefix path bien configuré, cmake peut trouver
d'autres projets cmake, peu importe leur
emplacement

C'est tout?

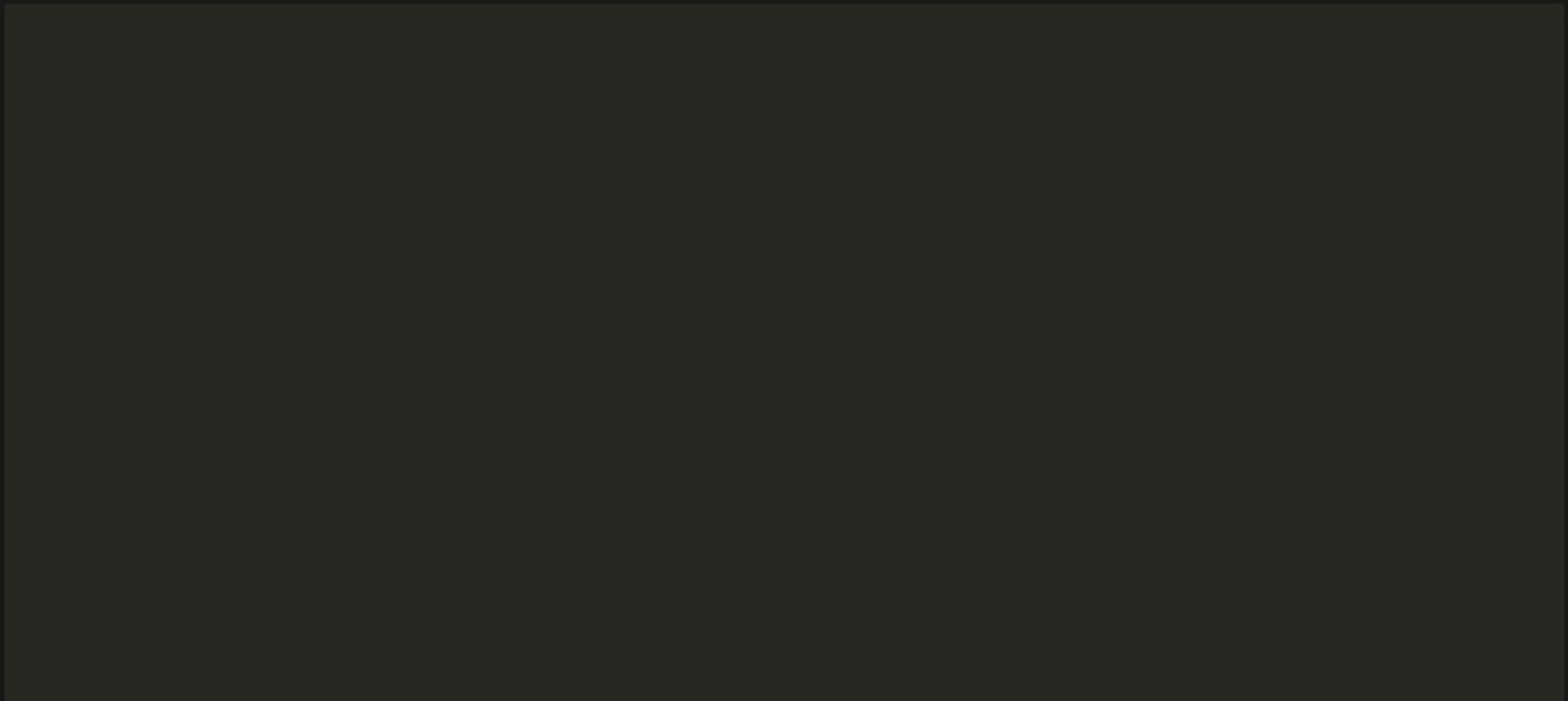
C'est tout? Non...

D'où vient ce xyz-config.cmake??
Comment générer ce fichier?

findXYZ.cmake

It's Time To Do CMake Right - If you want it done right do it yourself

DÉPENDANCES EXTERNES



DÉPENDANCES EXTERNES

```
cmake_minimum_required(VERSION 3.15)
project(projetA VERSION 12.03.1 CXX)
```

DÉPENDANCES EXTERNES

```
cmake_minimum_required(VERSION 3.15)
project(projetA VERSION 12.03.1 CXX)

add_library(lib-a a.cpp b.cpp c.cpp)
add_library(lib-b d.cpp e.cpp f.cpp)
```

DÉPENDANCES EXTERNES

```
cmake_minimum_required(VERSION 3.15)
project(projetA VERSION 12.03.1 CXX)

add_library(lib-a a.cpp b.cpp c.cpp)
add_library(lib-b d.cpp e.cpp f.cpp)

find_package(dep1 1.2.0 REQUIRED)
find_package(dep2 3.4 EXACT REQUIRED)
```

DÉPENDANCES EXTERNES

```
cmake_minimum_required(VERSION 3.15)
project(projetA VERSION 12.03.1 CXX)

add_library(lib-a a.cpp b.cpp c.cpp)
add_library(lib-b d.cpp e.cpp f.cpp)

find_package(dep1 1.2.0 REQUIRED)
find_package(dep2 3.4 EXACT REQUIRED)

target_link_libraries(lib-a PRIVATE dep1::dep1)
target_link_libraries(lib-b PUBLIC dep2::dep2 lib-a)
```

DÉPENDANCES EXTERNES

```
cmake_minimum_required(VERSION 3.15)
project(projetA VERSION 12.03.1 CXX)

add_library(lib-a a.cpp b.cpp c.cpp)
add_library(lib-b d.cpp e.cpp f.cpp)

find_package(dep1 1.2.0 REQUIRED)
find_package(dep2 3.4 EXACT REQUIRED)

target_link_libraries(lib-a PRIVATE dep1::dep1)
target_link_libraries(lib-b PUBLIC dep2::dep2 lib-a)

# logique spécifique au build, générateurs, propriétés
```

DÉPENDANCES EXTERNES

```
cmake_minimum_required(VERSION 3.15)
project(projetA VERSION 12.03.1 CXX)

add_library(lib-a a.cpp b.cpp c.cpp)
add_library(lib-b d.cpp e.cpp f.cpp)

find_package(dep1 1.2.0 REQUIRED)
find_package(dep2 3.4 EXACT REQUIRED)

target_link_libraries(lib-a PRIVATE dep1::dep1)
target_link_libraries(lib-b PUBLIC dep2::dep2 lib-a)

# logique spécifique au build, générateurs, propriétés
# exporter ses cibles comme une bonne personne...
```

DÉPENDANCES EXTERNES

```
cmake_minimum_required(VERSION 3.15)
project(projetA VERSION 12.03.1 CXX)

add_library(lib-a a.cpp b.cpp c.cpp)
add_library(lib-b d.cpp e.cpp f.cpp)

find_package(dep1 1.2.0 REQUIRED)
find_package(dep2 3.4 EXACT REQUIRED)

target_link_libraries(lib-a PRIVATE dep1::dep1)
target_link_libraries(lib-b PUBLIC dep2::dep2 lib-a)

# logique spécifique au build, générateurs, propriétés
# exporter ses cibles comme une bonne personne...
```

Rien d'autre?

DÉPENDANCES EXTERNES

Comment s'assurer que `find_package` trouve tout au bons endroits?

DÉPENDANCES EXTERNES

Comment s'assurer que `find_package` trouve tout au bons endroits?

```
package-info.cmake
```

```
set(BOOST_ROOT /usr/local/boost_1_68/)  
set(BOOST_INCLUDEDIR /usr/local/boost_1_68/include)  
set(Boost_NO_SYSTEM_PATHS ON)  
  
list(APPEND CMAKE_PREFIX_PATH "${PROJECT_SOURCE_DIR}/external")  
list(APPEND CMAKE_MODULE_PATH "${PROJECT_SOURCE_DIR}/cmake_modules")  
  
# ...
```

DÉPENDANCES EXTERNES

Comment s'assurer que `find_package` trouve tout au bons endroits?

```
package-info.cmake
```

```
set(BOOST_ROOT /usr/local/boost_1_68/)  
set(BOOST_INCLUDEDIR /usr/local/boost_1_68/include)  
set(Boost_NO_SYSTEM_PATHS ON)  
  
list(APPEND CMAKE_PREFIX_PATH "${PROJECT_SOURCE_DIR}/external")  
list(APPEND CMAKE_MODULE_PATH "${PROJECT_SOURCE_DIR}/cmake_modules")
```

```
# ...
```

`-DCMAKE_PROJECT_INCLUDE=package-info.cmake`

DÉPENDANCES EXTERNES

Comment s'assurer que `find_package` trouve tout au bons endroits?

```
package-info.cmake
```

```
set(BOOST_ROOT /usr/local/boost_1_68/)  
set(BOOST_INCLUDEDIR /usr/local/boost_1_68/include)  
set(Boost_NO_SYSTEM_PATHS ON)  
  
list(APPEND CMAKE_PREFIX_PATH "${PROJECT_SOURCE_DIR}/external")  
list(APPEND CMAKE_MODULE_PATH "${PROJECT_SOURCE_DIR}/cmake_modules")  
  
# ...
```

- DCMAKE_PROJECT_INCLUDE=package-info.cmake
- DCMAKE_PROJECT_<projet>_INCLUDE=package-info.cmake

DÉPENDANCES EXTERNES

```
cmake_minimum_required(VERSION 3.15)
project(projetA VERSION 12.03.1 CXX) # include(package-info.cmake)

add_library(lib-a a.cpp b.cpp c.cpp)
add_library(lib-b d.cpp e.cpp f.cpp)

find_package(dep1 1.2.0 REQUIRED)
find_package(dep2 3.4 EXACT REQUIRED)
```

DÉPENDANCES EXTERNES

```
cmake_minimum_required(VERSION 3.15)
project(projetA VERSION 12.03.1 CXX) # include(package-info.cmake)

add_library(lib-a a.cpp b.cpp c.cpp)
add_library(lib-b d.cpp e.cpp f.cpp)

find_package(dep1 1.2.0 REQUIRED)
find_package(dep2 3.4 EXACT REQUIRED)
```

Différentes configurations peuvent être placée dans
différent fichiers

LA PLACE DES PACKAGE MANAGERS

LA PLACE DES PACKAGE MANAGERS

- Télécharger et compiler les paquets
- Générer le fichier contenant la configuration

LA PLACE DES PACKAGE MANAGERS

- Pourrait utiliser les librairies système si les versions concordent
- Une option afin de partager des installations entre plusieurs projets
- Trouver aussi des librairies non installées
- Devrait trouver le problème du diamant

Tout comme un conteneur d'injection de dépendances va instancier les dépendances de vos classes et automatiser le câblage entre ceux-ci...

Tout comme un conteneur d'injection de dépendances va instancier les dépendances de vos classes et automatiser le câblage entre ceux-ci...

...un gestionnaire de paquet va télécharger et compiler les dépendances de vos projet et automatiser le câblage entre ceux-ci.

DES OUTILS POUR LA GESTION DES PAQUETS

+

PREUVE DE CONCEPT
(SUBGINE-PKG)

UN PEU D'HISTOIRE...

- Un nouveau projet
- 4 personnes +/- débutant en C++
- Moi développant le framework + le projet
- Coéquipiers surtout sur le projet

UN PEU D'HISTOIRE...

- Simple de mon côté (git push)
- Simple de leur côté (subgine-pkg update)
- Mettre à jour le framework, le compiler et l'installer
- Me permettre d'avoir les deux projets côté à côté

VÉRIFIER SI UN PAQUET EXISTE

```
cmake_minimum_required(VERSION 3.15)
project(testfind)
find_package(<nom-du-paquet> VERSION <version-voulue> QUIET)
```

VÉRIFIER SI UN PAQUET EXISTE

```
cmake_minimum_required(VERSION 3.15)
project(testfind)
find_package(<nom-du-paquet> VERSION <version-voulue> QUIET)

if(<nom-du-paquet>_FOUND)

else()

endif()
```

VÉRIFIER SI UN PAQUET EXISTE

```
cmake_minimum_required(VERSION 3.15)
project(testfind)
find_package(<nom-du-paquet> VERSION <version-voulue> QUIET)

if(<nom-du-paquet>_FOUND)
    message("${<nom-du-paquet>_DIR}")

else()

endif()
```

VÉRIFIER SI UN PAQUET EXISTE

```
cmake_minimum_required(VERSION 3.15)
project(testfind)
find_package(<nom-du-paquet> VERSION <version-voulue> QUIET)

if(<nom-du-paquet>_FOUND)
    message("${<nom-du-paquet>_DIR}")
    message("😊")
else()

endif()
```

VÉRIFIER SI UN PAQUET EXISTE

```
cmake_minimum_required(VERSION 3.15)
project(testfind)
find_package(<nom-du-paquet> VERSION <version-voulue> QUIET)

if(<nom-du-paquet>_FOUND)
    message("${<nom-du-paquet>_DIR}")
    message(":)" )
else()
    message(${<nom-du-paquet>_CONSIDERED_CONFIGS} )

endif()
```

VÉRIFIER SI UN PAQUET EXISTE

```
cmake_minimum_required(VERSION 3.15)
project(testfind)
find_package(<nom-du-paquet> VERSION <version-voulue> QUIET)

if(<nom-du-paquet>_FOUND)
    message("${<nom-du-paquet>_DIR}")
    message("😊")
else()
    message(${<nom-du-paquet>_CONSIDERED_CONFIGS})
    message(${<nom-du-paquet>_CONSIDERED_VERSIONS})

endif()
```

VÉRIFIER SI UN PAQUET EXISTE

```
cmake_minimum_required(VERSION 3.15)
project(testfind)
find_package(<nom-du-paquet> VERSION <version-voulue> QUIET)

if(<nom-du-paquet>_FOUND)
    message("${<nom-du-paquet>_DIR}")
    message(":)" )
else()
    message(${<nom-du-paquet>_CONSIDERED_CONFIGS} )
    message(${<nom-du-paquet>_CONSIDERED_VERSIONS} )
    message(FATAL_ERROR ":(" )
endif()
```

VÉRIFIER SI UN PAQUET EXISTE

```
cmake_minimum_required(VERSION 3.15)
project(testfind)
find_package(<nom-du-paquet> VERSION <version-voulue> QUIET)

if(<nom-du-paquet>_FOUND)
    message("${<nom-du-paquet>_DIR}")
    message(":)" )
else()
    message(${<nom-du-paquet>_CONSIDERED_CONFIGS})
    message(${<nom-du-paquet>_CONSIDERED_VERSIONS})
    message(FATAL_ERROR ":(" )
endif()
```

```
cmake . -DCMAKE_PREFIX_PATH="..."
```

VÉRIFIER SI UN PAQUET EXISTE

```
file(WRITE "CMakeLists.txt" "
cmake_minimum_required(VERSION 3.15)
project(testfind)
find_package(<nom-du-paquet> VERSION <version-voulue> QUIET)

if(<nom-du-paquet>_FOUND)
    message("${<nom-du-paquet>_DIR}")
    message("😊")
else()
    message(${<nom-du-paquet>_CONSIDERED_CONFIGS})
    message(${<nom-du-paquet>_CONSIDERED_VERSIONS})
    message(FATAL_ERROR "😢")
endif()
")
```

```
execute_process(
COMMAND
    cmake . -DCMAKE_PREFIX_PATH="..."
RESULT_VARIABLE
    trouvé-avec-succès
)
```

INSTALLATION LOCALE DE PAQUETS

- DCMAKE_INSTALL_PREFIX=". . ."

INSTALLATION LOCALE DE PAQUETS

```
-DCMAKE_INSTALL_PREFIX="..."  
list(APPEND CMAKE_PREFIX_PATH "...")
```

```
execute_process (
COMMAND
    ${CMAKE_COMMAND} .. -DCMAKE_INSTALL_PREFIX=${modules-path}
WORKING_DIRECTORY
    "./${dependency-name}/${build-directory-name}"
)
```

```
execute_process (
COMMAND
    ${CMAKE_COMMAND} --build . --target install
WORKING_DIRECTORY
    "./${dependency-name}/${build-directory-name}"
)
```

```
# Sauvegarder un artefact confirmant l'installation
```

UTILISATION DE PROFILS DIFFÉRENTS

```
arguments-debug.cmake
```

```
set(build-args "-DCMAKE_BUILD_TYPE=Debug")
```

```
profile-debug.cmake
```

```
list(APPEND CMAKE_PREFIX_PATH "prefix/path/debug/")
```

UTILISATION DE PROFILS DIFFÉRENTS

```
arguments-release.cmake
```

```
set(build-args "-DCMAKE_BUILD_TYPE=Release")
```

```
profile-release.cmake
```

```
list(APPEND CMAKE_PREFIX_PATH "prefix/path/release/")
```

DEMO

<https://github.com/gracicot/subgine-pkg>

EN RÉTROSPECTIVE

- CMake implémente déjà beaucoup!
- Avoir un gestionnaire de paquet est optionnel, mais peut sauver beaucoup de temps
- Beaucoup de librairies n'exportent pas leur cibles de la bonne façon

MERCI!

BONUS

NE PAS UTILISER file(GLOB)

QUOI CONSIDÉRER COMME UN
PROJET EXTERNE?

EXPORTATION DE CIBLES

```
install(TARGETS maLib1 maLib2 EXPORT monProjetTargets
        LIBRARY DESTINATION lib
        RUNTIME DESTINATION bin
        ARCHIVE DESTINATION lib
)

install(
    EXPORT monProjetTargets FILE monProjetTargets.cmake
    NAMESPACE monProjet::
    DESTINATION lib/cmake/monProjet
)
```

Optionnellement, exporter le *build tree*

```
export(
    EXPORT monProjetTargets FILE monProjetTargets.cmake
    NAMESPACE monProjet::
)
```

EXPORTATION DE CIBLES

```
include(CMakePackageConfigHelpers)
write_basic_package_version_file(
    monProjetConfigVersion.cmake
    COMPATIBILITY SameMajorVersion
)
configure_file(
    monProjetConfig.cmake.in
    monProjetConfig.cmake @ONLY
)
```

```
include(CMakeFindDependencyMacro)
find_dependency(cpplocate 0.5.0)
find_dependency(nlohmann_json 3.6.0)
include("${CMAKE_CURRENT_LIST_DIR}/monProjetTargets.cmake")
```

EXPORTATION DE CIBLES

```
include(CMakePackageConfigHelpers)
write_basic_package_version_file(
    monProjetConfigVersion.cmake
    COMPATIBILITY SameMajorVersion
)
```

```
configure_file(
    monProjetConfig.cmake.in
    monProjetConfig.cmake @ONLY
)
```

```
include(CMakeFindDependencyMacro)
find_dependency(cpplocate 0.5.0)
find_dependency(nlohmann_json 3.6.0)
include("${CMAKE_CURRENT_LIST_DIR}/monProjetTargets.cmake")
```

AnyNewerVersion, SameMajorVersion, ExactVersion