cmake_converter Documentation

Release 1.2

Estrada Matthieu

Aug 02, 2019

Contents

1 Introduction **3**
- 1.1 About CMake Converter . . . 3
- 1.2 Features . . . 3
- 1.3 Release cycle . . . 3
- 1.4 About CMake . . . 3

2 Install CMake Converter **5**
- 2.1 Requirements . . . 5
- 2.2 Installation (from Pip) . . . 5
- 2.3 Installation (from Sources) . . . 5
  - 2.3.1 Clone and install . . . 5
  - 2.3.2 External Libraries . . . 5

3 Use CMake Converter **7**
- 3.1 Quick Use . . . 7
- 3.2 Advance Usage . . . 7
- 3.3 Project Conversion . . . 8
- 3.4 Solution Conversion . . . 9
- 3.5 Hints . . . 10

4 Generated CMakeLists.txt **11**
- 4.1 Top of file . . . 11
- 4.2 Variables . . . 11
- 4.3 CMake Project . . . 12
- 4.4 Artefacts Output . . . 12
- 4.5 Includes . . . 12
- 4.6 Dependencies . . . 12
- 4.7 Files & Targets . . . 12
  - 4.7.1 Files . . . 12
  - 4.7.2 Library & Executable . . . 13
- 4.8 Flags . . . 13
  - 4.8.1 Linux . . . 13
  - 4.8.2 Windows . . . 13
- 4.9 Links . . . 13

5 Use CMake **15**

6 API Documentation **17**
6.1 DataConverter 17
6.2 Data Files 17
6.3 Dependencies 18
6.4 Flags 19
6.5 ProjectFiles 20
6.6 ProjectVariables 20
6.7 Utils 21

7 Indices and tables **23**

Python Module Index **25**

Index **27**

Documentation Content:

CHAPTER 1

Introduction

1.1 About CMake Converter

CMake Converter is an open source software written in Python under the terms of the GNU Affero General Public License .

This application is for developers and integrators who want to automate the creation of `CMakeLists.txt` for their compilations, from Visual Studio's `vcxproj` files.

1.2 Features

CMake Converter converts your `vcxproj` file into `CMakeLists.txt`. He adds all the contained data such as compilation Flags, project targets, project dependencies, outputs of the produced binaries and more.

1.3 Release cycle

CMake Converter has no strict schedule for releasing.

Other features will come in the next versions and you can propose new features through project issues. Each feature is discussed in a separate issue.

1.4 About CMake

In this documentation, you'll find some reminders about CMake and how the script handles your project's data inside. For example, how the generated **CMakeLists.txt** manage dependencies.

But a minimum of knowledge on CMake is **recommended** !

CHAPTER 2

Install CMake Converter

2.1 Requirements

You must have **Python 3** installed to make this library work.

Note: CMake Converter is **not compatible** with Python 2 !

2.2 Installation (from Pip)

You can install CMake-Converter as a standard python library, with `pip3`:

```
# If you're under Windows, use "pip" instead "pip3"
pip3 install cmake_converter
```

2.3 Installation (from Sources)

2.3.1 Clone and install

To install from sources, you've to clone this repository and make a pip install:

```
git clone https://github.com/algorys/cmakeconverter.git
cd cmakeconverter
# If you're under Windows, use "pip" instead "pip3"
pip3 install .
```

2.3.2 External Libraries

You need to install Python modules that are listed in `requirements.txt` file with pip:

```
colorama
docopt
lxml
```

CHAPTER 3

Use CMake Converter

3.1 Quick Use

To use CMake Converter, simply give your `vcxproj` file to cmake-converter command:

```
cmake-converter -p /path/to/your/project.vcxproj
```

3.2 Advance Usage

The `cmake-converter` command accepts a lot of parameters to try to suit the majority of situations.

CMake-Converter command line interface:

```
Usage:
    cmake-converter [-h]
    cmake-converter [-V]
    cmake-converter (-s solution | -p project) [-i]
                    [-c cmake]
                    [-D dependency...]
                    [-O cmakeoutput]
                    [-a codefile]
                    [-I cmakefile]
                    [-S std]

Options:
    -h, --help                  Show this help message and exit
    -V, --version               Show application version
    -s, --solution=solution     Path to a MSVC solution (.sln) (BETA)
    -p, --project=project       Path to a MSVC project (.vcxproj)
    -c, --cmake=cmake           Specify output of generated CMakeLists.txt
    -D, --dependencies=dep      Replace dependencies found in ".vcxproj" file. This␣
↪parameter,
```

(continues on next page)

(continued from previous page)

```
                                can be used multiple times for each dependency.
    -O, --cmakeoutput=output    Define output of artefact produces by CMake.
    -i, --include               Add include directories in CMakeLists.txt. [default:␣
↪False]
    -a, --additional=file       Import cmake code from any file to generated␣
↪CMakeLists.txt.
    -I, --includecmake=file     Add Include directive for given file in CMakeLists.
↪txt.
    -S, --std=std               Choose your C++ std version. [default: c++11]

Use cases:
    Display help message:
        cmake-converter (-h | --help)
    Display current version:
        cmake-converter (-V | --version)
    Convert an MSVC project and defines output of artefacts:
        cmake-converter -p ../msvc/foo/foo.vcxproj -c ../cmake/foo -O ../build/
↪compiler
    Convert an MSVC project and add ".cmake" file:
        cmake-converter -p ../msvc/foo/foo.vcxproj -I ../cmake/foo/file.cmake
    Convert an MSVC project with specific STD version:
        cmake-converter -p ../msvc/foo/foo.vcxproj -c ../cmake/foo -S c++17

Hint and tips:
    It is important to check that the generated CMake files are working properly␣
↪before using
    them in production.

    CMake Converter only manage Debug and Release build types. You can provide the␣
↪build type
    by add "-DCMAKE_BUILD_TYPE=<BUILD_TYPE>" to cmake command.

    CMake-Converter will try to respect as much as possible the data of your MSVC␣
↪project.
    However, it is necessary to pay attention to the relative paths of the files.

    If your project is in the following path: "../msvc/foo", your CMakeLists file␣
↪should have
    the same tree level. The same is to be done for the files to include !

Solutions conversion:
    The solution conversion is still in BETA and may therefore have some problems !

    Your Visual Studio projects must each have their respective folders for the␣
↪conversion
    to work.

    Note that the "--dependencies" and "--cmake" parameters are not used during␣
↪solution
    conversion !
```

3.3 Project Conversion

With the following project structure:

```
project/
├── cmake
│   ├── additional_code.txt
│   └── File.cmake
└── msvc
    ├── myexec.sln
    └── myexec.vcxproj
```

You can use cmake-converter as follows:

```
cmake-converter \
--project=../project/msvc/myexec.vcxproj \
--cmake=../project/cmake/ \
--cmakeoutput=../project/cmake \
--additional=../project/cmake/additional_code.txt \
--includecmake=../cmake/File.cmake \
--std=c++17 \
--include
```

3.4 Solution Conversion

With CMake-Converter, you can also convert full Visual Studio solutions. For the moment, this feature is still in BETA, but remains functional.

The script will extract data from all vcxproj and create the corresponding **CMakeLists.txt**.

IMPORTANT: Each `vcxproj` included in the solution must be in a **dedicated** directory, to ensure the smooth conversion.

Note: the `--dependencies` and `--cmake` parameters **can not** be used (and will not be used) during solution conversion !

With the following project structure:

```
project/
├── externals
│   └── cmake
│       └── File.cmake
└── msvc
    ├── libone
    │   └── libone.vcxproj
    ├── libtwo
    │   └── libtwo.vcxproj
    └── myexec
        ├── myexec.sln
        └── myexec.vcxproj
```

Then you'll run cmake-converter as follow:

```
cmake-converter \
--solution=project/msvc/myexec/myexec.sln \
--cmakeoutput=project/build/x64/ \
--includecmake=../../externals/cmake/File.cmake \
--std=c++17 \
--include
```

And you'll have the following CMakeLists.txt generated:

```
project/
├── externals
│   └── cmake
│       └── File.cmake
└── msvc
    ├── libone
    │   ├── CMakeLists.txt *
    │   └── libone.vcxproj
    ├── libtwo
    │   ├── CMakeLists.txt *
    │   └── libtwo.vcxproj
    └── myexec
        ├── CMakeLists.txt *
        ├── myexec.sln
        └── myexec.vcxproj
```

3.5 Hints

If you use **—cmake** parameter, ensure that given path has same directory level than your **.vcxproj**.

If you use **—includecmake** parameter, you have to give path relative to project itself, and not for script !

You can use CMake variables in parameters, by escaping `$` with a backslash. Example: `--cmake=../\${CMAKE_BINARY_DIR}/x64`.

If you use variables defined by yourself, make sure that they are defined in a **.cmake** file or the code you are importing !

CHAPTER 4

Generated CMakeLists.txt

All **CMakeLists.txt** generated by cmake-converter follow the same hierarchy. If you converted a solution, each converted vcxproj will have its own file.

In order to facilitate their understanding, the generated files are organized by "section". Here is a description for each of them.

4.1 Top of file

To keep track of the generation of the script and what it corresponds to, you will have in front of the file several information. Like the creation date, corresponding Visual Studio project, ...

4.2 Variables

The first part of the file groups the variables generated for your project. They will be used throughout the file. Nothing prevents you from modifying them if something does not please you, but you will need to be careful with these changes.

You normally have:

- `${PROJECT_NAME}`: name of your project, corresponding to name in vcxproj.
- `${OUTPUT_DEBUG}`: define folder for **Debug** compilations.
- `${OUTPUT_RELEASE}`: define folder for **Release** compilations.

If your project have some dependencies, CMake-converter will try to set variables to corresponding folders.

4.3 CMake Project

After variables definitions, you'll have the project itself and the default build type used (**Release**). You can override this by:

```
cmake -DCMAKE_BUILD_TYPE=<TARGET> .
```

Note: CMake Converter only manage *Debug* or *Release* build type, because Visual Studio projects doesn't manage other CMake build types (like *RELWITHDEBINFO* or *MINSIZEREL*).

4.4 Artefacts Output

If you've or not use `-O` parameter with CMake Converter, he try to define default output for your binaries produced during compilation. Each build type have a separate folder.

4.5 Includes

This part will vary depending on what you have pass as parameter to the converter.

- If your Visual Studio project have include folders, `--include` parameter will tell to converter to add them here.
- If you have pass `--additional` parameter to add code from a file, cmake-converter will add your code here.
- Finally, the parameter `--includecmake` will include your file here.

4.6 Dependencies

Dependencies are other projects you have set in your **vcxproj** project, like shared or static libraries. You have an option call **BUILD_DEPENDS** who attempt to link with them.

By default this option is set to `ON`.

If **ON**, CMake tries to find corresponding **CMakeLists.txt** inside each dependency folder. CMake-converter will define variables to define the folders of your dependencies, which you can find at the top of the file. These folders can be override by the `--dependencies` parameter.

If **OFF**, he tries to link library (*.so* or *.dll*) in a folder call *dependencies/libname/build*. This folder can also be change if needed.

4.7 Files & Targets

4.7.1 Files

Converter will also collect all your source files and add them to the corresponding target.

4.7.2 Library & Executable

After script get all information, he create your library (*STATIC* or *SHARED*) or your executable. If needed, he add dependencies too and link them.

4.8 Flags

The biggest part of the work done by CMake Converter. CMake-converter will add flags for each `${CMAKE_BUILD_TYPE}`.

4.8.1 Linux

On Linux, flags are usually set properly but script add minimum to use. The only flag that will change will be the version of the standard library (c++11 by default) with the **–std** parameter.

4.8.2 Windows

Each xml tag will be scanned to find all the flags to pass to the compiler, by `target_compile_options` directive.

4.9 Links

Finally if links with other libraries or project dependencies are required, you will find them at the end of the file.

CHAPTER 5

Use CMake

CMake Converter try to have the most informations as possible of your **vcxproj** file. However, it's recommended to read and test the generated **CMakeLists.txt** before using it in production !

Once CMake Converter has generated a **CMakeLists.txt** file, to compile with CMake, type the following commands:

```
# It's better to create a dedicated folder:
mkdir build && cd build
# Generate the "Makefile"
cmake ../to/cmakelists/folder/
# Launch compilation
cmake --build .
```

You can also provide a specific **Generator** with `-G "<GENERATOR_NAME>"`. Please refer to CMake Generator Documentation.

You can provide the build type by add `-DCMAKE_BUILD_TYPE=<BUILD_TYPE>`.

CMake provides a lot of other options that you can discover in their official documentation.

CHAPTER 6

API Documentation

6.1 DataConverter

Manage conversion of **vcxproj** data

class cmake_converter.data_converter.**DataConverter**(*data*)
Bases: object

Class who convert data to CMakeLists.txt.

close_cmake_file()
Close the "CMakeLists.txt" file

create_data()
Create the data and convert each part of "vcxproj" project

init_files(*vs_project*, *cmake_lists*)
Initialize opening of CMakeLists.txt and VS Project files

Parameters

- **vs_project** (*str*) – Visual Studio project file path
- **cmake_lists** (*str*) – CMakeLists.txt file path

write_cmake_headers(*vs_project*)
Write generation informations and set CMake version

6.2 Data Files

Manage the **VS Project** data and creation of **CMakeLists.txt** file

cmake_converter.data_files.**get_cmake_lists**(*cmake_path=None*)
Create CMakeLists.txt file in wanted "cmake_path"

Parameters **cmake_path** (*str*) – path where CMakeLists.txt should be write

Returns cmake file wrapper opened

Return type _io.TextIOWrapper

cmake_converter.data_files.**get_definitiongroup**(*target_platform*)
Return ItemDefinitionGroup namespace depends on platform and target

Parameters **target_platform** (*str*) – wanted target (Debug|Win32, Release|x64, ...)

Returns wanted ItemDefinitionGroup namespace

Return type str

cmake_converter.data_files.**get_propertygroup**(*target_platform*, *attributes=''*)
Return "propertygroup" value for wanted platform and target

Parameters

- **target_platform** (*str*) – wanted target (Debug|Win32, Release|x64, ...)
- **attributes** (*str*) – attributes to add to namespace (@Label="Configuration", ...)

Returns "propertygroup" value

Return type str

cmake_converter.data_files.**get_vcxproj_data**(*vs_project*)
Return xml data from "vcxproj" file

Parameters **vs_project** (*str*) – the vcxproj file

Returns dict with VS Project data

Return type dict

6.3 Dependencies

Manage directories and libraries of project dependencies

class cmake_converter.dependencies.**Dependencies**(*data*)
Bases: object

Class who find and write dependencies of project, additionnal directories...

add_dependencies()
Add dependencies to CMake project

get_dependency_target_name(*vs_project*)
Return dependency target name of VS Project

Parameters **vs_project** (*str*) – the .vcxproj file

Returns project name or empty string

Return type str

link_dependencies()
Write link dependencies of project.

write_dependencies()
Write on "CMakeLists.txt" subdirectories or link directories for external libraries.

write_include_dir()
Write on "CMakeLists.txt" include directories required for compilation.

6.4 Flags

Manage compilation flags of project

class cmake_converter.flags.**Flags**(*data*)
Bases: object

Class who check and create compilation flags

available_std = ['c++11', 'c++14', 'c++17']

define_defines()
DEFINES

define_group_properties()
Define the PropertyGroups and DefinitionGroups of XML properties

define_linux_flags()
Define the Flags for Linux platforms

define_settings()
Define settings of project for each configuration

define_windows_flags()
Define the Flags for Win32 platforms

set_exception_handling()
Set ExceptionHandling flag: /EHsc

set_function_level_linking()
Set FunctionLevelLinking flag: /Gy

set_generate_debug_information()
Set GenerateDebugInformation flag: /Zi

set_intrinsic_functions()
Set Intrinsic Functions flag: /Oi

set_optimization()
Set Optimization flag: /Od

set_runtime_library()
Set RuntimeLibrary flag: /MDd

set_runtime_type_info()
Set RuntimeTypeInfo flag: /GR

set_use_debug_libraries()
Set Use Debug Libraries flag: /MD

set_warning_level()
Set Warning level for Windows: /W

set_whole_program_optimization()
Set Whole Program Optimization flag: /GL

write_defines_and_flags()
Get and write Preprocessor Macros definitions

write_flags()
Parse all flags properties and write them inside "CMakeLists.txt" file

6.5 ProjectFiles

Manages the recovery of project files

class cmake_converter.project_files.**ProjectFiles**(*data*)
Bases: object

Class who collect and store project files

add_additional_code(*filename*)
Add additional file with CMake code inside

Parameters **filename** (*str*) – the file who contains CMake code

add_include_cmake(*filename*)
Add include directive for filename

Parameters **filename** (*str*) – name of file to include

add_target_artefact()
Add Library or Executable target

collects_source_files()
Collects the project source files in CMakeLists.txt file

write_source_files()
Write source files variables to file() cmake function

6.6 ProjectVariables

Manage creation of CMake variables that will be used during compilation

class cmake_converter.project_variables.**ProjectVariables**(*data*)
Bases: object

Class who defines all the CMake variables to be used by the project

add_cmake_output_directories()
Add output directory for each artefacts CMake target

add_cmake_project(*language*)
Add CMake Project

Parameters **language** (*list*) – type of project language: cpp | c

add_default_target()
Add default target release if not define.

add_dependencies_variables()
Add dependencies variables

add_output_variables()
Add output variables

add_project_variables()
Add main CMake project variables

static cleaning_output(*output*)
Clean Output string by remove VS Project Variables

Parameters **output** (*str*) – Output to clean

Returns clean output

Return type str

write_project_messages()
Write some messages for project to inform user during cmake generation

6.7 Utils

Utils manage function needed by converter

cmake_converter.utils.**get_title**(*title*, *text*)
Return formatted title for writing

Parameters

- **title** (*str*) – main title text
- **text** (*str*) – text related to title

Returns formatted title

Return type str

cmake_converter.utils.**message**(*text*, *status*)
Displays a message while the script is running

Parameters

- **text** (*str*) – content of the message
- **status** (*str*) – level of the message (change color)

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

cmake_converter.data_converter, 17
cmake_converter.data_files, 17
cmake_converter.dependencies, 18
cmake_converter.flags, 18
cmake_converter.main, 7
cmake_converter.project_files, 19
cmake_converter.project_variables, 20
cmake_converter.utils, 21

Index

A

add_additional_code() (*cmake_converter.project_files.ProjectFiles method*), 20
add_cmake_output_directories() (*cmake_converter.project_variables.ProjectVariables method*), 20
add_cmake_project() (*cmake_converter.project_variables.ProjectVariables method*), 20
add_default_target() (*cmake_converter.project_variables.ProjectVariables method*), 20
add_dependencies() (*cmake_converter.dependencies.Dependencies method*), 18
add_dependencies_variables() (*cmake_converter.project_variables.ProjectVariables method*), 20
add_include_cmake() (*cmake_converter.project_files.ProjectFiles method*), 20
add_output_variables() (*cmake_converter.project_variables.ProjectVariables method*), 20
add_project_variables() (*cmake_converter.project_variables.ProjectVariables method*), 20
add_target_artefact() (*cmake_converter.project_files.ProjectFiles method*), 20
available_std (*cmake_converter.flags.Flags attribute*), 19

C

cleaning_output() (*cmake_converter.project_variables.ProjectVariables static method*), 20
close_cmake_file() (*cmake_converter.data_converter.DataConverter method*), 17
cmake_converter.data_converter (*module*), 17
cmake_converter.data_files (*module*), 17
cmake_converter.dependencies (*module*), 18
cmake_converter.flags (*module*), 18
cmake_converter.main (*module*), 7
cmake_converter.project_files (*module*), 19
cmake_converter.project_variables (*module*), 20
cmake_converter.utils (*module*), 21
collects_source_files() (*cmake_converter.project_files.ProjectFiles method*), 20
create_data() (*cmake_converter.data_converter.DataConverter method*), 17

D

DataConverter (*class in cmake_converter.data_converter*), 17
define_defines() (*cmake_converter.flags.Flags method*), 19
define_group_properties() (*cmake_converter.flags.Flags method*), 19
define_linux_flags() (*cmake_converter.flags.Flags method*), 19
define_settings() (*cmake_converter.flags.Flags method*), 19
define_windows_flags() (*cmake_converter.flags.Flags method*), 19
Dependencies (*class in cmake_converter.dependencies*), 18

F

Flags (*class in cmake_converter.flags*), 19

G

get_cmake_lists() (*in module cmake_converter.data_files*), 17

get_definitiongroup() (*in module cmake_converter.data_files*), 18
get_dependency_target_name() (*cmake_converter.dependencies.Dependencies method*), 18
get_propertygroup() (*in module cmake_converter.data_files*), 18
get_title() (*in module cmake_converter.utils*), 21
get_vcxproj_data() (*in module cmake_converter.data_files*), 18

I

init_files() (*cmake_converter.data_converter.DataConverter method*), 17

L

link_dependencies() (*cmake_converter.dependencies.Dependencies method*), 18

M

message() (*in module cmake_converter.utils*), 21

P

ProjectFiles (*class in cmake_converter.project_files*), 20
ProjectVariables (*class in cmake_converter.project_variables*), 20

S

set_exception_handling() (*cmake_converter.flags.Flags method*), 19
set_function_level_linking() (*cmake_converter.flags.Flags method*), 19
set_generate_debug_information() (*cmake_converter.flags.Flags method*), 19
set_intrinsic_functions() (*cmake_converter.flags.Flags method*), 19
set_optimization() (*cmake_converter.flags.Flags method*), 19
set_runtime_library() (*cmake_converter.flags.Flags method*), 19
set_runtime_type_info() (*cmake_converter.flags.Flags method*), 19
set_use_debug_libraries() (*cmake_converter.flags.Flags method*), 19
set_warning_level() (*cmake_converter.flags.Flags method*), 19
set_whole_program_optimization() (*cmake_converter.flags.Flags method*), 19

W

write_cmake_headers() (*cmake_converter.data_converter.DataConverter method*), 17
write_defines_and_flags() (*cmake_converter.flags.Flags method*), 19
write_dependencies() (*cmake_converter.dependencies.Dependencies method*), 18
write_flags() (*cmake_converter.flags.Flags method*), 19
write_include_dir() (*cmake_converter.dependencies.Dependencies method*), 18
write_project_messages() (*cmake_converter.project_variables.ProjectVariables method*), 21
write_source_files() (*cmake_converter.project_files.ProjectFiles method*), 20