pycmake Documentation

Release 0.1

Matthieu Estrada

August 30, 2016

Contents

Contents:

CHAPTER 1

Introduction

This project is a Python 3 module to create, manage and build CMake projects.

WARNING: currently, PyCMake still under development and may not be install on production server.

You'll can make project, add library and executable, create and choose compilers, add variables, dependencies and most features as possible.

Finally, you'll can write your CMakeLists.txt and build it.

CHAPTER 2

Installation

2.1 Requirements

Actually, PyCMake only require a version of Python up to 3.0. this library is not test under Python 2.7 and above, but should work.

2.2 Installation

2.2.1 PyCMake with pip

Available as soon as possible

2.2.2 PyCMake Release

Available as soon as possible

2.2.3 PyCMake from Source

Simply clone repos of PyCmake and run *setup.py*:

```
sudo setup.py install
```

CHAPTER 3

CMake

Before beginning to create a project and try to compile it with PyCmake, you must create a *CMake* object. He will used to manage common features and can receive your compilers.

cmake = CMake()

Now you can add set global settings of CMake:

```
min_required = 'VERSION 3.5'
policy = 'VERSION 3.5'
cmake.add_settings(min_required, policy)
```

3.1 Compilers

You must add at least one compiler to get PyCmake functional. Then you can add other compilers, flags for each of them and manage global settings of CMake.

Valid **compiler_id** are currently:

- GCC or G++
- CLANG or CLANG++
- MSVC or MSVC++

Let's create a *Compiler* for GNU:

```
compiler = Compiler()
compiler_id = 'G++'
compiler.create('G++-5', 'C++', compiler_id, 5, '/usr/bin/g++-5')
```

Now that the compiler was created, we can add it to our cmake object. CMake object has method and members for each supported compiler:

```
cmake.gnu_compiler(compiler)
# Or for Clang:
# cmake.clang_compiler(compiler)
```

The advantage with the object Compiler is that you can easily use `create()` to create a new one and add it to our object CMake. But take care, it will replace the previous values.

3.2 Flags

Your compiler can receive flags to ensure your project compiles as needed. You need object *Flags* to make it:

```
gcc_flags = Flags('G++-5 Flags', '-std=c++11', 'Wall', '-GL')
cmake.flags_to_compiler(compiler_id, gcc_flags)
```

As you can see, flags name is not important, that's **compiler_id** who make the link between your flags and your compilers.

Now your CMake is ready to receive a Project.

CHAPTER 4

Project

4.1 Create a project

The *Project* is the heart of your script. He will contains all information about your project sources, dependencies, links, definitions, ...

Initialise object and create your project:

```
project = Project()
language = 'C++'
project.create('myLib', language)
```

Currently, only **C** and **C++** are valid language. During `create()`, PyCMake create a variable named **PROJECT_NAME** (See below).

4.2 CMake Variables

To facilitate read and management of your project, PyCMake will help you to generate variable you can use after along the process.

There is some default variables who will be created and you can create your own if needed.

4.2.1 Predefined Variables

- **PROJECT_NAME:** when the `create()` method is called, name of your project is automatically associated with this variable.
- **PROJECT_DIR:** you can use `project_dir()` method to set this variable. **WARNING:** you have to indicate a relative path from your future **CMakeLists.txt** location ! Cause this variable will define absolute path from this.
- **OUTPUTS:** you have 3 methods for each type of target. You have to give the path for each.
  - `library_output_path()`
  - `archive_output_path()`
  - `executable_output_path()`

Here is a way to use it:

```
project.variables.library_output_path('${PROJECT_DIR}/build')
```

Feel free to use existing variables in your paths.

4.2.2 Custom Variables

You can also add custom variables to your project. Simply type the following:

```
project.variables.add('TEST_DIR', '${PROJECT_DIR}/src/tests')
```

You can add as many variables as you want or replace existing ones. The *Project* object provides the `get_variable()` method to access any variable created.

4.3 Targets

Now that your project is defined, you must add target(s) to build. There is 2 types of targets : libraries and executables.

4.3.1 Libraries

You have to precise the **true** name of your library. She can be shared or static.

For a shared library called *libmylib.so* (or *mylib.dll* on Windows):

```
project.add_library_target('mylib', shared=True)
```

For a static library called *libmylib.a* (or *mylib.lib* on Windows):

```
project.add_library_target('mylib')
```

The **shared** option is false by default.

4.3.2 Executables

You have to give the **true** name of your executable. For an executable called *myexe* (or *myexe.exe* on Windows):

```
project.add_executable_target('myexe')
```

That's all.

4.4 Files and Directories

Note: these methods will be reworked in the future to facilitate the addition of files and folders.

There are two distinct methods in PyCMake to add folders or files to your target. Each must receive a `tuple` of them to get it work. They can be append to your *PROJECT_DIR* variable or not.

For folders, you can set *recursive* mode or not.

Here is a full example for a library and his folders:

```
project.add_library_target('mylib', shared=True)
project.add_source_directories('dir_cpp',
                               'mylib',
                               True,
                               False,
                               '../../lib/src/*.cpp',
                               '../../lib/src/test/*.cpp',
                               )
project.add_source_directories('dir_header',
                               'mylib',
                               True,
                               False,
                               '../../lib/src/includes/*.h',
                               '../../lib/src/test/includes/*.h',
                               )
```

And here, for add specific files:

```
project.add_source_files('cpp_files',
                         'mylib',
                         True,
                         '../../main.cpp',
                         '../../graphics.cpp',
                         )
project.add_source_files('headers_files',
                         'mylib',
                         True,
                         '../../stdafx.h',
                         '../../main.h',
                         '../../graphics.h',
                         )
```

PyCMake then associate these files to the target to compile.

4.5 Preprocessor Definitions

If your project need specific definitions for preprocessor, you can set it like that:

```
project.preprocessor_definitions('UNICODE', '_UNICODE', 'MYLIB_EXPORTS')
```

Easy and simple.

CHAPTER 5

Dependencies

5.1 Externals

CMake offers many way to add dependencies to your project. PyCmake use *Externals* object to manage this:

```
depends = Externals()
```

Currently, PyCMake supports *add_subdirectory* for other directory with CMakeLists projects. And you can *link_directories* to link binaries already built:

```
depends.add_subdirectory('zlib', '${PROJECT_DIR}/external/zlib/', '${PROJECT_DIR}/build/zlib')
depends.add_link_directories(('${PROJECT_DIR}/external/g3log')
```

5.2 Links

You can link your project with your dependencies. Simply tell which target you want to link with them. If the target exists in your project, PyCmake will link them:

```
depends.target_link_libraries('mylib', 'zlib', 'g3log')
project.add_dependencies(depends)
```

CHAPTER 6

CMakeLists

6.1 What you need to do before

You must have an instance of *CMake* and *Project* create and configured with your requirements to use *CMakeLists*.

6.2 Create CMakeLists

Once your project is properly configured, you can create your *CMakeLists.txt*. This file is needed by CMake (and of course by PyCMake too) to compile your project.

Create a *CMakeLists* object:

```
cmakelist = CMakeLists()
```

Initialize file and write it:

```
# PyCmake will try to create folders if not exists.
cmakelist.init_file('./platform/cmake')
cmakelist.write_cmakelists(cmake, project)
```

Normally, you have a **CMakeLists.txt** ready to use, created in the specified folder !

CHAPTER 7

pycmake package

7.1 Module contents

PyCMake

This module is a tool for CMake to help create, manage and build CMake Projects.

7.2 Submodules

7.3 pycmake.cmake

class pycmake.cmake.**CMake**
Bases: object

CMake is root module of **PyCMake**. He manage all to provide CMake project.

add_settings(*min_required*, *policy*)
Set **cmake_minimum_required** and **cmake_policy**.

Parameters

- **min_required** (*str*) – the cmake version minimum required.
- **policy** (*str*) – the policies of project.

clang_compiler(*compiler*)
Add a Clang Compiler to CMake.

Parameters **compiler** (Compiler) – Clang Compiler to add. Must be created before.

flags_to_compiler(*compiler_id*, *flags*)
Add Flags to a specific compiler.

Parameters

- **compiler_id** (*str*) – supported compiler_id.
  - [GCC or G++],
  - [CLANG or CLANG++],
  - [MSVC or MSVC++]
- **flags** (Flags) – Flags to add to the compiler.

gnu_compiler(*compiler*)
Add a GNU Compiler to CMake.

Parameters **compiler** (Compiler) – Gnu Compiler to add. Must be created before.

msvc_compiler(*compiler*)
Add a MSVC Compiler to CMake object.

Parameters **compiler** (Compiler) – MSVC Compiler to add. Must be created before.

7.4 pycmake.cmakelists

class pycmake.cmakelists.**CMakeLists**
Bases: object

CMakeLists create and generate CMakeLists.txt.

init_file(*path*)
Create folders and CMakeLists.txt.

Parameters **path** (*str*) – path where to create CMakeLists.txt.

write_clang_flags(*clang_flags*)
Write Flags for compilers.

Parameters **clang_flags** (*dict*) – Flags for Clang compiler.

write_cmakelists(*cmake*, *project*)
Write CMakeLists.txt from the CMake data.

Parameters

- **cmake** (CMake) – CMake object, with *Compiler* and *Flags*.
- **project** (Project) – Project object with his target, sources and *Externals*.

write_dependencies(*dependencies*)
Write dependencies of project.

Parameters **dependencies** (Externals) – Dependencies of the project.

write_directory_files(*sources_dirs*)
Write different variables for directories of project.

Parameters **sources_dirs** (*dict*) – Sources Directories.

write_global_settings(*settings*)
Write settings of CMake.

Parameters **settings** (*dict*) – global settings of CMake

write_gnu_flags(*gnu_flags*)
Write Flags for compilers.

Parameters **gnu_flags** (*dict*) – Flags for GNU compiler.

write_info()
Write global informations.

write_links(*dependencies*)
Write Links for dependencies of project.

Parameters **dependencies** (Externals) – Dependencies of the project.

write_msvc_flags(*msvc_flags*)
Write Flags for compilers.

Parameters **msvc_flags** (*dict*) – Flags for MSVC compiler.

write_project_data(*language*, *definitions*)
Write project and definitions.

Parameters

- **language** (*str*) – language of project.
- **definitions** (*tuple*) – definitions of project.

write_targets(*project*)
Write Targets and add sources Variables.

Parameters **project** (Project) – CMake Project.

write_title(*title*)

write_variables(*project*)
Write Project variables and data.

Parameters **project** (Project) – project to build.

7.5 pycmake.compiler

class pycmake.compiler.**Compiler**
Bases: object

Compilers define a compiler.

static check_compiler_options(*language*, *compiler_id*)
Check if compiler is valid. Used for each *create()*.

Parameters

- **language** (*str*) – language of compiler (C or CXX)
- **compiler_id** (*str*) – compiler_id (GCC, G++, CLANG, CLANG++, MSVC or MSVC++)

create(*name*, *language*, *compiler_id*, *version*, *executable*)
Create a compiler.

Parameters

- **name** (*str*) – name of compiler.
- **language** (*str*) – language of compiler
- **compiler_id** (*str*) – compiler (GCC, G++, CLANG, CLANG++, MSVC or MSVC++)
- **version** (*int or float*) – version of the compiler.
- **executable** (*str*) – full path to the executable.

7.6 pycmake.externals

class pycmake.externals.**Externals**
Bases: object

Externals contains all dependencies related to project.

add_link_directories(**directories*)
Link with the specified directories.

Parameters **directories** (*tuple*) – directories in which the linker will look for libraries.

add_subdirectory(*subdir_id*, *source_dir*, *binary_dir*)
Add one subdirectory to the build.

Parameters

- **subdir_id** (*str*) – id of the subdir.
- **source_dir** (*str*) – directory in which the source CMakeLists.txt is located
- **binary_dir** (*str*) – directory in which to place the output files.

target_link_libraries(*target*, **libraries*)
Link the libraries specified to the associated target.

Parameters

- **target** (*str*) – relevant target.
- **libraries** (*tuple*) – libraries to link to target.

7.7 pycmake.flags

class pycmake.flags.**Flags**(*flags_id*, *general*, *debug=''*, *release=''*)
Bases: object

Flags for general, debug and release compilations

debug = None

Parameters **debug** (*str*) – flags for debug target

flags_id = None

Parameters **flags_id** (*str*) – id of flags

general = None

Parameters **general** (*str*) – flags for all targets.

release = None

Parameters **release** (*str*) – flags for release target.

7.8 pycmake.project

class pycmake.project.**Project**
Bases: object

CMakeProject contains all data related to project.

add_dependencies(*dependencies*)
Add some dependencies to project.

Parameters **dependencies** (Externals) – dependencies of the project, like subdirectories or external link.

add_executable_target(*name*)
Add an executable target.

Parameters **name** (*str*) – name of the executable.

add_library_target(*name*, *shared=False*)
Add a Library target.

Parameters

- **name** (*str*) – the library name.
- **shared** (*bool*) – shared library or not.

add_source_directories(*dirs_id*, *target*, *recursive*, *from_proj*, **sources*)
Add one or many sources directories to project.

Parameters

- **dirs_id** (*str*) – id of the directories.
- **target** (*str*) – add directories to a specific target.
- **recursive** (*bool*) – recursive or not
- **from_proj** (*bool*) – if True, append to ${PROJECT_DIR} variable, see `project_dir()`
- **sources** (*tuple*) – source directories to add.

add_source_files(*files_id*, *target*, *from_proj=False*, **files*)
Add one or many sources files to project.

Parameters

- **files_id** (*str*) – id of the files.
- **target** (*str*) – add files to a specific target.
- **from_proj** (*bool*) – add ${PROJECT_DIR} to source files if True.
- **files** (*tuple*) – files to add.

add_version(*major*, *minor*, *patch*, *tweak=0*)

Parameters

- **major** (*int*) – number of Major Version
- **minor** (*int*) – Number of Minor Version
- **patch** (*int*) – Number of Patch version
- **tweak** (*int*) – Number of Tweak version.

create(*name*, *language*)
Create a project.

Parameters

- **name** (*str*) – name of the project.
- **language** (*str*) – language of the project.

get_variable(*name*)
Returns the contents of the specified **variable**. Will look into *Variables*

Parameters **name** (*str*) – the name of the desired variable.

Returns a variable of the project.

Return type dict

preprocessor_definitions(**definitions*)
Add Preprocessor Definitions.

Parameters **definitions** (*tuple*) – add preprocessor definitions to project: FOO BAR

7.9 pycmake.supported

This file is only to tell what's compatible or not with **PyCMake**.

7.10 pycmake.variables

class pycmake.variables.**Variables**
Bases: object

Variables hold all project variables

add(*name*, *value*, *option='set'*)
Add a variable.

Parameters

- **name** (*str*) – Name of the variable.
- **value** (*str*) – Value of variable.
- **option** (*str*) – option for variable: 'set' or 'get_filename_component'

archive_output_path(*path*)
Add ARCHIVE_OUTPUT_PATH variable for Static libraries.

Parameters **path** (*str*) – path where build Static libraries.

executable_output_path(*path*)
Add EXECUTABLE_OUTPUT_PATH variable for executables.

Parameters **path** (*str*) – path where build executables.

library_output_path(*path*)
Add LIBRARY_OUTPUT_PATH variable for Shared libraries.

Parameters **path** (*str*) – path where build Shared libraries.

project_dir(*path*)
Defines the main project directory in a variable named: PROJECT_DIR.

Parameters **path** (*str*) – relative path from CMakeLists.txt.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

p

A

C

D

E

F

G

I

L

M

P

R

T

V

W