pycmake Documentation

Release 0.1

Matthieu Estrada

September 08, 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 (2.7 or above).

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)
# And for MSVC
# cmake.msvc_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.

For Windows:

You have to precise full path of `.bat` you want to execute before compiling, if you want to use `NMake Makefiles`.

Example: "C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin\amd64\vcvars64.bat".

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.

3.3 Before Continuing

When you use PyCmake, you must pay attention to the way that you will give your projects.

Paths should be **relative** depending on the folder in which your *CMakeLists.txt* will be located. No matter where you run your python script, your paths must first consider this location.

For example, you will run your script in `/home/user/scripts` or other, no matter.

Let's say your project sources are located in `/home/user/workspace/project/src` and your **CMakeLists.txt** will be write in folder `/home/user/workspace/project/platform/cmake`.

If you want to add sources, your paths will be something like `../../src`. Example:

```
project.add_source_files('files',
                         'mylib',
                         True,
                         '../../src/file.cpp'
                        )
```

See *Files and Directories* for more information.

Other case is when you want define **PROJECT_DIR** variable and use it throughout your script. You have to give the following path:

```
project.variables.project_dir('../../')
```

Because CMake must go up two folders to define the root of your project (see *CMake Variables*).

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 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.

4.5 Sources

Your target will obviously need files to be built. They are added by *Sources* object. Once done, simply add them to a target that you created:

```
project.add_sources_to_target('myexe', src)
```

See the **Next Section** for more details.

CHAPTER 5

Sources

Sources are the files you want to add to your target project. They must be created before adding them to a *Project* object.

To create sources files, you need to know more about *SRC_TYPE*.

5.1 Sources Files

Simply instantiate a *Sources* object:

```
files = Sources()
# You can create a list before
src = ['../../src/main.cpp', '../../src/conf.cpp']
# 'myfiles' will be the ID of your sources.
files.add('myfiles', SRC_TYPE[1], src)
```

That's all. You cannot make files recursive, but you can make their path relative from **PROJECT_DIR** variable:

```
files.add('files', SRC_TYPE[1], src, from_proj=True)
```

The **from_proj** variable is **False** by default.

5.2 Sources Directory

For sources directory, PyCmake will create a variable with **file** cmake command to add them to your target after. The process is the same as *Sources Files*, but you can make them resursive or not. You can specify, in your directory listing, the file extensions you want to include:

```
dirs = Sources()
src_dir = ['../../src/*.cpp', '../../src/include/*.h']
dirs.add('mydirs', SRC_TYPE[0], src_dir)
# You can make them recursive
dirs.make_resursive(True)
```

Recursive is **False** by default.

5.3 Add Sources to a Project

Once you have defined your Sources, you add them to a Project. You must specify a **valid target** to get it work. You can add multiple files or directory to the same target:

```
project.add_sources_to_target('mytarget', files)
project.add_sources_to_target('mytarget', dirs)
```

When PyCMake generate *CMakeLists.txt*, it automatically adds the source to the specified target.

CHAPTER 6

Dependencies

6.1 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')
```

6.2 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 7

CMakeLists

7.1 What you need to do before

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

7.2 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 8

PyCMake Package

8.1 Module contents

PyCMake

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

8.2 Submodules

8.3 pycmake.cmake

CMake manage all common settings to provide CMake project.

class pycmake.cmake.**CMake**

Bases: object

Class to manage all common settings.

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*) – id of compiler. For more details, see *C_COMPILER* or *CXX_COMPILER*.
- **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.

8.4 pycmake.cmakelists

CMakeLists create and generate CMakeLists.txt from a Project object.

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

Class who manage 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_preprocessor_definitions (*definitions*)
Write preprocessor definitions of project.

Parameters **definitions** (*tuple*) – preprocessor definitions.

write_project (*language*)
Write project and definitions.

Parameters **language** (*str*) – language of project.

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

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

write_title (*title*)
Write title of each section in CMakeLists.txt

Parameters **title** (*str*) – title to write

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

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

write_version (*version*)
Write version variables.

Parameters **version** (*dict*) – version numbers of project.

8.5 pycmake.compiler

Compiler define every supported compilers.

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

Class to define a compiler.

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

Parameters

- **language** (*str*) – language of project. For more details, see *LANGUAGE*.
- **compiler_id** (*str*) – id of compiler. For more details, see *C_COMPILER* or *CXX_COMPILER*.

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

Parameters

- **name** (*str*) – name of compiler.
- **language** (*str*) – language of project. For more details, see *LANGUAGE*.

- **compiler_id** (*str*) – id of compiler. For more details, see *C_COMPILER* or *CXX_COMPILER*.
- **version** (*int or float*) – version of the compiler.
- **executable** (*str*) – full path to the executable.

8.6 pycmake.externals

Externals contains all dependencies related to project.

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

Class to manage dependencies.

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.

8.7 pycmake.flags

Flags for compilers.

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

Class to manage general, debug and release flags.

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.

8.8 pycmake.project

Project contains all data related to project.

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

Class to manage project data.

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_sources_to_target(*target*, *src*)
Add sources directory or files to a specific target.

Parameters

- **target** (*str*) – existing target.
- **src** (Sources) – the sources 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. Ex: UNICODE

8.9 pycmake.sources

Sources contains files or directory for a Project.

class pycmake.sources.**Sources**
Bases: object

Class to manage project sources.

add(*name*, *src_type*, *sources*, *from_proj=False*)
Add sources with a specific type.

Parameters

- **name** (*str*) – name of the sources.
- **src_type** (SRC_TYPE) – type of the sources: *DIR* or *FILE*
- **from_proj** (*bool*) – sources relative from project directory.
- **sources** (*list*) – sources to add.

is_recursive()
Check if sources are recursive are not.

make_recursive(*recursive*)
Make sources directory recursive.

Parameters **recursive** (*bool*) – recursive or not. Default: False.

8.10 pycmake.supported

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

pycmake.supported.**CXX_COMPILER = ('G++', 'CLANG++', 'MSVC++')**

Variables ***CXX_COMPILER*** – supported C++ Compiler.

pycmake.supported.**C_COMPILER = ('GCC', 'CLANG', 'MSVC')**

Variables ***C_COMPILER*** – supported C Compilers.

pycmake.supported.**LANGUAGE = ('C', 'C++')**

Variables ***LANGUAGE*** – supported languages.

pycmake.supported.**SRC_TYPE = ('DIR', 'FILE')**

Variables ***SRC_TYPE*** – supported Sources.

8.11 pycmake.variables

Variables hold all project variables.

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

Class to manage 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 9

Indices and tables

- genindex
- modindex
- search

p

A

C

D

E

F

G

I

L

M

P

R

S

T

V

W