
cmake_converter Documentation

Release 2.1

Estrada Matthieu

Aug 19, 2020

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	6
3	Use CMake Converter	7
3.1	Quick Use	7
3.2	Advance Usage	7
3.2.1	Main	7
3.3	Solution Conversion	7
3.4	Hints	8
4	Generated CMakeLists.txt	9
4.1	Root of CMake tree	9
4.2	Top of file (project)	9
4.3	Files & Targets	9
4.3.1	Files (source groups)	9
4.3.2	Library & Executable (target)	10
4.3.3	Property files (*.props)	10
4.4	Include directories	10
4.5	Compile definitions	10
4.6	Compile and link options	10
4.6.1	Linux	10
4.6.2	Windows	10
4.7	Dependencies	11
5	Use CMake	13
6	API Documentation	15

6.1	DataConverter	15
6.2	Context object descriptor	16
6.3	Data Files	16
6.4	Dependencies	17
6.5	Flags	18
6.6	ProjectFiles	18
6.7	ProjectVariables	18
6.8	Utils	19
7	Indices and tables	23
	Python Module Index	25
	Index	27

Documentation Content:

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 build systems from Visual Studio solution files.

1.2 Features

CMake Converter converts your `*.sln` file (`vcxproj` and `vfproj` are supported only) into corresponding tree of `CMakeLists.txt`. It tries to translate data such as compile and link flags, project files, project dependencies, outputs of the produced binaries and more into CMake language.

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** !

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 pip:

```
pip install cmake_converter
```

Install last pre-release or development version of cmake-converter:

```
pip install --pre 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
pip install .
```

2.3.2 External Libraries

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

```
colorama
lxml
```

3.1 Quick Use

To use `cmake-converter`, simply give your `*.sln` file to `cmake-converter` command:

```
cmake-converter -s <path/to/file.sln>
```

3.2 Advance Usage

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

3.2.1 Main

Manage script arguments and launch

3.3 Solution Conversion

With `cmake-converter`, you can convert full Visual Studio solutions.

The script will extract data from all supported `*proj` files and create the corresponding **CMakeLists.txt**.

With the following project structure:

```
project/  
├── msvc  
│   ├── libone  
│   │   └── libone.vcxproj  
│   ├── libtwo  
│   │   └── libtwo.vcxproj
```

(continues on next page)

(continued from previous page)

```
└─ myexec
  └─ myexec.sln
  └─ myexec.vcxproj
```

Then you'll run cmake-converter as follow:

```
cmake-converter \
--solution=project/msvc/myexec/myexec.sln \
--verbose-mode \
--private-include-directories \
--warning-level=3
```

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

```
project/
└─ msvc
  └─ libone
    └─ CMakeLists.txt      *
    └─ libone.vcxproj
  └─ libtwo
    └─ CMakeLists.txt      *
    └─ libtwo.vcxproj
  └─ myexec
    └─ CMake                *
      └─ Default*.cmake    *
      └─ Utils.cmake        *
    └─ CMakeLists.txt      *
    └─ myexec.sln
    └─ myexec.vcxproj
```

3.4 Hints

You can add CMake/GlobalSettingsInclude.cmake file for global custom CMake settings.

Pay attention on warnings and do proposed fixes.

Run cmake-converter --help for more info.

Generated CMakeLists.txt

All **CMakeLists.txt** generated by cmake-converter follow the same hierarchy. If you converted a solution, each converted directory with *.proj file 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. Actually each section at generated scripts is separated with comment and can be read well.

4.1 Root of CMake tree

After conversion the root of CMake tree appears beside *.sln file. So, give the path of the root CMakeLists.txt to cmake to parse all converted tree. Root CMakeLists.txt contains info about:

- ```
1. Architectures used in solution.
2. Solution configuration names.
3. Windows SDK version used.
4. Sets cmake minimum required version.
5. Includes optional GlobalSettingsInclude.cmake
6. Adds all converted subdirectories with projects.
```

### 4.2 Top of file (project)

Creating of corresponding Visual Studio project with used languages.

### 4.3 Files & Targets

#### 4.3.1 Files (source groups)

Converter will collect all your source files and add them to the corresponding target. The files will be added according source groups and sorted alphabetically.

**IMPORTANT:** names of source groups must be without special symbols (only CMake like variable). Spaces are accepted.

### 4.3.2 Library & Executable (target)

After script get all information, he create your library (*STATIC* or *SHARED*) or your executable. Also here may be used `add_precompiled_header` if `PCH` is turned on.

### 4.3.3 Property files (\*.props)

Converter does not support translation of property. It adds links to correspondent files at the same location. Example: if we have following link at source xml:

```
f1/f2/my-settings.props
```

You'll get usage of similar cmake file that should be created manually:

```
use_props(${PROJECT_NAME} "${CMAKE_CONFIGURATION_TYPES}" "f1/f2/my-
settings.cmake")
```

In case of absence of cmake file CMake will throw warning:

```
Corresponding cmake file from props <cmake-file-name> doesn't exist
```

## 4.4 Include directories

Adds include directories from corresponding visual studio project. Includes are `PUBLIC` by default. But you may use `-private-include-directories` to make them private and make your solution smarter.

## 4.5 Compile definitions

Adds compile definitions from corresponding visual studio project.

## 4.6 Compile and link options

The biggest part of the work done by CMake Converter. CMake-converter will add flags for each `$<CONFIG>`. Only `MSVC` and `ifort` compilers are supported. Flags applied with `target_compile_options` and `target_link_options`.

### 4.6.1 Linux

On Linux only translation of `ifort` options is supported.

### 4.6.2 Windows

`MSVC` and `ifort` options are supported.

## 4.7 Dependencies

Dependencies are binaries you have set in “Additional Dependencies” of your **\*proj** project, like shared or static libraries or references to other solution projects. `add_dependencies` command contains corresponding references. `target_link_libraries` command contains references that need to link and other external dependencies. `target_link_directories` may be used here as well.

Also cmake converter tries to read info about used NuGet packages and makes stubs for using it with `use_package` function.





CMake Converter try to take as much information as possible from your **\*proj** 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:

```
Go to CMake tree root (not necessary, may be relative):
cd path/to/Root/of/CMake/tree
Generate the "Makefile"
cmake -S . -B build
Launch compilation
cmake --build 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.



## 6.1 DataConverter

Manage conversion of data into CMake

**class** `cmake_converter.data_converter.DataConverter`

Bases: `object`

Base class for converters

**static** `collect_data` (*context*)

Collect data for converter.

**convert\_project** (*context*, *xml\_project\_path*, *cmake\_lists\_destination\_path*)

Method template for data collecting and writing

**static** `copy_cmake_utils` (*cmake\_lists\_path*)

Copy necessary util files into CMake folder

**do\_conversion** (*project\_context*, *input\_data\_for\_converter*)

Executes conversion with given projects input data

**merge\_data\_settings** (*context*)

Merge common settings found among configuration settings (reduce copy-paste)

**Parameters** `context` –

**Returns**

**run\_conversion** (*subdirectory\_targets\_data*)

Routine that converts projects located at the same directory

**verify\_data** (*context*)

Verify procedure after gathering information from source project

**static** `write_data` (*context*, *cmake\_file*)

Write data defined in converter.

**Parameters**

- **context** (*Context*) – converter context
- **cmake\_file** (*\_io.TextIOWrapper*) – CMakeLists IO wrapper

## 6.2 Context object descriptor

**class** `cmake_converter.context.Context`

Bases: `object`

Converter context

**clone** ()

Deep clone of Context

**Returns**

**static** `get_project_initialization_dict` ()

Get initializer functors mapped to path keys

**init** (*source\_project\_path*, *cmake\_lists\_destination\_dir*)

Initialize instance of Context with Initializer

**Parameters**

- **source\_project\_path** –
- **cmake\_lists\_destination\_dir** –

**Returns**

**set\_cmake\_lists\_path** (*cmake\_lists*)

Set CMakeLists.txt path in context, for given project

**Parameters** **cmake\_lists** (*str*) – path of CMakeLists related to project name

## 6.3 Data Files

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

`cmake_converter.data_files.get_cmake_lists` (*context*, *cmake\_path=None*, *open\_type='w'*)

Create CMakeLists.txt file in wanted “cmake\_path”

**Parameters**

- **context** (*Context*) – the context of converter
- **cmake\_path** (*str*) – path where CMakeLists.txt should be open
- **open\_type** (*str*) – type that CMakeLists.txt should be opened

**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** (*tuple[str, str]*) – wanted target: debug | release

**Returns** wanted ItemDefinitionGroup namespace

**Return type** `str`

`cmake_converter.data_files.get_propertygroup(target_platform, attributes=)`  
 Return “property\_groups” value for wanted platform and target

**Parameters**

- **target\_platform** (*tuple[str, str]*) – wanted target: debug | release
- **attributes** (*str*) – attributes to add to namespace

**Returns** “property\_groups” value

**Return type** *str*

`cmake_converter.data_files.get_vcxproj_data(context, vs_project)`  
 Return xml data from “vcxproj” file

**Parameters**

- **context** (*Context*) – the context of converter
- **vs\_project** (*str*) – the vcxproj file

**Returns** dict with VS Project data

**Return type** *dict*

`cmake_converter.data_files.get_xml_data(context, xml_file)`  
 Return xml data from “xml” file

**Parameters**

- **context** (*Context*) – the context of converter
- **xml\_file** (*str*) – the xml file

**Returns** dict with VS Project data

**Return type** *dict*

`cmake_converter.data_files.search_file_path(context, xml_file)`  
 Util function for checking file in path.

## 6.4 Dependencies

Manage directories and libraries of project dependencies

**class** `cmake_converter.dependencies.Dependencies`

Bases: *object*

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

**static** `get_dependency_target_name(context, vs_project)`

Return dependency target name

**Parameters**

- **context** (*Context*) – the context of converter
- **vs\_project** (*str*) – path to “vcxproj” file

**Returns** target name

**Return type** *str*

**static** `set_additional_include_directories(aid_text, setting, context)`

Return additional include directories of given context

**Parameters**

- **aid\_text** (*str*) – path to sources
- **setting** (*str*) – current setting (Debug|x64, Release|Win32,...)
- **context** (*Context*) – current context

**Returns** include directories of context, separated by semicolons

**Return type** *str*

**static set\_target\_additional\_dependencies\_impl** (*context, dependencies\_text, splitter*)

Implementation of Handler for additional link dependencies

## 6.5 Flags

Manage compilation flags of project

**class** `cmake_converter.flags.Flags`

Bases: `object`

Class who manage flags of projects

**static get\_no\_default\_lib\_link\_flags** (*flag\_value*)

Helper to get list of /NODEFAULTLIB flags

## 6.6 ProjectFiles

Manages the recovery of project files

**class** `cmake_converter.project_files.ProjectFiles`

Bases: `object`

Class that collects and store project files

**add\_file\_from\_node** (*context, \*\*kwargs*)

Adds file into source group and creates file context using into from xml node

**apply\_files\_to\_context** (*context*)

Analyzes collected set of files and initializes necessary variables

**find\_cmake\_target\_languages** (*context*)

Add CMake Project

**include\_directive\_case\_check** (*context, file\_path\_name, file\_lists\_for\_include\_paths*)

Dummy to fix crash

**init\_file\_lists\_for\_include\_paths** (*context*)

For include directive case ad path checking. Works only with vfproj. :param context: :return:

## 6.7 ProjectVariables

Manage creation of CMake variables that will be used during compilation

**class** `cmake_converter.project_variables.ProjectVariables`

Bases: `object`

Class that manages project variables

**static** `set_output_dir_impl (context, output_node_text)`

**Parameters**

- `context` –
- `output_node_text` –

**Returns**

**static** `set_output_file_impl (context, output_file_node_text)`

Common routine for evaluating path and name of output file

**static** `set_path_and_name_from_node (context, node_name, file_path, path_property, name_property)`

Common routine for evaluating path and name from node text

**static** `set_target_name (context, target_name_value)`

Evaluates target name and sets it into project context

## 6.8 Utils

Utils manage function needed by converter

**class** `cmake_converter.utils.Utils`

Bases: `object`

Basic Class for holding util functions needed by converter

**init\_context\_current\_setting (context)**

Define settings of converter.

**Parameters** `context` (`Context`) – converter context

**static** `lists_of_settings_to_merge ()`

Lists of keys of settings at context that will be merged

`cmake_converter.utils.check_for_relative_in_path (context, path, remove_relative=True)`

Return path by adding CMake variable or current path prefix, to remove relative

**Parameters**

- `context` (`Context`) – the context of converter
- `path` (`str`) – original path
- `remove_relative` (`bool`) – flag

**Returns** formatted path without relative

**Return type** `str`

`cmake_converter.utils.cleaning_output (context, output)`

Clean Output string by remove VS Project Variables

**Parameters**

- `context` (`Context`) – the context of converter
- `output` (`str`) – Output to clean

**Returns** clean output

**Return type** `str`

`cmake_converter.utils.escape_string` (*context*, *wrong\_chars\_regex*, *input\_str*)

Removes wrong chars from input string

`cmake_converter.utils.get_actual_filename` (*context*, *name*)

Return actual filename from given name if file is found, else return None

**Parameters**

- **context** (`Context`) – the context of converter
- **name** (`str`) – name of file

**Returns** None | `str`

**Return type** None | `str`

`cmake_converter.utils.get_dir_name_with_vars` (*context*, *path*)

Tries to split directory and filename from given path

`cmake_converter.utils.get_global_project_name_from_vcproj_file` (*vcproj*)

Return global project name from “.vcproj” file

**Parameters** **vcproj** (`dict`) – vcproj data

**Returns** project name

**Return type** `str`

`cmake_converter.utils.get_mapped_architectures` (*sln\_setting\_2\_project\_setting*, *arch*)

Get all projects architectures that mapped onto given solution one

`cmake_converter.utils.get_mount_point` (*path*)

Returns mount point of given path

`cmake_converter.utils.init_colorama` ()

Initialization of colorful console output

`cmake_converter.utils.insensitive_glob` (*path*)

Searches given path case insensitive

`cmake_converter.utils.is_settings_has_data` (*sln\_configurations\_map*, *settings*, *settings\_key*, *sln\_arch=None*, *conf=None*)

Checker of available settings in context

`cmake_converter.utils.make_cmake_configuration` (*context*, *sln\_configuration*)

Tries to make cmake configuration name from sln\_configuration

`cmake_converter.utils.make_cmake_literal` (*context*, *input\_str*)

Tries to make cmake literal from input string

`cmake_converter.utils.make_os_specific_shell_path` (*output*)

Tries to make path readable with CMake

`cmake_converter.utils.message` (*context*, *text*, *status*)

Displays a message while the script is running

**Parameters**

- **context** (`Context`) – the context of converter
- **text** (`str`) – content of the message
- **status** (`str`) – level of the message (change color)



`cmake_converter.utils.normalize_path` (*context*, *working\_path*, *path\_to\_normalize*, *remove\_relative=True*, *unix\_slash=True*)

Normalize path from working path

**Parameters**

- **context** (*Context*) – the context of converter
- **working\_path** (*str*) – current working path
- **path\_to\_normalize** (*str*) – path to be normalized
- **remove\_relative** (*bool*) – remove relative from path flag
- **unix\_slash** (*bool*) – apply UNIX slash

**Returns** normalized path

**Return type** *str*

`cmake_converter.utils.prepare_build_event_cmd_line_for_cmake` (*context*, *build\_event*)

Tries to fit build event command to be compliant CMake language

`cmake_converter.utils.replace_vs_var_with_cmake_var` (*context*, *var*)

Translate Visual studio variable into CMake variable

`cmake_converter.utils.replace_vs_vars_with_cmake_vars` (*context*, *output*)

Translates variables at given string to corresponding CMake ones

`cmake_converter.utils.resolve_path_variables_of_vs` (*context*, *path\_with\_vars*)

Evaluates paths with visual studio variables

`cmake_converter.utils.set_native_slash` (*raw\_path*)

Set native slash

**Parameters** **raw\_path** (*str*) – any style path

**Returns** unix style path

**Return type** *str*

`cmake_converter.utils.set_unix_slash` (*win\_path*)

Set windows path to unix style path

**Parameters** **win\_path** (*str*) – windows style path

**Returns** unix style path

**Return type** *str*

`cmake_converter.utils.take_name_from_list_case_ignore` (*context*, *search\_list*, *name\_to\_search*)

Return real name of name to search

**Parameters**

- **context** (*Context*) – the context of converter
- **search\_list** (*list*) – list to make research
- **name\_to\_search** (*str*) – name to search in list

**Returns** real name

**Return type** *str*



# CHAPTER 7

---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



### C

- `cmake_converter.context`, 16
- `cmake_converter.data_converter`, 15
- `cmake_converter.data_files`, 16
- `cmake_converter.dependencies`, 17
- `cmake_converter.flags`, 18
- `cmake_converter.main`, 7
- `cmake_converter.project_files`, 18
- `cmake_converter.project_variables`, 18
- `cmake_converter.utils`, 19



## A

`add_file_from_node()`  
(*cmake\_converter.project\_files.ProjectFiles*  
*method*), 18

`apply_files_to_context()`  
(*cmake\_converter.project\_files.ProjectFiles*  
*method*), 18

## C

`check_for_relative_in_path()` (*in module*  
*cmake\_converter.utils*), 19

`cleaning_output()` (*in module*  
*cmake\_converter.utils*), 19

`clone()` (*cmake\_converter.context.Context* *method*), 16

`cmake_converter.context` (*module*), 16

`cmake_converter.data_converter` (*module*),  
15

`cmake_converter.data_files` (*module*), 16

`cmake_converter.dependencies` (*module*), 17

`cmake_converter.flags` (*module*), 18

`cmake_converter.main` (*module*), 7

`cmake_converter.project_files` (*module*), 18

`cmake_converter.project_variables` (*mod-  
ule*), 18

`cmake_converter.utils` (*module*), 19

`collect_data()` (*cmake\_converter.data\_converter.DataConverter*  
*static method*), 15

`Context` (*class in cmake\_converter.context*), 16

`convert_project()`  
(*cmake\_converter.data\_converter.DataConverter*  
*method*), 15

`copy_cmake_utils()`  
(*cmake\_converter.data\_converter.DataConverter*  
*static method*), 15

## D

`DataConverter` (*class in*  
*cmake\_converter.data\_converter*), 15

`Dependencies` (*class in*  
*cmake\_converter.dependencies*), 17

`do_conversion()` (*cmake\_converter.data\_converter.DataConverter*  
*method*), 15

## E

`escape_string()` (*in module* *cmake\_converter.utils*),  
20

## F

`find_cmake_target_languages()`  
(*cmake\_converter.project\_files.ProjectFiles*  
*method*), 18

`Flags` (*class in* *cmake\_converter.flags*), 18

## G

`get_actual_filename()` (*in module*  
*cmake\_converter.utils*), 20

`get_cmake_lists()` (*in module*  
*cmake\_converter.data\_files*), 16

`get_definitiongroup()` (*in module*  
*cmake\_converter.data\_files*), 16

`get_dependency_target_name()`  
(*cmake\_converter.dependencies.Dependencies*  
*static method*), 17

`get_converter_name_with_vars()` (*in module*  
*cmake\_converter.utils*), 20

`get_global_project_name_from_vcxproj_file()`  
(*in module* *cmake\_converter.utils*), 20

`get_mapped_architectures()` (*in module*  
*cmake\_converter.utils*), 20

`get_mount_point()` (*in module*  
*cmake\_converter.utils*), 20

`get_no_default_lib_link_flags()`  
(*cmake\_converter.flags.Flags* *static method*),  
18

`get_project_initialization_dict()`  
(*cmake\_converter.context.Context* *static*  
*method*), 16

get\_propertygroup() (in module *cmake\_converter.data\_files*), 16  
 get\_vcxproj\_data() (in module *cmake\_converter.data\_files*), 17  
 get\_xml\_data() (in module *cmake\_converter.data\_files*), 17

## I

include\_directive\_case\_check() (*cmake\_converter.project\_files.ProjectFiles* method), 18  
 init() (*cmake\_converter.context.Context* method), 16  
 init\_colorama() (in module *cmake\_converter.utils*), 20  
 init\_context\_current\_setting() (*cmake\_converter.utils.Utils* method), 19  
 init\_file\_lists\_for\_include\_paths() (*cmake\_converter.project\_files.ProjectFiles* method), 18  
 insensitive\_glob() (in module *cmake\_converter.utils*), 20  
 is\_settings\_has\_data() (in module *cmake\_converter.utils*), 20

## L

lists\_of\_settings\_to\_merge() (*cmake\_converter.utils.Utils* static method), 19

## M

make\_cmake\_configuration() (in module *cmake\_converter.utils*), 20  
 make\_cmake\_literal() (in module *cmake\_converter.utils*), 20  
 make\_os\_specific\_shell\_path() (in module *cmake\_converter.utils*), 20  
 merge\_data\_settings() (*cmake\_converter.data\_converter.DataConverter* method), 15  
 message() (in module *cmake\_converter.utils*), 20

## N

normalize\_path() (in module *cmake\_converter.utils*), 20

## P

prepare\_build\_event\_cmd\_line\_for\_cmake() (in module *cmake\_converter.utils*), 21  
 ProjectFiles (class in *cmake\_converter.project\_files*), 18  
 ProjectVariables (class in *cmake\_converter.project\_variables*), 18

## R

replace\_vs\_var\_with\_cmake\_var() (in module *cmake\_converter.utils*), 21  
 replace\_vs\_vars\_with\_cmake\_vars() (in module *cmake\_converter.utils*), 21  
 resolve\_path\_variables\_of\_vs() (in module *cmake\_converter.utils*), 21  
 run\_conversion() (*cmake\_converter.data\_converter.DataConverter* method), 15

## S

search\_file\_path() (in module *cmake\_converter.data\_files*), 17  
 set\_additional\_include\_directories() (*cmake\_converter.dependencies.Dependencies* static method), 17  
 set\_cmake\_lists\_path() (*cmake\_converter.context.Context* method), 16  
 set\_native\_slash() (in module *cmake\_converter.utils*), 21  
 set\_output\_dir\_impl() (*cmake\_converter.project\_variables.ProjectVariables* static method), 19  
 set\_output\_file\_impl() (*cmake\_converter.project\_variables.ProjectVariables* static method), 19  
 set\_path\_and\_name\_from\_node() (*cmake\_converter.project\_variables.ProjectVariables* static method), 19  
 set\_target\_additional\_dependencies\_impl() (*cmake\_converter.dependencies.Dependencies* static method), 18  
 set\_target\_name() (*cmake\_converter.project\_variables.ProjectVariables* static method), 19  
 set\_unix\_slash() (in module *cmake\_converter.utils*), 21

## T

take\_name\_from\_list\_case\_ignore() (in module *cmake\_converter.utils*), 21

## U

Utils (class in *cmake\_converter.utils*), 19

## V

verify\_data() (*cmake\_converter.data\_converter.DataConverter* method), 15

## W

write\_data() (*cmake\_converter.data\_converter.DataConverter* static method), 15