makeprojects Documentation

Release 0.12.1

Rebecca Ann Heineman <becky@burgerbecky.com>

Feb 05, 2022

CONTENTS

The `makeprojects` module makes it easy to autogenerate project files for several popular Integrated Development Enviroments (IDEs)

- Documentation is found at https://makeprojects.readthedocs.io
- Doxygen generated documentation is found at https://makeprojects.readthedocs.io/en/latest/doxygen
- Python Packing Index (PyPI): https://pypi.python.org/pypi/makeprojects
- Source code and issue tracker: https://github.com/burgerbecky/makeprojects

CHAPTER

ONE

COMPATIBILITY

- Python 2.7.1 or higher
- Python 3.4 or higher

CHAPTER

TWO

INSTALLATION

Type in `pip install -U makeprojects`. Some platforms may require the `sudo` prefix.

CHAPTER

THREE

BUGS

If you find a bug, issue or have a feature request, please submit a bug report by emailing becky@burgerbecky.com and mention python version, integer size (32 bit or 64 bit) and what platform was used (Windows / Mac OSX / Linux).

CHAPTER

FOUR

TABLE OF CONTENTS

4.1 Constants

4.1.1 Setup strings

These strings are used for version control and setup.py for distribution.

__numversion__

makeprojects.__numversion__ = (0, 12, 1)
Current version of the library as a numeric tuple.

__version__

makeprojects.__version__ = '.'.join([str(num) for num in __numversion__])
Current version of the library.

__author__

makeprojects.__author__ = 'Rebecca Ann Heineman <becky@burgerbecky.com>'
Author's name.

__title__

makeprojects.__title__ = 'makeprojects'
Name of the module.

__summary__

makeprojects.__summary__ = 'IDE project generator for Visual Studio, XCode, etc...'
Summary of the module's use.

__uri__

makeprojects.__uri__ = 'http://makeprojects.readthedocs.io'
Home page.

__email__

makeprojects.__email__ = 'becky@burgerbecky.com'
Email address for bug reports.

__license__

makeprojects.__license__ = 'MIT License'
Type of license used for distribution.

__copyright__

makeprojects.__copyright__ = 'Copyright 2013-2022 Rebecca Ann Heineman'
Copyright owner.

4.1.2 Internal constants

Constants used internally by this package.

_XCODEPROJ_MATCH

makeprojects._XCODEPROJ_MATCH
Match *.xcodeproj.

4.1.3 Internal tables

enums._FILETYPES_LOOKUP

makeprojects.enums._FILETYPES_LOOKUP
Dictionary of default file extensions and mapped types.

When the directory is scanned for input files, the files will be tested against this list with a forced lowercase filename and determine the type of compiler to assign to an input file

This list can be appended or modified to allow other file types to be processed

See also:

makeprojects.enums.FileTypes.lookup()

enums._FILETYPES_READABLE

makeprojects.enums.**_FILETYPES_READABLE**
List of human readable strings.

Dictionary to map *FileTypes* enumerations into an human readable string

See also:

makeprojects.enums.FileTypes.__repr__()

enums._IDETYPES_CODES

makeprojects.enums.**_IDETYPES_CODES**
List of IDE short codes.

Dictionary to map *IDETypes* enumerations into a three letter code to append to a project filename

See also:

makeprojects.enums.IDETypes.get_short_code()

enums._IDETYPES_READABLE

makeprojects.enums.**_IDETYPES_READABLE**
List of human readable strings.

Dictionary to map *IDETypes* enumerations into an human readable string

See also:

makeprojects.enums.IDETypes.__repr__()

enums._PLATFORMTYPES_CODES

makeprojects.enums.**_PLATFORMTYPES_CODES**
List of platform short codes.

Dictionary to map *PlatformTypes* enumerations into a three or six letter code to append to a project filename

See also:

makeprojects.enums.PlatformTypes.get_short_code

enums._PLATFORMTYPES_EXPANDED

makeprojects.enums.**_PLATFORMTYPES_EXPANDED**
List of platforms that expand to multiple targets.

Dictionary to map generic *PlatformTypes* enumerations into lists.

See also:

makeprojects.enums.PlatformTypes.get_expanded

enums._PLATFORMTYPES_READABLE

makeprojects.enums.**_PLATFORMTYPES_READABLE**
List of human readable strings.

Dictionary to map *PlatformTypes* enumerations into an human readable string

See also:

makeprojects.enums.PlatformTypes.__repr__

enums._PLATFORMTYPES_VS

makeprojects.enums.**_PLATFORMTYPES_VS**
List of Visual Studio platform codes.

Visual Studio uses specific codes for tool chains used for video game consoles or CPUs

See also:

makeprojects.enums.PlatformTypes.get_vs_platform

enums._PROJECTTYPES_READABLE

makeprojects.enums.**_PROJECTTYPES_READABLE**
List of human readable strings.

Dictionary to map *ProjectTypes* enumerations into an human readable string

See also:

makeprojects.enums.ProjectTypes.__repr__()

4.1.4 Folder locations

config.BUILD_RULES_PY

makeprojects.config.**BUILD_RULES_PY = 'build_rules.py'**
build_rules.py file to detect secondly

config._BUILD_RULES_VAR

makeprojects.config.**_BUILD_RULES_VAR**
BUILD_RULES_PY location environment variable.

config.USER_HOME

makeprojects.config.**USER_HOME = os.path.expanduser('~')**
Location of the user's home directory.

config.PROJECTS_HOME

makeprojects.config.**PROJECTS_HOME = os.environ['MAKE_PROJECTS_HOME']**
Location of makeprojects home directory if redirected.

config.DEFAULT_BUILD_RULES

makeprojects.config.**DEFAULT_BUILD_RULES = find_default_build_rules()**
Full pathname of the configuration file.

4.1.5 Build Constants

buildme.BUILD_LIST

makeprojects.buildme.**BUILD_LIST = ((1, 'prebuild'),(40, 'build'),(99, 'postbuild'))**
Default build_rules.py command list, priority / entrypoint.

buildme.CODEWARRIOR_ERRORS

makeprojects.buildme.**CODEWARRIOR_ERRORS = (None,'error opening file','project not open','IDE is already building','invalid target name (for /t flag)','error changing current target','error removing objects','buildwas cancelled','buildfailed','process aborted','error importing project','error executing debug script','attempted use of /d together with /b and/or /r')**
Error code messages from Codewarrior.

buildme._CW_SUPPORTED_LINKERS

makeprojects.buildme.**_CW_SUPPORTED_LINKERS**
List of supported Codewarrior Linkers.

buildme._VS_VERSION_YEARS

makeprojects.buildme.**_VS_VERSION_YEARS**
Lookup for Visual Studio year in SLN file.

buildme._VS_OLD_VERSION_YEARS

makeprojects.buildme.**_VS_OLD_VERSION_YEARS**
Lookup for Visual Studio year in SLN file pre-2012.

buildme._VS_SDK_ENV_VARIABLE

makeprojects.buildme.**_VS_SDK_ENV_VARIABLE**
Lookup for Visual Studio SDK detector.

4.2 Classes

4.2.1 Enumerations

enums.FileTypes

makeprojects.enums.FileTypes : public IntEnum
Enumeration of supported file types for project input.

Each file that is to be added to a project has specific build rules, this enumeration helps determine which compiler to invoke to build the file if a build step is necessary.

Public Functions

__repr__(*self*)
Convert the enumeration into a human readable file description.

See also:

makeprojects.enums._FILETYPES_READABLE

Returns Human readable string or None if the enumeration is invalid

Public Static Functions

lookup(*test_name*)
Look up a file name extension and return the type.

Parse the filename extension and match it to a table of known extensions and return the enumeration for the file type. The test is case insensitive.

See also:

makeprojects.enums._FILETYPES_LOOKUP

Parameters **test_name** – Filename to test

Returns A *FileTypes* member or None on failure

Public Static Attributes

user = 0
User file type (Unknown)

generic = 1
Non compiling file type.

c = 2
Compile as C.

cpp = 3
Compile as C++.

h = 4
C/C++ header.

m = 5
Objective-C.

xml = 6
XML text file.

rc = 7
Windows resource file.

r = 8
Mac OS resource file.

hlsl = 9
HLSL DirectX Shader.

glsl = 10
GLSL OpenGL Shader.

x360sl = 11
Xbox 360 DirectX Shader.

vitacg = 12
Playstation Vita CG Shader.

frameworks = 13
Mac OSX Framework.

library = 14
Static libary.

object = 15
Object code.

exe = 16
Executable file.

xcconfig = 17
XCode configuration file.

x86 = 18
X86 assembly source.

x64 = 19
X64 assembly source.

a65 = 20
6502/65812 assembly source

ppc = 21
PowerPC assembly source.

a68 = 22
680x0 assembly source

image = 23
Image files.

ico = 24
Windows icon files.

icns = 25
MacOSX icon files.

appxmanifest = 26
Windows AppXManifest files.

enums.IDETypes

makeprojects.enums.IDETypes : public IntEnum

Enumeration of supported IDEs.

All supported IDEs and makefile formats are enumerated here.

Public Functions

get_short_code(*self*)

Create the ide code from the ide type.

Return the three letter code that determines the specfic IDE version that the project file is meant for.

See also:

makeprojects.enums._IDETYPES_CODES

Returns Three letter code specific to the IDE version or None.

is_visual_studio(*self*)

Determine if the IDE is Microsoft Visual Studio.

Returns True if the platform is Microsoft Visual Studio.

is_xcode(*self*)

Determine if the IDE is Apple XCode.

Returns True if the platform is Apple XCode.

is_codewarrior(*self*)

Determine if the IDE is Metrowerks / Freescale Codewarrior.

Returns True if the platform is Metrowerks / Freescale Codewarrior.

__repr__(*self*)

Convert the enumeration into a human readable file description.

See also:

makeprojects.enums._IDETYPES_READABLE

Returns Human readable string or None if the enumeration is invalid

Public Static Functions

lookup(*ide_name*)

Look up a *IDETypes* based on name.

For maximum compatiblity, the name will be scanned from several look up tables to attempt to cover all premutations of an input string.

Note: String comparisons are case insensitive.

Parameters **ide_name** – Platform string to test.

Returns A *IDETypes* member or None on failure.

default()
Determine the default *IDETypes* from the currently running platform.

Public Static Attributes

vs2003 = 0
Visual studio 2003.

vs2005 = 1
Visual studio 2005.

vs2008 = 2
Visual studio 2008.

vs2010 = 3
Visual studio 2010.

vs2012 = 4
Visual studio 2012.

vs2013 = 5
Visual studio 2013.

vs2015 = 6
Visual studio 2015.

vs2017 = 7
Visual studio 2017.

vs2019 = 8
Visual studio 2019.

vs2022 = 9
Visual studio 2022.

watcom = 10
Open Watcom 1.9 or later.

codewarrior50 = 11
Metrowerks Codewarrior 9 / 5.0 (Windows/Mac OS)

codewarrior58 = 12
Metrowerks Codewarrior 10 / 5.8 (Mac OS Carbon)

codewarrior59 = 13
Freescale Codewarrior 5.9 (Nintendo DSi)

xcode3 = 14
XCode 3 (PowerPC 3.1.4 is the target version)

xcode4 = 15
XCode 4.

xcode5 = 16
XCode 5.

xcode6 = 17
XCode 6.

xcode7 = 18
XCode 7.

xcode8 = 19
XCode 8.

xcode9 = 20
XCode 9.

xcode10 = 21
XCode 10.

xcode11 = 22
XCode 11.

codeblocks = 23
Codeblocks.

nmake = 24
nmake

make = 25
make

bazel = 26
bazel

mpw = 27
MPW.

enums.PlatformTypes

makeprojects.enums.PlatformTypes : public IntEnum

Enumeration of supported target platforms.

All supported tool chains for specific platforms are enumerated here.

Public Functions

get_short_code(*self*)

Convert the enumeration to a 3 letter code for filename suffix.

Create a three letter code for the target platform for the final filename for the project. For platforms that support multiple CPU architectures, the code will be six letters long with the CPU appended to the three letter platform code.

See also:

makeprojects.enums._PLATFORMTYPES_CODES

Returns A three or six letter code for the platform.

is_windows(*self*)

Determine if the platform is windows.

Returns True if the platform is for Microsoft windows.

is_xbox(*self*)

Determine if the platform is a version of the Xbox.

Returns True if the platform is for Xbox, Xbox 360, or Xbox ONE.

is_macosx(*self*)

Determine if the platform is macOS.

Returns True if the platform is Apple macOS.

is_ios(*self*)

Determine if the platform is iOS.

Returns True if the platform is Apple iOS.

is_macos(*self*)

Determine if the platform is MacOS classic or Carbon.

Returns True if Apple MacOS 1.0 through 9.2.2 or the Carbon API.

is_macos_carbon(*self*)

Determine if the platform is MacOS Carbon.

Returns True if the platform is Apple MacOS Carbon API.

is_macos_classic(*self*)

Determine if the platform is MacOS classic (MacOS 1.0 to 9.2.2).

Returns True if the platform is Apple MacOS 1.0 through 9.2.2.

is_msdos(*self*)

Determine if the platform is MSDos.

Returns True if the platform is MSDos

is_android(*self*)
Determine if the platform is Android.

Returns True if the platform is Android

is_switch(*self*)
Determine if the platform is Nintendo Switch.

Returns True if the platform is Nintendo Switch

get_platform_folder(*self*)
Return the name of a folder that would hold platform specific files.

match(*self*, *second*)
Test if two platform types are a match.

If two platform types are similar, this function will return True. An example would be a windows 32 bit and a windows 64 bit platform would match.

Returns True if the types are compatible.

get_vs_platform(*self*)
Create the platform codes from the platform type for Visual Studio.

Visual Studio uses specific codes for tool chains used for video game consoles or CPUs. This function returns a list of codes needed to support the requested platform.

See also:

makeprojects.enums._PLATFORMTYPES_VS

Returns A list of Visual Studio platforms for target.

get_expanded(*self*)
Return a list of platforms from a platform that is a group.

is_expandable(*self*)
Return True if the platform defines other platforms.

__repr__(*self*)
Convert the enumeration into a human readable file description.

See also:

makeprojects.enums._PLATFORMTYPES_READABLE

Returns Human readable string or None if the enumeration is invalid

Public Static Functions

lookup(*platform_name*)
Look up a PlatformType based on name.

For maximum compatiblity, the name will be scanned from several look up tables to attempt to cover all premutations of an input string.

See also:

makeprojects.enums._PLATFORMTYPES_READABLE

Note: String comparisons are case insensitive.

Parameters **platform_name** – Platform string to test.

Returns A *PlatformTypes* member or None on failure.

default()
Determine the *PlatformTypes* from the currently running platform.

Public Static Attributes

windows = 0
Windows 32 and 64 bit Intel and arm64.

windowsintel = 1
Windows 32 and 64 bit Intel.

windowsarm = 2
Windows 32 and 64 bit arm.

win32 = 3
Windows 32 bit intel only.

win64 = 4
Window 64 bit intel only.

winarm32 = 5
Windows 32 bit arm only.

winarm64 = 6
Windows 64 bit arm only.

winitanium = 7
Windows 64 bit itanium only.

macosx = 8
Mac OSX, all CPUs.

macosxppc32 = 9
Mac OSX PowerPC 32 bit only.

macosxppc64 = 10
Mac OSX PowerPC 64 bit only.

macosxintel32 = 11
Mac OSX Intel 32 bit only.

macosxintel64 = 12
Mac OSX Intel 64 bit only.

macos9 = 13
Mac OS 9, all CPUs.

macos968k = 14
Mac OS 9 680x0 only.

macos9ppc = 15
Mac OS 9 PowerPC 32 bit only.

maccarbon = 16
Mac OS Carbon, all CPUs.

maccarbon68k = 17
Mac OS Carbon 680x0 only (CFM)

maccarbonppc = 18
Mac OS Carbon PowerPC 32 bit only.

ios = 19
iOS, all CPUs

ios32 = 20
iOS 32 bit ARM only

ios64 = 21
iOS 64 bit ARM only

iosemu = 22
iOS emulator, all CPUs

iosemu32 = 23
iOS emulator 32 bit Intel only

iosemu64 = 24
iOS emulator 64 bit Intel only

xbox = 25
Microsoft Xbox classic.

xbox360 = 26
Microsoft Xbox 360.

xboxone = 27
Microsoft Xbox ONE.

ps1 = 28
Sony PS1.

ps2 = 29
Sony PS2.

ps3 = 30
Sony PS3.

ps4 = 31
Sony PS4.

psp = 32
Sony Playstation portable.

vita = 33
Sony Playstation VITA.

wii = 34
Nintendi Wii.

wiiu = 35
Nintendo WiiU.

switch = 36
Nintendo Switch.

switch32 = 37
Nintendo Switch 32 bit only.

switch64 = 38
Nintendo Switch 64 bit only.

dsi = 39
Nintendo 3DS.

ds = 40
Nintendo DS.

android = 41
Generic Android.

shield = 42
nVidia SHIELD

amico = 43
Intellivision Amico.

ouya = 44
Ouya (Now Razor)

tegra = 45
Android Tegra.

androidarm32 = 46
Android Arm32.

androidarm64 = 47
Android Arm64.

androidintel32 = 48
Android Intel x32.

androidintel64 = 49
Android Intel / AMD 64.

linux = 50
Generic Linux.

msdos = 51
MSDOS.

msdos4gw = 52
MSDOS Dos4GW.

msdosx32 = 53
MSDOS DosX32.

beos = 54
BeOS.

iigs = 55
Apple IIgs.

enums.ProjectTypes

makeprojects.enums.ProjectTypes : public IntEnum
Enumeration of supported project types.

Each configuration can build a specific type of file, this enumeration lists out the types of files that can be built.

Public Functions

is_library(*self*)
Determine if the project is a library.

Returns True if the project is a static or dynamic library.

__repr__(*self*)
Convert the enumeration into a human readable file description.

See also:

makeprojects.enums._PROJECTTYPES_READABLE

Returns Human readable string or None if the enumeration is invalid

Public Static Functions

lookup(*project_type_name*)
Look up a *ProjectTypes* based on name.

For maximum compatiblity, the name will be scanned from several look up tables to attempt to cover all premutations of an input string.

Note: String comparisons are case insensitive.

Parameters **project_type_name** – Project type string to test.

Returns A *ProjectTypes* member or None on failure.

default()
Determine the *ProjectTypes* default.

Public Static Attributes

library = 0
Code library.

tool = 1
Command line tool.

app = 2
Application.

screensaver = 3
Screen saver.

sharedlibrary = 4
Shared library (DLL)

empty = 5
Empty project.

4.2.2 Project Classes

core.SourceFile

class makeprojects.core.**SourceFile**
Object for each input file to insert to a solution.

For every file that could be included into a project file one of these objects is created and attached to a *Project* object for processing.

Public Functions

__init__(*self*, *relative_pathname*, *working_directory*, *filetype*)
Default constructor.

See also:

enums.FileTypes

Parameters

- **relative_pathname** – Filename of the input file (relative to the root)
- **working_directory** – Pathname of the root directory
- **filetype** – Compiler to apply

get_group_name(*self*)
Get the group location for this source file.

To determine if the file should be in a sub group in the project, scan the filename to find if it's a base filename or part of a directory. If it's a basename, return an empty string. If it's in a folder, remove any `..\` prefixes and `.\` prefixes and return the filename with the basename removed.

Returns The group name string with \ delimiters.

get_abspath(*self*)
Return the full pathname of the file entry.

Returns Absolute pathname for the file.

__repr__(*self*)
Convert the file record into a human readable file description.

Returns Human readable string.

Public Members

relative_pathname
File base name with extension using windows style slashes.

working_directory
Directory the file is relative to.

type
File type enumeration, see: *enums.FileTypes*.

core.Configuration

makeprojects.core.Configuration : public makeprojects.core.Attributes
Object for containing attributes specific to a build configuration.

This object contains all of the items needed to create a specific configuration of a project.

Valid attributes:

- `name` name of the configuration
- `short_code` Short code suffix for configuration name
- `platform` Platform to build for
- `project_type` Type of binary to generate
- `exclude_from_build_list` List of files to exclude from this configuration
- `include_folders_list` List of directories for headers
- `library_folders_list` List of directories for libraries
- `libraries_list` List of libraries to include
- `frameworks_list` List of frameworks to include (macOS/iOS)
- `define_list` List of defines for compilation
- `debug` True if debugging defaults are enabled
- `optimization` 0-4 level of optimization
- `link_time_code_generation` Enable link time code genration

See also:

Project

See also:

Solution

Public Functions

__init__(*self*, *args*, *kargs*)
Init defaults.

Parameters

- **args** – name and setting_name for get_configuration_settings(_
- **kargs** – List of defaults.

ide(*self*)
Return the preferred IDE.

short_code(*self*)
Return the short code.

short_code(*self*, *value*)
Set the filename suffix.

Parameters

- **self** – The 'this' reference.
- **value** – New short code

parse_attributes(*self*, *build_rules_list*, *working_directory*)
Initialize the default attributes.

Parameters

- **build_rules_list** – List to append a valid build_rules file instance.
- **working_directory** – Full path name of the build_rules.py to load.

get_suffix(*self*, *force_short=False*)
Return the proposed suffix.

Each configuration can generate a seperate binary and if they are stored in the same folder, a suffix is appened to make the filename unique.

Parameters **force_short** – True to force the platform code to 3 characters

Returns A suffix of the IDE, Platform and *Configuration* short codes.

__repr__(*self*)
Convert the configuration record into a human readable description.

Returns Human readable string.

Public Members

project
Project this *Configuration* is attached to.

Public Static Attributes

source_folders_list = NoneProperty('_source_folders_list')
Don't allow source folders in configuration.

source_files_list = StringListProperty('_source_files_list')
Don't allow source files to be added in a configuration.

vs_props = NoneProperty('_vs_props')
Don't allow Visual Studio props files.

vs_targets = NoneProperty('_vs_targets')
Don't allow Visual Studio targets files.

vs_rules = NoneProperty('_vs_rules')
Don't allow Visual Studio rules files.

core.Project

makeprojects.core.Project : public makeprojects.core.Attributes
Object for processing a project file.

This object contains all of the items needed to generate a project.

Note: On most IDEs, this is merged into one file, but Visual Studio generates a project file for each project.

Public Functions

__init__(*self*, *name=None*, *kargs*)
Set defaults.

Parameters

- **name** – Name of the project
- **kargs** – dict of arguments.

ide(*self*)
Return the preferred IDE.

add_configuration(*self*, *configuration*)
Add a configuration to the list of configurations found in this project.

Given a new *Configuration* class instance, append it to the list of configurations that this project is managing.

Exception TypeError if `configuration` is not a *Configuration*

Parameters

- **self** – The 'this' reference.
- **configuration** – Reference to an instance of a *Configuration*.

add_project(*self*, *project*)
Add a dependent project.

Exception TypeError if project is not a *Project*

Parameters **project** – *Project* to depend on.

get_project_list(*self*)
Return the project list for all projects.

Iterate over every project and sub project and return a flattened list.

Returns list of every project in the solution.

set_platforms(*self*, *platform*)
Update all configurations to a new platform.

If there are no configurations, Debug and Release will be created.

Parameters **platform** – Platform to change the configurations to.

parse_attributes(*self*, *build_rules_list*, *working_directory*)
Initialize the default attributes.

Parameters

- **build_rules_list** – List to append a valid build_rules file instance.
- **working_directory** – Full path name of the build_rules.py to load.

get_file_list(*self*, *acceptable_list*)
Obtain the list of source files.

Set up the variables `codefiles` with the list of source files found and `_source_include_list` with a list of relative to the working directory folders where the source code was found.

- `exclude_list` for wildcard matching for files to exclude
- `source_folders_list` for list of folders to search for source code
- `source_files_list` list of files to add

Parameters **acceptable_list** – List of acceptable FileTypes

__repr__(*self*)
Convert the solultion record into a human readable description.

Returns Human readable string or None if the solution is invalid

Public Members

name
Project name.

working_directory
Working directory for the project.

solution
No parent solution yet.

configuration_list
Generate the default configurations.

project_list
Initial array of *Project* records that need to be built first.

codefiles
Initial array of *SourceFile* in the solution.

file_list
Used by scan_directory.

include_list
Used by scan_directory.

platform_code
Platform code for generation.

Public Static Attributes

source_folders_list = StringListProperty('_source_folders_list')
List of directories to scan for source code.

List of folders to scan for source code.

source_files_list = StringListProperty('_source_files_list')
List of generated source files to include in the project.

List of files to add to the project.

vs_props = StringListProperty('_vs_props')
List of props files for Visual Studio.

vs_targets = StringListProperty('_vs_targets')
List of targets file for Visual Studio.

vs_rules = StringListProperty('_vs_rules')
List of rules file for Visual Studio 2005-2008.

core.Solution

makeprojects.core.Solution : public makeprojects.core.Attributes

Object for processing a solution file.

This object contains all of the items needed to create a solution.

Public Functions

__init__(*self*, *name=None*, *kargs*)

Init defaults.

Parameters

- **name** – Name of the *Solution*
- **kargs** – dict of arguments.

ide(*self*)

Return the ide type.

ide(*self*, *value*)

Set the IDE type with validation.

Parameters

- **self** – The 'this' reference.
- **value** – None or new IDE type

add_project(*self*, *project=None*, *project_type=None*)

Add a project to the list of projects found in this solution.

Given a new *Project* class instance, append it to the list of projects that this solution is managing.

Parameters

- **self** – The 'this' reference.
- **project** – Reference to an instance of a *Project*.
- **project_type** – Type of project to create.

add_tool(*self*, *project=None*)

Add a project to build a command line tool.

See also:

add_project

add_app(*self*, *project=None*)

Add a project to build an application.

See also:

add_project

add_library(*self*, *project=None*)

Add a project to build a static library.

See also:

add_project

`add_shared_library`(*self*, *project=None*)
Add a project to build a dynamic library.

See also:

add_project

`get_project_list`(*self*)
Return the project list for all sub projects.

Iterate over every sub project and return a flattened list.

Returns list of every project in the project.

`set_platforms`(*self*, *platform*)
Update all configurations to a new platform.

If there are no configurations, Debug and Release will be created.

Parameters `platform` – Platform to change the configurations to.

`generate`(*self*, *ide=None*)
Generate a project file and write it out to disk.

`__repr__`(*self*)
Convert the solultion record into a human readable description.

Returns Human readable string or None if the solution is invalid

Public Members

`name`
Solution name.

`working_directory`
Working directory for the solution.

`project_list`
Initial array of *Project* records for projects.

`ide_code`
IDE code for generation.

`platform_code`
Platform code for generation.

Public Static Attributes

source_folders_list = StringListProperty('_source_folders_list')
List of directories to scan for source code.

List of folders to scan for source code.

source_files_list = StringListProperty('_source_files_list')
List of generated source files to include in the project.

List of files to add to the projects.

vs_props = NoneProperty('_vs_props')
Don't allow Visual Studio props files.

vs_targets = NoneProperty('_vs_targets')
Don't allow Visual Studio targets files.

vs_rules = NoneProperty('_vs_rules')
Don't allow Visual Studio rules files.

perforce = BooleanProperty('_perforce')
Use perforce.

Enable perforce support.

verbose = BooleanProperty('_verbose')
Verbosity.

Enable output verbosity.

suffix_enable = BooleanProperty('_suffix_enable')
Enable the use of suffixes in creating filenames.

Enable appending suffixes to project names.

4.2.3 Build Classes

buildme.BuildError

makeprojects.buildme.BuildError : public object
Error message generated by builders.

When a builder completes, a *BuildError* class is created and appended to the `results` list for logging.

Public Functions

__init__(*self*, *error*, *filename*, *configuration=None*, *msg=None*)
Initializers for an *BuildError*.

Parameters

- **error** – Integer error code, zero if not error
- **filename** – File that generated the error
- **configuration** – If applicable, configuration that was compiled
- **msg** – Error message test, if available

__repr__(*self*)
Convert the error into a string.

Returns A full error string.

get_error_code(*self*)
Return the integer error code.

Public Members

error
Integer error code.

filename
File name that generated the error.

configuration
Build configuration.

msg
Error message.

buildme.BuildObject

makeprojects.buildme.BuildObject : public object
Object describing something to build.

When the directory is parsed, a list of BuildObjects is generated and then sorted by priority and then built.

Subclassed by makeprojects.buildme.BuildCodeBlocksFile, makeprojects.buildme.BuildCodeWarriorFile, makeprojects.buildme.BuildDoxygenFile, makeprojects.buildme.BuildMakeFile, makeprojects.buildme.BuildNinjaFile, makeprojects.buildme.BuildPythonFile, makeprojects.buildme.BuildRezFile, makeprojects.buildme.BuildSlicerFile, makeprojects.buildme.BuildVisualStudioFile, makeprojects.buildme.BuildWatcomFile, makeprojects.buildme.BuildXCodeFile

Public Functions

__init__(*self*, *file_name*, *priority*, *configuration=None*)
Initializers for an *BuildObject*.

Parameters

- **file_name** – Name of the file to build.
- **priority** – Integer priority, lower will be built first.
- **configuration** – Configuration to build

__repr__(*self*)
Convert the object into a string.

Returns A full string.

build(*self*)
Perform the build operation.

Returns *BuildError* object as Unimplemented build.

run_command(*self*, *cmd*, *verbose*)
Issue a command and return the generated *BuildError*.

Parameters

- **cmd** – command line to execute
- **verbose** – True if verbose output is required

Returns *BuildError* object with error condition, if any.

Public Members

file_name
Name of file to build.

priority
Numeric priorty in ascending order.

configuration
Configuration if applicable.

4.3 Functions

4.3.1 Dispatchers

makeprojects.build

makeprojects.**build**(*working_directory=None*, *args=None*)
Invoke the buildme command line from within Python.

See also:

makeprojects.buildme

Parameters

- **working_directory** – None for current working directory.
- **args** – Argument list to pass to the command, None uses sys.argv.

Returns Zero on success, system error code on failure

makeprojects.clean

makeprojects.**clean**(*working_directory=None*, *args=None*)
Invoke the cleanme command line from within Python.

See also:

makeprojects.cleanme

Parameters

- **working_directory** – None for current working directory.
- **args** – Argument list to pass to the command, None uses sys.argv

Returns Zero on success, system error code on failure

makeprojects.rebuild

makeprojects.**rebuild**(*working_directory=None*, *args=None*)
Invoke the rebuildme command line from within Python.

See also:

makeprojects.rebuildme

See also:

makeprojects.rebuildme.main

Parameters

- **working_directory** – Directory to rebuild
- **args** – Command line to use instead of sys.argv

Returns Zero on no error, non-zero on error

4.3.2 Generators

makeprojects.new_solution

makeprojects.**new_solution**(*name=None*, *platform=None*, *project_type=None*)

Create a new instance of a full solution.

Convenience routine to create a Solution with a Project and three configurations 'Debug', 'Release', 'Internal'

See also:

core.Solution

Parameters

- **name** – Name of the project
- **platform** – Platform for the project
- **project_type** – Type of project

Returns None, a fully stocked Solution

makeprojects.new_project

Warning: doxygenfunction: Cannot find function "makeprojects::new_project" in doxygen xml output for project "makeprojects" from directory: temp/xml/

makeprojects.new_configuration

makeprojects.**new_configuration**(*configuration_list*)

Create a new instance of a *core.Configuration*.

Convenience routine to create a *core.Configuration* instance.

See also:

core.Configuration

Parameters **configuration_list** – Array of dict() records to describe configurations

Returns None, a single Configuration or a list of valid Configuration records.

4.3.3 Configuration

config.save_default

makeprojects.config.**save_default**(*working_directory=None*, *destinationfile=BUILD_RULES_PY*)

Calls the internal function to save a default .projectsrc file.

Given a pathname, create and write out a default .projectsrc file that can be used as input to makeprojects to generate project files.

Parameters

- **working_directory** – Directory to save the destination file
- **destinationfile** – Pathname of where to save the default configuation file

config.find_default_build_rules

makeprojects.config.**find_default_build_rules**()

Search for the build_rules.py file.

Scan for the build_rules.py file starting from the current working directory and search downwards until the root directoy is it. If not found, search in the user's home directory or for linux/macOS, in /etc

Returns Pathname of the configuration file, or None if no file was found.

config.import_configuration

makeprojects.config.**import_configuration**(*file_name=None*, *verbose=True*)

Load in the configuration file.

Using the file PROJECTSRC, load it in and parse it as an INI file using the configparser python class.

Parameters

- **file_name** – File to load for configuration
- **verbose** – If True, print the loaded file"s name.

Returns An empty parser object or filled with a successfully loaded file

4.3.4 Clean

cleanme.dispatch

Warning: doxygenfunction: Cannot find function "makeprojects::cleanme::dispatch" in doxygen xml output for project "makeprojects" from directory: temp/xml/

cleanme.process

Warning: doxygenfunction: Cannot find function "makeprojects::cleanme::process" in doxygen xml output for project "makeprojects" from directory: temp/xml/

cleanme.main

makeprojects.cleanme.**main**(*working_directory=None*, *args=None*)

Command line shell for `cleanme`.

Entry point for the program `cleanme`, this function will either get the parameters from sys.argv or the paramater `args`.

- `--version`, show version.
- `-r`, Perform a recursive clean.

- `-v`, Verbose output.
- `--generate-rules`, Create build_rules.py and exit.
- `--rules-file`, Override the configruration file.
- `-d`, List of directories to clean.

Parameters

- **working_directory** – Directory to operate on, or None for os.getcwd()
- **args** – Command line to use instead of sys.argv

Returns Zero on no error, non-zero on error

4.3.5 Build

buildme.build_rez_script

Warning: doxygenfunction: Cannot find function "makeprojects::buildme::build_rez_script" in doxygen xml output for project "makeprojects" from directory: temp/xml/

buildme.build_slicer_script

Warning: doxygenfunction: Cannot find function "makeprojects::buildme::build_slicer_script" in doxygen xml output for project "makeprojects" from directory: temp/xml/

buildme.build_doxygen

Warning: doxygenfunction: Cannot find function "makeprojects::buildme::build_doxygen" in doxygen xml output for project "makeprojects" from directory: temp/xml/

buildme.build_watcom_makefile

Warning: doxygenfunction: Cannot find function "makeprojects::buildme::build_watcom_makefile" in doxygen xml output for project "makeprojects" from directory: temp/xml/

buildme.build_makefile

Warning: doxygenfunction: Cannot find function "makeprojects::buildme::build_makefile" in doxygen xml output for project "makeprojects" from directory: temp/xml/

buildme.parse_sln_file

makeprojects.buildme.**parse_sln_file**(*full_pathname*)
Find build targets in .sln file.

Given a .sln file for Visual Studio 2003, 2005, 2008, 2010, 2012, 2013, 2015, 2017 or 2019, locate and extract all of the build targets available and return the list.

It will also determine which version of Visual Studio this solution file requires.

See also:

build_visual_studio

Parameters **full_pathname** – Pathname to the .sln file

Returns tuple(list of configuration strings, integer Visual Studio version year)

buildme.build_visual_studio

Warning: doxygenfunction: Cannot find function "makeprojects::buildme::build_visual_studio" in doxygen xml output for project "makeprojects" from directory: temp/xml/

buildme.parse_mcp_file

makeprojects.buildme.**parse_mcp_file**(*full_pathname*)
Extract configurations from a Metrowerks CodeWarrior project file.

Given an .mcp file for Metrowerks Codewarrior, determine which version of Codewarrrior was used to build it.

It will parse Freescale Codewarrior for Nintendo (59), Metrowerks Codewarrior 9.0 for Windows (50) and Metrowerks Codewarrior 10.0 for macOS (58)

See also:

build_codewarrior

Parameters **full_pathname** – Pathname to the .mcp file

Returns tuple(list of configuration strings, integer CodeWarrior Version)

buildme.build_codewarrior

Warning: doxygenfunction: Cannot find function "makeprojects::buildme::build_codewarrior" in doxygen xml output for project "makeprojects" from directory: temp/xml/

buildme.parse_xcodeproj_file

makeprojects.buildme.**parse_xcodeproj_file**(*full_pathname*)
Extract configurations from an XCode project file.

Given a .xcodeproj directory for XCode for macOS locate and extract all of the build targets available and return the list.

See also:

build_xcode

Parameters **full_pathname** – Pathname to the .xcodeproj folder

Returns list of configuration strings

buildme.build_xcode

Warning: doxygenfunction: Cannot find function "makeprojects::buildme::build_xcode" in doxygen xml output for project "makeprojects" from directory: temp/xml/

buildme.parse_codeblocks_file

makeprojects.buildme.**parse_codeblocks_file**(*full_pathname*)

Extract configurations from a Codeblocks project file.

Given a .cbp file for Codeblocks locate and extract all of the build targets available and return the list.

See also:

build_codeblocks

Parameters **full_pathname** – Pathname to the .cdp file

Returns list of configuration strings

buildme.build_codeblocks

Warning: doxygenfunction: Cannot find function "makeprojects::buildme::build_codeblocks" in doxygen xml output for project "makeprojects" from directory: temp/xml/

buildme.add_build_rules

makeprojects.buildme.**add_build_rules**(*projects*, *file_name*, *args*, *build_rules=None*)

Add a build_rules.py to the build list.

Given a build_rules.py to parse, check it for a BUILD_LIST and use that for scanning for functions to call. If BUILD_LIST doesn't exist, use *buildme.BUILD_LIST* instead.

All valid entries will be appended to the projects list.

See also:

add_project

Parameters

- **projects** – List of projects to build.
- **file_name** – Pathname to the build_rules.py file.
- **args** – Args for determining verbosity for output.
- **build_rules** – Preloaded build_rules.py object.

buildme.add_project

makeprojects.buildme.**add_project**(*projects*, *processed*, *file_name*, *args*)

Detect the project type and add it to the list.

Parameters

- **projects** – List of projects to build.
- **processed** – List of directories already processed.
- **file_name** – Pathname to the build_rules.py file.
- **args** – Args for determining verbosity for output.

Returns True if the file was buildable, False if not.

buildme.get_projects

Warning: doxygenfunction: Cannot find function "makeprojects::buildme::get_projects" in doxygen xml output for project "makeprojects" from directory: temp/xml/

buildme.process

Warning: doxygenfunction: Cannot find function "makeprojects::buildme::process" in doxygen xml output for project "makeprojects" from directory: temp/xml/

buildme.main

makeprojects.buildme.**main**(*working_directory=None*, *args=None*)

Command line shell for `buildme`.

Entry point for the program `buildme`, this function will either get the parameters from `sys.argv` or the paramater `args`.

- `--version`, show version.
- `-r`, Perform a recursive rebuild.
- `-v`, Verbose output.
- `--generate-rules`, Create build_rules.py and exit.
- `--rules-file`, Override the configruration file.
- `-f`, Stop building on the first build failure.

- -d, List of directories to rebuild.
- -docs, Compile Doxyfile files.
- Additional terms are considered specific files to build.

Parameters

- **working_directory** – Directory to operate on, or None.
- **args** – Command line to use instead of sys.argv.

Returns Zero on no error, non-zero on error

4.3.6 Rebuild

rebuild.main

makeprojects.rebuildme.**main**(*working_directory=None*, *args=None*)

Invoke the command line rebuildme.

Entry point for the program rebuildme, this function will either get the parameters from sys.argv or the paramater args.

- --version, show version.
- -r, Perform a recursive rebuild.
- -v, Verbose output.
- --generate-rules, Create build_rules.py and exit.
- --rules-file, Override the configruration file.
- -f, Stop building on the first build failure.
- -d, List of directories to rebuild.
- -docs, Compile Doxyfile files.
- Additional terms are considered specific files to build.

Parameters

- **working_directory** – Directory to rebuild, or None for os.getcwd()
- **args** – Command line to use instead of sys.argv

Returns Zero on no error, non-zero on error

4.3.7 Enums

enums.get_installed_visual_studio

makeprojects.enums.**get_installed_visual_studio**()
Find installed Visual Studio version.

Scan the host computer and return the *IDETypes* for the most recent version of Visual Studio that's installed.

Returns *IDETypes* value or None

enums.get_installed_xcode

makeprojects.enums.**get_installed_xcode**()
Find installed Xcode version.

Scan the host computer and return the *IDETypes* for the most recent version of XCode that's installed.

Returns *IDETypes* value or None

enums.platformtype_short_code

makeprojects.enums.**platformtype_short_code**(*configurations*)
Iterate over a list of Configurations to determine the short code.

For files that create multiple platforms, determine if it matches a known expandable PlatformType

Parameters **configurations** – List of configurations to scan

Returns Either '' or the generic short code of the group or the first code in the configuration list.

4.3.8 Core

core.source_file_filter

makeprojects.core.**source_file_filter**(*file_list*, *file_type_list*)
Prune the file list for a specific type.

Parameters

- **file_list** – list of *SourceFile* entries.
- **file_type_list** – FileTypes to match.

Returns list of matching *SourceFile* entries.

4.4 License

4.4.1 MIT License

The gist of the license… Have fun using this code, I won't sue you and you can't sue me. However, please be nice about it and give me a credit in your software that you used my code in.

Please?

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X