burger Documentation

Release 1.2.2

Rebecca Ann Heineman

Feb 04, 2022

CONTENTS

The `burger` module is a set of simple subroutines used by the `Burgerlib` build system.

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

CHAPTER

ONE

COMPATIBILITY

- Python 2.7.1 or higher
- Python 3.4 or higher

CHAPTER

TWO

INSTALLATION

Type in `pip install -U burger`. 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__

burger.__numversion__ = (1, 2, 2)
Numeric version.

__version__

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

__author__

burger.__author__ = 'Rebecca Ann Heineman'
Author's name.

__title__

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

__summary__

burger.__summary__ = 'Burger Becky\'s shared python library.'
Summary of the module's use.

__uri__

burger.__uri__ = 'http://pyburger.readthedocs.io'
Home page.

__email__

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

__license__

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

__copyright__

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

4.1.2 Configurations

Detect the version of the Python interpreter

strutils.PY2

burger.strutils.PY2 = 2
True if the interpreter is Python 2.x.

strutils.PY3_OR_HIGHER

burger.strutils.PY3_OR_HIGHER = 3
True if the interpreter is Python 3.x or higher.

strutils.PY3_3_OR_HIGHER

burger.strutils.**PY3_3_OR_HIGHER = (3, 3, 0)**
True if the interpreter is Python 3.3 or higher.

strutils.PY3_4_OR_HIGHER

burger.strutils.**PY3_4_OR_HIGHER = (3, 4, 0)**
True if the interpreter is Python 3.4 or higher.

strutils.PY3_5_OR_HIGHER

burger.strutils.**PY3_5_OR_HIGHER = (3, 5, 0)**
True if the interpreter is Python 3.5 or higher.

strutils.PYPY

burger.strutils.**PYPY = 'PyPy'**
True if the interpreter is PyPy.

strutils.IS_LINUX

burger.strutils.**IS_LINUX = sys.platform.startswith('linux')**
Running on linux?

strutils.IS_MACOSX

burger.strutils.**IS_MACOSX = sys.platform.startswith('darwin')**
Running on macOS X.

strutils.IS_CYGWIN

burger.strutils.**IS_CYGWIN = sys.platform.startswith('cygwin')**
Running on Cygwin.

strutils.IS_MSYS

burger.strutils.**IS_MSYS = sys.platform.startswith('msys')**
Running on MSYS.

strutils.IS_WSL

burger.strutils.**IS_WSL = IS_LINUX and 'icrosoft' in platform.platform()**
Running on Windows Subsystem for Linux.

strutils.IS_WINDOWS

burger.strutils.**IS_WINDOWS = sys.platform.startswith('win')**
Running on Windows.

strutils.UNICODE

burger.strutils.**UNICODE = unicode**
Class for declaring unicode strings.

strutils.LONG

burger.strutils.**LONG = long**
Class for declaring 64 bit integers.

4.2 Classes

4.2.1 Node

class burger.**Node**
Node class for creating directory trees.

Needed for some projects that have to store file entries in nested trees

Public Functions

__init__(*self*, *value*, *children=None*)
Create a node with an initial value.

Parameters

- **value** – Object to be the value of this node
- **children** – Array of nodes to be added as children to this one

__repr__(*self*, *level=0*)
Display this node as a string.

Parameters `level` – Recursion depth (Used internally)

Public Members

`value`
Value contained in this node.

`children`
Array of children nodes to this node.

4.2.2 Interceptstdout

`burger.Interceptstdout : public list`
Handy class for capturing stdout from tools and python itself.

Examples

```
# Import the class
from burger import Interceptstdout

# Instanciate the class, which intercepts stdout
with Interceptstdout() as output:
    do_somethingthatprints()
    print('capture me!')

# Once out of scope, output has a list of strings
# of the captured stdout output.
print(output)
```

Public Functions

`__init__`(*self*)
Declares the internal variables.

`__enter__`(*self*)
Invoked on 'with' which intercepts all future stdout.

`__exit__`(*self*, *args*)
Disconnect the stdout and store the items into a list of lines.

Using splitlines(), output the buffer into a list of lines into the output field.

4.2.3 BooleanProperty

burger.validators.BooleanProperty : public burger.validators.Property

Class to enforce bool in member variable.

Examples

```
'Inherit from (object) for Python 2.7'
>>> class foo(object):
    'Init to false'
    x = BooleanProperty(False)
    'Init to true'
    y = BooleanProperty(True)
    'Init to None'
    z = BooleanProperty()

'Create the class'
f = foo()

'Print True'
print(f.x)

'Print False'
f.x = False
print(f.x)

'f.x is set to bool False with string'
f.x = 'False'
print(f.x)

'Exception on bad write'
f.x = 'not boolean'
Traceback (most recent call last):
    ...
ValueError: Not boolean value
```

See also:

strutils.string_to_bool

Public Functions

__set__(*self*, *instance*, *value*)

Set the boolean value.

See also:

strutils.string_to_bool

Exception ValueError on invalid input.

Parameters

- **instance** – Reference to object containing data

- **value** – None or value the can be converted to bool

4.2.4 IntegerProperty

burger.validators.IntegerProperty : public burger.validators.Property

Class to enforce 64 bit integer in variable.

Examples

```
'Inherit from (object) for Python 2.7'
>>> class foo(object):
    'Init to 1'
    x = IntegerProperty(1.0)
    'Init to 55'
    y = IntegerProperty('55')
    'Init to None'
    z = IntegerProperty()

'Create the class'
f = foo()

'Print 1'
print(f.x)

'Print 0'
f.x = False
print(f.x)

'f.x is set to 99 with string'
f.x = '99.00'
print(f.x)

'Exception on bad write'
f.x = 'not boolean'
Traceback (most recent call last):
    ...
ValueError: Not integer value
```

Public Functions

__set__(*self*, *instance*, *value*)

Set the integer value.

See also:

strutils.string_to_bool

Exception ValueError on invalid input.

Parameters

- **instance** – Reference to object containing data

- **value** – None or value the can be converted to bool

4.2.5 StringProperty

burger.validators.StringProperty : public burger.validators.Property

Class to enforce string in member variable.

Examples

```
'Inherit from (object) for Python 2.7'
>>> class foo(object):
    'Init to "foo"'
    x = StringProperty('foo')
    'Init to None'
    y = StringProperty()

'Create the class'
f = foo()

'Print foo'
print(f.x)

'Print False'
f.x = 'False'
print(f.x)

'Print True'
f.x = True
print(f.x)
```

Public Functions

__set__(*self*, *instance*, *value*)

Set the string value.

Parameters

- **instance** – Reference to object containing data
- **value** – None or value the can be converted to bool

4.2.6 StringListProperty

burger.validators.StringListProperty : public burger.validators.Property

Class to enforce string list in member variable.

Examples

```
'Inherit from (object) for Python 2.7'
>>> class foo(object):
    'Init to ["foo"]'
    x = StringListProperty('foo')
```

(continues on next page)

(continued from previous page)

```
    'Init to None'
    y = StringListProperty()
    'Init to ["a","b","c"]'
    z = StringListProperty(["a","b","c"])

'Create the class'
f = foo()

'Print ["foo"]'
print(f.x)

'Print ['False']
f.x = 'False'
print(f.x)

'Print True'
f.x = True
print(f.x)
```

Public Functions

__get__(*self*, *instance*, *owner=None*)
Return value.

Parameters

- **instance** – Reference to object containing data
- **owner** – Not used

Returns None, or verified data

__set__(*self*, *instance*, *value*)
Set the string value.

Parameters

- **instance** – Reference to object containing data
- **value** – None or value the can be converted to bool

4.2.7 EnumProperty

burger.validators.EnumProperty : public burger.validators.Property
Class to enforce string list in member variable.

Examples

```
j = (('a', 'b', 'c'), 'd', 'e', ['f', 'g', 'h'], 'i')
'Inherit from (object) for Python 2.7'
>>> class foo(object):
    'Init to 0'
    x = EnumProperty(j, "a")
    'Init to 0'
```

(continues on next page)

(continued from previous page)

```
    y = EnumProperty(j)
    'Init to 4'
    z = EnumProperty(j, "i")

'Create the class'
f = foo()

'Print 0'
print(f.x)

'Print 2
f.x = 'g'
print(f.x)

'Print 2'
f.x = 'h'
print(f.x)
```

Public Functions

__init__(*self*, *name*, *enums*)
Initialize to default.

Parameters

- **name** – Name of the instance storage index
- **enums** – list of enumeration strings

__set__(*self*, *instance*, *value*)
Set the string value.

Parameters

- **instance** – Reference to object containing data
- **value** – None or value the can be converted to bool

4.2.8 NoneProperty

burger.validators.NoneProperty : public object
Class to enforce None in member variable.

Examples

```
'Inherit from (object) for Python 2.7'
>>> class foo(object):
    'Init to None'
    x = NoneProperty()

'Create the class'
f = foo()
```

(continues on next page)

(continued from previous page)

```
'Print None'
print(f.x)

'Exception on non None data'
f.x = 'not None'
Traceback (most recent call last):
    ...
ValueError: Not None value
```

Public Functions

__init__(*self*, *name*)
Initialize to default.

Parameters **name** – Name of the instance storage index

__get__(*self*, *instance*, *owner=None*)
Return None.

Parameters

- **instance** – Reference to object containing data
- **owner** – Not used

Returns None, or verified data

__set__(*self*, *instance*, *value*)
Throw if not None.

Parameters

- **instance** – Reference to object containing data
- **value** – None or value the can be converted to bool

4.3 Functions

4.3.1 String Functions

strutils.unicode_print

burger.strutils.**unicode_print**(*input_string*)
Handle printing a unicode string to stdout.

On some platforms, printing a unicode string will trigger a UnicodeEncodeError exception. In these cases, handle the exception and recode the string to the native string encoding.

Parameters **input_string** – A unicode string to print to stdout.

strutils.is_string

burger.strutils.**is_string**(*item*)
Return True if input is a string object.

Test the input if it's either an instance of basestring in Python 2.x or (str, bytes) in Python 3.x

Parameters **item** – Object to test

Returns True if the object is a string instance, False if not.

strutils.convert_to_array

burger.strutils.**convert_to_array**(*input_array*)
Convert a string to a string array (list)

If the input is None, return an empty list. If it's a string, convert the string to a single entry list. Otherwise, assume it's an iterable dir, list or tuple of strings.

Parameters **input_array** – The object to test

Returns The input, or a string encapsulated into a single entry list.

strutils.string_to_bool

burger.strutils.**string_to_bool**(*item*)
Convert an item to a boolean.

Strings 'true', 't', ' on', 'yes', and, 'y' return True 'false', 'f', 'off', 'no', and 'n' return False

Non zero numbers or strings that are numbers become True Zero and '0' become False.

Exception ValueError on invalid input.

Parameters **item** – String or integer to convert.

Returns `True` or `False`

strutils.TrueFalse

burger.strutils.**TrueFalse**(*item*)
Convert the input into a boolean and return the string 'True' or 'False'.

If the input was a string of '0' or 'False' (Case insensitive comparision), this function will return 'False'. Empty dictionary, string or list objects, or the number zero will also return 'False'

See also:

truefalse

See also:

TRUEFALSE

Parameters **item** – Object to convert to a bool before converting into a string

Returns The string 'True' or 'False'

strutils.truefalse

burger.strutils.**truefalse**(*item*)

Convert the input into a boolean and return the string 'true' or 'false'.

If the input was a string of '0' or 'False' (Case insensitive comparision), this function will return 'false'. Empty dictionary, string or list objects, or the number zero will also return 'false'

See also:

TRUEFALSE

See also:

TrueFalse

Parameters **item** – Object to convert to a bool before converting into a string

Returns The string 'true' or 'false'

strutils.TRUEFALSE

burger.strutils.**TRUEFALSE**(*item*)

Convert the input into a boolean and return the string 'TRUE' or 'FALSE'.

If the input was a string of '0' or 'False' (Case insensitive comparision), this function will return 'FALSE'. Empty dictionary, string or list objects, or the number zero will also return 'FALSE'

See also:

truefalse

See also:

TrueFalse

Parameters **item** – Object to convert to a bool before converting into a string

Returns The string 'TRUE' or 'FALSE'

strutils.convert_to_windows_slashes

burger.strutils.**convert_to_windows_slashes**(*path_name*, *force_ending_slash=False*)

Convert a filename from Linux/macOS to Windows format.

Convert all '/' characters into '\' characters

If force_ending_slash is True, append a '\' if one is not present in the final string

See also:

convert_to_linux_slashes

Parameters

- **path_name** – A pathname to be converted to Windows slashes

- **force_ending_slash** – True if a '' character is to be forced at the end of the output

Returns A pathname using Windows type slashes ''

strutils.convert_to_linux_slashes

burger.strutils.**convert_to_linux_slashes**(*path_name*, *force_ending_slash=False*)
Convert a filename from Windows to Linux/macOS format.

Convert all '' characters into '/' characters

See also:

convert_to_windows_slashes

Parameters

- **path_name** – A string object that text substitution will occur
- **force_ending_slash** – True if a '/' character is to be forced at the end of the output

Returns A pathname using Linux/BSD type slashes '/'

strutils.encapsulate_path_windows

burger.strutils.**encapsulate_path_windows**(*input_path*)
Quote a pathname for use in the Windows system shell.

On Windows platforms, if the path has a space or other character that could confuse COMMAND.COM, the string will be quoted and double quotes within the string handled properly. All slash characters will be replaced with backslash characters.

See also:

encapsulate_path

Parameters **input_path** – string with the path to encapsulate using Windows rules

Returns Original input string if Windows can accept it or input properly quoted

strutils.encapsulate_path_linux

burger.strutils.**encapsulate_path_linux**(*input_path*)
Quote a pathname for use in the linux or BSD system shell.

On Linux platforms, if the path has a space or other character that could confuse bash, the string will be quoted and double quotes within the string handled properly. All backslash characters will be replaced with slash characters.

See also:

encapsulate_path

Parameters **input_path** – string with the path to encapsulate using Windows rules

Returns Original input string if Windows can accept it or input properly quoted

strutils.encapsulate_path

burger.strutils.**encapsulate_path**(*input_path*)
Quote a pathname for use in the native system shell.

On Windows platforms, if the path has a space or other character that could confuse COMMAND.COM, the string will be quoted, and for other platforms, it will be quoted using rules that work best for BASH. This will also quote if the path has a ';' which could be used to confuse bash.

See also:

encapsulate_path_windows

See also:

encapsulate_path_linux

Parameters **input_path** – string with the path to encapsulate

Returns Input string or input properly quoted

strutils.encapsulate_hosted_path

burger.strutils.**encapsulate_hosted_path**(*input_path*)
Quote a pathname for use in the windows system shell.

If the platform is hosted on Windows, convert the pathname to Windows and then use Windows encapsulation rules.

See also:

encapsulate_path_windows

See also:

encapsulate_path_linux

See also:

encapsulate_path

Parameters **input_path** – string with the path to encapsulate

Returns Input string or input properly quoted

strutils.split_comma_with_quotes

burger.strutils.**split_comma_with_quotes**(*comma_string*)
Split comma seperated string while handling quotes.

str.split(',') will split a string into a list but it doesn't handle entries that are encased in quotes. This function will scan for quote characters and skip over any comma that's encased in quotes.

Examples

```
# Result is ['"foo,bar"','foo','bar']
lines = burger.strutils.split_comma_with_quotes('"foo,bar",foo,bar')

# Will raise an error due to missing end quote
willraise = burger.strutils.split_comma_with_quote('"foo,bar')
```

Return List of string fragments for each comma seperated entries

Parameters **comma_string** – String of comma seperated strings

Throws ValueError

strutils.parse_csv

burger.strutils.**parse_csv**(*csv_string*)

Parse a comma seperated string allowing quoted strings.

Given a string of comma seperated entries and handle quotes properly.

Examples

```
# Result is ['foo,bar','foo','bar']
lines = burger.strutils.split_comma_with_quotes('"foo,bar",foo,bar')

# Result is ['foo"bar','"boo'boo'"]
lines = burger.strutils.split_comma_with_quotes(
    '"foo""bar","'boo,boo'")

# Will raise an error due to missing end quote
willraise = burger.strutils.split_comma_with_quote('"foo,bar')
```

Parameters **csv_string** – String of comma seperated entries

Throws ValueError

Returns List of entries with whitespace stripped from prefix and suffix

strutils.translate_to_regex_match

burger.strutils.**translate_to_regex_match**(*file_list*)

Translate filename wildcards into regexes.

Results List of re.compile().match entries

Parameters **file_list** – List of filename wildcards

strutils.host_machine

burger.strutils.**host_machine**()
Return the high level operating system's name.

Return the machine this script is running on, 'windows', 'macosx', 'linux' or 'unknown'

See also:

get_mac_host_type

See also:

get_windows_host_type

Returns The string 'windows', 'macosx', 'linux', or 'unknown'

strutils.get_windows_host_type

burger.strutils.**get_windows_host_type**(*wsl_allowed=False*)
Return windows host type (32 or 64 bit)

Return False if the host is not Windows, 'x86' if it's a 32 bit host and 'x64' if it's a 64 bit host, and possibly 'arm' if an arm host

See also:

get_mac_host_type

See also:

host_machine

Parameters **wsl_allowed** – If True, allow returning a host type if cygwin is the shell.

Returns The string 'x64', 'x86', 'arm', 'arm64', 'ia64' or False

strutils.get_mac_host_type

burger.strutils.**get_mac_host_type**()
Return Mac OSX host type (PowerPC/Intel)

Return False if the host is not Mac OSX. 'ppc' or 'ppc64' if it's a Power PC based system, 'x86' or 'x64' for Intel (Both 32 and 64 bit)

See also:

get_windows_host_type

See also:

host_machine

Returns The string 'x86', 'x64', 'ppc', 'ppc64' or False.

strutils.escape_xml_cdata

burger.strutils.**escape_xml_cdata**(*xml_string*)
Convert escape codes for CDATA XML records.

According to the XML docs, &, < and > cannot exist in a string so they must be replaced with "&", "<" and ">" respectively.

Return Original string if no changes, or a new string with escaped characters.

Parameters **xml_string** – String to convert to one compatible with XML CDATA

strutils.escape_xml_attribute

burger.strutils.**escape_xml_attribute**(*xml_string*)
Convert escape codes for XML element attribute records.

According to the XML docs, ", &, < and > cannot exist in a string

so they must be replaced with "&quot'", &", "<" and ">" respectively.

Tabs and line feeds will be converted to "
" and "	".

Return Original string if no changes, or a new string with escaped characters.

Note: https://www.w3.org/TR/REC-xml/#sec-line-ends

Parameters **xml_string** – String to convert to XML attribute strings.

strutils.packed_paths

burger.strutils.**packed_paths**(*entries*, *slashes=None*, *seperator=None*, *force_ending_slash=False*)
Convert a list of paths and convert to a PATH string.

Convert ['a','b','c'] to a;b;c

Return String of all entries seperated by ';' or `seperator`

Parameters

- **entries** – list of strings to concatenate
- **slashes** – None for no conversion, '/' or '\' path seperator
- **seperator** – Character to use to seperate entries, ';' is used for None
- **force_ending_slash** – Enforce a trailing slash if True

strutils.make_version_tuple

burger.strutils.**make_version_tuple**(*version_string*)
Convert numeric version string into an int tuple.

Given a string like '1.0.5rc' and return a tuple of (1, 0, 5). The numbers are seperated by periods and members that start with a non number are skipped.

Parameters **version_string** – String for the version number.

Returns tuple of integers with the version. Can be an empty tuple.

4.3.2 File Locators

buildutils.get_sdks_folder

burger.buildutils.**get_sdks_folder**(*verbose=False*, *refresh=False*, *folder=None*)
Return the path of the BURGER_SDKS folder.

If the environment variable BURGER_SDKS is set, return the pathname it contains. Otherwise, print a warning if verbose is True and then attempt to find the 'sdks' folder by traversing the current working directory for a folder named 'sdks'. If one isn't found, return None.

Examples

```
# Normal use
sdksfolder = burger.buildutils.get_sdks_folder()
if not sdksfolder:
    print('failure')
    raise NameError("sdks not found, set BURGER_SDKS")

# Alert the user if BURGER_SDKS isn't set
burger.buildutils.get_sdks_folder(verbose=True)

# Force the use of a supplied folder for sdks
burger.buildutils.get_sdks_folder(refresh=True, folder='./foo/sdks/')
```

Parameters

- **verbose** – If True, print a message if BURGER_SDKS was not present
- **refresh** – If True, reset the cache and force a reload.
- **folder** – Path to use as BURGER_SDKS in the cache as an override

Returns None if the environment variable is not set, or the value of BURGER_SDKS.

buildutils.find_in_path

burger.buildutils.**find_in_path**(*filename*, *search_path=None*, *executable=False*)
Using the system PATH environment variable, search for a file.

If the flag executable is False, the file will be found using a simple path search. If the flag is True, the file will be searched for using the extensions in the PATHEXT environment variable in addition to use the filename as is.

If search_path is a string, it will be seperated using os.pathsep. If not, it will be treated as an interable list of strings of full pathnames to search. If it is None, the PATH environment variable will be used.

Examples

```
# Can return 'doxygen', 'doxygen.exe' or 'doxygen.com' depending
# on what was found
burger.find_in_path('doxygen', executable=True)

# Will only find 'foo.txt'
burger.find_in_path('foo.txt')
```

See also:

burger.buildutils.get_path_ext

See also:

burger.buildutils.make_exe_path

Return None if not found, a full path if the file is found.

Parameters

- **filename** – File to locate
- **search_path** – Search paths to use instead of PATH
- **executable** – True to ensure it's an executable

buildutils.where_is_doxygen

burger.buildutils.**where_is_doxygen**(*verbose=False*, *refresh=False*, *path=None*)
Return the location of Doxygen's executable.

Look for an environment variable DOXYGEN and determine if the executable resides there, if so, return the string to the path

If running on a MacOSX client, look in the Applications folder for a copy of Doxygen.app and return the pathname to the copy of doxygen that resides within

PATH is then searched for doxygen, and if it's not found, None is returned.

Parameters

- **verbose** – If True, print a message if doxygen was not found
- **refresh** – If True, reset the cache and force a reload.
- **path** – Path to doxygen to place in the cache

Returns A path to the Doxygen command line executable or None if not found.

buildutils.where_is_git

burger.buildutils.**where_is_git**(*verbose=False*, *refresh=False*, *path=None*)
Return the location of the git executable.

Look for an environment variable GIT and determine if the executable resides there, if so, return the string to the path.

PATH is then searched for git, and if it's not found, None is returned.

See also:

where_is_p4

See also:

is_under_git_control

Parameters

- **verbose** – If True, print a message if git was not found
- **refresh** – If True, reset the cache and force a reload.
- **path** – Path to git to place in the cache

Returns A path to the git command line executable or None if not found.

buildutils.is_under_git_control

burger.buildutils.**is_under_git_control**(*working_directory*)
Test if the directory is under git source control.

First test if git is installed by calling *where_is_git()*. Then use the git tool to query if the working directory is under git source control.

See also:

where_is_git

Parameters **working_directory** – Directory to test.

Returns True if the directory is under git control, False if not.

buildutils.where_is_p4

burger.buildutils.**where_is_p4**(*verbose=False*, *refresh=False*, *path=None*)
Return the location of the p4 executable.

Look for an environment variable PERFORCE and determine if the executable resides there, if so, return the string to the path.

PATH is then searched for p4, and if it's not found, None is returned.

See also:

perforce_edit

See also:

perforce_add

See also:

where_is_git

See also:

is_under_p4_control

Parameters

- **verbose** – If True, print a message if Perforce was not found
- **refresh** – If True, reset the cache and force a reload.
- **path** – Path to Perforce to place in the cache

Returns A path to the Perforce command line executable or None if not found.

buildutils.is_under_p4_control

burger.buildutils.**is_under_p4_control**(*working_directory*)

Test if the directory is under Perforce source control.

First test if p4 is installed by calling *where_is_p4()*. Then use the p4 tool to query if the working directory is under Perforce source control.

See also:

where_is_p4

Note: On folders that are not under Perforce control, p4 may take as much as 15 seconds to return a result, so use this call with caution.

Parameters **working_directory** – Directory to test.

Returns True if the directory is under Perforce control, False if not.

buildutils.where_is_watcom

burger.buildutils.**where_is_watcom**(*command=None*, *verbose=False*, *refresh=False*, *path=None*)

Return the location of Watcom's executables.

Look for an environment variable WATCOM and determine if the executable resides there, if so, return the string to the path

In Windows, the boot drive is checked for a WATCOM folder and if found, that folder name is returned. If all checks failed, None is returned.

Parameters

- **command** – Watcom program to find.
- **verbose** – If True, print a message if watcom was not found
- **refresh** – If True, reset the cache and force a reload.

- **path** – Path to watcom to place in the cache

Returns A path to the Watcom folder or None if not found.

buildutils.where_is_visual_studio

burger.buildutils.**where_is_visual_studio**(*vs_version*)

Locate devenv.com for a specific version of Visual Studio.

Given a specific version by year, check for the appropriate environment variable that contains the path to the executable of the IDE

Examples

```
# Normal use
vs_path = burger.buildutils.where_is_visual_studio(2010)
if not vs_path:
    print('Visual Studio 2010 not found')
    raise NameError("Visual Studio 2010 not found")
```

Note: This function will always return None on non-windows hosts.

Parameters **vs_version** – Version year as number

Returns Path to devenv.com for the IDE or None.

buildutils.where_is_codeblocks

burger.buildutils.**where_is_codeblocks**(*verbose=False*, *refresh=False*, *path=None*)

Return the location of CodeBlocks's executable.

Look for an environment variable CODEBLOCKS and determine if the executable resides there, if so, return the string to the path

If running on a MacOSX client, look in the Applications folder for a copy of CodeBlocks.app and return the pathname to the copy of CodeBlocks that resides within

PATH is then searched for CodeBlocks, and if it's not found, None is returned.

Parameters

- **verbose** – If True, print a message if CodeBlocks was not found
- **refresh** – If True, reset the cache and force a reload.
- **path** – Path to CodeBlocks to place in the cache

Returns A path to the CodeBlocks command line executable or None if not found.

buildutils.where_is_xcode

burger.buildutils.**where_is_xcode**(*xcode_version=None*)
Locate xcodebuild for a specific version of XCode.

Given a specific version by version, scan the locations that the IDE would be found.

Examples

```
# Normal use
xcode_path = burger.buildutils.where_is_xcode(10)
if not xcode_path:
    print('XCode 10 not found')
    raise NameError("XCode 10 not found")
```

Note: This function will always return None on non-macOS hosts. Minimum version of XCode is 3.

Parameters **xcode_version** – Version number

Returns Path to xcodebuild for the XCode version or None.

4.3.3 File Functions

fileutils.is_write_protected

burger.fileutils.**is_write_protected**(*path_name*)
Test if a file is write protected.

If the file/directory exists, it is tested if it's write protected. If it exists and is write protected, True is returned, otherwise False is returned.

Parameters **path_name** – Path name to the file/directory

Returns True if the file exists and is write protected

fileutils.make_executable

burger.fileutils.**make_executable**(*exe_path*)
Set the executable flag to true on a file.

Parameters **exe_path** – Pathname to the executable to fix up.

fileutils.create_folder_if_needed

burger.fileutils.**create_folder_if_needed**(*path*)
Given a pathname to a folder, if the folder doesn't exist, create it.

Call os.makedirs(path) but does not throw an exception if the directory already exists. All other exceptions are passed through with raise.

See also:

delete_directory

Parameters **path** – A string object with the pathname.

fileutils.delete_file

burger.fileutils.**delete_file**(*filename*)

Given a pathname to a file, delete it.

If the file doesn't exist, it will return without raising an exception.

See also:

delete_directory

Parameters **filename** – A string object with the filename

fileutils.is_source_newer

burger.fileutils.**is_source_newer**(*source*, *destination*)

Return False if the source file is older then the destination file.

Check the modification times of both files to determine if the source file is newer. If the destination file is older or doesn't exist True is returned.

Return False if destination is newer, not False if not.

Examples

```
result = burger.fileutils.is_source_newer('file.c', 'file.obj')

if result == 2:
    build_file_c()

if result:
    compile('file.c', 'file.obj')
else:
    print('Already built')
```

Note: If the source file does not exist, the function will return 2. This is to allow proper error checking if the source is required to exist.

Parameters

- **source** – string pathname of the file to test
- **destination** – string pathname of the file to test against

Returns False if not newer, True if newer, 2 if there is no source file

fileutils.copy_file_if_needed

burger.fileutils.**copy_file_if_needed**(*source*, *destination*, *verbose=True*, *perforce=False*)
Copy a file only if newer than the destination.

Copy a file only if the destination is missing or is older than the source file.

See also:

is_source_newer

Parameters

- **source** – string pathname of the file to copy from
- **destination** – string pathname of the file to copy to
- **verbose** – True if print output is desired
- **perforce** – True if Perforce 'p4 edit' should be done on the destination.

Returns Zero if no error otherwise IOError.errno

fileutils.copy_directory_if_needed

burger.fileutils.**copy_directory_if_needed**(*source*, *destination*, *exception_list=None*, *verbose=True*)
Copy all of the files in a directory into a new directory.

Creating any necessary directories in the process, and it will skip files with specific extensions

See also:

copy_file_if_needed

See also:

create_folder_if_needed

Note: This is a recursive function

Parameters

- **source** – string pathname of the directory to copy from
- **destination** – string pathname of the directory to copy to
- **exception_list** – optional list of file extensions to ignore during copy
- **verbose** – True if print output is desired

Returns Zero if no error, non-zero on error

fileutils.shutil_readonly_cb

burger.fileutils.**shutil_readonly_cb**(*func*, *path*, *exception_info*)
Subroutine for shutil.rmtree() to delete read only files.

shutil.rmtree() raises an exception if there are read only files in the directory being deleted. Use this callback to allow read only files to be disposed of.

Examples

```
import burger
import shutil

shutil.rmtree(PATH_TO_DIRECTORY,onerror = burger.shutil_readonly_cb)
```

See also:

delete_directory

Note: This is a callback function

Parameters

- **func** – Not used
- **path** – pathname of the file that is read only
- **exception_info** – Information about the exception

fileutils.delete_directory

burger.fileutils.**delete_directory**(*path*, *delete_read_only=False*)
Recursively delete a directory.

Delete a directory and all of the files and directories within.

See also:

shutil_readonly_cb

See also:

create_folder_if_needed

Parameters

- **path** – Pathname of the directory to delete
- **delete_read_only** – True if read only files are to be deleted as well

fileutils.clean_directories

burger.fileutils.**clean_directories**(*path*, *name_list*, *recursive=False*)

Recursively clean directories with a name list.

Examples

```
# Delete all temp and __pycache__ files recursively
burger.fileutils.clean_directories(
    '.',
    ('*.temp', '__pycache__'),
    True)
```

See also:

clean_files

See also:

delete_directory

Parameters

- **path** – Pathname of the directory to scan
- **name_list** – Iterable of directory names
- **recursive** – Boolean if recursive clean is desired

fileutils.clean_files

burger.fileutils.**clean_files**(*path*, *name_list*, *recursive=False*)

Recursively clean files with a filename list.

Examples

```
# Delete all .obj and .lib files recursively
burger.fileutils.clean_files('temp', ('*.obj', '*.lib'), True)
```

See also:

delete_file

See also:

delete_directory

Parameters

- **path** – Pathname of the directory to scan
- **name_list** – Iterable of file names
- **recursive** – Boolean if recursive clean is desired

fileutils.get_tool_path

burger.fileutils.**get_tool_path**(*tool_folder*, *tool_name*, *encapsulate=False*)
Find executable tool directory.

For allowing builds on multiple operating system hosts under the Burgerlib way of project management, it's necessary to query what is the host operating system and glean out which folder to find a executable compiled for that specific host

Parameters

- **tool_folder** – Pathname to the folder that contains the executables
- **tool_name** – Bare name of the tool (Windows will append '.exe')
- **encapsulate** – False if a path is requested, True if it's quoted to be used as a string to be sent to command line shell

Returns Full pathname to the tool to execute

fileutils.traverse_directory

burger.fileutils.**traverse_directory**(*working_dir*, *filename_list*, *terminate=False*, *find_directory=False*)
Create a list of all copies of a file following a directory.

Starting with a working directory, test if a file exists and if so, insert it into a list. The list will be starting from the root with the last entry being at the working directory

Parameters

- **working_dir** – string with the path of the folder to start the search
- **filename_list** – string or an iterable of strings with the name(s) of the file(s) to find in the scanned folders
- **terminate** – True if searching will end on the first found file
- **find_directory** – True if searching for directories instead of files.

Returns List of pathnames (With filename appended)

fileutils.unlock_files

burger.fileutils.**unlock_files**(*working_dir*, *recursive=False*)
Iterate over a directory and unlock all read-only files.

This function will generate a list of fully qualified pathnames of every file that was unlocked. Directories will be skipped.

Examples

```
# Any file that is read only in this directory is now unlocked
lock_list = unlock_files("~/projects/lockedfiles")

# Do stuff on the files
do_code_on_unlocked_files()

# Re-lock all the files that were unlocked.
lock_files(lock_list)
```

See also:

lock_files

Parameters

- **working_dir** – Pathname to the directory to traverse for read-only files
- **recursive** – False (default) don't recurse through folders, True, recurse

Returns A list object with the name of every file that was unlocked.

fileutils.lock_files

burger.fileutils.**lock_files**(*lock_list*)

Iterate over the input list and mark all files as read-only.

See also:

unlock_files

Parameters **lock_list** – Iterable object containing a list of path names to files or directories to mark as "read-only"

fileutils.load_text_file

burger.fileutils.**load_text_file**(*file_name*)

Load in a text file as a list of lines.

Read in a text file as a list of lines and handle all three line ending types (\r, \n and \r\n)

See also:

save_text_file

See also:

compare_files

Note: This function assumes the file is utf-8 with or without a byte order mark.

Parameters **file_name** – File to load

Returns A list object with the file

fileutils.save_text_file

burger.fileutils.**save_text_file**(*file_name*, *text_lines*, *line_feed=None*, *bom=False*)
Save in a text file from an iterable of lines.

Save a text file from an iterable of lines and allow custom line endings. If line_feed is None, the line feed will be the system default.

See also:

load_text_file

Note: This function will write out the text file using utf-8 encoding.

Parameters

- **file_name** – File to load
- **text_lines** – Lines to save
- **line_feed** – String to use as a line feed
- **bom** – If True write the UTF-8 Byte Order Mark

fileutils.compare_files

burger.fileutils.**compare_files**(*filename1*, *filename2*)
Compare text files for equality.

Check if two text files are the same length, and then test the contents to verify equality.

See also:

compare_file_to_string

Parameters

- **filename1** – string object with the pathname of the file to test
- **filename2** – string object with the pathname of the file to test against

Returns True if the files are equal, False if not.

fileutils.compare_file_to_string

burger.fileutils.**compare_file_to_string**(*file_name*, *text_lines*)
Compare text file and a string for equality.

Check if a text file is the same as a string by loading the text file and testing line by line to verify the equality of the contents

See also:

compare_files

Parameters

- **file_name** – string object with the pathname of the file to test
- **text_lines** – string object to test against

Returns True if the file and the string are the same, False if not

fileutils.read_zero_terminated_string

burger.fileutils.**read_zero_terminated_string**(*filep*, *encoding='utf-8'*)

Read a zero terminated string from an open binary file.

Read in a stream of bytes and stop at the end of file or a terminating zero. The string will be converted from utf-8 into unicode by default before returning.

Parameters

- **filep** – File record of a file opened in binary mode
- **encoding** – Character set encoding of the string

Returns None or the unicode string (Without the terminating zero)

fileutils.save_text_file_if_newer

burger.fileutils.**save_text_file_if_newer**(*file_name*, *text_lines*, *line_feed=None*, *bom=False*, *perforce=False*, *verbose=False*)

Save in a text file from an iterable of lines if newer.

Compare an iterable of lines to a pre-existing text file. If the text file either exists or differs from the input, write a new text file to disk.

See also:

save_text_file

See also:

compare_file_to_string

See also:

perforce_edit

See also:

perforce_add

Note: This function will write out the text file using utf-8 encoding.

Parameters

- **file_name** – File to load
- **text_lines** – Lines to save
- **line_feed** – String to use as a line feed
- **bom** – If True write the UTF-8 Byte Order Mark

- **perforce** – Enable perforce checkout or add if True
- **verbose** – Enable messages if True

Returns True if no change was performed, False if the file was written

4.3.4 Build Helpers

buildutils.fix_csharp

burger.buildutils.**fix_csharp**(*csharp_application_path*)

Convert pathname to execute a C# exe file.

C# applications can launch as is on Windows platforms, however, on Mac OSX and Linux, it must be launched from mono. Determine the host machine and if not windows, automatically prepend 'mono' to the application's name to properly launch it

This will also encase the name in quotes in case there are spaces in the pathname

Parameters **csharp_application_path** – Pathname string to update

Returns List of commands for the platform to launch a C# application.

buildutils.is_exe

burger.buildutils.**is_exe**(*exe_path*)

Return True if the file is executable.

Note: Windows platforms don't support the 'x' bit so all files are executable if they exist.

Parameters **exe_path** – Full or partial pathname to test for existance

Returns True if the file is executable, False if the file doesn't exist or is not valid.

buildutils.get_path_ext

burger.buildutils.**get_path_ext**(*pathext=None*)

Return a list of executable extensions.

If pathext is None, query the environment variable PATHEXT and return the entries as a string list. If pathext is a string, parse it as if it was a system specific PATHEXT string and if it's an iterable, return the value as is. If PATHEXT doesn't exist or is empty, return an empty list.

Windows usually sets the value like this `PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC`

See also:

burger.buildutils.make_exe_path

See also:

burger.buildutils.find_in_path

Parameters **pathext** – String parsed as PATHEXT, iterable returned as is

Returns List of file name extension strings.

buildutils.make_exe_path

burger.buildutils.**make_exe_path**(*exe_path*, *pathext=None*)
Given a folder and a executable name, return the true absolute path.

Examples

```
# exe could be returned as exe, exe.exe, exe.cmd, etc...
path = make_exe_path('C:\\code\\exe')
if path is None:
    print('No file named exe at C:\\code')
```

See also:

burger.buildutils.get_path_ext

See also:

burger.buildutils.find_in_path

Note: On macOS and Linux, PATHEXT is not set, this is for supporting extension types for common batch files or other executable extensions.

Parameters

- **exe_path** – Path of the executable to test
- **pathext** – Extension list to test

Returns None if a match was not found, or a full pathname with extension.

buildutils.expand_and_verify

burger.buildutils.**expand_and_verify**(*file_string*)
Expand the input string with os.path.expandvars()

After expanding the string, test for the existence of the file and return the expanded path if True. Otherwise, return None

Examples

```
perforcepath = burger.expand_and_verify('${PERFORCE}\\p4.exe')
if perforcepath is None:
    return
```

Parameters **file_string** – Pathname with environment variable tokens

Returns None if the string couldn't be expanded or if the file didn't exist, otherwise, return the expanded pathname

buildutils.make_git_version_header

burger.buildutils.**make_git_version_header**(*working_dir*, *outputfilename*, *verbose=False*)
Create a C header with the git version.

This function assumes version control is with git!

Get the last change list and tag, and then create a header with this information (Only modify the output file if the contents have changed)

C++ defines are declared for GIT_HASH, GIT_CHANGEDATE, GIT_CHANGETIME, GIT_TAG_VERSION GIT_TAG_VERSION_INFO

Parameters

- **working_dir** – string with the path of the folder to obtain the git version for
- **outputfilename** – string with the path of the generated header
- **verbose** – Print git commands and other informational messages

Returns Zero if no error, non-zero on error

buildutils.perforce_command

burger.buildutils.**perforce_command**(*files*, *command*, *verbose=False*)
Given a list of files, send a command to execute on them in perforce.

Pass either a single string or a string list of pathnames of files to checkout in perforce using the 'p4' command with the command name

See also:

where_is_p4

Parameters

- **files** – list or string object of file(s) to checkout
- **command** – string to pass to p4 such as 'edit' or 'add'
- **verbose** – If True, print the command line and warnings

Returns Zero if no error, non-zero on error

buildutils.perforce_edit

burger.buildutils.**perforce_edit**(*files*, *verbose=False*)
Given a list of files, checkout (Edit) them in perforce.

Pass either a single string or a string list of pathnames of files to checkout in perforce using the 'p4 edit' command

See also:

where_is_p4

Parameters

- **files** – list or string object of file(s) to checkout

- **verbose** – If True, print the command line and warnings

Returns Zero if no error, non-zero on error

buildutils.perforce_add

burger.buildutils.**perforce_add**(*files*, *verbose=False*)

Given a list of files, add them in perforce.

Pass either a single string or a string list of pathnames of files to checkout in perforce using the 'p4 add' command

See also:

where_is_p4

Parameters

- **files** – list or string object of file(s) to add
- **verbose** – If True, print the command line and warnings

Returns Zero if no error, non-zero on error

buildutils.perforce_opened

burger.buildutils.**perforce_opened**(*files=None*, *verbose=False*)

Get the list of opened files in Perforce.

Check perforce if any files are opened and if so, return the list of files in Perforce format that are currently opened.

See also:

where_is_p4

Parameters

- **files** – List of files or directories to check, None for all.
- **verbose** – If True, print the command line and warnings.

Returns List of opened files, can be empty if no files are opened.

buildutils.run_command

burger.buildutils.**run_command**(*args*, *working_dir=None*, *quiet=False*, *capture_stdout=False*, *capture_stderr=False*)

Execute a program and capture the return code and text output.

Pass a command line formatted for the current shell and then this function will execute that command and capture both stdout and stderr if desired.

Note: The first parameter is passed to subprocess.Popen() as is.

Parameters

- **args** – List of command line entries, starting with the program pathname
- **working_dir** – Directory to set before executing command
- **quiet** – Set to True if errors should not be printed
- **capture_stdout** – Set to True if stdout is to be captured
- **capture_stderr** – Set to True if stderr is to be captured

Returns The return error_code, stdout, stderr

buildutils.make_version_header

burger.buildutils.**make_version_header**(*working_dir*, *outputfilename*, *verbose=False*)

Create a C header with the perforce version.

This function assumes version control is with perforce!

Get the last change list and create a header with this information (Only modify the output file if the contents have changed)

C++ defines are declared for P4_CHANGELIST, P4_CHANGEDATE, P4_CHANGETIME, P4_CLIENT P4_USER

Parameters

- **working_dir** – string with the path of the folder to obtain the perforce version for
- **outputfilename** – string with the path of the generated header
- **verbose** – Print perforce commands and other informational messages

Returns Zero if no error, non-zero on error

buildutils.is_codewarrior_mac_allowed

burger.buildutils.**is_codewarrior_mac_allowed**()

Return True if this machine can run Codewarrior for Mac OS Carbon.

Test first if the host platform is a mac, and if so, test if it's capable of running Mac OS Carbon Codewarrior 9 or 10

See also:

strutils.host_machine

Returns True if CodeWarrior for Mac OS can be run on this Macintosh

buildutils.import_py_script

burger.buildutils.**import_py_script**(*file_name*, *module_name=None*)
Manually load in a python file.

Load in a python script from disk and parse it, creating a .pyc file if needed and reading from a .pyc file if it exists.

See also:

run_py_script

Note: The module returned will not be present in the sys.modules cache, this is by design to allow python files with the same name to be loaded from different directories without creating a cache collision

---

Parameters

- **file_name** – Name of the file to load
- **module_name** – Name of the loaded module for __name__

Returns The imported python script object

buildutils.run_py_script

burger.buildutils.**run_py_script**(*file_name*, *function_name=None*, *arg=None*)
Manually load and run a function in a python file.

Load in a python script from disk and execute a specific function. Returns the value returned from the loaded script.

See also:

import_py_script

Note: The script will not be added to the module cache.

Parameters

- **file_name** – Name of the file to load
- **function_name** – Name of the function in the file to call
- **arg** – Argument to pass to the function

Returns The value returned from the python script.

4.3.5 Clean Helpers

cleanutils.clean_xcode

burger.cleanutils.**clean_xcode**(*path*, *recursive=False*)
Scan for XCode project folders and perform a clean.

Scan the current folder and for every folder that is an XCode project, remove all user files from the folder

See also:

clean_codeblocks

See also:

fileutils.clean_files

See also:

fileutils.clean_directories

Parameters

- **path** – Directory to begin scanning
- **recursive** – Boolean if recursive clean is desired

cleanutils.clean_codeblocks

burger.cleanutils.**clean_codeblocks**(*path*, *recursive=False*)
Scan for Codeblocks project files and perform a clean.

Scan the current folder and for every codeblocks project file, remove all .depend and .layout files from the folder

See also:

clean_xcode

Parameters

- **path** – Directory to begin scanning
- **recursive** – Boolean if recursive clean is desired

cleanutils.clean_setup_py

burger.cleanutils.**clean_setup_py**(*path*, *recursive=False*)
Scan for setup.py files and perform a clean.

Scan the current folder and if the file setup.py was found, remove the folders dist, build, _build, .tox, .pytestcache and *.egg-info

See also:

clean_xcode

See also:

clean_codeblocks

Parameters

- **path** – Directory to begin scanning
- **recursive** – Boolean if recursive clean is desired

4.3.6 Windows Functions

windowsutils.find_visual_studios

burger.windowsutils.**find_visual_studios**(*refresh=False*)

Find every Windows SDK from 5.0 and higher.

This function may take some time if multiple copies of Visual Studio are installed on the machine. For speed, the results are cached and the cache is used on subsequent calls.

More info is here Searching for Visual Studio

Note: This function will return an empty list on macOS and pure Linux. It has been tested on Windows, Cygwin, MSYS2 and Ubuntu Windows Subsystem for Linux.

Parameters **refresh** – Force the cache to be reset if True.

Returns list of WindowsSDKInstance for every SDK found.

4.4 True or False

This set of functions are used to convert a value that resolves to `True` or `False` to the strings "True" or "False". However, due to the need for some output streams to require a specific form of case, these functions will output the strings according to these rules. The function `burger.strutils.string_to_bool()` is helpful in converting a larger number of strings and numbers into a boolean.

`burger.strutils.truefalse()` outputs "true" or "false"

`burger.strutils.TRUEFALSE()` outputs "TRUE" or "FALSE"

`burger.strutils.TrueFalse()` outputs "True" or "False"

4.4.1 True False code example

```
# Prints "true"
print(burger.strutils.truefalse(True))

# Prints "FALSE"
print(burger.strutils.TRUEFALSE(False))

# Prints "True"
print(burger.strutils.TrueFalse(True))
```

(continues on next page)

(continued from previous page)

```
# Prints "True"
t = burger.strutils.string_to_bool(1.34)
print(burger.strutils.TrueFalse(t))
```

4.5 WSL, Cygwin, MSYS support

Burger for Python supports the Windows Subsystem for Linux (WSL), Cygwin and MSYS2 when attempting to execute or locate tools in the underlying Windows host. One of the main issues is the varied ways a Linux pathname is mapped to Windows.

When `burger.strutils` is loaded, it will perform tests to determine which python platform it's running under and will enable specialized code or run tools to properly handle pathname conversion. For WSL, the tool used is `wslpath` and for Cygwin and MSYS2 the tool is `cygpath`.

4.5.1 Pathname examples

If the path for windows looks like this, `C:\Windows\Notepad.exe`, it will look like this on these platforms.

- WSL `/mnt/c/Windows/Notepad.exe`
- Cygwin `/cygdrive/c/Windows/Notepad.exe`
- MSYS2 `/c/Windows/Notepad.exe`

4.5.2 Pathname translation

If the `burger` library is running under WSL, Cygwin or MSYS2, the function `wslwinreg.convert_to_windows_path()` will convert a Linux style path into a Windows path and `wslwinreg.convert_from_windows_path()` will convert from a Windows path to a Linux style path. If burger is not running under these environments, these pathname translators will return the input without modification.

4.5.3 Environment flags

The flag is set to `True` if `burger` is running on the named platform, otherwise it will be `False`.

- `burger.strutils.IS_MACOSX`
- `burger.strutils.IS_WINDOWS`
- `burger.strutils.IS_LINUX`
- `burger.strutils.IS_CYGWIN`
- `burger.strutils.IS_MSYS`
- `burger.strutils.IS_WSL`

`burger.strutils.IS_WINDOWS_HOST` is `True` if the platform can execute Windows executables natively (Without Wine).

4.5.4 PATHEXT

`PATHEXT` is an environment variable present on Windows systems that presents a list of acceptable suffixes for executables. `burger.buildutils.get_path_ext()` will obtain the list of extensions available taking into account the differences of MSYS2, Cygwin, WSL and Windows. This function will return an empty list for macOS and Linux. WSL is a special case, since both native Linux and Windows .exe files can be executed, so the list will only have the suffixes for Windows executables.

4.6 Searching for Visual Studio

The function `burger.windowsutils.find_visual_studios()` will return a list of `burger._vsinstance.WindowsSDKInstance` and `burger._vsinstance.VisualStudioInstance` objects. This function will work on Windows, Cygwin, MSYS2 and Windows Subsystem for Linux. Other platforms will return an empty list.

4.6.1 VisualStudioInstance

`name` will be one of these entries.

- Microsoft Visual Studio .NET 2003
- Microsoft Visual Studio 2005
- Microsoft Visual Studio 2008
- Microsoft Visual Studio 2010
- Microsoft Visual Studio 2012
- Microsoft Visual Studio 2013
- Microsoft Visual Studio 2015
- Microsoft Visual Studio 2017
- Microsoft Visual Studio 2019
- Microsoft Visual Studio 2022

`version` will be a string in the form of "16.8.30907.101"

`version_info` is a tuple in the form of (16, 8, 30907, 101). This is most useful in doing a numeric version comparison.

`path` is the root path of the copy of Visual Studio such as "C:\Program Files (x86)\Microsoft Visual Studio 14.0"

`known_paths` is a dict with full pathnames of a executable. A suffix is needed to denote the requested CPU. The suffixes are _x86, _x64, _arm and _arm64.

Note: If the key is missing, the binary, such as `msbuild.exe` may not be available or installed in that version of Visual Studio.

- `devenv.exe_x86` = "c:\Program Files (x86)\Microsoft Visual Studio .NET 2003\Common7\IDE\devenv.exe"
- `vcvarsall.bat` = "c:\Program Files (x86)\Microsoft Visual Studio .NET 2003\Common7\Tools\vsvars32.bat"
- `cl.exe_x86` = "c:\Program Files (x86)\Microsoft Visual Studio .NET 2003\Vc7\bin\cl.exe"
- `link.exe_x86` = "c:\Program Files (x86)\Microsoft Visual Studio .NET 2003\Vc7\bin\link.exe"
- `lib.exe_x86` = "c:\Program Files (x86)\Microsoft Visual Studio .NET 2003\Vc7\bin\lib.exe"
- `msbuild.exe_x86` "C:\Program Files (x86)\MSBuild\14.0\bin\msbuild.exe"

4.6.2 WindowsSDKInstance

`name` will be one of these entries.

- Windows 5 SDK
- Windows 6 SDK
- Windows 7 SDK
- Windows 8 SDK
- Windows 10 SDK

`version` will be a string in the form of "10.0.10150.0"

`version_info` is a tuple in the form of (10, 0, 10150, 0). This is most useful in doing a numeric version comparison. The first number will match the actual SDK version such as 5, 6, 7, 8 or 10.

`path` is the root path of the copy of Visual Studio such as "C:\Program Files (x86)\Windows Kits\8.0" or "C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A"

`known_paths` is a dict with full pathnames of a folder, library or executable. A suffix is needed to denote the requested CPU. The suffixes are _x86, _x64, _arm and _arm64.

Note: If the key is missing, the binary, such as `signtool.exe` may not be available or installed in that version of Visual Studio.

Folders

- `WinSDK.ucrt` = "C:\Program Files (x86)\Windows Kits\10\Include\10.0.18362.0\ucrt"
- `WinSDK.um` = "C:\Program Files (x86)\Windows Kits\10\Include\10.0.18362.0\um"
- `WinSDK.shared` = "C:\Program Files (x86)\Windows Kits\10\Include\10.0.18362.0\shared"
- `WinSDK.winrt` = "C:\Program Files (x86)\Windows Kits\10\Include\10.0.18362.0\winrt"
- `WinSDK.cppwinrt` == "C:\Program Files (x86)\Windows Kits\10\Include\10.0.18362.0\cppwinrt"

Library folders

- `WinSDK.libucrt_x86` = "C:\Program Files (x86)\Windows Kits\10\Lib\10.0.18362.0\ucrt\x86"
- `WinSDK.libucrt_x64` = "C:\Program Files (x86)\Windows Kits\10\Lib\10.0.18362.0\ucrt\x64"
- `WinSDK.libucrt_arm` = "C:\Program Files (x86)\Windows Kits\10\Lib\10.0.18362.0\ucrt\arm"
- `WinSDK.libucrt_arm64` = "C:\Program Files (x86)\Windows Kits\10\Lib\10.0.18362.0\ucrt\arm64"
- `WinSDK.lib_x86` = "C:\Program Files (x86)\Windows Kits\10\Lib\10.0.18362.0\um\x86"
- `WinSDK.lib_x64` = "C:\Program Files (x86)\Windows Kits\10\Lib\10.0.18362.0\um\x64"
- `WinSDK.lib_arm` = "C:\Program Files (x86)\Windows Kits\10\Lib\10.0.18362.0\um\arm"
- `WinSDK.lib_arm64` = "C:\Program Files (x86)\Windows Kits\10\Lib\10.0.18362.0\um\arm64"

Executables

- rc.exe_x86 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\x86\rc.exe"
- rc.exe_x64 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\x64\rc.exe"
- rc.exe_arm64 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\arm64\rc.exe"
- signtool.exe_x86 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\x86\signtool.exe"
- signtool.exe_x64 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\x64\signtool.exe"
- signtool.exe_arm = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\arm\signtool.exe"
- signtool.exe_arm64 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\arm64\signtool.exe"
- makecat.exe_x86 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\x86\makecat.exe"
- makecat.exe_x64 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\x64\makecat.exe"
- makecat.exe_arm64 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\arm64\makecat.exe"
- midl.exe_x86 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\x86\midl.exe"
- midl.exe_x64 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\x64\midl.exe"
- midl.exe_arm64 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\arm64\midl.exe"
- mc.exe_x86 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\x86\mc.exe"
- mc.exe_x64 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\x64\mc.exe"
- mc.exe_arm64 = "C:\Program Files (x86)\Windows Kits\10\bin\10.0.18362.0\arm64\mc.exe"

4.7 Data type validators

These classes are used in conjunction with a python class definition to create class attributes that will throw an exception if invalid data was stored. It allows for ensuring that data stored in these attributes are of only the type desired.

- burger.validators.BooleanProperty
- burger.validators.IntegerProperty
- burger.validators.StringProperty
- burger.validators.StringListProperty
- burger.validators.EnumProperty
- burger.validators.NoneProperty

4.7.1 How to use

They are implemented as class immutable attributes with a member variable definition. It is preferred to use an underscore as the first character of the attribute to denote it is a hidden variable.

```
# Class with validated attributes
from burger.validators import *

class foo(object):
```

(continues on next page)

(continued from previous page)

```
    # _bool_var is the class instance storage name
    bool_var = BooleanProperty('_bool_var')

    # Note the name can be any variable, but the use
    # of an alternate name may be an invitation for
    # its direct use and bypassing the validator
    int_var = IntegerProperty('int_var2')

    def __init__(self):
        # Set defaults
        self.bool_var = True

        # This will throw an exception due to
        # validation failure
        # self.bool_var = 'string'

        self.int_var = 42


a = foo()
# Will set to False
a.bool_var = False
# Will set to True (Due to 9 being non zero)
a.bool_var = 9
# Will set to False
a.bool_var = 'False'
# Will throw an exception
a.bool_var = 'foobar'
# Will set to None
a.bool_var = None

# Will set to 8675309
a.int_var = 8675309
# Will set to 13
a.int_var = '13'
# Will throw an exception
a.int_var = 'USA USA'

# Will bypass the validators and set the values to strings
a._bool_var = 'NSA NSA'
a.int_var2 = 'USA USA'
```

4.8 License

4.8.1 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

B

C

I

L

P

U

V