pathvalidate Documentation

Release 3.0.0

Tsuyoshi Hombashi

May 22, 2023

TABLE OF CONTENTS

CHAPTER

ONE

PATHVALIDATE

1.1 Summary

pathvalidate is a Python library to sanitize/validate a string such as filenames/file-paths/etc.

1.2 Features

- **Sanitize/Validate a string as a:**
  - file name
  - file path
- file name/path argument validator/sanitizer for `argparse` and `click`
- **Multi platform support:**
  - sanitize/validate file names/paths for a specific platform (`Linux`/`Windows`/`macOS`/`Posix`) or `universal` (platform independent)
- Multibyte character support

CHAPTER

TWO

INSTALLATION

2.1 Installation: pip

```
pip install pathvalidate
```

2.2 Installation: conda

```
conda install -c thombashi pathvalidate
```

2.3 Installation: apt

```
sudo add-apt-repository ppa:thombashi/ppa
sudo apt update
sudo apt install python3-pathvalidate
```

CHAPTER

THREE

DEPENDENCIES

Python 3.6+ no external dependencies.

CHAPTER

FOUR

EXAMPLES

4.1 Sanitize a filename

The *sanitize_filename()* function returns a filename which replaced invalid character(s) for a filename within the argument.

Sample Code

```
from pathvalidate import sanitize_filename

fname = "fi:l*e/p\"a?t>h|.t<xt"
print(f"{fname} -> {sanitize_filename(fname)}\n")

fname = "\0_a*b:c<d>e%f/(g)h+i_0.txt"
print(f"{fname} -> {sanitize_filename(fname)}\n")
```

Output

```
fi:l*e/p"a?t>h|.t<xt -> filepath.txt

_a*b:c<d>e%f/(g)h+i_0.txt -> _abcde%f(g)h+i_0.txt
```

The default target `platform` is `universal`. i.e. the sanitized file name is valid for any platform.

4.2 Sanitize a filepath

The *sanitize_filepath()* function returns a filepath which replaced invalid character(s) for a filepath within the argument.

Sample Code

```
from pathvalidate import sanitize_filepath

fpath = "fi:l*e/p\"a?t>h|.t<xt"
print(f"{fpath} -> {sanitize_filepath(fpath)}\n")

fpath = "\0_a*b:c<d>e%f/(g)h+i_0.txt"
print(f"{fpath} -> {sanitize_filepath(fpath)}\n")
```

Output

```
fi:l*e/p"a?t>h|.t<xt -> file/path.txt

_a*b:c<d>e%f/(g)h+i_0.txt -> _abcde%f/(g)h+i_0.txt
```

4.3 Replace symbols

The *replace_symbol()* function returns a string which replaced symbol(s) within the argument.

Sample Code

```
from pathvalidate import replace_symbol

name = "\0_a*b:c<d>e%f/(g)h+i_0.txt"
print(f"{name} -> {replace_symbol(name)}")
```

Output

```
_a*b:c<d>e%f/(g)h+i_0.txt -> abcdefghi0txt
```

4.4 Validate a filename

The *validate_filename()* function raise `ValueError` if the name includes invalid character(s) for a filename.

Sample Code

```
import sys
from pathvalidate import ValidationError, validate_filename

try:
    validate_filename("fi:l*e/p\"a?t>h|.t<xt")
except ValidationError as e:
    print(f"{e}\n", file=sys.stderr)

try:
    validate_filename("COM1")
except ValidationError as e:
    print(f"{e}\n", file=sys.stderr)
```

Output

```
invalid char found: invalids=(':', '*', '/', '"', '?', '>', '|', '<'), value='fi:l*e/p"a?t>h|.t<xt', reason=INVALID_CHARACTER, target-platform=Windows

'COM1' is a reserved name, reason=RESERVED_NAME, target-platform=universal
```

4.5 Validate a file path

The *validate_filepath()* function raise ValueError if the name includes invalid character(s) for a file path.

Sample Code

```
import sys
from pathvalidate import ValidationError, validate_filepath

try:
    validate_filepath("fi:l*e/p\"a?t>h|.t<xt")
except ValidationError as e:
    print(e, file=sys.stderr)
```

Output

```
invalid char found: invalids=(':', '*', '"', '?', '>', '|', '<'), value=
↪'fi:l*e/p"a?t>h|.t<xt', reason=INVALID_CHARACTER, target-platform=Windows
```

4.6 Check a filename

is_valid_filename() function returns True if a filename is valid for a specified platform.

Sample Code

```
from pathvalidate import is_valid_filename, sanitize_filename

fname = "fi:l*e/p\"a?t>h|.t<xt"
print(f"is_valid_filename('{fname}') return {is_valid_filename(fname)}\n")

sanitized_fname = sanitize_filename(fname)
print(f"is_valid_filename('{sanitized_fname}') return {is_valid_
↪filename(sanitized_fname)}\n")
```

Output

```
is_valid_filename('fi:l*e/p"a?t>h|.t<xt') return False

is_valid_filename('filepath.txt') return True
```

4.7 Check a filepath

is_valid_filepath() function returns True if a filepath is valid for a specified platform.

Sample Code

```
from pathvalidate import is_valid_filepath, sanitize_filepath

fpath = "fi:l*e/p\"a?t>h|.t<xt"
print(f"is_valid_filepath('{fpath}') return {is_valid_filepath(fpath)}\n")
```

(continues on next page)

(continued from previous page)

```
sanitized_fpath = sanitize_filepath(fpath)
print(f"is_valid_filepath('{sanitized_fpath}') return {is_valid_
↪filepath(sanitized_fpath)}\n")
```

Output

```
is_valid_filepath('fi:l*e/p"a?t>h|.t<xt') return False

is_valid_filepath('file/path.txt') return True
```

4.8 filename/filepath validator for argparse

Sample Code

```
from argparse import ArgumentParser

from pathvalidate.argparse import validate_filename_arg, validate_filepath_
↪arg

parser = ArgumentParser()
parser.add_argument("--filepath", type=validate_filepath_arg)
parser.add_argument("--filename", type=validate_filename_arg)
options = parser.parse_args()

if options.filename:
    print("filename: {}".format(options.filename))

if options.filepath:
    print("filepath: {}".format(options.filepath))
```

Output

```
$ ./examples/argparse_validate.py --filename eg
filename: eg
$ ./examples/argparse_validate.py --filepath e?g
usage: argparse_validate.py [-h] [--filepath FILEPATH] [--filename␣
↪FILENAME]
argparse_validate.py: error: argument --filepath: invalid char found:␣
↪invalids=('?'), value='e?g', reason=INVALID_CHARACTER, target-
↪platform=Windows
```

Note: `validate_filepath_arg` consider `platform` as of `"auto"` if the input is an absolute file path.

4.9 filename/filepath sanitizer for argparse

Sample Code

```
from argparse import ArgumentParser

from pathvalidate.argparse import sanitize_filename_arg, sanitize_filepath_
↪arg


parser = ArgumentParser()
parser.add_argument("--filename", type=sanitize_filename_arg)
parser.add_argument("--filepath", type=sanitize_filepath_arg)
options = parser.parse_args()

if options.filename:
    print("filename: {}".format(options.filename))

if options.filepath:
    print("filepath: {}".format(options.filepath))
```

Output

```
$ ./examples/argparse_sanitize.py --filename e/g
filename: eg
```

Note: `sanitize_filepath_arg` is set platform as `"auto"`.

4.10 filename/filepath validator for click

Sample Code

```
import click

from pathvalidate.click import validate_filename_arg, validate_filepath_arg


@click.command()
@click.option("--filename", callback=validate_filename_arg)
@click.option("--filepath", callback=validate_filepath_arg)
def cli(filename, filepath):
    if filename:
        click.echo("filename: {}".format(filename))
    if filepath:
        click.echo("filepath: {}".format(filepath))


if __name__ == "__main__":
    cli()
```

Output

```
$ ./examples/click_validate.py --filename ab
filename: ab
$ ./examples/click_validate.py --filepath e?g
Usage: click_validate.py [OPTIONS]

Error: Invalid value for "--filepath": invalid char found: invalids=('?'),␣
↪value='e?g', reason=INVALID_CHARACTER, target-platform=Windows
```

4.11 filename/filepath sanitizer for click

Sample Code

```
import click

from pathvalidate.click import sanitize_filename_arg, sanitize_filepath_arg


@click.command()
@click.option("--filename", callback=sanitize_filename_arg)
@click.option("--filepath", callback=sanitize_filepath_arg)
def cli(filename, filepath):
    if filename:
        click.echo("filename: {}".format(filename))
    if filepath:
        click.echo("filepath: {}".format(filepath))


if __name__ == "__main__":
    cli()
```

Output

```
$ ./examples/click_sanitize.py --filename a/b
filename: ab
```

CHAPTER

FIVE

REFERENCE

5.1 Errors

exception pathvalidate.error.**ErrorReason**(*value*)

Bases: Enum

Validation error reasons.

exception pathvalidate.error.**ValidationError**(**args*, ***kwargs*)

Bases: ValueError

Exception class of validation errors.

reason

The cause of the error.

Returns:
ErrorReason:

5.2 Functions

5.2.1 File name validation/sanitization

pathvalidate.**validate_filename**(*filename: PathType*, *platform: Optional[PlatformType] = None*, *min_len: int = 1*, *max_len: int = 255*, *fs_encoding: Optional[str] = None*, *check_reserved: bool = True*) → None

Verifying whether the filename is a valid file name or not.

Parameters

- **filename** – Filename to validate.
- **platform** – Target platform name of the filename.

  Valid specifiers are follows (case-insensitive):

  - "Linux"
  - "Windows"
  - "macOS"
  - "auto": automatically detect the execution platform
  - "POSIX"

  - "universal"/None: platform independent. note that absolute paths cannot specify this.

  Defaults to None.

- **min_len** – Minimum byte length of the filename. The value must be greater or equal to one. Defaults to 1.
- **max_len** – Maximum byte length of the filename. The value must be lower than:
  - Linux: 4096
  - macOS: 1024
  - Windows: 260
  - universal: 260

  Defaults to 255.
- **fs_encoding** – Filesystem encoding that used to calculate the byte length of the filename. If None, get the value from the execution environment.
- **check_reserved** – If True, check reserved names of the platform.

Raises

- *ValidationError* **(ErrorReason.INVALID_LENGTH)** – If the filename is longer than max_len characters.
- *ValidationError* **(ErrorReason.INVALID_CHARACTER)** – If the filename includes invalid character(s) for a filename: /, \\0. The following characters are also invalid for Windows platforms: \, :, *, ?, ", <, >, |, \t, \n, \r, \x0b, \x0c.
- *ValidationError* **(ErrorReason.RESERVED_NAME)** – If the filename equals reserved name by OS. Windows reserved name is as follows: "CON", "PRN", "AUX", "NUL", "COM[1-9]", "LPT[1-9]".

Example

Validate a filename

See also:

Naming Files, Paths, and Namespaces - Win32 apps | Microsoft Docs

pathvalidate.**sanitize_filename**(*filename: PathType*, *replacement_text: str = ''*, *platform: Optional[PlatformType] = None*, *max_len: Optional[int] = 255*, *fs_encoding: Optional[str] = None*, *check_reserved: bool = True*, *null_value_handler: Optional[Callable[[ValidationError], str]] = None*, *validate_after_sanitize: bool = False*) → PathType

Make a valid filename from a string.

To make a valid filename, the function does the following:

- Replace invalid characters as file names included in the filename with the replacement_text. Invalid characters are:
  - unprintable characters
  - /, \\0
  - for Windows (or universal) only: \, :, *, ?, ", <, >, |, \t, \n, \r, \x0b, \x0c

- Append underscore ("_") at the tail of the return value if a sanitized name is one of the reserved names by operating systems (only when `check_reserved` is `True`).

Parameters

- **filename** – Filename to sanitize.
- **replacement_text** – Replacement text for invalid characters. Defaults to "".
- **platform** – Target platform name of the filename.

  Valid specifiers are follows (case-insensitive):

  - "Linux"
  - "Windows"
  - "macOS"
  - "auto": automatically detect the execution platform
  - "POSIX"
  - "universal"/None: platform independent. note that absolute paths cannot specify this.

  Defaults to None.
- **max_len** – Maximum byte length of the `filename`. Truncate the name length if the `filename` length exceeds this value. Defaults to 255.
- **fs_encoding** – Filesystem encoding that used to calculate the byte length of the filename. If None, get the value from the execution environment.
- **check_reserved** – If `True`, sanitize reserved names of the `platform`.
- **null_value_handler** – Function called when a value after sanitization is an empty string. Defaults to `pathvalidate.handler.return_null_string()` function that return an empty string.
- **validate_after_sanitize** – Execute validation after sanitization to the file name.

Returns
Sanitized filename.

Return type
Same type as the `filename` (str or PathLike object)

Raises
ValueError – If the `filename` is an invalid filename.

Example

Sanitize a filename

5.2.2 Check a file name

pathvalidate.**is_valid_filename**(*filename: PathType*, *platform: Optional[PlatformType] = None*, *min_len: int = 1*, *max_len: Optional[int] = None*, *fs_encoding: Optional[str] = None*, *check_reserved: bool = True*) → bool

Check whether the `filename` is a valid name or not.

Parameters

filename – A filename to be checked.

Example

Check a filename

See also:

validate_filename()

5.2.3 File path validate/sanitize

pathvalidate.**validate_filepath**(*file_path: PathType*, *platform: Optional[PlatformType] = None*, *min_len: int = 1*, *max_len: Optional[int] = None*, *fs_encoding: Optional[str] = None*, *check_reserved: bool = True*) → None

Verifying whether the `file_path` is a valid file path or not.

Parameters

- **file_path** – File path to validate.
- **platform** – Target platform name of the file path.
- **min_len** – Minimum byte length of the `file_path`. The value must be greater or equal to one. Defaults to 1.
- **max_len** – Maximum byte length of the `file_path`. If the value is `None` or minus, automatically determined by the `platform`:
  - `Linux`: 4096
  - `macOS`: 1024
  - `Windows`: 260
  - `universal`: 260
- **fs_encoding** – Filesystem encoding that used to calculate the byte length of the file path. If `None`, get the value from the execution environment.
- **check_reserved** – If `True`, check reserved names of the `platform`.

Raises

- *ValidationError* **(ErrorReason.INVALID_CHARACTER)** – If the `file_path` includes invalid char(s): `\\0`. The following characters are also invalid for Windows platforms: `:`, `*`, `?`, `"`, `<`, `>`, `|`, `\t`, `\n`, `\r`, `\x0b`, `\x0c`
- *ValidationError* **(ErrorReason.INVALID_LENGTH)** – If the `file_path` is longer than `max_len` characters.
- *ValidationError* – If `file_path` include invalid values.

Example

Validate a file path

See also:

Naming Files, Paths, and Namespaces - Win32 apps | Microsoft Docs

pathvalidate.**sanitize_filepath**(*file_path: PathType*, *replacement_text: str = ''*, *platform: Optional[PlatformType] = None*, *max_len: Optional[int] = None*, *fs_encoding: Optional[str] = None*, *check_reserved: bool = True*, *null_value_handler: Optional[Callable[[ValidationError], str]] = None*, *normalize: bool = True*, *validate_after_sanitize: bool = False*) → PathType

Make a valid file path from a string.

To make a valid file path, the function does the following:

- replace invalid characters for a file path within the `file_path` with the `replacement_text`. Invalid characters are as follows:
  - unprintable characters
  - `\\0`
  - for Windows (or universal) only: `:`, `*`, `?`, `"`, `<`, `>`, `|`, `\t`, `\n`, `\r`, `\x0b`, `\x0c`
- Append underscore (`"_"`) at the tail of the name if sanitized name is one of the reserved names by operating systems (only when `check_reserved` is `True`).

Parameters

- **file_path** – File path to sanitize.
- **replacement_text** – Replacement text for invalid characters. Defaults to `""`.
- **platform** – Target platform name of the file path.
- **max_len** – Maximum byte length of the file path. Truncate the path if the value length exceeds the *max_len*. If the value is `None` or minus, `max_len` will automatically determined by the `platform`:
  - `Linux`: 4096
  - `macOS`: 1024
  - `Windows`: 260
  - `universal`: 260
- **fs_encoding** – Filesystem encoding that used to calculate the byte length of the file path. If `None`, get the value from the execution environment.
- **check_reserved** – If `True`, sanitize reserved names of the `platform`.
- **null_value_handler** – Function called when a value after sanitization is an empty string. Defaults to `pathvalidate.handler.return_null_string()` that just return `""`.
- **normalize** – If `True`, normalize the the file path.
- **validate_after_sanitize** – Execute validation after sanitization to the file path.

Returns
Sanitized filepath.

Return type
Same type as the argument (str or PathLike object)

Raises
ValueError – If the `file_path` is an invalid file path.

Example

Sanitize a filepath

5.2.4 Check a file path

`pathvalidate.`**`is_valid_filepath`**(*file_path: PathType*, *platform: Optional[PlatformType] = None*, *min_len: int = 1*, *max_len: Optional[int] = None*, *fs_encoding: Optional[str] = None*, *check_reserved: bool = True*) → bool

Check whether the `file_path` is a valid name or not.

Parameters
file_path – A filepath to be checked.

Example

Check a filepath

See also:

validate_filepath()

5.2.5 Symbol validate/sanitize

`pathvalidate.`**`validate_symbol`**(*text: str*) → None

Verifying whether symbol(s) included in the `text` or not.

Parameters
text – Input text to validate.

Raises
ValidationError **(ErrorReason.INVALID_CHARACTER)** – If symbol(s) included in the `text`.

`pathvalidate.`**`replace_symbol`**(*text: str*, *replacement_text: str = ''*, *exclude_symbols: Sequence[str] = []*, *is_replace_consecutive_chars: bool = False*, *is_strip: bool = False*) → str

Replace all of the symbols in the `text`.

Parameters

- **text** – Input text.
- **replacement_text** – Replacement text.
- **exclude_symbols** – Symbols that exclude from the replacement.
- **is_replace_consecutive_chars** – If `True`, replace consecutive multiple `replacement_text` characters to a single character.
- **is_strip** – If `True`, strip `replacement_text` from the beginning/end of the replacement text.

Returns
A replacement string.

Example

Replace symbols

CHAPTER

SIX

CHANGELOG

https://github.com/thombashi/pathvalidate/releases

CHAPTER

SEVEN

SPONSORS

Become a sponsor

CHAPTER

EIGHT

INDICES AND TABLES

- genindex

CHAPTER

NINE

LINKS

- GitHub repository
- Issue tracker
- pip: A tool for installing Python packages

CHAPTER

TEN

INDICES AND TABLES

- genindex

INDEX