Senf

Jul 11, 2018

Contents

CHAPTER 1

1.3.4 - 2018-01-23

- *fsn2bytes()* and *bytes2fsn()* now default to “wtf-8” for the Windows path encoding instead of having no default.

CHAPTER 2

1.3.3 - 2018-01-03

- Restore WinXP support
- Fix some warnings with Python 3.6

CHAPTER 3

1.3.2 - 2017-11-05

- Tests: Fix some errors with newer pytest and make the test suite work on native Windows.

CHAPTER 4

1.3.1 - 2017-07-29

- Fixed missing normalization with `path2fsn()` on Linux + Python 3

CHAPTER 5

1.3.0 - 2017-07-28

- New *supports_ansi_escape_codes()*
- New *fsn2norm()* (+ all API now returns normalized fsnative)
- *fsn2uri()*: various fixes

CHAPTER 6

1.2.2 - 2016-12-18

- *uri2fsn*: improve error handling on unescaped URIs #4

CHAPTER 7

1.2.1 - 2016-12-07

- `isinstance(path, fsnative)` now checks the value as well. If True passing the instance to *path2fsn* will never fail.

CHAPTER 8

1.2.0 - 2016-12-06

- *fsnative*: safeguard against containing null bytes. All operations converting to *fsnative* will now fail if the result would contain null bytes. This means passing *fsnative* to functions like `open()` is now always safe.

CHAPTER 9

1.1.0 - 2016-12-05

- *print_()*: Don't ignore `flush` in Windows redirect mode
- *argv*: Forwards changes to `sys.argv` #2
- *environ*: Forwards changes to `os.environ` #2
- *environ*: Handle case insensitive env vars on Windows
- *fsn2text()*: Add a `strict` mode
- *fsn2uri()*: Always return *text*
- *fsn2bytes()*: Merge surrogate pairs under Python 3 + Windows
- *fsn2bytes()*: Support `utf-16-be` under Python 2.7/3.3

CHAPTER 10

1.0.1 - 2016-10-25

- Python 2.6 support removed
- *print_()*: allow `None` for `end`, `sep` and `file` arguments
- *print_()*: always output utf-8 when redirected on Windows

CHAPTER 11

1.0.0 - 2016-09-09

- First stable release

CHAPTER 12

0.4.0 - 2016-09-07

- Support paths with surrogates under Windows

CHAPTER 13

0.3.0 - 2016-09-03

- Support `__fspath__` in *path2fsn()*. See PEP 519 for details.
- Rename fsn2uri_ascii to *fsn2uri()*, remove the later.
- Fix *fsn2uri()* output on Windows for certain unicode ranges.
- Add *expandvars()*

CHAPTER 14

0.2.0 - 2016-08-25

- *input_()*: Add Windows Unicode support

CHAPTER 15

0.1.0 - 2016-08-22

- Initial release

CHAPTER 16

Tutorial

There are various ways to create fsnative instances:

```
# create from unicode text
>>> senf.fsnative(u"foo")
'foo'

# create from some serialized format
>>> senf.bytes2fsn(b"foo", "utf-8")
'foo'

# create from an URI
>>> senf.uri2fsn("file:///foo")
'/foo'

# create from some Python path-like
>>> senf.path2fsn(b"foo")
'foo'
```

You can mix and match the fsnative type with ASCII str on all Python versions and platforms:

```
>>> senf.fsnative(u"foo") + "bar"
'foobar'
>>> senf.fsnative(u"foo").endswith("foo")
True
>>> "File: %s" % senf.fsnative(u"foo")
'File: foo'
```

Now that we have a *fsnative*, what can we do with it?

```
>>> path = senf.fsnative(u"/foo")

# We can print it
>>> senf.print_(path)
/foo
```

(continues on next page)

(continued from previous page)

```
# We can convert it to text for our favorite GUI toolkit
>>> senf.fsn2text(path)
'/foo'

# We can convert it to an ASCII only URI
>>> senf.fsn2uri(path)
'file:///foo'

# We can serialize the path so we can save it somewhere
>>> senf.fsn2bytes(path, "utf-8")
b'/foo'
```

The functions in the stdlib usually return the same type as was passed in. If we pass in a *fsnative* to os.listdir, we get one back as well.

```
>>> files = os.listdir(senf.fsnative(u"."))
>>> isinstance(files[0], senf.fsnative)
True
```

In some cases the stdlib functions don't take arguments and always return the same type. For those cases Senf provide alternative implementations.

```
>>> isinstance(senf.getcwd(), senf.fsnative)
True
```

A similar problem arises with stdlib collections. Senf provides alternatives for sys.argv and os.environ.

```
>>> isinstance(senf.argv[0], senf.fsnative)
True
>>> isinstance(senf.environ["PATH"], senf.fsnative)
True
```

Also for os.environ related functions.

```
>>> isinstance(senf.getenv("HOME"), fsnative)
True
>>> isinstance(senf.expanduser("~"), fsnative)
True
```

If you work with files a lot your unit tests will probably need temporary files. Senf provides wrappers for tempfile functions which always return a *fsnative*.

```
>>> senf.mkdtemp()
'/tmp/tmp26Daqo'
>>> isinstance(_, senf.fsnative)
True
```

CHAPTER 17

API Documentation

17.1 Fsnative Related

Helper functions for working with the *fsnative* type

fsnative()	Virtual path type and constructor
path2fsn()	Convert *pathlike* to *fsnative*
fsn2text()	Convert *fsnative* to *text*
text2fsn()	Convert *text* to *fsnative*
fsn2bytes()	Convert *fsnative* to *bytes*
bytes2fsn()	Convert *bytes* to *fsnative*
uri2fsn()	Convert URI to *fsnative*
fsn2uri()	Convert *fsnative* to ASCII URI
fsn2norm()	Normalize *fsnative*

17.2 Stdlib Replacements

Alternative implementations or wrappers of stdlib functions and constants. In some cases their default is changed to return an fsnative path (mkdtemp() with default arguments) or Unicode support for Windows is added (sys.argv)

environ	os.environ replacement
argv	sys.argv replacement
sep	os.sep replacement
pathsep	os.pathsep replacement
curdir	os.curdir replacement
pardir	os.pardir replacement
altsep	os.altsep replacement
extsep	os.extsep replacement
devnull	os.devnull replacement
defpath	os.defpath replacement
getcwd()	os.getcwd replacement
getenv()	os.getenv replacement
putenv()	os.putenv replacement
unsetenv()	os.unsetenv replacement
print_()	print() replacement
input_()	input() replacement
expanduser()	os.path.expanduser() replacement
expandvars()	os.path.expandvars() replacement
gettempdir()	tempfile.gettempdir() replacement
gettempprefix()	tempfile.gettempprefix() replacement
mkstemp()	tempfile.mkstemp() replacement
mkdtemp()	tempfile.mkdtemp() replacement

17.3 Misc Functions

supports_ansi_escape_codes()	if the output file supports ANSI codes

17.4 Package Related

senf.version	Version tuple
senf.version_string	Version string

---

senf.**version = (1, 3, 5)**
Tuple[int, int, int] – The version tuple (major, minor, micro)

senf.**version_string = '1.3.5'**
str – A version string

class senf.**fsnative**(*text=u""*)

Parameters **text** (text) – The text to convert to a path

Returns The new path.

Return type *fsnative*

Raises TypeError – In case something other then *text* has been passed

This type is a virtual base class for the real path type. Instantiating it returns an instance of the real path type and it overrides instance and subclass checks so that `isinstance` and `issubclass` checks work:

```
isinstance(fsnative(u"foo"), fsnative) == True
issubclass(type(fsnative(u"foo")), fsnative) == True
```

The real returned type is:

- **Python 2 + Windows:** `unicode`, with `surrogates`, without `null`
- **Python 2 + Unix:** `str`, without `null`
- **Python 3 + Windows:** `str`, with `surrogates`, without `null`
- **Python 3 + Unix:** `str`, with `surrogates`, without `null`, without code points not encodable with the locale encoding

Constructing a *fsnative* can't fail.

Passing a *fsnative* to `open()` will never lead to `ValueError` or `TypeError`.

Any operation on *fsnative* can also use the `str` type, as long as the `str` only contains ASCII and no NULL.

`senf.`**`path2fsn`**(*path*)

Parameters **`path`** (`pathlike`) – The path to convert

Returns *fsnative*

Raises

- `TypeError` – In case the type can't be converted to a *fsnative*
- `ValueError` – In case conversion fails

Returns a *fsnative* path for a *pathlike*.

`senf.`**`fsn2text`**(*path*, *strict=False*)

Parameters

- **`path`** (`fsnative`) – The path to convert
- **`strict`** (*bool*) – Fail in case the conversion is not reversible

Returns *text*

Raises

- `TypeError` – In case no *fsnative* has been passed
- `ValueError` – In case `strict` was True and the conversion failed

Converts a *fsnative* path to *text*.

Can be used to pass a path to some unicode API, like for example a GUI toolkit.

If `strict` is True the conversion will fail in case it is not reversible. This can be useful for converting program arguments that are supposed to be text and erroring out in case they are not.

Encoding with a Unicode encoding will always succeed with the result.

`senf.`**`text2fsn`**(*text*)

Parameters **`text`** (`text`) – The text to convert

Returns *fsnative*

Raises `TypeError` – In case no *text* has been passed

Takes *text* and converts it to a *fsnative*.

This operation is not reversible and can't fail.

senf.**fsn2bytes**(*path*, *encoding='utf-8'*)

Parameters

- **path** (fsnative) – The path to convert
- **encoding** (str) – encoding used for Windows

Returns *bytes*

Raises

- TypeError – If no *fsnative* path is passed
- ValueError – If encoding fails or the encoding is invalid

Converts a *fsnative* path to *bytes*.

The passed *encoding* is only used on platforms where paths are not associated with an encoding (Windows for example).

For Windows paths, lone surrogates will be encoded like normal code points and surrogate pairs will be merged before encoding. In case of utf-8 or utf-16-le this is equal to the WTF-8 and WTF-16 encoding.

senf.**bytes2fsn**(*data*, *encoding='utf-8'*)

Parameters

- **data** (bytes) – The data to convert
- **encoding** (str) – encoding used for Windows

Returns *fsnative*

Raises

- TypeError – If no *bytes* path is passed
- ValueError – If decoding fails or the encoding is invalid

Turns *bytes* to a *fsnative* path.

The passed *encoding* is only used on platforms where paths are not associated with an encoding (Windows for example).

For Windows paths WTF-8 is accepted if utf-8 is used and WTF-16 accepted if utf-16-le is used.

senf.**uri2fsn**(*uri*)

Parameters **uri** (*text* or str) – A file URI

Returns *fsnative*

Raises

- TypeError – In case an invalid type is passed
- ValueError – In case the URI isn't a valid file URI

Takes a file URI and returns a *fsnative* path

senf.**fsn2uri**(*path*)

Parameters **path** (fsnative) – The path to convert to an URI

Returns An ASCII only URI

Return type *text*

Raises

- TypeError – If no *fsnative* was passed
- ValueError – If the path can't be converted

Takes a *fsnative* path and returns a file URI.

On Windows non-ASCII characters will be encoded using utf-8 and then percent encoded.

senf.**fsn2norm**(*path*)

Parameters **path** (fsnative) – The path to normalize

Returns *fsnative*

Normalizes an fsnative path.

The same underlying path can have multiple representations as fsnative (due to surrogate pairs and variable length encodings). When concatenating fsnative the result might be different than concatenating the serialized form and then deserializing it.

This returns the normalized form i.e. the form which os.listdir() would return. This is useful when you alter fsnative but require that the same underlying path always maps to the same fsnative value.

All functions like *bytes2fsn()*, *fsnative()*, *text2fsn()* and *path2fsn()* always return a normalized path, independent of their input.

senf.**environ = {}**
Dict[*fsnative*, *fsnative*] – Like os.environ but contains unicode keys and values under Windows + Python 2.

Any changes made will be forwarded to os.environ.

senf.**argv = []**
List[*fsnative*] – Like sys.argv but contains unicode keys and values under Windows + Python 2.

Any changes made will be forwarded to sys.argv.

senf.**sep = '/'**
fsnative – Like os.sep but a *fsnative*

senf.**pathsep = ':'**
fsnative – Like os.pathsep but a *fsnative*

senf.**curdir = '.'**
fsnative – Like os.curdir but a *fsnative*

senf.**pardir = '..'**
fsnative – Like os.pardir but a fsnative

senf.**altsep = None**
fsnative or None – Like os.altsep but a *fsnative* or None

senf.**extsep = '.'**
fsnative – Like os.extsep but a *fsnative*

senf.**devnull = '/dev/null'**
fsnative – Like os.devnull but a *fsnative*

senf.**defpath = ':/bin:/usr/bin'**
fsnative – Like os.defpath but a *fsnative*

senf.**getcwd**()
Like os.getcwd but returns a *fsnative* path

Returns *fsnative*

senf.**getenv**(*key*, *value=None*)
Like os.getenv but returns unicode under Windows + Python 2

Parameters

- **key** (pathlike) – The env var to get
- **value** (*object*) – The value to return if the env var does not exist

Returns The env var or the passed value if it doesn't exist

Return type *fsnative* or object

senf.**putenv**(*key*, *value*)
Like os.putenv but takes unicode under Windows + Python 2

Parameters

- **key** (pathlike) – The env var to get
- **value** (pathlike) – The value to set

Raises ValueError

senf.**unsetenv**(*key*)
Like os.unsetenv but takes unicode under Windows + Python 2

Parameters **key** (pathlike) – The env var to unset

senf.**print_**(**objects*, *sep=None*, *end=None*, *file=None*, *flush=False*)

Parameters

- **objects** (*object*) – zero or more objects to print
- **sep** (*str*) – Object separator to use, defaults to `" "`
- **end** (*str*) – Trailing string to use, defaults to `"\n"`. If end is `"\n"` then os.linesep is used.
- **file** (*object*) – A file-like object, defaults to sys.stdout
- **flush** (*bool*) – If the file stream should be flushed

Raises EnvironmentError

Like print(), but:

- Supports printing filenames under Unix + Python 3 and Windows + Python 2
- Emulates ANSI escape sequence support under Windows
- Never fails due to encoding/decoding errors. Tries hard to get everything on screen as is, but will fall back to "?" if all fails.

This does not conflict with colorama, but will not use it on Windows.

senf.**input_**(*prompt=None*)

Parameters **prompt** (*object*) – Prints the passed object to stdout without adding a trailing newline

Returns *fsnative*

Raises EnvironmentError

Like input() but returns a *fsnative* and allows printing filenames as prompt to stdout.

Use *fsn2text()* on the result if you just want to deal with text.

senf.**expanduser**(*path*)

Parameters **path** (pathlike) – A path to expand

Returns *fsnative*

Like os.path.expanduser() but supports unicode home directories under Windows + Python 2 and always returns a *fsnative*.

senf.**expandvars**(*path*)

Parameters **path** (pathlike) – A path to expand

Returns *fsnative*

Like os.path.expandvars() but supports unicode under Windows + Python 2 and always returns a *fsnative*.

senf.**gettempdir**()

Returns *fsnative*

Like tempfile.gettempdir(), but always returns a *fsnative* path

senf.**gettempprefix**()

Returns *fsnative*

Like tempfile.gettempprefix(), but always returns a *fsnative* path

senf.**mkstemp**(*suffix=None*, *prefix=None*, *dir=None*, *text=False*)

Parameters

- **suffix** (*pathlike* or None) – suffix or None to use the default
- **prefix** (*pathlike* or None) – prefix or None to use the default
- **dir** (*pathlike* or None) – temp dir or None to use the default
- **text** (*bool*) – if the file should be opened in text mode

Returns A tuple containing the file descriptor and the file path

Return type Tuple[int, *fsnative*]

Raises EnvironmentError

Like tempfile.mkstemp() but always returns a *fsnative* path.

senf.**mkdtemp**(*suffix=None*, *prefix=None*, *dir=None*)

Parameters

- **suffix** (*pathlike* or None) – suffix or None to use the default
- **prefix** (*pathlike* or None) – prefix or None to use the default
- **dir** (*pathlike* or None) – temp dir or None to use the default

Returns A path to a directory

Return type *fsnative*

Raises EnvironmentError

Like `tempfile.mkstemp()` but always returns a *`fsnative`* path.

`senf.`**`supports_ansi_escape_codes`**(*fd*)
Returns whether the output device is capable of interpreting ANSI escape codes when *`print_()`* is used.

Parameters **`fd`** (*`int`*) – file descriptor (e.g. `sys.stdout.fileno()`)

Returns `bool`

17.5 Documentation Types

These types only exist for documentation purposes and represent different types depending on the Python version and platform used.

`class` `senf.`**`text`**
Represents `unicode` under Python 2 and `str` under Python 3. Does not include surrogates.

`class` `senf.`**`bytes`**
Represents `str` under Python 2 and `bytes` under Python 3.

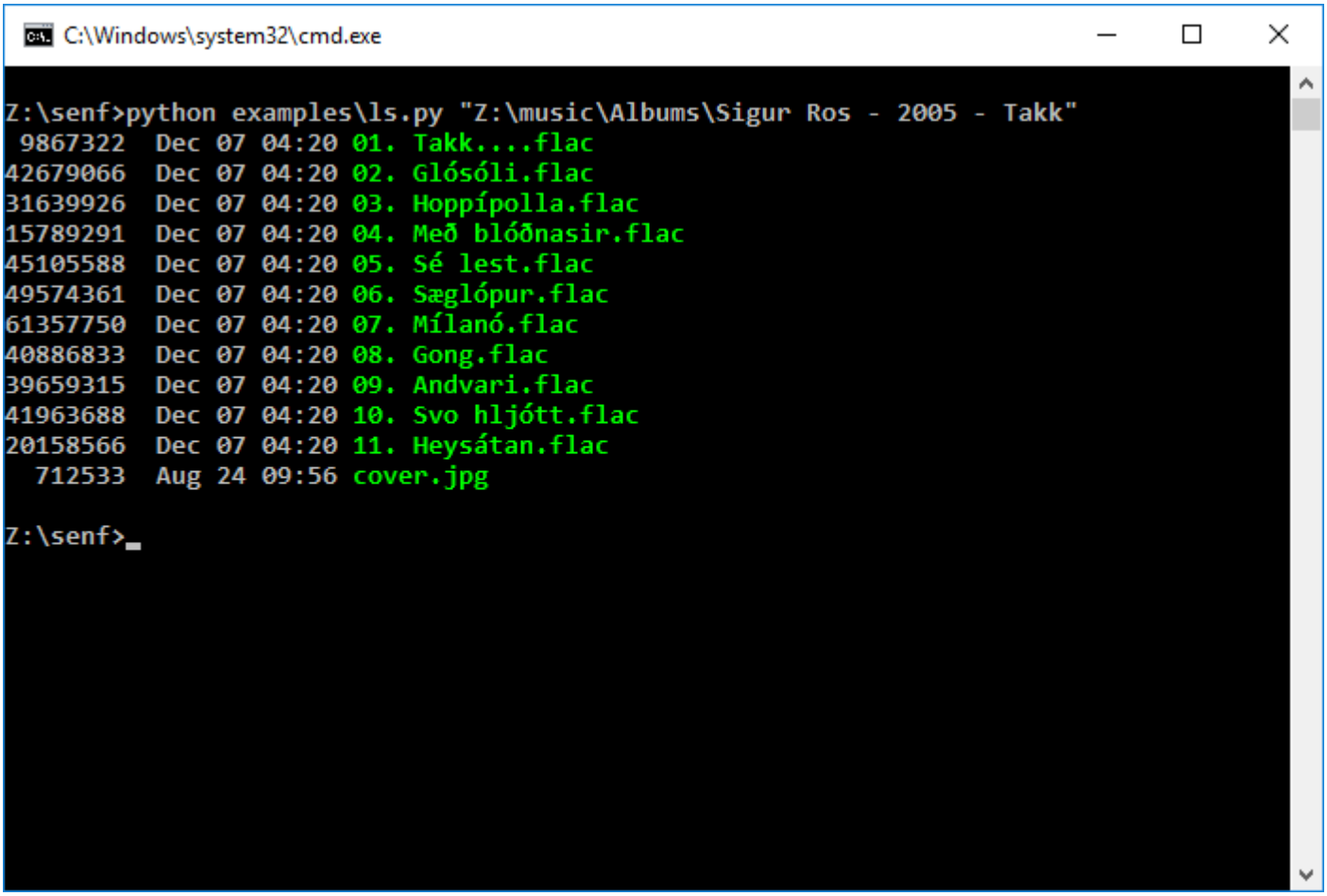
`class` `senf.`**`pathlike`**
Anything the Python stdlib allows as a path. In addition to *`fsnative`* this allows

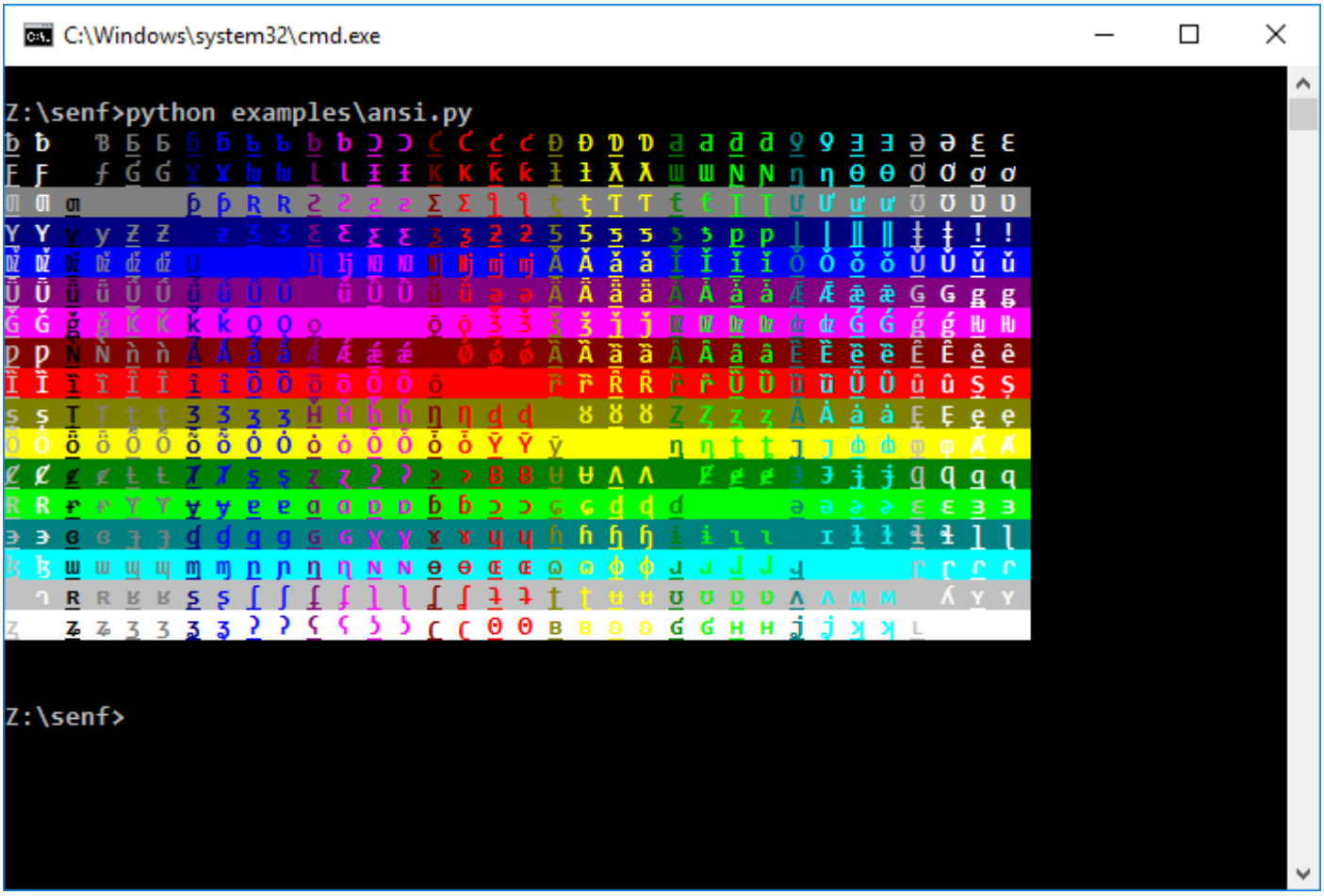
- *`bytes`* encoded with the default file system encoding (usually `mbcs`) on Windows.
- *`bytes`* under Python 3 + Unix.
- `unicode` under Python 2 + Unix if it can be encoded with the default file system encoding.
- (Python 3.6+) Instances where its type implements the `__fspath__` protocol. See PEP 519 for details.

CHAPTER 18

Examples

See https://github.com/quodlibet/senf/tree/master/examples for a few example programs.




C:\Windows\system32\cmd.exe
Z:\senf>python examples\ansi.py
Z:\senf>

CHAPTER 19

Frequently Asked Questions

Are there any existing users of Senf? It is currently used in Quod Libet and mutagen.

Why not use bytes for paths on Python 3 + Unix? Downsides of using str: str can not be pickled as it depends on the locale encoding. You have to use something like *fsn2bytes* first, or you have to make sure that the encoding doesn't change across program invocations.

Upsides of using str: str has more support in the stdlib (pathlib for example) and it can be used in combination with the string literal `"foo"`. The later makes `some_fsnative + "foo"` work for all Python versions and platforms as long as it contains ASCII only.

Why the weird "foo2bar" function naming? As the real types depend on the platform anything like "decode"/"encode" is confusing. So you end up with "a_to_b" or "a_from_b". And imo having things always go one direction, being fast to parse visually and not being too long makes this a good choice. But ymmv.

How can it be that *fsnative()* can't fail, even with an ASCII encoding? It falls back to utf-8 if encoding fails. Raising there would make everything complicated and there is no good way to handle that error case anyway.

Why not replace `sys.stdout` instead of providing a new `print()`? No monkey patching. Allows us to do our own error handling so print will never fail. Printing some question marks is better than a stack trace if the target is a user.

Senf introduces a new platform native string type called *fsnative*. It adds functions to convert text, bytes and paths to and from that new type and helper functions to integrate it nicely with the Python stdlib.

Senf supports Python 2.7, 3.3+, works with PyPy, works on Linux, Windows, macOS, is MIT licensed, and only depends on the stdlib. It does not monkey patch anything in the stdlib.

```
pip install senf
```

https://github.com/quodlibet/senf

CHAPTER 20

Why?

OS strings are used in many different places across the Python stdlib. They are used for filesystem paths, for environemnt variables (`os.environ`), for program arguments (`sys.argv` and `subprocess`), for printing to the console (`sys.stdout`, `sys.stderr`) and more.

The problem with them is that they come in many shapes and forms and handling them has changed significantly between Python 2 and Python 3.

A valid platform native string is either *`bytes`*, `unicode`, `str` + surrogates (either through the `surrogatepass` or the `surrogateescape` error handler) or anything implementing the `__fspath__` protocol. The values of those types depend on the Python version, the platform and the enviroment the program was started in. Ideally we don't want to care about any of those details.

For example, assume you want to check the extension of a file name:

```
import os
from senf import path2fsn

def has_extension(filename, ext):
    root, filename_ext = os.path.splitext(path2fsn(filemame))
    return filename_ext == path2fsn(ext)
```

This will just work everywhere. *`path2fsn()`* will convert anything which is considered a valid path by Python to a *`fsnative`* and then we can just compare by value. Note that Python stdlib functions will always returns the same type which was passed in, so `os.path.splitext()` will return two *`fsnative`* values.

Or you want to send a filename over some binary interface:

```
from senf import fsnative, fsn2bytes, bytes2fsn

def send(filename):
    assert isinstance(filename, fsnative)
```

(continues on next page)

(continued from previous page)

```
    data = fsn2bytes(filename, "utf-8")
    return data

def receive(data):
    filename = bytes2fsn(data, "utf-8")
    return filename
```

fsn2bytes() converts the path to binary ("utf-8" is used on Windows, or "wtf-8" to be exact) and the receiving end re-creates the filename with *bytes2fsn()*.

Another example is printing filenames and text to a console:

```
import os
from senf import print_, argv

for filename in os.listdir(argv[1]):
    print_(u"File: ", filename)
```

Senf provids its own print function which can output platform strings as is and mix them with text. No more encoding/decoding errors.

In addition, **Senf** emulates ANSI escape sequence handling when using the Windows console and extends Python 2 under Windows with Unicode support for `sys.argv` and `os.environ`.

CHAPTER 21

Who?

Senf is used by the following software:

- Quod Libet - A multi platform music player
- mutagen - A Python multimedia tagging library

Index