Pathlab

Release 0.1

Feb 16, 2020

Contents

Pathlab provides an object-oriented path interface to archives, images, remote filesystems, etc. It is built upon pathlib and includes built-in support for:

- `tar` archives
- `zip` archives
- `iso` disc images (inc Rock Ridge; exc Joliet and UDF)
- JFrog Artifactory (via `requests`)

You can also define your own `Path` subclass with its own accessor.

CHAPTER 1

Installation

Requires Python 3.6+. Use pip:

```
pip install --user pathlab
```

CHAPTER 2

Usage

These usage examples are adapted from the pathlib documentation.

Getting a path type:

```
>>> from pathlab import TarAccessor
>>> TarPath = TarAccessor('project.tgz').TarPath
```

Listing subdirectories:

```
>>> root = TarPath('/')
>>> [x for x in root.iterdir() if x.is_dir()]
[TarAccessor('project.tgz').TarPath('/docs')
 TarAccessor('project.tgz').TarPath('/etc'),
 TarAccessor('project.tgz').TarPath('/project')]
```

Listing Python source files in this directory tree:

```
>>> list(root.glob('**/*.py'))
[TarAccessor('project.tgz').TarPath('/setup.py'),
 TarAccessor('project.tgz').TarPath('/docs/conf.py'),
 TarAccessor('project.tgz').TarPath('/project/__init__.py')]
```

Navigating inside a directory tree:

```
>>> p = TarPath('/etc')
>>> q = p / 'init.d' / 'reboot'
>>> q
TarAccessor('project.tgz').TarPath('/etc/init.d/reboot')
>>> q.resolve()
TarAccessor('project.tgz').TarPath('/etc/rc.d/init.d/halt')
```

Querying path properties:

```
>>> q.exists()
True
>>> q.is_dir()
False
```

Opening a file:

```
>>> with q.open() as f: f.readline()
...
'#!/bin/bash\n'
```

CHAPTER 3

Documentation

3.1 Using Pathlab

Use the Python standard library's `pathlib` module to interact with the local filesystem:

```
>>> import pathlib
>>> etc = pathlib.Path('/etc')
>>> etc.exists()
True
```

3.1.1 Tar Archives

Use a *pathlab.TarAccessor* object to interact with a `tar` file:

```
>>> import pathlab
>>> archive = pathlab.TarAccessor('myproject.tar.gz')
>>> root = archive.TarPath('/')
>>> readme = root / 'readme.txt'
>>> readme.exists()
True
```

3.1.2 Zip Archives

Use a *pathlab.ZipAccessor* object to interact with a `zip` file:

```
>>> import pathlab
>>> archive = pathlab.ZipAccessor('myproject.zip')
>>> root = archive.ZipPath('/')
>>> readme = root / 'readme.txt'
>>> readme.exists()
True
```

3.1.3 Iso Images

Use an *pathlab.IsoAccessor* object to interact with an `iso` file:

```
>>> import pathlab
>>> disk = pathlab.IsoAccessor('myproject.iso')
>>> root = disc.IsoPath('/')
>>> readme = root / 'readme.txt'
>>> readme.exists()
True
```

3.1.4 Artifactory Instances

Use an *pathlab.RtAccessor* object to interact with a JFrog Artifactory instance:

```
>>> import pathlab
>>> rt = pathlab.RtAccessor('http://artifactory/')
>>> repo = rt.RtPath('/myproject/latest')
>>> readme = repo / 'readme.txt'
>>> readme.exists()
True
```

See also:

See the *API Reference* and `pathlib` documentation for more detail!

3.2 Extending Pathlab

Here's how you can create your own path-like class by subclassing *Path* and *Accessor*:

```
import pathlab

class MyPath(pathlab.Path):
    __slots__ = ()

class MyAccessor(pathlab.Accessor):
    factory = MyPath

    def __repr__(self):
        return "MyAccessor()"
```

At this point we can instantiate our accessor, which acts like a module with a `MyPath` class:

```
>>> accessor = MyAccessor()
>>> accessor
MyAccessor()
>>> root = accessor.MyPath("/")
>>> root
MyAccessor().MyPath('/')
```

Pure methods work as we'd expect, whereas impure methods raise `NotImplementedError`:

```
>>> docs = root / 'docs'
>>> docs
```

(continues on next page)

(continued from previous page)

```
MyAccessor().MyPath('/docs')
>>> docs.exists()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File ".../pathlib.py", line 1339, in exists
    self.stat()
  File ".../pathlib.py", line 1161, in stat
    return self._accessor.stat(self)
  File ".../pathlab/core/accessor.py", line 76, in stat
    raise NotImplementedError
NotImplementedError
```

Now we can begin adding methods to our accessor:

```
class MyAccessor(pathlab.Accessor):
    factory = MyPath

    def __init__(self, children):
        self.children = children

    def __repr__(self):
        return "MyAccessor(%r)" % self.children

    def stat(self, path):
        return pathlab.Stat(type='dir')

    def listdir(self, path):
        return self.children
```

Refer to the *Accessor* API documentation for a full list of abstract methods you may wish to implement. Refer to the pathlab source code for example accessor implementations.

3.3 Implementation Notes

3.3.1 Using the Accessor

The standard library's `pathlib` module already includes the notion of an *accessor*, but it is bypassed in certain cases, such as:

- `str(self)` called to get a local filesystem path
- `os.close()` called to close file descriptors
- `os.getcwd()` called to get the working directory
- `os.fsencode()` called to get a `bytes` representation of a path
- `os.environ` accessed to expand ~
- `pwd` and `grp` used to retrieve user and group names

Pathlab fixes these instances by subclassing `pathlib.Path` as *pathlab.Path* and re-implementing the problematic methods. This includes a few additions and changes to the accessor interface.

3.3.2 Flavouring the Accessor

The standard library's `pathlib` module uses a *flavour* object to handle pure path semantics. As before, this abstraction is leaky. Pathlab makes no distinction between the path accessor and flavour, and so allows methods like `casefold()` to be re-implemented.

3.3.3 Binding the Accessor

The standard library's `pathlib` module provides limited means of storing state. A path instance may have its `_accessor` attribute customized, and in *some* cases derived path instances are initialized with `path._init(template=self)` to make the new path use the same accessor. However, this mechanism is used inconsistently.

Pathlab solves this by creating a new path *type* per accessor *instance*. The accessor instance is bound to the new type as a class attribute. Therefore the inheritance chain of an accessor's path type is as follows:

Table 1: Path inheritance chain

Abstract	Pure	Class
	✓	`pathlib.PurePath`
		`pathlib.Path`
✓	✓	*`pathlab.Path`*
✓	✓	`mylib.MyPath`
		`mylib.MyAccessor(...).MyPath`

3.4 API Reference

3.4.1 Concrete Accessors

class pathlab.**TarAccessor**(*file*)
Accessor for `.tar` archives. Supports writing of files, but not other forms of modification.

Parameters **file** – Path to `.tar` file, or file object.

class pathlab.**ZipAccessor**(*file*)
Accessor for `.zip` archives. Supports writing of files, but not other forms of modification.

Parameters **file** – Path to `.zip` file, or file object.

class pathlab.**IsoAccessor**(*file*, *ignore_susp=False*, *cache_size=1024*)
Accessor for `.iso` image. Supports plain ISOs and those with SUSP/Rock Ridge data. Does not support Joliet or UDF (yet). Does not support modification.

Parameters

- **file** – Path to `.iso` file, or file object.
- **ignore_susp** – Whether to ignore SUSP/Rock Ridge data
- **cache_size** – specifies the size of the LRU cache to use for `_load_record()` results. Set to `0` to disable caching.

class pathlab.**RtAccessor**(*url*, *cache_size=1024*)
Accessor for JFrog Artifactory.

Parameters

- **url** – specifies the Artifactory base URL (excluding the repo name)
- **cache_size** – specifies the size of the LRU cache to use for `stat()` results. Set to `0` to disable caching.

3.4.2 Abstract Accessor

class pathlab.**Accessor**

An accessor object allows instances of an associated *Path* type to access some kind of filesystem.

Subclasses are free to define an initializer and store state, such as a socket object or file descriptor. Methods such as *listdir()* may then reference that state.

To create a path object, access its type as an attribute of an accessor object.

This table shows all abstract methods that should be implemented in your own *Accessor* subclass.

Abstract Method	Description
open()	Open path and return a file object
stat()	Return the a *Stat* object for the path
listdir()	Yield names of directory children
readlink()	Return the target of the this symbolic link
create()	Create the file, if it doesn't exist
chmod()	Change file permissions
move()	Move/rename the file
delete()	Delete the file
download()	Download to the *local* filesystem
upload()	Upload from the *local* filesystem
fspath()	Return a *local* path for this path
getcwd()	Return the working directory
gethomedir()	Return the user's home directory
close()	Close this accessor object

This table shows utility methods you may call from your methods to raise an exception:

Method	Raises	Errno
not_found()	FileNotFoundError	ENOENT
already_exists()	FileExistsError	EEXIST
not_a_directory()	NotADirectoryError	ENOTDIR
is_a_directory()	IsADirectoryError	EISDIR
not_a_symlink()	OSError	EINVAL
permission_denied()	PermissionError	EACCES

This table shows all methods with default implementations that you may wish to re-implement:

Method	Uses	Like
splitroot()		
casefold()		
casefold_parts()		
is_reserved()		
make_uri()		
resolve()	*readlink()*	os.path.abspath()
scandir()	*listdir()*	os.scandir()
touch()	*create()*	
mkdir()	*create()*	os.mkdir()
symlink()	*create()*	os.symlink()
unlink()	*delete()*	os.unlink()
rmdir()	*delete()*	os.rmdir()
rename()	*move()*	os.rename()
replace()	*move()*	os.replace()
lstat()	*stat()*	os.lstat()
lchmod()	*chmod()*	os.lchmod()
__enter__()		
__exit__()	*close()*	

factory = None
Must be set to a subclass of *pathlab.Path*

open(*path*, *mode='r'*, *buffering=-1*)
Open the path and return a file object, like io.open().

The underlying stream *must* be opened in binary mode (not text mode). The file mode is as in io.open(), except that it will not contain any of b, t or U.

In w mode, you may wish to return a *Creator* object.

stat(*path*, *, *follow_symlinks=True*)
Return a *Stat* object for the path, like os.stat().

listdir(*path*)
Yield names of the files in the directory, a bit like os.listdir().

readlink(*path*)
Return a string representing the path to which the symbolic link points, like os.readlink()

create(*path*, *stat*, *fileobj=None*)
Create the file. The given *Stat* object provides file metadata, and the *fileobj*, where given, provides a readable stream of the new file's content.

chmod(*path*, *mode*, *, *follow_symlinks=True*)
Change the permissions of the path.

move(*path*, *dest*)
Move/rename the file.

delete(*path*)
Remove the file.

fspath(*path*)
Return an string representing the given path as a *local* filesystem path. The path need not exist.

download(*src*, *dst*)
Download from *src* to *dst*. *src* is an instance of your path class.

upload(*src*, *dst*)
Upload from *src* to *dst*. *dst* is an instance of your path class.

getcwd()
Return the current working directory, like `os.getcwd()`.

gethomedir(*username=None*)
Return the user's home directory.

close()
Close this accessor object.

static not_found(*path*)
Raise a `FileNotFoundError`

static already_exists(*path*)
Raise a `FileExistsError` with `EEXIST`

static not_a_directory(*path*)
Raise a `NotADirectoryError` with `ENOTDIR`

static is_a_directory(*path*)
Raise an `IsADirectoryError` with `EISDIR`

static not_a_symlink(*path*)
Raise an `OSError` with `EINVAL`

static permission_denied(*path*)
Raise a `PermissionError` with `EACCES`

3.4.3 Path

class pathlab.**Path**
Bases: `pathlib.Path`

Path-like object.

This table shows all methods and attributes available from *`Path`* instances. You should not need to re-implement any of these methods. Methods marked as pure will work even without an accessor; other methods will call at least one method of the accessor.

Pure	Method	Description	Returns
✓	`parts`	Path components	`str` sequence
✓	`drive`	Drive letter, if any	`str`
✓	`root`	Root, e.g. `/`	`str`
✓	`anchor`	Drive letter and root	`str`
✓	`name`	Final path component	`str`
✓	`stem`	Final path component without its suffix	`str`
✓	`suffix`	File extension of final path component	`str`
✓	`suffixes`	File extensions of final path component	`str` sequence
✓	`as_posix()`	With forward slashes	`str`

Continued on next page

Table 2 – continued from previous page

Pure	Method	Description	Returns
✓	as_uri()	With file:// prefix	str
	owner()	Name of file owner	str
	group()	Name of file group	str
	stat()	Status of file (follow symlinks)	*Stat*
	lstat()	Status of symlink	*Stat*
	samefile()	Paths are equivalent	bool
	sameaccessor()	Paths use the same accessor	bool
	exists()	Path exists	bool
	is_dir()	Path is a directory	bool
	is_file()	Path is a regular file	bool
	is_mount()	Path is a mount point	bool
	is_symlink()	Path is a symlink	bool
	is_block_device()	Path is a block device	bool
	is_char_device()	Path is a character device	bool
	is_fifo()	Path is a FIFO	bool
	is_socket()	Path is a socket	bool
✓	is_absolute()	Path is absolute	bool
✓	is_reserved()	Path is reserved	bool
✓	match()	Path matches a glob pattern	bool
✓	joinpath()	Append path components	*Path*
✓	parent	Immediate ancestor	*Path*
✓	parents	Ancestors	*Path* sequence
	iterdir()	Files in directory	*Path* sequence
	glob()	Files in subtree matching pattern	*Path* sequence
	rglob()	As above, but includes directories	*Path* sequence
✓	relative_to()	Make relative	*Path*
✓	with_name()	Change name	*Path*
✓	with_suffix()	Change suffix	*Path*
	resolve()	Resolve symlinks and make absolute	*Path*
	expanduser()	Expand ~ and ~user	*Path*
	touch()	Create file	
	mkdir()	Create directory	
	symlink_to()	Create symlink	
	unlink()	Delete file or link	
	rmdir()	Delete directory	
	rename()	Move without clobbering	
	replace()	Move	
	chmod()	Change perms of file (follow symlinks)	
	lchmod()	Change perms of symlink0	
	open()	Open file and return file object	fileobj
	read_bytes()	Read file content as bytes	bytes

Continued on next page

Table 2 – continued from previous page

Pure	Method	Description	Returns
	`read_text()`	Read file content as text	`str`
	`write_bytes()`	Write file content as bytes	
	`write_text()`	Write file content as text	
	`upload_from()`	Upload from *local* filesystem	
	`download_to()`	Download to *local* filesystem	

Only *additional* methods are documented here; see the `pathlib` documentation for other methods.

sameaccessor(*other_path*)
Returns whether this path uses the same accessor as *other_path*.

upload_from(*source*)
Upload/add this path from the given *local* filesystem path.

download_to(*target*)
Download/extract this path to the given *local* filesystem path.

3.4.4 Stat

class pathlab.**Stat**(***params*)
Mutable version of `os.stat_result`. The usual `st_` attributes are available as read-only properties. Other attributes may be passed to the initializer or set directly.

Objects of this type (or a subclass) may be returned from *`Accessor.stat()`*.

type = 'file'
File type, for example `file`, `dir`, `symlink`, `socket`, `fifo`, `char_device` or `block_device`. Other values may be used, but will result in a `stat.st_mode` that indicates a regular file.

size = 0
File size in bytes

permissions = 0
Permission bits

user = None
Name of file owner

group = None
Name of file group

user_id = 0
ID of file owner

group_id = 0
ID of file group

device_id = 0
ID of containing device

file_id = 0
ID that uniquely identifies the file on the device

hard_link_count = 0
Number of hard links to this file

create_time = datetime.datetime(1970, 1, 1, 0, 0)
Time of creation

access_time = datetime.datetime(1970, 1, 1, 0, 0)
Time of last access

modify_time = datetime.datetime(1970, 1, 1, 0, 0)
Time of last modification

status_time = datetime.datetime(1970, 1, 1, 0, 0)
Time of last status modification

target = None
Link target

3.4.5 Creator

class pathlab.**Creator**(*target*, *target_mode='delete'*, *parent_mode='raise'*, *stat=None*)
A creator object is a `BytesIO` object that writes its contents to a path when closed. It calls *`Accessor.create()`* to achieve this.

A creator object may be returned from *`Accessor.open()`* in `w` mode.

Parameters

- **target** – The path to be created
- **target_mode** – Action to take when path exists. One of `ignore`, `raise`, or `delete` (the default).
- **parent_mode** – Action to take when parent doesn't exist. One of `ignore`, `raise` (the default), or `create`.
- **stat** – The initial *`Stat`* object, which will have its `size` attribute changed.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

U

Z