pyfilesystem

...or how can I access various filesystems with a common API?

Pyfilesystem – What is it?

- It allows you to access local files (EG via the POSIX file API), remote filesystems (EG SMB, S3), archive files (EG tar, zip) – all with a common Python API
- It works on Python 2.7 and 3.3+
- If you want to access such a pyfilesystem from C or Java, you're out of luck.

Compared to Fuse and Dokan

- Fuse exposes various filesystem types as “local files and directories”, though Fuse is Linux (including Android) and *BSD (including macOS)
- So does Dokan, though Dokan is Windows only.
- You can access these filesystems in any language; they look local.
- Pyfilesystem is an API, not a local filesystem.

Version as of 2018-04-30

- Pyfilesystem’s current version, at the time of this writing, is 2.0.20.

Red Herring

- BTW, pyfs sounds like it would be the Pypi name for pyfilesystem, but it’s not. Pyfs is an unrelated package.
- Pyfilesystem is simply “fs”.

Example filesystem-independent code

- Count nonblank lines of .py files:

```
def count_python_loc(fs):
    """Count non-blank lines of Python code."""
    count = 0
    for path in fs.walk.files(filter=['*.py']):
        with fs.open(path) as python_file:
            count += sum(1 for line in python_file if line.strip())
    return count
```

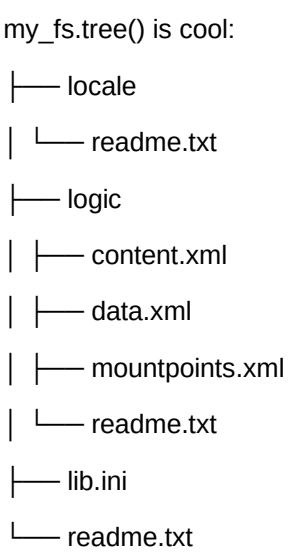
Calling count_python_loc

- We can call count_python_loc as follows:

- from fs import open_fs
- projects_fs = open_fs('~/projects')
- # project_fs = open_fs(‘ftp://ftp.eg.org/pub’)
- print(count_python_loc(projects_fs))
- # Or sshfs or s3 or smb/cifs

Tree example

- my_fs.tree() is cool, and can be useful in debugging:
- ├── locale
- │   └── readme.txt
- ├── logic
- │   ├── content.xml
- │   ├── data.xml
- │   ├── mountpoints.xml
- │   └── readme.txt
- ├── lib.ini
- └── readme.txt



Use as a context manager

- You can fs.close(), or you can use it as a context manager:
- >>> with open_fs('osfs://~/') as home_fs:
- ... home_fs.settext('reminder.txt', 'buy coffee')

- osfs is a local file. It is also the default.
- We’re writing to it this time.

Listing a directory

- Similar to os.listdir('~/projects'):

- >>> home_fs.listdir('/projects')
- ['fs', 'moya', 'README.md']

- You can also get back mtime, size, etcetera:
- >>> directory = list(home_fs.scandir('/projects'))
- >>> directory
- [<dir 'fs'>, <dir 'moya'>, <file 'README.md'>]

- A little unfortunately, it appears to be eager, not lazy.

Scandir returns info objects

- Info objects have a number of advantages over just a filename:
  - You can tell if an info object references a file or a directory with the is_dir attribute.
  - Info objects may also contain information such as size, modified time, etc. if you request it in the namespaces parameter.

Reading and writing text and bytes

- home_fs.gettext(‘filename’)
- home_fs.getbytes(‘filename’)
- home_fs.settext(‘filename’, u’abc’)
- home_fs.setbytes(‘filename’, b’def’)

- The author of the package appears to prefer these methods over opening file and iterating, although these are probably limited to available virtual memory.

Copying or moving a file: same filesystem

- Copy a file: home_fs.copy(‘from’, ‘to’)
- Move a file: home_fs.move(‘from’, ‘to’)
- Copy a directory: home_fs.copydir(‘from’, ‘to’)
- Move a directory: home_fs.movedir(‘from’, ‘to’)

Copying or moving a file: different filesystem

- >>> from fs.copy import copy_fs
- >>> copy_fs('~/projects', 'zip://projects.zip')

- You can use a Walker instance to restrict what gets copied (more later)

Paths

- Paths are unix-style:
  - / is the directory separator
  - .. is one level up
- However, paths are treated as unicode
  - Nice for many applications
  - Not good for *ix programs that need to operate on arbitrary files: it can raise UnicodeDecodeError for some filenames and terminate traversal
  - Apparently this is a known issue and is being worked on. However, it will likely require some sort of API change.

getcwd(), chdir()

- There is no concept of a “current working directory” in pyfilesystem
- Instead, you specify entire paths, relative to the root of the filesystem (directory hierarchy) you specified when open_fs()’ing.
- The closest thing to chdir() is fs.opendir()

Getting metadata

- resource_info = fs.getinfo('myfile.txt', namespaces=['details', 'access'])
- resource_info = fs.getinfo('myfile.txt', namespaces=['link'])

- Note that unknown namespaces (for the filesystem type in question) are ignored. No error is returned/raised.
- However, you can:
- if info.has_namespace('access'):
- print('user is {}'.format(info.user))
- Namespaces are filesystem-specific, but include things like mtime or size

Example URL's

- osfs://~/projects
- osfs://c://system32
- ftp://ftp.example.org/pub
- mem://
- ftp://will:daffodil@ftp.example.org/private
- sshfs://hostname/dir/ect/ory

- from fs import open_fs
- projects_fs = open_fs('osfs://~/projects')

- BTW, for passwordless access, it's OK to leave the username and password out of the URL.
- Also, osfs is the default URL type

Filesystem types

- Builtin
- Official but not builtin
- Third-party

Builtin filesystem types

- APP systems: Windows profile directories (?)
- FTP Filesystem
- Memory Filesystem: Used for caches, temporary data stores, unit testing…
- Mount Filesystem: Can put two filesystems under a common /
- Multi Filesystem: Can overlay 2 (or more?) filesystems
- OS Filesystem
- Sub Filesystem: Used by opendir(), the chdir()-like method
- Tar Filesystem
- Temporary Filesystem: manage filesystems in /tmp or similar
- Zip Filesystem

Official but not builtin filesystem types

- S3FS: Amazon AWS S3 Filesystem.
- WebDavFS: WebDav Filesystem.

Third party filesystems

- fs.archive Enhanced archive filesystems. Appears to autodetect archive type.
- fs.dropboxfs Dropbox Filesystem.
- fs.onedrivefs Microsoft OneDrive Filesystem.
- fs.smbfs A filesystem running over the SMB protocol. AKA CIFS.
- fs.sshfs A filesystem running over the SSH protocol using paramiko. Some old doc says this only works on Python 2.x, but I tried it on 3.6 and it appeared to work.
- fs.youtube A filesystem for accessing YouTube Videos and Playlists.
- fs.dnla A filesystem for accessing accessing DLNA Servers. It's an old digital media streaming protocol that still appears to be going strong. It appears to be misdocumented on the pyfilesystem website as being a youtube thing.

Restricting filesystem traversal

- >>> from fs import open_fs
- >>> from fs.copy import copy_fs
- >>> from fs.walk import Walker
- >>> py_walker = Walker(filter=['*.py'], exclude_dirs=['*.git'])
- >>> copy_fs("~/projects", "zip://~/projects.zip", py_walker)

That’s it.

- Questions?