libgfapi-python Documentation

Release 0

Red Hat Inc.

Sep 25, 2018

Contents

This is the official python bindings for GlusterFS libgfapi C library interface.

CHAPTER 1

Example Usage

```
from gluster import gfapi

# Create virtual mount
volume = gfapi.Volume('10.7.1.99', 'datavolume')
volume.mount()

# Create directory
volume.mkdir('dir1', 0755)

# List directories
volume.listdir('/')

# Create new file and write to it
with volume.fopen('somefile.txt', 'w+') as f:
    f.write("Winter is coming.")

# Open and read file
with volume.fopen('somefile.txt', 'r') as f:
  print f.read()

# Delete file
volume.unlink('somefile.txt')

# Unmount the volume
volume.umount()
```

1.1 API Reference

1.1.1 Volume Class

class gluster.gfapi.**Volume**(*hosts*, *volname*, *proto=u'tcp'*, *port=24007*, *log_file=u'/dev/null'*, *log_level=7*)

__init__(*hosts*, *volname*, *proto=u'tcp'*, *port=24007*, *log_file=u'/dev/null'*, *log_level=7*)
Create a Volume object instance.

Parameters

- **hosts** – Host (string) or hosts (list of strings) with glusterd daemon running OR path to socket file which glusterd is listening on.
- **volname** – Name of GlusterFS volume to be mounted and used.
- **proto** – Transport protocol to be used to connect to management daemon. Permitted values are "tcp" and "rdma".
- **port** – Port number where gluster management daemon is listening.
- **log_file** – Path to log file. When this is set to None, a new logfile will be created in default log directory i.e /var/log/glusterfs. The default is "/dev/null"
- **log_level** – Integer specifying the degree of verbosity. Higher the value, more verbose the logging.

access(***kwargs*)
Use the real uid/gid to test for access to path.

Parameters

- **path** – Path to be checked.
- **mode** – mode should be F_OK to test the existence of path, or it can be the inclusive OR of one or more of R_OK, W_OK, and X_OK to test permissions

Returns True if access is allowed, False if not

chdir(***kwargs*)
Change the current working directory to the given path.

Parameters **path** – Path to change current working directory to

Raises OSError on failure

chmod(***kwargs*)
Change mode of path

Parameters

- **path** – the item to be modified
- **mode** – new mode

Raises OSError on failure

chown(***kwargs*)
Change owner and group id of path

Parameters

- **path** – the path to be modified

- **uid** – new user id for path
- **gid** – new group id for path

Raises OSError on failure

copy (*src*, *dst*)
Copy data and mode bits ("cp src dst")

Copy the file src to the file or directory dst. If dst is a directory, a file with the same basename as src is created (or overwritten) in the directory specified. Permission bits are copied. src and dst are path names given as strings.

Parameters

- **src** – Path of source file
- **dst** – Path of destination file or directory

Raises OSError on failure

copy2 (*src*, *dst*)
Similar to copy(), but metadata is copied as well - in fact, this is just copy() followed by copystat(). This is similar to the Unix command cp -p.

The destination may be a directory.

Parameters

- **src** – Path of source file
- **dst** – Path of destination file or directory

Raises OSError on failure

copyfile (*src*, *dst*)
Copy the contents (no metadata) of the file named src to a file named dst. dst must be the complete target file name. If src and dst are the same, Error is raised. The destination location must be writable. If dst already exists, it will be replaced. Special files such as character or block devices and pipes cannot be copied with this function. src and dst are path names given as strings.

Parameters

- **src** – Path of source file
- **dst** – Path of destination file

Raises Error if src and dst file are same file. OSError on failure to read/write.

classmethod copyfileobj (*fsrc*, *fdst*, *length=131072*)
Copy the contents of the file-like object fsrc to the file-like object fdst. The integer length, if given, is the buffer size. Note that if the current file position of the fsrc object is not 0, only the contents from the current file position to the end of the file will be copied.

Parameters

- **fsrc** – Source file object
- **fdst** – Destination file object
- **length** – Size of buffer in bytes to be used in copying

Raises OSError on failure

copymode (*src*, *dst*)
Copy the permission bits from src to dst. The file contents, owner, and group are unaffected. src and dst are path names given as strings.

Parameters

- **src** – Path of source file
- **dst** – Path of destination file

Raises OSError on failure.

copystat(*src*, *dst*)
Copy the permission bits, last access time, last modification time, and flags from src to dst. The file contents, owner, and group are unaffected. src and dst are path names given as strings.

Parameters

- **src** – Path of source file
- **dst** – Path of destination file

Raises OSError on failure.

copytree(*src*, *dst*, *symlinks=False*, *ignore=None*)
Recursively copy a directory tree using copy2().

The destination directory must not already exist. If exception(s) occur, an Error is raised with a list of reasons.

If the optional symlinks flag is true, symbolic links in the source tree result in symbolic links in the destination tree; if it is false, the contents of the files pointed to by symbolic links are copied.

The optional ignore argument is a callable. If given, it is called with the 'src' parameter, which is the directory being visited by copytree(), and 'names' which is the list of 'src' contents, as returned by os.listdir():

callable(src, names) -> ignored_names

Since copytree() is called recursively, the callable will be called once for each directory that is copied. It returns a list of names relative to the 'src' directory that should not be copied.

disable_logging()
Sends logs to /dev/null effectively disabling them

exists(*path*)
Returns True if path refers to an existing path. Returns False for broken symbolic links. This function may return False if permission is not granted to execute stat() on the requested file, even if the path physically exists.

fopen(***kwargs*)
Similar to Python's built-in File object returned by Python's open()

Unlike Python's open(), fopen() provided here is only for convenience and it does NOT invoke glibc's fopen and does NOT do any kind of I/O bufferring as of today.

The most commonly-used values of mode are 'r' for reading, 'w' for writing (truncating the file if it already exists), and 'a' for appending. If mode is omitted, it defaults to 'r'.

Modes 'r+', 'w+' and 'a+' open the file for updating (reading and writing); note that 'w+' truncates the file.

Append 'b' to the mode to open the file in binary mode but this has no effect as of today.

Parameters

- **path** – Path of file to be opened
- **mode** – Mode to open the file with. This is a string.

Returns an instance of File class

Raises OSError on failure to create/open file. TypeError and ValueError if mode is invalid.

get_volume_id(***kwargs*)
Returns the volume ID (of type uuid.UUID) for the currently mounted volume.

getatime(*path*)
Returns the last access time as reported by stat()

getctime(*path*)
Returns the time when changes were made to the path as reported by stat(). This time is updated when changes are made to the file or dir's inode or the contents of the file

getcwd(***kwargs*)
Returns current working directory.

getmtime(*path*)
Returns the time when changes were made to the content of the path as reported by stat()

getsize(*path*)
Return the size of a file in bytes, reported by stat()

getxattr(***kwargs*)
Retrieve the value of the extended attribute identified by key for path specified.

Parameters

- **path** – Path to file or directory
- **key** – Key of extended attribute
- **size** – If size is specified as zero, we first determine the size of xattr and then allocate a buffer accordingly. If size is non-zero, it is assumed the caller knows the size of xattr.

Returns Value of extended attribute corresponding to key specified.

Raises OSError on failure

isdir(*path*)
Returns True if path is an existing directory. Returns False on all failure cases including when path does not exist.

isfile(*path*)
Return True if path is an existing regular file. Returns False on all failure cases including when path does not exist.

islink(*path*)
Return True if path refers to a directory entry that is a symbolic link. Returns False on all failure cases including when path does not exist.

link(***kwargs*)
Create a hard link pointing to source named link_name.

Raises OSError on failure

listdir(*path*)
Return a list containing the names of the entries in the directory given by path. The list is in arbitrary order. It does not include the special entries '.' and '..' even if they are present in the directory.

Parameters **path** – Path to directory

Raises OSError on failure

Returns List of names of directory entries

listdir_with_stat(*path*)
Return a list containing the name and stat of the entries in the directory given by path. The list is in arbitrary order. It does not include the special entries '.' and '..' even if they are present in the directory.

Parameters **path** – Path to directory

Raises OSError on failure

Returns A list of tuple. The tuple is of the form (name, stat) where name is a string indicating name of the directory entry and stat contains stat info of the entry.

listxattr(***kwargs*)
Retrieve list of extended attribute keys for the specified path.

Parameters

- **path** – Path to file or directory.
- **size** – If size is specified as zero, we first determine the size of list and then allocate a buffer accordingly. If size is non-zero, it is assumed the caller knows the size of the list.

Returns List of extended attribute keys.

Raises OSError on failure

lstat(***kwargs*)
Return stat information of path. If path is a symbolic link, then it returns information about the link itself, not the file that it refers to.

Raises OSError on failure

makedirs(*path*, *mode=511*)
Recursive directory creation function. Like mkdir(), but makes all intermediate-level directories needed to contain the leaf directory. The default mode is 0777 (octal).

Raises OSError if the leaf directory already exists or cannot be created. Can also raise OSError if creation of any non-leaf directories fails.

mkdir(***kwargs*)
Create a directory named path with numeric mode mode. The default mode is 0777 (octal).

Raises OSError on failure

mknod(***kwargs*)
Create special or ordinary file; see mknod(2) for more details.

Parameters

- **path** – Path of file to be created.
- **mode** – Operation to be performed on the given range
- **dev** – Major and minor numbers for newly created device special file; use os.makedev to build value. Ignored for other types.

Raises OSError on failure

mount()
Mount a GlusterFS volume for use.

Raises LibgfapiException on failure

mounted
Read-only attribute that returns True if the volume is mounted. The value of the attribute is internally changed on invoking mount() and umount() functions.

open (***kwargs*)
Similar to Python's os.open()

As of today, the only way to consume the raw glfd returned is by passing it to File class.

Parameters

- **path** – Path of file to be opened
- **flags** – Integer which flags must include one of the following access modes: os.O_RDONLY, os.O_WRONLY, or os.O_RDWR.
- **mode** – specifies the permissions to use in case a new file is created. The default mode is 0777 (octal)

Returns the raw glfd (pointer to memory in C, number in python)

Raises OSError on failure to create/open file. TypeError if flags is not an integer.

opendir (***kwargs*)
Open a directory.

Parameters

- **path** – Path to the directory
- **readdirplus** – Enable readdirplus which will also fetch stat information for each entry of directory.

Returns Returns a instance of Dir class

Raises OSError on failure

readlink (***kwargs*)
Return a string representing the path to which the symbolic link points. The result may be either an absolute or relative pathname.

Parameters **path** – Path of symbolic link

Returns Contents of symlink

Raises OSError on failure

remove (*path*)
Remove (delete) the file path. If path is a directory, OSError is raised. This is identical to the unlink() function.

Raises OSError on failure

removexattr (***kwargs*)
Remove a extended attribute of the path.

Parameters

- **path** – Path to the file or directory.
- **key** – The key of extended attribute.

Raises OSError on failure

rename (***kwargs*)
Rename the file or directory from src to dst. If dst is a directory, OSError will be raised. If dst exists and is a file, it will be replaced silently if the user has permission.

Raises OSError on failure

rmdir (***kwargs*)
Remove (delete) the directory path. Only works when the directory is empty, otherwise, OSError is raised. In order to remove whole directory trees, rmtree() can be used.

Raises OSError on failure

rmtree (*path*, *ignore_errors=False*, *onerror=None*)
Delete a whole directory tree structure. Raises OSError if path is a symbolic link.

Parameters

- **path** – Directory tree to remove
- **ignore_errors** – If True, errors are ignored
- **onerror** – If set, it is called to handle the error with arguments (func, path, exc) where func is the function that raised the error, path is the argument that caused it to fail; and exc is the exception that was raised. If ignore_errors is False and onerror is None, an exception is raised

Raises OSError on failure if onerror is None

samefile (*path1*, *path2*)
Return True if both pathname arguments refer to the same file or directory (as indicated by device number and inode number). Raise an exception if a stat() call on either pathname fails.

Parameters

- **path1** – Path to one file
- **path2** – Path to another file

Raises OSError if stat() fails

scandir (*path*)
Return an iterator of `DirEntry` objects corresponding to the entries in the directory given by path. The entries are yielded in arbitrary order, and the special entries '.' and '..' are not included.

Using scandir() instead of listdir() can significantly increase the performance of code that also needs file type or file attribute information, because `DirEntry` objects expose this information.

scandir() provides same functionality as listdir_with_stat() except that scandir() does not return a list and is an iterator. Hence scandir is less memory intensive on large directories.

Parameters **path** – Path to directory

Raises OSError on failure

Yields Instance of `DirEntry` class.

set_logging (*log_file*, *log_level*)
Set logging parameters. Can be invoked either before or after invoking mount().

When invoked before mount(), the preferred log file and log level choices are recorded and then later enforced internally as part of mount()

When invoked at any point after mount(), the change in log file and log level is instantaneous.

Parameters

- **log_file** – Path of log file. If set to "/dev/null", nothing will be logged. If set to None, a new logfile will be created in default log directory (/var/log/glusterfs)
- **log_level** – Integer specifying the degree of verbosity. Higher the value, more verbose the logging.

setfsgid(*gid*)
setfsgid() changes the value of the caller's filesystem group ID-the group ID that the Linux kernel uses to check for all accesses to the filesystem.

Raises OSError on failure

setfsuid(*uid*)
setfsuid() changes the value of the caller's filesystem user ID-the user ID that the Linux kernel uses to check for all accesses to the filesystem.

Raises OSError on failure

setxattr(***kwargs*)
Set extended attribute of the path.

Parameters

- **path** – Path to file or directory.
- **key** – The key of extended attribute.
- **value** – The valiue of extended attribute.
- **flags** – Possible values are 0 (default), 1 and 2. If 0 - xattr will be created if it does not exist, or the value will be replaced if the xattr exists. If 1 - it performs a pure create, which fails if the named attribute already exists. If 2 - it performs a pure replace operation, which fails if the named attribute does not already exist.

Raises OSError on failure

stat(***kwargs*)
Returns stat information of path.

Raises OSError on failure

statvfs(***kwargs*)
Returns information about a mounted glusterfs volume. path is the pathname of any file within the mounted filesystem.

Returns An object whose attributes describe the filesystem on the given path, and correspond to the members of the statvfs structure, namely: f_bsize, f_frsize, f_blocks, f_bfree, f_bavail, f_files, f_ffree, f_favail, f_fsid, f_flag, and f_namemax.

Raises OSError on failure

symlink(***kwargs*)
Create a symbolic link 'link_name' which points to 'source'

Raises OSError on failure

umount()
Unmount a mounted GlusterFS volume.

Provides users a way to free resources instead of just waiting for python garbage collector to call __del__() at some point later.

Raises LibgfapiException on failure

unlink(***kwargs*)
Delete the file 'path'

Raises OSError on failure

utime(***kwargs*)
Set the access and modified times of the file specified by path. If times is None, then the file's access and

modified times are set to the current time. (The effect is similar to running the Unix program touch on the path.) Otherwise, times must be a 2-tuple of numbers, of the form (atime, mtime) which is used to set the access and modified times, respectively.

Raises OSError on failure to change time. TypeError if invalid times is passed.

walk(*top*, *topdown=True*, *onerror=None*, *followlinks=False*)

Generate the file names in a directory tree by walking the tree either top-down or bottom-up.

Slight difference in behaviour in comparison to os.walk(): When os.walk() is called with 'followlinks=False' (default), symlinks to directories are included in the 'dirnames' list. When Volume.walk() is called with 'followlinks=False' (default), symlinks to directories are included in 'filenames' list. This is NOT a bug. http://python.6.x6.nabble.com/os-walk-with-followlinks-False-td3559133.html

Parameters

- **top** – Directory path to walk
- **topdown** – If topdown is True or not specified, the triple for a directory is generated before the triples for any of its subdirectories. If topdown is False, the triple for a directory is generated after the triples for all of its subdirectories.
- **onerror** – If optional argument onerror is specified, it should be a function; it will be called with one argument, an OSError instance. It can report the error to continue with the walk, or raise exception to abort the walk.
- **followlinks** – Set followlinks to True to visit directories pointed to by symlinks.

Raises OSError on failure if onerror is None

Yields a 3-tuple (dirpath, dirnames, filenames) where dirpath is a string, the path to the directory. dirnames is a list of the names of the subdirectories in dirpath (excluding '.' and '..'). filenames is a list of the names of the non-directory files in dirpath.

class gluster.gfapi.**DirEntry**(*vol*, *scandir_path*, *name*, *lstat*)

Object yielded by scandir() to expose the file path and other file attributes of a directory entry. scandir() will provide stat information without making additional calls. DirEntry instances are not intended to be stored in long-lived data structures; if you know the file metadata has changed or if a long time has elapsed since calling scandir(), call Volume.stat(entry.path) to fetch up-to-date information.

DirEntry() instances from Python 3.5 have follow_symlinks set to True by default. In this implementation, follow_symlinks is set to False by default as it incurs an additional stat call over network.

inode()

Return the inode number of the entry.

is_dir(*follow_symlinks=False*)

Return True if this entry is a directory; return False if the entry is any other kind of file, or if it doesn't exist anymore.

is_file(*follow_symlinks=False*)

Return True if this entry is a file; return False if the entry is a directory or other non-file entry, or if it doesn't exist anymore.

is_symlink()

Return True if this entry is a symbolic link (even if broken); return False if the entry points to a directory or any kind of file, or if it doesn't exist anymore.

name

The entry's base filename, relative to the scandir() path argument.

path
The entry's full path name: equivalent to os.path.join(scandir_path, entry.name) where scandir_path is the scandir() path argument. The path is only absolute if the scandir() path argument was absolute.

stat (*follow_symlinks=False*)
Returns information equivalent of a lstat() system call on the entry. This does not follow symlinks by default.

1.1.2 File Class

class gluster.gfapi.**File**(*fd*, *path=None*, *mode=None*)

__init__(*fd*, *path=None*, *mode=None*)
Create a File object equivalent to Python's built-in File object.

Parameters

- **fd** – glfd pointer
- **path** – Path of the file. This is optional.
- **mode** – The I/O mode of the file.

close(***kwargs*)
Close the file. A closed file cannot be read or written any more.

Raises OSError on failure

closed
Bool indicating the current state of the file object. This is a read-only attribute; the close() method changes the value.

discard(***kwargs*)
This is similar to UNMAP command that is used to return the unused/freed blocks back to the storage system. In this implementation, fallocate with FALLOC_FL_PUNCH_HOLE is used to eventually release the blocks to the filesystem. If the brick has been mounted with '-o discard' option, then the discard request will eventually reach the SCSI storage if the storage device supports UNMAP.

Parameters

- **offset** – Starting offset
- **len** – Length or size in bytes to discard

Raises OSError on failure

dup(***kwargs*)
Return a duplicate of File object. This duplicate File class instance encapsulates a duplicate glfd obtained by invoking glfs_dup().

Raises OSError on failure

fallocate(***kwargs*)
This is a Linux-specific sys call, unlike posix_fallocate()

Allows the caller to directly manipulate the allocated disk space for the file for the byte range starting at offset and continuing for length bytes.

Parameters

- **mode** – Operation to be performed on the given range
- **offset** – Starting offset

- **length** – Size in bytes, starting at offset

Raises OSError on failure

fchmod(***kwargs*)
Change this file's mode

Parameters **mode** – new mode

Raises OSError on failure

fchown(***kwargs*)
Change this file's owner and group id

Parameters

- **uid** – new user id for file
- **gid** – new group id for file

Raises OSError on failure

fdatasync(***kwargs*)
Flush buffer cache pages pertaining to data, but not the ones pertaining to metadata.

Raises OSError on failure

fgetsize(***kwargs*)
Return the size of a file, as reported by fstat()

Returns the size of the file in bytes

fgetxattr(***kwargs*)
Retrieve the value of the extended attribute identified by key for the file.

Parameters

- **key** – Key of extended attribute
- **size** – If size is specified as zero, we first determine the size of xattr and then allocate a buffer accordingly. If size is non-zero, it is assumed the caller knows the size of xattr.

Returns Value of extended attribute corresponding to key specified.

Raises OSError on failure

fileno
Return the internal file descriptor (glfd) that is used by the underlying implementation to request I/O operations.

flistxattr(***kwargs*)
Retrieve list of extended attributes for the file.

Parameters **size** – If size is specified as zero, we first determine the size of list and then allocate a buffer accordingly. If size is non-zero, it is assumed the caller knows the size of the list.

Returns List of extended attributes.

Raises OSError on failure

fremovexattr(***kwargs*)
Remove a extended attribute of the file.

Parameters **key** – The key of extended attribute.

Raises OSError on failure

fsetxattr(***kwargs*)
Set extended attribute of file.

Parameters

- **key** – The key of extended attribute.
- **value** – The valiue of extended attribute.
- **flags** – Possible values are 0 (default), 1 and 2. If 0 - xattr will be created if it does not exist, or the value will be replaced if the xattr exists. If 1 - it performs a pure create, which fails if the named attribute already exists. If 2 - it performs a pure replace operation, which fails if the named attribute does not already exist.

Raises OSError on failure

fstat(***kwargs*)
Returns Stat object for this file.

Returns Returns the stat information of the file.

Raises OSError on failure

fsync(***kwargs*)
Flush buffer cache pages pertaining to data and metadata.

Raises OSError on failure

ftruncate(***kwargs*)
Truncated the file to a size of length bytes.

If the file previously was larger than this size, the extra data is lost. If the file previously was shorter, it is extended, and the extended part reads as null bytes.

Parameters **length** – Length to truncate the file to in bytes.

Raises OSError on failure

lseek(***kwargs*)
Set the read/write offset position of this file. The new position is defined by 'pos' relative to 'how'

Parameters

- **pos** – sets new offset position according to 'how'
- **how** – SEEK_SET, sets offset position 'pos' bytes relative to beginning of file, SEEK_CUR, the position is set relative to the current position and SEEK_END sets the position relative to the end of the file.

Returns the new offset position

Raises OSError on failure

mode
The I/O mode for the file. If the file was created using the Volume.fopen() function, this will be the value of the mode parameter. This is a read-only attribute.

name
If the file object was created using Volume.fopen(), the name of the file.

read(***kwargs*)
Read at most size bytes from the file.

Parameters **buflen** – length of read buffer. If less than 0, then whole file is read. Default is -1.

Returns buffer of 'size' length

Raises OSError on failure

readinto(***kwargs*)
Read up to len(buf) bytes into buf which must be a bytearray. (buf cannot be a string as strings are immutable in python)

This method is useful when you have to read a large file over multiple read calls. While read() allocates a buffer every time it's invoked, readinto() copies data to an already allocated buffer passed to it.

Returns the number of bytes read (0 for EOF).

Raises OSError on failure

write(***kwargs*)
Write data to the file.

Parameters **data** – The data to be written to file.

Returns The size in bytes actually written

Raises OSError on failure

zerofill(***kwargs*)
Fill 'length' number of bytes with zeroes starting from 'offset'.

Parameters

- **offset** – Start at offset location
- **length** – Size/length in bytes

Raises OSError on failure

1.2 Contact

You can get in touch with the developer & user community in any of the following ways:

- IRC: `#gluster-dev` on Freenode
- Mailing list: `gluster-devel@gluster.org` (see the gluster-devel homepage for usage details)

1.3 Installation

Install libgfapi-python from yum/dnf repo:

```
$ sudo dnf install python-glusterfs-api
```

OR

Install libgfapi-python from PyPI:

```
$ sudo pip install gfapi
```

OR

Install libgfapi-python from source:

```
$ git clone https://review.gluster.org/libgfapi-python
$ cd libgfapi-python
$ sudo python setup.py install
```

Note: libgfapi-python bindings has been tested only against **Linux x86-64** and Python versions **2.7** and **>=3.4**.

1.4 Troubleshooting

1.4.1 Mount GlusterFS volume as non-root user (Glusterfs <3.7.3)

Changed in version 3.7.3: GlusterFS versions prior to version **3.7.3** requires the following additional steps to allow non-root users to mount the volume over libgfapi. Following these steps is **NOT required** for recent versions i.e >= 3.7.3

One can follow the following steps to allow a non-root user to virtual mount a GlusterFS volume over libgfapi. This requires a configuration change which enables GlusterFS server to accept client connections from non-priveledged ports.

```
# gluster volume set <volname> server.allow-insecure on
# gluster volume stop <volname>
# gluster volume start <volname>
```

Edit */etc/glusterfs/glusterd.vol* or */usr/local/etc/glusterfs/glusterd.vol* and set:

```
option rpc-auth-allow-insecure on
```

Restart glusterd service:

```
# service glusterd restart
```

Further, use *chown* and/or *chmod* commands to change permissions on mount point or required directories to allow non-root access to appropriate users.

Index

Symbols