
rfslib

Release 2.0.5

Přemysl Šťastný

Aug 13, 2021

CONTENTS:

1	rfslib.abstract_pconnection module	3
2	rfslib.sftp_pconnection module	7
3	rfslib.ftp_pconnection module	11
4	rfslib.smb12_pconnection module	15
5	rfslib.smb23_pconnection module	19
6	rfslib.fs_pconnection module	23
7	rfslib.path_utils module	27
	Python Module Index	29
	Index	31

This is a documentation of rfslib.

To create a new development enviroment, it is recommended to create python virtual enviroment and install dependencies in requirements.txt

If you want to create a new pdf documentation, you are required to install also texlive on your system.

RFSLIB.ABSTRACT_PCONNECTION MODULE

```
class rfslib.abstract_pconnection.PConnection(settings:
                                                    rfslib.abstract_pconnection.p_connection_settings)
    Bases: abc.ABC
    __init__(settings: rfslib.abstract_pconnection.p_connection_settings)
        The constructor of a abstract class. If it is not called from child class, the behavior is undefined.
        If local_encoding and remote_encoding have same values, no recoding is done. Analogically if local_crlf
        and remote_crlf is same, no substitution between LF and CRLF is done.
        Parameters settings – A p_connection_settings object with all generic settings for PConnec-
            tion.
    abstract _exists(remote_path: str) → bool
        Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns
        False.
        Parameters remote_path – Path of a remote file.
        Returns True, if remote file is exist. False, if remote file doesn't exist
    abstract _isdir(remote_path: str) → bool
        Protected method which checks, whether a remote file is a directory.
        Parameters remote_path – A path of a directory.
        Returns True, if remote file is folder. False, if it isn't a folder. Undefined if the file doesn't exist.
    abstract _lexists(remote_path: str) → bool
        Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns
        True.
        KNOWN BUG: Behavior is undefined in case of broken symlinks.
        Parameters remote_path – Path of a remote file.
        Returns True, if remote file is exist. False, if remote file doesn't exist
    abstract _listdir(remote_path: str) → List[str]
        Protected method which returns a list of files in the folder including hidden files. Undefined if the remote
        file doesn't exist or isn't a folder.
        Parameters remote_path – The remote path of a remote folder.
        Returns List of files in the remote folder
    abstract _mkdir(remote_path: str)
        Protected method which creates a new directory. Behavior is undefined if remote folder already exist, or
        destination folder doesn't exist.
```

Parameters **remote_path** – A path of a new remote directory.

abstract **_pull**(*remote_path: str, local_path: str*)

Protected method which downloads/pulls a nondirectory file from a remote storage to a local storage in the binary form. Behavior is undefined if source file or destination folder doesn't exist.

Parameters

- **remote_path** – Path of a remote file to download.
- **local_path** – Path of a local file, where to download/pull a remote file or local file already exists.

abstract **_push**(*local_path: str, remote_path: str*)

Protected method which uploads/pushes a nondirectory file from a local storage to a remote storage in the binary form. Behavior is undefined if destination folder or source file doesn't exist, source is directory or remote file already exists.

Parameters

- **local_path** – Path of a local file to upload.
- **remote_path** – Path on the remote storage, where to upload/push a local file.

abstract **_rename**(*old_name: str, new_name: str*)

Protected method which renames/moves a file. Behavior is undefined, if *new_name* file exists or *old_name* file doesn't exist.

Parameters

- **old_name** – Remote path a file to move.
- **new_name** – Remote path to which move the file.

abstract **_rmdir**(*remote_path: str*)

Protected method which removes an empty remote directory. Behavior is undefined if remote directory doesn't exist or it isn't empty.

Parameters **remote_path** – Path of an empty remote directory to delete.

abstract **_stat**(*remote_path: str*) → `os.stat_result`

Protected method which returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file. Undefined behavior if remote file doesn't exist or it is a broken symlink.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exist. False, if remote file doesn't exist.

abstract **_unlink**(*remote_path: str*)

Protected method which removes a nondirectory file. Behavior is undefined if remote file doesn't exist or is a directory.

Parameters **remote_path** – Path of a remote regular file to delete.

abstract **close**()

Method to close the opened connection.

cp(*old_names, new_name, recursive=False*)

dcp(*old_names, target_dir, recursive=False*)

dmv(*old_names, target_dir*)

exists(*remote_path: str*) → bool

Method which checks, whether a remote file exist. Returns False for broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exists. False, if remote file doesn't exist.

fcv(*old_name*, *new_name*)

find(*remote_path*, *child_first=False*)

fmv(*old_name*, *new_name*)

get_settings() → *rfslib.abstract_pconnection.p_connection_settings*

The procedure sets all generic settings for PConnection.

Returns A *p_connection_settings* object with all generic settings of PConnection.

isdir(*remote_path*)

lexists(*remote_path*)

Method which checks, whether a remote file exist. Returns True for broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exists. False, if remote file doesn't exist.

listdir(*remote_path*)

ls(*remote_path*)

lstat(*remote_path: str*) → *os.stat_result*

Returns statistics of a file (eg. size, last date modified,...) Doesn't follow symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file is exist. False, if remote file doesn't exist.

mkdir(*remote_path*)

mv(*old_names*, *new_name*)

pmkdir(*remote_path*)

pull(*remote_path*, *local_path*)

push(*local_path*, *remote_path*)

Uploads/pushes a file from a local storage to a remote storage in the binary form.

Parameters

- **local_path** – Path of a local file to upload.
- **remote_path** – Path on the remote storage, where to upload/push a local file.

rename(*old_name*, *new_name*)

rm(*remote_path*, *recursive=False*)

rmdir(*remote_path*)

rpull(*remote_path*, *local_path*)

rpush(*local_path*, *remote_path*)

set_settings(*settings: rfslib.abstract_pconnection.p_connection_settings*)

The procedure sets all generic settings for PConnection.

Parameters **settings** – A *p_connection_settings* object with all generic settings for PConnection.

stat(*remote_path: str*) → os.stat_result

Returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exist. False, if remote file doesn't exist.

touch(*remote_path*)

unlink(*remote_path*)

xls(*remote_path*)

class rfslib.abstract_pconnection.p_connection_settings

Bases: object

This object represents settings applicable for all PConnection instances (instances of class, which inherits from PConnection).

__init__()

The constructor inicializes the class to default values.

direct_write: bool = False

NOT IMPLEMENTED YET. If True, push will write output directly to file. If False all push operations on regular files will create firstly tmp file in target folder and then move result to file.

local_crlf: bool = False

Does local files use CRLF? If True, it is supposed, they do. If False, it is supposed, they use LF.

local_encoding: str = 'UTF8'

The encoding of local text files. (eg. 'UTF8')

remote_crlf: bool = False

Does remote files use CRLF? If True, it is supposed, they do. If False, it is supposed, they use LF.

remote_encoding: str = 'UTF8'

The encoding of remote text files. (eg. 'cp1250')

skip_validation: bool = False

NOT IMPLEMENTED YED. If True, all validations of input will be skipped. Undefined behavior may happen if input is wrong. Increses performance.

text_transmission: bool = False

If true, all files, which will be transmitted, will be recoded from local_encoding to remote_encoding and from local_crlf to remote_crlf. If False, there will be no encoding done during transmission.

RFSLIB.SFTP_PCONNECTION MODULE

```
class rfslib.sftp_pconnection.SftpPConnection(settings:
    rfslib.abstract_pconnection.p_connection_settings, host:
    str, username: str, password: Optional[str] = None,
    key_filename: str = '~/.ssh/id_rsa', port: int = 22,
    no_host_key_checking: bool = False)
```

Bases: `rfslib.abstract_pconnection.PConnection`

Class for SFTP connection. Public interface with an exception of `__init__` and `close` is inherited from `PConnection`.

```
__init__(settings: rfslib.abstract_pconnection.p_connection_settings, host: str, username: str, password:
    Optional[str] = None, key_filename: str = '~/.ssh/id_rsa', port: int = 22, no_host_key_checking:
    bool = False)
```

The constructor of `SftpPConnection`. Opens SFTP connection, when called. If None password is specified, the key authentication will be used. Otherwise the password authentication will be used.

Parameters

- **settings** – The settings for the super class `PConnection`.
- **host** – Remote address of the server.
- **port** – Port for the SFTP connection.
- **username** – Remote username
- **password** – Password for a SFTP connection. If None is provided, key authentication will be used.
- **key_filename** – A path to key.
- **no_host_key_checking** – Specifies, whether remote host key should be verified or not.

```
_exists(remote_path)
```

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns False.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file is exist. False, if remote file doesn't exist

```
_isdir(remote_path)
```

Protected method which checks, whether a remote file is a directory.

Parameters **remote_path** – A path of a directory.

Returns True, if remote file is folder. False, if it isn't a folder. Undefined if the file doesn't exist.

_lexists(*remote_path*)

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns True.

KNOWN BUG: Behavior is undefined in case of broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file is exist. False, if remote file doesn't exist

_listdir(*remote_path*)

Protected method which returns a list of files in the folder including hidden files. Undefined if the remote file doesn't exist or isn't a folder.

Parameters **remote_path** – The remote path of a remote folder.

Returns List of files in the remote folder

_mkdir(*remote_path*)

Protected method which creates a new directory. Behavior is undefined if remote folder already exist, or destination folder doesn't exist.

Parameters **remote_path** – A path of a new remote directory.

_pull(*remote_path*, *local_path*)

Protected method which downloads/pulls a nondirectory file from a remote storage to a local storage in the binary form. Behavior is undefined if source file or destination folder doesn't exist.

Parameters

- **remote_path** – Path of a remote file to download.
- **local_path** – Path of a local file, where to download/pull a remote file or local file already exists.

_push(*local_path*, *remote_path*)

Protected method which uploads/pushes a nondirectory file from a local storage to a remote storage in the binary form. Behavior is undefined if destination folder or source file doesn't exist, source is directory or remote file already exists.

Parameters

- **local_path** – Path of a local file to upload.
- **remote_path** – Path on the remote storage, where to upload/push a local file.

_rename(*old_name*, *new_name*)

Protected method which renames/moves a file. Behavior is undefined, if *new_name* file exists or *old_name* file doesn't exist.

Parameters

- **old_name** – Remote path a file to move.
- **new_name** – Remote path to which move the file.

_rmdir(*remote_path*)

Protected method which removes an empty remote directory. Behavior is undefined if remote directory doesn't exist or it isn't empty.

Parameters **remote_path** – Path of an empty remote directory to delete.

_stat(*remote_path*)

Protected method which returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file. Undefined behavior if remote file doesn't exist or it is a broken symlink.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exist. False, if remote file doesn't exist.

_unlink(*remote_path*)

Protected method which removes a nondirectory file. Behavior is undefined if remote file doesn't exist or is a directory.

Parameters **remote_path** – Path of a remote regular file to delete.

close()

Method to close the opened connection.

cp(*old_names*, *new_name*, *recursive=False*)

dcp(*old_names*, *target_dir*, *recursive=False*)

dmv(*old_names*, *target_dir*)

exists(*remote_path: str*) → bool

Method which checks, whether a remote file exist. Returns False for broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exists. False, if remote file doesn't exist.

fcop(*old_name*, *new_name*)

find(*remote_path*, *child_first=False*)

fmv(*old_name*, *new_name*)

get_settings() → *rfslib.abstract_pconnection.p_connection_settings*

The procedure sets all generic settings for PConnection.

Returns A *p_connection_settings* object with all generic settings of PConnection.

isdir(*remote_path*)

lexists(*remote_path*)

Method which checks, whether a remote file exist. Returns True for broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exists. False, if remote file doesn't exist.

listdir(*remote_path*)

ls(*remote_path*)

lstat(*remote_path: str*) → *os.stat_result*

Returns statistics of a file (eg. size, last date modified,...) Doesn't follow symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file is exist. False, if remote file doesn't exist.

mkdir(*remote_path*)

mv(*old_names*, *new_name*)

pmkdir(*remote_path*)

pull(*remote_path*, *local_path*)

push(*local_path*, *remote_path*)

Uploads/pushes a file from a local storage to a remote storage in the binary form.

Parameters

- **local_path** – Path of a local file to upload.
- **remote_path** – Path on the remote storage, where to upload/push a local file.

rename(*old_name*, *new_name*)

rm(*remote_path*, *recursive=False*)

rmdir(*remote_path*)

rpull(*remote_path*, *local_path*)

rpush(*local_path*, *remote_path*)

set_settings(*settings*: [rfslib.abstract_pconnection.p_connection_settings](#))

The procedure sets all generic settings for PConnection.

Parameters settings – A `p_connection_settings` object with all generic settings for PConnection.

stat(*remote_path*: *str*) → `os.stat_result`

Returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file.

Parameters remote_path – Path of a remote file.

Returns True, if remote file exist. False, if remote file doesn't exist.

touch(*remote_path*)

unlink(*remote_path*)

xls(*remote_path*)

RFSLIB.FTP_PCONNECTION MODULE

```
class rfslib.ftp_pconnection.FtpPConnection(settings:
    rfslib.abstract_pconnection.p_connection_settings, host:
    str, username: str, password: str, port: int = 21, tls: bool
    = False, passive_mode: bool = False, debug_level: int = 1,
    connection_encoding: str = 'UTF8')
```

Bases: *rfslib.abstract_pconnection.PConnection*

Class for FTP connection. Public interface with an exception of `__init__` and `close` is inherited from `PConnection`.

```
__init__(settings: rfslib.abstract_pconnection.p_connection_settings, host: str, username: str, password:
    str, port: int = 21, tls: bool = False, passive_mode: bool = False, debug_level: int = 1,
    connection_encoding: str = 'UTF8')
```

The constructor of `FtpPConnection`.

Parameters

- **settings** – The settings for the super class `PConnection`.
- **host** – Remote address of the server.
- **port** – Port for a connection.
- **username** – Remote username.
- **password** – Remote password.
- **tls** – Enables TLS.
- **passive_mode** – Enables passive mode of FTP connection.
- **debug_level** – Specifies how much logs should be generated. 0 - almost non, 1 - more, 2 - log almost everything
- **connection_encoding** – Encoding used for a connection.

```
_exists(remote_path)
```

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns `False`.

Parameters **remote_path** – Path of a remote file.

Returns `True`, if remote file is exist. `False`, if remote file doesn't exist

```
_isdir(remote_path)
```

Protected method which checks, whether a remote file is a directory.

Parameters **remote_path** – A path of a directory.

Returns `True`, if remote file is folder. `False`, if it isn't a folder. Undefined if the file doesn't exist.

_lexists(*remote_path*)

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns True.

KNOWN BUG: Behavior is undefined in case of broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file is exist. False, if remote file doesn't exist

_listdir(*remote_path*)

Protected method which returns a list of files in the folder including hidden files. Undefined if the remote file doesn't exist or isn't a folder.

Parameters **remote_path** – The remote path of a remote folder.

Returns List of files in the remote folder

_mkdir(*remote_path*)

Protected method which creates a new directory. Behavior is undefined if remote folder already exist, or destination folder doesn't exist.

Parameters **remote_path** – A path of a new remote directory.

_pull(*remote_path*, *local_path*)

Protected method which downloads/pulls a nondirectory file from a remote storage to a local storage in the binary form. Behavior is undefined if source file or destination folder doesn't exist.

Parameters

- **remote_path** – Path of a remote file to download.
- **local_path** – Path of a local file, where to download/pull a remote file or local file already exists.

_push(*local_path*, *remote_path*)

Protected method which uploads/pushes a nondirectory file from a local storage to a remote storage in the binary form. Behavior is undefined if destination folder or source file doesn't exist, source is directory or remote file already exists.

Parameters

- **local_path** – Path of a local file to upload.
- **remote_path** – Path on the remote storage, where to upload/push a local file.

_rename(*old_name*, *new_name*)

Protected method which renames/moves a file. Behavior is undefined, if *new_name* file exists or *old_name* file doesn't exist.

Parameters

- **old_name** – Remote path a file to move.
- **new_name** – Remote path to which move the file.

_rmdir(*remote_path*)

Protected method which removes an empty remote directory. Behavior is undefined if remote directory doesn't exist or it isn't empty.

Parameters **remote_path** – Path of an empty remote directory to delete.

_stat(*remote_path*)

Protected method which returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file. Undefined behavior if remote file doesn't exist or it is a broken symlink.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exist. False, if remote file doesn't exist.

_unlink(*remote_path*)

Protected method which removes a nondirectory file. Behavior is undefined if remote file doesn't exist or is a directory.

Parameters **remote_path** – Path of a remote regular file to delete.

close()

Method to close the opened connection.

cp(*old_names*, *new_name*, *recursive=False*)

dcp(*old_names*, *target_dir*, *recursive=False*)

dmv(*old_names*, *target_dir*)

exists(*remote_path: str*) → bool

Method which checks, whether a remote file exist. Returns False for broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exists. False, if remote file doesn't exist.

fcop(*old_name*, *new_name*)

find(*remote_path*, *child_first=False*)

fmv(*old_name*, *new_name*)

get_settings() → *rfslib.abstract_pconnection.p_connection_settings*

The procedure sets all generic settings for PConnection.

Returns A *p_connection_settings* object with all generic settings of PConnection.

isdir(*remote_path*)

lexists(*remote_path*)

Method which checks, whether a remote file exist. Returns True for broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exists. False, if remote file doesn't exist.

listdir(*remote_path*)

ls(*remote_path*)

lstat(*remote_path: str*) → *os.stat_result*

Returns statistics of a file (eg. size, last date modified,...) Doesn't follow symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file is exist. False, if remote file doesn't exist.

mkdir(*remote_path*)

mv(*old_names*, *new_name*)

pmkdir(*remote_path*)

pull(*remote_path*, *local_path*)

push(*local_path*, *remote_path*)

Uploads/pushes a file from a local storage to a remote storage in the binary form.

Parameters

- **local_path** – Path of a local file to upload.
- **remote_path** – Path on the remote storage, where to upload/push a local file.

rename(*old_name*, *new_name*)

rm(*remote_path*, *recursive=False*)

rmdir(*remote_path*)

rpull(*remote_path*, *local_path*)

rpush(*local_path*, *remote_path*)

set_settings(*settings*: [rfslib.abstract_pconnection.p_connection_settings](#))

The procedure sets all generic settings for PConnection.

Parameters settings – A `p_connection_settings` object with all generic settings for PConnection.

stat(*remote_path*: *str*) → `os.stat_result`

Returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file.

Parameters remote_path – Path of a remote file.

Returns True, if remote file exist. False, if remote file doesn't exist.

touch(*remote_path*)

unlink(*remote_path*)

xls(*remote_path*)

RFSLIB.SMB12_PCONNECTION MODULE

```
class rfslib.smb12_pconnection.Smb12PConnection(settings:
    rfslib.abstract_pconnection.p_connection_settings,
    host: str, service_name: str, username: str, password:
    str, port: int = 139, use_direct_tcp: bool = False,
    client_name: str = 'RFS', use_ntlm_v1: bool = False)
```

Bases: *rfslib.abstract_pconnection.PConnection*

Class for SMB connection version 1 or 2. Public interface with an exception of `__init__` and `close` is inherited from `PConnection`.

```
__init__(settings: rfslib.abstract_pconnection.p_connection_settings, host: str, service_name: str,
    username: str, password: str, port: int = 139, use_direct_tcp: bool = False, client_name: str =
    'RFS', use_ntlm_v1: bool = False)
```

The constructor of `Smb12PConnection`. Opens SMB connection version 1 or 2, when called.

Parameters

- **settings** – The settings for the super class `PConnection`.
- **host** – Remote address of the server.
- **service_name** – Name of a shared folder.
- **port** – Port for the SMB connection.
- **username** – Remote username.
- **password** – Remote password.
- **use_direct_tcp** – Activates direct tcp mode for SMB.
- **client_name** – Name of this client, which will be sent to a server.
- **use_ntlm_v1** – Enables NTLM version 1 instead of NTLM version 2.

```
_exists(remote_path)
```

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns `False`.

Parameters **remote_path** – Path of a remote file.

Returns `True`, if remote file is exist. `False`, if remote file doesn't exist

```
_isdir(remote_path)
```

Protected method which checks, whether a remote file is a directory.

Parameters **remote_path** – A path of a directory.

Returns `True`, if remote file is folder. `False`, if it isn't a folder. Undefined if the file doesn't exist.

_lexists(*remote_path*)

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns True.

KNOWN BUG: Behavior is undefined in case of broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file is exist. False, if remote file doesn't exist

_listdir(*remote_path*)

Protected method which returns a list of files in the folder including hidden files. Undefined if the remote file doesn't exist or isn't a folder.

Parameters **remote_path** – The remote path of a remote folder.

Returns List of files in the remote folder

_mkdir(*remote_path*)

Protected method which creates a new directory. Behavior is undefined if remote folder already exist, or destination folder doesn't exist.

Parameters **remote_path** – A path of a new remote directory.

_pull(*remote_path*, *local_path*)

Protected method which downloads/pulls a nondirectory file from a remote storage to a local storage in the binary form. Behavior is undefined if source file or destination folder doesn't exist.

Parameters

- **remote_path** – Path of a remote file to download.
- **local_path** – Path of a local file, where to download/pull a remote file or local file already exists.

_push(*local_path*, *remote_path*)

Protected method which uploads/pushes a nondirectory file from a local storage to a remote storage in the binary form. Behavior is undefined if destination folder or source file doesn't exist, source is directory or remote file already exists.

Parameters

- **local_path** – Path of a local file to upload.
- **remote_path** – Path on the remote storage, where to upload/push a local file.

_rename(*old_name*, *new_name*)

Protected method which renames/moves a file. Behavior is undefined, if *new_name* file exists or *old_name* file doesn't exist.

Parameters

- **old_name** – Remote path a file to move.
- **new_name** – Remote path to which move the file.

_rmdir(*remote_path*)

Protected method which removes an empty remote directory. Behavior is undefined if remote directory doesn't exist or it isn't empty.

Parameters **remote_path** – Path of an empty remote directory to delete.

_stat(*remote_path*)

Protected method which returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file. Undefined behavior if remote file doesn't exist or it is a broken symlink.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exist. False, if remote file doesn't exist.

_unlink(*remote_path*)

Protected method which removes a nondirectory file. Behavior is undefined if remote file doesn't exist or is a directory.

Parameters **remote_path** – Path of a remote regular file to delete.

close()

Method to close the opened connection.

cp(*old_names, new_name, recursive=False*)

dcp(*old_names, target_dir, recursive=False*)

dmv(*old_names, target_dir*)

exists(*remote_path: str*) → bool

Method which checks, whether a remote file exist. Returns False for broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exists. False, if remote file doesn't exist.

fcop(*old_name, new_name*)

find(*remote_path, child_first=False*)

fmv(*old_name, new_name*)

get_settings() → *rfslib.abstract_pconnection.p_connection_settings*

The procedure sets all generic settings for PConnection.

Returns A *p_connection_settings* object with all generic settings of PConnection.

isdir(*remote_path*)

lexists(*remote_path*)

Method which checks, whether a remote file exist. Returns True for broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exists. False, if remote file doesn't exist.

listdir(*remote_path*)

ls(*remote_path*)

lstat(*remote_path: str*) → *os.stat_result*

Returns statistics of a file (eg. size, last date modified,...) Doesn't follow symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file is exist. False, if remote file doesn't exist.

mkdir(*remote_path*)

mv(*old_names, new_name*)

pmkdir(*remote_path*)

pull(*remote_path, local_path*)

push(*local_path, remote_path*)

Uploads/pushes a file from a local storage to a remote storage in the binary form.

Parameters

- **local_path** – Path of a local file to upload.
- **remote_path** – Path on the remote storage, where to upload/push a local file.

rename(*old_name*, *new_name*)

rm(*remote_path*, *recursive=False*)

rmdir(*remote_path*)

rpull(*remote_path*, *local_path*)

rpush(*local_path*, *remote_path*)

set_settings(*settings*: [rfslib.abstract_pconnection.p_connection_settings](#))

The procedure sets all generic settings for PConnection.

Parameters settings – A `p_connection_settings` object with all generic settings for PConnection.

stat(*remote_path*: *str*) → `os.stat_result`

Returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file.

Parameters remote_path – Path of a remote file.

Returns True, if remote file exist. False, if remote file doesn't exist.

touch(*remote_path*)

unlink(*remote_path*)

xls(*remote_path*)

RFSLIB.SMB23_PCONNECTION MODULE

```
class rfslib.smb23_pconnection.Smb23PConnection(settings:
    rfslib.abstract_pconnection.p_connection_settings,
    host: str, service_name: str, username: str, password:
    str, port: int = 445, enable_encryption: bool = False,
    dont_require_signing: bool = False)
```

Bases: *rfslib.abstract_pconnection.PConnection*

Class for SMB connection version 2 or 3. Public interface with an exception of `__init__` and `close` is inherited from `PConnection`.

```
__init__(settings: rfslib.abstract_pconnection.p_connection_settings, host: str, service_name: str,
    username: str, password: str, port: int = 445, enable_encryption: bool = False,
    dont_require_signing: bool = False)
```

The constructor of `Smb23PConnection`. Opens SMB connection version 2 or 3, when called.

Parameters

- **settings** – The settings for the super class `PConnection`.
- **service_name** – Name of a shared folder.
- **host** – Remote address of the server.
- **port** – Port for a SMB connection.
- **username** – Remote username
- **password** – Password for a SMB connection.
- **enable_encryption** – Enables encryption for a SMB3 connection.
- **dont_require_signing** – Disables signing requirement.

```
_exists(remote_path)
```

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns `False`.

Parameters **remote_path** – Path of a remote file.

Returns `True`, if remote file is exist. `False`, if remote file doesn't exist

```
_isdir(remote_path)
```

Protected method which checks, whether a remote file is a directory.

Parameters **remote_path** – A path of a directory.

Returns `True`, if remote file is folder. `False`, if it isn't a folder. Undefined if the file doesn't exist.

_lexists(*remote_path*)

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns True.

KNOWN BUG: Behavior is undefined in case of broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file is exist. False, if remote file doesn't exist

_listdir(*remote_path*)

Protected method which returns a list of files in the folder including hidden files. Undefined if the remote file doesn't exist or isn't a folder.

Parameters **remote_path** – The remote path of a remote folder.

Returns List of files in the remote folder

_mkdir(*remote_path*)

Protected method which creates a new directory. Behavior is undefined if remote folder already exist, or destination folder doesn't exist.

Parameters **remote_path** – A path of a new remote directory.

_pull(*remote_path*, *local_path*)

Protected method which downloads/pulls a nondirectory file from a remote storage to a local storage in the binary form. Behavior is undefined if source file or destination folder doesn't exist.

Parameters

- **remote_path** – Path of a remote file to download.
- **local_path** – Path of a local file, where to download/pull a remote file or local file already exists.

_push(*local_path*, *remote_path*)

Protected method which uploads/pushes a nondirectory file from a local storage to a remote storage in the binary form. Behavior is undefined if destination folder or source file doesn't exist, source is directory or remote file already exists.

Parameters

- **local_path** – Path of a local file to upload.
- **remote_path** – Path on the remote storage, where to upload/push a local file.

_rename(*old_name*, *new_name*)

Protected method which renames/moves a file. Behavior is undefined, if *new_name* file exists or *old_name* file doesn't exist.

Parameters

- **old_name** – Remote path a file to move.
- **new_name** – Remote path to which move the file.

_rmdir(*remote_path*)

Protected method which removes an empty remote directory. Behavior is undefined if remote directory doesn't exist or it isn't empty.

Parameters **remote_path** – Path of an empty remote directory to delete.

_stat(*remote_path*)

Protected method which returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file. Undefined behavior if remote file doesn't exist or it is a broken symlink.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exist. False, if remote file doesn't exist.

_unlink(*remote_path*)

Protected method which removes a nondirectory file. Behavior is undefined if remote file doesn't exist or is a directory.

Parameters **remote_path** – Path of a remote regular file to delete.

close()

Method to close the opened connection.

cp(*old_names*, *new_name*, *recursive=False*)

dcp(*old_names*, *target_dir*, *recursive=False*)

dmv(*old_names*, *target_dir*)

exists(*remote_path: str*) → bool

Method which checks, whether a remote file exist. Returns False for broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exists. False, if remote file doesn't exist.

fcop(*old_name*, *new_name*)

find(*remote_path*, *child_first=False*)

fmv(*old_name*, *new_name*)

get_settings() → *rfslib.abstract_pconnection.p_connection_settings*

The procedure sets all generic settings for PConnection.

Returns A *p_connection_settings* object with all generic settings of PConnection.

isdir(*remote_path*)

lexists(*remote_path*)

Method which checks, whether a remote file exist. Returns True for broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exists. False, if remote file doesn't exist.

listdir(*remote_path*)

ls(*remote_path*)

lstat(*remote_path: str*) → *os.stat_result*

Returns statistics of a file (eg. size, last date modified,...) Doesn't follow symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file is exist. False, if remote file doesn't exist.

mkdir(*remote_path*)

mv(*old_names*, *new_name*)

pmkdir(*remote_path*)

pull(*remote_path*, *local_path*)

push(*local_path*, *remote_path*)

Uploads/pushes a file from a local storage to a remote storage in the binary form.

Parameters

- **local_path** – Path of a local file to upload.
- **remote_path** – Path on the remote storage, where to upload/push a local file.

rename(*old_name*, *new_name*)

rm(*remote_path*, *recursive=False*)

rmdir(*remote_path*)

rpull(*remote_path*, *local_path*)

rpush(*local_path*, *remote_path*)

set_settings(*settings*: [rfslib.abstract_pconnection.p_connection_settings](#))

The procedure sets all generic settings for PConnection.

Parameters settings – A `p_connection_settings` object with all generic settings for PConnection.

stat(*remote_path*: *str*) → `os.stat_result`

Returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file.

Parameters remote_path – Path of a remote file.

Returns True, if remote file exist. False, if remote file doesn't exist.

touch(*remote_path*)

unlink(*remote_path*)

xls(*remote_path*)

`rfslib.smb23_pconnection.config_smb23`(*no_dfs*: *bool = False*, *disable_secure_negotiate*: *bool = False*,
dfs_domain_controller: *Optional[str] = None*)

The procedure changes global setting for SMB version 2 or 3 across all connection. Don't change value, if any SMB connection version 2 or 3 is active.

Parameters

- **no_dfs** – Disables DFS support - useful as a bug fix.
- **disable_secure_negotiate** – Disables secure negotiate requirement for a SMB connection.
- **dfs_domain_controller** – The DFS domain controller address. Useful in case, when `rfstools` fails to find it itself.

RFSLIB.FS_PCONNECTION MODULE

class `rfslib.fs_pconnection.FsPConnection(settings: rfslib.abstract_pconnection.p_connection_settings)`
Bases: `rfslib.abstract_pconnection.PConnection`

Class for operating with local filesystem. Public interface with an exception of `__init__` and `close` is inherited from `PConnection`.

`__init__`(*settings: rfslib.abstract_pconnection.p_connection_settings*)
The constructor of `FsPConnection`.

Parameters **settings** – The settings for super class `PConnection`.

`_exists`(*remote_path*)
Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns `False`.

Parameters **remote_path** – Path of a remote file.

Returns `True`, if remote file is exist. `False`, if remote file doesn't exist

`_isdir`(*remote_path*)
Protected method which checks, whether a remote file is a directory.

Parameters **remote_path** – A path of a directory.

Returns `True`, if remote file is folder. `False`, if it isn't a folder. Undefined if the file doesn't exist.

`_lexists`(*remote_path*)
Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns `True`.

KNOWN BUG: Behavior is undefined in case of broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns `True`, if remote file is exist. `False`, if remote file doesn't exist

`_listdir`(*remote_path*)
Protected method which returns a list of files in the folder including hidden files. Undefined if the remote file doesn't exist or isn't a folder.

Parameters **remote_path** – The remote path of a remote folder.

Returns List of files in the remote folder

`_mkdir`(*remote_path*)
Protected method which creates a new directory. Behavior is undefined if remote folder already exist, or destination folder doesn't exist.

Parameters **remote_path** – A path of a new remote directory.

`_pull(remote_path, local_path)`

Protected method which downloads/pulls a nondirectory file from a remote storage to a local storage in the binary form. Behavior is undefined if source file or destination folder doesn't exist.

Parameters

- **`remote_path`** – Path of a remote file to download.
- **`local_path`** – Path of a local file, where to download/pull a remote file or local file already exists.

`_push(local_path, remote_path)`

Protected method which uploads/pushes a nondirectory file from a local storage to a remote storage in the binary form. Behavior is undefined if destination folder or source file doesn't exist, source is directory or remote file already exists.

Parameters

- **`local_path`** – Path of a local file to upload.
- **`remote_path`** – Path on the remote storage, where to upload/push a local file.

`_rename(old_name, new_name)`

Protected method which renames/moves a file. Behavior is undefined, if *new_name* file exists or *old_name* file doesn't exist.

Parameters

- **`old_name`** – Remote path a file to move.
- **`new_name`** – Remote path to which move the file.

`_rmdir(remote_path)`

Protected method which removes an empty remote directory. Behavior is undefined if remote directory doesn't exist or it isn't empty.

Parameters **`remote_path`** – Path of an empty remote directory to delete.

`_stat(remote_path)`

Protected method which returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file. Undefined behavior if remote file doesn't exist or it is a broken symlink.

Parameters **`remote_path`** – Path of a remote file.

Returns True, if remote file exist. False, if remote file doesn't exist.

`_unlink(remote_path)`

Protected method which removes a nondirectory file. Behavior is undefined if remote file doesn't exist or is a directory.

Parameters **`remote_path`** – Path of a remote regular file to delete.

`close()`

Method to close the opened connection.

`cp(old_names, new_name, recursive=False)`

`dcp(old_names, target_dir, recursive=False)`

`dmv(old_names, target_dir)`

`exists(remote_path: str) → bool`

Method which checks, whether a remote file exist. Returns False for broken symlinks.

Parameters **`remote_path`** – Path of a remote file.

Returns True, if remote file exists. False, if remote file doesn't exist.

fc*p*(old_name, new_name)

find(remote_path, child_first=False)

fm*v*(old_name, new_name)

get_settings() → *rfslib.abstract_pconnection.p_connection_settings*

The procedure sets all generic settings for PConnection.

Returns A p_connection_settings object with all generic settings of PConnection.

isdir(remote_path)

lexists(remote_path)

Method which checks, whether a remote file exist. Returns True for broken symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exists. False, if remote file doesn't exist.

listdir(remote_path)

ls(remote_path)

lstat(remote_path: str) → os.stat_result

Returns statistics of a file (eg. size, last date modified,...) Doesn't follow symlinks.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file is exist. False, if remote file doesn't exist.

mkdir(remote_path)

mv(old_names, new_name)

pmkdir(remote_path)

pull(remote_path, local_path)

push(local_path, remote_path)

Uploads/pushes a file from a local storage to a remote storage in the binary form.

Parameters

- **local_path** – Path of a local file to upload.
- **remote_path** – Path on the remote storage, where to upload/push a local file.

rename(old_name, new_name)

rm(remote_path, recursive=False)

rmdir(remote_path)

rpull(remote_path, local_path)

rpush(local_path, remote_path)

set_settings(settings: *rfslib.abstract_pconnection.p_connection_settings*)

The procedure sets all generic settings for PConnection.

Parameters **settings** – A p_connection_settings object with all generic settings for PConnection.

stat(remote_path: str) → os.stat_result

Returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file.

Parameters **remote_path** – Path of a remote file.

Returns True, if remote file exist. False, if remote file doesn't exist.

touch(*remote_path*)

unlink(*remote_path*)

xls(*remote_path*)

RFSLIB.PATH_UTILS MODULE

```
class rfslib.path_utils.GenericPath(path)
    Bases: object
        __init__(path)
rfslib.path_utils.add_r_prefix(path)
rfslib.path_utils.generic_cp(conn, sources, dest, recursive=False)
rfslib.path_utils.generic_mv(conn, sources, dest)
rfslib.path_utils.generic_path_normalize(path)
rfslib.path_utils.is_remote(path)
rfslib.path_utils.path_normalize(path)
rfslib.path_utils.remove_r_prefix(path)
```


PYTHON MODULE INDEX

r

- `rfslib.abstract_pconnection`, [3](#)
- `rfslib.fs_pconnection`, [23](#)
- `rfslib.ftp_pconnection`, [11](#)
- `rfslib.path_utils`, [27](#)
- `rfslib.sftp_pconnection`, [7](#)
- `rfslib.smb12_pconnection`, [15](#)
- `rfslib.smb23_pconnection`, [19](#)

Symbols

<code>__init__()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 3	<code>_lexists()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 23
<code>__init__()</code> (<i>rfslib.abstract_pconnection.p_connection_settings</i> method), 6	<code>_lexists()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 11
<code>__init__()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 23	<code>_lexists()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 7
<code>__init__()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 11	<code>_lexists()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 15
<code>__init__()</code> (<i>rfslib.path_utils.GenericPath</i> method), 27	<code>_lexists()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 19
<code>__init__()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 7	<code>_listdir()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 3
<code>__init__()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 15	<code>_listdir()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 23
<code>__init__()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 19	<code>_listdir()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 12
<code>_exists()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 3	<code>_listdir()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 8
<code>_exists()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 23	<code>_listdir()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 16
<code>_exists()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 11	<code>_listdir()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 20
<code>_exists()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 7	<code>_mkdir()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 3
<code>_exists()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 15	<code>_mkdir()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 23
<code>_exists()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 19	<code>_mkdir()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 12
<code>_isdir()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 3	<code>_mkdir()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 8
<code>_isdir()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 23	<code>_mkdir()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 16
<code>_isdir()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 11	<code>_mkdir()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 20
<code>_isdir()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 7	<code>_pull()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 4
<code>_isdir()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 15	<code>_pull()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 23
<code>_isdir()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 19	<code>_pull()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 12
<code>_lexists()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 3	<code>_pull()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 8

<code>_pull()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 16	<code>_unlink()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 24
<code>_pull()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 20	<code>_unlink()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 13
<code>_push()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 4	<code>_unlink()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 9
<code>_push()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 24	<code>_unlink()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 17
<code>_push()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 12	<code>_unlink()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 21
<code>_push()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 8	
<code>_push()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 16	A
<code>_push()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 20	<code>add_r_prefix()</code> (in module <i>rfslib.path_utils</i>), 27
<code>_rename()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 4	C
<code>_rename()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 24	<code>close()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 4
<code>_rename()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 12	<code>close()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 24
<code>_rename()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 8	<code>close()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 13
<code>_rename()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 16	<code>close()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 9
<code>_rename()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 20	<code>close()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 17
<code>_rmdir()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 4	<code>close()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 21
<code>_rmdir()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 24	<code>config_smb23()</code> (in module <i>rfslib.smb23_pconnection</i>), 22
<code>_rmdir()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 12	<code>cp()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 4
<code>_rmdir()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 8	<code>cp()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 24
<code>_rmdir()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 16	<code>cp()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 13
<code>_rmdir()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 20	<code>cp()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 9
<code>_stat()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 4	<code>cp()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 17
<code>_stat()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 24	<code>cp()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 21
<code>_stat()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 12	
<code>_stat()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 8	D
<code>_stat()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 16	<code>dcp()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 4
<code>_stat()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 20	<code>dcp()</code> (<i>rfslib.fs_pconnection.FsPConnection</i> method), 24
<code>_unlink()</code> (<i>rfslib.abstract_pconnection.PConnection</i> method), 4	<code>dcp()</code> (<i>rfslib.ftp_pconnection.FtpPConnection</i> method), 13
	<code>dcp()</code> (<i>rfslib.sftp_pconnection.SftpPConnection</i> method), 9
	<code>dcp()</code> (<i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 17
	<code>dcp()</code> (<i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 21

`direct_write(rfslib.abstract_pconnection.p_connection_attribute)`, 6

`dmv()` (*rfslib.abstract_pconnection.PConnection method*), 4

`dmv()` (*rfslib.fs_pconnection.FsPConnection method*), 24

`dmv()` (*rfslib.ftp_pconnection.FtpPConnection method*), 13

`dmv()` (*rfslib.sftp_pconnection.SftpPConnection method*), 9

`dmv()` (*rfslib.smb12_pconnection.Smb12PConnection method*), 17

`dmv()` (*rfslib.smb23_pconnection.Smb23PConnection method*), 21

E

`exists()` (*rfslib.abstract_pconnection.PConnection method*), 4

`exists()` (*rfslib.fs_pconnection.FsPConnection method*), 24

`exists()` (*rfslib.ftp_pconnection.FtpPConnection method*), 13

`exists()` (*rfslib.sftp_pconnection.SftpPConnection method*), 9

`exists()` (*rfslib.smb12_pconnection.Smb12PConnection method*), 17

`exists()` (*rfslib.smb23_pconnection.Smb23PConnection method*), 21

F

`fcv()` (*rfslib.abstract_pconnection.PConnection method*), 5

`fcv()` (*rfslib.fs_pconnection.FsPConnection method*), 25

`fcv()` (*rfslib.ftp_pconnection.FtpPConnection method*), 13

`fcv()` (*rfslib.sftp_pconnection.SftpPConnection method*), 9

`fcv()` (*rfslib.smb12_pconnection.Smb12PConnection method*), 17

`fcv()` (*rfslib.smb23_pconnection.Smb23PConnection method*), 21

`find()` (*rfslib.abstract_pconnection.PConnection method*), 5

`find()` (*rfslib.fs_pconnection.FsPConnection method*), 25

`find()` (*rfslib.ftp_pconnection.FtpPConnection method*), 13

`find()` (*rfslib.sftp_pconnection.SftpPConnection method*), 9

`find()` (*rfslib.smb12_pconnection.Smb12PConnection method*), 17

`find()` (*rfslib.smb23_pconnection.Smb23PConnection method*), 21

`fmv()` (*rfslib.abstract_pconnection.PConnection method*), 5

`fmv()` (*rfslib.fs_pconnection.FsPConnection method*), 25

`fmv()` (*rfslib.ftp_pconnection.FtpPConnection method*), 13

`fmv()` (*rfslib.sftp_pconnection.SftpPConnection method*), 9

`fmv()` (*rfslib.smb12_pconnection.Smb12PConnection method*), 17

`fmv()` (*rfslib.smb23_pconnection.Smb23PConnection method*), 21

`FsPConnection` (class in *rfslib.fs_pconnection*), 23

`FtpPConnection` (class in *rfslib.ftp_pconnection*), 11

G

`generic_cp()` (in module *rfslib.path_utils*), 27

`generic_mv()` (in module *rfslib.path_utils*), 27

`generic_path_normalize()` (in module *rfslib.path_utils*), 27

`GenericPath` (class in *rfslib.path_utils*), 27

`get_settings()` (*rfslib.abstract_pconnection.PConnection method*), 5

`get_settings()` (*rfslib.fs_pconnection.FsPConnection method*), 25

`get_settings()` (*rfslib.ftp_pconnection.FtpPConnection method*), 13

`get_settings()` (*rfslib.sftp_pconnection.SftpPConnection method*), 9

`get_settings()` (*rfslib.smb12_pconnection.Smb12PConnection method*), 17

`get_settings()` (*rfslib.smb23_pconnection.Smb23PConnection method*), 21

I

`is_remote()` (in module *rfslib.path_utils*), 27

`isdir()` (*rfslib.abstract_pconnection.PConnection method*), 5

`isdir()` (*rfslib.fs_pconnection.FsPConnection method*), 25

`isdir()` (*rfslib.ftp_pconnection.FtpPConnection method*), 13

`isdir()` (*rfslib.sftp_pconnection.SftpPConnection method*), 9

`isdir()` (*rfslib.smb12_pconnection.Smb12PConnection method*), 17

`isdir()` (*rfslib.smb23_pconnection.Smb23PConnection method*), 21

L

`lexists()` (*rfslib.abstract_pconnection.PConnection method*), 5

`lexists()` (*rfslib.fs_pconnection.FsPConnection method*), 25

`lexists()` (*rfslib.ftp_pconnection.FtpPConnection method*), 13

lexists() (rfslib.sftp_pconnection.SftpPConnection method), 9	mkdir() (rfslib.sftp_pconnection.SftpPConnection method), 9
lexists() (rfslib.smb12_pconnection.Smb12PConnection method), 17	mkdir() (rfslib.smb12_pconnection.Smb12PConnection method), 17
lexists() (rfslib.smb23_pconnection.Smb23PConnection method), 21	mkdir() (rfslib.smb23_pconnection.Smb23PConnection method), 21
listdir() (rfslib.abstract_pconnection.PConnection method), 5	module
listdir() (rfslib.fs_pconnection.FsPConnection method), 25	rfslib.abstract_pconnection, 3
listdir() (rfslib.ftp_pconnection.FtpPConnection method), 13	rfslib.fs_pconnection, 23
listdir() (rfslib.sftp_pconnection.SftpPConnection method), 9	rfslib.ftp_pconnection, 11
listdir() (rfslib.smb12_pconnection.Smb12PConnection method), 17	rfslib.path_utils, 27
listdir() (rfslib.smb23_pconnection.Smb23PConnection method), 21	rfslib.sftp_pconnection, 7
local_crlf (rfslib.abstract_pconnection.p_connection_settings attribute), 6	rfslib.smb12_pconnection, 15
local_encoding (rfslib.abstract_pconnection.p_connection_settings attribute), 6	rfslib.smb23_pconnection, 19
ls() (rfslib.abstract_pconnection.PConnection method), 5	mv() (rfslib.abstract_pconnection.PConnection method), 5
ls() (rfslib.fs_pconnection.FsPConnection method), 25	mv() (rfslib.fs_pconnection.FsPConnection method), 25
ls() (rfslib.ftp_pconnection.FtpPConnection method), 13	mv() (rfslib.ftp_pconnection.FtpPConnection method), 13
ls() (rfslib.sftp_pconnection.SftpPConnection method), 9	mv() (rfslib.sftp_pconnection.SftpPConnection method), 9
ls() (rfslib.smb12_pconnection.Smb12PConnection method), 17	mv() (rfslib.smb12_pconnection.Smb12PConnection method), 17
ls() (rfslib.smb23_pconnection.Smb23PConnection method), 21	mv() (rfslib.smb23_pconnection.Smb23PConnection method), 21
lstat() (rfslib.abstract_pconnection.PConnection method), 5	
lstat() (rfslib.fs_pconnection.FsPConnection method), 25	
lstat() (rfslib.ftp_pconnection.FtpPConnection method), 13	
lstat() (rfslib.sftp_pconnection.SftpPConnection method), 9	
lstat() (rfslib.smb12_pconnection.Smb12PConnection method), 17	
lstat() (rfslib.smb23_pconnection.Smb23PConnection method), 21	

M

mkdir() (rfslib.abstract_pconnection.PConnection method), 5	
mkdir() (rfslib.fs_pconnection.FsPConnection method), 25	
mkdir() (rfslib.ftp_pconnection.FtpPConnection method), 13	

P

p_connection_settings (class in rfslib.abstract_pconnection), 6	
path_normalize() (in module rfslib.path_utils), 27	
PConnection (class in rfslib.abstract_pconnection), 3	
pmkdir() (rfslib.abstract_pconnection.PConnection method), 5	
pmkdir() (rfslib.fs_pconnection.FsPConnection method), 25	
pmkdir() (rfslib.ftp_pconnection.FtpPConnection method), 13	
pmkdir() (rfslib.sftp_pconnection.SftpPConnection method), 9	
pmkdir() (rfslib.smb12_pconnection.Smb12PConnection method), 17	
pmkdir() (rfslib.smb23_pconnection.Smb23PConnection method), 21	
pull() (rfslib.abstract_pconnection.PConnection method), 5	
pull() (rfslib.fs_pconnection.FsPConnection method), 25	
pull() (rfslib.ftp_pconnection.FtpPConnection method), 13	
pull() (rfslib.sftp_pconnection.SftpPConnection method), 9	
pull() (rfslib.smb12_pconnection.Smb12PConnection method), 17	

pull() (*rfslib.smb23_pconnection.Smb23PConnection* method), 21
 push() (*rfslib.abstract_pconnection.PConnection* method), 5
 push() (*rfslib.fs_pconnection.FsPConnection* method), 25
 push() (*rfslib.ftp_pconnection.FtpPConnection* method), 13
 push() (*rfslib.sftp_pconnection.SftpPConnection* method), 9
 push() (*rfslib.smb12_pconnection.Smb12PConnection* method), 17
 push() (*rfslib.smb23_pconnection.Smb23PConnection* method), 21

R

remote_crlf(*rfslib.abstract_pconnection.p_connection_settings* attribute), 6
 remote_encoding (*rfslib.abstract_pconnection.p_connection_settings* attribute), 6
 remove_r_prefix() (in module *rfslib.path_utils*), 27
 rename() (*rfslib.abstract_pconnection.PConnection* method), 5
 rename() (*rfslib.fs_pconnection.FsPConnection* method), 25
 rename() (*rfslib.ftp_pconnection.FtpPConnection* method), 14
 rename() (*rfslib.sftp_pconnection.SftpPConnection* method), 10
 rename() (*rfslib.smb12_pconnection.Smb12PConnection* method), 18
 rename() (*rfslib.smb23_pconnection.Smb23PConnection* method), 22
 rfslib.abstract_pconnection module, 3
 rfslib.fs_pconnection module, 23
 rfslib.ftp_pconnection module, 11
 rfslib.path_utils module, 27
 rfslib.sftp_pconnection module, 7
 rfslib.smb12_pconnection module, 15
 rfslib.smb23_pconnection module, 19
 rm() (*rfslib.abstract_pconnection.PConnection* method), 5
 rm() (*rfslib.fs_pconnection.FsPConnection* method), 25
 rm() (*rfslib.ftp_pconnection.FtpPConnection* method), 14
 rm() (*rfslib.sftp_pconnection.SftpPConnection* method), 10
 rm() (*rfslib.smb12_pconnection.Smb12PConnection* method), 18
 rm() (*rfslib.smb23_pconnection.Smb23PConnection* method), 22
 rmdir() (*rfslib.abstract_pconnection.PConnection* method), 5
 rmdir() (*rfslib.fs_pconnection.FsPConnection* method), 25
 rmdir() (*rfslib.ftp_pconnection.FtpPConnection* method), 14
 rmdir() (*rfslib.sftp_pconnection.SftpPConnection* method), 10
 rmdir() (*rfslib.smb12_pconnection.Smb12PConnection* method), 18
 rmdir() (*rfslib.smb23_pconnection.Smb23PConnection* method), 22
 rpull() (*rfslib.abstract_pconnection.PConnection* method), 5
 rpull() (*rfslib.fs_pconnection.FsPConnection* method), 25
 rpull() (*rfslib.ftp_pconnection.FtpPConnection* method), 14
 rpull() (*rfslib.sftp_pconnection.SftpPConnection* method), 10
 rpull() (*rfslib.smb12_pconnection.Smb12PConnection* method), 18
 rpull() (*rfslib.smb23_pconnection.Smb23PConnection* method), 22
 rpush() (*rfslib.abstract_pconnection.PConnection* method), 5
 rpush() (*rfslib.fs_pconnection.FsPConnection* method), 25
 rpush() (*rfslib.ftp_pconnection.FtpPConnection* method), 14
 rpush() (*rfslib.sftp_pconnection.SftpPConnection* method), 10
 rpush() (*rfslib.smb12_pconnection.Smb12PConnection* method), 18
 rpush() (*rfslib.smb23_pconnection.Smb23PConnection* method), 22

S

set_settings() (*rfslib.abstract_pconnection.PConnection* method), 5
 set_settings() (*rfslib.fs_pconnection.FsPConnection* method), 25
 set_settings() (*rfslib.ftp_pconnection.FtpPConnection* method), 14
 set_settings() (*rfslib.sftp_pconnection.SftpPConnection* method), 10
 set_settings() (*rfslib.smb12_pconnection.Smb12PConnection* method), 18

~~set_settings()~~ (*rfslib.smb23_pconnection.Smb23PConnection*
method), 22

SftpPConnection (class in *rfslib.sftp_pconnection*), 7

skip_validation (*rfslib.abstract_pconnection.p_connection_settings*
attribute), 6

Smb12PConnection (class in *rfslib.smb12_pconnection*),
15

Smb23PConnection (class in *rfslib.smb23_pconnection*),
19

stat() (*rfslib.abstract_pconnection.PConnection*
method), 5

stat() (*rfslib.fs_pconnection.FsPConnection* method),
25

stat() (*rfslib.ftp_pconnection.FtpPConnection*
method), 14

stat() (*rfslib.sftp_pconnection.SftpPConnection*
method), 10

stat() (*rfslib.smb12_pconnection.Smb12PConnection*
method), 18

stat() (*rfslib.smb23_pconnection.Smb23PConnection*
method), 22

T

text_transmission (*rfslib.abstract_pconnection.p_connection_settings*
attribute), 6

touch() (*rfslib.abstract_pconnection.PConnection*
method), 6

touch() (*rfslib.fs_pconnection.FsPConnection* method),
26

touch() (*rfslib.ftp_pconnection.FtpPConnection*
method), 14

touch() (*rfslib.sftp_pconnection.SftpPConnection*
method), 10

touch() (*rfslib.smb12_pconnection.Smb12PConnection*
method), 18

touch() (*rfslib.smb23_pconnection.Smb23PConnection*
method), 22

U

unlink() (*rfslib.abstract_pconnection.PConnection*
method), 6

unlink() (*rfslib.fs_pconnection.FsPConnection*
method), 26

unlink() (*rfslib.ftp_pconnection.FtpPConnection*
method), 14

unlink() (*rfslib.sftp_pconnection.SftpPConnection*
method), 10

unlink() (*rfslib.smb12_pconnection.Smb12PConnection*
method), 18

unlink() (*rfslib.smb23_pconnection.Smb23PConnection*
method), 22