pysmb Documentation

Release 1.2.7

Michael Teo

May 30, 2021

Contents

pysmb is a pure Python implementation of the client-side SMB/CIFS protocol (SMB1 and SMB2) which is the underlying protocol that facilitates file sharing and printing between Windows machines, as well as with Linux machines via the Samba server application. pysmb is developed in Python 2.7.x and Python 3.8.x and has been tested against shared folders on Windows 7, Windows 10 and Samba 4.x.

The latest version of pysmb is always available at the pysmb project page at miketeo.net.

CHAPTER 1

License

pysmb itself is licensed under an opensource license. You are free to use pysmb in any applications, including for commercial purposes. For more details on the terms of use, please read the LICENSE file that comes with your pysmb source.

pysmb depends on other 3rd-party modules whose terms of use are not covered by pysmb. Use of these modules could possibly conflict with your licensing needs. Please exercise your own discretion to determine their suitabilities. I have listed these modules in the following section.

CHAPTER 2

Credits

pysmb is not alone. It is made possible with support from other modules.

- **pyasn1** : Pure Python implementation of ASN.1 parsing and encoding (not included together with pysmb; needs to be installed separately)
- **md4** and **U32** : Pure Python implementation of MD4 hashing algorithm and 32-bit unsigned integer by Dmitry Rozmanov. Licensed under LGPL and included together with pysmb.
- **pyDes** : Pure Python implementation of the DES encryption algorithm by Todd Whiteman. Free domain and included together with pysmb.
- **sha256** : Pure Python implementation of SHA-256 message digest by Thomas Dixon. Licensed under MIT and included together with pysmb. This module is imported only when the Python standard library (usually Python 2.4) does not provide the SHA-256 hash algorithm.

In various places, there are references to different specifications. Most of these referenced specifications can be downloaded from Microsoft web site under Microsoft's "Open Specification Promise". If you need to download a copy of these specifications, please google for it. For example, google for "MS-CIFS" to download the CIFS specification for NT LM dialect.

CHAPTER 3

Package Contents and Description

pysmb is organized into 2 main packages: smb and nmb. The smb package contains all the functionalities related to Server Message Block (SMB) implementation. As an application developer, you will be importing this module into your application. Hence, please take some time to familiarize yourself with the smb package contents.

- **nmb/base.py** : Contains the NetBIOSSession and NBNS abstract class which implements NetBIOS session and NetBIOS Name Service communication without any network transport specifics.
- **nmb/NetBIOS.py**: Provides a NBNS implementation to query IP addresses for machine names. All operations are blocking I/O.
- **nmb/NetBIOSProtocol.py** : Provides the NBNS protocol implementation for use in Twisted framework.
- **smb/base.py** : Contains the SMB abstract class which implements the SMB communication without any network transport specifics.
- **smb/ntlm.py** : Contains the NTLMv1 and NTLMv2 authentication routines and the decoding/encoding of NTLM authentication messages within SMB messages.
- **smb/securityblob.py** : Provides routines to encode/decode the NTLMSSP security blob in the SMB messages.
- **smb/smb_constants.py** : All the constants used in the smb package for SMB1 protocol
- **smb/smb_structs.py** : Contains the internal classes used in the SMB package for SMB1 protocol. These classes are usually used to encode/decode the parameter and data blocks of specific SMB1 message.
- **smb/smb2_constants.py** : All the constants used in the smb package for SMB2 protocol
- **smb/smb2_structs.py** : Contains the internal classes used in the SMB package for SMB2 protocol. These classes are usually used to encode/decode the parameter and data blocks of specific SMB2 message.
- **smb/SMBConnection.py** : Contains a SMB protocol implementation. All operations are blocking I/O.
- **smb/SMBProtocol.py** : Contains the SMB protocol implementation for use in the Twisted framework.
- **smb/SMBHandler.py** : Provides support for "smb://" URL in the urllib2 python package.

CHAPTER 4

Using pysmb

As an application developer who is looking to use pysmb to translate NetBIOS names to IP addresses,

- To use pysmb in applications where you want the file operations to return after they have completed (synchronous style), please read *nmb.NetBIOS.NetBIOS* documentation.
- To use pysmb in Twisted, please read *nmb.NetBIOSProtocol.NBNSProtocol* documentation.

As an application developer who is looking to use pysmb to implement file transfer or authentication over SMB:

- To use pysmb in applications where you want the file operations to return after they have completed (synchronous style), please read *smb.SMBConnection.SMBConnection* documentation.
- To use pysmb in Twisted, please read *smb.SMBProtocol.SMBProtocolFactory* documentation.
- To support "smb://" URL in urllib2 python package, read *smb.SMBHandler.SMBHandler* documentation.

As a software developer who is looking to modify pysmb so that you can integrate it to other network frameworks:

- Read *Extending pysmb For Other Frameworks*

If you are upgrading from older pysmb versions:

- Read *Upgrading from older pysmb versions*

CHAPTER 5

Indices and tables

5.1 NBNSProtocol Class

pysmb has a *NBNSProtocol* implementation for Twisted framework. This allows you to perform name query asynchronously without having your application to block and wait for the results.

In your project,

1. Create a NBNSProtocol instance.
2. Just call *queryName* method which will return a *Deferred* instance. Add your callback function to the *Deferred* instance via *addCallback* method to receive the results of the name query.
3. When you are done with the NBNSProtocol instance, call its <NBNSProtocol instance>.transport.stopListening method to remove this instance from the reactor.

class nmb.NetBIOSProtocol.**NBNSProtocol**(*broadcast=True*, *listen_port=0*)

__init__(*broadcast=True*, *listen_port=0*)
Instantiate a NBNSProtocol instance.

This automatically calls reactor.listenUDP method to start listening for incoming packets, so you **must not** call the listenUDP method again.

Parameters

- **broadcast** (*boolean*) – A boolean flag to indicate if we should setup the listening UDP port in broadcast mode
- **listen_port** (*integer*) – Specifies the UDP port number to bind to for listening. If zero, OS will automatically select a free port number.

queryIPForName(*ip*, *port=137*, *timeout=30*)
Send a query to the machine with *ip* and hopes that the machine will reply back with its name.

The implementation of this function is contributed by Jason Anderson.

Parameters

- **ip** (*string*) – If the NBNSProtocol instance was instianted with broadcast=True, then this parameter can be an empty string. We will leave it to the OS to determine an appropriate broadcast address. If the NBNSProtocol instance was instianted with broadcast=False, then you should provide a target IP to send the query.
- **port** (*integer*) – The NetBIOS-NS port (IANA standard defines this port to be 137). You should not touch this parameter unless you know what you are doing.
- **timeout** (*integer/float*) – Number of seconds to wait for a reply, after which the method will return None

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with a list of names of the machine at *ip*. On timeout, the errback function will be called with a Failure instance wrapping around a NetBIOSTimeout exception

queryName(*name*, *ip=''*, *port=137*, *timeout=30*)

Send a query on the network and hopes that if machine matching the *name* will reply with its IP address.

Parameters

- **ip** (*string*) – If the NBNSProtocol instance was instianted with broadcast=True, then this parameter can be an empty string. We will leave it to the OS to determine an appropriate broadcast address. If the NBNSProtocol instance was instianted with broadcast=False, then you should provide a target IP to send the query.
- **port** (*integer*) – The NetBIOS-NS port (IANA standard defines this port to be 137). You should not touch this parameter unless you know what you are doing.
- **timeout** (*integer/float*) – Number of seconds to wait for a reply, after which the returned Deferred instance will be called with a NetBIOSTimeout exception.

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with a list of IP addresses in dotted notation (aaa.bbb.ccc.ddd). On timeout, the errback function will be called with a Failure instance wrapping around a NetBIOSTimeout exception

class nmb.NetBIOSProtocol.**NetBIOSTimeout**

Raised in NBNSProtocol via Deferred.errback method when queryName method has timeout waiting for reply

5.2 NetBIOS class

To use the NetBIOS class in your application,

1. Create a new NetBIOS instance
2. Call *queryName* method for each name you wish to query. The method will block until a reply is received from the remote SMB/CIFS service, or until timeout.
3. When you are done, call *close* method to release the underlying resources.

class nmb.NetBIOS.**NetBIOS**(*broadcast=True*, *listen_port=0*)

__init__(*broadcast=True*, *listen_port=0*)

Instantiate a NetBIOS instance, and creates a IPv4 UDP socket to listen/send NBNS packets.

Parameters

- **broadcast** (*boolean*) – A boolean flag to indicate if we should setup the listening UDP port in broadcast mode

- **listen_port** (*integer*) – Specifies the UDP port number to bind to for listening. If zero, OS will automatically select a free port number.

close()
Close the underlying and free resources.

The NetBIOS instance should not be used to perform any operations after this method returns.

Returns None

queryIPForName(*ip*, *port=137*, *timeout=30*)
Send a query to the machine with *ip* and hopes that the machine will reply back with its name.

The implementation of this function is contributed by Jason Anderson.

Parameters

- **ip** (*string*) – If the NBNSProtocol instance was instianted with broadcast=True, then this parameter can be an empty string. We will leave it to the OS to determine an appropriate broadcast address. If the NBNSProtocol instance was instianted with broadcast=False, then you should provide a target IP to send the query.
- **port** (*integer*) – The NetBIOS-NS port (IANA standard defines this port to be 137). You should not touch this parameter unless you know what you are doing.
- **timeout** (*integer/float*) – Number of seconds to wait for a reply, after which the method will return None

Returns A list of string containing the names of the machine at *ip*. On timeout, returns None.

queryName(*name*, *ip=''*, *port=137*, *timeout=30*)
Send a query on the network and hopes that if machine matching the *name* will reply with its IP address.

Parameters

- **ip** (*string*) – If the NBNSProtocol instance was instianted with broadcast=True, then this parameter can be an empty string. We will leave it to the OS to determine an appropriate broadcast address. If the NBNSProtocol instance was instianted with broadcast=False, then you should provide a target IP to send the query.
- **port** (*integer*) – The NetBIOS-NS port (IANA standard defines this port to be 137). You should not touch this parameter unless you know what you are doing.
- **timeout** (*integer/float*) – Number of seconds to wait for a reply, after which the method will return None

Returns A list of IP addresses in dotted notation (aaa.bbb.ccc.ddd). On timeout, returns None.

5.3 SMBConnection Class

The SMBConnection is suitable for developers who wish to use pysmb to perform file operations with a remote SMB/CIFS server sequentially.

Each file operation method, when invoked, will block and return after it has completed or has encountered an error.

5.3.1 Example

The following illustrates a simple file retrieving implementation.:

```
import tempfile
from smb.SMBConnection import SMBConnection

# There will be some mechanism to capture userID, password, client_machine_name,␣
↪server_name and server_ip
# client_machine_name can be an arbitary ASCII string
# server_name should match the remote machine name, or else the connection will be␣
↪rejected
conn = SMBConnection(userID, password, client_machine_name, server_name, use_ntlm_v2␣
↪= True)
assert conn.connect(server_ip, 139)

file_obj = tempfile.NamedTemporaryFile()
file_attributes, filesize = conn.retrieveFile('smbtest', '/rfc1001.txt', file_obj)

# Retrieved file contents are inside file_obj
# Do what you need with the file_obj and then close it
# Note that the file obj is positioned at the end-of-file,
# so you might need to perform a file_obj.seek() if you need
# to read from the beginning
file_obj.close()
```

5.3.2 SMB2 Support

Starting from pysmb 1.1.0, pysmb will utilize SMB2 protocol for communication if the remote SMB/CIFS service supports SMB2. Otherwise, it will fallback automatically back to using SMB1 protocol.

To disable SMB2 protocol in pysmb, set the *SUPPORT_SMB2* flag in the *smb_structs* module to *False* before creating the *SMBConnection* instance.:

```
from smb import smb_structs
smb_structs.SUPPORT_SMB2 = False
```

5.3.3 Caveats

- It is not meant to be used asynchronously.
- A single *SMBConnection* instance should not be used to perform more than one operation concurrently at the same time.
- Do not keep a *SMBConnection* instance "idle" for too long, i.e. keeping a *SMBConnection* instance but not using it. Most SMB/CIFS servers have some sort of keepalive mechanism and impose a timeout limit. If the clients fail to respond within the timeout limit, the SMB/CIFS server may disconnect the client.

class smb.SMBConnection.**SMBConnection**(*username*, *password*, *my_name*, *remote_name*, *domain=''*, *use_ntlm_v2=True*, *sign_options=2*, *is_direct_tcp=False*)

__init__(*username*, *password*, *my_name*, *remote_name*, *domain=''*, *use_ntlm_v2=True*, *sign_options=2*, *is_direct_tcp=False*)

Create a new SMBConnection instance.

username and *password* are the user credentials required to authenticate the underlying SMB connection with the remote server. *password* can be a string or a callable returning a string. File operations can only be proceeded after the connection has been authenticated successfully.

Note that you need to call *connect* method to actually establish the SMB connection to the remote server and perform authentication.

The default TCP port for most SMB/CIFS servers using NetBIOS over TCP/IP is 139. Some newer server installations might also support Direct hosting of SMB over TCP/IP; for these servers, the default TCP port is 445.

Parameters

- **my_name** (*string*) – The local NetBIOS machine name that will identify where this connection is originating from. You can freely choose a name as long as it contains a maximum of 15 alphanumeric characters and does not contain spaces and any of `\/:*?";|+`
- **remote_name** (*string*) – The NetBIOS machine name of the remote server. On windows, you can find out the machine name by right-clicking on the "My Computer" and selecting "Properties". This parameter must be the same as what has been configured on the remote server, or else the connection will be rejected.
- **domain** (*string*) – The network domain. On windows, it is known as the workgroup. Usually, it is safe to leave this parameter as an empty string.
- **use_ntlm_v2** (*boolean*) – Indicates whether pysmb should be NTLMv1 or NTLMv2 authentication algorithm for authentication. The choice of NTLMv1 and NTLMv2 is configured on the remote server, and there is no mechanism to auto-detect which algorithm has been configured. Hence, we can only "guess" or try both algorithms. On Sambda, Windows Vista and Windows 7, NTLMv2 is enabled by default. On Windows XP, we can use NTLMv1 before NTLMv2.
- **sign_options** (*int*) – Determines whether SMB messages will be signed. Default is *SIGN_WHEN_REQUIRED*. If *SIGN_WHEN_REQUIRED* (value=2), SMB messages will only be signed when remote server requires signing. If *SIGN_WHEN_SUPPORTED* (value=1), SMB messages will be signed when remote server supports signing but not requires signing. If *SIGN_NEVER* (value=0), SMB messages will never be signed regardless of remote server's configurations; access errors will occur if the remote server requires signing.
- **is_direct_tcp** (*boolean*) – Controls whether the NetBIOS over TCP/IP (is_direct_tcp=False) or the newer Direct hosting of SMB over TCP/IP (is_direct_tcp=True) will be used for the communication. The default parameter is False which will use NetBIOS over TCP/IP for wider compatibility (TCP port: 139).

close()

Terminate the SMB connection (if it has been started) and release any sources held by the underlying socket.

connect(*ip*, *port=139*, *sock_family=2*, *timeout=60*)

Establish the SMB connection to the remote SMB/CIFS server.

You must call this method before attempting any of the file operations with the remote server. This method will block until the SMB connection has attempted at least one authentication.

Returns A boolean value indicating the result of the authentication atttempt: True if authentication is successful; False, if otherwise.

createDirectory(*service_name*, *path*, *timeout=30*)

Creates a new directory *path* on the *service_name*.

Parameters

- **service_name** (*string/unicode*) – Contains the name of the shared folder.

- **path** (*string/unicode*) – The path of the new folder (relative to) the shared folder. If the path contains non-English characters, an unicode string must be used to pass in the path.

Returns None

deleteDirectory(*service_name*, *path*, *timeout=30*)
Delete the empty folder at *path* on *service_name*

Parameters

- **service_name** (*string/unicode*) – Contains the name of the shared folder.
- **path** (*string/unicode*) – The path of the to-be-deleted folder (relative to) the shared folder. If the path contains non-English characters, an unicode string must be used to pass in the path.

Returns None

deleteFiles(*service_name*, *path_file_pattern*, *delete_matching_folders=False*, *timeout=30*)
Delete one or more regular files. It supports the use of wildcards in file names, allowing for deletion of multiple files in a single request.

If delete_matching_folders is True, immediate sub-folders that match the path_file_pattern will be deleted recursively.

Parameters

- **service_name** (*string/unicode*) – Contains the name of the shared folder.
- **path_file_pattern** (*string/unicode*) – The pathname of the file(s) to be deleted, relative to the service_name. Wildcards may be used in th filename component of the path. If your path/filename contains non-English characters, you must pass in an unicode string.

Returns None

echo(*data*, *timeout=10*)
Send an echo command containing *data* to the remote SMB/CIFS server. The remote SMB/CIFS will reply with the same *data*.

Parameters **data** (*bytes*) – Data to send to the remote server. Must be a bytes object.

Returns The *data* parameter

getAttributes(*service_name*, *path*, *timeout=30*)
Retrieve information about the file at *path* on the *service_name*.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – Path of the file on the remote server. If the file cannot be opened for reading, an *OperationFailure* will be raised.

Returns A *smb.base.SharedFile* instance containing the attributes of the file.

getSecurity(*service_name*, *path*, *timeout=30*)
Retrieve the security descriptor of the file at *path* on the *service_name*.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – Path of the file on the remote server. If the file cannot be opened for reading, an *OperationFailure* will be raised.

Returns A *smb.security_descriptors.SecurityDescriptor* instance containing the security information of the file.

listPath (*service_name*, *path*, *search=65591*, *pattern='*'*, *timeout=30*)
Retrieve a directory listing of files/folders at *path*

For simplicity, pysmb defines a "normal" file as a file entry that is not read-only, not hidden, not system, not archive and not a directory. It ignores other attributes like compression, indexed, sparse, temporary and encryption.

Note that the default search parameter will query for all read-only (SMB_FILE_ATTRIBUTE_READONLY), hidden (SMB_FILE_ATTRIBUTE_HIDDEN), system (SMB_FILE_ATTRIBUTE_SYSTEM), archive (SMB_FILE_ATTRIBUTE_ARCHIVE), normal (SMB_FILE_ATTRIBUTE_INCL_NORMAL) files and directories (SMB_FILE_ATTRIBUTE_DIRECTORY). If you do not need to include "normal" files in the result, define your own search parameter without the SMB_FILE_ATTRIBUTE_INCL_NORMAL constant. SMB_FILE_ATTRIBUTE_NORMAL should be used by itself and not be used with other bit constants.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – path relative to the *service_name* where we are interested to learn about its files/sub-folders.
- **search** (*integer*) – integer value made up from a bitwise-OR of *SMB_FILE_ATTRIBUTE_xxx* bits (see smb_constants.py).
- **pattern** (*string/unicode*) – the filter to apply to the results before returning to the client.

Returns A list of *smb.base.SharedFile* instances.

listShares (*timeout=30*)
Retrieve a list of shared resources on remote server.

Returns A list of *smb.base.SharedDevice* instances describing the shared resource

listSnapshots (*service_name*, *path*, *timeout=30*)
Retrieve a list of available snapshots (shadow copies) for *path*.

Note that snapshot features are only supported on Windows Vista Business, Enterprise and Ultimate, and on all Windows 7 editions.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – path relative to the *service_name* where we are interested in the list of available snapshots

Returns A list of python *datetime.DateTime* instances in GMT/UTC time zone

rename (*service_name*, *old_path*, *new_path*, *timeout=30*)
Rename a file or folder at *old_path* to *new_path* shared at *service_name*. Note that this method cannot be used to rename file/folder across different shared folders

old_path and *new_path* are string/unicode referring to the old and new path of the renamed resources (relative to) the shared folder. If the path contains non-English characters, an unicode string must be used to pass in the path.

Parameters **service_name** (*string/unicode*) – Contains the name of the shared folder.

Returns None

resetFileAttributes(*service_name*, *path_file_pattern*, *file_attributes=128*, *timeout=30*)

Reset file attributes of one or more regular files or folders. It supports the use of wildcards in file names, allowing for unlocking of multiple files/folders in a single request. This function is very helpful when deleting files/folders that are read-only. By default, it sets the ATTR_NORMAL flag, therefore clearing all other flags. (See https://msdn.microsoft.com/en-us/library/cc232110.aspx for further information)

Note: this function is currently only implemented for SMB2!

Parameters

- **service_name** (*string/unicode*) – Contains the name of the shared folder.
- **path_file_pattern** (*string/unicode*) – The pathname of the file(s) to be deleted, relative to the service_name. Wildcards may be used in the filename component of the path. If your path/filename contains non-English characters, you must pass in an unicode string.
- **file_attributes** (*int*) – The desired file attributes to set. Defaults to *ATTR_NORMAL*.

Returns None

retrieveFile(*service_name*, *path*, *file_obj*, *timeout=30*)

Retrieve the contents of the file at *path* on the *service_name* and write these contents to the provided *file_obj*.

Use *retrieveFileFromOffset()* method if you wish to specify the offset to read from the remote *path* and/or the number of bytes to write to the *file_obj*.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – Path of the file on the remote server. If the file cannot be opened for reading, an *OperationFailure* will be raised.
- **file_obj** – A file-like object that has a *write* method. Data will be written continuously to *file_obj* until EOF is received from the remote service.

Returns A 2-element tuple of (file attributes of the file on server, number of bytes written to *file_obj*). The file attributes is an integer value made up from a bitwise-OR of *SMB_FILE_ATTRIBUTE_xxx* bits (see smb_constants.py)

retrieveFileFromOffset(*service_name*, *path*, *file_obj*, *offset=0L*, *max_length=-1L*, *timeout=30*)

Retrieve the contents of the file at *path* on the *service_name* and write these contents to the provided *file_obj*.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – Path of the file on the remote server. If the file cannot be opened for reading, an *OperationFailure* will be raised.
- **file_obj** – A file-like object that has a *write* method. Data will be written continuously to *file_obj* up to *max_length* number of bytes.
- **offset** (*integer/long*) – the offset in the remote *path* where the first byte will be read and written to *file_obj*. Must be either zero or a positive integer/long value.

- **max_length** (*integer/long*) – maximum number of bytes to read from the remote *path* and write to the *file_obj*. Specify a negative value to read from *offset* to the EOF. If zero, the method returns immediately after the file is opened successfully for reading.

Returns A 2-element tuple of (file attributes of the file on server, number of bytes written to *file_obj*). The file attributes is an integer value made up from a bitwise-OR of *SMB_FILE_ATTRIBUTE_xxx* bits (see smb_constants.py)

storeFile(*service_name*, *path*, *file_obj*, *timeout=30*)
Store the contents of the *file_obj* at *path* on the *service_name*. If the file already exists on the remote server, it will be truncated and overwritten.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – Path of the file on the remote server. If the file at *path* does not exist, it will be created. Otherwise, it will be overwritten. If the *path* refers to a folder or the file cannot be opened for writing, an *OperationFailure* will be raised.
- **file_obj** – A file-like object that has a *read* method. Data will read continuously from *file_obj* until EOF.

Returns Number of bytes uploaded

storeFileFromOffset(*service_name*, *path*, *file_obj*, *offset=0L*, *truncate=False*, *timeout=30*)
Store the contents of the *file_obj* at *path* on the *service_name*.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – Path of the file on the remote server. If the file at *path* does not exist, it will be created. If the *path* refers to a folder or the file cannot be opened for writing, an *OperationFailure* will be raised.
- **file_obj** – A file-like object that has a *read* method. Data will read continuously from *file_obj* until EOF.
- **offset** – Long integer value which specifies the offset in the remote server to start writing. First byte of the file is 0.
- **truncate** – Boolean value. If True and the file exists on the remote server, it will be truncated first before writing. Default is False.

Returns the file position where the next byte will be written.

isUsingSMB2
A convenient property to return True if the underlying SMB connection is using SMB2 protocol.

5.4 SMbHandler Class

The SMBHandler class provides support for "smb://" URLs in the urllib2 python package.

5.4.1 Notes

- The host component of the URL must be one of the following:
  - A fully-qualified hostname that can be resolved by your local DNS service. Example: myserver.test.com

  - An IP address. Example: 192.168.1.1
  - A comma-separated string "<NBName>,<IP>" where *<NBName>* is the Windows/NetBIOS machine name for remote SMB service, and *<IP>* is the service's IP address. Example: MYSERVER,192.168.1.1

- The first component of the path in the URL points to the name of the shared folder. Subsequent path components will point to the directory/folder of the file.
- You can retrieve and upload files, but you cannot delete files/folders or create folders. In uploads, if the parent folders do not exist, an *urllib2.URLError* will be raised.

5.4.2 Example

The following code snippet illustrates file retrieval with Python 2.:

```
# -*- coding: utf-8 -*-
import urllib2
from smb.SMBHandler import SMBHandler

director = urllib2.build_opener(SMBHandler)
fh = director.open('smb://myuserID:mypassword@192.168.1.1/sharedfolder/rfc1001.txt')

# Process fh like a file-like object and then close it.
fh.close()

# For paths/files with unicode characters, simply pass in the URL as an unicode string
fh2 = director.open(u'smb://myuserID:mypassword@192.168.1.1/sharedfolder//.dat')

# Process fh2 like a file-like object and then close it.
fh2.close()
```

The following code snippet illustrates file upload with Python 2. You need to provide a file-like object for the *data* parameter in the *open()* method:

```
import urllib2
from smb.SMBHandler import SMBHandler

file_fh = open('local_file.dat', 'rb')

director = urllib2.build_opener(SMBHandler)
fh = director.open('smb://myuserID:mypassword@192.168.1.1/sharedfolder/upload_file.dat
↪', data = file_fh)

# Reading from fh will only return an empty string
fh.close()
```

The following code snippet illustrates file retrieval with Python 3.:

```
import urllib
from smb.SMBHandler import SMBHandler

director = urllib.request.build_opener(SMBHandler)
fh = director.open('smb://myuserID:mypassword@192.168.1.1/sharedfolder/rfc1001.txt')

# Process fh like a file-like object and then close it.
fh.close()
```

(continues on next page)

(continued from previous page)

```
# For paths/files with unicode characters, simply pass in the URL as an unicode string
fh2 = director.open(u'smb://myuserID:mypassword@192.168.1.1/sharedfolder//.dat')

# Process fh2 like a file-like object and then close it.
fh2.close()
```

The following code snippet illustrates file upload with Python 3. You need to provide a file-like object for the *data* parameter in the *open()* method:

```
import urllib
from smb.SMBHandler import SMBHandler

file_fh = open('local_file.dat', 'rb')

director = urllib.request.build_opener(SMBHandler)
fh = director.open('smb://myuserID:mypassword@192.168.1.1/sharedfolder/upload_file.dat
↪', data = file_fh)

# Reading from fh will only return an empty string
fh.close()
```

5.5 SMBProtocolFactory Class

For those who want to utilize pysmb in Twisted framework, pysmb has a *smb.SMBProtocol.SMBProtocol* implementation. In most cases, you do not need to touch or import the *SMBProtocol* directly. All the SMB functionalities are exposed in the *SMBProtocolFactory*.

In your project,

1. Create a new class and subclass *SMBProtocolFactory*.
2. Override the *SMBProtocolFactory.onAuthOK* and *SMBProtocolFactory.onAuthFailed* instance methods to provide your own post-authenthentication handling. Once *SMBProtocolFactory.onAuthOK* has been called by pymsb internals, your application is ready to communicate with the remote SMB/CIFS service through the *SMBProtocolFactory* public methods such as *SMBProtocolFactory.storeFile*, *SMBProtocolFactory.retrieveFile*, etc.
3. When you want to disconnect from the remote SMB/CIFS server, just call *SMBProtocolFactory.closeConnection* method.

All the *SMBProtocolFactory* public methods that provide file functionlities will return a *twisted.internet.defer.Deferred* instance. A *NotReadyError* exception is raised when the underlying SMB is not authenticated. If the underlying SMB connection has been terminated, a *NotConnectedError* exception is raised.

All the file operation methods in *SMBProtocolFactory* class accept a *timeout* parameter. This parameter specifies the time limit where pysmb will wait for the entire file operation (except *storeFile* and *retrieveFile* methods) to complete. If the file operation fails to complete within the timeout period, the returned *Deferred* instance's *errback* method will be called with a *SMBTimeout* exception.

If you are interested in learning the results of the operation or to know when the operation has completed, you should add a handling method to the returned *Deferred* instance via *Deferred.addCallback*. If the file operation fails, the *Deferred.errback* function will be called with an *OperationFailure*; on timeout, it will be called with a *SMBTimeout*.

5.5.1 Example

The following illustrates a simple file retrieving implementation.:

```
import tempfile
from twisted.internet import reactor
from smb.SMBProtocol import SMBProtocolFactory

class RetrieveFileFactory(SMBProtocolFactory):

    def __init__(self, *args, **kwargs):
        SMBProtocolFactory.__init__(self, *args, **kwargs)

    def fileRetrieved(self, write_result):
        file_obj, file_attributes, file_size = write_result

        # Retrieved file contents are inside file_obj
        # Do what you need with the file_obj and then close it
        # Note that the file obj is positioned at the end-of-file,
        # so you might need to perform a file_obj.seek() to if you
        # need to read from the beginning
        file_obj.close()

        self.transport.loseConnection()

    def onAuthOK(self):
        d = self.retrieveFile(self.service, self.path, tempfile.NamedTemporaryFile())
        d.addCallback(self.fileRetrieved)
        d.addErrback(self.d.errback)

    def onAuthFailed(self):
        print 'Auth failed'

# There will be some mechanism to capture userID, password, client_machine_name,␣
↪server_name and server_ip
# client_machine_name can be an arbitary ASCII string
# server_name should match the remote machine name, or else the connection will be␣
↪rejected
factory = RetrieveFileFactory(userID, password, client_machine_name, server_name, use_
↪ntlm_v2 = True)
factory.service = 'smbtest'
factory.path = '/rfc1001.txt'
reactor.connectTCP(server_ip, 139, factory)
```

5.5.2 SMB2 Support

Starting from pysmb 1.1.0, pysmb will utilize SMB2 protocol for communication if the remote SMB/CIFS service supports SMB2. Otherwise, it will fallback automatically back to using SMB1 protocol.

To disable SMB2 protocol in pysmb, set the *SUPPORT_SMB2* flag in the *smb_structs* module to *False* before creating the *SMBProtocolFactory* instance.:

```
from smb import smb_structs
smb_structs.SUPPORT_SMB2 = False
```

5.5.3 Caveats

- A new factory instance must be created for each SMB connection to the remote SMB/CIFS service. Avoid reusing the same factory instance for more than one SMB connection.
- The remote SMB/CIFS server usually imposes a limit of the number of concurrent file operations for each client. For example, to transfer a thousand files, you may need to setup a queue in your application and call *storeFile* or *retrieveFile* in batches.
- The *timeout* parameter in the file operation methods are not precise; it is accurate to within 1 second interval, i.e. with a timeout of 0.5 sec, pysmb might raise *SMBTimeout* exception after 1.5 sec.

class smb.SMBProtocol.**SMBProtocolFactory**(*username*, *password*, *my_name*, *remote_name*, *domain=''*, *use_ntlm_v2=True*, *sign_options=2*, *is_direct_tcp=False*)

__init__(*username*, *password*, *my_name*, *remote_name*, *domain=''*, *use_ntlm_v2=True*, *sign_options=2*, *is_direct_tcp=False*)

Create a new SMBProtocolFactory instance. You will pass this instance to *reactor.connectTCP()* which will then instantiate the TCP connection to the remote SMB/CIFS server. Note that the default TCP port for most SMB/CIFS servers using NetBIOS over TCP/IP is 139. Some newer server installations might also support Direct hosting of SMB over TCP/IP; for these servers, the default TCP port is 445.

username and *password* are the user credentials required to authenticate the underlying SMB connection with the remote server. File operations can only be proceeded after the connection has been authenticated successfully.

Parameters

- **my_name** (*string*) – The local NetBIOS machine name that will identify where this connection is originating from. You can freely choose a name as long as it contains a maximum of 15 alphanumeric characters and does not contain spaces and any of `\/:*?";|+`.
- **remote_name** (*string*) – The NetBIOS machine name of the remote server. On windows, you can find out the machine name by right-clicking on the "My Computer" and selecting "Properties". This parameter must be the same as what has been configured on the remote server, or else the connection will be rejected.
- **domain** (*string*) – The network domain. On windows, it is known as the workgroup. Usually, it is safe to leave this parameter as an empty string.
- **use_ntlm_v2** (*boolean*) – Indicates whether pysmb should be NTLMv1 or NTLMv2 authentication algorithm for authentication. The choice of NTLMv1 and NTLMv2 is configured on the remote server, and there is no mechanism to auto-detect which algorithm has been configured. Hence, we can only "guess" or try both algorithms. On Sambda, Windows Vista and Windows 7, NTLMv2 is enabled by default. On Windows XP, we can use NTLMv1 before NTLMv2.
- **sign_options** (*int*) – Determines whether SMB messages will be signed. Default is *SIGN_WHEN_REQUIRED*. If *SIGN_WHEN_REQUIRED* (value=2), SMB messages will only be signed when remote server requires signing. If *SIGN_WHEN_SUPPORTED* (value=1), SMB messages will be signed when remote server supports signing but not requires signing. If *SIGN_NEVER* (value=0), SMB messages will never be signed regardless of remote server's configurations; access errors will occur if the remote server requires signing.
- **is_direct_tcp** (*boolean*) – Controls whether the NetBIOS over TCP/IP (is_direct_tcp=False) or the newer Direct hosting of SMB over TCP/IP

(is_direct_tcp=True) will be used for the communication. The default parameter is False which will use NetBIOS over TCP/IP for wider compatibility (TCP port: 139).

closeConnection()
Disconnect from the remote SMB/CIFS server. The TCP connection will be closed at the earliest opportunity after this method returns.

Returns None

createDirectory(*service_name*, *path*)
Creates a new directory *path* on the *service_name*.

Parameters

- **service_name** (*string/unicode*) – Contains the name of the shared folder.
- **path** (*string/unicode*) – The path of the new folder (relative to) the shared folder. If the path contains non-English characters, an unicode string must be used to pass in the path.
- **timeout** (*integer/float*) – Number of seconds that pysmb will wait before raising *SMBTimeout* via the returned *Deferred* instance's *errback* method.

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with the *path* parameter.

deleteDirectory(*service_name*, *path*)
Delete the empty folder at *path* on *service_name*

Parameters

- **service_name** (*string/unicode*) – Contains the name of the shared folder.
- **path** (*string/unicode*) – The path of the to-be-deleted folder (relative to) the shared folder. If the path contains non-English characters, an unicode string must be used to pass in the path.
- **timeout** (*integer/float*) – Number of seconds that pysmb will wait before raising *SMBTimeout* via the returned *Deferred* instance's *errback* method.

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with the *path* parameter.

deleteFiles(*service_name*, *path_file_pattern*, *timeout=30*)
Delete one or more regular files. It supports the use of wildcards in file names, allowing for deletion of multiple files in a single request.

Parameters

- **service_name** (*string/unicode*) – Contains the name of the shared folder.
- **path_file_pattern** (*string/unicode*) – The pathname of the file(s) to be deleted, relative to the service_name. Wildcards may be used in th filename component of the path. If your path/filename contains non-English characters, you must pass in an unicode string.
- **timeout** (*integer/float*) – Number of seconds that pysmb will wait before raising *SMBTimeout* via the returned *Deferred* instance's *errback* method.

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with the *path_file_pattern* parameter.

echo(*data*, *timeout=10*)
Send an echo command containing *data* to the remote SMB/CIFS server. The remote SMB/CIFS will reply with the same *data*.

Parameters

- **data** (*bytes*) – Data to send to the remote server. Must be a bytes object.
- **timeout** (*integer/float*) – Number of seconds that pysmb will wait before raising *SMBTimeout* via the returned *Deferred* instance's *errback* method.

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with the *data* parameter.

getAttributes(*service_name*, *path*, *timeout=30*)
Retrieve information about the file at *path* on the *service_name*.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – Path of the file on the remote server. If the file cannot be opened for reading, an *OperationFailure* will be raised.

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with a *smb.base.SharedFile* instance containing the attributes of the file.

listPath(*service_name*, *path*, *search=65591*, *pattern='*'*, *timeout=30*)
Retrieve a directory listing of files/folders at *path*

For simplicity, pysmb defines a "normal" file as a file entry that is not read-only, not hidden, not system, not archive and not a directory. It ignores other attributes like compression, indexed, sparse, temporary and encryption.

Note that the default search parameter will query for all read-only (SMB_FILE_ATTRIBUTE_READONLY), hidden (SMB_FILE_ATTRIBUTE_HIDDEN), system (SMB_FILE_ATTRIBUTE_SYSTEM), archive (SMB_FILE_ATTRIBUTE_ARCHIVE), normal (SMB_FILE_ATTRIBUTE_INCL_NORMAL) files and directories (SMB_FILE_ATTRIBUTE_DIRECTORY). If you do not need to include "normal" files in the result, define your own search parameter without the SMB_FILE_ATTRIBUTE_INCL_NORMAL constant. SMB_FILE_ATTRIBUTE_NORMAL should be used by itself and not be used with other bit constants.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – path relative to the *service_name* where we are interested to learn about its files/sub-folders.
- **search** (*integer*) – integer value made up from a bitwise-OR of *SMB_FILE_ATTRIBUTE_xxx* bits (see smb_constants.py).
- **pattern** (*string/unicode*) – the filter to apply to the results before returning to the client.
- **timeout** (*integer/float*) – Number of seconds that pysmb will wait before raising *SMBTimeout* via the returned *Deferred* instance's *errback* method.

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with a list of *smb.base.SharedFile* instances.

listShares(*timeout=30*)
Retrieve a list of shared resources on remote server.

Parameters **timeout** (*integer/float*) – Number of seconds that pysmb will wait before raising *SMBTimeout* via the returned *Deferred* instance's *errback* method.

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with a list of *smb.base.SharedDevice* instances.

listSnapshots(*service_name*, *path*, *timeout=30*)

Retrieve a list of available snapshots (a.k.a. shadow copies) for *path*.

Note that snapshot features are only supported on Windows Vista Business, Enterprise and Ultimate, and on all Windows 7 editions.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – path relative to the *service_name* where we are interested in the list of available snapshots

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with a list of python *datetime.DateTime* instances in GMT/UTC time zone

onAuthFailed()

Override this method in your *SMBProtocolFactory* subclass to add in post-authentication handling. This method will be called when the server has replied that the SMB connection has been successfully authenticated.

If you want to retry authenticating from this method,

1. Disconnect the underlying SMB connection (call `self.instance.transport.loseConnection()`)
2. Create a new SMBProtocolFactory subclass instance with different user credientials or different NTLM algorithm flag.
3. Call `reactor.connectTCP` with the new instance to re-establish the SMB connection

onAuthOK()

Override this method in your *SMBProtocolFactory* subclass to add in post-authentication handling. This method will be called when the server has replied that the SMB connection has been successfully authenticated. File operations can proceed when this method has been called.

rename(*service_name*, *old_path*, *new_path*)

Rename a file or folder at *old_path* to *new_path* shared at *service_name*. Note that this method cannot be used to rename file/folder across different shared folders

old_path and *new_path* are string/unicode referring to the old and new path of the renamed resources (relative to) the shared folder. If the path contains non-English characters, an unicode string must be used to pass in the path.

Parameters

- **service_name** (*string/unicode*) – Contains the name of the shared folder.
- **timeout** (*integer/float*) – Number of seconds that pysmb will wait before raising *SMBTimeout* via the returned *Deferred* instance's *errback* method.

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with a 2-element tuple of (*old_path*, *new_path*).

retrieveFile(*service_name*, *path*, *file_obj*, *timeout=30*)

Retrieve the contents of the file at *path* on the *service_name* and write these contents to the provided *file_obj*.

Use *retrieveFileFromOffset()* method if you need to specify the offset to read from the remote *path* and/or the maximum number of bytes to write to the *file_obj*.

The meaning of the *timeout* parameter will be different from other file operation methods. As the downloaded file usually exceeeds the maximum size of each SMB/CIFS data message, it will be packetized into a series of request messages (each message will request about about 60kBytes). The *timeout* parameter is an integer/float value that specifies the timeout interval for these individual SMB/CIFS message to be transmitted and downloaded from the remote SMB/CIFS server.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – Path of the file on the remote server. If the file cannot be opened for reading, an *OperationFailure* will be called in the returned *Deferred* errback.
- **file_obj** – A file-like object that has a *write* method. Data will be written continuously to *file_obj* until EOF is received from the remote service.

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with a 3-element tuple of (*file_obj*, file attributes of the file on server, number of bytes written to *file_obj*). The file attributes is an integer value made up from a bitwise-OR of *SMB_FILE_ATTRIBUTE_xxx* bits (see smb_constants.py)

retrieveFileFromOffset(*service_name*, *path*, *file_obj*, *offset=0L*, *max_length=-1L*, *timeout=30*)

Retrieve the contents of the file at *path* on the *service_name* and write these contents to the provided *file_obj*.

The meaning of the *timeout* parameter will be different from other file operation methods. As the downloaded file usually exceeeds the maximum size of each SMB/CIFS data message, it will be packetized into a series of request messages (each message will request about about 60kBytes). The *timeout* parameter is an integer/float value that specifies the timeout interval for these individual SMB/CIFS message to be transmitted and downloaded from the remote SMB/CIFS server.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – Path of the file on the remote server. If the file cannot be opened for reading, an *OperationFailure* will be called in the returned *Deferred* errback.
- **file_obj** – A file-like object that has a *write* method. Data will be written continuously to *file_obj* until EOF is received from the remote service.
- **offset** (*integer/long*) – the offset in the remote *path* where the first byte will be read and written to *file_obj*. Must be either zero or a positive integer/long value.
- **max_length** (*integer/long*) – maximum number of bytes to read from the remote *path* and write to the *file_obj*. Specify a negative value to read from *offset* to the EOF. If zero, the *Deferred* callback is invoked immediately after the file is opened successfully for reading.

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with a 3-element tuple of (*file_obj*, file attributes of the file on server, number of bytes written to *file_obj*). The file attributes is an integer value made up from a bitwise-OR of *SMB_FILE_ATTRIBUTE_xxx* bits (see smb_constants.py)

storeFile(*service_name*, *path*, *file_obj*, *timeout=30*)

Store the contents of the *file_obj* at *path* on the *service_name*.

The meaning of the *timeout* parameter will be different from other file operation methods. As the uploaded file usually exceeeds the maximum size of each SMB/CIFS data message, it will be packetized into a

series of messages (usually about 60kBytes). The *timeout* parameter is an integer/float value that specifies the timeout interval for these individual SMB/CIFS message to be transmitted and acknowledged by the remote SMB/CIFS server.

Parameters

- **service_name** (*string/unicode*) – the name of the shared folder for the *path*
- **path** (*string/unicode*) – Path of the file on the remote server. If the file at *path* does not exist, it will be created. Otherwise, it will be overwritten. If the *path* refers to a folder or the file cannot be opened for writing, an *OperationFailure* will be called in the returned *Deferred* errback.
- **file_obj** – A file-like object that has a *read* method. Data will read continuously from *file_obj* until EOF.

Returns A *twisted.internet.defer.Deferred* instance. The callback function will be called with a 2-element tuple of (*file_obj*, number of bytes uploaded).

isReady
A convenient property to return True if the underlying SMB connection has connected to remote server, has successfully authenticated itself and is ready for file operations.

isUsingSMB2
A convenient property to return True if the underlying SMB connection is using SMB2 protocol.

5.6 SharedDevice Class

class smb.base.**SharedDevice**(*type*, *name*, *comments*)
Contains information about a single shared device on the remote server.

The following attributes are available:

- name : An unicode string containing the name of the shared device
- comments : An unicode string containing the user description of the shared device

isSpecial
Returns True if this shared device is a special share reserved for interprocess communication (IPC$) or remote administration of the server (ADMIN$). Can also refer to administrative shares such as C$, D$, E$, and so forth

isTemporary
Returns True if this is a temporary share that is not persisted for creation each time the file server initializes.

type

Returns one of the following integral constants.

- SharedDevice.DISK_TREE
- SharedDevice.PRINT_QUEUE
- SharedDevice.COMM_DEVICE
- SharedDevice.IPC

5.7 SharedFile Class

class smb.base.**SharedFile**(*create_time*, *last_access_time*, *last_write_time*, *last_attr_change_time*, *file_size*, *alloc_size*, *file_attributes*, *short_name*, *filename*, *file_id=None*)

Contain information about a file/folder entry that is shared on the shared device.

As an application developer, you should not need to instantiate a *SharedFile* instance directly in your application. These *SharedFile* instances are usually returned via a call to *listPath* method in *smb.SMBProtocol.SMBProtocolFactory*.

If you encounter *SharedFile* instance where its short_name attribute is empty but the filename attribute contains a short name which does not correspond to any files/folders on your remote shared device, it could be that the original filename on the file/folder entry on the shared device contains one of these prohibited characters: "/[]:+|<>=;?,* (see [MS-CIFS]: 2.2.1.1.1 for more details).

The following attributes are available:

- create_time : Float value in number of seconds since 1970-01-01 00:00:00 to the time of creation of this file resource on the remote server
- last_access_time : Float value in number of seconds since 1970-01-01 00:00:00 to the time of last access of this file resource on the remote server
- last_write_time : Float value in number of seconds since 1970-01-01 00:00:00 to the time of last modification of this file resource on the remote server
- last_attr_change_time : Float value in number of seconds since 1970-01-01 00:00:00 to the time of last attribute change of this file resource on the remote server
- file_size : File size in number of bytes
- alloc_size : Total number of bytes allocated to store this file
- file_attributes : A SMB_EXT_FILE_ATTR integer value. See [MS-CIFS]: 2.2.1.2.3. You can perform bit-wise tests to determine the status of the file using the ATTR_xxx constants in smb_constants.py.
- short_name : Unicode string containing the short name of this file (usually in 8.3 notation)
- filename : Unicode string containing the long filename of this file. Each OS has a limit to the length of this file name. On Windows, it is 256 characters.
- file_id : Long value representing the file reference number for the file. If the remote system does not support this field, this field will be None or 0. See [MS-FSCC]: 2.4.17

isDirectory
A convenient property to return True if this file resource is a directory on the remote server

isNormal
A convenient property to return True if this is a normal file.

Note that pysmb defines a normal file as a file entry that is not read-only, not hidden, not system, not archive and not a directory. It ignores other attributes like compression, indexed, sparse, temporary and encryption.

isReadOnly
A convenient property to return True if this file resource is read-only on the remote server

5.8 SMB Exceptions

class smb.base.**SMBTimeout**
Raised when a timeout has occurred while waiting for a response or for a SMB/CIFS operation to complete.

class smb.base.**NotReadyError**
Raised when SMB connection is not ready (i.e. not authenticated or authentication failed)

class smb.base.**NotConnectedError**
Raised when underlying SMB connection has been disconnected or not connected yet

class smb.smb_structs.**UnsupportedFeature**
Raised when an supported feature is present/required in the protocol but is not currently supported by pysmb

class smb.smb_structs.**ProtocolError**(*message*, *data_buf=None*, *smb_message=None*)

class smb.smb_structs.**OperationFailure**(*message*, *smb_messages*)

5.9 Security Descriptors

This module implements security descriptors, and associated data structures, as specified in [MS-DTYP].

class smb.security_descriptors.**SID**(*revision*, *identifier_authority*, *subauthorities*)
A Windows security identifier. Represents a single principal, such a user or a group, as a sequence of numbers consisting of the revision, identifier authority, and a variable-length list of subauthorities.

See [MS-DTYP]: 2.4.2

class smb.security_descriptors.**ACE**(*type_*, *flags*, *mask*, *sid*, *additional_data*)
Represents a single access control entry.

See [MS-DTYP]: 2.4.4

isInheritOnly
Convenience property which indicates if this ACE is inherit only, meaning that it doesn't apply to the object itself.

class smb.security_descriptors.**ACL**(*revision*, *aces*)
Access control list, encapsulating a sequence of access control entries.

See [MS-DTYP]: 2.4.5

class smb.security_descriptors.**SecurityDescriptor**(*flags*, *owner*, *group*, *dacl*, *sacl*)
Represents a security descriptor.

See [MS-DTYP]: 2.4.6

5.10 Extending pysmb For Other Frameworks

This page briefly describes the steps involved in extending pysmb for other frameworks.

In general, you need to take care of the SMB TCP connection setup, i.e. finding the IP address of the remote server and connect to the SMB/CIFS service. Then you need to read/write synchronously or asynchronously from and to the SMB socket. And you need to handle post-authentication callback methods, and from these methods, initiate file operations with the remote SMB/CIFS server.

Now the above steps in more technical details:

1. Create a new class which subclasses the *smb.base.SMB* class. Most often, the connection setup will be part of the *__init__* method.
2. Override the *write(self, data)* method to provide an implementation which will write *data* to the socket.
3. Write your own loop handling method to read data from the socket. Once data have been read, call *feedData* method with the parameter. The *feedData* method has its own internal buffer, so it can accept incomplete NetBIOS session packet data.
4. Override

- *onAuthOK* method to include your own operations to perform when authentication is successful. You can initiate file operations in this method.
- *onAuthFailed* method to include your own processing on what to do when authentication fails. You can report this as an error, or to try a different NTLM authentication algorithm (*use_ntlm_v2* parameter in the constructor).
- *onNMBSessionFailed* method to include your own processing on what to do when pysmb fails to setup the NetBIOS session with the remote server. Usually, this is due to a wrong *remote_name* parameter in the constructor.

5.11 Upgrading from older pysmb versions

This page documents the improvements and changes to the API that could be incompatible with previous releases.

5.11.1 pysmb 1.2.0

- Add new *delete_matching_folders* parameter to *deleteFiles()* method in SMBProtocolFactory and SMBConnection class to support deletion of sub-folders. If you are passing timeout parameter to the *deleteFiles()* method in your application, please switch to using named parameter for timeout.

5.11.2 pysmb 1.1.28

- SharedFile instances returned from the *listPath()* method now has a new property *file_id* attribute which represents the file reference number given by the remote SMB server.

5.11.3 pysmb 1.1.26

- SMBConnection class can now be used as a context manager

5.11.4 pysmb 1.1.25

- SharedFile class has a new property *isNormal* which will be True if the file is a 'normal' file. pysmb defines a 'normal' file as a file entry that is not read-only, not hidden, not system, not archive and not a directory; it ignores other attributes like compression, indexed, sparse, temporary and encryption.
- *listPath()* method in SMBProtocolFactory and SMBConnection class will now include 'normal' files by default if you do not specify the *search* parameter.

5.11.5 pysmb 1.1.20

- A new method *getSecurity()* was added to SMBConnection and SMBProtocolFactory class.

5.11.6 pysmb 1.1.15

- Add new *truncate* parameter to *storeFileFromOffset()* in SMBProtocolFactory and SMBConnection class to support truncation of the file before writing. If you are passing timeout parameter to the *storeFileFromOffset()* method in your application, please switch to using named parameter for timeout.

5.11.7 pysmb 1.1.11

- A new method *storeFileFromOffset()* was added to SMBConnection and SMBProtocolFactory class.

5.11.8 pysmb 1.1.10

- A new method *getAttributes()* was added to SMBConnection and SMBProtocolFactory class
- SharedFile class has a new property *isReadOnly* to indicate the file is read-only on the remote filesystem.

5.11.9 pysmb 1.1.2

- *queryIPForName()* method in nmb.NetBIOS and nmb.NBNSProtocol class will now return only the server machine name and ignore workgroup names.

5.11.10 pysmb 1.0.3

- Two new methods were added to NBNSProtocol class: *queryIPForName()* and *NetBIOS.queryIPForName()* to support querying for a machine's NetBIOS name at the given IP address.
- A new method *retrieveFileFromOffset()* was added to SMBProtocolFactory and SMBConnection to support finer control of file retrieval operation.

5.11.11 pysmb 1.0.0

pysmb was completely rewritten in version 1.0.0. If you are upgrading from pysmb 0.x, you most likely have to rewrite your application for the new 1.x API.

- genindex
- search

Python Module Index

S

Index

Symbols

A

C

D

E

G

I

L

N

O

P

Q

R

S

T

U