NetSieben Technologies INC.
SSH Library API Documentation

Author: Andrew Useckas

USA
370 Interlocken Boulevard
Level 4, Suite 125
Broomfield, CO 80021
Ph.: (+1) 303 543 0300
Fax: (+1) 303 484 3644
www.netsieben.com

Australia
Level 57, MLC Centre 19-25
Martin Place Sydney, NSW 2000
Ph.: (+61) 2 9238 6130
Fax: (+ 61) 2 9238 6877
www.netsieben.com.au

Table of Contents

1. Overview

The Secure Shell (SSH) protocol is used by many as a secure way of managing servers, firewalls and other network appliances.

Nowadays many custom built applications require Secure Shell client functionality. Instead of spending countless hours building SSH functionality into an application one can now use the NetSieben Technologies Inc. (NetSieben) SSH library to securely communicate with a variety of SSH server implementations.

The library was developed by NetSieben after researching the limited number of options available. A lot of available solutions were either wrappers to the well-known, but license restricted openssh application or libraries with very limited functionality, and at best, still in alpha or beta stages of development.

At this time, version 1 of the SSH protocol is outdated with well known security flaws inherently in its design. Therefore, NetSieben SSH library supports only version 2 of the SSH protocol.

Currently NetSieben's library supports only SSH client functionality.

1.1 Features

Feature	Supported Algorithms
Key exchange	Diffie Hellman Group 1, SHA1, Diffie Hellman Group 14, SHA1
Signatures	Ssh-dss, ssh-rsa
User authentication	public key, password
Authentication keys	DSA (512bit to 1024bit), RSA
Encryption	aes256-cbc, twofish-cbc, twofish256-cbc, blowfish-cbc, 3des-cbc, aes128-cbc, cast128-cbc
HMAC	hmac-md5, hmac-sha1, none
Compression	not supported
Interoperability	SSH Library should work with most SSH2 server implementations. Tested with openssh on Linux. Solaris, FreeBSD and NetBSD. Also tested with the Juniper Netscreen ssh server implementation.
Subsystems	Secure FTP.

Portability: The NetSieben SSH Library is based on the Botan crypto library, making it highly portable. It has been tested on Linux x86, Linux x86_64, Linux PPC, Linux SPARC, Solaris, Windows 2000 and XP.

1.2 Dependencies

The NetSiben SSH Library requires version 1.4.9 of the Botan crypto library or higher. The recommended version is 1.4.12.

The Botan library can be found here:

http://botan.randombit.net

1.3 Installation

1.3.1 All *nix Platforms

The public version of NetSieben's SSH library is distributed in source code form. The current version uses autoconf to handle the generation of Makefile(s) for your system. Generally you should be able to install NetSieben SSH Library by executing the following commands after decompressing the tarball:

```
./configure
make
make install
```

For more information please refer to INSTALL file included in the software package.

Binary distributions of the NetSieben SSH Library may be provided. However, these packages are not officially supported by NetSieben Technologies, Inc. For installation Instructions on these platforms, please refer to the Operating System or distribution documentation.

1.3.2 Binary Packages on Microsoft® Windows™

Starting with version 1.1.6, NetSieben's SSH Library is available as an InstallShield package for installation. In order to read this documentation, Adobe Acrobat is required. To install the package, extract the archive and simply run the setup.exe file and follow the on-screen instructions.

2. Usage

2.1 Setting up the Environment

On Unix Platforms, simply add the include path based on the configured –prefix path. The include directory is in ${PREFIX}/include (-I/path/to/include/files) and the libraries are in ${PREFIX}/lib (-L /path/to/lib/files), unless the package was configured with custom includedir and libdir values, in which case the files will be in those respective locations. Remember that when running an application linked to the NetSieben SSH Library, the library directory needs to be in LD_LIBRARY_PATH (DYLD_LIBRARY_PATH on Mac® OSX), or in the standard Library Path.

On Windows Platforms, after your project has been set up, under Project Settings, add an Include Search Path of $(InstallationDir)\include (typically C:\Program Files\NetSieben Technologies\ne7ssh\include). Then for linking, under Additional Libraries add ne7ssh.lib, while providing the $(InstallationDir)\lib (C:\Program Files\NetSieben Technologies\ne7ssh\lib) for Additional Library Locations. The SFTP functionality provided by the library uses ANSI buffer file objects for managing local files. Accordingly, for this functionality to work, your application must use the (non-debug) DLL version of the C Runtime Libraries.

NetSieben's SSH library is not provided as a .NET class library, however it can be utilized from within the .NET platform by compiling a C++/CLI project in mixed mode. Managed mode classes can't use a primitive reference to any of the NetSieben classes, but can use an unmanaged pointer. Once such a class has been written in C++/CLI utilizing the NetSieben SSH library on the back end, other .NET projects can use that resource to extend their functionality.

2.2 Initializing the class

Before using the library's functionality, it needs to first be initialized. The initialization process will allocate the required memory resources and prepare cryptographic methods for usage. It is recommended to do this, when first starting the application.

Only one instance of the ne7ssh class can be used at a time. A single instance of the ne7ssh class is able to handle multiple connections to multiple servers. It is recommended to initialize the class when starting, using the same class for all of your SSH connections, subsequently destroying it on exit.

The class should be initialized with the following command:

```
ne7ssh *ssh = new ne7ssh ();
```

The constructor of the class requires no arguments. The application will exit when trying to initialize the class more than once within the same application.

2.3 Setting the options

Before opening Secure connections, custom options can be configured for use in all future connections. Currently only the desired cipher and integrity checking algorithms can be configured. The following method can be used to configure the options:

```
setOptions (const char *prefCipher, const char *prefHmac)

prefCipher              your preferred cipher algorithm string representation.
                        Supported options are: aes256-cbc, twofish-cbc, twofish256-cbc,
                        blowfish-cbc, 3des-cbc, aes128-cbc, cast128-cbc.

prefHmac                the preferred integrity checking algorithm string.
                        Supported optionss are: hmac-md5, hmac-sha1 and none.
```

This step is optional and if skipped the SSH library will use the default settings. If desired algorithms are not supported by the server, the next one from the list of supported algorithms will be used.

2.4 Connecting to a remote server

After the options are set, connections to remote servers can be initiated. NetSieben's library supports Password and Public Key authentication methods.

2.4.1 Key based authentication

NetSieben's SSH library supports key based authentication. For this to work one needs to generate a key pair either by using the generateKeyPair method from ne7ssh class or by using the ssh-keygen program included in openssh distributions. The server also must support key based authentication and the newly generated public key needs to be added to servers authorized keys. The process to do so may differ, depending on the SSH server vendor and implementation.

Currently RSA and DSA keys are supported. The NetSieben library keys are compatible with unencrypted OpenSSH keys.

No password should be specified when creating a key pair with ssh-keygen.

To establish connection to a remote server using a private key use the following method:

```
int ssh->connectWithKey (const char* host, uint32 port, const char* username, const char*
privKeyFile, bool shell, const int timeout);

host                    Hostname or IP of the server.
port                    Port the SSH server binds to.
                        usually be 22 (standard SSH port).
username                Username to be used in authentication.
PrivKeyFile             Full path to a PEM encoded private key file.
shell                   Should the shell be spawned on the remote end.  True by defaut.
                        Should be set to false if using sendCmd() method after
                        authentication or if planning to launch SFTP subsystem.
timeout                 Timeout for the connect procedure, in seconds.
```

If the connection succeeds, the method will return the newly created channel ID. This ID should be used in all further communications via newly created connection.

If the connection failed for any reason, -1 will be returned by the method.

Generating a key pair

NetSieben's SSH library can be used to generate key pairs. Currently RSA and DSA key algorithms are supported. DSA keys can only be between 512bits and 1024bits (a restriction inherited from the Botan library). The newly generated keys are OpenSSH compatible and public keys can be pasted straight into an authorized_keys file. The following method generates a key pair:

```
bool generateKeyPair (const char* type, const char* fqdn, const char* privKeyFileName,
const char* pubKeyFileName, uint16 keySize);

type                    String specifying key type. Currently "dsa" and "rsa" are
supported.
fqdn                    User id. Such as an Email address.
privKeyFileName         Full path to a file where generated private key will be written.
pubKeyFileName          Full path to a file where generated public key will be written.
keySize                 Desired key size in bits. If not specified will default to 2048.
                        The value has to be changed if generating a dsa keypair.
```

2.4.2 Password based authentication

NetSieben's SSH library supports password based authentication. Password authentication should be enabled in the server configuration. If using this method, make sure that it's enabled, as many distributions of OpenSSH disable this method by default. The following method will establish a connection using the password authentication:

```
int connectWithPassword (const char *host, uint32 port, const char *username, const char
*password, bool shell, const int timeout)

host                       Hostname or IP of the server.
port                       Port SSH server binds to, usually 22 (the standard SSH port).
username                   Username used in authentication.
password                   Password used in authentication.
shell                      Should the shell be spawned on the remote end.  True by defaut.
                           Should be set to false if using sendCmd() method after
                           authentication or if planning to launch SFTP subsystem.
timeout                    Timeout for the connect procedure, in seconds.
```

If the connection succeeds, the method will return the newly created channel ID. This ID should be used in all further communications via newly established connection.

If the connection fails for any reason, -1 will be returned by the method.

2.5 Communications

NetSieben's SSH library supports two command modes. The first is interactive mode used for sending text commands and waiting for text results. This works much like perl “Expect” module.

The second mode is a single command mode, where library sends a single command to the remote end, waits for results and disconnects. This method can be used to handle binary results.

2.5.1 Interactive mode

Send command

Before sending a command the waitFor method should be used to wait for shell prompt, ensuring the remote end is ready for interactive commands.

Sending a command to the remote end can be accomplished using this method:

```
bool send (const char *data, int channel )

data                       Command to send. This should be a string terminated with an EOL.
                           Most terminals will not process a command without end-line.
                           (character(s) (\n in Unix) to it).
channel                    Channel ID.
```

This method will return true if the write to send buffer succeeded. Otherwise, false will be returned.

Note: Keep in mind that each use of send() method will flush the receive buffer.

Wait for results

If the connection is established and a shell launched at the remote end one needs to pause until a specific string is received. This could be used to wait for shell prompt after connecting or after a command execution.

The SSH protocol is designed to send data in packets. If results are read from the buffer right after sending a command, there is no guarantee that the entire result has been received by the library. Thus it is necessary to wait for a particular string to appear in the receive buffer. This ensures that all the data has been received.

If the specified string is not received, to avoid blocking condition, a timeout value should be specified. If the desired result is not received in specified timeframe, the function will return false.

The following method is used for interactive communications:

```
bool waitFor (int channel, const char *str, uint32 timeout = 0)

channel                 Channel ID  received from the connection methods, specifying
                        the interactive channel.
str                     String containing text to wait for.
timeout                 Timeout in seconds.  Timeout depends on the speed of your
                        connection to the remote side. 2 seconds should be enough for
                        most connections.  If 0 is specified, the method will block
                        until the requested string arrives in the receive buffer.
```

If the expected string is received the method will return true. If the timeout has been reached, the method will return false.

Fetching the result

After receiving the expected string or reaching the timeout threshold, the received results can be accessed by using the following method:

```
const char *read (int channel)

channel                 Channel ID  received from the connection methods, specifying
                        the interactive channel.
```

This method will return pointer to the receive buffer. The memory for the buffer doesn't need to be allocated nor freed by a programmer, its storage is handled entirely by the ne7ssh class.

This method should always be executed after the waitFor() method.

2.5.2 Single command

A single command can be used when only one command needs to be executed before disconnecting. Or when expected result is binary.

Sending command

This method will not work if the shell has been spawned at the remote end. One needs to make sure that "shell" parameter of authentication method is set to "false" before using this method:

```
bool sendCmd (const char* cmd, int channel, int timeout);

cmd                     Command to send. This should be a string terminated with an EOL.
                        Most terminals will not process a command without end-line.
                        (character(s) (\n in Unix) to it).
channel                 Channel ID.
Timeout                 How long to wait for completion.  This value will depend on the
                        speed of connection and size of results.  Recommended value is 30
                        seconds.
```

This method will true if command executed successfully. If an error occurred during execution false is returned. The last error can be obtained by using errors()->pop() method.

Result buffer size

After sending a command followed by a successful execution, the result is received by the SSH library and placed into the receive buffer. If the data received is binary, one needs to know the size of the buffer before reading it. This can be obtained by the following method:

```
int getReceivedSize (int channel)

channel                 Channel ID.
```

This method will return the size of the receive buffer, or zero if the buffer is empty.

Fetching the result

If the expected result is a string one can use the above mentioned read() method to fetch the data. However if the result is binary the following method should be used:

```
void *readBinary (int channel)
```

```
channel                 Channel ID  received from the connection methods.
```

This method will return a pointer to the receive buffer. The memory for the buffer doesn't need to be allocated nor freed by a programmer, its storage is handled entirely by the ne7ssh class.

Having a pointer to the receive buffer and the size of the buffer, one can easily fetch the binary data stored.

2.6 Closing the connection

When the desired communications are completed the connection needs to be closed using the following method:

```
bool close (int channel)
```

```
channel                 Channel ID  received from the connection methods.
```

This method returns true if the sending of the close command succeeds. False is returned if the channel has already been closed.

It is highly recommended to use a shell command to close the interactive connection instead of this method.

2.7 Error handling

Starting with version 1.1.5 the SSH library integrates contextual error reporting. Errors are bound to the channel they occur in. The core messages are bound to the Core context. All errors are stored in a static instance of Ne7sshError class and can be access via ne7ssh::errors() method.

2.7.1 Core context

The errors that are not bound to a channel context are considered to be core errors and can be retrieved using the following command:

```
const char* Ne7sshError::pop()
```

The command returns a pointer to the last error message within Core context, removing it from the stack. Continued execution of this command will result in returning of all Core error messages and at the same time removing them from the stack. If there are no error message in the Core context zero is returned.

2.7.2 Channel context

The errors that are bound to a channel context can be retrieved using the following command:

```
const char* Ne7sshError::pop(in channel)

channel                 Channel ID.
```

This command returns a pointer to the last error message within particular channel context, also removing it from the stack. Continued execution of this command will result in returning of all particular channel error messages at the same time removing them from the stack. If there are no error message in a channel context zero is returned.

2.8 Secure FTP support

Secure FTP (SFTP) client is supported by the NetSieben library starting with version 1.2.0.

2.8.1 Initializing the subsystem

In order to utilize the Secure FTP functionality sftp specific class instances need to be icreated and sftp subsystem started on the server side. In order to accomplish this task the following variable needs to be defined:

```
Ne7SftpSubsystem _sftp
```

The following method is used to initialize the subsystem. The method needs channel ID, so has to be executed after one of the connect methods. Make sure remote shell is not spawned.

```
bool initSftp (class Ne7SftpSubsystem& _sftp, int channel)

_sftp                   Reference to SFTP subsystem to be initialized.
channel                 Channel ID returned by one of the connect methods.
```

This command returns "true" if the new subsystem was successfully initialized. And "false" is returned if any error occurs.

2.8.2 Setting a timeout for SFTP communications.

The SFTP subsystem has a default timeout for all of the communications, set to 30 seconds. This should work well under most circumstances, however sometimes it maybe desirable to modify this value. The following method accomplishes the task:

```
bool setTimeout (uint32 _timeout)

timeout                  Timeout in seconds.
```

This command returns "true" upon successful setting of the timeout. Returns "false" on any error.

2.8.3 Downloading a file

The following method is used to download a file from a remote server. It functions like sftp "get" command:

```
bool get (const char* remoteFile, FILE* localFile)

remoteFile               Full or relative path to the file on the remote side.
LocalFile                Pointer to FILE structure. If the file being retrieved is binary,
                         use "w+" attributes in fopen function.
```

This command returns "true" if the file was successfully downloaded. Returns "false" on any error.

2.8.4 Uploading a file

The following method is used for uploading a file to a remote server. It functions like SFTP "put" command:

```
bool put (FILE* localFile, const char* remoteFile)

LocalFile                Pointer to FILE structure. If the file being retrieved is binary,
                         use "r+" attributes in fopen function.
remoteFile               Full or relative path to the file on the remote side.
```

This command returns "true" if the file was successfully uploaded. Returns "false" if any error is encountered.

2.8.5 Removing a file

The following method is used to remove a file. It functions like sftp "rm" command:

```
bool rm (const char* remoteFile)

remoteFile              Full or relative path to the file on the remote side.
```

This command returns "true" if the file was successfully removed. Returns "false" if any error is encountered.

2.8.6 Renaming or moving a file

The following method is used to rename or move a file. It functions like sftp "rename" command:

```
bool mv (const char* oldFile, const char* newFile)

oldFile                 Full or relative path to the file being moved or renamed.
newFIle                 Full or relative path to the new location/name of the file.
```

This command returns "true" if the file was successfully renamed or moved. Returns "false" if any error is encountered..

2.8.7 Changing the current context

The following method is used to change the current context. It functions like sftp "cd" command:

```
bool cd (const char* remoteDir)

remoteDir               Full or relative path to a new work path.
```

This command returns "true" if the context was successfully changed. Returns "false" if any error is encountered.

2.8.8 Creating a new directory

The following method is used to create a new directory. It functions like sftp “mkdir” command:

```
bool mkdir (const char* remoteDir)

remoteDir              Full or relative path to a new directory.
```

This command returns “true” if the new directory was successfully created. Returns “false” if any error is encountered.

2.8.9 Removing a directory

The following method if used to remove a directory. It functions like sftp “rmdir” command:

```
bool rmdir (const char* remoteDir)

remoteDir              Full or relative path to a directory.
```

This command returns “true” if the directory was successfully removed. Returns “false” if any error is encountered.

2.8.10 Getting a directory listing

The following method is used to get a directory listing. It functions like sftp “ls” command:

```
const char* ls (const char* remoteDir, bool longNames=false)

remoteDir              Full or relative path to a directory.
LongNames              If set to "true" the returned string in addition to file strings
                       will contain attributes for each file.
```

This command returns pointer to a buffer containing directory listing. Returns NULL if any error is encountered.

2.8.11 Changing permissions

The following method is used to change the permissions of a file or directory. It functions like sftp "chmod" command:

```
bool chmod (const char* remoteFile, const char* mode)

remotFile               Full or relative path to a file.
mode                    Mode string.  It can be either a numerical mode expression such
                        as "755" or an expression showing the modifications to be made,
                        such as "ug+w".  Mode string is the same as used by *nix chmod
                        command.
```

This command returns "true" if the new permissions are successfully applied. Returns "false" if any error is encountered..

2.8.12 Changing ownership

The following method is used to change the ownership of a file or directory. It functions like sftp "chown" command:

```
bool chown (const char* remoteFile, uint32_t uid, uint32_t gid)

remoteFile              Full or relative path to a file.
uid                     Numerical user ID of the new owner.
gid                     Numerical group ID of the new owner.
```

This command returns "true" if the new ownership is successfully applied. Returns "false" if any error is encountered..

3. Commercial License

For details about commercial license please fill out the form located at:

https://netsieben.com/sshlib_info.phtml

4. Notes