

libssh
0.5.2

Generated by Doxygen 1.7.3

Tue Feb 28 2012 08:19:43

Contents

1	Main Page	1
1.1	Linking	1
1.2	Tutorial	1
1.3	Features	1
1.4	Copyright Policy	2
1.5	Internet standard	2
1.5.1	Secure Shell (SSH)	2
1.5.2	Secure Shell File Transfer Protocol (SFTP)	3
1.5.3	Secure Shell Extensions	4
2	The Tutorial	5
2.1	Introduction	5
2.2	Chapter 1: A typical SSH session	6
2.2.1	A typical SSH session	6
2.2.1.1	Creating the session and setting options	7
2.2.1.2	Connecting to the server	8
2.2.1.3	Authenticating the server	8
2.2.1.4	Authenticating the user	10
2.2.1.5	Doing something	12
2.2.1.6	Handling the errors	13
2.3	Chapter 2: A deeper insight on authentication	14
2.3.1	A deeper insight on authentication	14
2.3.1.1	Authenticating with public keys	14
2.3.1.2	Authenticating with a password	15
2.3.1.3	The keyboard-interactive authentication method	16
2.3.1.4	Authenticating with "none" method	18
2.3.1.5	Getting the list of supported authentications	19
2.3.1.6	Getting the banner	20
2.4	Chapter 3: Opening a remote shell	20
2.4.1	Opening a remote shell	20
2.4.1.1	Opening and closing a channel	20
2.4.1.2	Interactive and non-interactive sessions	21
2.4.1.3	Displaying the data sent by the remote computer	22
2.4.1.4	Sending user input to the remote computer	22
2.4.1.5	A more elaborate way to get the remote data	24
2.4.1.6	Using graphical applications on the remote side	25
2.5	Chapter 4: Passing a remote command	26
2.5.1	Passing a remote command	26
2.5.1.1	Executing a remote command	26

2.6	Chapter 5: The SFTP subsystem	27
2.6.1	The SFTP subsystem	27
2.6.1.1	Opening and closing a SFTP session	28
2.6.1.2	Analyzing SFTP errors	29
2.6.1.3	Creating a directory	30
2.6.1.4	Copying a file to the remote computer	30
2.6.1.5	Reading a file from the remote computer	31
2.6.1.6	Listing the contents of a directory	33
2.7	Chapter 6: The SCP subsystem	35
2.7.1	The SCP subsystem	35
2.7.1.1	Opening and closing a SCP session	35
2.7.1.2	Creating files and directories	36
2.7.1.3	Copying full directory trees to the remote server	37
2.7.1.4	Reading files and directories	38
2.7.1.5	Receiving full directory trees from the remote server	39
2.8	Chapter 7: Forwarding connections (tunnel)	39
2.8.1	Forwarding connections	39
2.8.1.1	Direct port forwarding	39
2.8.1.2	Reverse port forwarding	40
2.8.1.3	X11 tunnels	40
2.8.1.4	Doing direct port forwarding with libssh	40
2.8.1.5	Doing reverse port forwarding with libssh	41
2.9	Chapter 8: Threads with libssh	43
2.9.1	How to use libssh with threads	43
2.9.1.1	Initialization of threads	43
2.9.1.2	Using libpthread with libssh	43
2.9.1.3	Using another threading library	44
2.10	To be done	44
2.10.1	Writing a libssh-based server	44
2.10.2	The libssh C++ wrapper	44
3	The Linking HowTo	45
3.1	Dynamic Linking	45
3.2	Static Linking	45
4	Deprecated List	47
5	Bug List	49
6	Module Index	51
6.1	Modules	51
7	Data Structure Index	53
7.1	Data Structures	53
8	File Index	55
8.1	File List	55
9	Module Documentation	57
9.1	The libssh callbacks	57
9.1.1	Detailed Description	59

9.1.2	Define Documentation	59
9.1.2.1	ssh_callbacks_init	59
9.1.2.2	SSH_PACKET_CALLBACK	59
9.1.2.3	SSH_PACKET_USED	59
9.1.3	Typedef Documentation	60
9.1.3.1	ssh_auth_callback	60
9.1.3.2	ssh_channel_close_callback	60
9.1.3.3	ssh_channel_data_callback	60
9.1.3.4	ssh_channel_eof_callback	61
9.1.3.5	ssh_channel_exit_signal_callback	61
9.1.3.6	ssh_channel_exit_status_callback	61
9.1.3.7	ssh_channel_signal_callback	61
9.1.3.8	ssh_global_request_callback	62
9.1.3.9	ssh_log_callback	62
9.1.3.10	ssh_packet_callback	62
9.1.3.11	ssh_status_callback	63
9.1.4	Function Documentation	63
9.1.4.1	ssh_set_callbacks	63
9.1.4.2	ssh_set_channel_callbacks	63
9.2	The libssh C++ wrapper	64
9.2.1	Detailed Description	64
9.3	The libssh server API	65
9.3.1	Typedef Documentation	66
9.3.1.1	ssh_bind_incoming_connection_callback	66
9.3.2	Function Documentation	66
9.3.2.1	ssh_bind_accept	66
9.3.2.2	ssh_bind_fd_toaccept	67
9.3.2.3	ssh_bind_free	67
9.3.2.4	ssh_bind_get_fd	67
9.3.2.5	ssh_bind_listen	67
9.3.2.6	ssh_bind_new	68
9.3.2.7	ssh_bind_options_set	68
9.3.2.8	ssh_bind_set_blocking	69
9.3.2.9	ssh_bind_set_callbacks	69
9.3.2.10	ssh_bind_set_fd	70
9.3.2.11	ssh_handle_key_exchange	70
9.3.2.12	ssh_set_message_callback	70
9.4	The libssh SFTP API	71
9.4.1	Function Documentation	75
9.4.1.1	sftp_async_read	75
9.4.1.2	sftp_async_read_begin	75
9.4.1.3	sftp_attributes_free	76
9.4.1.4	sftp_canonicalize_path	76
9.4.1.5	sftp_chmod	76
9.4.1.6	sftp_chown	77
9.4.1.7	sftp_close	77
9.4.1.8	sftp_closedir	77
9.4.1.9	sftp_dir_eof	77
9.4.1.10	sftp_extension_supported	78
9.4.1.11	sftp_extensions_get_count	78

9.4.1.12	sftp_extensions_get_data	78
9.4.1.13	sftp_extensions_get_name	79
9.4.1.14	sftp_free	79
9.4.1.15	sftp_fstat	79
9.4.1.16	sftp_fstatvfs	79
9.4.1.17	sftp_get_error	80
9.4.1.18	sftp_init	80
9.4.1.19	sftp_lstat	80
9.4.1.20	sftp_mkdir	81
9.4.1.21	sftp_new	81
9.4.1.22	sftp_open	81
9.4.1.23	sftp_opendir	82
9.4.1.24	sftp_read	82
9.4.1.25	sftp_readdir	82
9.4.1.26	sftp_readlink	83
9.4.1.27	sftp_rename	83
9.4.1.28	sftp_rewind	83
9.4.1.29	sftp_rmdir	83
9.4.1.30	sftp_seek	84
9.4.1.31	sftp_seek64	84
9.4.1.32	sftp_server_version	84
9.4.1.33	sftp_setstat	84
9.4.1.34	sftp_stat	85
9.4.1.35	sftp_statvfs	85
9.4.1.36	sftp_statvfs_free	85
9.4.1.37	sftp_symlink	85
9.4.1.38	sftp_tell	86
9.4.1.39	sftp_tell64	86
9.4.1.40	sftp_unlink	86
9.4.1.41	sftp_utimes	86
9.4.1.42	sftp_write	87
9.5	The SSH authentication functions.	87
9.5.1	Detailed Description	89
9.5.2	Function Documentation	89
9.5.2.1	privatekey_free	89
9.5.2.2	privatekey_from_file	89
9.5.2.3	publickey_from_file	90
9.5.2.4	publickey_from_privatekey	90
9.5.2.5	publickey_to_string	91
9.5.2.6	ssh_auth_list	91
9.5.2.7	ssh_privatekey_type	92
9.5.2.8	ssh_publickey_to_file	92
9.5.2.9	ssh_try_publickey_from_file	92
9.5.2.10	ssh_userauth_agent_pubkey	93
9.5.2.11	ssh_userauth_autopubkey	93
9.5.2.12	ssh_userauth_kbdint	94
9.5.2.13	ssh_userauth_kbdint_getinstruction	95
9.5.2.14	ssh_userauth_kbdint_getname	95
9.5.2.15	ssh_userauth_kbdint_getnprompts	95
9.5.2.16	ssh_userauth_kbdint_getprompt	95

9.5.2.17	ssh_userauth_kbdint_setanswer	96
9.5.2.18	ssh_userauth_list	96
9.5.2.19	ssh_userauth_none	97
9.5.2.20	ssh_userauth_offer_pubkey	97
9.5.2.21	ssh_userauth_password	98
9.5.2.22	ssh_userauth_privatekey_file	98
9.5.2.23	ssh_userauth_pubkey	99
9.6	The SSH buffer functions.	100
9.6.1	Detailed Description	100
9.6.2	Function Documentation	100
9.6.2.1	ssh_buffer_free	100
9.6.2.2	ssh_buffer_get_begin	101
9.6.2.3	ssh_buffer_get_len	101
9.6.2.4	ssh_buffer_new	101
9.7	The SSH channel functions	102
9.7.1	Detailed Description	104
9.7.2	Function Documentation	104
9.7.2.1	channel_read_buffer	104
9.7.2.2	ssh_channel_accept_x11	105
9.7.2.3	ssh_channel_change_pty_size	105
9.7.2.4	ssh_channel_close	106
9.7.2.5	ssh_channel_free	106
9.7.2.6	ssh_channel_get_exit_status	106
9.7.2.7	ssh_channel_get_session	107
9.7.2.8	ssh_channel_is_closed	107
9.7.2.9	ssh_channel_is_eof	107
9.7.2.10	ssh_channel_is_open	107
9.7.2.11	ssh_channel_new	108
9.7.2.12	ssh_channel_open_forward	108
9.7.2.13	ssh_channel_open_session	109
9.7.2.14	ssh_channel_poll	109
9.7.2.15	ssh_channel_read	109
9.7.2.16	ssh_channel_read_nonblocking	110
9.7.2.17	ssh_channel_request_env	111
9.7.2.18	ssh_channel_request_exec	111
9.7.2.19	ssh_channel_request_pty	112
9.7.2.20	ssh_channel_request_pty_size	112
9.7.2.21	ssh_channel_request_send_signal	112
9.7.2.22	ssh_channel_request_shell	113
9.7.2.23	ssh_channel_request_subsystem	113
9.7.2.24	ssh_channel_request_x11	114
9.7.2.25	ssh_channel_select	114
9.7.2.26	ssh_channel_send_eof	115
9.7.2.27	ssh_channel_set_blocking	115
9.7.2.28	ssh_channel_write	115
9.7.2.29	ssh_forward_accept	116
9.7.2.30	ssh_forward_cancel	116
9.7.2.31	ssh_forward_listen	116
9.8	The SSH error functions.	117
9.8.1	Detailed Description	117

9.8.2	Function Documentation	117
9.8.2.1	ssh_get_error	117
9.8.2.2	ssh_get_error_code	118
9.9	The libssh API	118
9.9.1	Detailed Description	119
9.9.2	Function Documentation	119
9.9.2.1	ssh_finalize	119
9.9.2.2	ssh_init	120
9.10	The SSH logging functions.	120
9.10.1	Detailed Description	120
9.10.2	Enumeration Type Documentation	121
9.10.2.1	"@0	121
9.10.3	Function Documentation	121
9.10.3.1	ssh_log	121
9.11	The SSH message functions	121
9.11.1	Detailed Description	122
9.11.2	Function Documentation	122
9.11.2.1	ssh_message_get	122
9.12	The SSH helper functions.	122
9.12.1	Detailed Description	123
9.12.2	Function Documentation	123
9.12.2.1	ssh_basename	123
9.12.2.2	ssh_dirname	123
9.12.2.3	ssh_getpass	124
9.12.2.4	ssh_mkdir	124
9.12.2.5	ssh_path_expand_tilde	125
9.12.2.6	ssh_timeout_update	125
9.12.2.7	ssh_version	125
9.13	The SSH Public Key Infrastructure	126
9.13.1	Detailed Description	126
9.13.2	Function Documentation	126
9.13.2.1	ssh_key_clean	126
9.13.2.2	ssh_key_free	127
9.13.2.3	ssh_key_import_private	127
9.13.2.4	ssh_key_new	127
9.13.2.5	ssh_key_type	127
9.14	The SSH poll functions.	128
9.14.1	Detailed Description	129
9.14.2	Function Documentation	129
9.14.2.1	ssh_poll_add_events	129
9.14.2.2	ssh_poll_ctx_add	129
9.14.2.3	ssh_poll_ctx_add_socket	130
9.14.2.4	ssh_poll_ctx_dopoll	130
9.14.2.5	ssh_poll_ctx_free	130
9.14.2.6	ssh_poll_ctx_new	131
9.14.2.7	ssh_poll_ctx_remove	131
9.14.2.8	ssh_poll_free	131
9.14.2.9	ssh_poll_get_ctx	131
9.14.2.10	ssh_poll_get_events	132
9.14.2.11	ssh_poll_get_fd	132

9.14.2.12	ssh_poll_new	132
9.14.2.13	ssh_poll_remove_events	133
9.14.2.14	ssh_poll_set_callback	133
9.14.2.15	ssh_poll_set_events	133
9.14.2.16	ssh_poll_set_fd	133
9.15	The SSH scp functions	134
9.15.1	Detailed Description	135
9.15.2	Function Documentation	135
9.15.2.1	ssh_scp_accept_request	135
9.15.2.2	ssh_scp_deny_request	135
9.15.2.3	ssh_scp_integer_mode	136
9.15.2.4	ssh_scp_leave_directory	136
9.15.2.5	ssh_scp_new	136
9.15.2.6	ssh_scp_pull_request	136
9.15.2.7	ssh_scp_push_directory	137
9.15.2.8	ssh_scp_push_file	137
9.15.2.9	ssh_scp_read	138
9.15.2.10	ssh_scp_read_string	138
9.15.2.11	ssh_scp_request_get_filename	138
9.15.2.12	ssh_scp_request_get_permissions	139
9.15.2.13	ssh_scp_request_get_size	139
9.15.2.14	ssh_scp_request_get_warning	139
9.15.2.15	ssh_scp_string_mode	139
9.15.2.16	ssh_scp_write	139
9.16	The SSH session functions.	140
9.16.1	Detailed Description	142
9.16.2	Function Documentation	142
9.16.2.1	ssh_blocking_flush	142
9.16.2.2	ssh_clean_pubkey_hash	142
9.16.2.3	ssh_connect	143
9.16.2.4	ssh_disconnect	143
9.16.2.5	ssh_free	143
9.16.2.6	ssh_get_disconnect_message	144
9.16.2.7	ssh_get_fd	144
9.16.2.8	ssh_get_issue_banner	144
9.16.2.9	ssh_get_openssh_version	145
9.16.2.10	ssh_get_pubkey_hash	145
9.16.2.11	ssh_get_status	145
9.16.2.12	ssh_get_version	146
9.16.2.13	ssh_is_blocking	146
9.16.2.14	ssh_is_connected	146
9.16.2.15	ssh_is_server_known	147
9.16.2.16	ssh_new	147
9.16.2.17	ssh_options_copy	147
9.16.2.18	ssh_options_getopt	148
9.16.2.19	ssh_options_parse_config	148
9.16.2.20	ssh_options_set	149
9.16.2.21	ssh_select	152
9.16.2.22	ssh_set_blocking	152
9.16.2.23	ssh_set_fd_except	153

9.16.2.24	ssh_set_fd_toread	153
9.16.2.25	ssh_set_fd_towrite	153
9.16.2.26	ssh_silent_disconnect	153
9.16.2.27	ssh_write_knownhost	154
9.17	The SSH string functions	154
9.17.1	Detailed Description	155
9.17.2	Function Documentation	155
9.17.2.1	ssh_string_burn	155
9.17.2.2	ssh_string_copy	155
9.17.2.3	ssh_string_data	155
9.17.2.4	ssh_string_fill	156
9.17.2.5	ssh_string_free	156
9.17.2.6	ssh_string_free_char	156
9.17.2.7	ssh_string_from_char	157
9.17.2.8	ssh_string_len	157
9.17.2.9	ssh_string_new	157
9.17.2.10	ssh_string_to_char	158
9.18	The SSH threading functions.	158
9.18.1	Detailed Description	158
9.18.2	Function Documentation	159
9.18.2.1	ssh_threads_get_noop	159
9.18.2.2	ssh_threads_set_callbacks	159
10	Data Structure Documentation	161
10.1	ssh::Channel Class Reference	161
10.1.1	Detailed Description	162
10.1.2	Member Function Documentation	162
10.1.2.1	acceptX11	162
10.1.2.2	changePtySize	162
10.1.2.3	close	163
10.1.2.4	isClosed	163
10.1.2.5	isEof	163
10.1.2.6	isOpen	163
10.1.2.7	write	163
10.2	ssh::Session Class Reference	164
10.2.1	Detailed Description	166
10.2.2	Member Function Documentation	166
10.2.2.1	acceptForward	166
10.2.2.2	connect	166
10.2.2.3	disconnect	167
10.2.2.4	getAuthList	167
10.2.2.5	getDisconnectMessage	167
10.2.2.6	getIssueBanner	167
10.2.2.7	getOpensshVersion	168
10.2.2.8	getSocket	168
10.2.2.9	getVersion	168
10.2.2.10	isServerKnown	168
10.2.2.11	optionsCopy	169
10.2.2.12	optionsParseConfig	169
10.2.2.13	setOption	169

10.2.2.14	setOption	170
10.2.2.15	setOption	170
10.2.2.16	silentDisconnect	171
10.2.2.17	userauthAutopubkey	171
10.2.2.18	userauthNone	171
10.2.2.19	userauthOfferPubkey	172
10.2.2.20	userauthPassword	172
10.2.2.21	userauthPubkey	172
10.2.2.22	writeKnownhost	173
10.3	ssh_bind_callbacks Struct Reference	173
10.3.1	Detailed Description	174
10.3.2	Field Documentation	174
10.3.2.1	incoming_connection	174
10.3.2.2	size	174
10.4	ssh_callbacks Struct Reference	174
10.4.1	Detailed Description	175
10.4.2	Field Documentation	175
10.4.2.1	auth_function	175
10.4.2.2	size	175
10.4.2.3	userdata	175
10.5	ssh_socket_callbacks Struct Reference	175
10.5.1	Detailed Description	176
10.5.2	Field Documentation	176
10.5.2.1	controlflow	176
10.5.2.2	data	176
10.5.2.3	exception	176
10.5.2.4	userdata	176
10.6	ssh::SshException Class Reference	176
10.6.1	Detailed Description	177
10.6.2	Member Function Documentation	177
10.6.2.1	getCode	177
10.6.2.2	getError	177
11	File Documentation	179
11.1	include/libssh/sftp.h File Reference	179
11.1.1	Detailed Description	183

Chapter 1

Main Page

This is the online reference for developing with the libssh library. It documents the libssh C API and the C++ wrapper.

1.1 Linking

We created a small howto how to link libssh against your application, read [The Linking HowTo](#).

1.2 Tutorial

You should start by reading [The Tutorial](#), then reading the documentation of the interesting functions as you go.

1.3 Features

The libssh library provides:

- Full C library functions for manipulating a client-side SSH connection
- SSH2 and SSH1 protocol compliant
- Fully configurable sessions
- Server support
- SSH agent authentication support
- Support for AES-128, AES-192, AES-256, Blowfish, 3DES in CBC mode, and AES in CTR mode
- Supports OpenSSL and GCrypt

- Use multiple SSH connections in a same process, at same time
- Use multiple channels in the same connection
- Thread safety when using different sessions at same time
- POSIX-like SFTP (Secure File Transfer) implementation with openssh extension support
- SCP implementation
- Large file system support (files bigger than 4GB)
- RSA and DSS server public key supported
- Compression support (with zlib)
- Public key (RSA and DSS), password and keyboard-interactive authentication
- Full poll()/WSAPoll() support and a poll-emulation for Win32.
- Runs and tested under x86_64, x86, ARM, Sparc32, PPC under Linux, BSD, MacOSX, Solaris and Windows

1.4 Copyright Policy

The developers of libssh have a policy of asking for contributions to be made under the personal copyright of the contributor, instead of a corporate copyright.

There are some reasons for the establishment of this policy:

- Individual copyrights make copyright registration in the US a simpler process.
- If libssh is copyrighted by individuals rather than corporations, decisions regarding enforcement and protection of copyright will, more likely, be made in the interests of the project, and not in the interests of any corporation's shareholders.
- If we ever need to relicense a portion of the code contacting individuals for permission to do so is much easier than contacting a company.

1.5 Internet standard

1.5.1 Secure Shell (SSH)

The following RFC documents described SSH-2 protocol as an Internet standard.

- [RFC 4250](#), The Secure Shell (SSH) Protocol Assigned Numbers
- [RFC 4251](#), The Secure Shell (SSH) Protocol Architecture
- [RFC 4252](#), The Secure Shell (SSH) Authentication Protocol

- [RFC 4253](#), The Secure Shell (SSH) Transport Layer Protocol
- [RFC 4254](#), The Secure Shell (SSH) Connection Protocol
- [RFC 4255](#), Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints
- [RFC 4256](#), Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)
- [RFC 4335](#), The Secure Shell (SSH) Session Channel Break Extension
- [RFC 4344](#), The Secure Shell (SSH) Transport Layer Encryption Modes
- [RFC 4345](#), Improved Arcfour Modes for the Secure Shell (SSH) Transport Layer Protocol

It was later modified and expanded by the following RFCs.

- [RFC 4419](#), Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol
- [RFC 4432](#), RSA Key Exchange for the Secure Shell (SSH) Transport Layer Protocol
- [RFC 4462](#), Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol
- [RFC 4716](#), The Secure Shell (SSH) Public Key File Format
- [RFC 5656](#), Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer

Interesting cryptography documents:

- [PKCS #11](#), PKCS #11 reference documents, describing interface with smart-cards.

1.5.2 Secure Shell File Transfer Protocol (SFTP)

The protocol is not an Internet standard but it is still widely implemented. OpenSSH and most other implementation implement Version 3 of the protocol. We do the same in libssh.

- [draft-ietf-secsh-filexfer-02.txt](#), SSH File Transfer Protocol

1.5.3 Secure Shell Extensions

The OpenSSH project has defined some extensions to the protocol. We support some of them like the statvfs calls in SFTP or the ssh-agent.

- [OpenSSH's deviations and extensions](#)
- [OpenSSH's ssh-agent](#)

Chapter 2

The Tutorial

2.1 Introduction

libssh is a C library that enables you to write a program that uses the SSH protocol. With it, you can remotely execute programs, transfer files, or use a secure and transparent tunnel for your remote programs. The SSH protocol is encrypted, ensures data integrity, and provides strong means of authenticating both the server and the client. The library hides a lot of technical details from the SSH protocol, but this does not mean that you should not try to know about and understand these details.

libssh is a Free Software / Open Source project. The libssh library is distributed under LGPL license. The libssh project has nothing to do with "libssh2", which is a completely different and independent project.

libssh can run on top of either libgcrypt (<http://directory.fsf.org/project/libgcrypt/>) or libcrypto (<http://www.openssl.org/docs/crypto/crypto.html>), two general-purpose cryptographic libraries.

This tutorial concentrates for its main part on the "client" side of libssh. To learn how to accept incoming SSH connections (how to write a SSH server), you'll have to jump to the end of this document.

This tutorial describes libssh version 0.5.0. This version is a little different from the 0.4.X series. However, the examples should work with little changes on versions like 0.4.2 and later.

Table of contents:

[Chapter 1: A typical SSH session](#)

[Chapter 2: A deeper insight on authentication](#)

[Chapter 3: Opening a remote shell](#)

[Chapter 4: Passing a remote command](#)

[Chapter 5: The SFTP subsystem](#)

[Chapter 6: The SCP subsystem](#)

[Chapter 7: Forwarding connections \(tunnel\)](#)

[Chapter 8: Threads with libssh](#)

[To be done](#)

2.2 Chapter 1: A typical SSH session

2.2.1 A typical SSH session

A SSH session goes through the following steps:

- Before connecting to the server, you can set up if you wish one or other server public key authentication, i.e. DSA or RSA. You can choose cryptographic algorithms you trust and compression algorithms if any. You must of course set up the hostname.
- The connection is established. A secure handshake is made, and resulting from it, a public key from the server is gained. You **MUST** verify that the public key is legitimate, using for instance the MD5 fingerprint or the known hosts file.
- The client must authenticate: the classical ways are password, or public keys (from dsa and rsa key-pairs generated by openssh). If a SSH agent is running, it is possible to use it.
- Now that the user has been authenticated, you must open one or several channels. Channels are different subways for information into a single ssh connection. Each channel has a standard stream (stdout) and an error stream (stderr). You can theoretically open an infinity of channels.
- With the channel you opened, you can do several things:
 - Execute a single command.
 - Open a shell. You may want to request a pseudo-terminal before.
 - Invoke the sftp subsystem to transfer files.
 - Invoke the scp subsystem to transfer files.
 - Invoke your own subsystem. This is outside the scope of this document, but can be done.
- When everything is finished, just close the channels, and then the connection.

The sftp and scp subsystems use channels, but libssh hides them to the programmer. If you want to use those subsystems, instead of a channel, you'll usually open a "sftp session" or a "scp session".

2.2.1.1 Creating the session and setting options

The most important object in a SSH connection is the SSH session. In order to allocate a new SSH session, you use `ssh_new()`. Don't forget to always verify that the allocation succeeded.

```
#include <libssh/libssh.h>
#include <stdlib.h>

int main()
{
    ssh_session my_ssh_session = ssh_new();
    if (my_ssh_session == NULL)
        exit(-1);
    ...
    ssh_free(my_ssh_session);
}
```

libssh follows the allocate-it-deallocate-it pattern. Each object that you allocate using `xxxxx_new()` must be deallocated using `xxxxx_free()`. In this case, `ssh_new()` does the allocation and `ssh_free()` does the contrary.

The `ssh_options_set()` function sets the options of the session. The most important options are:

- `SSH_OPTIONS_HOST`: the name of the host you want to connect to
- `SSH_OPTIONS_PORT`: the used port (default is port 22)
- `SSH_OPTIONS_USER`: the system user under which you want to connect
- `SSH_OPTIONS_LOG_VERBOSITY`: the quantity of messages that are printed

The complete list of options can be found in the documentation of `ssh_options_set()`. The only mandatory option is `SSH_OPTIONS_HOST`. If you don't use `SSH_OPTIONS_USER`, the local username of your account will be used.

Here is a small example of how to use it:

```
#include <libssh/libssh.h>
#include <stdlib.h>

int main()
{
    ssh_session my_ssh_session;
    int verbosity = SSH_LOG_PROTOCOL;
    int port = 22;

    my_ssh_session = ssh_new();
    if (my_ssh_session == NULL)
        exit(-1);

    ssh_options_set(my_ssh_session, SSH_OPTIONS_HOST, "localhost");
    ssh_options_set(my_ssh_session, SSH_OPTIONS_LOG_VERBOSITY, &verbosity);
    ssh_options_set(my_ssh_session, SSH_OPTIONS_PORT, &port);

    ...

    ssh_free(my_ssh_session);
}
```

Please notice that all parameters are passed to `ssh_options_set()` as pointers, even if you need to set an integer value.

See also

- [ssh_new](#)
- [ssh_free](#)
- [ssh_options_set](#)
- [ssh_options_parse_config](#)
- [ssh_options_copy](#)
- [ssh_options_getopt](#)

2.2.1.2 Connecting to the server

Once all settings have been made, you can connect using `ssh_connect()`. That function will return `SSH_OK` if the connection worked, `SSH_ERROR` otherwise.

You can get the English error string with `ssh_get_error()` in order to show the user what went wrong. Then, use `ssh_disconnect()` when you want to stop the session.

Here's an example:

```
#include <libssh/libssh.h>
#include <stdlib.h>
#include <stdio.h>

int main()
{
    ssh_session my_ssh_session;
    int rc;

    my_ssh_session = ssh_new();
    if (my_ssh_session == NULL)
        exit(-1);

    ssh_options_set(my_ssh_session, SSH_OPTIONS_HOST, "localhost");

    rc = ssh_connect(my_ssh_session);
    if (rc != SSH_OK)
    {
        fprintf(stderr, "Error connecting to localhost: %s\n",
                ssh_get_error(my_ssh_session));
        exit(-1);
    }

    ...

    ssh_disconnect(my_ssh_session);
    ssh_free(my_ssh_session);
}
```

2.2.1.3 Authenticating the server

Once you're connected, the following step is mandatory: you must check that the server you just connected to is known and safe to use (remember, SSH is about security and authentication).

There are two ways of doing this:

- The first way (recommended) is to use the `ssh_is_server_known()` function. This function will look into the known host file (`~/.ssh/known_hosts` on UNIX), look for the server hostname's pattern, and determine whether this host is present or not in the list.
- The second way is to use `ssh_get_pubkey_hash()` to get a binary version of the public key hash value. You can then use your own database to check if this public key is known and secure.

You can also use the `ssh_get_pubkey_hash()` to show the public key hash value to the user, in case he knows what the public key hash value is (some paranoid people write their public key hash values on paper before going abroad, just in case ...).

If the remote host is being used to for the first time, you can ask the user whether he/she trusts it. Once he/she concluded that the host is valid and worth being added in the known hosts file, you use `ssh_write_knownhost()` to register it in the known hosts file, or any other way if you use your own database.

The following example is part of the examples suite available in the `examples/` directory:

```
#include <errno.h>
#include <string.h>

int verify_knownhost(ssh_session session)
{
    int state, hlen;
    unsigned char *hash = NULL;
    char *hexa;
    char buf[10];

    state = ssh_is_server_known(session);

    hlen = ssh_get_pubkey_hash(session, &hash);
    if (hlen < 0)
        return -1;

    switch (state)
    {
        case SSH_SERVER_KNOWN_OK:
            break; /* ok */

        case SSH_SERVER_KNOWN_CHANGED:
            fprintf(stderr, "Host key for server changed: it is now:\n");
            ssh_print_hexa("Public key hash", hash, hlen);
            fprintf(stderr, "For security reasons, connection will be stopped\n");
            free(hash);
            return -1;

        case SSH_SERVER_FOUND_OTHER:
            fprintf(stderr, "The host key for this server was not found but an other"
                    "type of key exists.\n");
            fprintf(stderr, "An attacker might change the default server key to"
                    "confuse your client into thinking the key does not exist\n");
            free(hash);
            return -1;
    }
}
```

```

case SSH_SERVER_FILE_NOT_FOUND:
    fprintf(stderr, "Could not find known host file.\n");
    fprintf(stderr, "If you accept the host key here, the file will be"
        "automatically created.\n");
    /* fallback to SSH_SERVER_NOT_KNOWN behavior */

case SSH_SERVER_NOT_KNOWN:
    hexa = ssh_get_hexa(hash, hlen);
    fprintf(stderr, "The server is unknown. Do you trust the host key?\n");
    fprintf(stderr, "Public key hash: %s\n", hexa);
    free(hexa);
    if (fgets(buf, sizeof(buf), stdin) == NULL)
    {
        free(hash);
        return -1;
    }
    if (strncasecmp(buf, "yes", 3) != 0)
    {
        free(hash);
        return -1;
    }
    if (ssh_write_knownhost(session) < 0)
    {
        fprintf(stderr, "Error %s\n", strerror(errno));
        free(hash);
        return -1;
    }
    break;

case SSH_SERVER_ERROR:
    fprintf(stderr, "Error %s", ssh_get_error(session));
    free(hash);
    return -1;
}

free(hash);
return 0;
}

```

See also

[ssh_connect](#)
[ssh_disconnect](#)
[ssh_get_error](#)
[ssh_get_error_code](#)
[ssh_get_pubkey_hash](#)
[ssh_is_server_known](#)
[ssh_write_knownhost](#)

2.2.1.4 Authenticating the user

The authentication process is the way a service provider can identify a user and verify his/her identity. The authorization process is about enabling the authenticated user the access to resources. In SSH, the two concepts are linked. After authentication, the server can grant the user access to several resources such as port forwarding, shell, sftp subsystem, and so on.

libssh supports several methods of authentication:

- "none" method. This method allows to get the available authentications methods. It also gives the server a chance to authenticate the user with just his/her login. Some very old hardware uses this feature to fallback the user on a "telnet over SSH" style of login.
- password method. A password is sent to the server, which accepts it or not.
- keyboard-interactive method. The server sends several challenges to the user, who must answer correctly. This makes possible the authentication via a code-book for instance ("give code at 23:R on page 3").
- public key method. The host knows the public key of the user, and the user must prove he knows the associated private key. This can be done manually, or delegated to the SSH agent as we'll see later.

All these methods can be combined. You can for instance force the user to authenticate with at least two of the authentication methods. In that case, one speaks of "Partial authentication". A partial authentication is a response from authentication functions stating that your credential was accepted, but yet another one is required to get in.

The example below shows an authentication with password:

```
#include <libssh/libssh.h>
#include <stdlib.h>
#include <stdio.h>

int main()
{
    ssh_session my_ssh_session;
    int rc;
    char *password;

    // Open session and set options
    my_ssh_session = ssh_new();
    if (my_ssh_session == NULL)
        exit(-1);
    ssh_options_set(my_ssh_session, SSH_OPTIONS_HOST, "localhost");

    // Connect to server
    rc = ssh_connect(my_ssh_session);
    if (rc != SSH_OK)
    {
        fprintf(stderr, "Error connecting to localhost: %s\n",
                ssh_get_error(my_ssh_session));
        ssh_free(my_ssh_session);
        exit(-1);
    }

    // Verify the server's identity
    // For the source code of verify_knownhost(), check previous example
    if (verify_knownhost(my_ssh_session) < 0)
    {
        ssh_disconnect(my_ssh_session);
        ssh_free(my_ssh_session);
        exit(-1);
    }
}
```

```

// Authenticate ourselves
password = getpass("Password: ");
rc = ssh_userauth_password(my_ssh_session, NULL, password);
if (rc != SSH_AUTH_SUCCESS)
{
    fprintf(stderr, "Error authenticating with password: %s\n",
            ssh_get_error(my_ssh_session));
    ssh_disconnect(my_ssh_session);
    ssh_free(my_ssh_session);
    exit(-1);
}

...

ssh_disconnect(my_ssh_session);
ssh_free(my_ssh_session);
}

```

See also

[A deeper insight on authentication](#)

2.2.1.5 Doing something

At this point, the authenticity of both server and client is established. Time has come to take advantage of the many possibilities offered by the SSH protocol: execute a remote command, open remote shells, transfer files, forward ports, etc.

The example below shows how to execute a remote command:

```

int show_remote_processes(ssh_session session)
{
    ssh_channel channel;
    int rc;
    char buffer[256];
    unsigned int nbytes;

    channel = ssh_channel_new(session);
    if (channel == NULL)
        return SSH_ERROR;

    rc = ssh_channel_open_session(channel);
    if (rc != SSH_OK)
    {
        ssh_channel_free(channel);
        return rc;
    }

    rc = ssh_channel_request_exec(channel, "ps aux");
    if (rc != SSH_OK)
    {
        ssh_channel_close(channel);
        ssh_channel_free(channel);
        return rc;
    }

    nbytes = ssh_channel_read(channel, buffer, sizeof(buffer), 0);
    while (nbytes > 0)

```



```
{
    if (write(1, buffer, nbytes) != nbytes)
    {
        ssh_channel_close(channel);
        ssh_channel_free(channel);
        return SSH_ERROR;
    }
    nbytes = ssh_channel_read(channel, buffer, sizeof(buffer), 0);
}

if (nbytes < 0)
{
    ssh_channel_close(channel);
    ssh_channel_free(channel);
    return SSH_ERROR;
}

ssh_channel_send_eof(channel);
ssh_channel_close(channel);
ssh_channel_free(channel);

return SSH_OK;
}
```

See also

[Opening a remote shell](#)
[Passing a remote command](#)
[The SFTP subsystem](#)
[The SCP subsystem](#)

2.2.1.6 Handling the errors

All the libssh functions which return an error value also set an English error message describing the problem.

Error values are typically SSH_ERROR for integer values, or NULL for pointers.

The function [ssh_get_error\(\)](#) returns a pointer to the static error message.

[ssh_error_code\(\)](#) returns the error code number: SSH_NO_ERROR, SSH_REQUEST_DENIED, SSH_INVALID_REQUEST, SSH_CONNECTION_LOST, SSH_FATAL, or SSH_INVALID_DATA. SSH_REQUEST_DENIED means the ssh server refused your request, but the situation is recoverable. The others mean something happened to the connection (some encryption problems, server problems, ...). SSH_INVALID_REQUEST means the library got some garbage from server, but might be recoverable. SSH_FATAL means the connection has an important problem and isn't probably recoverable.

Most of time, the error returned are SSH_FATAL, but some functions (generally the [ssh_request_xxx](#) ones) may fail because of server denying request. In these cases, SSH_REQUEST_DENIED is returned.

[ssh_get_error\(\)](#) and [ssh_get_error_code\(\)](#) take a [ssh_session](#) as a parameter. That's for thread safety, error messages that can be attached to a session aren't static anymore. Any error that happens during [ssh_options_xxx\(\)](#) or [ssh_connect\(\)](#) (i.e., outside of any session) can be retrieved by giving NULL as argument.

The SFTP subsystem has its own error codes, in addition to libssh ones.

2.3 Chapter 2: A deeper insight on authentication

2.3.1 A deeper insight on authentication

In our guided tour, we merely mentioned that the user needed to authenticate. We didn't explain much in detail how that was supposed to happen. This chapter explains better the four authentication methods: with public keys, with a password, with challenges and responses (keyboard-interactive), and with no authentication at all.

If your software is supposed to connect to an arbitrary server, then you might need to support all authentication methods. If your software will connect only to a given server, then it might be enough for your software to support only the authentication methods used by that server. If you are the administrator of the server, it might be your call to choose those authentication methods.

It is not the purpose of this document to review in detail the advantages and drawbacks of each authentication method. You are therefore invited to read the abundant documentation on this topic to fully understand the advantages and security risks linked to each method.

2.3.1.1 Authenticating with public keys

libssh is fully compatible with the openssh public and private keys. You can either use the automatic public key authentication method provided by libssh, or roll your own using the public key functions.

The process of authenticating by public key to a server is the following:

- you scan a list of files that contain public keys. each key is sent to the SSH server, until the server acknowledges a key (a key it knows can be used to authenticate the user).
- then, you retrieve the private key for this key and send a message proving that you know that private key.

The function `ssh_userauth_autopubkey()` does this using the available keys in "`~/.ssh/`". The return values are the following:

- `SSH_AUTH_ERROR`: some serious error happened during authentication
- `SSH_AUTH_DENIED`: no key matched
- `SSH_AUTH_SUCCESS`: you are now authenticated
- `SSH_AUTH_PARTIAL`: some key matched but you still have to provide an other mean of authentication (like a password).

The `ssh_userauth_autopubkey()` function also tries to authenticate using the SSH agent, if you have one running, or the "none" method otherwise.

If you wish to authenticate with public key by your own, follow these steps:

- Retrieve the public key in a `ssh_string` using `publickey_from_file()`.
- Offer the public key to the SSH server using `ssh_userauth_offer_pubkey()`. If the return value is `SSH_AUTH_SUCCESS`, the SSH server accepts to authenticate using the public key and you can go to the next step.
- Retrieve the private key, using the `privatekey_from_file()` function. If a passphrase is needed, either the passphrase specified as argument or a callback (see callbacks section) will be used.
- Authenticate using `ssh_userauth_pubkey()` with your public key string and private key.
- Do not forget cleaning up memory using `string_free()` and `privatekey_free()`.

Here is a minimalistic example of public key authentication:

```
int authenticate_pubkey(ssh_session session)
{
    int rc;

    rc = ssh_userauth_autopubkey(session, NULL);

    if (rc == SSH_AUTH_ERROR)
    {
        fprintf(stderr, "Authentication failed: %s\n",
            ssh_get_error(session));
        return SSH_AUTH_ERROR;
    }

    return rc;
}
```

See also

[ssh_userauth_autopubkey](#)
[ssh_userauth_offer_pubkey](#)
[ssh_userauth_pubkey](#)
[publickey_from_file](#)
[publickey_from_privatekey](#)
[string_free](#)
[privatekey_from_file](#)
[privatekey_free](#)

2.3.1.2 Authenticating with a password

The function `ssh_userauth_password()` serves the purpose of authenticating using a password. It will return `SSH_AUTH_SUCCESS` if the password worked, or one of other constants otherwise. It's your work to ask the password and to deallocate it in a secure manner.

If your server complains that the password is wrong, but you can still authenticate using openssh's client (issuing password), it's probably because openssh only accept keyboard-interactive. Switch to keyboard-interactive authentication, or try to configure plain text passwords on the SSH server.

Here is a small example of password authentication:

```
int authenticate_password(ssh_session session)
{
    char *password;
    int rc;

    password = getpass("Enter your password: ");
    rc = ssh_userauth_password(session, NULL, password);
    if (rc == SSH_AUTH_ERROR)
    {
        fprintf(stderr, "Authentication failed: %s\n",
            ssh_get_error(session));
        return SSH_AUTH_ERROR;
    }

    return rc;
}
```

See also

[ssh_userauth_password](#)

2.3.1.3 The keyboard-interactive authentication method

The keyboard-interactive method is, as its name tells, interactive. The server will issue one or more challenges that the user has to answer, until the server takes an authentication decision.

[ssh_userauth_kbdint\(\)](#) is the the main keyboard-interactive function. It will return `SSH_AUTH_SUCCESS`, `SSH_AUTH_DENIED`, `SSH_AUTH_PARTIAL`, `SSH_AUTH_ERROR`, or `SSH_AUTH_INFO`, depending on the result of the request.

The keyboard-interactive authentication method of SSH2 is a feature that permits the server to ask a certain number of questions in an interactive manner to the client, until it decides to accept or deny the login.

To begin, you call [ssh_userauth_kbdint\(\)](#) (just set user and submethods to `NULL`) and store the answer.

If the answer is `SSH_AUTH_INFO`, it means that the server has sent a few questions that you should ask the user. You can retrieve these questions with the following functions: [ssh_userauth_kbdint_getnprompts\(\)](#), [ssh_userauth_kbdint_getname\(\)](#), [ssh_userauth_kbdint_getinstruction\(\)](#), and [ssh_userauth_kbdint_getprompt\(\)](#).

Set the answer for each question in the challenge using [ssh_userauth_kbdint_setanswer\(\)](#).

Then, call again [ssh_userauth_kbdint\(\)](#) and start the process again until these functions returns something else than `SSH_AUTH_INFO`.

Here are a few remarks:

- Even the first call can return `SSH_AUTH_DENIED` or `SSH_AUTH_SUCCESS`.

- The server can send an empty question set (this is the default behavior on my system) after you have sent the answers to the first questions. You must still parse the answer, it might contain some message from the server saying hello or such things. Just call `ssh_userauth_kbdint()` until needed.
- The meaning of "name", "prompt", "instruction" may be a little confusing. An explanation is given in the RFC section that follows.

Here is a little note about how to use the information from keyboard-interactive authentication, coming from the RFC itself (rfc4256):

3.3 User Interface Upon receiving a request message, the client SHOULD prompt the user as follows: A command line interface (CLI) client SHOULD print the name and instruction (if non-empty), adding newlines. Then for each prompt in turn, the client SHOULD display the prompt and read the user input.

A graphical user interface (GUI) client has many choices on how to prompt the user. One possibility is to use the name field (possibly prefixed with the application's name) as the title of a dialog window in which the prompt(s) are presented. In that dialog window, the instruction field would be a text message, and the prompts would be labels for text entry fields. All fields SHOULD be presented to the user, for example an implementation SHOULD NOT discard the name field because its windows lack titles; it SHOULD instead find another way to display this information. If prompts are presented in a dialog window, then the client SHOULD NOT present each prompt in a separate window.

All clients MUST properly handle an instruction field with embedded newlines. They SHOULD also be able to display at least 30 characters for the name and prompts. If the server presents names or prompts longer than 30 characters, the client MAY truncate these fields to the length it can display. If the client does truncate any fields, there MUST be an obvious indication that such truncation has occurred.

The instruction field SHOULD NOT be truncated. Clients SHOULD use control character filtering as discussed in [SSH-ARCH] to avoid attacks by including terminal control characters in the fields to be displayed.

For each prompt, the corresponding echo field indicates whether or not the user input should be echoed as characters are typed. Clients SHOULD correctly echo/mask user input for each prompt independently of other prompts in the request message. If a client does not honor the echo field for whatever reason, then the client MUST err on the side of masking input. A GUI client might like to have a checkbox toggling echo/mask. Clients SHOULD NOT add any additional characters to the prompt such as ": " (colon-space); the server is responsible for supplying all text to be displayed to the user. Clients MUST also accept empty responses from the user and pass them on as empty strings.

The following example shows how to perform keyboard-interactive authentication:

```
int authenticate_kbdint(ssh_session session)
{
    int rc;

    rc = ssh_userauth_kbdint(session, NULL, NULL);
    while (rc == SSH_AUTH_INFO)
    {
```

```

const char *name, *instruction;
int nprompts, iprompt;

name = ssh_userauth_kbdint_getname(session);
instruction = ssh_userauth_kbdint_getinstruction(session);
nprompts = ssh_userauth_kbdint_getnprompts(session);

if (strlen(name) > 0)
    printf("%s\n", name);
if (strlen(instruction) > 0)
    printf("%s\n", instruction);
for (iprompt = 0; iprompt < nprompts; iprompt++)
{
    const char *prompt;
    char echo;

    prompt = ssh_userauth_kbdint_getprompt(session, iprompt, &echo);
    if (echo)
    {
        char buffer[128], *ptr;

        printf("%s", prompt);
        if (fgets(buffer, sizeof(buffer), stdin) == NULL)
            return SSH_AUTH_ERROR;
        buffer[sizeof(buffer) - 1] = '\0';
        if ((ptr = strchr(buffer, '\n')) != NULL)
            *ptr = '\0';
        if (ssh_userauth_kbdint_setanswer(session, iprompt, buffer) < 0)
            return SSH_AUTH_ERROR;
        memset(buffer, 0, strlen(buffer));
    }
    else
    {
        char *ptr;

        ptr = getpass(prompt);
        if (ssh_userauth_kbdint_setanswer(session, iprompt, ptr) < 0)
            return SSH_AUTH_ERROR;
    }
}
rc = ssh_userauth_kbdint(session, NULL, NULL);
}
return rc;
}

```

See also

[ssh_userauth_kbdint\(\)](#)
[ssh_userauth_kbdint_getnprompts](#)
[ssh_userauth_kbdint_getname](#)
[ssh_userauth_kbdint_getinstruction](#)
[ssh_userauth_kbdint_getprompt](#)
[ssh_userauth_kbdint_setanswer\(\)](#)

2.3.1.4 Authenticating with "none" method

The primary purpose of the "none" method is to get authenticated **without** any credential. Don't do that, use one of the other authentication methods, unless you really want to grant anonymous access.

If the account has no password, and if the server is configured to let you pass, `ssh_userauth_none()` might answer `SSH_AUTH_SUCCESS`.

The following example shows how to perform "none" authentication:

```
int authenticate_kbdint (ssh_session session)
{
    int rc;

    rc = ssh_userauth_none(session, NULL, NULL);
    return rc;
}
```

2.3.1.5 Getting the list of supported authentications

You are not meant to choose a given authentication method, you can let the server tell you which methods are available. Once you know them, you try them one after the other.

The following example shows how to get the list of available authentication methods with `ssh_userauth_list()` and how to use the result:

```
int test_several_auth_methods(ssh_session session)
{
    int method, rc;

    method = ssh_userauth_list(session, NULL);

    if (method & SSH_AUTH_METHOD_NONE)
    { // For the source code of function authenticate_none(),
      // refer to the corresponding example
      rc = authenticate_none(session);
      if (rc == SSH_AUTH_SUCCESS) return rc;
    }
    if (method & SSH_AUTH_METHOD_PUBLICKEY)
    { // For the source code of function authenticate_pubkey(),
      // refer to the corresponding example
      rc = authenticate_pubkey(session);
      if (rc == SSH_AUTH_SUCCESS) return rc;
    }
    if (method & SSH_AUTH_METHOD_INTERACTIVE)
    { // For the source code of function authenticate_kbdint(),
      // refer to the corresponding example
      rc = authenticate_kbdint(session);
      if (rc == SSH_AUTH_SUCCESS) return rc;
    }
    if (method & SSH_AUTH_METHOD_PASSWORD)
    { // For the source code of function authenticate_password(),
      // refer to the corresponding example
      rc = authenticate_password(session);
      if (rc == SSH_AUTH_SUCCESS) return rc;
    }
    return SSH_AUTH_ERROR;
}
```

2.3.1.6 Getting the banner

The SSH server might send a banner, which you can retrieve with [ssh_get_issue_banner\(\)](#), then display to the user.

The following example shows how to retrieve and dispose the issue banner:

```
int display_banner(ssh_session session)
{
    int rc;
    char *banner;

    /*
     * Does not work without calling ssh_userauth_none() first ***
     * That will be fixed ***
     */
    rc = ssh_userauth_none(session, NULL);
    if (rc == SSH_AUTH_ERROR)
        return rc;

    banner = ssh_get_issue_banner(session);
    if (banner)
    {
        printf("%s\n", banner);
        free(banner);
    }

    return rc;
}
```

2.4 Chapter 3: Opening a remote shell

2.4.1 Opening a remote shell

We already mentioned that a single SSH connection can be shared between several "channels". Channels can be used for different purposes.

This chapter shows how to open one of these channels, and how to use it to start a command interpreter on a remote computer.

2.4.1.1 Opening and closing a channel

The [ssh_channel_new\(\)](#) function creates a channel. It returns the channel as a variable of type `ssh_channel`.

Once you have this channel, you open a SSH session that uses it with [ssh_channel_open_session\(\)](#).

Once you don't need the channel anymore, you can send an end-of-file to it with [ssh_channel_close\(\)](#). At this point, you can destroy the channel with [ssh_channel_free\(\)](#).

The code sample below achieves these tasks:

```
int shell_session(ssh_session session)
```



```

{
    ssh_channel channel;
    int rc;

    channel = ssh_channel_new(session);
    if (channel == NULL)
        return SSH_ERROR;

    rc = ssh_channel_open_session(channel);
    if (rc != SSH_OK)
    {
        ssh_channel_free(channel);
        return rc;
    }

    ...

    ssh_channel_close(channel);
    ssh_channel_send_eof(channel);
    ssh_channel_free(channel);

    return SSH_OK;
}

```

2.4.1.2 Interactive and non-interactive sessions

A "shell" is a command interpreter. It is said to be "interactive" if there is a human user typing the commands, one after the other. The contrary, a non-interactive shell, is similar to the execution of commands in the background: there is no attached terminal.

If you plan using an interactive shell, you need to create a pseud-terminal on the remote side. A remote terminal is usually referred to as a "pty", for "pseudo-teletype". The remote processes won't see the difference with a real text-oriented terminal.

If needed, you request the pty with the function `ssh_channel_request_pty()`. Then you define its dimensions (number of rows and columns) with `ssh_channel_change_pty_size()`.

Be your session interactive or not, the next step is to request a shell with `ssh_channel_request_shell()`.

```

int interactive_shell_session(ssh_channel channel)
{
    int rc;

    rc = ssh_channel_request_pty(channel);
    if (rc != SSH_OK) return rc;

    rc = ssh_channel_change_pty_size(channel, 80, 24);
    if (rc != SSH_OK) return rc;

    rc = ssh_channel_request_shell(channel);
    if (rc != SSH_OK) return rc;

    ...

    return rc;
}

```

2.4.1.3 Displaying the data sent by the remote computer

In your program, you will usually need to receive all the data "displayed" into the remote pty. You will usually analyse, log, or display this data.

[ssh_channel_read\(\)](#) and [ssh_channel_read_nonblocking\(\)](#) are the simplest way to read data from a channel. If you only need to read from a single channel, they should be enough.

The example below shows how to wait for remote data using [ssh_channel_read\(\)](#):

```
int interactive_shell_session(ssh_channel channel)
{
    int rc;
    char buffer[256];
    int nbytes;

    rc = ssh_channel_request_pty(channel);
    if (rc != SSH_OK) return rc;

    rc = ssh_channel_change_pty_size(channel, 80, 24);
    if (rc != SSH_OK) return rc;

    rc = ssh_channel_request_shell(channel);
    if (rc != SSH_OK) return rc;

    while (ssh_channel_is_open(channel) &&
           !ssh_channel_is_eof(channel))
    {
        nbytes = ssh_channel_read(channel, buffer, sizeof(buffer), 0);
        if (nbytes < 0)
            return SSH_ERROR;

        if (nbytes > 0)
            write(1, buffer, nbytes);
    }

    return rc;
}
```

Unlike [ssh_channel_read\(\)](#), [ssh_channel_read_nonblocking\(\)](#) never waits for remote data to be ready. It returns immediately.

If you plan to use [ssh_channel_read_nonblocking\(\)](#) repeatedly in a loop, you should use a "passive wait" function like `usleep(3)` in the same loop. Otherwise, your program will consume all the CPU time, and your computer might become unresponsive.

2.4.1.4 Sending user input to the remote computer

User's input is sent to the remote site with [ssh_channel_write\(\)](#).

The following example shows how to combine a nonblocking read from a SSH channel with a nonblocking read from the keyboard. The local input is then sent to the remote computer:

```
/* Under Linux, this function determines whether a key has been pressed.
   Under Windows, it is a standard function, so you need not redefine it.
```

```

*/
int kbhit()
{
    struct timeval tv = { 0L, 0L };
    fd_set fds;

    FD_ZERO(&fds);
    FD_SET(0, &fds);

    return select(1, &fds, NULL, NULL, &tv);
}

/* A very simple terminal emulator:
   - print data received from the remote computer
   - send keyboard input to the remote computer
*/
int interactive_shell_session(ssh_channel channel)
{
    /* Session and terminal initialization skipped */
    ...

    char buffer[256];
    int nbytes, nwritten;

    while (ssh_channel_is_open(channel) &&
           !ssh_channel_is_eof(channel))
    {
        nbytes = ssh_channel_read_nonblocking(channel, buffer, sizeof(buffer), 0);
        if (nbytes < 0) return SSH_ERROR;
        if (nbytes > 0)
        {
            nwritten = write(1, buffer, nbytes);
            if (nwritten != nbytes) return SSH_ERROR;

            if (!kbhit())
            {
                usleep(50000L); // 0.05 second
                continue;
            }

            nbytes = read(0, buffer, sizeof(buffer));
            if (nbytes < 0) return SSH_ERROR;
            if (nbytes > 0)
            {
                nwritten = ssh_channel_write(channel, buffer, nbytes);
                if (nwritten != nbytes) return SSH_ERROR;
            }
        }
    }

    return rc;
}

```

Of course, this is a poor terminal emulator, since the echo from the keys pressed should not be done locally, but should be done by the remote side. Also, user's input should not be sent once "Enter" key is pressed, but immediately after each key is pressed. This can be accomplished by setting the local terminal to "raw" mode with the `cfmakeraw(3)` function. `cfmakeraw()` is a standard function under Linux, on other systems you can recode it with:

```
static void cfmakeraw(struct termios *termios_p)
```

```

{
    termios_p->c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL|IXON);
    termios_p->c_oflag &= ~OPOST;
    termios_p->c_lflag &= ~(ECHO|ECHONL|ICANON|ISIG|IEXTEN);
    termios_p->c_cflag &= ~(CSIZE|PARENB);
    termios_p->c_cflag |= CS8;
}

```

If you are not using a local terminal, but some kind of graphical environment, the solution to this kind of "echo" problems will be different.

2.4.1.5 A more elaborate way to get the remote data

Warning: `ssh_select()` and `ssh_channel_select()` are not relevant anymore, since libssh is about to provide an easier system for asynchronous communications. This subsection should be removed then. ***

`ssh_channel_read()` and `ssh_channel_read_nonblocking()` functions are simple, but they are not adapted when you expect data from more than one SSH channel, or from other file descriptors. Last example showed how getting data from the standard input (the keyboard) at the same time as data from the SSH channel was complicated. The functions `ssh_select()` and `ssh_channel_select()` provide a more elegant way to wait for data coming from many sources.

The functions `ssh_select()` and `ssh_channel_select()` remind of the standard UNIX `select(2)` function. The idea is to wait for "something" to happen: incoming data to be read, outgoing data to block, or an exception to occur. Both these functions do a "passive wait", i.e. you can safely use them repeatedly in a loop, it will not consume exaggerate processor time and make your computer unresponsive. It is quite common to use these functions in your application's main loop.

The difference between `ssh_select()` and `ssh_channel_select()` is that `ssh_channel_select()` is simpler, but allows you only to watch SSH channels. `ssh_select()` is more complete and enables watching regular file descriptors as well, in the same function call.

Below is an example of a function that waits both for remote SSH data to come, as well as standard input from the keyboard:

```

int interactive_shell_session(ssh_session session, ssh_channel channel)
{
    /* Session and terminal initialization skipped */
    ...

    char buffer[256];
    int nbytes, nwritten;

    while (ssh_channel_is_open(channel) &&
           !ssh_channel_is_eof(channel))
    {
        struct timeval timeout;
        ssh_channel in_channels[2], out_channels[2];
        fd_set fds;
        int maxfd;

```

```

    timeout.tv_sec = 30;
    timeout.tv_usec = 0;
    in_channels[0] = channel;
    in_channels[1] = NULL;
    FD_ZERO(&fds);
    FD_SET(0, &fds);
    FD_SET(ssh_get_fd(session), &fds);
    maxfd = ssh_get_fd(session) + 1;

    ssh_select(in_channels, out_channels, maxfd, &fds, &timeout);

    if (out_channels[0] != NULL)
    {
        nbytes = ssh_channel_read(channel, buffer, sizeof(buffer), 0);
        if (nbytes < 0) return SSH_ERROR;
        if (nbytes > 0)
        {
            nwritten = write(1, buffer, nbytes);
            if (nwritten != nbytes) return SSH_ERROR;
        }
    }

    if (FD_ISSET(0, &fds))
    {
        nbytes = read(0, buffer, sizeof(buffer));
        if (nbytes < 0) return SSH_ERROR;
        if (nbytes > 0)
        {
            nwritten = ssh_channel_write(channel, buffer, nbytes);
            if (nbytes != nwritten) return SSH_ERROR;
        }
    }
}

return rc;
}

```

2.4.1.6 Using graphical applications on the remote side

If your remote application is graphical, you can forward the X11 protocol to your local computer.

To do that, you first declare that you accept X11 connections with [ssh_channel_accept_x11\(\)](#). Then you create the forwarding tunnel for the X11 protocol with [ssh_channel_request_x11\(\)](#).

The following code performs channel initialization and shell session opening, and handles a parallel X11 connection:

```

int interactive_shell_session(ssh_channel channel)
{
    int rc;
    ssh_channel x11channel;

    rc = ssh_channel_request_pty(channel);
    if (rc != SSH_OK) return rc;

    rc = ssh_channel_change_pty_size(channel, 80, 24);
    if (rc != SSH_OK) return rc;
}

```

```
rc = ssh_channel_request_x11(channel, 0, NULL, NULL, 0);
if (rc != SSH_OK) return rc;

rc = ssh_channel_request_shell(channel);
if (rc != SSH_OK) return rc;

/* Read the data sent by the remote computer here */
...
}
```

Don't forget to set the `$DISPLAY` environment variable on the remote side, or the remote applications won't try using the X11 tunnel:

```
$ export DISPLAY=:0
$ xclock &
```

2.5 Chapter 4: Passing a remote command

2.5.1 Passing a remote command

Previous chapter has shown how to open a full shell session, with an attached terminal or not. If you only need to execute a command on the remote end, you don't need all that complexity.

The method described here is suited for executing only one remote command. If you need to issue several commands in a row, you should consider using a non-interactive remote shell, as explained in previous chapter.

See also

[shell](#)

2.5.1.1 Executing a remote command

The first steps for executing a remote command are identical to those for opening remote shells. You first need a SSH channel, and then a SSH session that uses this channel:

```
int show_remote_files(ssb_session session)
{
    ssh_channel channel;
    int rc;

    channel = ssh_channel_new(session);
    if (channel == NULL) return SSH_ERROR;

    rc = ssh_channel_open_session(channel);
    if (rc != SSH_OK)
    {
        ssh_channel_free(channel);
        return rc;
    }
}
```

Once a session is open, you can start the remote command with `ssh_channel_request_exec()`:

```
rc = ssh_channel_request_exec(channel, "ls -l");
if (rc != SSH_OK)
{
    ssh_channel_close(channel);
    ssh_channel_free(channel);
    return rc;
}
```

If the remote command displays data, you get them with `ssh_channel_read()`. This function returns the number of bytes read. If there is no more data to read on the channel, this function returns 0, and you can go to next step. If an error has been encountered, it returns a negative value:

```
char buffer[256];
unsigned int nbytes;

nbytes = ssh_channel_read(channel, buffer, sizeof(buffer), 0);
while (nbytes > 0)
{
    if (fwrite(buffer, 1, nbytes, stdout) != nbytes)
    {
        ssh_channel_close(channel);
        ssh_channel_free(channel);
        return SSH_ERROR;
    }
    nbytes = ssh_channel_read(channel, buffer, sizeof(buffer), 0);
}

if (nbytes < 0)
{
    ssh_channel_close(channel);
    ssh_channel_free(channel);
    return SSH_ERROR;
}
```

Once you read the result of the remote command, you send an end-of-file to the channel, close it, and free the memory that it used:

```
ssh_channel_send_eof(channel);
ssh_channel_close(channel);
ssh_channel_free(channel);

return SSH_OK;
}
```

2.6 Chapter 5: The SFTP subsystem

2.6.1 The SFTP subsystem

SFTP stands for "Secure File Transfer Protocol". It enables you to safely transfer files between the local and the remote computer. It reminds a lot of the old FTP protocol.

SFTP is a rich protocol. It lets you do over the network almost everything that you can do with local files:

- send files
- modify only a portion of a file
- receive files
- receive only a portion of a file
- get file owner and group
- get file permissions
- set file owner and group
- set file permissions
- remove files
- rename files
- create a directory
- remove a directory
- retrieve the list of files in a directory
- get the target of a symbolic link
- create symbolic links
- get information about mounted filesystems.

The current implemented version of the SFTP protocol is version 3. All functions aren't implemented yet, but the most important are.

2.6.1.1 Opening and closing a SFTP session

Unlike with remote shells and remote commands, when you use the SFTP subsystem, you don't handle directly the SSH channels. Instead, you open a "SFTP session".

The function `sftp_new()` creates a new SFTP session. The function `sftp_init()` initializes it. The function `sftp_free()` deletes it.

As you see, all the SFTP-related functions start with the "sftp_" prefix instead of the usual "ssh_" prefix.

The example below shows how to use these functions:

```
#include <libssh/sftp.h>

int sftp_helloworld(ssh_session session)
{
    sftp_session sftp;
    int rc;
```



```
sftp = sftp_new(session);
if (sftp == NULL)
{
    fprintf(stderr, "Error allocating SFTP session: %s\n", ssh_get_error(session)
    );
    return SSH_ERROR;
}

rc = sftp_init(sftp);
if (rc != SSH_OK)
{
    fprintf(stderr, "Error initializing SFTP session: %s.\n", sftp_get_error(sftp)
    );
    sftp_free(sftp);
    return rc;
}

...

sftp_free(sftp);
return SSH_OK;
}
```

2.6.1.2 Analyzing SFTP errors

In case of a problem, the function [sftp_get_error\(\)](#) returns a SFTP-specific error number, in addition to the regular SSH error number returned by [ssh_get_error_number\(\)](#).

Possible errors are:

- SSH_FX_OK: no error
- SSH_FX_EOF: end-of-file encountered
- SSH_FX_NO_SUCH_FILE: file does not exist
- SSH_FX_PERMISSION_DENIED: permission denied
- SSH_FX_FAILURE: generic failure
- SSH_FX_BAD_MESSAGE: garbage received from server
- SSH_FX_NO_CONNECTION: no connection has been set up
- SSH_FX_CONNECTION_LOST: there was a connection, but we lost it
- SSH_FX_OP_UNSUPPORTED: operation not supported by libssh yet
- SSH_FX_INVALID_HANDLE: invalid file handle
- SSH_FX_NO_SUCH_PATH: no such file or directory path exists
- SSH_FX_FILE_ALREADY_EXISTS: an attempt to create an already existing file or directory has been made
- SSH_FX_WRITE_PROTECT: write-protected filesystem
- SSH_FX_NO_MEDIA: no media was in remote drive

2.6.1.3 Creating a directory

The function `sftp_mkdir()` takes the "SFTP session" we just created as its first argument. It also needs the name of the file to create, and the desired permissions. The permissions are the same as for the usual `mkdir()` function. To get a comprehensive list of the available permissions, use the "man 2 stat" command. The desired permissions are combined with the remote user's mask to determine the effective permissions.

The code below creates a directory named "helloworld" in the current directory that can be read and written only by its owner:

```
#include <libssh/sftp.h>
#include <sys/stat.h>

int sftp_helloworld(ssh_session session, sftp_session sftp)
{
    int rc;

    rc = sftp_mkdir(sftp, "helloworld", S_IRWXU);
    if (rc != SSH_OK)
    {
        if (sftp_get_error(sftp) != SSH_FX_FILE_ALREADY_EXISTS)
        {
            fprintf(stderr, "Can't create directory: %s\n", ssh_get_error(session));
            return rc;
        }
    }

    ...

    return SSH_OK;
}
```

Unlike its equivalent in the SCP subsystem, this function does NOT change the current directory to the newly created subdirectory.

2.6.1.4 Copying a file to the remote computer

You handle the contents of a remote file just like you would do with a local `file:` you open the file in a given mode, move the file pointer in it, read or write data, and close the file.

The `sftp_open()` function is very similar to the regular `open()` function, excepted that it returns a file handle of type `sftp_file`. This file handle is then used by the other file manipulation functions and remains valid until you close the remote file with `sftp_close()`.

The example below creates a new file named "helloworld.txt" in the newly created "helloworld" directory. If the file already exists, it will be truncated. It then writes the famous "Hello, World!" sentence to the file, followed by a new line character. Finally, the file is closed:

```
#include <libssh/sftp.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```

int sftp_helloworld(ssh_session session, sftp_session sftp)
{
    int access_type = O_WRONLY | O_CREAT | O_TRUNC;
    sftp_file file;
    const char *helloworld = "Hello, World!\n";
    int length = strlen(helloworld);
    int rc, nwritten;

    ...

    file = sftp_open(sftp, "helloworld/helloworld.txt", access_type, S_IRWXU);
    if (file == NULL)
    {
        fprintf(stderr, "Can't open file for writing: %s\n", ssh_get_error(session));

        return SSH_ERROR;
    }

    nwritten = sftp_write(file, helloworld, length);
    if (nwritten != length)
    {
        fprintf(stderr, "Can't write data to file: %s\n", ssh_get_error(session));
        sftp_close(file);
        return SSH_ERROR;
    }

    rc = sftp_close(file);
    if (rc != SSH_OK)
    {
        fprintf(stderr, "Can't close the written file: %s\n", ssh_get_error(session));
        ;
        return rc;
    }

    return SSH_OK;
}

```

2.6.1.5 Reading a file from the remote computer

The nice thing with reading a file over the network through SFTP is that it can be done both in a synchronous way or an asynchronous way. If you read the file asynchronously, your program can do something else while it waits for the results to come.

Synchronous read is done with [sftp_read\(\)](#).

The following example prints the contents of remote file `/etc/profile`. For each 1024 bytes of information read, it waits until the end of the read operation:

```

int sftp_read_sync(ssh_session session, sftp_session sftp)
{
    int access_type;
    sftp_file file;
    char buffer[1024];
    int nbytes, rc;

    access_type = O_RDONLY;
    file = sftp_open(sftp, "/etc/profile", access_type, 0);
    if (file == NULL)
    {

```

```

        fprintf(stderr, "Can't open file for reading: %s\n", ssh_get_error(session));

        return SSH_ERROR;
    }

    nbytes = sftp_read(file, buffer, sizeof(buffer));
    while (nbytes > 0)
    {
        if (write(l, buffer, nbytes) != nbytes)
        {
            sftp_close(file);
            return SSH_ERROR;
        }
        nbytes = sftp_read(file, buffer, sizeof(buffer));
    }

    if (nbytes < 0)
    {
        fprintf(stderr, "Error while reading file: %s\n", ssh_get_error(session));
        sftp_close(file);
        return SSH_ERROR;
    }

    rc = sftp_close(file);
    if (rc != SSH_OK)
    {
        fprintf(stderr, "Can't close the read file: %s\n", ssh_get_error(session));
        return rc;
    }

    return SSH_OK;
}

```

Asynchronous read is done in two steps, first [sftp_async_read_begin\(\)](#), which returns a "request handle", and then [sftp_async_read\(\)](#), which uses that request handle. If the file has been opened in nonblocking mode, then [sftp_async_read\(\)](#) might return `SSH_AGAIN`, which means that the request hasn't completed yet and that the function should be called again later on. Otherwise, [sftp_async_read\(\)](#) waits for the data to come. To open a file in nonblocking mode, call [sftp_file_set_nonblocking\(\)](#) right after you opened it. Default is blocking mode.

The example below reads a very big file in asynchronous, nonblocking, mode. Each time the data are not ready yet, a counter is incremented.

```

int sftp_read_async(ssh_session session, sftp_session sftp)
{
    int access_type;
    sftp_file file;
    char buffer[1024];
    int async_request;
    int nbytes;
    long counter;
    int rc;

    access_type = O_RDONLY;
    file = sftp_open(sftp, "some_very_big_file", access_type, 0);
    if (file == NULL)
    {
        fprintf(stderr, "Can't open file for reading: %s\n", ssh_get_error(session));
    }
}

```

```

    return SSH_ERROR;
}
sftp_file_set_nonblocking(file);

async_request = sftp_async_read_begin(file, sizeof(buffer));
counter = 0L;
usleep(10000);
if (async_request >= 0)
    nbytes = sftp_async_read(file, buffer, sizeof(buffer), async_request);
else nbytes = -1;
while (nbytes > 0 || nbytes == SSH_AGAIN)
{
    if (nbytes > 0)
    {
        write(1, buffer, nbytes);
        async_request = sftp_async_read_begin(file, sizeof(buffer));
    }
    else counter++;
    usleep(10000);
    if (async_request >= 0)
        nbytes = sftp_async_read(file, buffer, sizeof(buffer), async_request);
    else nbytes = -1;
}

if (nbytes < 0)
{
    fprintf(stderr, "Error while reading file: %s\n", ssh_get_error(session));
    sftp_close(file);
    return SSH_ERROR;
}

printf("The counter has reached value: %ld\n", counter);

rc = sftp_close(file);
if (rc != SSH_OK)
{
    fprintf(stderr, "Can't close the read file: %s\n", ssh_get_error(session));
    return rc;
}

return SSH_OK;
}

```

2.6.1.6 Listing the contents of a directory

The functions [sftp_opendir\(\)](#), [sftp_readdir\(\)](#), [sftp_dir_eof\(\)](#), and [sftp_closedir\(\)](#) enable to list the contents of a directory. They use a new `handle_type`, "sftp_dir", which gives access to the directory being read.

In addition, [sftp_readdir\(\)](#) returns a "sftp_attributes" which is a pointer to a structure with informations about a directory entry:

- name: the name of the file or directory
- size: its size in bytes
- etc.

[sftp_readdir\(\)](#) might return NULL under two conditions:

- when the end of the directory has been met
- when an error occurred

To tell the difference, call `sftp_dir_eof()`.

The attributes must be freed with `sftp_attributes_free()` when no longer needed.

The following example reads the contents of some remote directory:

```
int sftp_list_dir(ssb_session session, sftp_session sftp)
{
    sftp_dir dir;
    sftp_attributes attributes;
    int rc;

    dir = sftp_opendir(sftp, "/var/log");
    if (!dir)
    {
        fprintf(stderr, "Directory not opened: %s\n", ssh_get_error(session));
        return SSH_ERROR;
    }

    printf("Name                               Size Perms   Owner\tGroup\n");

    while ((attributes = sftp_readdir(sftp, dir)) != NULL)
    {
        printf("%-22s %10llu %.8o %s(%d)\t%s(%d)\n",
            attributes->name,
            (long long unsigned int) attributes->size,
            attributes->permissions,
            attributes->owner,
            attributes->uid,
            attributes->group,
            attributes->gid);

        sftp_attributes_free(attributes);
    }

    if (!sftp_dir_eof(dir))
    {
        fprintf(stderr, "Can't list directory: %s\n", ssh_get_error(session));
        sftp_closedir(dir);
        return SSH_ERROR;
    }

    rc = sftp_closedir(dir);
    if (rc != SSH_OK)
    {
        fprintf(stderr, "Can't close directory: %s\n", ssh_get_error(session));
        return rc;
    }
}
```

2.7 Chapter 6: The SCP subsystem

2.7.1 The SCP subsystem

The SCP subsystem has far less functionality than the SFTP subsystem. However, if you only need to copy files from and to the remote system, it does its job.

2.7.1.1 Opening and closing a SCP session

Like in the SFTP subsystem, you don't handle the SSH channels directly. Instead, you open a "SCP session".

When you open your SCP session, you have to choose between read or write mode. You can't do both in the same session. So you specify either `SSH_SCP_READ` or `SSH_SCP_WRITE` as the second parameter of function `ssh_scp_new()`.

Another important mode flag for opening your SCP session is `SSH_SCP_RECURSIVE`. When you use `SSH_SCP_RECURSIVE`, you declare that you are willing to emulate the behaviour of "scp -r" command in your program, no matter it is for reading or for writing.

Once your session is created, you initialize it with `ssh_scp_init()`. When you have finished transferring files, you terminate the SCP connection with `ssh_scp_close()`. Finally, you can dispose the SCP connection with `ssh_scp_free()`.

The example below does the maintenance work to open a SCP connection for writing in recursive mode:

```
int scp_write(ssh_session session)
{
    ssh_scp scp;
    int rc;

    scp = ssh_scp_new
        (session, SSH_SCP_WRITE | SSH_SCP_RECURSIVE, ".");
    if (scp == NULL)
    {
        fprintf(stderr, "Error allocating scp session: %s\n", ssh_get_error(session));
        return SSH_ERROR;
    }

    rc = ssh_scp_init(scp);
    if (rc != SSH_OK)
    {
        fprintf(stderr, "Error initializing scp session: %s\n", ssh_get_error(session));
        ssh_scp_free(scp);
        return rc;
    }

    ...

    ssh_scp_close(scp);
    ssh_scp_free(scp);
    return SSH_OK;
}
```

```
}
```

The example below shows how to open a connection to read a single [file](#):

```
int scp_read(ssh_session session)
{
    ssh_scp scp;
    int rc;

    scp = ssh_scp_new
        (session, SSH_SCP_READ, "helloworld/helloworld.txt");
    if (scp == NULL)
    {
        fprintf(stderr, "Error allocating scp session: %s\n", ssh_get_error(session));
        return SSH_ERROR;
    }

    rc = ssh_scp_init(scp);
    if (rc != SSH_OK)
    {
        fprintf(stderr, "Error initializing scp session: %s\n", ssh_get_error(session));
        ssh_scp_free(scp);
        return rc;
    }

    ...

    ssh_scp_close(scp);
    ssh_scp_free(scp);
    return SSH_OK;
}
```

2.7.1.2 Creating files and directories

You create directories with [ssh_scp_push_directory\(\)](#). In recursive mode, you are placed in this directory once it is created. If the directory already exists and if you are in recursive mode, you simply enter that directory.

Creating files is done in two steps. First, you prepare the writing with [ssh_scp_push_file\(\)](#). Then, you write the data with [ssh_scp_write\(\)](#). The length of the data to write must be identical between both function calls. There's no need to "open" nor "close" the file, this is done automatically on the remote end. If the file already exists, it is overwritten and truncated.

The following example creates a new directory named "helloworld/", then creates a file named "helloworld.txt" in that directory:

```
int scp_helloworld(ssh_session session, ssh_scp scp)
{
    int rc;
    const char *helloworld = "Hello, world!\n";
    int length = strlen(helloworld);

    rc = ssh_scp_push_directory(scp, "helloworld", S_IRWXU);
    if (rc != SSH_OK)
```



```

    {
        fprintf(stderr, "Can't create remote directory: %s\n", ssh_get_error(session)
        );
        return rc;
    }

    rc = ssh_scp_push_file
        (scp, "helloworld.txt", length, S_IRUSR | S_IWUSR);
    if (rc != SSH_OK)
    {
        fprintf(stderr, "Can't open remote file: %s\n", ssh_get_error(session));
        return rc;
    }

    rc = ssh_scp_write(scp, helloworld, length);
    if (rc != SSH_OK)
    {
        fprintf(stderr, "Can't write to remote file: %s\n", ssh_get_error(session));
        return rc;
    }

    return SSH_OK;
}

```

2.7.1.3 Copying full directory trees to the remote server

Let's say you want to copy the following tree of files to the remote site:

```

                +-- file1
            +-- B ---+
            |         +-- file2
-- A ---+
    |         +-- file3
    +-- C ---+
        +-- file4

```

You would do it that way:

- open the session in recursive mode
- enter directory A
- enter its subdirectory B
- create file1 in B
- create file2 in B
- leave directory B
- enter subdirectory C
- create file3 in C
- create file4 in C
- leave directory C

- leave directory A

To leave a directory, call `ssh_scp_leave_directory()`.

2.7.1.4 Reading files and directories

To receive files, you pull requests from the other side with `ssh_scp_pull_request()`. If this function returns `SSH_SCP_REQUEST_NEWFILE`, then you must get ready for the reception. You can get the size of the data to receive with `ssh_scp_request_get_size()` and allocate a buffer accordingly. When you are ready, you accept the request with `ssh_scp_accept_request()`, then read the data with `ssh_scp_read()`.

The following example receives a single file. The name of the file to receive has been given earlier, when the scp session was opened:

```
int scp_receive(ssh_session session, ssh_scp scp)
{
    int rc;
    int size, mode;
    char *filename, *buffer;

    rc = ssh_scp_pull_request(scp);
    if (rc != SSH_SCP_REQUEST_NEWFILE)
    {
        fprintf(stderr, "Error receiving information about file: %s\n",
            ssh_get_error(session));
        return SSH_ERROR;
    }

    size = ssh_scp_request_get_size(scp);
    filename = strdup(ssh_scp_request_get_filename(scp));
    mode = ssh_scp_request_get_permissions(scp);
    printf("Receiving file %s, size %d, permissions %0o\n", filename, size, mode);

    free(filename);

    buffer = malloc(size);
    if (buffer == NULL)
    {
        fprintf(stderr, "Memory allocation error\n");
        return SSH_ERROR;
    }

    ssh_scp_accept_request(scp);
    rc = ssh_scp_read(scp, buffer, size);
    if (rc == SSH_ERROR)
    {
        fprintf(stderr, "Error receiving file data: %s\n", ssh_get_error(session));
        free(buffer);
        return rc;
    }
    printf("Done\n");

    write(1, buffer, size);
    free(buffer);

    rc = ssh_scp_pull_request(scp);
    if (rc != SSH_SCP_REQUEST_EOF)
```

```

    {
        fprintf(stderr, "Unexpected request: %s\n", ssh_get_error(session));
        return SSH_ERROR;
    }

    return SSH_OK;
}

```

In this example, since we just requested a single file, we expect `ssh_scp_request()` to return `SSH_SCP_REQUEST_NEWFILE` first, then `SSH_SCP_REQUEST_EOF`. That's quite a naive approach; for example, the remote server might send a warning as well (return code `SSH_SCP_REQUEST_WARNING`) and the example would fail. A more comprehensive reception program would receive the requests in a loop and analyze them carefully until `SSH_SCP_REQUEST_EOF` has been received.

2.7.1.5 Receiving full directory trees from the remote server

If you opened the SCP session in recursive mode, the remote end will be telling you when to change directory.

In that case, when `ssh_scp_pull_request()` answers `SSH_SCP_REQUEST_NEWDIRECTORY`, you should make that local directory (if it does not exist yet) and enter it. When `ssh_scp_pull_request()` answers `SSH_SCP_REQUEST_ENDDIRECTORY`, you should leave the current directory.

2.8 Chapter 7: Forwarding connections (tunnel)

2.8.1 Forwarding connections

Port forwarding comes in SSH protocol in two different flavours: direct or reverse port forwarding. Direct port forwarding is also named local port forwarding, and reverse port forwarding is also called remote port forwarding. SSH also allows X11 tunnels.

2.8.1.1 Direct port forwarding

Direct port forwarding is from client to server. The client opens a tunnel, and forwards whatever data to the server. Then, the server connects to an end point. The end point can reside on another machine or on the SSH server itself.

Example of use of direct port forwarding:

```

Mail client application   Google Mail
      |                   ^
      | 5555 (arbitrary)   |
      |                   |
      | 143 (IMAP2)        |
      |                   |
SSH client  =====>  SSH server

```

Legend:

```

--P-->: port connexion through port P
=====>: SSH tunnel

```


- you need a separate channel for the tunnel as first parameter;
- second and third parameters are the remote endpoint;
- fourth and fifth parameters are sent to the remote server so that they can be logged on that server.

If you don't plan to forward the data you will receive to any local port, just put fake values like "localhost" and 5555 as your local host and port.

The example below shows how to open a direct channel that would be used to retrieve google's home page from the remote SSH server.

```
int direct_forwarding(ssh_session session)
{
    ssh_channel forwarding_channel;
    int rc;
    char *http_get = "GET / HTTP/1.1\nHost: www.google.com\n\n";
    int nbytes, nwritten;

    forwarding_channel = ssh_channel_new(session);
    if (rc != SSH_OK) return rc;

    rc = channel_open_forward(forwarding_channel,
                              "www.google.com", 80,
                              "localhost", 5555);

    if (rc != SSH_OK)
    {
        ssh_channel_free(forwarding_channel);
        return rc;
    }

    nbytes = strlen(http_get);
    nwritten = channel_write(forwarding_channel, http_get, nbytes);
    if (nbytes != nwritten)
    {
        ssh_channel_free(forwarding_channel);
        return SSH_ERROR;
    }

    ...

    ssh_channel_free(forwarding_channel);
    return SSH_OK;
}
```

The data sent by Google can be retrieved for example with [ssh_select\(\)](#) and [ssh_channel_read\(\)](#). Google's home page can then be displayed on the local SSH client, saved into a local file, made available on a local port, or whatever use you have for it.

2.8.1.5 Doing reverse port forwarding with libssh

To do reverse port forwarding, call `ssh_channel_forward_listen()`, then `ssh_channel_forward_accept()`.

When you call `ssh_channel_forward_listen()`, you can let the remote server chose the non-privileged port it should listen to. Otherwise, you can chose your own privileged

or non-privileged port. Beware that you should have administrative privileges on the remote server to open a privileged port (port number < 1024).

Below is an example of a very rough web server waiting for connections on port 8080 of remote SSH server. The incoming connections are passed to the local libssh application, which handles them:

```
int web_server(ssh_session session)
{
    int rc;
    ssh_channel channel;
    char buffer[256];
    int nbytes, nwritten;
    char *helloworld = "
HTTP/1.1 200 OK\n"
"Content-Type: text/html\n"
"Content-Length: 113\n"
"\n"
"<html>\n"
"  <head>\n"
"    <title>Hello, World!</title>\n"
"  </head>\n"
"  <body>\n"
"    <h1>Hello, World!</h1>\n"
"  </body>\n"
"</html>\n";

    rc = ssh_channel_forward_listen(session, NULL, 8080, NULL);
    if (rc != SSH_OK)
    {
        fprintf(stderr, "Error opening remote port: %s\n", ssh_get_error(session));
        return rc;
    }

    channel = ssh_channel_forward_accept(session, 60000);
    if (channel == NULL)
    {
        fprintf(stderr, "Error waiting for incoming connection: %s\n", ssh_get_error(
            session));
        return SSH_ERROR;
    }

    while (1)
    {
        nbytes = ssh_channel_read(channel, buffer, sizeof(buffer), 0);
        if (nbytes < 0)
        {
            fprintf(stderr, "Error reading incoming data: %s\n", ssh_get_error(session)
            );
            ssh_channel_send_eof(channel);
            ssh_channel_free(channel);
            return SSH_ERROR;
        }
        if (strncmp(buffer, "GET /", 5)) continue;

        nbytes = strlen(helloworld);
        nwritten = ssh_channel_write(channel, helloworld, nbytes);
        if (nwritten != nbytes)
        {
            fprintf(stderr, "Error sending answer: %s\n", ssh_get_error(session));
            ssh_channel_send_eof(channel);
            ssh_channel_free(channel);
        }
    }
}
```

```
        return SSH_ERROR;
    }
    printf("Sent answer\n");
}

ssh_channel_send_eof(channel);
ssh_channel_free(channel);
return SSH_OK;
}
```

2.9 Chapter 8: Threads with libssh

2.9.1 How to use libssh with threads

libssh may be used in multithreaded applications, but under several conditions :

- Threading must be initialized during the initialization of libssh. This initialization must be done outside of any threading context.
- If pthreads is being used by your application (or your framework's backend), you must link with libssh_threads_pthread dynamic library and initialize threading with the ssh_threads_pthreads threading object.
- If an other threading library is being used by your application, you must implement all the methods of the ssh_threads_callbacks_struct structure and initialize libssh with it.
- At all times, you may use different sessions inside threads, make parallel connections, read/write on different sessions and so on. You can use a single session in several channels at the same time. This will lead to internal state corruption. This limitation is being worked out and will maybe disappear later.

2.9.1.1 Initialization of threads

To initialize threading, you must first select the threading model you want to use, using [ssh_threads_set_callbacks\(\)](#), then call [ssh_init\(\)](#).

```
#include <libssh/callbacks.h>
...
ssh_threads_set_callbacks(ssh_threads_noop);
ssh_init();
```

ssh_threads_noop is the threading structure that does nothing. It's the threading callbacks being used by default when you're not using threading.

2.9.1.2 Using libpthread with libssh

If your application is using libpthread, you may simply use the libpthread threading backend:

```
#include <libssh/callbacks.h>
...
ssh_threads_set_callbacks(ssh_threads_pthread);
ssh_init();
```

However, you must be sure to link with the library `ssh_threads_pthread`. If you're using `gcc`, you must use the commandline

```
gcc -o output input.c -lssh -lssh_threads_pthread
```

2.9.1.3 Using another threading library

You must find your way in the `ssh_threads_callbacks_struct` structure. You must implement the following methods :

- `mutex_lock`
- `mutex_unlock`
- `mutex_init`
- `mutex_destroy`
- `thread_id`

Good luck !

2.10 To be done

To be written ***

2.10.1 Writing a libssh-based server

To be written ***

2.10.2 The libssh C++ wrapper

To be written ***

Chapter 3

The Linking HowTo

3.1 Dynamic Linking

On UNIX and Windows systems its the same, you need at least the [libssh.h](#) header file and the libssh shared library.

3.2 Static Linking

Warning

The libssh library is licensed under the LGPL! Make sure you understand what this means to your codebase if you want to distribute binaries and link statically against LGPL code!

On UNIX systems linking against the static version of the library is the same as linking against the shared library. Both have the same name. Some build system require to use the full path to the static library.

On Windows you need to define LIBSSH_STATIC in the compiler command line. This is required cause the dynamic library needs to specify the dllimport attribute.

Chapter 4

Deprecated List

Global [channel_read_buffer](#)(ssh_channel channel, ssh_buffer buffer, uint32_t count, int is_stderr)
Please use ssh_channel_read instead

Global [ssh_auth_list](#)(ssh_session session)

Chapter 5

Bug List

Global [ssh_channel_set_blocking](#)(ssh_channel channel, int blocking) This functionality is still under development and doesn't work correctly.

Global [ssh_is_server_known](#)(ssh_session session) There is no current way to remove or modify an entry into the known host table.

Global [ssh_set_blocking](#)(ssh_session session, int blocking) nonblocking code is in development and won't work as expected

Chapter 6

Module Index

6.1 Modules

Here is a list of all modules:

The libssh C++ wrapper	64
The libssh server API	65
The libssh SFTP API	71
The libssh API	118
The libssh callbacks	57
The SSH authentication functions.	87
The SSH buffer functions.	100
The SSH channel functions	102
The SSH error functions.	117
The SSH logging functions.	120
The SSH message functions	121
The SSH helper functions.	122
The SSH Public Key Infrastructure	126
The SSH poll functions.	128
The SSH scp functions	134
The SSH session functions.	140
The SSH string functions	154
The SSH threading functions.	158

Chapter 7

Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

ssh::Channel (Ssh::Channel class describes the state of an SSH channel) . . .	161
ssh::Session (The ssh::Session class contains the state of a SSH connection) .	164
ssh_bind_callbacks (These are the callbacks exported by the ssh_bind structure)	173
ssh_callbacks (The structure to replace libssh functions with appropriate callbacks)	174
ssh_socket_callbacks (These are the callbacks exported by the socket structure They are called by the socket module when a socket event appears)	175
ssh::SshException (Some people do not like C++ exceptions)	176

Chapter 8

File Index

8.1 File List

Here is a list of all documented files with brief descriptions:

<code>include/libssh/agent.h</code>	??
<code>include/libssh/auth.h</code>	??
<code>include/libssh/bind.h</code>	??
<code>include/libssh/buffer.h</code>	??
<code>include/libssh/callbacks.h</code>	??
<code>include/libssh/channels.h</code>	??
<code>include/libssh/crypto.h</code>	??
<code>include/libssh/dh.h</code>	??
<code>include/libssh/kex.h</code>	??
<code>include/libssh/keyfiles.h</code>	??
<code>include/libssh/keys.h</code>	??
<code>include/libssh/legacy.h</code>	??
<code>include/libssh/libcrypto.h</code>	??
<code>include/libssh/libgrypt.h</code>	??
<code>include/libssh/libssh.h</code>	??
<code>include/libssh/libsshpp.hpp</code>	??
<code>include/libssh/messages.h</code>	??
<code>include/libssh/misc.h</code>	??
<code>include/libssh/packet.h</code>	??
<code>include/libssh/pcap.h</code>	??
<code>include/libssh/pki.h</code>	??
<code>include/libssh/poll.h</code>	??
<code>include/libssh/priv.h</code>	??
<code>include/libssh/scp.h</code>	??
<code>include/libssh/server.h</code>	??
<code>include/libssh/session.h</code>	??
<code>include/libssh/sftp.h</code> (SFTP handling functions)	179
<code>include/libssh/socket.h</code>	??
<code>include/libssh/ssh1.h</code>	??

include/libssh/ ssh2.h	??
include/libssh/ string.h	??
include/libssh/ threads.h	??
include/libssh/ wrapper.h	??

Chapter 9

Module Documentation

9.1 The libssh callbacks

Callback which can be replaced in libssh.

Data Structures

- struct [ssh_callbacks](#)

The structure to replace libssh functions with appropriate callbacks.

- struct [ssh_socket_callbacks](#)

These are the callbacks exported by the socket structure They are called by the socket module when a socket event appears.

Defines

- #define [ssh_callbacks_init\(p\)](#)

Initializes an [ssh_callbacks_struct](#) A call to this macro is mandatory when you have set a new [ssh_callback_struct](#) structure.

- #define [SSH_PACKET_CALLBACK\(name\)](#) int name (ssh_session session, uint8_t type, ssh_buffer packet, void *user)

This macro declares a packet callback handler.

- #define [SSH_PACKET_NOT_USED](#) 2

Packet was not used and should be passed to any other callback available.

- #define [SSH_PACKET_USED](#) 1

return values for a [ssh_packet_callback](#)

Typedefs

- typedef int(* [ssh_auth_callback](#))(const char *prompt, char *buf, size_t len, int echo, int verify, void *userdata)
SSH authentication callback.
- typedef void(* [ssh_channel_close_callback](#))(ssh_session session, ssh_channel channel, void *userdata)
SSH channel close callback.
- typedef int(* [ssh_channel_data_callback](#))(ssh_session session, ssh_channel channel, void *data, uint32_t len, int is_stderr, void *userdata)
SSH channel data callback.
- typedef void(* [ssh_channel_eof_callback](#))(ssh_session session, ssh_channel channel, void *userdata)
SSH channel eof callback.
- typedef void(* [ssh_channel_exit_signal_callback](#))(ssh_session session, ssh_channel channel, const char *signal, int core, const char *errmsg, const char *lang, void *userdata)
SSH channel exit signal callback.
- typedef void(* [ssh_channel_exit_status_callback](#))(ssh_session session, ssh_channel channel, int exit_status, void *userdata)
SSH channel exit status callback.
- typedef void(* [ssh_channel_signal_callback](#))(ssh_session session, ssh_channel channel, const char *signal, void *userdata)
SSH channel signal callback.
- typedef void(* [ssh_global_request_callback](#))(ssh_session session, ssh_message message, void *userdata)
SSH global request callback.
- typedef void(* [ssh_log_callback](#))(ssh_session session, int priority, const char *message, void *userdata)
SSH log callback.
- typedef int(* [ssh_packet_callback](#))(ssh_session session, uint8_t type, ssh_buffer packet, void *user)
Prototype for a packet callback, to be called when a new packet arrives.
- typedef void(* [ssh_status_callback](#))(ssh_session session, float status, void *userdata)
SSH Connection status callback.

Functions

- LIBSSH_API int [ssh_set_callbacks](#) (ssh_session session, ssh_callbacks cb)
Set the session callback functions.
- LIBSSH_API int [ssh_set_channel_callbacks](#) (ssh_channel channel, ssh_channel_callbacks cb)
Set the channel callback functions.

9.1.1 Detailed Description

Callback which can be replaced in libssh.

9.1.2 Define Documentation

9.1.2.1 `#define ssh_callbacks_init(p)`

Value:

```
do { \
    (p)->size=sizeof(* (p)); \
} while (0);
```

Initializes an [ssh_callbacks_struct](#) A call to this macro is mandatory when you have set a new `ssh_callback_struct` structure.

Its goal is to maintain the binary compatibility with future versions of libssh as the structure evolves with time.

9.1.2.2 `#define SSH_PACKET_CALLBACK(name) int name (ssh_session session, uint8_t type, ssh_buffer packet, void *user)`

This macro declares a packet callback handler.

```
SSH_PACKET_CALLBACK(mycallback) {
    ...
}
```

9.1.2.3 `#define SSH_PACKET_USED 1`

return values for a `ssh_packet_callback`

Packet was used and should not be parsed by another callback

9.1.3 Typedef Documentation

9.1.3.1 `typedef int(* ssh_auth_callback)(const char *prompt, char *buf, size_t len, int echo, int verify, void *userdata)`

SSH authentication callback.

Parameters

<i>prompt</i>	Prompt to be displayed.
<i>buf</i>	Buffer to save the password. You should null-terminate it.
<i>len</i>	Length of the buffer.
<i>echo</i>	Enable or disable the echo of what you type.
<i>verify</i>	Should the password be verified?
<i>userdata</i>	Userdata to be passed to the callback function. Useful for GUI applications.

Returns

0 on success, < 0 on error.

9.1.3.2 `typedef void(* ssh_channel_close_callback)(ssh_session session, ssh_channel channel, void *userdata)`

SSH channel close callback.

Called when a channel is closed by remote peer

Parameters

<i>session</i>	Current session handler
<i>channel</i>	the actual channel
<i>userdata</i>	Userdata to be passed to the callback function.

9.1.3.3 `typedef int(* ssh_channel_data_callback)(ssh_session session, ssh_channel channel, void *data, uint32_t len, int is_stderr, void *userdata)`

SSH channel data callback.

Called when data is available on a channel

Parameters

<i>session</i>	Current session handler
<i>channel</i>	the actual channel
<i>data</i>	the data that has been read on the channel
<i>len</i>	the length of the data
<i>is_stderr</i>	is 0 for stdout or 1 for stderr
<i>userdata</i>	Userdata to be passed to the callback function.

9.1.3.4 `typedef void(* ssh_channel_eof_callback)(ssh_session session, ssh_channel channel, void *userdata)`

SSH channel eof callback.

Called when a channel receives EOF

Parameters

<i>session</i>	Current session handler
<i>channel</i>	the actual channel
<i>userdata</i>	Userdata to be passed to the callback function.

9.1.3.5 `typedef void(* ssh_channel_exit_signal_callback)(ssh_session session, ssh_channel channel, const char *signal, int core, const char *errmsg, const char *lang, void *userdata)`

SSH channel exit signal callback.

Called when a channel has received an exit signal

Parameters

<i>session</i>	Current session handler
<i>channel</i>	the actual channel
<i>signal</i>	the signal name (without the SIG prefix)
<i>core</i>	a boolean telling whether a core has been dumped or not
<i>errmsg</i>	the description of the exception
<i>lang</i>	the language of the description (format: RFC 3066)
<i>userdata</i>	Userdata to be passed to the callback function.

9.1.3.6 `typedef void(* ssh_channel_exit_status_callback)(ssh_session session, ssh_channel channel, int exit_status, void *userdata)`

SSH channel exit status callback.

Called when a channel has received an exit status

Parameters

<i>session</i>	Current session handler
<i>channel</i>	the actual channel
<i>userdata</i>	Userdata to be passed to the callback function.

9.1.3.7 `typedef void(* ssh_channel_signal_callback)(ssh_session session, ssh_channel channel, const char *signal, void *userdata)`

SSH channel signal callback.

Called when a channel has received a signal

Parameters

<i>session</i>	Current session handler
<i>channel</i>	the actual channel
<i>signal</i>	the signal name (without the SIG prefix)
<i>userdata</i>	Userdata to be passed to the callback function.

9.1.3.8 `typedef void(* ssh_global_request_callback)(ssh_session session, ssh_message message, void *userdata)`

SSH global request callback.

All global request will go through this callback.

Parameters

<i>session</i>	Current session handler
<i>message</i>	the actual message
<i>userdata</i>	Userdata to be passed to the callback function.

9.1.3.9 `typedef void(* ssh_log_callback)(ssh_session session, int priority, const char *message, void *userdata)`

SSH log callback.

All logging messages will go through this callback

Parameters

<i>session</i>	Current session handler
<i>priority</i>	Priority of the log, the smaller being the more important
<i>message</i>	the actual message
<i>userdata</i>	Userdata to be passed to the callback function.

9.1.3.10 `typedef int(* ssh_packet_callback)(ssh_session session, uint8_t type, ssh_buffer packet, void *user)`

Prototype for a packet callback, to be called when a new packet arrives.

Parameters

<i>session</i>	The current session of the packet
<i>type</i>	packet type (see ssh2.h)
<i>packet</i>	buffer containing the packet, excluding size, type and padding fields
<i>user</i>	user argument to the callback and are called each time a packet shows up

Returns

SSH_PACKET_USED Packet was parsed and used

SSH_PACKET_NOT_USED Packet was not used or understood, processing must continue

9.1.3.11 `typedef void(* ssh_status_callback)(ssh_session session, float status, void *userdata)`

SSH Connection status callback.

Parameters

<i>session</i>	Current session handler
<i>status</i>	Percentage of connection status, going from 0.0 to 1.0 once connection is done.
<i>userdata</i>	Userdata to be passed to the callback function.

9.1.4 Function Documentation

9.1.4.1 `LIBSSH_API int ssh_set_callbacks (ssh_session session, ssh_callbacks cb)`

Set the session callback functions.

This functions sets the callback structure to use your own callback functions for auth, logging and status.

```
struct ssh_callbacks_struct cb = {
    .userdata = data,
    .auth_function = my_auth_function
};
ssh_callbacks_init(&cb);
ssh_set_callbacks(session, &cb);
```

Parameters

<i>session</i>	The session to set the callback structure.
<i>cb</i>	The callback structure itself.

Returns

SSH_OK on success, SSH_ERROR on error.

9.1.4.2 `LIBSSH_API int ssh_set_channel_callbacks (ssh_channel channel, ssh_channel_callbacks cb)`

Set the channel callback functions.

This functions sets the callback structure to use your own callback functions for channel data and exceptions

```

struct ssh_channel_callbacks_struct cb = {
    .userdata = data,
    .channel_data = my_channel_data_function
};
ssh_callbacks_init(&cb);
ssh_set_channel_callbacks(channel, &cb);

```

Parameters

<i>channel</i>	The channel to set the callback structure.
<i>cb</i>	The callback structure itself.

Returns

SSH_OK on success, SSH_ERROR on error.

9.2 The libssh C++ wrapper

The C++ bindings for libssh are completely embedded in a single .hpp file, and this for two reasons:

- C++ is hard to keep binary compatible, C is easy.

9.2.1 Detailed Description

The C++ bindings for libssh are completely embedded in a single .hpp file, and this for two reasons:

- C++ is hard to keep binary compatible, C is easy.

We try to keep libssh C version as much as possible binary compatible between releases, while this would be hard for C++. If you compile your program with these headers, you will only link to the C version of libssh which will be kept ABI compatible. No need to recompile your C++ program each time a new binary-compatible version of libssh is out

- Most of the functions in this file are really short and are probably worth the "inline" linking mode, which the compiler can decide to do in some case. There would be nearly no performance penalty of using the wrapper rather than native calls.

Please visit the documentation of [ssh::Session](#) and [ssh::Channel](#)

See also

[ssh::Session](#)
[ssh::Channel](#)

If you wish not to use C++ exceptions, please define SSH_NO_CPP_EXCEPTIONS:

```
#define SSH_NO_CPP_EXCEPTIONS
#include <libssh/libsshpp.hpp>
```

All functions will then return SSH_ERROR in case of error.

9.3 The libssh server API

Data Structures

- struct [ssh_bind_callbacks](#)

These are the callbacks exported by the ssh_bind structure.

Typedefs

- typedef void(* [ssh_bind_incoming_connection_callback](#))(ssh_bind sshbind, void *userdata)

Incoming connection callback.

Functions

- LIBSSH_API int [ssh_bind_accept](#) (ssh_bind ssh_bind_o, ssh_session session)
Accept an incoming ssh connection and initialize the session.
- LIBSSH_API void [ssh_bind_fd_toaccept](#) (ssh_bind ssh_bind_o)
Allow the file descriptor to accept new sessions.
- LIBSSH_API void [ssh_bind_free](#) (ssh_bind ssh_bind_o)
Free a ssh servers bind.
- LIBSSH_API socket_t [ssh_bind_get_fd](#) (ssh_bind ssh_bind_o)
Recover the file descriptor from the session.
- LIBSSH_API int [ssh_bind_listen](#) (ssh_bind ssh_bind_o)
Start listening to the socket.
- LIBSSH_API ssh_bind [ssh_bind_new](#) (void)
Creates a new SSH server bind.
- LIBSSH_API int [ssh_bind_options_set](#) (ssh_bind sshbind, enum ssh_bind_options_e type, const void *value)
Set the options for the current SSH server bind.
- LIBSSH_API void [ssh_bind_set_blocking](#) (ssh_bind ssh_bind_o, int blocking)

Set the session to blocking/nonblocking mode.

- LIBSSH_API int [ssh_bind_set_callbacks](#) (ssh_bind sshbind, ssh_bind_callbacks callbacks, void *userdata)

Set the callback for this bind.

- LIBSSH_API void [ssh_bind_set_fd](#) (ssh_bind ssh_bind_o, socket_t fd)

Set the file descriptor for a session.

- LIBSSH_API int [ssh_handle_key_exchange](#) (ssh_session session)

Handles the key exchange and set up encryption.

- LIBSSH_API void [ssh_set_message_callback](#) (ssh_session session, int(*ssh_bind_message_callback)(ssh_session session, ssh_message msg, void *data), void *data)

defines the ssh_message callback

9.3.1 Typedef Documentation

- 9.3.1.1 **typedef void(* ssh_bind_incoming_connection_callback)(ssh_bind sshbind, void *userdata)**

Incoming connection callback.

This callback is called when a ssh_bind has a new incoming connection.

Parameters

<i>sshbind</i>	Current sshbind session handler
<i>message</i>	the actual message
<i>userdata</i>	Userdata to be passed to the callback function.

9.3.2 Function Documentation

- 9.3.2.1 **int ssh_bind_accept (ssh_bind ssh_bind_o, ssh_session session)**

Accept an incoming ssh connection and initialize the session.

Parameters

<i>ssh_bind_o</i>	The ssh server bind to accept a connection.
<i>session</i>	A preallocated ssh session

See also

[ssh_new](#)

Returns

SSH_OK when a connection is established

References privatekey_free().

9.3.2.2 void ssh_bind_fd_toaccept (ssh_bind *ssh_bind_o*)

Allow the file descriptor to accept new sessions.

Parameters

<i>ssh_bind_o</i>	The ssh server bind to use.
-------------------	-----------------------------

9.3.2.3 void ssh_bind_free (ssh_bind *ssh_bind_o*)

Free a ssh servers bind.

Parameters

<i>ssh_bind_o</i>	The ssh server bind to free.
-------------------	------------------------------

9.3.2.4 socket_t ssh_bind_get_fd (ssh_bind *ssh_bind_o*)

Recover the file descriptor from the session.

Parameters

<i>ssh_bind_o</i>	The ssh server bind to get the fd from.
-------------------	---

Returns

The file descriptor.

9.3.2.5 int ssh_bind_listen (ssh_bind *ssh_bind_o*)

Start listening to the socket.

Parameters

<i>ssh_bind_o</i>	The ssh server bind to use.
-------------------	-----------------------------

Returns

0 on success, < 0 on error.

References ssh_init().

9.3.2.6 `ssh_bind ssh_bind_new (void)`

Creates a new SSH server bind.

Returns

A newly allocated `ssh_bind` session pointer.

9.3.2.7 `LIBSSH_API int ssh_bind_options_set (ssh_bind sshbind, enum ssh_bind_options_e type, const void * value)`

Set the options for the current SSH server bind.

Parameters

<i>sshbind</i>	The ssh server bind to configure.
<i>type</i>	The option type to set. This could be one of the following:

- `SSH_BIND_OPTIONS_BINDADDR` The ip address to bind (const char *).
- `SSH_BIND_OPTIONS_BINDPORT` The port to bind (unsigned int).
- `SSH_BIND_OPTIONS_BINDPORT_STR` The port to bind (const char *).
- `SSH_BIND_OPTIONS_HOSTKEY` This specifies the file containing the private host key used by SSHv1. (const char *).
- `SSH_BIND_OPTIONS_DSAKEY` This specifies the file containing the private host dsa key used by SSHv2. (const char *).
- `SSH_BIND_OPTIONS_RSAKEY` This specifies the file containing the private host dsa key used by SSHv2. (const char *).
- `SSH_BIND_OPTIONS_BANNER` That the server banner (version string) for SSH. (const char *).
- `SSH_BIND_OPTIONS_LOG_VERBOSITY` Set the session logging verbosity (int).

The verbosity of the messages. Every log smaller or equal to verbosity will be shown.

- `SSH_LOG_NOLOG`: No logging
- `SSH_LOG_RARE`: Rare conditions or warnings
- `SSH_LOG_ENTRY`: API-accessible entrypoints
- `SSH_LOG_PACKET`: Packet id and size
- `SSH_LOG_FUNCTIONS`: Function entering and leaving

- `SSH_BIND_OPTIONS_LOG_VERBOSITY_STR` Set the session logging verbosity (const char *).

The verbosity of the messages. Every log smaller or equal to verbosity will be shown.

- `SSH_LOG_NOLOG`: No logging
 - `SSH_LOG_RARE`: Rare conditions or warnings
 - `SSH_LOG_ENTRY`: API-accessible entrypoints
 - `SSH_LOG_PACKET`: Packet id and size
 - `SSH_LOG_FUNCTIONS`: Function entering and leaving
- See the corresponding numbers in [libssh.h](#).

Parameters

<i>value</i>	The value to set. This is a generic pointer and the datatype which is used should be set according to the type set.
--------------	---

Returns

`SSH_OK` on success, `SSH_ERROR` on invalid option or parameter.

9.3.2.8 void ssh_bind_set_blocking (ssh_bind *ssh_bind_o*, int *blocking*)

Set the session to blocking/nonblocking mode.

Parameters

<i>ssh_bind_o</i>	The ssh server bind to use.
<i>blocking</i>	Zero for nonblocking mode.

9.3.2.9 int ssh_bind_set_callbacks (ssh_bind *sshbind*, ssh_bind_callbacks *callbacks*, void * *userdata*)

Set the callback for this bind.

Parameters

in	<i>sshbind</i>	The bind to set the callback on.
in	<i>callbacks</i>	An already set up <code>ssh_bind_callbacks</code> instance.
in	<i>userdata</i>	A pointer to private data to pass to the callbacks.

Returns

`SSH_OK` on success, `SSH_ERROR` if an error occurred.

```
struct ssh_callbacks_struct cb = {
    .userdata = data,
    .auth_function = my_auth_function
}
```

```
};
ssh_callbacks_init(&cb);
ssh_bind_set_callbacks(session, &cb);
```

9.3.2.10 void ssh_bind_set_fd (ssh_bind *ssh_bind_o*, socket_t *fd*)

Set the file descriptor for a session.

Parameters

<i>ssh_bind_o</i>	The ssh server bind to set the fd.
<i>fd</i>	The file descriptssh_bind B

9.3.2.11 int ssh_handle_key_exchange (ssh_session *session*)

Handles the key exchange and set up encryption.

Parameters

<i>session</i>	A connected ssh session
----------------	-------------------------

See also

[ssh_bind_accept](#)

Returns

SSH_OK if the key exchange was successful

References ssh_log(), and SSH_LOG_PACKET.

9.3.2.12 void ssh_set_message_callback (ssh_session *session*, int(*)(*ssh_session session*, *ssh_message msg*, void **data*) *ssh_bind_message_callback*, void * *data*)

defines the ssh_message callback

Parameters

	<i>session</i>	the current ssh session
in	<i>ssh_bind_message_callback</i>	a function pointer to a callback taking the current ssh session and received message as parameters. the function returns 0 if the message has been parsed and treated successfully, 1 otherwise (libssh must take care of the response).
in	<i>data</i>	void pointer to be passed to callback functions

9.4 The libssh SFTP API

Functions

- int [sftp_async_read](#) (sftp_file file, void *data, uint32_t len, uint32_t id)
Wait for an asynchronous read to complete and save the data.
- int [sftp_async_read_begin](#) (sftp_file file, uint32_t len)
Start an asynchronous read from a file using an opened sftp file handle.
- void [sftp_attributes_free](#) (sftp_attributes file)
Free a sftp attribute structure.
- char * [sftp_canonicalize_path](#) (sftp_session sftp, const char *path)
Canonicalize a sftp path.
- int [sftp_chmod](#) (sftp_session sftp, const char *file, mode_t mode)
Change permissions of a file.
- int [sftp_chown](#) (sftp_session sftp, const char *file, uid_t owner, gid_t group)
Change the file owner and group.
- int [sftp_close](#) (sftp_file file)
Close an open file handle.
- int [sftp_closedir](#) (sftp_dir dir)
Close a directory handle opened by [sftp_opendir\(\)](#).
- int [sftp_dir_eof](#) (sftp_dir dir)
Tell if the directory has reached EOF (End Of File).
- int [sftp_extension_supported](#) (sftp_session sftp, const char *name, const char *data)
Check if the given extension is supported.
- unsigned int [sftp_extensions_get_count](#) (sftp_session sftp)
Get the count of extensions provided by the server.
- const char * [sftp_extensions_get_data](#) (sftp_session sftp, unsigned int indexn)
Get the data of the extension provided by the server.
- const char * [sftp_extensions_get_name](#) (sftp_session sftp, unsigned int indexn)
Get the name of the extension provided by the server.
- void [sftp_free](#) (sftp_session sftp)
Close and deallocate a sftp session.

- sftp_attributes [sftp_fstat](#) (sftp_file file)
Get information about a file or directory from a file handle.
- sftp_statvfs_t [sftp_fstatvfs](#) (sftp_file file)
Get information about a mounted file system.
- int [sftp_get_error](#) (sftp_session sftp)
Get the last sftp error.
- int [sftp_init](#) (sftp_session sftp)
Initialize the sftp session with the server.
- sftp_attributes [sftp_lstat](#) (sftp_session session, const char *path)
Get information about a file or directory.
- int [sftp_mkdir](#) (sftp_session sftp, const char *directory, mode_t mode)
Create a directory.
- sftp_session [sftp_new](#) (ssh_session session)
Start a new sftp session.
- sftp_file [sftp_open](#) (sftp_session session, const char *file, int accesstype, mode_t mode)
Open a file on the server.
- sftp_dir [sftp_opendir](#) (sftp_session session, const char *path)
Open a directory used to obtain directory entries.
- ssize_t [sftp_read](#) (sftp_file file, void *buf, size_t count)
Read from a file using an opened sftp file handle.
- sftp_attributes [sftp_readdir](#) (sftp_session session, sftp_dir dir)
Get a single file attributes structure of a directory.
- char * [sftp_readlink](#) (sftp_session sftp, const char *path)
Read the value of a symbolic link.
- int [sftp_rename](#) (sftp_session sftp, const char *original, const char *newname)
Rename or move a file or directory.
- void [sftp_rewind](#) (sftp_file file)
Rewinds the position of the file pointer to the beginning of the file.
- int [sftp_rmdir](#) (sftp_session sftp, const char *directory)
Remove a directory.

- int [sftp_seek](#) (sftp_file file, uint32_t new_offset)
Seek to a specific location in a file.
- int [sftp_seek64](#) (sftp_file file, uint64_t new_offset)
Seek to a specific location in a file.
- int [sftp_server_version](#) (sftp_session sftp)
Get the version of the SFTP protocol supported by the server.
- int [sftp_setstat](#) (sftp_session sftp, const char *file, sftp_attributes attr)
Set file attributes on a file, directory or symbolic link.
- sftp_attributes [sftp_stat](#) (sftp_session session, const char *path)
Get information about a file or directory.
- sftp_statvfs_t [sftp_statvfs](#) (sftp_session sftp, const char *path)
Get information about a mounted file system.
- void [sftp_statvfs_free](#) (sftp_statvfs_t statvfs_o)
Free the memory of an allocated statvfs.
- int [sftp_symlink](#) (sftp_session sftp, const char *target, const char *dest)
Create a symbolic link.
- unsigned long [sftp_tell](#) (sftp_file file)
Report current byte position in file.
- uint64_t [sftp_tell64](#) (sftp_file file)
Report current byte position in file.
- int [sftp_unlink](#) (sftp_session sftp, const char *file)
Unlink (delete) a file.
- int [sftp_utimes](#) (sftp_session sftp, const char *file, const struct timeval *times)
Change the last modification and access time of a file.
- ssize_t [sftp_write](#) (sftp_file file, const void *buf, size_t count)
Write to a file using an opened sftp file handle.

Server responses

Responses returned by the sftp server.

- #define [SSH_FX_OK](#) 0

No error.

- #define [SSH_FX_EOF](#) 1
End-of-file encountered.
- #define [SSH_FX_NO_SUCH_FILE](#) 2
File doesn't exist.
- #define [SSH_FX_PERMISSION_DENIED](#) 3
Permission denied.
- #define [SSH_FX_FAILURE](#) 4
Generic failure.
- #define [SSH_FX_BAD_MESSAGE](#) 5
Garbage received from server.
- #define [SSH_FX_NO_CONNECTION](#) 6
No connection has been set up.
- #define [SSH_FX_CONNECTION_LOST](#) 7
There was a connection, but we lost it.
- #define [SSH_FX_OP_UNSUPPORTED](#) 8
Operation not supported by the server.
- #define [SSH_FX_INVALID_HANDLE](#) 9
Invalid file handle.
- #define [SSH_FX_NO_SUCH_PATH](#) 10
No such file or directory path exists.
- #define [SSH_FX_FILE_ALREADY_EXISTS](#) 11
An attempt to create an already existing file or directory has been made.
- #define [SSH_FX_WRITE_PROTECT](#) 12
We are trying to write on a write-protected filesystem.
- #define [SSH_FX_NO_MEDIA](#) 13
No media in remote drive.

9.4.1 Function Documentation

9.4.1.1 `int sftp_async_read (sftp_file file, void * data, uint32_t len, uint32_t id)`

Wait for an asynchronous read to complete and save the data.

Parameters

<i>file</i>	The opened sftp file handle to be read from.
<i>data</i>	Pointer to buffer to receive read data.
<i>len</i>	Size of the buffer in bytes. It should be bigger or equal to the length parameter of the sftp_async_read_begin() call.
<i>id</i>	The identifier returned by the sftp_async_read_begin() function.

Returns

Number of bytes read, 0 on EOF, SSH_ERROR if an error occurred, SSH_AGAIN if the file is opened in nonblocking mode and the request hasn't been executed yet.

Warning

A call to this function with an invalid identifier will never return.

See also

[sftp_async_read_begin\(\)](#)

9.4.1.2 `int sftp_async_read_begin (sftp_file file, uint32_t len)`

Start an asynchronous read from a file using an opened sftp file handle.

Its goal is to avoid the slowdowns related to the request/response pattern of a synchronous read. To do so, you must call 2 functions:

[sftp_async_read_begin\(\)](#) and [sftp_async_read\(\)](#).

The first step is to call [sftp_async_read_begin\(\)](#). This function returns a request identifier. The second step is to call [sftp_async_read\(\)](#) using the returned identifier.

Parameters

<i>file</i>	The opened sftp file handle to be read from.
<i>len</i>	Size to read in bytes.

Returns

An identifier corresponding to the sent request, < 0 on error.

Warning

When calling this function, the internal offset is updated corresponding to the len parameter.

A call to [sftp_async_read_begin\(\)](#) sends a request to the server. When the server answers, libssh allocates memory to store it until [sftp_async_read\(\)](#) is called. Not calling [sftp_async_read\(\)](#) will lead to memory leaks.

See also

[sftp_async_read\(\)](#)
[sftp_open\(\)](#)

9.4.1.3 void sftp_attributes_free (sftp_attributes *file*)

Free a sftp attribute structure.

Parameters

<i>file</i>	The sftp attribute structure to free.
-------------	---------------------------------------

9.4.1.4 char* sftp_canonicalize_path (sftp_session *sftp*, const char * *path*)

Canonicalize a sftp path.

Parameters

<i>sftp</i>	The sftp session handle.
<i>path</i>	The path to be canonicalized.

Returns

The canonicalize path, NULL on error.

9.4.1.5 int sftp_chmod (sftp_session *sftp*, const char * *file*, mode_t *mode*)

Change permissions of a file.

Parameters

<i>sftp</i>	The sftp session handle.
<i>file</i>	The file which owner and group should be changed.
<i>mode</i>	Specifies the permissions to use. It is modified by the process's umask in the usual way: The permissions of the created file are (mode & ~umask)

Returns

0 on success, < 0 on error with ssh and sftp error set.

9.4.1.6 int sftp_chown (sftp_session *sftp*, const char * *file*, uid_t *owner*, gid_t *group*)

Change the file owner and group.

Parameters

<i>sftp</i>	The sftp session handle.
<i>file</i>	The file which owner and group should be changed.
<i>owner</i>	The new owner which should be set.
<i>group</i>	The new group which should be set.

Returns

0 on success, < 0 on error with ssh and sftp error set.

9.4.1.7 int sftp_close (sftp_file *file*)

Close an open file handle.

Parameters

<i>file</i>	The open sftp file handle to close.
-------------	-------------------------------------

Returns

Returns SSH_NO_ERROR or SSH_ERROR if an error occurred.

See also

[sftp_open\(\)](#)

9.4.1.8 int sftp_closedir (sftp_dir *dir*)

Close a directory handle opened by [sftp_opendir\(\)](#).

Parameters

<i>dir</i>	The sftp directory handle to close.
------------	-------------------------------------

Returns

Returns SSH_NO_ERROR or SSH_ERROR if an error occurred.

9.4.1.9 int sftp_dir_eof (sftp_dir *dir*)

Tell if the directory has reached EOF (End Of File).

Parameters

<i>dir</i>	The sftp directory handle.
------------	----------------------------

Returns

1 if the directory is EOF, 0 if not.

See also

[sftp_readdir\(\)](#)

9.4.1.10 `int sftp_extension_supported (sftp_session sftp, const char * name, const char * data)`

Check if the given extension is supported.

Parameters

<i>sftp</i>	The sftp session to use.
<i>name</i>	The name of the extension.
<i>data</i>	The data of the extension.

Returns

1 if supported, 0 if not.

Example:

```
sftp_extension_supported(sftp, "statvfs@openssh.com", "2");
```

9.4.1.11 `unsigned int sftp_extensions_get_count (sftp_session sftp)`

Get the count of extensions provided by the server.

Parameters

<i>sftp</i>	The sftp session to use.
-------------	--------------------------

Returns

The count of extensions provided by the server, 0 on error or not available.

9.4.1.12 `const char* sftp_extensions_get_data (sftp_session sftp, unsigned int indexn)`

Get the data of the extension provided by the server.

This is normally the version number of the extension.

Parameters

<i>sftp</i>	The sftp session to use.
<i>indexn</i>	The index number of the extension data you want.

Returns

The data of the extension.

9.4.1.13 `const char* sftp_extensions_get_name (sftp_session sftp, unsigned int indexn)`

Get the name of the extension provided by the server.

Parameters

<i>sftp</i>	The sftp session to use.
<i>indexn</i>	The index number of the extension name you want.

Returns

The name of the extension.

9.4.1.14 `void sftp_free (sftp_session sftp)`

Close and deallocate a sftp session.

Parameters

<i>sftp</i>	The sftp session handle to free.
-------------	----------------------------------

9.4.1.15 `sftp_attributes sftp_fstat (sftp_file file)`

Get information about a file or directory from a file handle.

Parameters

<i>file</i>	The sftp file handle to get the stat information.
-------------	---

Returns

The sftp attributes structure of the file or directory, NULL on error with ssh and sftp error set.

9.4.1.16 `sftp_statvfs_t sftp_fstatvfs (sftp_file file)`

Get information about a mounted file system.

Parameters

<i>file</i>	An opened file.
-------------	-----------------

Returns

A statvfs structure or NULL on error.

9.4.1.17 int sftp_get_error (sftp_session sftp)

Get the last sftp error.

Use this function to get the latest error set by a posix like sftp function.

Parameters

<i>sftp</i>	The sftp session where the error is saved.
-------------	--

Returns

The saved error (see server responses), < 0 if an error in the function occurred.

9.4.1.18 int sftp_init (sftp_session sftp)

Initialize the sftp session with the server.

Parameters

<i>sftp</i>	The sftp session to initialize.
-------------	---------------------------------

Returns

0 on success, < 0 on error with ssh error set.

9.4.1.19 sftp_attributes sftp_lstat (sftp_session session, const char * path)

Get information about a file or directory.

Identical to sftp_stat, but if the file or directory is a symbolic link, then the link itself is stated, not the file that it refers to.

Parameters

<i>session</i>	The sftp session handle.
<i>path</i>	The path to the file or directory to obtain the information.

Returns

The sftp attributes structure of the file or directory, NULL on error with ssh and sftp error set.

9.4.1.20 `int sftp_mkdir (sftp_session sftp, const char * directory, mode_t mode)`

Create a directory.

Parameters

<i>sftp</i>	The sftp session handle.
<i>directory</i>	The directory to create.
<i>mode</i>	Specifies the permissions to use. It is modified by the process's umask in the usual way: The permissions of the created file are (mode & ~umask)

Returns

0 on success, < 0 on error with ssh and sftp error set.

9.4.1.21 `sftp_session sftp_new (ssh_session session)`

Start a new sftp session.

Parameters

<i>session</i>	The ssh session to use.
----------------	-------------------------

Returns

A new sftp session or NULL on error.

9.4.1.22 `sftp_file sftp_open (sftp_session session, const char * file, int accesstype, mode_t mode)`

Open a file on the server.

Parameters

<i>session</i>	The sftp session handle.
<i>file</i>	The file to be opened.
<i>accesstype</i>	Is one of O_RDONLY, O_WRONLY or O_RDWR which request opening the file read-only, write-only or read/write. Access may also be bitwise-or'd with one or more of the following: O_CREAT - If the file does not exist it will be created. O_EXCL - When used with O_CREAT, if the file already exists it is an error and the open will fail. O_TRUNC - If the file already exists it will be truncated.
<i>mode</i>	Mode specifies the permissions to use if a new file is created. It is modified by the process's umask in the usual way: The permissions of the created file are (mode & ~umask)

Returns

A sftp file handle, NULL on error with ssh and sftp error set.

9.4.1.23 `sftp_dir sftp_opendir (sftp_session session, const char * path)`

Open a directory used to obtain directory entries.

Parameters

<i>session</i>	The sftp session handle to open the directory.
<i>path</i>	The path of the directory to open.

Returns

A sftp directory handle or NULL on error with ssh and sftp error set.

See also

[sftp_readdir](#)
[sftp_closedir](#)

9.4.1.24 `ssize_t sftp_read (sftp_file file, void * buf, size_t count)`

Read from a file using an opened sftp file handle.

Parameters

<i>file</i>	The opened sftp file handle to be read from.
<i>buf</i>	Pointer to buffer to receive read data.
<i>count</i>	Size of the buffer in bytes.

Returns

Number of bytes written, < 0 on error with ssh and sftp error set.

9.4.1.25 `sftp_attributes sftp_readdir (sftp_session session, sftp_dir dir)`

Get a single file attributes structure of a directory.

Parameters

<i>session</i>	The sftp session handle to read the directory entry.
<i>dir</i>	The opened sftp directory handle to read from.

Returns

A file attribute structure or NULL at the end of the directory.

See also

[sftp_opendir\(\)](#)
[sftp_attribute_free\(\)](#)
[sftp_closedir\(\)](#)

9.4.1.26 `char* sftp_readlink (sftp_session sftp, const char * path)`

Read the value of a symbolic link.

Parameters

<i>sftp</i>	The sftp session handle.
<i>path</i>	Specifies the path name of the symlink to be read.

Returns

The target of the link, NULL on error.

9.4.1.27 `int sftp_rename (sftp_session sftp, const char * original, const char * newname)`

Rename or move a file or directory.

Parameters

<i>sftp</i>	The sftp session handle.
<i>original</i>	The original url (source url) of file or directory to be moved.
<i>newname</i>	The new url (destination url) of the file or directory after the move.

Returns

0 on success, < 0 on error with ssh and sftp error set.

9.4.1.28 `void sftp_rewind (sftp_file file)`

Rewinds the position of the file pointer to the beginning of the file.

Parameters

<i>file</i>	Open sftp file handle.
-------------	------------------------

9.4.1.29 `int sftp_rmdir (sftp_session sftp, const char * directory)`

Remove a directory.

Parameters

<i>sftp</i>	The sftp session handle.
<i>directory</i>	The directory to remove.

Returns

0 on success, < 0 on error with ssh and sftp error set.

9.4.1.30 int sftp_seek (sftp_file file, uint32_t new_offset)

Seek to a specific location in a file.

Parameters

<i>file</i>	Open sftp file handle to seek in.
<i>new_offset</i>	Offset in bytes to seek.

Returns

0 on success, < 0 on error.

9.4.1.31 int sftp_seek64 (sftp_file file, uint64_t new_offset)

Seek to a specific location in a file.

This is the 64bit version.

Parameters

<i>file</i>	Open sftp file handle to seek in.
<i>new_offset</i>	Offset in bytes to seek.

Returns

0 on success, < 0 on error.

9.4.1.32 int sftp_server_version (sftp_session sftp)

Get the version of the SFTP protocol supported by the server.

Parameters

<i>sftp</i>	The sftp session handle.
-------------	--------------------------

Returns

The server version.

9.4.1.33 int sftp_setstat (sftp_session sftp, const char * file, sftp_attributes attr)

Set file attributes on a file, directory or symbolic link.

Parameters

<i>sftp</i>	The sftp session handle.
<i>file</i>	The file which attributes should be changed.
<i>attr</i>	The file attributes structure with the attributes set which should be changed.

Returns

0 on success, < 0 on error with ssh and sftp error set.

9.4.1.34 sftp_attributes sftp_stat (sftp_session session, const char * path)

Get information about a file or directory.

Parameters

<i>session</i>	The sftp session handle.
<i>path</i>	The path to the file or directory to obtain the information.

Returns

The sftp attributes structure of the file or directory, NULL on error with ssh and sftp error set.

9.4.1.35 sftp_statvfs_t sftp_statvfs (sftp_session sftp, const char * path)

Get information about a mounted file system.

Parameters

<i>sftp</i>	The sftp session handle.
<i>path</i>	The pathname of any file within the mounted file system.

Returns

A statvfs structure or NULL on error.

9.4.1.36 void sftp_statvfs_free (sftp_statvfs_t statvfs_o)

Free the memory of an allocated statvfs.

Parameters

<i>statvfs_o</i>	The statvfs to free.
------------------	----------------------

9.4.1.37 int sftp_symlink (sftp_session sftp, const char * target, const char * dest)

Create a symbolic link.

Parameters

<i>sftp</i>	The sftp session handle.
<i>target</i>	Specifies the target of the symlink.
<i>dest</i>	Specifies the path name of the symlink to be created.

Returns

0 on success, < 0 on error with ssh and sftp error set.

9.4.1.38 unsigned long sftp_tell (sftp_file file)

Report current byte position in file.

Parameters

<i>file</i>	Open sftp file handle.
-------------	------------------------

Returns

The offset of the current byte relative to the beginning of the file associated with the file descriptor. < 0 on error.

9.4.1.39 uint64_t sftp_tell64 (sftp_file file)

Report current byte position in file.

Parameters

<i>file</i>	Open sftp file handle.
-------------	------------------------

Returns

The offset of the current byte relative to the beginning of the file associated with the file descriptor. < 0 on error.

9.4.1.40 int sftp_unlink (sftp_session sftp, const char * file)

Unlink (delete) a file.

Parameters

<i>sftp</i>	The sftp session handle.
<i>file</i>	The file to unlink/delete.

Returns

0 on success, < 0 on error with ssh and sftp error set.

9.4.1.41 int sftp_utimes (sftp_session sftp, const char * file, const struct timeval * times)

Change the last modification and access time of a file.

Parameters

<i>sftp</i>	The sftp session handle.
<i>file</i>	The file which owner and group should be changed.
<i>times</i>	A timeval structure which contains the desired access and modification time.

Returns

0 on success, < 0 on error with ssh and sftp error set.

9.4.1.42 ssize_t sftp_write (sftp_file file, const void * buf, size_t count)

Write to a file using an opened sftp file handle.

Parameters

<i>file</i>	Open sftp file handle to write to.
<i>buf</i>	Pointer to buffer to write data.
<i>count</i>	Size of buffer in bytes.

Returns

Number of bytes written, < 0 on error with ssh and sftp error set.

See also

[sftp_open\(\)](#)
[sftp_read\(\)](#)
[sftp_close\(\)](#)

9.5 The SSH authentication functions.

Functions to authenticate with a server.

Functions

- void [privatekey_free](#) (ssh_private_key prv)
Deallocate a private key object.
- ssh_private_key [privatekey_from_file](#) (ssh_session session, const char *filename, int type, const char *passphrase)
Reads a SSH private key from a file.
- ssh_string [publickey_from_file](#) (ssh_session session, const char *filename, int *type)
Retrieve a public key from a file.

- `ssh_public_key publickey_from_privatekey (ssh_private_key prv)`
Make a public_key object out of a private_key object.
- `ssh_string publickey_to_string (ssh_public_key key)`
Convert a public_key object into a a SSH string.
- `int ssh_auth_list (ssh_session session)`
retrieves available authentication methods for this session
- `enum ssh_keytypes_e ssh_privatekey_type (ssh_private_key privatekey)`
returns the type of a private key
- `int ssh_publickey_to_file (ssh_session session, const char *file, ssh_string pubkey, int type)`
Write a public key to a file.
- `int ssh_try_publickey_from_file (ssh_session session, const char *keyfile, ssh_string *publickey, int *type)`
Try to read the public key from a given file.
- `int ssh_userauth_agent_pubkey (ssh_session session, const char *username, ssh_public_key publickey)`
Try to authenticate through public key with an ssh agent.
- `int ssh_userauth_autopubkey (ssh_session session, const char *passphrase)`
Tries to automatically authenticate with public key and "none".
- `int ssh_userauth_kbdint (ssh_session session, const char *user, const char *submethods)`

Try to authenticate through the "keyboard-interactive" method.
- `const char * ssh_userauth_kbdint_getinstruction (ssh_session session)`
Get the "instruction" of the message block.
- `const char * ssh_userauth_kbdint_getname (ssh_session session)`
Get the "name" of the message block.
- `int ssh_userauth_kbdint_getnprompts (ssh_session session)`
Get the number of prompts (questions) the server has given.
- `const char * ssh_userauth_kbdint_getprompt (ssh_session session, unsigned int i, char *echo)`
Get a prompt from a message block.
- `int ssh_userauth_kbdint_setanswer (ssh_session session, unsigned int i, const char *answer)`
Set the answer for a question from a message block.

- int [ssh_userauth_list](#) (ssh_session session, const char *username)
retrieves available authentication methods for this session
- int [ssh_userauth_none](#) (ssh_session session, const char *username)
Try to authenticate through the "none" method.
- int [ssh_userauth_offer_pubkey](#) (ssh_session session, const char *username, int type, ssh_string publickey)
Try to authenticate through public key.
- int [ssh_userauth_password](#) (ssh_session session, const char *username, const char *password)
Try to authenticate by password.
- int [ssh_userauth_privatekey_file](#) (ssh_session session, const char *username, const char *filename, const char *passphrase)
Try to authenticate through a private key file.
- int [ssh_userauth_pubkey](#) (ssh_session session, const char *username, ssh_string publickey, ssh_private_key privatekey)
Try to authenticate through public key.

9.5.1 Detailed Description

Functions to authenticate with a server.

9.5.2 Function Documentation

9.5.2.1 void privatekey_free (ssh_private_key prv)

Deallocate a private key object.

Parameters

in	prv	The private_key object to free.
----	-----	---------------------------------

Referenced by [ssh_bind_accept\(\)](#), [ssh_free\(\)](#), [ssh_userauth_autopubkey\(\)](#), and [ssh_userauth_privatekey_file\(\)](#).

9.5.2.2 ssh_private_key privatekey_from_file (ssh_session session, const char * filename, int type, const char * passphrase)

Reads a SSH private key from a file.

Parameters

in	<i>session</i>	The SSH Session to use.
in	<i>filename</i>	The filename of the the private key.
in	<i>type</i>	The type of the private key. This could be SSH_KEYTYPE_DSS or SSH_KEYTYPE_RSA. Pass 0 to automatically detect the type.
in	<i>passphrase</i>	The passphrase to decrypt the private key. Set to null if none is needed or it is unknown.

Returns

A `private_key` object containing the private key, or NULL on error.

See also

[privatekey_free\(\)](#)
[publickey_from_privatekey\(\)](#)

References `ssh_init()`, `ssh_log()`, and `SSH_LOG_RARE`.

Referenced by `ssh_key_import_private()`, `ssh_userauth_autopubkey()`, and `ssh_userauth_privatekey_file()`.

9.5.2.3 ssh_string publickey_from_file (ssh_session session, const char * filename, int * type)

Retrieve a public key from a file.

Parameters

in	<i>session</i>	The SSH session to use.
in	<i>filename</i>	The filename of the public key.
out	<i>type</i>	The Pointer to a integer. If it is not NULL, it will contain the type of the key after execution.

Returns

A SSH String containing the public key, or NULL if it failed.

See also

[string_free\(\)](#)
[publickey_from_privatekey\(\)](#)

References `ssh_buffer_free()`, `ssh_string_fill()`, and `ssh_string_new()`.

Referenced by `ssh_try_publickey_from_file()`, and `ssh_userauth_privatekey_file()`.

9.5.2.4 ssh_public_key publickey_from_privatekey (ssh_private_key prv)

Make a `public_key` object out of a `private_key` object.

Parameters

<code>in</code>	<code>prv</code>	The private key to generate the public key.
-----------------	------------------	---

Returns

The generated public key, NULL on error.

See also

[publickey_to_string\(\)](#)

References `ssh_string_burn()`, `ssh_string_data()`, `ssh_string_fill()`, `ssh_string_free()`, `ssh_string_len()`, and `ssh_string_new()`.

Referenced by `ssh_userauth_autopubkey()`, and `ssh_userauth_pubkey()`.

9.5.2.5 `ssh_string publickey_to_string (ssh_public_key key)`

Convert a `public_key` object into a a SSH string.

Parameters

<code>in</code>	<code>key</code>	The public key to convert.
-----------------	------------------	----------------------------

Returns

An allocated SSH String containing the public key, NULL on error.

See also

`string_free()`

References `ssh_buffer_free()`, `ssh_buffer_new()`, `ssh_string_fill()`, `ssh_string_free()`, `ssh_string_from_char()`, and `ssh_string_new()`.

Referenced by `ssh_userauth_agent_pubkey()`, `ssh_userauth_autopubkey()`, and `ssh_userauth_pubkey()`.

9.5.2.6 `int ssh_auth_list (ssh_session session)`

retrieves available authentication methods for this session

Deprecated**See also**

[ssh_userauth_list](#)

References `ssh_userauth_list()`.

9.5.2.7 enum `ssh_keytypes_e` `ssh_privatekey_type` (`ssh_private_key` *privatekey*)

returns the type of a private key

Parameters

in	<i>privatekey</i>	the private key handle
----	-------------------	------------------------

Returns

one of `SSH_KEYTYPE_RSA`, `SSH_KEYTYPE_DSS`, `SSH_KEYTYPE_RSA1`
`SSH_KEYTYPE_UNKNOWN` if the type is unknown

See also

[privatekey_from_file](#)
[ssh_userauth_offer_pubkey](#)

9.5.2.8 int `ssh_publickey_to_file` (`ssh_session` *session*, `const char *` *file*, `ssh_string` *pubkey*, int *type*)

Write a public key to a file.

Parameters

in	<i>session</i>	The ssh session to use.
in	<i>file</i>	The filename to write the key into.
in	<i>pubkey</i>	The public key to write.
in	<i>type</i>	The type of the public key.

Returns

0 on success, -1 on error.

References `ssh_log()`, `SSH_LOG_PACKET`, `SSH_LOG_RARE`, and `ssh_string_len()`.

Referenced by `ssh_userauth_autopubkey()`.

9.5.2.9 int `ssh_try_publickey_from_file` (`ssh_session` *session*, `const char *` *keyfile*, `ssh_string *` *publickey*, int * *type*)

Try to read the public key from a given file.

Parameters

in	<i>session</i>	The ssh session to use.
in	<i>keyfile</i>	The name of the private keyfile.
out	<i>publickey</i>	A <code>ssh_string</code> to store the public key.
out	<i>type</i>	A pointer to an integer to store the type.

Returns

0 on success, -1 on error or the private key doesn't exist, 1 if the public key doesn't exist.

References `publickey_from_file()`, `ssh_get_error()`, `ssh_log()`, and `SSH_LOG_PACKET`.

Referenced by `ssh_userauth_autopubkey()`.

9.5.2.10 `int ssh_userauth_agent_pubkey (ssh_session session, const char * username, ssh_public_key publickey)`

Try to authenticate through public key with an ssh agent.

Parameters

in	<i>session</i>	The ssh session to use.
in	<i>username</i>	The username to authenticate. You can specify NULL if <code>ssh_option_set_username()</code> has been used. You cannot try two different logins in a row.
in	<i>publickey</i>	The public key provided by the agent.

Returns

`SSH_AUTH_ERROR`: A serious error happened.

`SSH_AUTH_DENIED`: Authentication failed: use another method.

`SSH_AUTH_PARTIAL`: You've been partially authenticated, you still have to use another method.

`SSH_AUTH_SUCCESS`: Authentication successful.

See also

[publickey_from_file\(\)](#)
[privatekey_from_file\(\)](#)
[privatekey_free\(\)](#)
[ssh_userauth_offer_pubkey\(\)](#)

References `publickey_to_string()`, `ssh_string_free()`, and `ssh_string_from_char()`.

Referenced by `ssh_userauth_autopubkey()`.

9.5.2.11 `int ssh_userauth_autopubkey (ssh_session session, const char * passphrase)`

Tries to automatically authenticate with public key and "none".

It may fail, for instance it doesn't ask for a password and uses a default asker for passphrases (in case the private key is encrypted).

Parameters

in	<i>session</i>	The ssh session to authenticate with.
in	<i>passphrase</i>	Use this passphrase to unlock the privatekey. Use NULL if you don't want to use a passphrase or the user should be asked.

Returns

SSH_AUTH_ERROR: A serious error happened
 SSH_AUTH_DENIED: Authentication failed: use another method
 SSH_AUTH_PARTIAL: You've been partially authenticated, you still have to use another method
 SSH_AUTH_SUCCESS: Authentication success

See also

[ssh_userauth_kbdint\(\)](#)
[ssh_userauth_password\(\)](#)

References `privatekey_free()`, `privatekey_from_file()`, `publickey_from_privatekey()`, `publickey_to_string()`, `ssh_log()`, `SSH_LOG_PACKET`, `SSH_LOG_PROTOCOL`, `SSH_LOG_RARE`, `ssh_publickey_to_file()`, `ssh_string_free()`, `ssh_try_publickey_from_file()`, `ssh_userauth_agent_pubkey()`, `ssh_userauth_none()`, `ssh_userauth_offer_pubkey()`, and `ssh_userauth_pubkey()`.

Referenced by `ssh::Session::userauthAutopubkey()`.

9.5.2.12 `int ssh_userauth_kbdint (ssh_session session, const char * user, const char * submethods)`

Try to authenticate through the "keyboard-interactive" method.

Parameters

<i>in</i>	<i>session</i>	The ssh session to use.
<i>in</i>	<i>user</i>	The username to authenticate. You can specify NULL if <code>ssh_option_set_username()</code> has been used. You cannot try two different logins in a row.
<i>in</i>	<i>submethods</i>	Undocumented. Set it to NULL.

Returns

SSH_AUTH_ERROR: A serious error happened
 SSH_AUTH_DENIED: Authentication failed : use another method
 SSH_AUTH_PARTIAL: You've been partially authenticated, you still have to use another method
 SSH_AUTH_SUCCESS: Authentication success
 SSH_AUTH_INFO: The server asked some questions. Use [ssh_userauth_kbdint_getnprompts\(\)](#) and such.
 SSH_AUTH_AGAIN: In nonblocking mode, you've got to call this again later.

See also

[ssh_userauth_kbdint_getnprompts\(\)](#)
[ssh_userauth_kbdint_getname\(\)](#)
[ssh_userauth_kbdint_getinstruction\(\)](#)
[ssh_userauth_kbdint_getprompt\(\)](#)
[ssh_userauth_kbdint_setanswer\(\)](#)

9.5.2.13 `const char* ssh_userauth_kbdint_getinstruction (ssh_session session)`

Get the "instruction" of the message block.

You have called `ssh_userauth_kbdint()` and got SSH_AUTH_INFO. This function returns the questions from the server.

Parameters

in	<i>session</i>	The ssh session to use.
----	----------------	-------------------------

Returns

The instruction of the message block.

9.5.2.14 `const char* ssh_userauth_kbdint_getname (ssh_session session)`

Get the "name" of the message block.

You have called `ssh_userauth_kbdint()` and got SSH_AUTH_INFO. This function returns the questions from the server.

Parameters

in	<i>session</i>	The ssh session to use.
----	----------------	-------------------------

Returns

The name of the message block. Do not free it.

9.5.2.15 `int ssh_userauth_kbdint_getnprompts (ssh_session session)`

Get the number of prompts (questions) the server has given.

You have called `ssh_userauth_kbdint()` and got SSH_AUTH_INFO. This function returns the questions from the server.

Parameters

in	<i>session</i>	The ssh session to use.
----	----------------	-------------------------

Returns

The number of prompts.

9.5.2.16 `const char* ssh_userauth_kbdint_getprompt (ssh_session session, unsigned int i, char * echo)`

Get a prompt from a message block.

You have called `ssh_userauth_kbdint()` and got `SSH_AUTH_INFO`. This function returns the questions from the server.

Parameters

in	<i>session</i>	The ssh session to use.
in	<i>i</i>	The index number of the i'th prompt.
in	<i>echo</i>	When different of NULL, it will obtain a boolean meaning that the resulting user input should be echoed or not (like passwords).

Returns

A pointer to the prompt. Do not free it.

9.5.2.17 `int ssh_userauth_kbdint_setanswer (ssh_session session, unsigned int i, const char * answer)`

Set the answer for a question from a message block.

If you have called `ssh_userauth_kbdint()` and got `SSH_AUTH_INFO`, this function returns the questions from the server.

Parameters

in	<i>session</i>	The ssh session to use.
in	<i>i</i>	index The number of the ith prompt.
in	<i>answer</i>	The answer to give to the server.

Returns

0 on success, < 0 on error.

9.5.2.18 `int ssh_userauth_list (ssh_session session, const char * username)`

retrieves available authentication methods for this session

Parameters

in	<i>session</i>	the SSH session
in	<i>username</i>	Deprecated, set to NULL.

Returns

A bitfield of values `SSH_AUTH_METHOD_PASSWORD`, `SSH_AUTH_METHOD_PUBLICKEY`, `SSH_AUTH_METHOD_HOSTBASED`, `SSH_AUTH_METHOD_INTERACTIVE`.

Warning

Other reserved flags may appear in future versions.

This call will block, even in nonblocking mode, if run for the first time before a (complete) call to `ssh_userauth_none`.

References `ssh_userauth_none()`.

Referenced by `ssh::Session::getAuthList()`, and `ssh_auth_list()`.

9.5.2.19 `int ssh_userauth_none (ssh_session session, const char * username)`

Try to authenticate through the "none" method.

Parameters

in	<i>session</i>	The ssh session to use.
in	<i>username</i>	Deprecated, set to NULL.

Returns

SSH_AUTH_ERROR: A serious error happened.

SSH_AUTH_DENIED: Authentication failed: use another method

SSH_AUTH_PARTIAL: You've been partially authenticated, you still have to use another method

SSH_AUTH_SUCCESS: Authentication success

SSH_AUTH_AGAIN: In nonblocking mode, you've got to call this again later.

References `ssh_string_free()`, and `ssh_string_from_char()`.

Referenced by `ssh_userauth_autopubkey()`, `ssh_userauth_list()`, and `ssh::Session::userauthNone()`.

9.5.2.20 `int ssh_userauth_offer_pubkey (ssh_session session, const char * username, int type, ssh_string publickey)`

Try to authenticate through public key.

Parameters

in	<i>session</i>	The ssh session to use.
in	<i>username</i>	The username to authenticate. You can specify NULL if <code>ssh_option_set_username()</code> has been used. You cannot try two different logins in a row.
in	<i>type</i>	The type of the public key. This value is given by publickey_from_file() or ssh_privatekey_type() .
in	<i>publickey</i>	A public key returned by publickey_from_file() .

Returns

SSH_AUTH_ERROR: A serious error happened.

SSH_AUTH_DENIED: The server doesn't accept that public key as an authentication token. Try another key or another method.

SSH_AUTH_PARTIAL: You've been partially authenticated, you still have to use another method.

SSH_AUTH_SUCCESS: The public key is accepted, you want now to use [ssh_userauth_pubkey\(\)](#).

See also

[publickey_from_file\(\)](#)
[privatekey_from_file\(\)](#)
[ssh_privatekey_type\(\)](#)
[ssh_userauth_pubkey\(\)](#)

References [ssh_string_free\(\)](#), and [ssh_string_from_char\(\)](#).

Referenced by [ssh_userauth_autopubkey\(\)](#), and [ssh::Session::userauthOfferPubkey\(\)](#).

9.5.2.21 `int ssh_userauth_password (ssh_session session, const char * username, const char * password)`

Try to authenticate by password.

Parameters

in	<i>session</i>	The ssh session to use.
in	<i>username</i>	The username to authenticate. You can specify NULL if ssh_option_set_username() has been used. You cannot try two different logins in a row.
in	<i>password</i>	The password to use. Take care to clean it after the authentication.

Returns

SSH_AUTH_ERROR: A serious error happened.
SSH_AUTH_DENIED: Authentication failed: use another method.
SSH_AUTH_PARTIAL: You've been partially authenticated, you still have to use another method.
SSH_AUTH_SUCCESS: Authentication successful.
SSH_AUTH_AGAIN: In nonblocking mode, you've got to call this again later.

See also

[ssh_userauth_kbdint\(\)](#)
BURN_STRING

References [ssh_string_burn\(\)](#), [ssh_string_free\(\)](#), and [ssh_string_from_char\(\)](#).

Referenced by [ssh::Session::userauthPassword\(\)](#).

9.5.2.22 `int ssh_userauth_privatekey_file (ssh_session session, const char * username, const char * filename, const char * passphrase)`

Try to authenticate through a private key file.

Parameters

in	<i>session</i>	The ssh session to use.
in	<i>username</i>	The username to authenticate. You can specify NULL if <code>ssh_option_set_username()</code> has been used. You cannot try two different logins in a row.
in	<i>filename</i>	Filename containing the private key.
in	<i>passphrase</i>	Passphrase to decrypt the private key. Set to null if none is needed or it is unknown.

Returns

SSH_AUTH_ERROR: A serious error happened.

SSH_AUTH_DENIED: Authentication failed: use another method.

SSH_AUTH_PARTIAL: You've been partially authenticated, you still have to use another method.

SSH_AUTH_SUCCESS: Authentication successful.

SSH_AUTH_AGAIN: In nonblocking mode, you've got to call this again later.

See also

[publickey_from_file\(\)](#)
[privatekey_from_file\(\)](#)
[privatekey_free\(\)](#)
[ssh_userauth_pubkey\(\)](#)

References `privatekey_free()`, `privatekey_from_file()`, `publickey_from_file()`, `ssh_log()`, `SSH_LOG_RARE`, `ssh_string_free()`, and `ssh_userauth_pubkey()`.

9.5.2.23 `int ssh_userauth_pubkey (ssh_session session, const char * username, ssh_string publickey, ssh_private_key privatekey)`

Try to authenticate through public key.

Parameters

in	<i>session</i>	The ssh session to use.
in	<i>username</i>	The username to authenticate. You can specify NULL if <code>ssh_option_set_username()</code> has been used. You cannot try two different logins in a row.
in	<i>publickey</i>	A public key returned by publickey_from_file() , or NULL to generate automatically from <code>privatekey</code> .
in	<i>privatekey</i>	A private key returned by privatekey_from_file() .

Returns

SSH_AUTH_ERROR: A serious error happened.

SSH_AUTH_DENIED: Authentication failed: use another method.

SSH_AUTH_PARTIAL: You've been partially authenticated, you still have to use another method.

SSH_AUTH_SUCCESS: Authentication successful.

See also

[publickey_from_file\(\)](#)
[privatekey_from_file\(\)](#)
[privatekey_free\(\)](#)
[ssh_userauth_offer_pubkey\(\)](#)

References [publickey_from_privatekey\(\)](#), [publickey_to_string\(\)](#), [ssh_string_free\(\)](#), and [ssh_string_from_char\(\)](#).

Referenced by [ssh_userauth_autopubkey\(\)](#), [ssh_userauth_privatekey_file\(\)](#), and [ssh::Session::userauthPubkey\(\)](#).

9.6 The SSH buffer functions.

Functions to handle SSH buffers.

Functions

- void [ssh_buffer_free](#) (struct ssh_buffer_struct *buffer)
Deallocate a SSH buffer.
- void * [ssh_buffer_get_begin](#) (struct ssh_buffer_struct *buffer)
Get a pointer on the head of a buffer.
- uint32_t [ssh_buffer_get_len](#) (struct ssh_buffer_struct *buffer)
Get the length of the buffer; not counting position.
- struct ssh_buffer_struct * [ssh_buffer_new](#) (void)
Create a new SSH buffer.

9.6.1 Detailed Description

Functions to handle SSH buffers.

9.6.2 Function Documentation**9.6.2.1 void ssh_buffer_free (struct ssh_buffer_struct * *buffer*)**

Deallocate a SSH buffer.

Parameters

<i>in</i>	<i>buffer</i>	The buffer to free.
-----------	---------------	---------------------

Referenced by `publickey_from_file()`, `publickey_to_string()`, `ssh_channel_change_pty_size()`, `ssh_channel_free()`, `ssh_channel_new()`, `ssh_channel_open_forward()`, `ssh_channel_request_env()`, `ssh_channel_request_exec()`, `ssh_channel_request_pty_size()`, `ssh_channel_request_send_signal()`, `ssh_channel_request_subsystem()`, `ssh_channel_request_x11()`, `ssh_forward_cancel()`, `ssh_forward_listen()`, and `ssh_free()`.

9.6.2.2 `void* ssh_buffer_get_begin (struct ssh_buffer_struct * buffer)`

Get a pointer on the head of a buffer.

Parameters

<code>in</code>	<code>buffer</code>	The buffer to get the head pointer.
-----------------	---------------------	-------------------------------------

Returns

A data pointer on the head. It doesn't take the position into account.

Warning

Don't expect data to be nul-terminated.

See also

`buffer_get_rest()`
`buffer_get_len()`

9.6.2.3 `uint32_t ssh_buffer_get_len (struct ssh_buffer_struct * buffer)`

Get the length of the buffer, not counting position.

Parameters

<code>in</code>	<code>buffer</code>	The buffer to get the length from.
-----------------	---------------------	------------------------------------

Returns

The length of the buffer.

See also

`buffer_get()`

9.6.2.4 `struct ssh_buffer_struct* ssh_buffer_new (void) [read]`

Create a new SSH buffer.

Returns

A newly initialized SSH buffer, NULL on error.

Referenced by `publickey_to_string()`, `ssh_channel_change_pty_size()`, `ssh_channel_new()`, `ssh_channel_open_forward()`, `ssh_channel_request_env()`, `ssh_channel_request_exec()`, `ssh_channel_request_pty_size()`, `ssh_channel_request_send_signal()`, `ssh_channel_request_subsystem()`, `ssh_channel_request_x11()`, `ssh_forward_cancel()`, `ssh_forward_listen()`, and `ssh_new()`.

9.7 The SSH channel functions

Functions that manage a SSH channel.

Functions

- int `channel_read_buffer` (ssh_channel channel, ssh_buffer buffer, uint32_t count, int is_stderr)
Read data from a channel into a buffer.
- ssh_channel `ssh_channel_accept_x11` (ssh_channel channel, int timeout_ms)
Accept an X11 forwarding channel.
- int `ssh_channel_change_pty_size` (ssh_channel channel, int cols, int rows)
Change the size of the terminal associated to a channel.
- int `ssh_channel_close` (ssh_channel channel)
Close a channel.
- void `ssh_channel_free` (ssh_channel channel)
Close and free a channel.
- int `ssh_channel_get_exit_status` (ssh_channel channel)
Get the exit status of the channel (error code from the executed instruction).
- ssh_session `ssh_channel_get_session` (ssh_channel channel)
Recover the session in which belongs a channel.
- int `ssh_channel_is_closed` (ssh_channel channel)
Check if the channel is closed or not.
- int `ssh_channel_is_eof` (ssh_channel channel)
Check if remote has sent an EOF.
- int `ssh_channel_is_open` (ssh_channel channel)
Check if the channel is open or not.
- ssh_channel `ssh_channel_new` (ssh_session session)
Allocate a new channel.

- int [ssh_channel_open_forward](#) (ssh_channel channel, const char *remotehost, int remoteport, const char *sourcehost, int localport)
Open a TCP/IP forwarding channel.
- int [ssh_channel_open_session](#) (ssh_channel channel)
Open a session channel (suited for a shell, not TCP forwarding).
- int [ssh_channel_poll](#) (ssh_channel channel, int is_stderr)
Polls a channel for data to read.
- int [ssh_channel_read](#) (ssh_channel channel, void *dest, uint32_t count, int is_stderr)
Reads data from a channel.
- int [ssh_channel_read_nonblocking](#) (ssh_channel channel, void *dest, uint32_t count, int is_stderr)
Do a nonblocking read on the channel.
- int [ssh_channel_request_env](#) (ssh_channel channel, const char *name, const char *value)
Set environment variables.
- int [ssh_channel_request_exec](#) (ssh_channel channel, const char *cmd)
Run a shell command without an interactive shell.
- int [ssh_channel_request_pty](#) (ssh_channel channel)
Request a PTY.
- int [ssh_channel_request_pty_size](#) (ssh_channel channel, const char *terminal, int col, int row)
Request a pty with a specific type and size.
- int [ssh_channel_request_send_signal](#) (ssh_channel channel, const char *sig)
Send a signal to remote process (as described in RFC 4254, section 6.9).
- int [ssh_channel_request_shell](#) (ssh_channel channel)
Request a shell.
- int [ssh_channel_request_subsystem](#) (ssh_channel channel, const char *subsys)
Request a subsystem (for example "sftp").
- int [ssh_channel_request_x11](#) (ssh_channel channel, int single_connection, const char *protocol, const char *cookie, int screen_number)
Sends the "x11-req" channel request over an existing session channel.

- int [ssh_channel_select](#) (ssh_channel *readchans, ssh_channel *writechans, ssh_channel *exceptchans, struct timeval *timeout)
Act like the standard select(2) on channels.
- int [ssh_channel_send_eof](#) (ssh_channel channel)
Send an end of file on the channel.
- void [ssh_channel_set_blocking](#) (ssh_channel channel, int blocking)
Put the channel into blocking or nonblocking mode.
- int [ssh_channel_write](#) (ssh_channel channel, const void *data, uint32_t len)
Blocking write on a channel.
- ssh_channel [ssh_forward_accept](#) (ssh_session session, int timeout_ms)
Accept an incoming TCP/IP forwarding channel.
- int [ssh_forward_cancel](#) (ssh_session session, const char *address, int port)
Sends the "cancel-tcpip-forward" global request to ask the server to cancel the tcpip-forward request.
- int [ssh_forward_listen](#) (ssh_session session, const char *address, int port, int *bound_port)
Sends the "tcpip-forward" global request to ask the server to begin listening for in-bound connections.

9.7.1 Detailed Description

Functions that manage a SSH channel.

9.7.2 Function Documentation

9.7.2.1 int [channel_read_buffer](#) (ssh_channel *channel*, ssh_buffer *buffer*, uint32_t *count*, int *is_stderr*)

Read data from a channel into a buffer.

Parameters

in	<i>channel</i>	The channel to read from.
in	<i>buffer</i>	The buffer which will get the data.
in	<i>count</i>	The count of bytes to be read. If it is bigger than 0, the exact size will be read, else (bytes=0) it will return once anything is available.
	<i>is_stderr</i>	A boolean value to mark reading from the stderr stream.

Returns

The number of bytes read, 0 on end of file or SSH_ERROR on error.

Deprecated

Please use `ssh_channel_read` instead

See also

[ssh_channel_read](#)

References `ssh_channel_is_eof()`, `ssh_channel_poll()`, and `ssh_channel_read()`.

9.7.2.2 ssh_channel_ssh_channel_accept_x11 (ssh_channel *channel*, int *timeout_ms*)

Accept an X11 forwarding channel.

Parameters

in	<i>channel</i>	An x11-enabled session channel.
in	<i>timeout_ms</i>	Timeout in milliseconds.

Returns

A newly created channel, or NULL if no X11 request from the server.

Referenced by `ssh::Channel::acceptX11()`.

9.7.2.3 int ssh_channel_change_pty_size (ssh_channel *channel*, int *cols*, int *rows*)

Change the size of the terminal associated to a channel.

Parameters

in	<i>channel</i>	The channel to change the size.
in	<i>cols</i>	The new number of columns.
in	<i>rows</i>	The new number of rows.

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

Warning

Do not call it from a signal handler if you are not sure any other libssh function using the same channel/session is running at same time (not 100% threadsafe).

References `ssh_buffer_free()`, and `ssh_buffer_new()`.

Referenced by `ssh::Channel::changePtySize()`.

9.7.2.4 `int ssh_channel_close (ssh_channel channel)`

Close a channel.

This sends an end of file and then closes the channel. You won't be able to recover any data the server was going to send or was in buffers.

Parameters

<code>in</code>	<code><i>channel</i></code>	The channel to close.
-----------------	-----------------------------	-----------------------

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

See also

`channel_free()`
`channel_eof()`

References `ssh_channel_send_eof()`, `ssh_log()`, and `SSH_LOG_PACKET`.

Referenced by `ssh::Channel::close()`, and `ssh_channel_free()`.

9.7.2.5 `void ssh_channel_free (ssh_channel channel)`

Close and free a channel.

Parameters

<code>in</code>	<code><i>channel</i></code>	The channel to free.
-----------------	-----------------------------	----------------------

Warning

Any data unread on this channel will be lost.

References `ssh_buffer_free()`, and `ssh_channel_close()`.

Referenced by `ssh_disconnect()`, and `ssh_free()`.

9.7.2.6 `int ssh_channel_get_exit_status (ssh_channel channel)`

Get the exit status of the channel (error code from the executed instruction).

Parameters

<code>in</code>	<code><i>channel</i></code>	The channel to get the status from.
-----------------	-----------------------------	-------------------------------------

Returns

The exit status, -1 if no exit status has been returned or eof not sent.

9.7.2.7 ssh_session ssh_channel_get_session (ssh_channel *channel*)

Recover the session in which belongs a channel.

Parameters

<i>in</i>	<i>channel</i>	The channel to recover the session from.
-----------	----------------	--

Returns

The session pointer.

9.7.2.8 int ssh_channel_is_closed (ssh_channel *channel*)

Check if the channel is closed or not.

Parameters

<i>in</i>	<i>channel</i>	The channel to check.
-----------	----------------	-----------------------

Returns

0 if channel is opened, nonzero otherwise.

See also

channel_is_open()

Referenced by ssh::Channel::isClosed().

9.7.2.9 int ssh_channel_is_eof (ssh_channel *channel*)

Check if remote has sent an EOF.

Parameters

<i>in</i>	<i>channel</i>	The channel to check.
-----------	----------------	-----------------------

Returns

0 if there is no EOF, nonzero otherwise.

Referenced by channel_read_buffer(), ssh::Channel::isEof(), and ssh_scp_pull_request().

9.7.2.10 int ssh_channel_is_open (ssh_channel *channel*)

Check if the channel is open or not.

Parameters

<i>in</i>	<i>channel</i>	The channel to check.
-----------	----------------	-----------------------

Returns

0 if channel is closed, nonzero otherwise.

See also

channel_is_closed()

Referenced by ssh::Channel::isOpen().

9.7.2.11 ssh_channel_ssh_channel_new (ssh_session *session*)

Allocate a new channel.

Parameters

in	<i>session</i>	The ssh session to use.
----	----------------	-------------------------

Returns

A pointer to a newly allocated channel, NULL on error.

References ssh_buffer_free(), and ssh_buffer_new().

9.7.2.12 int ssh_channel_open_forward (ssh_channel *channel*, const char * *remotehost*, int *remoteport*, const char * *sourcehost*, int *localport*)

Open a TCP/IP forwarding channel.

Parameters

in	<i>channel</i>	An allocated channel.
in	<i>remotehost</i>	The remote host to connect (host name or IP).
in	<i>remoteport</i>	The remote port.
in	<i>sourcehost</i>	The numeric IP address of the machine from where the connection request originates. This is mostly for logging purposes.
in	<i>localport</i>	The port on the host from where the connection originated. This is mostly for logging purposes.

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

Warning

This function does not bind the local port and does not automatically forward the content of a socket to the channel. You still have to use channel_read and channel_write for this.

References ssh_buffer_free(), ssh_buffer_new(), ssh_string_free(), and ssh_string_from_char().

9.7.2.13 `int ssh_channel_open_session (ssh_channel channel)`

Open a session channel (suited for a shell, not TCP forwarding).

Parameters

<code>in</code>	<code>channel</code>	An allocated channel.
-----------------	----------------------	-----------------------

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

See also

`channel_open_forward()`
`channel_request_env()`
`channel_request_shell()`
`channel_request_exec()`

9.7.2.14 `int ssh_channel_poll (ssh_channel channel, int is_stderr)`

Polls a channel for data to read.

Parameters

<code>in</code>	<code>channel</code>	The channel to poll.
<code>in</code>	<code>is_stderr</code>	A boolean to select the stderr stream.

Returns

The number of bytes available for reading, 0 if nothing is available or SSH_ERROR on error.

Warning

When the channel is in EOF state, the function returns SSH_EOF.

See also

`channel_is_eof()`

Referenced by `channel_read_buffer()`, `ssh_channel_read_nonblocking()`, `ssh_scp_write()`, and `ssh_select()`.

9.7.2.15 `int ssh_channel_read (ssh_channel channel, void * dest, uint32_t count, int is_stderr)`

Reads data from a channel.

Parameters

<code>in</code>	<code>channel</code>	The channel to read from.
-----------------	----------------------	---------------------------

in	<i>dest</i>	The destination buffer which will get the data.
in	<i>count</i>	The count of bytes to be read.
in	<i>is_stderr</i>	A boolean value to mark reading from the stderr flow.

Returns

The number of bytes read, 0 on end of file or SSH_ERROR on error.

Warning

This function may return less than count bytes of data, and won't block until count bytes have been read.

The read function using a buffer has been renamed to [channel_read_buffer\(\)](#).

References [ssh_log\(\)](#), and [SSH_LOG_PROTOCOL](#).

Referenced by [channel_read_buffer\(\)](#), [ssh_channel_read_nonblocking\(\)](#), [ssh_scp_leave_directory\(\)](#), [ssh_scp_push_directory\(\)](#), [ssh_scp_push_file\(\)](#), [ssh_scp_read\(\)](#), [ssh_scp_read_string\(\)](#), and [ssh_scp_write\(\)](#).

9.7.2.16 `int ssh_channel_read_nonblocking (ssh_channel channel, void * dest, uint32_t count, int is_stderr)`

Do a nonblocking read on the channel.

A nonblocking read on the specified channel. it will return <= count bytes of data read atomically.

Parameters

in	<i>channel</i>	The channel to read from.
in	<i>dest</i>	A pointer to a destination buffer.
in	<i>count</i>	The count of bytes of data to be read.
in	<i>is_stderr</i>	A boolean to select the stderr stream.

Returns

The number of bytes read, 0 if nothing is available or SSH_ERROR on error.

Warning

Don't forget to check for EOF as it would return 0 here.

See also

[channel_is_eof\(\)](#)

References [ssh_channel_poll\(\)](#), and [ssh_channel_read\(\)](#).

9.7.2.17 `int ssh_channel_request_env (ssh_channel channel, const char * name, const char * value)`

Set environment variables.

Parameters

in	<i>channel</i>	The channel to set the environment variables.
in	<i>name</i>	The name of the variable.
in	<i>value</i>	The value to set.

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

Warning

Some environment variables may be refused by security reasons.

References `ssh_buffer_free()`, `ssh_buffer_new()`, `ssh_string_free()`, and `ssh_string_from_char()`.

9.7.2.18 `int ssh_channel_request_exec (ssh_channel channel, const char * cmd)`

Run a shell command without an interactive shell.

This is similar to 'sh -c command'.

Parameters

in	<i>channel</i>	The channel to execute the command.
in	<i>cmd</i>	The command to execute (e.g. "ls ~/ -al grep -i reports").

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

```
rc = channel_request_exec(channel, "ps aux");
if (rc > 0) {
    return -1;
}

while ((rc = channel_read(channel, buffer, sizeof(buffer), 0)) > 0) {
    if (fwrite(buffer, 1, rc, stdout) != (unsigned int) rc) {
        return -1;
    }
}
```

See also

`channel_request_shell()`

References `ssh_buffer_free()`, `ssh_buffer_new()`, `ssh_string_free()`, and `ssh_string_from_char()`.

9.7.2.19 `int ssh_channel_request_pty (ssh_channel channel)`

Request a PTY.

Parameters

<i>in</i>	<i>channel</i>	The channel to send the request.
-----------	----------------	----------------------------------

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

See also

`channel_request_pty_size()`

References `ssh_channel_request_pty_size()`.

9.7.2.20 `int ssh_channel_request_pty_size (ssh_channel channel, const char * terminal, int col, int row)`

Request a pty with a specific type and size.

Parameters

<i>in</i>	<i>channel</i>	The channel to sent the request.
<i>in</i>	<i>terminal</i>	The terminal type ("vt100, xterm,...").
<i>in</i>	<i>col</i>	The number of columns.
<i>in</i>	<i>row</i>	The number of rows.

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

References `ssh_buffer_free()`, `ssh_buffer_new()`, `ssh_string_free()`, and `ssh_string_from_char()`.

Referenced by `ssh_channel_request_pty()`.

9.7.2.21 `int ssh_channel_request_send_signal (ssh_channel channel, const char * sig)`

Send a signal to remote process (as described in RFC 4254, section 6.9).

Sends a signal 'sig' to the remote process. Note, that remote system may not support signals concept. In such a case this request will be silently ignored. Only SSH-v2 is supported (I'm not sure about SSH-v1).

OpenSSH doesn't support signals yet, see: https://bugzilla.mindrot.org/show_bug.cgi?id=1424

Parameters

in	<i>channel</i>	The channel to send signal.
in	<i>sig</i>	The signal to send (without SIG prefix) SIGABRT -> ABRT SIGALRM -> ALRM SIGFPE -> FPE SIGHUP -> HUP SIGILL -> ILL SIGINT -> INT SIGKILL -> KILL SIGPIPE -> PIPE SIGQUIT -> QUIT SIGSEGV -> SEGV SIGTERM -> TERM SIGUSR1 -> USR1 SIGUSR2 -> USR2

Returns

SSH_OK on success, SSH_ERROR if an error occurred (including attempts to send signal via SSH-v1 session).

References `ssh_buffer_free()`, `ssh_buffer_new()`, `ssh_string_free()`, and `ssh_string_from_char()`.

9.7.2.22 int ssh_channel_request_shell (ssh_channel *channel*)

Request a shell.

Parameters

in	<i>channel</i>	The channel to send the request.
----	----------------	----------------------------------

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

9.7.2.23 int ssh_channel_request_subsystem (ssh_channel *channel*, const char * *subsys*)

Request a subsystem (for example "sftp").

Parameters

in	<i>channel</i>	The channel to send the request.
in	<i>subsys</i>	The subsystem to request (for example "sftp").

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

Warning

You normally don't have to call it for sftp, see [sftp_new\(\)](#).

References [ssh_buffer_free\(\)](#), [ssh_buffer_new\(\)](#), [ssh_string_free\(\)](#), and [ssh_string_from_char\(\)](#).

9.7.2.24 `int ssh_channel_request_x11 (ssh_channel channel, int single_connection, const char * protocol, const char * cookie, int screen_number)`

Sends the "x11-req" channel request over an existing session channel.

This will enable redirecting the display of the remote X11 applications to local X server over an secure tunnel.

Parameters

in	<i>channel</i>	An existing session channel where the remote X11 applications are going to be executed.
in	<i>single_connection</i>	A boolean to mark only one X11 app will be redirected.
in	<i>protocol</i>	A x11 authentication protocol. Pass NULL to use the default value MIT-MAGIC-COOKIE-1.
in	<i>cookie</i>	A x11 authentication cookie. Pass NULL to generate a random cookie.
in	<i>screen_number</i>	The screen number.

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

References [ssh_buffer_free\(\)](#), [ssh_buffer_new\(\)](#), [ssh_string_free\(\)](#), and [ssh_string_from_char\(\)](#).

9.7.2.25 `int ssh_channel_select (ssh_channel * readchans, ssh_channel * writechans, ssh_channel * exceptchans, struct timeval * timeout)`

Act like the standard select(2) on channels.

The list of pointers are then actualized and will only contain pointers to channels that are respectively readable, writable or have an exception to trap.

Parameters

in	<i>readchans</i>	A NULL pointer or an array of channel pointers, terminated by a NULL.
in	<i>writechans</i>	A NULL pointer or an array of channel pointers, terminated by a NULL.
in	<i>exceptchans</i>	A NULL pointer or an array of channel pointers, terminated by a NULL.
in	<i>timeout</i>	Timeout as defined by select(2).

Returns

SSH_OK on a successful operation, SSH_EINTR if the select(2) syscall was interrupted, then relaunch the function.

9.7.2.26 int ssh_channel_send_eof (ssh_channel *channel*)

Send an end of file on the channel.

This doesn't close the channel. You may still read from it but not write.

Parameters

in	<i>channel</i>	The channel to send the eof to.
----	----------------	---------------------------------

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

See also

channel_close()
channel_free()

References ssh_log(), and SSH_LOG_PACKET.

Referenced by ssh_channel_close().

9.7.2.27 void ssh_channel_set_blocking (ssh_channel *channel*, int *blocking*)

Put the channel into blocking or nonblocking mode.

Parameters

in	<i>channel</i>	The channel to use.
in	<i>blocking</i>	A boolean for blocking or nonblocking.

Bug

This functionality is still under development and doesn't work correctly.

9.7.2.28 int ssh_channel_write (ssh_channel *channel*, const void * *data*, uint32_t *len*)

Blocking write on a channel.

Parameters

in	<i>channel</i>	The channel to write to.
in	<i>data</i>	A pointer to the data to write.
in	<i>len</i>	The length of the buffer to write to.

Returns

The number of bytes written, SSH_ERROR on error.

See also

channel_read()

Referenced by ssh_scp_accept_request(), ssh_scp_deny_request(), ssh_scp_leave_directory(), ssh_scp_pull_request(), ssh_scp_push_directory(), ssh_scp_push_file(), ssh_scp_read(), ssh_scp_write(), and ssh::Channel::write().

9.7.2.29 ssh_channel_ssh_forward_accept (ssh_session *session*, int *timeout_ms*)

Accept an incoming TCP/IP forwarding channel.

Parameters

in	<i>session</i>	The ssh session to use.
in	<i>timeout_ms</i>	A timeout in milliseconds.

Returns

Newly created channel, or NULL if no incoming channel request from the server

Referenced by ssh::Session::acceptForward().

9.7.2.30 int ssh_forward_cancel (ssh_session *session*, const char * *address*, int *port*)

Sends the "cancel-tcpip-forward" global request to ask the server to cancel the tcpip-forward request.

Parameters

in	<i>session</i>	The ssh session to send the request.
in	<i>address</i>	The bound address on the server.
in	<i>port</i>	The bound port on the server.

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

References ssh_buffer_free(), ssh_buffer_new(), ssh_string_free(), and ssh_string_from_char().

9.7.2.31 int ssh_forward_listen (ssh_session *session*, const char * *address*, int *port*, int * *bound_port*)

Sends the "tcpip-forward" global request to ask the server to begin listening for inbound connections.

Parameters

in	<i>session</i>	The ssh session to send the request.
in	<i>address</i>	The address to bind to on the server. Pass NULL to bind to all available addresses on all protocol families supported by the server.
in	<i>port</i>	The port to bind to on the server. Pass 0 to ask the server to allocate the next available unprivileged port number
in	<i>bound_port</i>	The pointer to get actual bound port. Pass NULL to ignore.

Returns

SSH_OK on success, SSH_ERROR if an error occurred.

References `ssh_buffer_free()`, `ssh_buffer_new()`, `ssh_string_free()`, and `ssh_string_from_char()`.

9.8 The SSH error functions.

Functions for error handling.

Functions

- `const char * ssh_get_error (void *error)`
Retrieve the error text message from the last error.
- `int ssh_get_error_code (void *error)`
Retrieve the error code from the last error.

9.8.1 Detailed Description

Functions for error handling.

9.8.2 Function Documentation**9.8.2.1 `const char* ssh_get_error (void * error)`**

Retrieve the error text message from the last error.

Parameters

<i>error</i>	The SSH session pointer.
--------------	--------------------------

Returns

A static string describing the error.

Referenced by `ssh_scp_leave_directory()`, `ssh_scp_push_directory()`, `ssh_scp_push_file()`, and `ssh_try_publickey_from_file()`.

9.8.2.2 `int ssh_get_error_code (void * error)`

Retrieve the error code from the last error.

Parameters

<i>error</i>	The SSH session pointer.
--------------	--------------------------

Returns

`SSH_NO_ERROR` No error occurred
`SSH_REQUEST_DENIED` The last request was denied but situation is recoverable
`SSH_FATAL` A fatal error occurred. This could be an unexpected disconnection

Other error codes are internal but can be considered same than `SSH_FATAL`.

9.9 The libssh API

The libssh library is implementing the SSH protocols and some of its extensions.

Modules

- [The libssh callbacks](#)
Callback which can be replaced in libssh.
- [The SSH authentication functions.](#)
Functions to authenticate with a server.
- [The SSH buffer functions.](#)
Functions to handle SSH buffers.
- [The SSH channel functions](#)
Functions that manage a SSH channel.
- [The SSH error functions.](#)
Functions for error handling.
- [The SSH logging functions.](#)
Logging functions for debugging and problem resolving.
- [The SSH message functions](#)
This file contains the Message parsing utilities for server programs using libssh.

- [The SSH helper functions.](#)

Different helper functions used in the SSH Library.

- [The SSH Public Key Infrastructure](#)

Functions for the creation, importation and manipulation of public and private keys in the context of the SSH protocol.

- [The SSH poll functions.](#)

Add a generic way to handle sockets asynchronously.

- [The SSH scp functions](#)

SCP protocol over SSH functions.

- [The SSH session functions.](#)

Functions that manage a session.

- [The SSH string functions](#)

String manipulations used in libssh.

- [The SSH threading functions.](#)

Threading with libssh.

Functions

- int [ssh_finalize](#) (void)

Finalize and cleanup all libssh and cryptographic data structures.

- int [ssh_init](#) (void)

Initialize global cryptographic data structures.

9.9.1 Detailed Description

The libssh library is implementing the SSH protocols and some of its extensions. This group of functions is mostly used to implement a SSH client. Some function are needed to implement a SSH server too.

9.9.2 Function Documentation

9.9.2.1 int [ssh_finalize](#) (void)

Finalize and cleanup all libssh and cryptographic data structures.

This function should only be called once, at the end of the program!

Returns

0 on succes, -1 if an error occured.
0 otherwise

9.9.2.2 int ssh_init (void)

Initialize global cryptographic data structures.

This function should only be called once, at the beginning of the program, in the main thread. It may be omitted if your program is not multithreaded.

Returns

0 on success, -1 if an error occured.

Referenced by privatekey_from_file(), ssh_bind_listen(), and ssh_connect().

9.10 The SSH logging functions.

Logging functions for debugging and problem resolving.

Enumerations

- enum {
SSH_LOG_NOLOG = 0, SSH_LOG_RARE, SSH_LOG_PROTOCOL, SSH_LOG_PACKET,
SSH_LOG_FUNCTIONS }

Verbosity level for logging and help to debugging.

Functions

- void [ssh_log](#) (ssh_session session, int verbosity, const char *format,...)
Log a SSH event.

9.10.1 Detailed Description

Logging functions for debugging and problem resolving.

9.10.2 Enumeration Type Documentation

9.10.2.1 anonymous enum

Verbosity level for logging and help to debugging.

Enumerator:

SSH_LOG_NOLOG No logging at all.

SSH_LOG_RARE Only rare and noteworthy events.

SSH_LOG_PROTOCOL High level protocol information.

SSH_LOG_PACKET Lower level protocol informations, packet level.

SSH_LOG_FUNCTIONS Every function path.

9.10.3 Function Documentation

9.10.3.1 void ssh_log (ssh_session session, int verbosity, const char * format, ...)

Log a SSH event.

Parameters

<i>session</i>	The SSH session.
<i>verbosity</i>	The verbosity of the event.
<i>format</i>	The format string of the log entry.

References SSH_LOG_FUNCTIONS.

Referenced by privatekey_from_file(), ssh_channel_close(), ssh_channel_read(), ssh_channel_send_eof(), ssh_connect(), ssh_handle_key_exchange(), ssh_publickey_to_file(), ssh_scp_pull_request(), ssh_scp_push_file(), ssh_try_publickey_from_file(), ssh_userauth_autopubkey(), and ssh_userauth_privatekey_file().

9.11 The SSH message functions

This file contains the Message parsing utilities for server programs using libssh.

Functions

- ssh_message [ssh_message_get](#) (ssh_session session)

Retrieve a SSH message from a SSH session.

9.11.1 Detailed Description

This file contains the Message parsing utilities for server programs using libssh. The main loop of the program will call `ssh_message_get(session)` to get messages as they come. they are not 1-1 with the protocol messages. then, the user will know what kind of a message it is and use the appropriate functions to handle it (or use the default handlers if she doesn't know what to do

9.11.2 Function Documentation

9.11.2.1 `ssh_message` `ssh_message_get (ssh_session session)`

Retrieve a SSH message from a SSH session.

Parameters

<code>in</code>	<code>session</code>	The SSH session to get the message.
-----------------	----------------------	-------------------------------------

Returns

The SSH message received, NULL in case of error.

Warning

This function blocks until a message has been received. Better set up a callback if this behavior is unwanted.

9.12 The SSH helper functions.

Different helper functions used in the SSH Library.

Functions

- `char * ssh_basename (const char *path)`
basename - parse filename component.
- `char * ssh_dirname (const char *path)`
Parse directory component.
- `int ssh_getpass (const char *prompt, char *buf, size_t len, int echo, int verify)`
Get a password from the console.
- `int ssh_mkdir (const char *pathname, mode_t mode)`
Attempts to create a directory with the given pathname.
- `char * ssh_path_expand_tilde (const char *d)`
Expand a directory starting with a tilde '~'.

- int [ssh_timeout_update](#) (struct ssh_timestamp *ts, int timeout)
updates a timeout value so it reflects the remaining time
- const char * [ssh_version](#) (int req_version)
Check if libssh is the required version or get the version string.

9.12.1 Detailed Description

Different helper functions used in the SSH Library.

9.12.2 Function Documentation

9.12.2.1 char* ssh_basename (const char * *path*)

basename - parse filename component.

basename breaks a null-terminated pathname string into a filename component. [ssh_basename\(\)](#) returns the component following the final '/'. Trailing '/' characters are not counted as part of the pathname.

Parameters

in	<i>path</i>	The path to parse.
----	-------------	--------------------

Returns

The filename of path or NULL if we can't allocate memory. If path is a the string "/", basename returns the string "/". If path is NULL or an empty string, "." is returned.

Referenced by ssh_scp_push_directory(), and ssh_scp_push_file().

9.12.2.2 char* ssh_dirname (const char * *path*)

Parse directory component.

dirname breaks a null-terminated pathname string into a directory component. In the usual case, [ssh_dirname\(\)](#) returns the string up to, but not including, the final '/'. Trailing '/' characters are not counted as part of the pathname. The caller must free the memory.

Parameters

in	<i>path</i>	The path to parse.
----	-------------	--------------------

Returns

The dirname of path or NULL if we can't allocate memory. If path does not contain a slash, `c_dirname()` returns the string ".". If path is the string "/", it returns the string "/". If path is NULL or an empty string, "." is returned.

Referenced by `ssh_write_knownhost()`.

9.12.2.3 int ssh_getpass (const char * *prompt*, char * *buf*, size_t *len*, int *echo*, int *verify*)

Get a password from the console.

You should make sure that the buffer is an empty string!

You can also use this function to ask for a username. Then you can fill the buffer with the username and it is shown to the users. If the user just presses enter the buffer will be untouched.

```
char username[128];

snprintf(username, sizeof(username), "john");

ssh_getpass("Username:", username, sizeof(username), 1, 0);
```

The prompt will look like this:

Username: [john]

If you press enter then john is used as the username, or you can type it in to change it.

Parameters

in	<i>prompt</i>	The prompt to show to ask for the password.
out	<i>buf</i>	The buffer the password should be stored. It NEEDS to be empty or filled out.
in	<i>len</i>	The length of the buffer.
in	<i>echo</i>	Should we echo what you type.
in	<i>verify</i>	Should we ask for the password twice.

Returns

0 on success, -1 on error.

9.12.2.4 int ssh_mkdir (const char * *pathname*, mode_t *mode*)

Attempts to create a directory with the given pathname.

This is the portable version of `mkdir`, mode is ignored on Windows systems.

Parameters

in	<i>pathname</i>	The path name to create the directory.
in	<i>mode</i>	The permissions to use.

Returns

0 on success, < 0 on error with errno set.

Referenced by ssh_write_knownhost().

9.12.2.5 char* ssh_path_expand_tilde (const char * *d*)

Expand a directory starting with a tilde '~'.

Parameters

in	<i>d</i>	The directory to expand.
----	----------	--------------------------

Returns

The expanded directory, NULL on error.

Referenced by ssh_options_set().

9.12.2.6 int ssh_timeout_update (struct ssh_timestamp * *ts*, int *timeout*)

updates a timeout value so it reflects the remaining time

Parameters

in	<i>ts</i>	pointer to an existing timestamp
in	<i>timeout</i>	timeout in milliseconds. Negative values mean infinite timeout

Returns

remaining time in milliseconds, 0 if elapsed, -1 if never, -2 if option-set-timeout.

Referenced by ssh_blocking_flush().

9.12.2.7 const char* ssh_version (int *req_version*)

Check if libssh is the required version or get the version string.

Parameters

in	<i>req_version</i>	The version required.
----	--------------------	-----------------------

Returns

If the version of libssh is newer than the version required it will return a version string. NULL if the version is older.

Example:

```
if (ssh_version(SSH_VERSION_INT(0,2,1)) == NULL) {
```

```

    fprintf(stderr, "libssh version is too old!\n");
    exit(1);
}

if (debug) {
    printf("libssh %s\n", ssh_version(0));
}

```

9.13 The SSH Public Key Infrastructure

Functions for the creation, importation and manipulation of public and private keys in the context of the SSH protocol.

Functions

- void [ssh_key_clean](#) (ssh_key key)
clean up the key and deallocate all existing keys
- void [ssh_key_free](#) (ssh_key key)
deallocate a SSH key
- int [ssh_key_import_private](#) (ssh_key key, ssh_session session, const char *filename, const char *passphrase)
import a key from a file
- ssh_key [ssh_key_new](#) (void)
creates a new empty SSH key
- enum ssh_keytypes_e [ssh_key_type](#) (ssh_key key)
returns the type of a ssh key

9.13.1 Detailed Description

Functions for the creation, importation and manipulation of public and private keys in the context of the SSH protocol.

9.13.2 Function Documentation

9.13.2.1 void [ssh_key_clean](#) (ssh_key key)

clean up the key and deallocate all existing keys

Parameters

in	key	ssh_key to clean
----	-----	------------------

Referenced by `ssh_key_free()`, and `ssh_key_import_private()`.

9.13.2.2 void `ssh_key_free` (`ssh_key key`)

deallocate a SSH key

Parameters

in	<i>key</i>	ssh_key handle to free
----	------------	------------------------

References `ssh_key_clean()`.

9.13.2.3 int `ssh_key_import_private` (`ssh_key key`, `ssh_session session`, `const char * filename`, `const char * passphrase`)

import a key from a file

Parameters

out	<i>key</i>	the <code>ssh_key</code> to update
in	<i>session</i>	The SSH Session to use. If a key decryption callback is set, it will be used to ask for the passphrase.
in	<i>filename</i>	The filename of the the private key.
in	<i>passphrase</i>	The passphrase to decrypt the private key. Set to null if none is needed or it is unknown.

Returns

SSH_OK on success, SSH_ERROR otherwise.

References `privatekey_from_file()`, and `ssh_key_clean()`.

9.13.2.4 `ssh_key` `ssh_key_new` (void)

creates a new empty SSH key

Returns

an empty `ssh_key` handle, or NULL on error.

9.13.2.5 enum `ssh_keytypes_e` `ssh_key_type` (`ssh_key key`)

returns the type of a ssh key

Parameters

in	<i>key</i>	the <code>ssh_key</code> handle
----	------------	---------------------------------

Returns

one of SSH_KEYTYPE_RSA, SSH_KEYTYPE_DSS, SSH_KEYTYPE_RSA1
SSH_KEYTYPE_UNKNOWN if the type is unknown

9.14 The SSH poll functions.

Add a generic way to handle sockets asynchronously.

Functions

- void [ssh_poll_add_events](#) (ssh_poll_handle p, short events)
Add extra events to a poll object.
- int [ssh_poll_ctx_add](#) (ssh_poll_ctx ctx, ssh_poll_handle p)
Add a poll object to a poll context.
- int [ssh_poll_ctx_add_socket](#) (ssh_poll_ctx ctx, ssh_socket s)
Add a socket object to a poll context.
- int [ssh_poll_ctx_dopoll](#) (ssh_poll_ctx ctx, int timeout)
Poll all the sockets associated through a poll object with a poll context.
- void [ssh_poll_ctx_free](#) (ssh_poll_ctx ctx)
Free a poll context.
- ssh_poll_ctx [ssh_poll_ctx_new](#) (size_t chunk_size)
Create a new poll context.
- void [ssh_poll_ctx_remove](#) (ssh_poll_ctx ctx, ssh_poll_handle p)
Remove a poll object from a poll context.
- void [ssh_poll_free](#) (ssh_poll_handle p)
Free a poll object.
- ssh_poll_ctx [ssh_poll_get_ctx](#) (ssh_poll_handle p)
Get the poll context of a poll object.
- short [ssh_poll_get_events](#) (ssh_poll_handle p)
Get the events of a poll object.
- socket_t [ssh_poll_get_fd](#) (ssh_poll_handle p)
Get the raw socket of a poll object.
- ssh_poll_handle [ssh_poll_new](#) (socket_t fd, short events, ssh_poll_callback cb, void *userdata)

Allocate a new poll object, which could be used within a poll context.

- void [ssh_poll_remove_events](#) (ssh_poll_handle p, short events)
Remove events from a poll object.
- void [ssh_poll_set_callback](#) (ssh_poll_handle p, ssh_poll_callback cb, void *userdata)

Set the callback of a poll object.

- void [ssh_poll_set_events](#) (ssh_poll_handle p, short events)
Set the events of a poll object.
- void [ssh_poll_set_fd](#) (ssh_poll_handle p, socket_t fd)
Set the file descriptor of a poll object.

9.14.1 Detailed Description

Add a generic way to handle sockets asynchronously. It's based on poll objects, each of which store a socket, its events and a callback, which gets called whenever an event is set. The poll objects are attached to a poll context, which should be allocated on per thread basis.

Polling the poll context will poll all the attached poll objects and call their callbacks (handlers) if any of the socket events are set. This should be done within the main loop of an application.

9.14.2 Function Documentation

9.14.2.1 void ssh_poll_add_events (ssh_poll_handle p, short events)

Add extra events to a poll object.

Duplicates are ignored. The events will also be propagated to an associated poll context.

Parameters

<i>p</i>	Pointer to an already allocated poll object.
<i>events</i>	Poll events.

References [ssh_poll_get_events\(\)](#), and [ssh_poll_set_events\(\)](#).

9.14.2.2 int ssh_poll_ctx_add (ssh_poll_ctx ctx, ssh_poll_handle p)

Add a poll object to a poll context.

Parameters

<i>ctx</i>	Pointer to an already allocated poll context.
<i>p</i>	Pointer to an already allocated poll object.

Returns

0 on success, < 0 on error

Referenced by `ssh_poll_ctx_add_socket()`.

9.14.2.3 int ssh_poll_ctx_add_socket (ssh_poll_ctx ctx, ssh_socket s)

Add a socket object to a poll context.

Parameters

<i>ctx</i>	Pointer to an already allocated poll context.
<i>s</i>	A SSH socket handle

Returns

0 on success, < 0 on error

References `ssh_poll_ctx_add()`.

9.14.2.4 int ssh_poll_ctx_dopoll (ssh_poll_ctx ctx, int timeout)

Poll all the sockets associated through a poll object with a poll context.

If any of the events are set after the poll, the call back function of the socket will be called. This function should be called once within the programs main loop.

Parameters

<i>ctx</i>	Pointer to an already allocated poll context.
<i>timeout</i>	An upper limit on the time for which <code>ssh_poll_ctx()</code> will block, in milliseconds. Specifying a negative value means an infinite timeout. This parameter is passed to the <code>poll()</code> function.

Returns

SSH_OK No error. SSH_ERROR Error happened during the poll. SSH_AGAIN
Timeout occurred

9.14.2.5 void ssh_poll_ctx_free (ssh_poll_ctx ctx)

Free a poll context.

Parameters

<i>ctx</i>	Pointer to an already allocated poll context.
------------	---

References `ssh_poll_ctx_remove()`.

Referenced by `ssh_free()`.

9.14.2.6 `ssh_poll_ctx ssh_poll_ctx_new (size_t chunk_size)`

Create a new poll context.

It could be associated with many poll object which are going to be polled at the same time as the poll context. You would need a single poll context per thread.

Parameters

<i>chunk_size</i>	The size of the memory chunk that will be allocated, when more memory is needed. This is for efficiency reasons, i.e. don't allocate memory for each new poll object, but for the next 5. Set it to 0 if you want to use the library's default value.
-------------------	---

9.14.2.7 `void ssh_poll_ctx_remove (ssh_poll_ctx ctx, ssh_poll_handle p)`

Remove a poll object from a poll context.

Parameters

<i>ctx</i>	Pointer to an already allocated poll context.
<i>p</i>	Pointer to an already allocated poll object.

Referenced by `ssh_poll_ctx_free()`, and `ssh_poll_free()`.

9.14.2.8 `void ssh_poll_free (ssh_poll_handle p)`

Free a poll object.

Parameters

<i>p</i>	Pointer to an already allocated poll object.
----------	--

References `ssh_poll_ctx_remove()`.

9.14.2.9 `ssh_poll_ctx ssh_poll_get_ctx (ssh_poll_handle p)`

Get the poll context of a poll object.

Parameters

<i>p</i>	Pointer to an already allocated poll object.
----------	--

Returns

Poll context or NULL if the poll object isn't attached.

9.14.2.10 short ssh_poll_get_events (ssh_poll_handle *p*)

Get the events of a poll object.

Parameters

<i>p</i>	Pointer to an already allocated poll object.
----------	--

Returns

Poll events.

Referenced by ssh_poll_add_events(), and ssh_poll_remove_events().

9.14.2.11 socket_t ssh_poll_get_fd (ssh_poll_handle *p*)

Get the raw socket of a poll object.

Parameters

<i>p</i>	Pointer to an already allocated poll object.
----------	--

Returns

Raw socket.

9.14.2.12 ssh_poll_handle ssh_poll_new (socket_t *fd*, short *events*, ssh_poll_callback *cb*, void * *userdata*)

Allocate a new poll object, which could be used within a poll context.

Parameters

<i>fd</i>	Socket that will be polled.
<i>events</i>	Poll events that will be monitored for the socket. i.e. POLLIN, POLLPRI, POLLOUT
<i>cb</i>	Function to be called if any of the events are set.
<i>userdata</i>	Userdata to be passed to the callback function. NULL if not needed.

Returns

A new poll object, NULL on error

9.14.2.13 void ssh_poll_remove_events (ssh_poll_handle *p*, short *events*)

Remove events from a poll object.

Non-existent are ignored. The events will also be propagated to an associated poll context.

Parameters

<i>p</i>	Pointer to an already allocated poll object.
<i>events</i>	Poll events.

References ssh_poll_get_events(), and ssh_poll_set_events().

9.14.2.14 void ssh_poll_set_callback (ssh_poll_handle *p*, ssh_poll_callback *cb*, void * *userdata*)

Set the callback of a poll object.

Parameters

<i>p</i>	Pointer to an already allocated poll object.
<i>cb</i>	Function to be called if any of the events are set.
<i>userdata</i>	Userdata to be passed to the callback function. NULL if not needed.

9.14.2.15 void ssh_poll_set_events (ssh_poll_handle *p*, short *events*)

Set the events of a poll object.

The events will also be propagated to an associated poll context.

Parameters

<i>p</i>	Pointer to an already allocated poll object.
<i>events</i>	Poll events.

Referenced by ssh_poll_add_events(), and ssh_poll_remove_events().

9.14.2.16 void ssh_poll_set_fd (ssh_poll_handle *p*, socket_t *fd*)

Set the file descriptor of a poll object.

The FD will also be propagated to an associated poll context.

Parameters

<i>p</i>	Pointer to an already allocated poll object.
<i>fd</i>	New file descriptor.

9.15 The SSH scp functions

SCP protocol over SSH functions.

Functions

- `int ssh_scp_accept_request (ssh_scp scp)`
Accepts transfer of a file or creation of a directory coming from the remote party.
- `int ssh_scp_deny_request (ssh_scp scp, const char *reason)`
Deny the transfer of a file or creation of a directory coming from the remote party.
- `int ssh_scp_integer_mode (const char *mode)`
Convert a scp text mode to an integer.
- `int ssh_scp_leave_directory (ssh_scp scp)`
Leave a directory.
- `ssh_scp ssh_scp_new (ssh_session session, int mode, const char *location)`
Create a new scp session.
- `int ssh_scp_pull_request (ssh_scp scp)`
Wait for a scp request (file, directory).
- `int ssh_scp_push_directory (ssh_scp scp, const char *dirname, int mode)`
Create a directory in a scp in sink mode.
- `int ssh_scp_push_file (ssh_scp scp, const char *filename, size_t size, int mode)`
Initialize the sending of a file to a scp in sink mode.
- `int ssh_scp_read (ssh_scp scp, void *buffer, size_t size)`
Read from a remote scp file.
- `int ssh_scp_read_string (ssh_scp scp, char *buffer, size_t len)`
Read a string on a channel, terminated by ' '.
- `const char * ssh_scp_request_get_filename (ssh_scp scp)`
Get the name of the directory or file being pushed from the other party.
- `int ssh_scp_request_get_permissions (ssh_scp scp)`
Get the permissions of the directory or file being pushed from the other party.
- `size_t ssh_scp_request_get_size (ssh_scp scp)`
Get the size of the file being pushed from the other party.

- `const char * ssh_scp_request_get_warning (ssh_scp scp)`
Get the warning string from a scp handle.
- `char * ssh_scp_string_mode (int mode)`
Convert a unix mode into a scp string.
- `int ssh_scp_write (ssh_scp scp, const void *buffer, size_t len)`
Write into a remote scp file.

9.15.1 Detailed Description

SCP protocol over SSH functions.

9.15.2 Function Documentation

9.15.2.1 `int ssh_scp_accept_request (ssh_scp scp)`

Accepts transfer of a file or creation of a directory coming from the remote party.

Parameters

<code>in</code>	<code>scp</code>	The scp handle.
-----------------	------------------	-----------------

Returns

SSH_OK if the message was sent, SSH_ERROR if sending the message failed, or sending it in a bad state.

References `ssh_channel_write()`.

Referenced by `ssh_scp_read()`.

9.15.2.2 `int ssh_scp_deny_request (ssh_scp scp, const char * reason)`

Deny the transfer of a file or creation of a directory coming from the remote party.

Parameters

<code>in</code>	<code>scp</code>	The scp handle.
<code>in</code>	<code>reason</code>	A nul-terminated string with a human-readable explanation of the deny.

Returns

SSH_OK if the message was sent, SSH_ERROR if the sending the message failed, or sending it in a bad state.

References `ssh_channel_write()`.

9.15.2.3 int ssh_scp_integer_mode (const char * mode)

Convert a scp text mode to an integer.

Parameters

<i>in</i>	<i>mode</i>	The mode to convert, e.g. "0644".
-----------	-------------	-----------------------------------

Returns

An integer value, e.g. 420 for "0644".

Referenced by ssh_scp_pull_request().

9.15.2.4 int ssh_scp_leave_directory (ssh_scp scp)

Leave a directory.

Returns

SSH_OK if the directory has been left, SSH_ERROR if an error occurred.

See also

[ssh_scp_push_directory\(\)](#)

References ssh_channel_read(), ssh_channel_write(), and ssh_get_error().

9.15.2.5 ssh_scp ssh_scp_new (ssh_session session, int mode, const char * location)

Create a new scp session.

Parameters

<i>in</i>	<i>session</i>	The SSH session to use.
<i>in</i>	<i>mode</i>	One of SSH_SCP_WRITE or SSH_SCP_READ, depending if you need to drop files remotely or read them. It is not possible to combine read and write.
<i>in</i>	<i>location</i>	The directory in which write or read will be done. Any push or pull will be relative to this place.

Returns

A ssh_scp handle, NULL if the creation was impossible.

9.15.2.6 int ssh_scp_pull_request (ssh_scp scp)

Wait for a scp request (file, directory).

Returns

SSH_SCP_REQUEST_NEWFILE: The other side is sending a file
 SSH_SCP_REQUEST_NEWDIR: The other side is sending a directory
 SSH_SCP_REQUEST_ENDDIR: The other side has finished with the current directory
 SSH_SCP_REQUEST_WARNING: The other side sent us a warning
 SSH_SCP_REQUEST_EOF: The other side finished sending us files and data.
 SSH_ERROR: Some error happened

See also

[ssh_scp_read\(\)](#)
[ssh_scp_deny_request\(\)](#)
[ssh_scp_accept_request\(\)](#)
[ssh_scp_request_get_warning\(\)](#)

References [ssh_channel_is_eof\(\)](#), [ssh_channel_write\(\)](#), [ssh_log\(\)](#), [SSH_LOG_PROTOCOL](#), [ssh_scp_integer_mode\(\)](#), and [ssh_scp_read_string\(\)](#).

9.15.2.7 int ssh_scp_push_directory (ssh_scp scp, const char * dirname, int mode)

Create a directory in a scp in sink mode.

Parameters

in	<i>scp</i>	The scp handle.
in	<i>dirname</i>	The name of the directory being created.
in	<i>mode</i>	The UNIX permissions for the new directory, e.g. 0755.

Returns

SSH_OK if the directory has been created, SSH_ERROR if an error occurred.

See also

[ssh_scp_leave_directory\(\)](#)

References [ssh_basename\(\)](#), [ssh_channel_read\(\)](#), [ssh_channel_write\(\)](#), [ssh_get_error\(\)](#), and [ssh_scp_string_mode\(\)](#).

9.15.2.8 int ssh_scp_push_file (ssh_scp scp, const char * filename, size_t size, int mode)

Initialize the sending of a file to a scp in sink mode.

Parameters

in	<i>scp</i>	The scp handle.
in	<i>filename</i>	The name of the file being sent. It should not contain any path indicator
in	<i>size</i>	Exact size in bytes of the file being sent.
in	<i>mode</i>	The UNIX permissions for the new file, e.g. 0644.

Returns

SSH_OK if the file is ready to be sent, SSH_ERROR if an error occurred.

References ssh_basename(), ssh_channel_read(), ssh_channel_write(), ssh_get_error(), ssh_log(), SSH_LOG_PROTOCOL, and ssh_scp_string_mode().

9.15.2.9 int ssh_scp_read (ssh_scp scp, void * buffer, size_t size)

Read from a remote scp file.

Parameters

in	<i>scp</i>	The scp handle.
in	<i>buffer</i>	The destination buffer.
in	<i>size</i>	The size of the buffer.

Returns

The nNumber of bytes read, SSH_ERROR if an error occurred while reading.

References ssh_channel_read(), ssh_channel_write(), and ssh_scp_accept_request().

9.15.2.10 int ssh_scp_read_string (ssh_scp scp, char * buffer, size_t len)

Read a string on a channel, terminated by ' '.

Parameters

in	<i>scp</i>	The scp handle.
out	<i>buffer</i>	A pointer to a buffer to place the string.
in	<i>len</i>	The size of the buffer in bytes. If the string is bigger than len-1, only len-1 bytes are read and the string is null-terminated.

Returns

SSH_OK if the string was read, SSH_ERROR if an error occurred while reading.

References ssh_channel_read().

Referenced by ssh_scp_pull_request().

9.15.2.11 const char* ssh_scp_request_get_filename (ssh_scp scp)

Get the name of the directory or file being pushed from the other party.

Returns

The file name, NULL on error. The string should not be freed.

9.15.2.12 `int ssh_scp_request_get_permissions (ssh_scp scp)`

Get the permissions of the directory or file being pushed from the other party.

Returns

The UNIX permission, e.g 0644, -1 on error.

9.15.2.13 `size_t ssh_scp_request_get_size (ssh_scp scp)`

Get the size of the file being pushed from the other party.

Returns

The numeric size of the file being read.

9.15.2.14 `const char* ssh_scp_request_get_warning (ssh_scp scp)`

Get the warning string from a scp handle.

Parameters

<i>in</i>	<i>scp</i>	The scp handle.
-----------	------------	-----------------

Returns

A warning string, or NULL on error. The string should not be freed.

9.15.2.15 `char* ssh_scp_string_mode (int mode)`

Convert a unix mode into a scp string.

Parameters

<i>in</i>	<i>mode</i>	The mode to convert, e.g. 420 or 0644.
-----------	-------------	--

Returns

A pointer to a malloc'ed string containing the scp mode, e.g. "0644".

Referenced by `ssh_scp_push_directory()`, and `ssh_scp_push_file()`.

9.15.2.16 `int ssh_scp_write (ssh_scp scp, const void * buffer, size_t len)`

Write into a remote scp file.

Parameters

in	<i>scp</i>	The scp handle.
in	<i>buffer</i>	The buffer to write.
in	<i>len</i>	The number of bytes to write.

Returns

SSH_OK if the write was successful, SSH_ERROR an error occurred while writing.

References `ssh_channel_poll()`, `ssh_channel_read()`, and `ssh_channel_write()`.

9.16 The SSH session functions.

Functions that manage a session.

Functions

- int `ssh_blocking_flush` (ssh_session session, int timeout)
Blocking flush of the outgoing buffer.
- void `ssh_clean_pubkey_hash` (unsigned char **hash)
Deallocate the hash obtained by `ssh_get_pubkey_hash`.
- int `ssh_connect` (ssh_session session)
Connect to the ssh server.
- void `ssh_disconnect` (ssh_session session)
Disconnect from a session (client or server).
- void `ssh_free` (ssh_session session)
Deallocate a SSH session handle.
- const char * `ssh_get_disconnect_message` (ssh_session session)
Get the disconnect message from the server.
- socket_t `ssh_get_fd` (ssh_session session)
Get the fd of a connection.
- char * `ssh_get_issue_banner` (ssh_session session)
Get the issue banner from the server.
- int `ssh_get_openssh_version` (ssh_session session)
Get the version of the OpenSSH server, if it is not an OpenSSH server then 0 will be returned.

- int [ssh_get_pubkey_hash](#) (ssh_session session, unsigned char **hash)
Allocates a buffer with the MD5 hash of the server public key.
- int [ssh_get_status](#) (ssh_session session)
Get session status.
- int [ssh_get_version](#) (ssh_session session)
Get the protocol version of the session.
- int [ssh_is_blocking](#) (ssh_session session)
Return the blocking mode of libssh.
- int [ssh_is_connected](#) (ssh_session session)
Check if we are connected.
- int [ssh_is_server_known](#) (ssh_session session)
Check if the server is known.
- ssh_session [ssh_new](#) (void)
Create a new ssh session.
- int [ssh_options_copy](#) (ssh_session src, ssh_session *dest)
Duplicate the options of a session structure.
- int [ssh_options_getopt](#) (ssh_session session, int *argcptr, char **argv)
Parse command line arguments.
- int [ssh_options_parse_config](#) (ssh_session session, const char *filename)
Parse the ssh config file.
- int [ssh_options_set](#) (ssh_session session, enum ssh_options_e type, const void *value)
This function can set all possible ssh options.
- int [ssh_select](#) (ssh_channel *channels, ssh_channel *outchannels, socket_t maxfd, fd_set *readfds, struct timeval *timeout)
A wrapper for the select syscall.
- void [ssh_set_blocking](#) (ssh_session session, int blocking)
Set the session in blocking/nonblocking mode.
- void [ssh_set_fd_except](#) (ssh_session session)
Tell the session it has an exception to catch on the file descriptor.
- void [ssh_set_fd_toread](#) (ssh_session session)
Tell the session it has data to read on the file descriptor without blocking.

- void [ssh_set_fd_towrite](#) (ssh_session session)
Tell the session it may write to the file descriptor without blocking.
- void [ssh_silent_disconnect](#) (ssh_session session)
Disconnect impolitely from a remote host by closing the socket.
- int [ssh_write_knownhost](#) (ssh_session session)
Write the current server as known in the known hosts file.

9.16.1 Detailed Description

Functions that manage a session.

9.16.2 Function Documentation

9.16.2.1 int [ssh_blocking_flush](#) (ssh_session session, int timeout)

Blocking flush of the outgoing buffer.

Parameters

in	<i>session</i>	The SSH session
in	<i>timeout</i>	Set an upper limit on the time for which this function will block, in milliseconds. Specifying a negative value means an infinite timeout. This parameter is passed to the poll() function.

Returns

SSH_OK on success, SSH_AGAIN if timeout occurred, SSH_ERROR otherwise.

References [ssh_timeout_update](#)().

9.16.2.2 void [ssh_clean_pubkey_hash](#) (unsigned char ** hash)

Deallocate the hash obtained by [ssh_get_pubkey_hash](#).

This is required under Microsoft platform as this library might use a different C library than your software, hence a different heap.

Parameters

in	<i>hash</i>	The buffer to deallocate.
----	-------------	---------------------------

See also

[ssh_get_pubkey_hash\(\)](#)

9.16.2.3 int ssh_connect (ssh_session session)

Connect to the ssh server.

Parameters

in	session	The ssh session to connect.
----	---------	-----------------------------

Returns

SSH_OK on success, SSH_ERROR on error.

SSH_AGAIN, if the session is in nonblocking mode, and call must be done again.

See also

[ssh_new\(\)](#)

[ssh_disconnect\(\)](#)

References [ssh_init\(\)](#), [ssh_is_blocking\(\)](#), [ssh_log\(\)](#), [SSH_LOG_PACKET](#), [SSH_LOG_PROTOCOL](#), and [SSH_LOG_RARE](#).

Referenced by [ssh::Session::connect\(\)](#).

9.16.2.4 void ssh_disconnect (ssh_session session)

Disconnect from a session (client or server).

The session can then be reused to open a new session.

Parameters

in	session	The SSH session to use.
----	---------	-------------------------

References [ssh_channel_free\(\)](#), [ssh_string_free\(\)](#), and [ssh_string_from_char\(\)](#).

Referenced by [ssh::Session::disconnect\(\)](#), and [ssh_silent_disconnect\(\)](#).

9.16.2.5 void ssh_free (ssh_session session)

Deallocate a SSH session handle.

Parameters

in	session	The SSH session to free.
----	---------	--------------------------

See also

[ssh_disconnect\(\)](#)

[ssh_new\(\)](#)

References [privatekey_free\(\)](#), [ssh_buffer_free\(\)](#), [ssh_channel_free\(\)](#), and [ssh_poll_ctx_free\(\)](#).

Referenced by `ssh_new()`.

9.16.2.6 `const char* ssh_get_disconnect_message (ssh_session session)`

Get the disconnect message from the server.

Parameters

<i>in</i>	<i>session</i>	The ssh session to use.
-----------	----------------	-------------------------

Returns

The message sent by the server along with the disconnect, or NULL in which case the reason of the disconnect may be found with `ssh_get_error`.

See also

[ssh_get_error\(\)](#)

Referenced by `ssh::Session::getDisconnectMessage()`.

9.16.2.7 `socket_t ssh_get_fd (ssh_session session)`

Get the fd of a connection.

In case you'd need the file descriptor of the connection to the server/client.

Parameters

<i>in</i>	<i>session</i>	The ssh session to use.
-----------	----------------	-------------------------

Returns

The file descriptor of the connection, or -1 if it is not connected

Referenced by `ssh::Session::getSocket()`.

9.16.2.8 `char* ssh_get_issue_banner (ssh_session session)`

Get the issue banner from the server.

This is the banner showing a disclaimer to users who log in, typically their right or the fact that they will be monitored.

Parameters

<i>in</i>	<i>session</i>	The SSH session to use.
-----------	----------------	-------------------------

Returns

A newly allocated string with the banner, NULL on error.

References `ssh_string_to_char()`.

Referenced by `ssh::Session::getIssueBanner()`.

9.16.2.9 `int ssh_get_openssh_version (ssh_session session)`

Get the version of the OpenSSH server, if it is not an OpenSSH server then 0 will be returned.

You can use the `SSH_VERSION_INT` macro to compare version numbers.

Parameters

<code>in</code>	<code>session</code>	The SSH session to use.
-----------------	----------------------	-------------------------

Returns

The version number if available, 0 otherwise.

Referenced by `ssh::Session::getOpensshVersion()`.

9.16.2.10 `int ssh_get_pubkey_hash (ssh_session session, unsigned char ** hash)`

Allocates a buffer with the MD5 hash of the server public key.

Parameters

<code>in</code>	<code>session</code>	The SSH session to use.
<code>in</code>	<code>hash</code>	The buffer to allocate.

Returns

The bytes allocated or < 0 on error.

Warning

It is very important that you verify at some moment that the hash matches a known server. If you don't do it, cryptography won't help you at making things secure

See also

[ssh_is_server_known\(\)](#)
`ssh_get_hexa()`
`ssh_print_hexa()`

References `ssh_string_len()`.

9.16.2.11 `int ssh_get_status (ssh_session session)`

Get session status.

Parameters

<i>session</i>	The ssh session to use.
----------------	-------------------------

Returns

A bitmask including SSH_CLOSED, SSH_READ_PENDING or SSH_CLOSED_ERROR which respectively means the session is closed, has data to read on the connection socket and session was closed due to an error.

9.16.2.12 int ssh_get_version (ssh_session *session*)

Get the protocol version of the session.

Parameters

<i>session</i>	The ssh session to use.
----------------	-------------------------

Returns

1 or 2, for ssh1 or ssh2, < 0 on error.

Referenced by ssh::Session::getVersion().

9.16.2.13 int ssh_is_blocking (ssh_session *session*)

Return the blocking mode of libssh.

Parameters

<i>in</i>	<i>session</i>	The SSH session
-----------	----------------	-----------------

Returns

0 if the session is nonblocking,
1 if the functions may block.

Referenced by ssh_connect().

9.16.2.14 int ssh_is_connected (ssh_session *session*)

Check if we are connected.

Parameters

<i>in</i>	<i>session</i>	The session to check if it is connected.
-----------	----------------	--

Returns

1 if we are connected, 0 if not.

9.16.2.15 `int ssh_is_server_known (ssh_session session)`

Check if the server is known.

Checks the user's known host file for a previous connection to the current server.

Parameters

<code>in</code>	<code>session</code>	The SSH session to use.
-----------------	----------------------	-------------------------

Returns

SSH_SERVER_KNOWN_OK: The server is known and has not changed.

SSH_SERVER_KNOWN_CHANGED: The server key has changed. Either you are under attack or the administrator changed the key. You HAVE to warn the user about a possible attack.

SSH_SERVER_FOUND_OTHER: The server gave use a key of a type while we had an other type recorded. It is a possible attack.

SSH_SERVER_NOT_KNOWN: The server is unknown. User should confirm the MD5 is correct.

SSH_SERVER_FILE_NOT_FOUND: The known host file does not exist. The host is thus unknown. File will be created if host key is accepted.

SSH_SERVER_ERROR: Some error happened.

See also

[ssh_get_pubkey_hash\(\)](#)

Bug

There is no current way to remove or modify an entry into the known host table.

References `ssh_write_knownhost()`.

Referenced by `ssh::Session::isServerKnown()`.

9.16.2.16 `ssh_session ssh_new (void)`

Create a new ssh session.

Returns

A new `ssh_session` pointer, NULL on error.

References `ssh_buffer_new()`, `ssh_free()`, and `ssh_set_blocking()`.

9.16.2.17 `int ssh_options_copy (ssh_session src, ssh_session * dest)`

Duplicate the options of a session structure.

If you make several sessions with the same options this is useful. You cannot use twice the same option structure in `ssh_session_connect`.

Parameters

<i>src</i>	The session to use to copy the options.
<i>dest</i>	The session to copy the options to.

Returns

0 on success, -1 on error with errno set.

See also

`ssh_session_connect()`

Referenced by `ssh::Session::optionsCopy()`.

9.16.2.18 int ssh_options_getopt (ssh_session session, int * argcptr, char ** argv)

Parse command line arguments.

This is a helper for your application to generate the appropriate options from the command line arguments.

The argv array and argc value are changed so that the parsed arguments won't appear anymore in them.

The single arguments (without switches) are not parsed. thus, `myssh -l user localhost`

The command won't set the hostname value of options to localhost.

Parameters

<i>session</i>	The session to configure.
<i>argcptr</i>	The pointer to the argument count.
<i>argv</i>	The arguments list pointer.

Returns

0 on success, < 0 on error.

See also

`ssh_session_new()`

References `ssh_options_set()`.

9.16.2.19 int ssh_options_parse_config (ssh_session session, const char * filename)

Parse the ssh config file.

This should be the last call of all options, it may overwrite options which are already set. It requires that the host name is already set with `ssh_options_set_host()`.

Parameters

<i>session</i>	SSH session handle
<i>filename</i>	The options file to use, if NULL the default ~/.ssh/config will be used.

Returns

0 on success, < 0 on error.

See also

ssh_options_set_host()

References ssh_options_set().

Referenced by ssh::Session::optionsParseConfig().

9.16.2.20 `int ssh_options_set (ssh_session session, enum ssh_options_e type, const void * value)`

This function can set all possible ssh options.

Parameters

<i>session</i>	An allocated SSH session structure.
<i>type</i>	The option type to set. This could be one of the following:

- SSH_OPTIONS_HOST: The hostname or ip address to connect to (const char *).
- SSH_OPTIONS_PORT: The port to connect to (unsigned int).
- SSH_OPTIONS_PORT_STR: The port to connect to (const char *).
- SSH_OPTIONS_FD: The file descriptor to use (socket_t).
If you wish to open the socket yourself for a reason or another, set the file descriptor. Don't forget to set the hostname as the hostname is used as a key in the known_host mechanism.
- SSH_OPTIONS_BINDADDR: The address to bind the client to (const char *).
- SSH_OPTIONS_USER: The username for authentication (const char *).
If the value is NULL, the username is set to the default username.
- SSH_OPTIONS_SSH_DIR: Set the ssh directory (const char *,format string).
If the value is NULL, the directory is set to the default ssh directory.
The ssh directory is used for files like known_hosts and identity (private and public key). It may include "%s" which will be replaced by the user home directory.

- **SSH_OPTIONS_KNOWNHOSTS**: Set the known hosts file name (const char *,format string).

If the value is NULL, the directory is set to the default known hosts file, normally `~/.ssh/known_hosts`.

The known hosts file is used to certify remote hosts are genuine. It may include "%s" which will be replaced by the user home directory.

- **SSH_OPTIONS_IDENTITY**: Set the identity file name (const char *,format string).

By default identity, id_dsa and id_rsa are checked.

The identity file used authenticate with public key. It may include "%s" which will be replaced by the user home directory.

- **SSH_OPTIONS_TIMEOUT**: Set a timeout for the connection in seconds (long).

- **SSH_OPTIONS_TIMEOUT_USEC**: Set a timeout for the connection in micro seconds (long).

- **SSH_OPTIONS_SSH1**: Allow or deny the connection to SSH1 servers (int, 0 is false).

- **SSH_OPTIONS_SSH2**: Allow or deny the connection to SSH2 servers (int, 0 is false).

- **SSH_OPTIONS_LOG_VERBOSITY**: Set the session logging verbosity (int).

The verbosity of the messages. Every log smaller or equal to verbosity will be shown.

- **SSH_LOG_NOLOG**: No logging
- **SSH_LOG_RARE**: Rare conditions or warnings
- **SSH_LOG_ENTRY**: API-accessible entrypoints
- **SSH_LOG_PACKET**: Packet id and size
- **SSH_LOG_FUNCTIONS**: Function entering and leaving

- **SSH_OPTIONS_LOG_VERBOSITY_STR**: Set the session logging verbosity (const char *).

The verbosity of the messages. Every log smaller or equal to verbosity will be shown.

- **SSH_LOG_NOLOG**: No logging
- **SSH_LOG_RARE**: Rare conditions or warnings
- **SSH_LOG_ENTRY**: API-accessible entrypoints
- **SSH_LOG_PACKET**: Packet id and size
- **SSH_LOG_FUNCTIONS**: Function entering and leaving

See the corresponding numbers in [libssh.h](#).

- `SSH_OPTIONS_AUTH_CALLBACK`: Set a callback to use your own authentication function (function pointer).
- `SSH_OPTIONS_AUTH_USERDATA`: Set the user data passed to the authentication function (generic pointer).
- `SSH_OPTIONS_LOG_CALLBACK`: Set a callback to use your own logging function (function pointer).
- `SSH_OPTIONS_LOG_USERDATA`: Set the user data passed to the logging function (generic pointer).
- `SSH_OPTIONS_STATUS_CALLBACK`: Set a callback to show connection status in realtime (function pointer).

```
fn(void *arg, float status)
```

During `ssh_connect()`, libssh will call the callback with status from 0.0 to 1.0.

- `SSH_OPTIONS_STATUS_ARG`: Set the status argument which should be passed to the status callback (generic pointer).
- `SSH_OPTIONS_CIPHERS_C_S`: Set the symmetric cipher client to server (const char *, comma-separated list).
- `SSH_OPTIONS_CIPHERS_S_C`: Set the symmetric cipher server to client (const char *, comma-separated list).
- `SSH_OPTIONS_COMPRESSION_C_S`: Set the compression to use for client to server communication (const char *, "yes", "no" or a specific algorithm name if needed ("zlib", "zlib@openssh.com", "none")).
- `SSH_OPTIONS_COMPRESSION_S_C`: Set the compression to use for server to client communication (const char *, "yes", "no" or a specific algorithm name if needed ("zlib", "zlib@openssh.com", "none")).
- `SSH_OPTIONS_COMPRESSION`: Set the compression to use for both directions communication (const char *, "yes", "no" or a specific algorithm name if needed ("zlib", "zlib@openssh.com", "none")).
- `SSH_OPTIONS_COMPRESSION_LEVEL`: Set the compression level to use for zlib functions. (int, value from 1 to 9, 9 being the most efficient but slower).
- `SSH_OPTIONS_STRICTHOSTKEYCHECK`: Set the parameter StrictHostKeyChecking to avoid asking about a fingerprint (int, 0 = false).
- `SSH_OPTIONS_PROXYCOMMAND`: Set the command to be executed in order to connect to server (const char *).

Parameters

<i>value</i>	The value to set. This is a generic pointer and the datatype which is used should be set according to the type set.
--------------	---

Returns

0 on success, < 0 on error.

References `ssh_options_set()`, and `ssh_path_expand_tilde()`.

Referenced by `ssh::Session::setOption()`, `ssh_options_getopt()`, `ssh_options_parse_config()`, and `ssh_options_set()`.

9.16.2.21 `int ssh_select (ssh_channel * channels, ssh_channel * outchannels, socket_t maxfd, fd_set * readfds, struct timeval * timeout)`

A wrapper for the select syscall.

This functions acts more or less like the select(2) syscall.

There is no support for writing or exceptions.

Parameters

in	<i>channels</i>	Arrays of channels pointers terminated by a NULL. It is never rewritten.
out	<i>outchannels</i>	Arrays of same size that "channels", there is no need to initialize it.
in	<i>maxfd</i>	Maximum +1 file descriptor from readfds.
in	<i>readfds</i>	A fd_set of file descriptors to be select'ed for reading.
in	<i>timeout</i>	A timeout for the select.

Returns

-1 if an error occurred. SSH_EINTR if it was interrupted, in that case, just restart it.

Warning

libssh is not threadsafe here. That means that if a signal is caught during the processing of this function, you cannot call ssh functions on sessions that are busy with `ssh_select()`.

See also

`select(2)`

References `ssh_channel_poll()`.

9.16.2.22 `void ssh_set_blocking (ssh_session session, int blocking)`

Set the session in blocking/nonblocking mode.

Parameters

in	<i>session</i>	The ssh session to change.
in	<i>blocking</i>	Zero for nonblocking mode.

Bug

nonblocking code is in development and won't work as expected

Referenced by `ssh_new()`.

9.16.2.23 void ssh_set_fd_except (ssh_session *session*)

Tell the session it has an exception to catch on the file descriptor.

Parameters

in	<i>session</i>	The ssh session to use.
----	----------------	-------------------------

9.16.2.24 void ssh_set_fd_toread (ssh_session *session*)

Tell the session it has data to read on the file descriptor without blocking.

Parameters

in	<i>session</i>	The ssh session to use.
----	----------------	-------------------------

9.16.2.25 void ssh_set_fd_towrite (ssh_session *session*)

Tell the session it may write to the file descriptor without blocking.

Parameters

in	<i>session</i>	The ssh session to use.
----	----------------	-------------------------

9.16.2.26 void ssh_silent_disconnect (ssh_session *session*)

Disconnect impolitely from a remote host by closing the socket.

Suitable if you forked and want to destroy this session.

Parameters

in	<i>session</i>	The SSH session to disconnect.
----	----------------	--------------------------------

References `ssh_disconnect()`.

Referenced by `ssh::Session::silentDisconnect()`.

9.16.2.27 `int ssh_write_knownhost (ssh_session session)`

Write the current server as known in the known hosts file.

This will create the known hosts file if it does not exist. You generally use it when `ssh_is_server_known()` answered `SSH_SERVER_NOT_KNOWN`.

Parameters

<code>in</code>	<code>session</code>	The ssh session to use.
-----------------	----------------------	-------------------------

Returns

`SSH_OK` on success, `SSH_ERROR` on error.

References `ssh_dirname()`, `ssh_mkdir()`, and `ssh_string_len()`.

Referenced by `ssh_is_server_known()`, and `ssh::Session::writeKnownhost()`.

9.17 The SSH string functions

String manipulations used in libssh.

Functions

- void `ssh_string_burn` (struct `ssh_string_struct` *s)
Destroy the data in a string so it couldn't appear in a core dump.
- struct `ssh_string_struct` * `ssh_string_copy` (struct `ssh_string_struct` *s)
Copy a string, return a newly allocated string.
- void * `ssh_string_data` (struct `ssh_string_struct` *s)
Get the payload of the string.
- int `ssh_string_fill` (struct `ssh_string_struct` *s, const void *data, size_t len)
Fill a string with given data.
- void `ssh_string_free` (struct `ssh_string_struct` *s)
Deallocate a SSH string object.
- void `ssh_string_free_char` (char *s)
Deallocate a char string object.
- struct `ssh_string_struct` * `ssh_string_from_char` (const char *what)
Create a ssh string using a C string.

- `size_t ssh_string_len` (`struct ssh_string_struct *s`)
Return the size of a SSH string.
- `struct ssh_string_struct * ssh_string_new` (`size_t size`)
Create a new SSH String object.
- `char * ssh_string_to_char` (`struct ssh_string_struct *s`)
Convert a SSH string to a C nul-terminated string.

9.17.1 Detailed Description

String manipulations used in libssh.

9.17.2 Function Documentation

9.17.2.1 `void ssh_string_burn (struct ssh_string_struct * s)`

Destroy the data in a string so it couldn't appear in a core dump.

Parameters

<code>in</code>	<code>s</code>	The string to burn.
-----------------	----------------	---------------------

References `ssh_string_len()`.

Referenced by `publickey_from_privatekey()`, and `ssh_userauth_password()`.

9.17.2.2 `struct ssh_string_struct* ssh_string_copy (struct ssh_string_struct * s)` `[read]`

Copy a string, return a newly allocated string.

The caller has to free the string.

Parameters

<code>in</code>	<code>s</code>	String to copy.
-----------------	----------------	-----------------

Returns

Newly allocated copy of the string, NULL on error.

9.17.2.3 `void* ssh_string_data (struct ssh_string_struct * s)`

Get the payload of the string.

Parameters

<i>s</i>	The string to get the data from.
----------	----------------------------------

Returns

Return the data of the string or NULL on error.

Referenced by `publickey_from_privatekey()`.

9.17.2.4 int ssh_string_fill (struct ssh_string_struct * s, const void * data, size_t len)

Fill a string with given data.

The string should be big enough.

Parameters

<i>s</i>	An allocated string to fill with data.
<i>data</i>	The data to fill the string with.
<i>len</i>	Size of data.

Returns

0 on success, < 0 on error.

Referenced by `publickey_from_file()`, `publickey_from_privatekey()`, and `publickey_to_string()`.

9.17.2.5 void ssh_string_free (struct ssh_string_struct * s)

Deallocate a SSH string object.

Parameters

<i>in</i>	<i>s</i>	The SSH string to delete.
-----------	----------	---------------------------

Referenced by `publickey_from_privatekey()`, `publickey_to_string()`, `ssh_channel_open_forward()`, `ssh_channel_request_env()`, `ssh_channel_request_exec()`, `ssh_channel_request_pty_size()`, `ssh_channel_request_send_signal()`, `ssh_channel_request_subsystem()`, `ssh_channel_request_x11()`, `ssh_disconnect()`, `ssh_forward_cancel()`, `ssh_forward_listen()`, `ssh_userauth_agent_pubkey()`, `ssh_userauth_autopubkey()`, `ssh_userauth_none()`, `ssh_userauth_offer_pubkey()`, `ssh_userauth_password()`, `ssh_userauth_privatekey_file()`, and `ssh_userauth_pubkey()`.

9.17.2.6 void ssh_string_free_char (char * s)

Deallocate a char string object.

Parameters

<i>in</i>	<i>s</i>	The string to delete.
-----------	----------	-----------------------

9.17.2.7 struct ssh_string_struct* ssh_string_from_char (const char * *what*) [read]

Create a ssh string using a C string.

Parameters

<i>in</i>	<i>what</i>	The source 0-terminated C string.
-----------	-------------	-----------------------------------

Returns

The newly allocated string, NULL on error with errno set.

Note

The nul byte is not copied nor counted in the output string.

Referenced by `publickey_to_string()`, `ssh_channel_open_forward()`, `ssh_channel_request_env()`, `ssh_channel_request_exec()`, `ssh_channel_request_pty_size()`, `ssh_channel_request_send_signal()`, `ssh_channel_request_subsystem()`, `ssh_channel_request_x11()`, `ssh_disconnect()`, `ssh_forward_cancel()`, `ssh_forward_listen()`, `ssh_userauth_agent_pubkey()`, `ssh_userauth_none()`, `ssh_userauth_offer_pubkey()`, `ssh_userauth_password()`, and `ssh_userauth_pubkey()`.

9.17.2.8 size_t ssh_string_len (struct ssh_string_struct * *s*)

Return the size of a SSH string.

Parameters

<i>in</i>	<i>s</i>	The the input SSH string.
-----------	----------	---------------------------

Returns

The size of the content of the string, 0 on error.

Referenced by `publickey_from_privatekey()`, `ssh_get_pubkey_hash()`, `ssh_publickey_to_file()`, `ssh_string_burn()`, and `ssh_write_knownhost()`.

9.17.2.9 struct ssh_string_struct* ssh_string_new (size_t *size*) [read]

Create a new SSH String object.

Parameters

<i>in</i>	<i>size</i>	The size of the string.
-----------	-------------	-------------------------

Returns

The newly allocated string, NULL on error.

Referenced by `publickey_from_file()`, `publickey_from_privatekey()`, and `publickey_to_string()`.

9.17.2.10 char* ssh_string_to_char (struct ssh_string_struct * s)

Convert a SSH string to a C nul-terminated string.

Parameters

in	s	The SSH input string.
----	---	-----------------------

Returns

An allocated string pointer, NULL on error with `errno` set.

Note

If the input SSH string contains zeroes, some parts of the output string may not be readable with regular libc functions.

Referenced by `ssh_get_issue_banner()`.

9.18 The SSH threading functions.

Threading with libssh.

Functions

- `struct ssh_threads_callbacks_struct * ssh_threads_get_noop ()`
returns a pointer on the noop threads callbacks, to be used with `ssh_threads_set_callbacks`.
- `int ssh_threads_set_callbacks (struct ssh_threads_callbacks_struct *cb)`
sets the thread callbacks necessary if your program is using libssh in a multithreaded fashion.

9.18.1 Detailed Description

Threading with libssh.

9.18.2 Function Documentation

9.18.2.1 `struct ssh_threads_callbacks_struct* ssh_threads_get_noop (void)` [read]

returns a pointer on the noop threads callbacks, to be used with `ssh_threads_set_callbacks`.

These callbacks do nothing and are being used by default.

See also

[ssh_threads_set_callbacks](#)

9.18.2.2 `int ssh_threads_set_callbacks (struct ssh_threads_callbacks_struct * cb)`

sets the thread callbacks necessary if your program is using libssh in a multithreaded fashion.

This function must be called first, outside of any threading context (in your `main()` for instance), before [ssh_init\(\)](#).

Parameters

<i>cb</i>	pointer to a <code>ssh_threads_callbacks_struct</code> structure, which contains the different callbacks to be set.
-----------	---

See also

`ssh_threads_callbacks_struct`
`SSH_THREADS_PTHREAD`

Chapter 10

Data Structure Documentation

10.1 ssh::Channel Class Reference

the [ssh::Channel](#) class describes the state of an SSH channel.

```
#include <include/libssh/libsshpp.hpp>
```

Public Member Functions

- [Channel](#) * [acceptX11](#) (int timeout_ms)
accept an incoming X11 connection
- void [changePtySize](#) (int cols, int rows)
change the size of a pseudoterminal
- void [close](#) ()
closes a channel
- bool [isClosed](#) ()
returns true if channel is in closed state
- bool [isEof](#) ()
returns true if channel is in EOF state
- bool [isOpen](#) ()
returns true if channel is in open state
- int [write](#) (const void *data, size_t len, bool is_stderr=false)
Writes on a channel.

10.1.1 Detailed Description

the [ssh::Channel](#) class describes the state of an SSH channel.

See also

[ssh_channel](#)

10.1.2 Member Function Documentation

10.1.2.1 `Channel* ssh::Channel::acceptX11 (int timeout_ms)` `[inline]`

accept an incoming X11 connection

Parameters

<code>in</code>	<code><i>timeout_ms</i></code>	timeout for waiting, in ms
-----------------	--------------------------------	----------------------------

Returns

new [Channel](#) pointer on the X11 connection
NULL in case of error

Warning

you have to delete this pointer after use

See also

[ssh_channel_accept_x11](#)
[Channel::requestX11](#)

References [ssh_channel_accept_x11\(\)](#).

10.1.2.2 `void ssh::Channel::changePtySize (int cols, int rows)` `[inline]`

change the size of a pseudoterminal

Parameters

<code>in</code>	<code><i>cols</i></code>	number of columns
<code>in</code>	<code><i>rows</i></code>	number of rows

Exceptions

SshException	on error
------------------------------	----------

See also

[ssh_channel_change_pty_size](#)

References [ssh_channel_change_pty_size\(\)](#).

10.1.2.3 void ssh::Channel::close () [inline]

closes a channel

Exceptions

<i>SshException</i>	on error
-------------------------------------	----------

See also

[ssh_channel_close](#)

References [ssh_channel_close\(\)](#).

10.1.2.4 bool ssh::Channel::isClosed () [inline]

returns true if channel is in closed state

See also

[ssh_channel_is_closed](#)

References [ssh_channel_is_closed\(\)](#).

10.1.2.5 bool ssh::Channel::isEof () [inline]

returns true if channel is in EOF state

See also

[ssh_channel_is_eof](#)

References [ssh_channel_is_eof\(\)](#).

10.1.2.6 bool ssh::Channel::isOpen () [inline]

returns true if channel is in open state

See also

[ssh_channel_is_open](#)

References [ssh_channel_is_open\(\)](#).

10.1.2.7 int ssh::Channel::write (const void * *data*, size_t *len*, bool *is_stderr* = false) [inline]

Writes on a channel.

Parameters

<i>data</i>	data to write.
<i>len</i>	number of bytes to write.
<i>is_stderr</i>	write should be done on the stderr channel (server only)

Returns

number of bytes written

Exceptions

SshException	in case of error
------------------------------	------------------

See also

channel_write
channel_write_stderr

References ssh_channel_write().

The documentation for this class was generated from the following file:

- include/libssh/libsshpp.hpp

10.2 ssh::Session Class Reference

The [ssh::Session](#) class contains the state of a SSH connection.

```
#include <include/libssh/libsshpp.hpp>
```

Public Member Functions

- [Channel](#) * [acceptForward](#) (int timeout_ms)
accept an incoming forward connection
- void [connect](#) ()
connects to the remote host
- void [disconnect](#) ()
Disconnects from the SSH server and closes connection.
- int [getAuthList](#) ()
Returns the available authentication methods from the server.
- const char * [getDisconnectMessage](#) ()
Returns the disconnect message from the server, if any.
- std::string [getIssueBanner](#) ()

gets the Issue banner from the ssh server

- int [getOpensshVersion](#) ()
returns the OpenSSH version (server) if possible
- socket_t [getSocket](#) ()
returns the file descriptor used for the communication
- int [getVersion](#) ()
returns the version of the SSH protocol being used
- int [isServerKnown](#) ()
verifies that the server is known
- void [optionsCopy](#) (const [Session](#) &source)
copies options from a session to another
- void [optionsParseConfig](#) (const char *file)
parses a configuration file for options
- void [setOption](#) (enum ssh_options_e type, long int option)
sets an SSH session options
- void [setOption](#) (enum ssh_options_e type, void *option)
sets an SSH session options
- void [setOption](#) (enum ssh_options_e type, const char *option)
sets an SSH session options
- void [silentDisconnect](#) ()
silently disconnect from remote host
- int [userauthAutopubkey](#) (void)
Authenticates automatically using public key.
- int [userauthNone](#) ()
Authenticates using the "none" method.
- int [userauthOfferPubkey](#) (int type, ssh_string pubkey)
Try to authenticate using the publickey method.
- int [userauthPassword](#) (const char *password)
Authenticates using the password method.
- int [userauthPubkey](#) (ssh_string pubkey, ssh_private_key privkey)
Authenticates using the publickey method.

- int [writeKnownhost](#) ()

Writes the known host file with current host key.

10.2.1 Detailed Description

The [ssh::Session](#) class contains the state of a SSH connection.

10.2.2 Member Function Documentation

10.2.2.1 Channel * ssh::Session::acceptForward (int *timeout_ms*)

accept an incoming forward connection

Parameters

in	<i>timeout_ms</i>	timeout for waiting, in ms
----	-------------------	----------------------------

Returns

new [Channel](#) pointer on the forward connection
NULL in case of error

Warning

you have to delete this pointer after use

See also

[ssh_channel_forward_accept](#)
[Session::listenForward](#)

References [ssh_forward_accept\(\)](#).

10.2.2.2 void ssh::Session::connect () [inline]

connects to the remote host

Exceptions

SshException	on error
------------------------------	----------

See also

[ssh_connect](#)

References [ssh_connect\(\)](#).

10.2.2.3 void ssh::Session::disconnect () [inline]

Disconnects from the SSH server and closes connection.

See also

[ssh_disconnect](#)

References `ssh_disconnect()`.

10.2.2.4 int ssh::Session::getAuthList () [inline]

Returns the available authentication methods from the server.

Exceptions

SshException	on error
------------------------------	----------

Returns

Bitfield of available methods.

See also

[ssh_userauth_list](#)

References `ssh_userauth_list()`.

10.2.2.5 const char* ssh::Session::getDisconnectMessage () [inline]

Returns the disconnect message from the server, if any.

Returns

pointer to the message, or NULL. Do not attempt to free the pointer.

References `ssh_get_disconnect_message()`.

10.2.2.6 std::string ssh::Session::getIssueBanner () [inline]

gets the Issue banner from the ssh server

Returns

the issue banner. This is generally a MOTD from server

See also

[ssh_get_issue_banner](#)

References `ssh_get_issue_banner()`.

10.2.2.7 `int ssh::Session::getOpensshVersion () [inline]`

returns the OpenSSH version (server) if possible

Returns

openssh version code

See also

[ssh_get_openssh_version](#)

References `ssh_get_openssh_version()`.

10.2.2.8 `socket_t ssh::Session::getSocket () [inline]`

returns the file descriptor used for the communication

Returns

the file descriptor

Warning

if a proxycommand is used, this function will only return one of the two file descriptors being used

See also

[ssh_get_fd](#)

References `ssh_get_fd()`.

10.2.2.9 `int ssh::Session::getVersion () [inline]`

returns the version of the SSH protocol being used

Returns

the SSH protocol version

See also

[ssh_get_version](#)

References `ssh_get_version()`.

10.2.2.10 `int ssh::Session::isServerKnown () [inline]`

verifies that the server is known

Exceptions

<i>SshException</i>	on error
-------------------------------------	----------

Returns

Integer value depending on the knowledge of the server key

See also

[*ssh_is_server_known*](#)

References `ssh_is_server_known()`.

10.2.2.11 `void ssh::Session::optionsCopy (const Session & source)` `[inline]`

copies options from a session to another

Exceptions

<i>SshException</i>	on error
-------------------------------------	----------

See also

[*ssh_options_copy*](#)

References `ssh_options_copy()`.

10.2.2.12 `void ssh::Session::optionsParseConfig (const char * file)` `[inline]`

parses a configuration file for options

Exceptions

<i>SshException</i>	on error
-------------------------------------	----------

Parameters

<code>in</code>	<code><i>file</i></code>	configuration file name
-----------------	--------------------------	-------------------------

See also

[*ssh_options_parse_config*](#)

References `ssh_options_parse_config()`.

10.2.2.13 `void ssh::Session::setOption (enum ssh_options_e type, const char * option)`
`[inline]`

sets an SSH session options

Parameters

<i>type</i>	Type of option
<i>option</i>	cstring containing the value of option

Exceptions

<i>SshException</i>	on error
-------------------------------------	----------

See also[ssh_options_set](#)

References `ssh_options_set()`.

10.2.2.14 `void ssh::Session::setOption (enum ssh_options_e type, long int option)`
[inline]

sets an SSH session options

Parameters

<i>type</i>	Type of option
<i>option</i>	long integer containing the value of option

Exceptions

<i>SshException</i>	on error
-------------------------------------	----------

See also[ssh_options_set](#)

References `ssh_options_set()`.

10.2.2.15 `void ssh::Session::setOption (enum ssh_options_e type, void * option)`
[inline]

sets an SSH session options

Parameters

<i>type</i>	Type of option
<i>option</i>	void pointer containing the value of option

Exceptions

<i>SshException</i>	on error
-------------------------------------	----------

See also[ssh_options_set](#)

References `ssh_options_set()`.

10.2.2.16 `void ssh::Session::silentDisconnect ()` `[inline]`

silently disconnect from remote host

See also

[ssh_silent_disconnect](#)

References `ssh_silent_disconnect()`.

10.2.2.17 `int ssh::Session::userauthAutopubkey (void)` `[inline]`

Authenticates automatically using public key.

Exceptions

SshException	on error
------------------------------	----------

Returns

SSH_AUTH_SUCCESS, SSH_AUTH_PARTIAL, SSH_AUTH_DENIED

See also

[ssh_userauth_autopubkey](#)

References `ssh_userauth_autopubkey()`.

10.2.2.18 `int ssh::Session::userauthNone ()` `[inline]`

Authenticates using the "none" method.

Prefer using autopubkey if possible.

Exceptions

SshException	on error
------------------------------	----------

Returns

SSH_AUTH_SUCCESS, SSH_AUTH_PARTIAL, SSH_AUTH_DENIED

See also

[ssh_userauth_none](#)

`Session::userauthAutoPubkey`

References `ssh_userauth_none()`.

10.2.2.19 `int ssh::Session::userauthOfferPubkey (int type, ssh_string pubkey)` `[inline]`

Try to authenticate using the publickey method.

Parameters

<i>in</i>	<i>type</i>	public key type
<i>in</i>	<i>pubkey</i>	public key to use for authentication

Exceptions

<i>SshException</i>	on error
-------------------------------------	----------

Returns

SSH_AUTH_SUCCESS if the pubkey is accepted,
SSH_AUTH_DENIED if the pubkey is denied

See also

[ssh_userauth_offer_pubkey](#)

References `ssh_userauth_offer_pubkey()`.

10.2.2.20 `int ssh::Session::userauthPassword (const char * password)` `[inline]`

Authenticates using the password method.

Parameters

<i>in</i>	<i>password</i>	password to use for authentication
-----------	-----------------	------------------------------------

Exceptions

<i>SshException</i>	on error
-------------------------------------	----------

Returns

SSH_AUTH_SUCCESS, SSH_AUTH_PARTIAL, SSH_AUTH_DENIED

See also

[ssh_userauth_password](#)

References `ssh_userauth_password()`.

10.2.2.21 `int ssh::Session::userauthPubkey (ssh_string pubkey, ssh_private_key privkey)`
`[inline]`

Authenticates using the publickey method.

Parameters

in	<i>pubkey</i>	public key to use for authentication
in	<i>privkey</i>	private key to use for authentication

Exceptions

<i>SshException</i>	on error
-------------------------------------	----------

Returns

SSH_AUTH_SUCCESS, SSH_AUTH_PARTIAL, SSH_AUTH_DENIED

See also

[ssh_userauth_pubkey](#)

References `ssh_userauth_pubkey()`.

10.2.2.22 `int ssh::Session::writeKnownhost()` `[inline]`

Writes the known host file with current host key.

Exceptions

<i>SshException</i>	on error
-------------------------------------	----------

See also

[ssh_write_knownhost](#)

References `ssh_write_knownhost()`.

The documentation for this class was generated from the following file:

- `include/libssh/libsshpp.hpp`

10.3 ssh_bind_callbacks Struct Reference

These are the callbacks exported by the `ssh_bind` structure.

```
#include <include/libssh/server.h>
```

Data Fields

- [ssh_bind_incoming_connection_callback](#) `incoming_connection`
A new connection is available.
- `size_t` [size](#)
DON'T SET THIS use [ssh_callbacks_init\(\)](#) instead.

10.3.1 Detailed Description

These are the callbacks exported by the `ssh_bind` structure. They are called by the server module when events appear on the network.

10.3.2 Field Documentation

10.3.2.1 `ssh_bind_incoming_connection_callback` `ssh_bind_callbacks::incoming_connection`

A new connection is available.

10.3.2.2 `size_t ssh_bind_callbacks::size`

DON'T SET THIS use `ssh_callbacks_init()` instead.

The documentation for this struct was generated from the following file:

- `include/libssh/server.h`

10.4 `ssh_callbacks` Struct Reference

The structure to replace libssh functions with appropriate callbacks.

```
#include <include/libssh/callbacks.h>
```

Data Fields

- `ssh_auth_callback auth_function`
This functions will be called if e.g.
- `void(* connect_status_function)(void *userdata, float status)`
This function gets called during connection time to indicate the percentage of connection steps completed.
- `ssh_global_request_callback global_request_function`
This function will be called each time a global request is received.
- `ssh_log_callback log_function`
This function will be called each time a loggable event happens.
- `size_t size`
DON'T SET THIS use `ssh_callbacks_init()` instead.
- `void * userdata`
User-provided data.

10.4.1 Detailed Description

The structure to replace libssh functions with appropriate callbacks.

10.4.2 Field Documentation

10.4.2.1 `ssh_auth_callback` `ssh_callbacks::auth_function`

This functions will be called if e.g.
a keyphrase is needed.

10.4.2.2 `size_t` `ssh_callbacks::size`

DON'T SET THIS use [ssh_callbacks_init\(\)](#) instead.

10.4.2.3 `void*` `ssh_callbacks::userdata`

User-provided data.

User is free to set anything he wants here

The documentation for this struct was generated from the following file:

- `include/libssh/callbacks.h`

10.5 `ssh_socket_callbacks` Struct Reference

These are the callbacks exported by the socket structure They are called by the socket module when a socket event appears.

```
#include <include/libssh/callbacks.h>
```

Data Fields

- `ssh_callback_int_int` [connected](#)
This function is called when the `ssh_socket_connect` was used on the socket on non-blocking state, and the connection succeeded.
- `ssh_callback_int` [controlflow](#)
This function will be called each time a controlflow state changes, i.e.
- `ssh_callback_data` [data](#)
This function will be called each time data appears on socket.
- `ssh_callback_int_int` [exception](#)

This function will be called each time an exception appears on socket.

- void * [userdata](#)

User-provided data.

10.5.1 Detailed Description

These are the callbacks exported by the socket structure. They are called by the socket module when a socket event appears.

10.5.2 Field Documentation

10.5.2.1 `ssh_callback_int ssh_socket_callbacks::controlflow`

This function will be called each time a controlflow state changes, i.e. the socket is available for reading or writing.

10.5.2.2 `ssh_callback_data ssh_socket_callbacks::data`

This function will be called each time data appears on socket. The data not consumed will appear on the next data event.

10.5.2.3 `ssh_callback_int_int ssh_socket_callbacks::exception`

This function will be called each time an exception appears on socket. An exception can be a socket problem (timeout, ...) or an end-of-file.

10.5.2.4 `void* ssh_socket_callbacks::userdata`

User-provided data.

User is free to set anything he wants here

The documentation for this struct was generated from the following file:

- `include/libssh/callbacks.h`

10.6 `ssh::SshException` Class Reference

Some people do not like C++ exceptions.

```
#include <include/libssh/libsshpp.hpp>
```

Public Member Functions

- int [getCode](#) ()
returns the Error code
- std::string [getError](#) ()
returns the error message of the last exception

10.6.1 Detailed Description

Some people do not like C++ exceptions. With this define, we give the choice to use or not exceptions. if defined, disable C++ exceptions for libssh c++ wrapperThis class describes a SSH Exception object. This object can be thrown by several SSH functions that interact with the network, and may fail because of socket, protocol or memory errors.

10.6.2 Member Function Documentation

10.6.2.1 int ssh::SshException::getCode () [inline]

returns the Error code

Returns

SSH_FATAL Fatal error happened (not recoverable)
SSH_REQUEST_DENIED Request was denied by remote host

See also

[ssh_get_error_code](#)

10.6.2.2 std::string ssh::SshException::getError () [inline]

returns the error message of the last exception

Returns

pointer to a c string containing the description of error

See also

[ssh_get_error](#)

The documentation for this class was generated from the following file:

- include/libssh/libsshpp.hpp

Chapter 11

File Documentation

11.1 include/libssh/sftp.h File Reference

SFTP handling functions.

```
#include <sys/types.h>
#include "libssh.h"
```

Defines

Server responses

Responses returned by the sftp server.

- #define [SSH_FX_OK](#) 0
No error.
- #define [SSH_FX_EOF](#) 1
End-of-file encountered.
- #define [SSH_FX_NO_SUCH_FILE](#) 2
File doesn't exist.
- #define [SSH_FX_PERMISSION_DENIED](#) 3
Permission denied.
- #define [SSH_FX_FAILURE](#) 4
Generic failure.
- #define [SSH_FX_BAD_MESSAGE](#) 5
Garbage received from server.
- #define [SSH_FX_NO_CONNECTION](#) 6
No connection has been set up.

- #define [SSH_FX_CONNECTION_LOST](#) 7
There was a connection, but we lost it.
- #define [SSH_FX_OP_UNSUPPORTED](#) 8
Operation not supported by the server.
- #define [SSH_FX_INVALID_HANDLE](#) 9
Invalid file handle.
- #define [SSH_FX_NO_SUCH_PATH](#) 10
No such file or directory path exists.
- #define [SSH_FX_FILE_ALREADY_EXISTS](#) 11
An attempt to create an already existing file or directory has been made.
- #define [SSH_FX_WRITE_PROTECT](#) 12
We are trying to write on a write-protected filesystem.
- #define [SSH_FX_NO_MEDIA](#) 13
No media in remote drive.

Functions

- int [sftp_async_read](#) (sftp_file file, void *data, uint32_t len, uint32_t id)
Wait for an asynchronous read to complete and save the data.
- int [sftp_async_read_begin](#) (sftp_file file, uint32_t len)
Start an asynchronous read from a file using an opened sftp file handle.
- void [sftp_attributes_free](#) (sftp_attributes file)
Free a sftp attribute structure.
- char * [sftp_canonicalize_path](#) (sftp_session sftp, const char *path)
Canonicalize a sftp path.
- int [sftp_chmod](#) (sftp_session sftp, const char *file, mode_t mode)
Change permissions of a file.
- int [sftp_chown](#) (sftp_session sftp, const char *file, uid_t owner, gid_t group)
Change the file owner and group.
- int [sftp_close](#) (sftp_file file)
Close an open file handle.
- int [sftp_closedir](#) (sftp_dir dir)

Close a directory handle opened by [sftp_opendir\(\)](#).

- int [sftp_dir_eof](#) (sftp_dir dir)
Tell if the directory has reached EOF (End Of File).
- int [sftp_extension_supported](#) (sftp_session sftp, const char *name, const char *data)
Check if the given extension is supported.
- unsigned int [sftp_extensions_get_count](#) (sftp_session sftp)
Get the count of extensions provided by the server.
- const char * [sftp_extensions_get_data](#) (sftp_session sftp, unsigned int indexn)
Get the data of the extension provided by the server.
- const char * [sftp_extensions_get_name](#) (sftp_session sftp, unsigned int indexn)
Get the name of the extension provided by the server.
- void [sftp_free](#) (sftp_session sftp)
Close and deallocate a sftp session.
- sftp_attributes [sftp_fstat](#) (sftp_file file)
Get information about a file or directory from a file handle.
- sftp_statvfs_t [sftp_fstatvfs](#) (sftp_file file)
Get information about a mounted file system.
- int [sftp_get_error](#) (sftp_session sftp)
Get the last sftp error.
- int [sftp_init](#) (sftp_session sftp)
Initialize the sftp session with the server.
- sftp_attributes [sftp_lstat](#) (sftp_session session, const char *path)
Get information about a file or directory.
- int [sftp_mkdir](#) (sftp_session sftp, const char *directory, mode_t mode)
Create a directory.
- sftp_session [sftp_new](#) (ssh_session session)
Start a new sftp session.
- sftp_file [sftp_open](#) (sftp_session session, const char *file, int accesstype, mode_t mode)
Open a file on the server.

- `sftp_dir sftp_opendir` (`sftp_session session`, `const char *path`)
Open a directory used to obtain directory entries.
- `ssize_t sftp_read` (`sftp_file file`, `void *buf`, `size_t count`)
Read from a file using an opened sftp file handle.
- `sftp_attributes sftp_readdir` (`sftp_session session`, `sftp_dir dir`)
Get a single file attributes structure of a directory.
- `char * sftp_readlink` (`sftp_session sftp`, `const char *path`)
Read the value of a symbolic link.
- `int sftp_rename` (`sftp_session sftp`, `const char *original`, `const char *newname`)
Rename or move a file or directory.
- `void sftp_rewind` (`sftp_file file`)
Rewinds the position of the file pointer to the beginning of the file.
- `int sftp_rmdir` (`sftp_session sftp`, `const char *directory`)
Remove a directory.
- `int sftp_seek` (`sftp_file file`, `uint32_t new_offset`)
Seek to a specific location in a file.
- `int sftp_seek64` (`sftp_file file`, `uint64_t new_offset`)
Seek to a specific location in a file.
- `int sftp_server_version` (`sftp_session sftp`)
Get the version of the SFTP protocol supported by the server.
- `int sftp_setstat` (`sftp_session sftp`, `const char *file`, `sftp_attributes attr`)
Set file attributes on a file, directory or symbolic link.
- `sftp_attributes sftp_stat` (`sftp_session session`, `const char *path`)
Get information about a file or directory.
- `sftp_statvfs_t sftp_statvfs` (`sftp_session sftp`, `const char *path`)
Get information about a mounted file system.
- `void sftp_statvfs_free` (`sftp_statvfs_t statvfs_o`)
Free the memory of an allocated statvfs.
- `int sftp_symlink` (`sftp_session sftp`, `const char *target`, `const char *dest`)
Create a symbolic link.
- `unsigned long sftp_tell` (`sftp_file file`)

Report current byte position in file.

- uint64_t [sftp_tell64](#) (sftp_file file)
Report current byte position in file.
- int [sftp_unlink](#) (sftp_session sftp, const char *file)
Unlink (delete) a file.
- int [sftp_utimes](#) (sftp_session sftp, const char *file, const struct timeval *times)
Change the last modification and access time of a file.
- ssize_t [sftp_write](#) (sftp_file file, const void *buf, size_t count)
Write to a file using an opened sftp file handle.

11.1.1 Detailed Description

SFTP handling functions. SFTP commands are channeled by the ssh sftp subsystem. Every packet is sent/read using a sftp_packet type structure. Related to these packets, most of the server answers are messages having an ID and a message specific part. It is described by sftp_message when reading a message, the sftp system puts it into the queue, so the process having asked for it can fetch it, while continuing to read for other messages (it is unspecified in which order messages may be sent back to the client

Index

- acceptForward
 - ssh::Session, [166](#)
- acceptX11
 - ssh::Channel, [162](#)
- auth_function
 - ssh_callbacks_struct, [175](#)
- changePtySize
 - ssh::Channel, [162](#)
- channel_read_buffer
 - libssh_channel, [104](#)
- close
 - ssh::Channel, [162](#)
- connect
 - ssh::Session, [166](#)
- controlflow
 - ssh_socket_callbacks_struct, [176](#)
- data
 - ssh_socket_callbacks_struct, [176](#)
- disconnect
 - ssh::Session, [166](#)
- exception
 - ssh_socket_callbacks_struct, [176](#)
- getAuthList
 - ssh::Session, [167](#)
- getCode
 - ssh::SshException, [177](#)
- getDisconnectMessage
 - ssh::Session, [167](#)
- getError
 - ssh::SshException, [177](#)
- getIssueBanner
 - ssh::Session, [167](#)
- getOpensshVersion
 - ssh::Session, [167](#)
- getSocket
 - ssh::Session, [168](#)
- getVersion
 - ssh::Session, [168](#)
- include/libssh/sftp.h, [179](#)
- incoming_connection
 - ssh_bind_callbacks_struct, [174](#)
- isClosed
 - ssh::Channel, [163](#)
- isEof
 - ssh::Channel, [163](#)
- isOpen
 - ssh::Channel, [163](#)
- isServerKnown
 - ssh::Session, [168](#)
- libssh
 - ssh_finalize, [119](#)
 - ssh_init, [120](#)
- libssh_log
 - SSH_LOG_FUNCTIONS, [121](#)
 - SSH_LOG_NOLOG, [121](#)
 - SSH_LOG_PACKET, [121](#)
 - SSH_LOG_PROTOCOL, [121](#)
 - SSH_LOG_RARE, [121](#)
- libssh_auth
 - privatekey_free, [89](#)
 - privatekey_from_file, [89](#)
 - publickey_from_file, [90](#)
 - publickey_from_privatekey, [90](#)
 - publickey_to_string, [91](#)
 - ssh_auth_list, [91](#)
 - ssh_privatekey_type, [91](#)
 - ssh_publickey_to_file, [92](#)
 - ssh_try_publickey_from_file, [92](#)
 - ssh_userauth_agent_pubkey, [93](#)
 - ssh_userauth_autopubkey, [93](#)
 - ssh_userauth_kbdint, [94](#)
 - ssh_userauth_kbdint_getinstruction, [94](#)
 - ssh_userauth_kbdint_getname, [95](#)
 - ssh_userauth_kbdint_getnprompts, [95](#)
 - ssh_userauth_kbdint_getprompt, [95](#)
 - ssh_userauth_kbdint_setanswer, [96](#)
 - ssh_userauth_list, [96](#)
 - ssh_userauth_none, [97](#)

- ssh_userauth_offer_pubkey, 97
- ssh_userauth_password, 98
- ssh_userauth_privatekey_file, 98
- ssh_userauth_pubkey, 99
- libssh_buffer
 - ssh_buffer_free, 100
 - ssh_buffer_get_begin, 101
 - ssh_buffer_get_len, 101
 - ssh_buffer_new, 101
- libssh_callbacks
 - ssh_auth_callback, 60
 - ssh_callbacks_init, 59
 - ssh_channel_close_callback, 60
 - ssh_channel_data_callback, 60
 - ssh_channel_eof_callback, 60
 - ssh_channel_exit_signal_callback, 61
 - ssh_channel_exit_status_callback, 61
 - ssh_channel_signal_callback, 61
 - ssh_global_request_callback, 62
 - ssh_log_callback, 62
 - SSH_PACKET_CALLBACK, 59
 - ssh_packet_callback, 62
 - SSH_PACKET_USED, 59
 - ssh_set_callbacks, 63
 - ssh_set_channel_callbacks, 63
 - ssh_status_callback, 63
- libssh_channel
 - channel_read_buffer, 104
 - ssh_channel_accept_x11, 105
 - ssh_channel_change_pty_size, 105
 - ssh_channel_close, 105
 - ssh_channel_free, 106
 - ssh_channel_get_exit_status, 106
 - ssh_channel_get_session, 106
 - ssh_channel_is_closed, 107
 - ssh_channel_is_eof, 107
 - ssh_channel_is_open, 107
 - ssh_channel_new, 108
 - ssh_channel_open_forward, 108
 - ssh_channel_open_session, 108
 - ssh_channel_poll, 109
 - ssh_channel_read, 109
 - ssh_channel_read_nonblocking, 110
 - ssh_channel_request_env, 110
 - ssh_channel_request_exec, 111
 - ssh_channel_request_pty, 111
 - ssh_channel_request_pty_size, 112
 - ssh_channel_request_send_signal, 112
 - ssh_channel_request_shell, 113
 - ssh_channel_request_subsystem, 113
 - ssh_channel_request_x11, 114
 - ssh_channel_select, 114
 - ssh_channel_send_eof, 115
 - ssh_channel_set_blocking, 115
 - ssh_channel_write, 115
 - ssh_forward_accept, 116
 - ssh_forward_cancel, 116
 - ssh_forward_listen, 116
- libssh_error
 - ssh_get_error, 117
 - ssh_get_error_code, 118
- libssh_log
 - ssh_log, 121
- libssh_messages
 - ssh_message_get, 122
- libssh_misc
 - ssh_basename, 123
 - ssh_dirname, 123
 - ssh_getpass, 124
 - ssh_mkdir, 124
 - ssh_path_expand_tilde, 125
 - ssh_timeout_update, 125
 - ssh_version, 125
- libssh_pki
 - ssh_key_clean, 126
 - ssh_key_free, 127
 - ssh_key_import_private, 127
 - ssh_key_new, 127
 - ssh_key_type, 127
- libssh_poll
 - ssh_poll_add_events, 129
 - ssh_poll_ctx_add, 129
 - ssh_poll_ctx_add_socket, 130
 - ssh_poll_ctx_dopoll, 130
 - ssh_poll_ctx_free, 130
 - ssh_poll_ctx_new, 131
 - ssh_poll_ctx_remove, 131
 - ssh_poll_free, 131
 - ssh_poll_get_ctx, 131
 - ssh_poll_get_events, 132
 - ssh_poll_get_fd, 132
 - ssh_poll_new, 132
 - ssh_poll_remove_events, 133
 - ssh_poll_set_callback, 133
 - ssh_poll_set_events, 133
 - ssh_poll_set_fd, 133
- libssh_scp
 - ssh_scp_accept_request, 135
 - ssh_scp_deny_request, 135
 - ssh_scp_integer_mode, 135

- ssh_scp_leave_directory, 136
- ssh_scp_new, 136
- ssh_scp_pull_request, 136
- ssh_scp_push_directory, 137
- ssh_scp_push_file, 137
- ssh_scp_read, 138
- ssh_scp_read_string, 138
- ssh_scp_request_get_filename, 138
- ssh_scp_request_get_permissions, 138
- ssh_scp_request_get_size, 139
- ssh_scp_request_get_warning, 139
- ssh_scp_string_mode, 139
- ssh_scp_write, 139
- libssh_server
 - ssh_bind_accept, 66
 - ssh_bind_fd_toaccept, 67
 - ssh_bind_free, 67
 - ssh_bind_get_fd, 67
 - ssh_bind_incoming_connection_callback, 66
 - ssh_bind_listen, 67
 - ssh_bind_new, 67
 - ssh_bind_options_set, 68
 - ssh_bind_set_blocking, 69
 - ssh_bind_set_callbacks, 69
 - ssh_bind_set_fd, 70
 - ssh_handle_key_exchange, 70
 - ssh_set_message_callback, 70
- libssh_session
 - ssh_blocking_flush, 142
 - ssh_clean_pubkey_hash, 142
 - ssh_connect, 142
 - ssh_disconnect, 143
 - ssh_free, 143
 - ssh_get_disconnect_message, 144
 - ssh_get_fd, 144
 - ssh_get_issue_banner, 144
 - ssh_get_openssh_version, 145
 - ssh_get_pubkey_hash, 145
 - ssh_get_status, 145
 - ssh_get_version, 146
 - ssh_is_blocking, 146
 - ssh_is_connected, 146
 - ssh_is_server_known, 146
 - ssh_new, 147
 - ssh_options_copy, 147
 - ssh_options_getopt, 148
 - ssh_options_parse_config, 148
 - ssh_options_set, 149
 - ssh_select, 152
 - ssh_set_blocking, 152
 - ssh_set_fd_except, 153
 - ssh_set_fd_toread, 153
 - ssh_set_fd_towrite, 153
 - ssh_silent_disconnect, 153
 - ssh_write_knownhost, 154
- libssh_sftp
 - sftp_async_read, 75
 - sftp_async_read_begin, 75
 - sftp_attributes_free, 76
 - sftp_canonicalize_path, 76
 - sftp_chmod, 76
 - sftp_chown, 76
 - sftp_close, 77
 - sftp_closedir, 77
 - sftp_dir_eof, 77
 - sftp_extension_supported, 78
 - sftp_extensions_get_count, 78
 - sftp_extensions_get_data, 78
 - sftp_extensions_get_name, 79
 - sftp_free, 79
 - sftp_fstat, 79
 - sftp_fstatvfs, 79
 - sftp_get_error, 80
 - sftp_init, 80
 - sftp_lstat, 80
 - sftp_mkdir, 80
 - sftp_new, 81
 - sftp_open, 81
 - sftp_opendir, 81
 - sftp_read, 82
 - sftp_readdir, 82
 - sftp_readlink, 82
 - sftp_rename, 83
 - sftp_rewind, 83
 - sftp_rmdir, 83
 - sftp_seek, 83
 - sftp_seek64, 84
 - sftp_server_version, 84
 - sftp_setstat, 84
 - sftp_stat, 85
 - sftp_statvfs, 85
 - sftp_statvfs_free, 85
 - sftp_symlink, 85
 - sftp_tell, 86
 - sftp_tell64, 86
 - sftp_unlink, 86
 - sftp_utimes, 86
 - sftp_write, 87
- libssh_string

- ssh_string_burn, [155](#)
- ssh_string_copy, [155](#)
- ssh_string_data, [155](#)
- ssh_string_fill, [156](#)
- ssh_string_free, [156](#)
- ssh_string_free_char, [156](#)
- ssh_string_from_char, [157](#)
- ssh_string_len, [157](#)
- ssh_string_new, [157](#)
- ssh_string_to_char, [158](#)
- libssh_threads
 - ssh_threads_get_noop, [159](#)
 - ssh_threads_set_callbacks, [159](#)
- optionsCopy
 - ssh::Session, [169](#)
- optionsParseConfig
 - ssh::Session, [169](#)
- privatekey_free
 - libssh_auth, [89](#)
- privatekey_from_file
 - libssh_auth, [89](#)
- publickey_from_file
 - libssh_auth, [90](#)
- publickey_from_privatekey
 - libssh_auth, [90](#)
- publickey_to_string
 - libssh_auth, [91](#)
- setOption
 - ssh::Session, [169](#), [170](#)
- sftp_async_read
 - libssh_sftp, [75](#)
- sftp_async_read_begin
 - libssh_sftp, [75](#)
- sftp_attributes_free
 - libssh_sftp, [76](#)
- sftp_canonicalize_path
 - libssh_sftp, [76](#)
- sftp_chmod
 - libssh_sftp, [76](#)
- sftp_chown
 - libssh_sftp, [76](#)
- sftp_close
 - libssh_sftp, [77](#)
- sftp_closedir
 - libssh_sftp, [77](#)
- sftp_dir_eof
 - libssh_sftp, [77](#)
- sftp_extension_supported
 - libssh_sftp, [78](#)
- sftp_extensions_get_count
 - libssh_sftp, [78](#)
- sftp_extensions_get_data
 - libssh_sftp, [78](#)
- sftp_extensions_get_name
 - libssh_sftp, [79](#)
- sftp_free
 - libssh_sftp, [79](#)
- sftp_fstat
 - libssh_sftp, [79](#)
- sftp_fstatvfs
 - libssh_sftp, [79](#)
- sftp_get_error
 - libssh_sftp, [80](#)
- sftp_init
 - libssh_sftp, [80](#)
- sftp_lstat
 - libssh_sftp, [80](#)
- sftp_mkdir
 - libssh_sftp, [80](#)
- sftp_new
 - libssh_sftp, [81](#)
- sftp_open
 - libssh_sftp, [81](#)
- sftp_opendir
 - libssh_sftp, [81](#)
- sftp_read
 - libssh_sftp, [82](#)
- sftp_readdir
 - libssh_sftp, [82](#)
- sftp_readlink
 - libssh_sftp, [82](#)
- sftp_rename
 - libssh_sftp, [83](#)
- sftp_rewind
 - libssh_sftp, [83](#)
- sftp_rmdir
 - libssh_sftp, [83](#)
- sftp_seek
 - libssh_sftp, [83](#)
- sftp_seek64
 - libssh_sftp, [84](#)
- sftp_server_version
 - libssh_sftp, [84](#)
- sftp_setstat
 - libssh_sftp, [84](#)
- sftp_stat
 - libssh_sftp, [85](#)

- sftp_statvfs
 - libssh_sftp, 85
- sftp_statvfs_free
 - libssh_sftp, 85
- sftp_symlink
 - libssh_sftp, 85
- sftp_tell
 - libssh_sftp, 86
- sftp_tell64
 - libssh_sftp, 86
- sftp_unlink
 - libssh_sftp, 86
- sftp_utimes
 - libssh_sftp, 86
- sftp_write
 - libssh_sftp, 87
- silentDisconnect
 - ssh::Session, 171
- size
 - ssh_bind_callbacks_struct, 174
 - ssh_callbacks_struct, 175
- ssh::Channel, 161
 - acceptX11, 162
 - changePtySize, 162
 - close, 162
 - isClosed, 163
 - isEof, 163
 - isOpen, 163
 - write, 163
- ssh::Session, 164
 - acceptForward, 166
 - connect, 166
 - disconnect, 166
 - getAuthList, 167
 - getDisconnectMessage, 167
 - getIssueBanner, 167
 - getOpensshVersion, 167
 - getSocket, 168
 - getVersion, 168
 - isServerKnown, 168
 - optionsCopy, 169
 - optionsParseConfig, 169
 - setOption, 169, 170
 - silentDisconnect, 171
 - userauthAutopubkey, 171
 - userauthNone, 171
 - userauthOfferPubkey, 171
 - userauthPassword, 172
 - userauthPubkey, 172
 - writeKnownhost, 173
- ssh::SshException, 176
 - getCode, 177
 - getError, 177
- SSH_LOG_FUNCTIONS
 - libssh_log, 121
- SSH_LOG_NOLOG
 - libssh_log, 121
- SSH_LOG_PACKET
 - libssh_log, 121
- SSH_LOG_PROTOCOL
 - libssh_log, 121
- SSH_LOG_RARE
 - libssh_log, 121
- ssh_auth_callback
 - libssh_callbacks, 60
- ssh_auth_list
 - libssh_auth, 91
- ssh_basename
 - libssh_misc, 123
- ssh_bind_accept
 - libssh_server, 66
- ssh_bind_callbacks_struct, 173
 - incoming_connection, 174
 - size, 174
- ssh_bind_fd_toaccept
 - libssh_server, 67
- ssh_bind_free
 - libssh_server, 67
- ssh_bind_get_fd
 - libssh_server, 67
- ssh_bind_incoming_connection_callback
 - libssh_server, 66
- ssh_bind_listen
 - libssh_server, 67
- ssh_bind_new
 - libssh_server, 67
- ssh_bind_options_set
 - libssh_server, 68
- ssh_bind_set_blocking
 - libssh_server, 69
- ssh_bind_set_callbacks
 - libssh_server, 69
- ssh_bind_set_fd
 - libssh_server, 70
- ssh_blocking_flush
 - libssh_session, 142
- ssh_buffer_free
 - libssh_buffer, 100
- ssh_buffer_get_begin
 - libssh_buffer, 101

- ssh_buffer_get_len
 - libssh_buffer, 101
- ssh_buffer_new
 - libssh_buffer, 101
- ssh_callbacks_init
 - libssh_callbacks, 59
- ssh_callbacks_struct, 174
 - auth_function, 175
 - size, 175
 - userdata, 175
- ssh_channel_accept_x11
 - libssh_channel, 105
- ssh_channel_change_pty_size
 - libssh_channel, 105
- ssh_channel_close
 - libssh_channel, 105
- ssh_channel_close_callback
 - libssh_callbacks, 60
- ssh_channel_data_callback
 - libssh_callbacks, 60
- ssh_channel_eof_callback
 - libssh_callbacks, 60
- ssh_channel_exit_signal_callback
 - libssh_callbacks, 61
- ssh_channel_exit_status_callback
 - libssh_callbacks, 61
- ssh_channel_free
 - libssh_channel, 106
- ssh_channel_get_exit_status
 - libssh_channel, 106
- ssh_channel_get_session
 - libssh_channel, 106
- ssh_channel_is_closed
 - libssh_channel, 107
- ssh_channel_is_eof
 - libssh_channel, 107
- ssh_channel_is_open
 - libssh_channel, 107
- ssh_channel_new
 - libssh_channel, 108
- ssh_channel_open_forward
 - libssh_channel, 108
- ssh_channel_open_session
 - libssh_channel, 108
- ssh_channel_poll
 - libssh_channel, 109
- ssh_channel_read
 - libssh_channel, 109
- ssh_channel_read_nonblocking
 - libssh_channel, 110
- ssh_channel_request_env
 - libssh_channel, 110
- ssh_channel_request_exec
 - libssh_channel, 111
- ssh_channel_request_pty
 - libssh_channel, 111
- ssh_channel_request_pty_size
 - libssh_channel, 112
- ssh_channel_request_send_signal
 - libssh_channel, 112
- ssh_channel_request_shell
 - libssh_channel, 113
- ssh_channel_request_subsystem
 - libssh_channel, 113
- ssh_channel_request_x11
 - libssh_channel, 114
- ssh_channel_select
 - libssh_channel, 114
- ssh_channel_send_eof
 - libssh_channel, 115
- ssh_channel_set_blocking
 - libssh_channel, 115
- ssh_channel_signal_callback
 - libssh_callbacks, 61
- ssh_channel_write
 - libssh_channel, 115
- ssh_clean_pubkey_hash
 - libssh_session, 142
- ssh_connect
 - libssh_session, 142
- ssh_dirname
 - libssh_misc, 123
- ssh_disconnect
 - libssh_session, 143
- ssh_finalize
 - libssh, 119
- ssh_forward_accept
 - libssh_channel, 116
- ssh_forward_cancel
 - libssh_channel, 116
- ssh_forward_listen
 - libssh_channel, 116
- ssh_free
 - libssh_session, 143
- ssh_get_disconnect_message
 - libssh_session, 144
- ssh_get_error
 - libssh_error, 117
- ssh_get_error_code
 - libssh_error, 118

- ssh_get_fd
 - libssh_session, 144
- ssh_get_issue_banner
 - libssh_session, 144
- ssh_get_openssh_version
 - libssh_session, 145
- ssh_get_pubkey_hash
 - libssh_session, 145
- ssh_get_status
 - libssh_session, 145
- ssh_get_version
 - libssh_session, 146
- ssh_getpass
 - libssh_misc, 124
- ssh_global_request_callback
 - libssh_callbacks, 62
- ssh_handle_key_exchange
 - libssh_server, 70
- ssh_init
 - libssh, 120
- ssh_is_blocking
 - libssh_session, 146
- ssh_is_connected
 - libssh_session, 146
- ssh_is_server_known
 - libssh_session, 146
- ssh_key_clean
 - libssh_pki, 126
- ssh_key_free
 - libssh_pki, 127
- ssh_key_import_private
 - libssh_pki, 127
- ssh_key_new
 - libssh_pki, 127
- ssh_key_type
 - libssh_pki, 127
- ssh_log
 - libssh_log, 121
- ssh_log_callback
 - libssh_callbacks, 62
- ssh_message_get
 - libssh_messages, 122
- ssh_mkdir
 - libssh_misc, 124
- ssh_new
 - libssh_session, 147
- ssh_options_copy
 - libssh_session, 147
- ssh_options_getopt
 - libssh_session, 148
- ssh_options_parse_config
 - libssh_session, 148
- ssh_options_set
 - libssh_session, 149
- SSH_PACKET_CALLBACK
 - libssh_callbacks, 59
- ssh_packet_callback
 - libssh_callbacks, 62
- SSH_PACKET_USED
 - libssh_callbacks, 59
- ssh_path_expand_tilde
 - libssh_misc, 125
- ssh_poll_add_events
 - libssh_poll, 129
- ssh_poll_ctx_add
 - libssh_poll, 129
- ssh_poll_ctx_add_socket
 - libssh_poll, 130
- ssh_poll_ctx_dopoll
 - libssh_poll, 130
- ssh_poll_ctx_free
 - libssh_poll, 130
- ssh_poll_ctx_new
 - libssh_poll, 131
- ssh_poll_ctx_remove
 - libssh_poll, 131
- ssh_poll_free
 - libssh_poll, 131
- ssh_poll_get_ctx
 - libssh_poll, 131
- ssh_poll_get_events
 - libssh_poll, 132
- ssh_poll_get_fd
 - libssh_poll, 132
- ssh_poll_new
 - libssh_poll, 132
- ssh_poll_remove_events
 - libssh_poll, 133
- ssh_poll_set_callback
 - libssh_poll, 133
- ssh_poll_set_events
 - libssh_poll, 133
- ssh_poll_set_fd
 - libssh_poll, 133
- ssh_privatekey_type
 - libssh_auth, 91
- ssh_publickey_to_file
 - libssh_auth, 92
- ssh_scp_accept_request
 - libssh_scp, 135

- ssh_scp_deny_request
 - libssh_scp, 135
- ssh_scp_integer_mode
 - libssh_scp, 135
- ssh_scp_leave_directory
 - libssh_scp, 136
- ssh_scp_new
 - libssh_scp, 136
- ssh_scp_pull_request
 - libssh_scp, 136
- ssh_scp_push_directory
 - libssh_scp, 137
- ssh_scp_push_file
 - libssh_scp, 137
- ssh_scp_read
 - libssh_scp, 138
- ssh_scp_read_string
 - libssh_scp, 138
- ssh_scp_request_get_filename
 - libssh_scp, 138
- ssh_scp_request_get_permissions
 - libssh_scp, 138
- ssh_scp_request_get_size
 - libssh_scp, 139
- ssh_scp_request_get_warning
 - libssh_scp, 139
- ssh_scp_string_mode
 - libssh_scp, 139
- ssh_scp_write
 - libssh_scp, 139
- ssh_select
 - libssh_session, 152
- ssh_set_blocking
 - libssh_session, 152
- ssh_set_callbacks
 - libssh_callbacks, 63
- ssh_set_channel_callbacks
 - libssh_callbacks, 63
- ssh_set_fd_except
 - libssh_session, 153
- ssh_set_fd_toread
 - libssh_session, 153
- ssh_set_fd_towrite
 - libssh_session, 153
- ssh_set_message_callback
 - libssh_server, 70
- ssh_silent_disconnect
 - libssh_session, 153
- ssh_socket_callbacks_struct, 175
 - controlflow, 176
 - data, 176
 - exception, 176
 - userdata, 176
- ssh_status_callback
 - libssh_callbacks, 63
- ssh_string_burn
 - libssh_string, 155
- ssh_string_copy
 - libssh_string, 155
- ssh_string_data
 - libssh_string, 155
- ssh_string_fill
 - libssh_string, 156
- ssh_string_free
 - libssh_string, 156
- ssh_string_free_char
 - libssh_string, 156
- ssh_string_from_char
 - libssh_string, 157
- ssh_string_len
 - libssh_string, 157
- ssh_string_new
 - libssh_string, 157
- ssh_string_to_char
 - libssh_string, 158
- ssh_threads_get_noop
 - libssh_threads, 159
- ssh_threads_set_callbacks
 - libssh_threads, 159
- ssh_timeout_update
 - libssh_misc, 125
- ssh_try_publickey_from_file
 - libssh_auth, 92
- ssh_userauth_agent_pubkey
 - libssh_auth, 93
- ssh_userauth_autopubkey
 - libssh_auth, 93
- ssh_userauth_kbdint
 - libssh_auth, 94
- ssh_userauth_kbdint_getinstruction
 - libssh_auth, 94
- ssh_userauth_kbdint_getname
 - libssh_auth, 95
- ssh_userauth_kbdint_getnprompts
 - libssh_auth, 95
- ssh_userauth_kbdint_getprompt
 - libssh_auth, 95
- ssh_userauth_kbdint_setanswer
 - libssh_auth, 96
- ssh_userauth_list

- libssh_auth, 96
- ssh_userauth_none
 - libssh_auth, 97
- ssh_userauth_offer_pubkey
 - libssh_auth, 97
- ssh_userauth_password
 - libssh_auth, 98
- ssh_userauth_privatekey_file
 - libssh_auth, 98
- ssh_userauth_pubkey
 - libssh_auth, 99
- ssh_version
 - libssh_misc, 125
- ssh_write_knownhost
 - libssh_session, 154
- The libssh API, 118
- The libssh C++ wrapper, 64
- The libssh callbacks, 57
- The libssh server API, 65
- The libssh SFTP API, 71
- The SSH authentication functions., 87
- The SSH buffer functions., 100
- The SSH channel functions, 102
- The SSH error functions., 117
- The SSH helper functions., 122
- The SSH logging functions., 120
- The SSH message functions, 121
- The SSH poll functions., 128
- The SSH Public Key Infrastructure, 126
- The SSH scp functions, 134
- The SSH session functions., 140
- The SSH string functions, 154
- The SSH threading functions., 158
- userauthAutopubkey
 - ssh::Session, 171
- userauthNone
 - ssh::Session, 171
- userauthOfferPubkey
 - ssh::Session, 171
- userauthPassword
 - ssh::Session, 172
- userauthPubkey
 - ssh::Session, 172
- userdata
 - ssh_callbacks_struct, 175
 - ssh_socket_callbacks_struct, 176
- write
 - ssh::Channel, 163
 - writeKnownhost
 - ssh::Session, 173