

NAME

ssl – OpenSSL SSL/TLS library

SYNOPSIS

See the individual manual pages for details.

DESCRIPTION

The OpenSSL **ssl** library implements the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols. It provides a rich API which is documented here.

An **SSL_CTX** object is created as a framework to establish TLS/SSL enabled connections (see **SSL_CTX_new**(3)). Various options regarding certificates, algorithms etc. can be set in this object.

When a network connection has been created, it can be assigned to an **SSL** object. After the **SSL** object has been created using **SSL_new**(3), **SSL_set_fd**(3) or **SSL_set_bio**(3) can be used to associate the network connection with the object.

When the TLS/SSL handshake is performed using **SSL_accept**(3) or **SSL_connect**(3) respectively. **SSL_read_ex**(3), **SSL_read**(3), **SSL_write_ex**(3) and **SSL_write**(3) are used to read and write data on the TLS/SSL connection. **SSL_shutdown**(3) can be used to shut down the TLS/SSL connection.

DATA STRUCTURES

Currently the OpenSSL **ssl** library functions deals with the following data structures:

SSL_METHOD (SSL Method)

This is a dispatch structure describing the internal **ssl** library methods/functions which implement the various protocol versions (SSLv3 TLSv1, ...). It's needed to create an **SSL_CTX**.

SSL_CIPHER (SSL Cipher)

This structure holds the algorithm information for a particular cipher which are a core part of the SSL/TLS protocol. The available ciphers are configured on a **SSL_CTX** basis and the actual ones used are then part of the **SSL_SESSION**.

SSL_CTX (SSL Context)

This is the global context structure which is created by a server or client once per program life-time and which holds mainly default values for the **SSL** structures which are later created for the connections.

SSL_SESSION (SSL Session)

This is a structure containing the current TLS/SSL session details for a connection: **SSL_CIPHER**s, client and server certificates, keys, etc.

SSL (SSL Connection)

This is the main SSL/TLS structure which is created by a server or client per established connection. This actually is the core structure in the SSL API. At run-time the application usually deals with this structure which has links to mostly all other structures.

HEADER FILES

Currently the OpenSSL **ssl** library provides the following C header files containing the prototypes for the data structures and functions:

ssl.h

This is the common header file for the SSL/TLS API. Include it into your program to make the API of the **ssl** library available. It internally includes both more private SSL headers and headers from the **crypto** library. Whenever you need hard-core details on the internals of the SSL API, look inside this header file.

ssl2.h

Unused. Present for backwards compatibility only.

ssl3.h

This is the sub header file dealing with the SSLv3 protocol only. *Usually you don't have to include it explicitly because it's already included by ssl.h.*

tls1.h

This is the sub header file dealing with the TLSv1 protocol only. *Usually you don't have to include it explicitly because it's already included by ssl.h.*

API FUNCTIONS

Currently the OpenSSL **ssl** library exports 214 API functions. They are documented in the following:

Dealing with Protocol Methods

Here we document the various API functions which deal with the SSL/TLS protocol methods defined in **SSL_METHOD** structures.

```
const SSL_METHOD *TLS_method(void);
```

Constructor for the *version-flexible* **SSL_METHOD** structure for clients, servers or both. See **SSL_CTX_new**(3) for details.

```
const SSL_METHOD *TLS_client_method(void);
```

Constructor for the *version-flexible* **SSL_METHOD** structure for clients. Must be used to support the TLSv1.3 protocol.

```
const SSL_METHOD *TLS_server_method(void);
```

Constructor for the *version-flexible* **SSL_METHOD** structure for servers. Must be used to support the TLSv1.3 protocol.

```
const SSL_METHOD *TLSv1_2_method(void);
```

Constructor for the TLSv1.2 **SSL_METHOD** structure for clients, servers or both.

```
const SSL_METHOD *TLSv1_2_client_method(void);
```

Constructor for the TLSv1.2 **SSL_METHOD** structure for clients.

```
const SSL_METHOD *TLSv1_2_server_method(void);
```

Constructor for the TLSv1.2 **SSL_METHOD** structure for servers.

```
const SSL_METHOD *TLSv1_1_method(void);
```

Constructor for the TLSv1.1 **SSL_METHOD** structure for clients, servers or both.

```
const SSL_METHOD *TLSv1_1_client_method(void);
```

Constructor for the TLSv1.1 **SSL_METHOD** structure for clients.

```
const SSL_METHOD *TLSv1_1_server_method(void);
```

Constructor for the TLSv1.1 **SSL_METHOD** structure for servers.

```
const SSL_METHOD *TLSv1_method(void);
```

Constructor for the TLSv1 **SSL_METHOD** structure for clients, servers or both.

```
const SSL_METHOD *TLSv1_client_method(void);
```

Constructor for the TLSv1 **SSL_METHOD** structure for clients.

```
const SSL_METHOD *TLSv1_server_method(void);
```

Constructor for the TLSv1 **SSL_METHOD** structure for servers.

```
const SSL_METHOD *SSLv3_method(void);
```

Constructor for the SSLv3 **SSL_METHOD** structure for clients, servers or both.

```
const SSL_METHOD *SSLv3_client_method(void);
```

Constructor for the SSLv3 **SSL_METHOD** structure for clients.

```
const SSL_METHOD *SSLv3_server_method(void);
```

Constructor for the SSLv3 **SSL_METHOD** structure for servers.

Dealing with Ciphers

Here we document the various API functions which deal with the SSL/TLS ciphers defined in **SSL_CIPHER** structures.

```
char *SSL_CIPHER_description(SSL_CIPHER *cipher, char *buf, int len);
```

Write a string to *buf* (with a maximum size of *len*) containing a human readable description of *cipher*. Returns *buf*.

```
int SSL_CIPHER_get_bits(SSL_CIPHER *cipher, int *alg_bits);
```

Determine the number of bits in *cipher*. Because of export crippled ciphers there are two bits: The bits the algorithm supports in general (stored to *alg_bits*) and the bits which are actually used (the return value).

```
const char *SSL_CIPHER_get_name(SSL_CIPHER *cipher);
```

Return the internal name of *cipher* as a string. These are the various strings defined by the *SSL3_TXT_xxx* and *TLS1_TXT_xxx* definitions in the header files.

```
const char *SSL_CIPHER_get_version(SSL_CIPHER *cipher);
```

Returns a string like "SSLv3" or "TLSv1.2" which indicates the SSL/TLS protocol version to which *cipher* belongs (i.e. where it was defined in the specification the first time).

Dealing with Protocol Contexts

Here we document the various API functions which deal with the SSL/TLS protocol context defined in the **SSL_CTX** structure.

```
int SSL_CTX_add_client_CA(SSL_CTX *ctx, X509 *x);
long SSL_CTX_add_extra_chain_cert(SSL_CTX *ctx, X509 *x509);
int SSL_CTX_add_session(SSL_CTX *ctx, SSL_SESSION *c);
int SSL_CTX_check_private_key(const SSL_CTX *ctx);
long SSL_CTX_ctrl(SSL_CTX *ctx, int cmd, long larg, char *parg);
void SSL_CTX_flush_sessions(SSL_CTX *s, long t);
void SSL_CTX_free(SSL_CTX *a);
char *SSL_CTX_get_app_data(SSL_CTX *ctx);
X509_STORE *SSL_CTX_get_cert_store(SSL_CTX *ctx);
STACK *SSL_CTX_get_ciphers(const SSL_CTX *ctx);
STACK *SSL_CTX_get_client_CA_list(const SSL_CTX *ctx);
int (*SSL_CTX_get_client_cert_cb(SSL_CTX *ctx))(SSL *ssl, X509 **x509, EVP_PKEY **pkey);
void SSL_CTX_get_default_read_ahead(SSL_CTX *ctx);
char *SSL_CTX_get_ex_data(const SSL_CTX *s, int idx);
int SSL_CTX_get_ex_new_index(long arg1, char *argp, int (*new_func)(void), int (*dup_func)(void), void (*free_func)(void));
void (*SSL_CTX_get_info_callback(SSL_CTX *ctx))(SSL *ssl, int cb, int ret);
int SSL_CTX_get_quiet_shutdown(const SSL_CTX *ctx);
void SSL_CTX_get_read_ahead(SSL_CTX *ctx);
int SSL_CTX_get_session_cache_mode(SSL_CTX *ctx);
long SSL_CTX_get_timeout(const SSL_CTX *ctx);
int (*SSL_CTX_get_verify_callback(const SSL_CTX *ctx))(int ok, X509_STORE_CTX *ctx);
int SSL_CTX_get_verify_mode(SSL_CTX *ctx);
int SSL_CTX_load_verify_locations(SSL_CTX *ctx, const char *CAfile, const char *CApath);
SSL_CTX *SSL_CTX_new(const SSL_METHOD *meth);
int SSL_CTX_up_ref(SSL_CTX *ctx);
int SSL_CTX_remove_session(SSL_CTX *ctx, SSL_SESSION *c);
int SSL_CTX_sess_accept(SSL_CTX *ctx);
int SSL_CTX_sess_accept_good(SSL_CTX *ctx);
int SSL_CTX_sess_accept_renegotiate(SSL_CTX *ctx);
int SSL_CTX_sess_cache_full(SSL_CTX *ctx);
int SSL_CTX_sess_cb_hits(SSL_CTX *ctx);
int SSL_CTX_sess_connect(SSL_CTX *ctx);
int SSL_CTX_sess_connect_good(SSL_CTX *ctx);
int SSL_CTX_sess_connect_renegotiate(SSL_CTX *ctx);
int SSL_CTX_sess_get_cache_size(SSL_CTX *ctx);
SSL_SESSION *(*SSL_CTX_sess_get_get_cb(SSL_CTX *ctx))(SSL *ssl, unsigned char *data, int len, int *copy);
```

```

int (*SSL_CTX_sess_get_new_cb(SSL_CTX *ctx)(SSL *ssl, SSL_SESSION *sess);
void (*SSL_CTX_sess_get_remove_cb(SSL_CTX *ctx)(SSL_CTX *ctx, SSL_SESSION *sess);
int SSL_CTX_sess_hits(SSL_CTX *ctx);
int SSL_CTX_sess_misses(SSL_CTX *ctx);
int SSL_CTX_sess_number(SSL_CTX *ctx);
void SSL_CTX_sess_set_cache_size(SSL_CTX *ctx, t);
void SSL_CTX_sess_set_get_cb(SSL_CTX *ctx, SSL_SESSION *(*cb)(SSL *ssl, unsigned char *data, int
len, int *copy));
void SSL_CTX_sess_set_new_cb(SSL_CTX *ctx, int (*cb)(SSL *ssl, SSL_SESSION *sess));
void SSL_CTX_sess_set_remove_cb(SSL_CTX *ctx, void (*cb)(SSL_CTX *ctx, SSL_SESSION *sess));
int SSL_CTX_sess_timeouts(SSL_CTX *ctx);
LHASH *SSL_CTX_sessions(SSL_CTX *ctx);
int SSL_CTX_set_app_data(SSL_CTX *ctx, void *arg);
void SSL_CTX_set_cert_store(SSL_CTX *ctx, X509_STORE *cs);
void SSL_CTX_set1_cert_store(SSL_CTX *ctx, X509_STORE *cs);
void SSL_CTX_set_cert_verify_cb(SSL_CTX *ctx, int (*cb)(), char *arg)
int SSL_CTX_set_cipher_list(SSL_CTX *ctx, char *str);
void SSL_CTX_set_client_CA_list(SSL_CTX *ctx, STACK *list);
void SSL_CTX_set_client_cert_cb(SSL_CTX *ctx, int (*cb)(SSL *ssl, X509 **x509, EVP_PKEY
**pkey));
int SSL_CTX_set_ct_validation_callback(SSL_CTX *ctx, ssl_ct_validation_cb callback, void *arg);
void SSL_CTX_set_default_passwd_cb(SSL_CTX *ctx, int (*cb);(void))
void SSL_CTX_set_default_read_ahead(SSL_CTX *ctx, int m);
int SSL_CTX_set_default_verify_paths(SSL_CTX *ctx);
    Use the default paths to locate trusted CA certificates. There is one default directory path and one
    default file path. Both are set via this call.

int SSL_CTX_set_default_verify_dir(SSL_CTX *ctx)
    Use the default directory path to locate trusted CA certificates.

int SSL_CTX_set_default_verify_file(SSL_CTX *ctx)
    Use the file path to locate trusted CA certificates.

int SSL_CTX_set_ex_data(SSL_CTX *s, int idx, char *arg);
void SSL_CTX_set_info_callback(SSL_CTX *ctx, void (*cb)(SSL *ssl, int cb, int ret));
void SSL_CTX_set_msg_callback(SSL_CTX *ctx, void (*cb)(int write_p, int version, int content_type,
const void *buf, size_t len, SSL *ssl, void *arg));
void SSL_CTX_set_msg_callback_arg(SSL_CTX *ctx, void *arg);
unsigned long SSL_CTX_clear_options(SSL_CTX *ctx, unsigned long op);
unsigned long SSL_CTX_get_options(SSL_CTX *ctx);
unsigned long SSL_CTX_set_options(SSL_CTX *ctx, unsigned long op);
void SSL_CTX_set_quiet_shutdown(SSL_CTX *ctx, int mode);
void SSL_CTX_set_read_ahead(SSL_CTX *ctx, int m);
void SSL_CTX_set_session_cache_mode(SSL_CTX *ctx, int mode);
int SSL_CTX_set_ssl_version(SSL_CTX *ctx, const SSL_METHOD *meth);
void SSL_CTX_set_timeout(SSL_CTX *ctx, long t);
long SSL_CTX_set_tmp_dh(SSL_CTX *ctx, DH *dh);
long SSL_CTX_set_tmp_dh_callback(SSL_CTX *ctx, DH *(*cb)(void));
void SSL_CTX_set_verify(SSL_CTX *ctx, int mode, int (*cb);(void))
int SSL_CTX_use_PrivateKey(SSL_CTX *ctx, EVP_PKEY *pkey);
int SSL_CTX_use_PrivateKey_ASN1(int type, SSL_CTX *ctx, unsigned char *d, long len);
int SSL_CTX_use_PrivateKey_file(SSL_CTX *ctx, const char *file, int type);
int SSL_CTX_use_RSAPrivateKey(SSL_CTX *ctx, RSA *rsa);
int SSL_CTX_use_RSAPrivateKey_ASN1(SSL_CTX *ctx, unsigned char *d, long len);

```

```

int SSL_CTX_use_RSAPrivateKey_file(SSL_CTX *ctx, const char *file, int type);
int SSL_CTX_use_certificate(SSL_CTX *ctx, X509 *x);
int SSL_CTX_use_certificate_ASN1(SSL_CTX *ctx, int len, unsigned char *d);
int SSL_CTX_use_certificate_file(SSL_CTX *ctx, const char *file, int type);
int SSL_CTX_use_cert_and_key(SSL_CTX *ctx, X509 *x, EVP_PKEY *pkey, STACK_OF(X509) *chain,
int override);
X509 *SSL_CTX_get0_certificate(const SSL_CTX *ctx);
EVP_PKEY *SSL_CTX_get0_privatekey(const SSL_CTX *ctx);
void SSL_CTX_set_psk_client_callback(SSL_CTX *ctx, unsigned int (*callback)(SSL *ssl, const char
*hint, char *identity, unsigned int max_identity_len, unsigned char *psk, unsigned int max_psk_len));
int SSL_CTX_use_psk_identity_hint(SSL_CTX *ctx, const char *hint);
void SSL_CTX_set_psk_server_callback(SSL_CTX *ctx, unsigned int (*callback)(SSL *ssl, const char
*identity, unsigned char *psk, int max_psk_len));

```

Dealing with Sessions

Here we document the various API functions which deal with the SSL/TLS sessions defined in the **SSL_SESSION** structures.

```

int SSL_SESSION_cmp(const SSL_SESSION *a, const SSL_SESSION *b);
void SSL_SESSION_free(SSL_SESSION *ss);
char *SSL_SESSION_get_app_data(SSL_SESSION *s);
char *SSL_SESSION_get_ex_data(const SSL_SESSION *s, int idx);
int SSL_SESSION_get_ex_new_index(long arg1, char *argp, int (*new_func)(void), int
(*dup_func)(void), void (*free_func)(void));
long SSL_SESSION_get_time(const SSL_SESSION *s);
long SSL_SESSION_get_timeout(const SSL_SESSION *s);
unsigned long SSL_SESSION_hash(const SSL_SESSION *a);
SSL_SESSION *SSL_SESSION_new(void);
int SSL_SESSION_print(BIO *bp, const SSL_SESSION *x);
int SSL_SESSION_print_fp(FILE *fp, const SSL_SESSION *x);
int SSL_SESSION_set_app_data(SSL_SESSION *s, char *a);
int SSL_SESSION_set_ex_data(SSL_SESSION *s, int idx, char *arg);
long SSL_SESSION_set_time(SSL_SESSION *s, long t);
long SSL_SESSION_set_timeout(SSL_SESSION *s, long t);

```

Dealing with Connections

Here we document the various API functions which deal with the SSL/TLS connection defined in the **SSL** structure.

```

int SSL_accept(SSL *ssl);
int SSL_add_dir_cert_subjects_to_stack(STACK *stack, const char *dir);
int SSL_add_file_cert_subjects_to_stack(STACK *stack, const char *file);
int SSL_add_client_CA(SSL *ssl, X509 *x);
char *SSL_alert_desc_string(int value);
char *SSL_alert_desc_string_long(int value);
char *SSL_alert_type_string(int value);
char *SSL_alert_type_string_long(int value);
int SSL_check_private_key(const SSL *ssl);
void SSL_clear(SSL *ssl);
long SSL_clear_num_renegotiations(SSL *ssl);
int SSL_connect(SSL *ssl);
int SSL_copy_session_id(SSL *t, const SSL *f);
    Sets the session details for t to be the same as in f. Returns 1 on success or 0 on failure.
long SSL_ctrl(SSL *ssl, int cmd, long larg, char *parg);

```

```
int SSL_do_handshake(SSL *ssl);
SSL *SSL_dup(SSL *ssl);
```

SSL_dup() allows applications to configure an SSL handle for use in multiple SSL connections, and then duplicate it prior to initiating each connection with the duplicated handle. Use of **SSL_dup()** avoids the need to repeat the configuration of the handles for each connection.

For **SSL_dup()** to work, the connection **MUST** be in its initial state and **MUST NOT** have not yet have started the SSL handshake. For connections that are not in their initial state **SSL_dup()** just increments an internal reference count and returns the *same* handle. It may be possible to use **SSL_clear** (3) to recycle an SSL handle that is not in its initial state for re-use, but this is best avoided. Instead, save and restore the session, if desired, and construct a fresh handle for each connection.

```
STACK *SSL_dup_CA_list(STACK *sk);
void SSL_free(SSL *ssl);
SSL_CTX *SSL_get_SSL_CTX(const SSL *ssl);
char *SSL_get_app_data(SSL *ssl);
X509 *SSL_get_certificate(const SSL *ssl);
const char *SSL_get_cipher(const SSL *ssl);
int SSL_is_dtls(const SSL *ssl);
int SSL_get_cipher_bits(const SSL *ssl, int *alg_bits);
char *SSL_get_cipher_list(const SSL *ssl, int n);
char *SSL_get_cipher_name(const SSL *ssl);
char *SSL_get_cipher_version(const SSL *ssl);
STACK *SSL_get_ciphers(const SSL *ssl);
STACK *SSL_get_client_CA_list(const SSL *ssl);
SSL_CIPHER *SSL_get_current_cipher(SSL *ssl);
long SSL_get_default_timeout(const SSL *ssl);
int SSL_get_error(const SSL *ssl, int i);
char *SSL_get_ex_data(const SSL *ssl, int idx);
int SSL_get_ex_data_X509_STORE_CTX_idx(void);
int SSL_get_ex_new_index(long argl, char *argp, int (*new_func)(void), int (*dup_func)(void), void (*free_func)(void));
int SSL_get_fd(const SSL *ssl);
void (*SSL_get_info_callback(const SSL *ssl))();
int SSL_get_key_update_type(SSL *s);
STACK *SSL_get_peer_cert_chain(const SSL *ssl);
X509 *SSL_get_peer_certificate(const SSL *ssl);
const STACK_OF(SCT) *SSL_get0_peer_scts(SSL *s);
EVP_PKEY *SSL_get_privatekey(const SSL *ssl);
int SSL_get_quiet_shutdown(const SSL *ssl);
BIO *SSL_get_rbio(const SSL *ssl);
int SSL_get_read_ahead(const SSL *ssl);
SSL_SESSION *SSL_get_session(const SSL *ssl);
char *SSL_get_shared_ciphers(const SSL *ssl, char *buf, int size);
int SSL_get_shutdown(const SSL *ssl);
const SSL_METHOD *SSL_get_ssl_method(SSL *ssl);
int SSL_get_state(const SSL *ssl);
long SSL_get_time(const SSL *ssl);
long SSL_get_timeout(const SSL *ssl);
int (*SSL_get_verify_callback(const SSL *ssl))(int, X509_STORE_CTX *);
int SSL_get_verify_mode(const SSL *ssl);
long SSL_get_verify_result(const SSL *ssl);
char *SSL_get_version(const SSL *ssl);
BIO *SSL_get_wbio(const SSL *ssl);
```

```

int SSL_in_accept_init(SSL *ssl);
int SSL_in_before(SSL *ssl);
int SSL_in_connect_init(SSL *ssl);
int SSL_in_init(SSL *ssl);
int SSL_is_init_finished(SSL *ssl);
int SSL_key_update(SSL *s, int updatetype);
STACK *SSL_load_client_CA_file(const char *file);
SSL *SSL_new(SSL_CTX *ctx);
int SSL_up_ref(SSL *s);
long SSL_num_renegotiations(SSL *ssl);
int SSL_peek(SSL *ssl, void *buf, int num);
int SSL_pending(const SSL *ssl);
int SSL_read(SSL *ssl, void *buf, int num);
int SSL_renegotiate(SSL *ssl);
char *SSL_rstate_string(SSL *ssl);
char *SSL_rstate_string_long(SSL *ssl);
long SSL_session_reused(SSL *ssl);
void SSL_set_accept_state(SSL *ssl);
void SSL_set_app_data(SSL *ssl, char *arg);
void SSL_set_bio(SSL *ssl, BIO *rbio, BIO *wbio);
int SSL_set_cipher_list(SSL *ssl, char *str);
void SSL_set_client_CA_list(SSL *ssl, STACK *list);
void SSL_set_connect_state(SSL *ssl);
int SSL_set_ct_validation_callback(SSL *ssl, ssl_ct_validation_cb callback, void *arg);
int SSL_set_ex_data(SSL *ssl, int idx, char *arg);
int SSL_set_fd(SSL *ssl, int fd);
void SSL_set_info_callback(SSL *ssl, void (*cb);(void))
void SSL_set_msg_callback(SSL *ctx, void (*cb)(int write_p, int version, int content_type, const void
*buf, size_t len, SSL *ssl, void *arg));
void SSL_set_msg_callback_arg(SSL *ctx, void *arg);
unsigned long SSL_clear_options(SSL *ssl, unsigned long op);
unsigned long SSL_get_options(SSL *ssl);
unsigned long SSL_set_options(SSL *ssl, unsigned long op);
void SSL_set_quiet_shutdown(SSL *ssl, int mode);
void SSL_set_read_ahead(SSL *ssl, int yes);
int SSL_set_rfd(SSL *ssl, int fd);
int SSL_set_session(SSL *ssl, SSL_SESSION *session);
void SSL_set_shutdown(SSL *ssl, int mode);
int SSL_set_ssl_method(SSL *ssl, const SSL_METHOD *meth);
void SSL_set_time(SSL *ssl, long t);
void SSL_set_timeout(SSL *ssl, long t);
void SSL_set_verify(SSL *ssl, int mode, int (*callback);(void))
void SSL_set_verify_result(SSL *ssl, long arg);
int SSL_set_wfd(SSL *ssl, int fd);
int SSL_shutdown(SSL *ssl);
OSSL_HANDSHAKE_STATE SSL_get_state(const SSL *ssl);
    Returns the current handshake state.

char *SSL_state_string(const SSL *ssl);
char *SSL_state_string_long(const SSL *ssl);
long SSL_total_renegotiations(SSL *ssl);
int SSL_use_PrivateKey(SSL *ssl, EVP_PKEY *pkey);
int SSL_use_PrivateKey_ASN1(int type, SSL *ssl, unsigned char *d, long len);

```

```

int SSL_use_PrivateKey_file(SSL *ssl, const char *file, int type);
int SSL_use_RSAPrivateKey(SSL *ssl, RSA *rsa);
int SSL_use_RSAPrivateKey_ASN1(SSL *ssl, unsigned char *d, long len);
int SSL_use_RSAPrivateKey_file(SSL *ssl, const char *file, int type);
int SSL_use_certificate(SSL *ssl, X509 *x);
int SSL_use_certificate_ASN1(SSL *ssl, int len, unsigned char *d);
int SSL_use_certificate_file(SSL *ssl, const char *file, int type);
int SSL_use_cert_and_key(SSL *ssl, X509 *x, EVP_PKEY *pkey, STACK_OF(X509) *chain, int override);
int SSL_version(const SSL *ssl);
int SSL_want(const SSL *ssl);
int SSL_want_nothing(const SSL *ssl);
int SSL_want_read(const SSL *ssl);
int SSL_want_write(const SSL *ssl);
int SSL_want_x509_lookup(const SSL *ssl);
int SSL_write(SSL *ssl, const void *buf, int num);
void SSL_set_psk_client_callback(SSL *ssl, unsigned int (*callback)(SSL *ssl, const char *hint, char
*identity, unsigned int max_identity_len, unsigned char *psk, unsigned int max_psk_len));
int SSL_use_psk_identity_hint(SSL *ssl, const char *hint);
void SSL_set_psk_server_callback(SSL *ssl, unsigned int (*callback)(SSL *ssl, const char *identity,
unsigned char *psk, int max_psk_len));
const char *SSL_get_psk_identity_hint(SSL *ssl);
const char *SSL_get_psk_identity(SSL *ssl);

```

RETURN VALUES

See the individual manual pages for details.

SEE ALSO

openssl(1), crypto(7), **CRYPTO_get_ex_new_index**(3), **SSL_accept**(3), **SSL_clear**(3),
SSL_connect(3), **SSL_CIPHER_get_name**(3), **SSL_COMP_add_compression_method**(3),
SSL_CTX_add_extra_chain_cert(3), **SSL_CTX_add_session**(3), **SSL_CTX_ctrl**(3),
SSL_CTX_flush_sessions(3), **SSL_CTX_get_verify_mode**(3), **SSL_CTX_load_verify_locations**(3)
SSL_CTX_new(3), **SSL_CTX_sess_number**(3), **SSL_CTX_sess_set_cache_size**(3),
SSL_CTX_sess_set_get_cb(3), **SSL_CTX_sessions**(3), **SSL_CTX_set_cert_store**(3),
SSL_CTX_set_cert_verify_callback(3), **SSL_CTX_set_cipher_list**(3),
SSL_CTX_set_client_CA_list(3), **SSL_CTX_set_client_cert_cb**(3),
SSL_CTX_set_default_passwd_cb(3), **SSL_CTX_set_generate_session_id**(3),
SSL_CTX_set_info_callback(3), **SSL_CTX_set_max_cert_list**(3), **SSL_CTX_set_mode**(3),
SSL_CTX_set_msg_callback(3), **SSL_CTX_set_options**(3), **SSL_CTX_set_quiet_shutdown**(3),
SSL_CTX_set_read_ahead(3), **SSL_CTX_set_security_level**(3),
SSL_CTX_set_session_cache_mode(3), **SSL_CTX_set_session_id_context**(3),
SSL_CTX_set_ssl_version(3), **SSL_CTX_set_timeout**(3), **SSL_CTX_set_tmp_dh_callback**(3),
SSL_CTX_set_verify(3), **SSL_CTX_use_certificate**(3), **SSL_alert_type_string**(3),
SSL_do_handshake(3), **SSL_enable_ct**(3), **SSL_get_SSL_CTX**(3), **SSL_get_ciphers**(3),
SSL_get_client_CA_list(3), **SSL_get_default_timeout**(3), **SSL_get_error**(3),
SSL_get_ex_data_X509_STORE_CTX_idx(3), **SSL_get_fd**(3), **SSL_get_peer_cert_chain**(3),
SSL_get_rbio(3), **SSL_get_session**(3), **SSL_get_verify_result**(3), **SSL_get_version**(3),
SSL_load_client_CA_file(3), **SSL_new**(3), **SSL_pending**(3), **SSL_read_ex**(3), **SSL_read**(3),
SSL_rstate_string(3), **SSL_session_reused**(3), **SSL_set_bio**(3), **SSL_set_connect_state**(3),
SSL_set_fd(3), **SSL_set_session**(3), **SSL_set_shutdown**(3), **SSL_shutdown**(3), **SSL_state_string**(3),
SSL_want(3), **SSL_write_ex**(3), **SSL_write**(3), **SSL_SESSION_free**(3), **SSL_SESSION_get_time**(3),
d2i_SSL_SESSION(3), **SSL_CTX_set_psk_client_callback**(3), **SSL_CTX_use_psk_identity_hint**(3),
SSL_get_psk_identity(3), **DTLSv1_listen**(3)

HISTORY

SSLv2_client_method, **SSLv2_server_method** and **SSLv2_method** were removed in OpenSSL 1.1.0.

The return type of **SSL_copy_session_id** was changed from void to int in OpenSSL 1.1.0.

COPYRIGHT

Copyright 2000–2018 The OpenSSL Project Authors. All Rights Reserved.

Licensed under the OpenSSL license (the “License”). You may not use this file except in compliance with the License. You can obtain a copy in the file LICENSE in the source distribution or at [<https://www.openssl.org/source/license.html>](https://www.openssl.org/source/license.html).