Chapter 17: wolfSSL (formerly CyaSSL) API Reference

17.1 Initialization / Shutdown

The functions in this section have to do with initializing the wolfSSL library and shutting it down (freeing resources) after it is no longer needed by the application.

wolfSSL_Init

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_Init(void);

Description:
Initializes the wolfSSL library for use. Must be called once per application and before any other call to the library.

Return Values:
If successful the call will return **SSL_SUCCESS**.

BAD_MUTEX_E is an error that may be returned.

WC_INIT_E wolfCrypt initialization error returned.

Parameters:

This function has no parameters.

Example:

```
int ret = 0;
ret = wolfSSL_Init();
if (ret != SSL_SUCCESS) {
      /*failed to initialize wolfSSL library*/
}
```

See Also:
wolfSSL_Cleanup

wolfSSL_library_init

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_library_init(void)

Description:
This function is called internally in wolfSSL_CTX_new().

This function is a wrapper around wolfSSL_Init() and exists for OpenSSL compatibility (SSL_library_init) when wolfSSL has been compiled with OpenSSL compatibility layer. wolfSSL_Init() is the more typically-used wolfSSL initialization function.

Return Values:
If successful the call will return **SSL_SUCCESS**.

SSL_FATAL_ERROR is returned upon failure.

Parameters:

This function takes no parameters.

Example:

```
int ret = 0;
ret = wolfSSL_library_init();
if (ret != SSL_SUCCESS) {
      /*failed to initialize wolfSSL*/
}
...
```

See Also:
wolfSSL_Init
wolfSSL_Cleanup

wolfSSL_Cleanup

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_Cleanup(void);

Description:
Un-initializes the wolfSSL library from further use. Doesn’t have to be called, though it will free any resources used by the library.

Return Values:

SSL_SUCCESS return no errors.

BAD_MUTEX_E a mutex error return.

Parameters:

There are no parameters for this function.

Example:

```
wolfSSL_Cleanup();
```

See Also:
wolfSSL_Init

wolfSSL_shutdown

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_shutdown(WOLFSSL* ssl);

Description:

This function shuts down an active SSL/TLS connection using the SSL session, **ssl**. This function will try to send a “close notify” alert to the peer.

The calling application can choose to wait for the peer to send its “close notify” alert in response or just go ahead and shut down the underlying connection after directly calling wolfSSL_shutdown (to save resources). Either option is allowed by the TLS specification. If the underlying connection will be used again in the future, the complete two-directional shutdown procedure must be performed to keep synchronization intact between the peers.

wolfSSL_shutdown() works with both blocking and non-blocking I/O. When the underlying I/O is non-blocking, wolfSSL_shutdown() will return an error if the underlying I/O could not satisfy the needs of wolfSSL_shutdown() to continue. In this case, a call to wolfSSL_get_error() will yield either **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**. The calling process must then repeat the call to wolfSSL_shutdown() when the underlying I/O is ready.

Return Values:

SSL_SUCCESS - will be returned upon success.

SSL_SHUTDOWN_NOT_DONE - will be returned when shutdown has not finished, and the function should be called again.

SSL_FATAL_ERROR - will be returned upon failure. Call wolfSSL_get_error() for a more specific error code.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

Example:

```
int ret = 0;
WOLFSSL* ssl = 0;
...
```

```
ret = wolfSSL_shutdown(ssl);
if (ret != 0) {
      /*failed to shut down SSL connection*/
}
```

See Also:

wolfSSL_free
wolfSSL_CTX_free

wolfSSL_get_shutdown

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_get_shutdown(WOLFSSL* ssl);

Description:

This function checks the shutdown conditions in closeNotify or connReset or sentNotify members of the Options structure. The Options structure is within the WOLFSSL structure.

Return Values:

1 - SSL_SENT_SHUTDOWN is returned.

2 - SSL_RECEIVED_SHUTDOWN is returned.

Parameters:

ssl - a constant pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = WOLFSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = WOLFSSL_new(ctx);
…
int ret;
ret = wolfSSL_get_shutdown(ssl);

if(ret == 1){
      /*SSL_SENT_SHUTDOWN */
} else if(ret == 2){
      /*SSL_RECEIVED_SHUTDOWN */
} else {
      /*Fatal error.*/
}
```

See Also:

wolfSSL_SESSION_free

wolfSSL_is_init_finished

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_is_init_finished(WOLFSSL* ssl);

Description:

This function checks to see if the connection is established.

Return Values:

0 - returned if the connection is not established, i.e. the WOLFSSL struct is NULL or the handshake is not done.

1 - returned if the handshake is done.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
...
if(wolfSSL_is_init_finished(ssl)){
      /*Handshake is done and connection is established*/
}
```

See Also:

wolfSSL_set_accept_state
wolfSSL_get_keys

wolfSSL_set_shutdown

wolfSSL_ALPN_GetPeerProtocol

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_ALPN_GetPeerProtocol(WOLFSSL* ssl, char** list, word16* listSz);

Description:
This function copies the alpn_client_list data from the SSL object to the buffer.

Return Values:
SSL_SUCCESS - returned if the function executed without error. The alpn_client_list member of the SSL object has been copied to the **list** parameter.

BAD_FUNC_ARG - returned if the **list** or **listSz** parameter is NULL.

BUFFER_ERROR - returned if there will be a problem with the **list** buffer (either it's NULL or the size is 0).

MEMORY_ERROR - returned if there was a problem dynamically allocating memory.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

list - a pointer to the buffer. The data from the SSL object will be copied into it.

listSz - the buffer size.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
…
#ifdef HAVE_ALPN
```

```
char* list = NULL;
word16 listSz = 0;
…
err = wolfSSL_ALPN_GetPeerProtocol(ssl, &list, &listSz);

if(err == SSL_SUCCESS){
      /*List of protocols names sent by client */
}
```

See Also:

wolfSSL_UseALPN

wolfSSL_SetMinVersion

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_SetMinVersion(WOLFSSL* ssl, int version);

Description:

This function sets the minimum downgrade version allowed. Applicable only when the connection allows downgrade using (wolfSSLv23_client_method or wolfSSLv23_server_method).

Return Values:

SSL_SUCCESS - returned if this function and its subroutine executes without error.

BAD_FUNC_ARG - returned if the SSL object is NULL. In the subroutine this error is thrown if there is not a good version match.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

version - an integer representation of the version to be set as the minimum: WOLFSSL_SSLV3 = 0, WOLFSSL_TLSV1 = 1, WOLFSSL_TLSV1_1 = 2 or WOLFSSL_TLSV1_2 = 3.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
```

```
WOLFSSL* ssl = wolfSSL_new(ctx);
int version = /*version id (see internal.h enum Misc)*/
…
if(version != SSL_SUCCESS){
      /*The minimum version failed to set properly */
} else {
      /*You have successfully set the min version */
}
```

See Also:

SetMinVersionHelper
wolfSSL_CTX_SetMinVersion

wolfSSL_MakeTlsMasterSecret

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_MakeTlsMasterSecret(byte* ms, word32 msLen, const byte* pms,
        word32 pmsLen, const byte* cr, const byte* sr, int tls1_2, int hash_type);

Description:

This function copies the values of **cr** and **sr** then passes through to PRF (pseudo random function) and returns that value.

Return Values:

This function returns **0** on success.

BUFFER_E - returned if there will be an error with the size of the buffer.

MEMORY_E - returned if a subroutine failed to allocate dynamic memory.

Parameters:

ms - the master secret held in the Arrays structure.

msLen - the length of the master secret.

pms - the pre-master secret held in the Arrays structure.

pmsLen - the length of the pre-master secret.

cr - the client random.

sr - the server random.

tls1_2 - signifies that the version is at least tls version 1.2.

hash_type - signifies the hash type.

Example:

```
WOLFSSL* ssl;  /*Initialize*/

/*called in MakeTlsMasterSecret and retrieves the necessary information as
follows:*/

int MakeTlsMasterSecret(WOLFSSL* ssl){
      int ret;
      ret = wolfSSL_makeTlsMasterSecret(ssl->arrays->masterSecret,
SECRET_LEN,
                        ssl->arrays->preMasterSecret, ssl->arrays-
                        >preMasterSz,
                        ssl->arrays->clientRandom, ssl->arrays->serverRandom,
                        IsAtLeastTLSv1_2(ssl), ssl->specs.mac_algorithm);

…
return ret;

}
```

See Also:
PRF
doPRF
p_hash
MakeTlsMasterSecret

wolfSSL_SetServerID

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_SetServerID(WOLFSSL* ssl, const byte* id, int len, int newSession);

Description:

This function associates the client session with the server id. If the newSession flag is on, an existing session won’t be reused.

Return Values:

SSL_SUCCESS - returned if the function executed without an error.

BAD_FUNC_ARG - returned if the WOLFSSL struct or **id** parameter is NULL or if **len** is not greater than zero.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

id - a constant byte pointer that will be copied to the **serverID** member of the WOLFSSL_SESSION structure.

len - an int type representing the length of the session **id** parameter.

newSession - an int type representing the flag to denote whether to reuse a session or not.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol*/);
WOLFSSL* ssl = WOLFSSL_new(ctx);
const byte id[MAX_SIZE];  /*or dynamically create space*/
int len = 0; /*initialize length*/
int newSession = 0; /*flag to allow*/
…
int ret = wolfSSL_SetServerID(ssl, id, len, newSession);

if(ret){
      /*The Id was successfully set*/
}
```

See Also:

GetSessionClient

wolfSSL_ALPN_GetProtocol

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_ALPN_GetProtocol(WOLFSSL* ssl, char** protocol_name, word16* size);

Description:
This function gets the protocol name set by the server.

Return Values:
SSL_SUCCESS - returned on successful execution where no errors were thrown.

SSL_FATAL_ERROR - returned if the extension was not found or if there was no protocol match with peer. There will also be an error thrown if there is more than one protocol name accepted.

SSL_ALPN_NOT_FOUND - returned signifying that no protocol match with peer was found.

BAD_FUNC_ARG - returned if there was a NULL argument passed into the function.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

protocol_name - a pointer to a char that represents the protocol name and will be held in the ALPN structure.

size - a word16 type that represents the size of the protocol_name.

Example:

```
WOLFSSL_CTX* ctx = WOLFSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = WOLFSSL_new(ctx);
...
int err;
char* protocol_name = NULL;
Word16 protocol_nameSz = 0;
err = wolfSSL_ALPN_GetProtocol(ssl, &protocol_name, &protocol_nameSz);

if(err == SSL_SUCCESS){
      /*Sent ALPN protocol*/
}
```

See Also:

TLSX_ALPN_GetRequest
TLSX_Find

17.2 Certificates and Keys

The functions in this section have to do with loading certificates and keys into wolfSSL.

wolfSSL_CTX_load_verify_buffer

Synopsis:
int wolfSSL_CTX_load_verify_buffer(WOLFSSL_CTX* ctx, const unsigned char* in,
long sz, int format);

Description:
This function loads a CA certificate buffer into the WOLFSSL Context. It behaves like the non-buffered version, only differing in its ability to be called with a buffer as input instead of a file. The buffer is provided by the **in** argument of size **sz**. **format** specifies the format type of the buffer; **SSL_FILETYPE_ASN1** or **SSL_FILETYPE_PEM**. More than one CA certificate may be loaded per buffer as long as the format is in PEM. Please see the examples for proper usage.

Return Values:
If successful the call will return **SSL_SUCCESS**.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn’t exist, can’t be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

BUFFER_E will be returned if a chain buffer is bigger than the receiving buffer.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

in - pointer to the CA certificate buffer

sz - size of the input CA certificate buffer, **in**.

format - format of the buffer certificate, either SSL_FILETYPE_ASN1 or SSL_FILETYPE_PEM.

Example:

```
int ret = 0;
int sz = 0;
WOLFSSL_CTX* ctx;
byte certBuff[...];

...

ret = wolfSSL_CTX_load_verify_buffer(ctx, certBuff, sz, SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      /*error loading CA certs from buffer*/
}

...
```

See Also:
wolfSSL_CTX_load_verify_locations
wolfSSL_CTX_use_certificate_buffer
wolfSSL_CTX_use_PrivateKey_buffer
wolfSSL_CTX_use_NTRUPrivateKey_file
wolfSSL_CTX_use_certificate_chain_buffer
wolfSSL_use_certificate_buffer
wolfSSL_use_PrivateKey_buffer
wolfSSL_use_certificate_chain_buffer

wolfSSL_CTX_load_verify_locations

Synopsis:
int wolfSSL_CTX_load_verify_locations(WOLFSSL_CTX* ctx, const char* file,
                                                                        const char* path);

Description:

This function loads PEM-formatted CA certificate files into the SSL context (WOLFSSL_CTX). These certificates will be treated as trusted root certificates and used to verify certs received from peers during the SSL handshake.

The root certificate file, provided by the **file** argument, may be a single certificate or a file containing multiple certificates. If multiple CA certs are included in the same file, wolfSSL will load them in the same order they are presented in the file. The **path** argument is a pointer to the name of a directory that contains certificates of trusted root CAs. If the value of **file** is not NULL, **path** may be specified as NULL if not needed. If **path** is specified and NO_WOLFSSL_DIR was not defined when building the library, wolfSSL will load all CA certificates located in the given directory. This function will attempt to load all files in the directory and locate any files with the PEM header "-----BEGIN CERTIFICATE-----".

Please see the examples for proper usage.

Return Values:

If successful the call will return **SSL_SUCCESS**.

SSL_FAILURE will be returned if **ctx** is NULL, or if both **file** and **path** are NULL.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn't exist, can't be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

BUFFER_E will be returned if a chain buffer is bigger than the receiving buffer.

BAD_PATH_ERROR will be returned if opendir() fails when trying to open **path**.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

file - pointer to name of the file containing PEM-formatted CA certificates

path - pointer to the name of a directory to load PEM-formatted certificates from.

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx;

...

ret = wolfSSL_CTX_load_verify_locations(ctx, “./ca-cert.pem”, 0);
if (ret != SSL_SUCCESS) {
      /*error loading CA certs*/
}

...
```

See Also:
wolfSSL_CTX_load_verify_buffer
wolfSSL_CTX_use_certificate_file
wolfSSL_CTX_use_PrivateKey_file
wolfSSL_CTX_use_NTRUPrivateKey_file
wolfSSL_CTX_use_certificate_chain_file
wolfSSL_use_certificate_file
wolfSSL_use_PrivateKey_file
wolfSSL_use_certificate_chain_file

wolfSSL_CTX_use_PrivateKey_buffer

Synopsis:
int wolfSSL_CTX_use_PrivateKey_buffer(WOLFSSL_CTX* ctx, const unsigned char*
in, long sz, int format);

Description:
This function loads a private key buffer into the SSL Context. It behaves like the non-buffered version, only differing in its ability to be called with a buffer as input instead of a file. The buffer is provided by the **in** argument of size **sz**. **format** specifies the format type of the buffer; **SSL_FILETYPE_ASN1**or **SSL_FILETYPE_PEM**. Please see the examples for proper usage.

Return Values:
If successful the call will return **SSL_SUCCESS**.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn’t exist, can’t be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

NO_PASSWORD will be returned if the key file is encrypted but no password is provided.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

in - the input buffer containing the private key to be loaded.

sz - the size of the input buffer.

format - the format of the private key located in the input buffer (**in**). Possible values are SSL_FILETYPE_ASN1 or SSL_FILETYPE_PEM.

Example:

```
int ret = 0;
int sz = 0;
WOLFSSL_CTX* ctx;
byte keyBuff[...];

...

ret = wolfSSL_CTX_use_PrivateKey_buffer(ctx, keyBuff, sz, SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      /*error loading private key from buffer*/
}

...
```

See Also:
wolfSSL_CTX_load_verify_buffer
wolfSSL_CTX_use_certificate_buffer
wolfSSL_CTX_use_NTRUPrivateKey_file

wolfSSL_CTX_use_certificate_chain_buffer
wolfSSL_use_certificate_buffer
wolfSSL_use_PrivateKey_buffer
wolfSSL_use_certificate_chain_buffer

wolfSSL_CTX_use_PrivateKey_file

Synopsis:

int wolfSSL_CTX_use_PrivateKey_file(WOLFSSL_CTX* ctx, const char* file,
int format);

Description:

This function loads a private key file into the SSL context (WOLFSSL_CTX). The file is provided by the **file** argument. The **format** argument specifies the format type of the file - **SSL_FILETYPE_ASN1**or **SSL_FILETYPE_PEM**. Please see the examples for proper usage.

Return Values:

If successful the call will return **SSL_SUCCESS**, otherwise **SSL_FAILURE** will be returned. If the function call fails, possible causes might include:

- The file is in the wrong format, or the wrong format has been given using the “format” argument
- The file doesn’t exist, can’t be read, or is corrupted
- An out of memory condition occurs
- Base16 decoding fails on the file
- The key file is encrypted but no password is provided

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx;

...

ret = wolfSSL_CTX_use_PrivateKey_file(ctx, “./server-key.pem”,
                                      SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      /*error loading key file*/
}

. . .
```

See Also:
wolfSSL_CTX_use_PrivateKey_buffer
wolfSSL_use_PrivateKey_file
wolfSSL_use_PrivateKey_buffer

wolfSSL_CTX_use_certificate_buffer

Synopsis:
int wolfSSL_CTX_use_certificate_buffer(WOLFSSL_CTX* ctx, const unsigned char* in,
long sz, int format);

Description:
This function loads a certificate buffer into the WOLFSSL Context. It behaves like the non-buffered version, only differing in its ability to be called with a buffer as input instead of a file. The buffer is provided by the **in** argument of size **sz**. **format** specifies the format type of the buffer; **SSL_FILETYPE_ASN1** or **SSL_FILETYPE_PEM**. Please see the examples for proper usage.

Return Values:
If successful the call will return **SSL_SUCCESS**.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn't exist, can't be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

in - the input buffer containing the certificate to be loaded.

sz - the size of the input buffer.

format - the format of the certificate located in the input buffer (**in**). Possible values are SSL_FILETYPE_ASN1 or SSL_FILETYPE_PEM.

Example:

```
int ret = 0;
int sz = 0;
WOLFSSL_CTX* ctx;
byte certBuff[...];

...

ret = wolfSSL_CTX_use_certificate_buffer(ctx, certBuff, sz,
SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      /*error loading certificate from buffer*/
}

...
```

See Also:
wolfSSL_CTX_load_verify_buffer
wolfSSL_CTX_use_PrivateKey_buffer
wolfSSL_CTX_use_NTRUPrivateKey_file
wolfSSL_CTX_use_certificate_chain_buffer
wolfSSL_use_certificate_buffer
wolfSSL_use_PrivateKey_buffer
wolfSSL_use_certificate_chain_buffer

wolfSSL_CTX_use_certificate_chain_buffer

Synopsis:
```
int wolfSSL_CTX_use_certificate_chain_buffer(WOLFSSL_CTX* ctx,
                                              const unsigned char* in, long sz);
```

Description:
This function loads a certificate chain buffer into the WOLFSSL Context. It behaves like the non-buffered version, only differing in its ability to be called with a buffer as input

instead of a file. The buffer is provided by the **in** argument of size **sz**. The buffer must be in **PEM** format and start with the subject's certificate, ending with the root certificate. Please see the examples for proper usage.

Return Values:

If successful the call will return **SSL_SUCCESS**.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn't exist, can't be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

BUFFER_E will be returned if a chain buffer is bigger than the receiving buffer.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

in - the input buffer containing the PEM-formatted certificate chain to be loaded.

sz - the size of the input buffer.

Example:

```
int ret = 0;
int sz = 0;
WOLFSSL_CTX* ctx;
byte certChainBuff[...];

...

ret = wolfSSL_CTX_use_certificate_chain_buffer(ctx, certChainBuff, sz);
if (ret != SSL_SUCCESS) {
      /*error loading certificate chain from buffer*/
}

...
```

See Also:

wolfSSL_CTX_load_verify_buffer

wolfSSL_CTX_use_certificate_buffer
wolfSSL_CTX_use_PrivateKey_buffer
wolfSSL_CTX_use_NTRUPrivateKey_file
wolfSSL_use_certificate_buffer
wolfSSL_use_PrivateKey_buffer
wolfSSL_use_certificate_chain_buffer

wolfSSL_CTX_use_certificate_chain_file

Synopsis:

int wolfSSL_CTX_use_certificate_chain_file(WOLFSSL_CTX* ctx, const char* file);

Description:

This function loads a chain of certificates into the SSL context (WOLFSSL_CTX). The file containing the certificate chain is provided by the **file** argument, and must contain PEM-formatted certificates. This function will process up to MAX_CHAIN_DEPTH (default = 9, defined in internal.h) certificates, plus the subject cert.

Return Values:

If successful the call will return **SSL_SUCCESS**, otherwise **SSL_FAILURE** will be returned. If the function call fails, possible causes might include:

- The file is in the wrong format, or the wrong format has been given using the “format” argument
- file doesn’t exist, can’t be read, or is corrupted
- an out of memory condition occurs

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new()

file - a pointer to the name of the file containing the chain of certificates to be loaded into the wolfSSL SSL context. Certificates must be in PEM format.

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx;
```

```
...

ret = wolfSSL_CTX_use_certificate_chain_file(ctx, “./cert-chain.pem”);
if (ret != SSL_SUCCESS) {
      /*error loading cert file*/
}

...
```

See Also:
wolfSSL_CTX_use_certificate_file
wolfSSL_CTX_use_certificate_buffer
wolfSSL_use_certificate_file
wolfSSL_use_certificate_buffer

wolfSSL_CTX_use_certificate_file

Synopsis:
int wolfSSL_CTX_use_certificate_file(WOLFSSL_CTX* ctx, const char* file, int format);

Description:
This function loads a certificate file into the SSL context (WOLFSSL_CTX). The file is provided by the **file** argument. The **format** argument specifies the format type of the file, either **SSL_FILETYPE_ASN1** or **SSL_FILETYPE_PEM**. Please see the examples for proper usage.

Return Values:
If successful the call will return **SSL_SUCCESS**, otherwise **SSL_FAILURE** will be returned. If the function call fails, possible causes might include:

- The file is in the wrong format, or the wrong format has been given using the “format” argument
- file doesn’t exist, can’t be read, or is corrupted
- an out of memory condition occurs
- Base16 decoding fails on the file

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new()

file - a pointer to the name of the file containing the certificate to be loaded into the wolfSSL SSL context.

format - format of the certificates pointed to by **file**. Possible options are SSL_FILETYPE_ASN1 or SSL_FILETYPE_PEM.

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx;

...

ret = wolfSSL_CTX_use_certificate_file(ctx, “./client-cert.pem”,
                                     SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      /*error loading cert file*/
}

...
```

See Also:
wolfSSL_CTX_use_certificate_buffer
wolfSSL_use_certificate_file
wolfSSL_use_certificate_buffer

wolfSSL_SetTmpDH

Synopsis:
#include <wolfssl/ssl.h>

```
int wolfSSL_SetTmpDH(WOLFSSL* ssl, unsigned char* p, int pSz, unsigned char* g,
                     int gSz);
```

Description:
Server Diffie-Hellman Ephemeral parameters setting. This function sets up the group parameters to be used if the server negotiates a cipher suite that uses DHE.

Return Values:
If successful the call will return **SSL_SUCCESS**.

MEMORY_ERROR will be returned if a memory error was encountered.

SIDE_ERROR will be returned if this function is called on an SSL client instead of an SSL server.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

p - Diffie-Hellman prime number parameter.

pSz - size of **p**.

g - Diffie-Hellman “generator” parameter.

gSz - size of **g**.

Example:

```
WOLFSSL* ssl;
static unsigned char p[] = {...};
static unsigned char g[] = {...};
...
wolfSSL_SetTmpDH(ssl, p, sizeof(p), g, sizeof(g));
```

See Also:
SSL_accept

wolfSSL_use_PrivateKey

Synopsis:
#include <wolfssl/ssl.h>

SSL_use_PrivateKey ->
int wolfSSL_use_PrivateKey(WOLFSSL* ssl, WOLFSSL_EVP_PKEY* pkey);

Description:
This is used to set the private key for the WOLFSSL structure.

Return Values:

SSL_SUCCESS: On successful setting argument.

SSL_FAILURE: If an NULL ssl passed in.

All error cases will be negative values.

Parameters:

ssl - WOLFSSL structure to set argument in.

pkey - private key to use.

Example:

```
WOLFSSL* ssl;

WOLFSSL_EVP_PKEY* pkey;

int ret;

// create ssl object and set up private key

ret  = wolfSSL_use_PrivateKey(ssl, pkey);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free, wolfSSL_use_PrivateKey

wolfSSL_use_PrivateKey_ASN1

Synopsis:

#include <wolfssl/ssl.h>

SSL_use_PrivateKey_ASN1 ->
int wolfSSL_use_PrivateKey_ASN1(int pri, WOLFSSL* ssl, unsigned char* der, long derSz);

Description:

This is used to set the private key for the WOLFSSL structure. A DER formatted key buffer is expected

Return Values:

SSL_SUCCESS: On successful setting parsing and setting the private key.

SSL_FAILURE: If an NULL ssl passed in.

All error cases will be negative values.

Parameters:

pri - type of private key.

ssl - WOLFSSL structure to set argument in.

der -buffer holding DER key.

derSz - size of der buffer.

Example:

```
WOLFSSL* ssl;

unsigned char* pkey;

long pkeySz;

int ret;

// create ssl object and set up private key

ret  = wolfSSL_use_PrivateKey_ASN1(1, ssl, pkey, pkeySz);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free, wolfSSL_use_PrivateKey

wolfSSL_use_RSAPrivateKey_ASN1

Synopsis:

#include <wolfssl/ssl.h>

SSL_use_RSAPrivateKey_ASN1 ->
int wolfSSL_use_RSAPrivateKey_ASN1(WOLFSSL* ssl, unsigned char* der, long derSz);

Description:

This is used to set the private key for the WOLFSSL structure. A DER formatted RSA key buffer is expected

Return Values:

SSL_SUCCESS: On successful setting parsing and setting the private key.

SSL_FAILURE: If an NULL ssl passed in.

All error cases will be negative values.

Parameters:

ssl - WOLFSSL structure to set argument in.

der -buffer holding DER key.

derSz - size of der buffer.

Example:

```
WOLFSSL* ssl;

unsigned char* pkey;

long pkeySz;

int ret;

// create ssl object and set up RSA private key

ret  = wolfSSL_use_RSAPrivateKey_ASN1(ssl, pkey, pkeySz);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free, wolfSSL_use_PrivateKey

wolfSSL_use_PrivateKey_buffer

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_use_PrivateKey_buffer(WOLFSSL* ssl, const unsigned char* in,
long sz, int format);

Description:

This function loads a private key buffer into the WOLFSSL object. It behaves like the non-buffered version, only differing in its ability to be called with a buffer as input instead of a file. The buffer is provided by the **in** argument of size **sz**. **format** specifies the format type of the buffer; **SSL_FILETYPE_ASN1** or **SSL_FILETYPE_PEM**. Please see the examples for proper usage.

Return Values:

If successful the call will return **SSL_SUCCESS**.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn't exist, can't be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

NO_PASSWORD will be returned if the key file is encrypted but no password is provided.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

in - buffer containing private key to load.

sz - size of the private key located in **buffer**.

format - format of the private key to be loaded. Possible values are SSL_FILETYPE_ASN1 or SSL_FILETYPE_PEM.

Example:

```
int buffSz;
int ret;
byte keyBuff[...];
WOLFSSL* ssl = 0;
...

ret = wolfSSL_use_PrivateKey_buffer(ssl, keyBuff, buffSz, SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      /*failed to load private key from buffer*/
}
```

See Also:
wolfSSL_use_PrivateKey
wolfSSL_CTX_load_verify_buffer
wolfSSL_CTX_use_certificate_buffer
wolfSSL_CTX_use_PrivateKey_buffer
wolfSSL_CTX_use_NTRUPrivateKey_file
wolfSSL_CTX_use_certificate_chain_buffer
wolfSSL_use_certificate_buffer
wolfSSL_use_certificate_chain_buffer

wolfSSL_use_certificate_buffer

Synopsis:
#include <wolfssl/ssl.h>

```
int wolfSSL_use_certificate_buffer(WOLFSSL* ssl, const unsigned char* in,
                                    long sz, int format);
```

Description:
This function loads a certificate buffer into the WOLFSSL object. It behaves like the non-buffered version, only differing in its ability to be called with a buffer as input instead of a file. The buffer is provided by the **in** argument of size **sz**. **format** specifies the format type of the buffer; **SSL_FILETYPE_ASN1** or **SSL_FILETYPE_PEM**. Please see the examples for proper usage.

Return Values:
If successful the call will return **SSL_SUCCESS**.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn’t exist, can’t be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

in - buffer containing certificate to load.

sz - size of the certificate located in **buffer**.

format - format of the certificate to be loaded. Possible values are SSL_FILETYPE_ASN1 or SSL_FILETYPE_PEM.

Example:

```
int buffSz;
int ret;
byte certBuff[...];
WOLFSSL* ssl = 0;
...

ret = wolfSSL_use_certificate_buffer(ssl, certBuff, buffSz,
SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      /*failed to load certificate from buffer*/
}
```

See Also:

wolfSSL_CTX_load_verify_buffer
wolfSSL_CTX_use_certificate_buffer
wolfSSL_CTX_use_PrivateKey_buffer
wolfSSL_CTX_use_NTRUPrivateKey_file
wolfSSL_CTX_use_certificate_chain_buffer
wolfSSL_use_PrivateKey_buffer
wolfSSL_use_certificate_chain_buffer

wolfSSL_use_certificate_chain_buffer

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_use_certificate_chain_buffer(WOLFSSL* ssl, const unsigned char* in,
long sz);

Description:

This function loads a certificate chain buffer into the WOLFSSL object. It behaves like the non-buffered version, only differing in its ability to be called with a buffer as input instead of a file. The buffer is provided by the **in** argument of size **sz**. The buffer must be in **PEM** format and start with the subject's certificate, ending with the root certificate. Please see the examples for proper usage.

Return Values:

If successful the call will return **SSL_SUCCESS**.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn't exist, can't be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

BUFFER_E will be returned if a chain buffer is bigger than the receiving buffer.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

in - buffer containing certificate to load.

sz - size of the certificate located in **buffer**.

Example:

```
int buffSz;
int ret;
byte certChainBuff[...];
WOLFSSL* ssl = 0;
...

ret = wolfSSL_use_certificate_chain_buffer(ssl, certChainBuff, buffSz);
if (ret != SSL_SUCCESS) {
      /*failed to load certificate chain from buffer*/
}
```

See Also:

wolfSSL_CTX_load_verify_buffer

wolfSSL_CTX_use_certificate_buffer
wolfSSL_CTX_use_PrivateKey_buffer
wolfSSL_CTX_use_NTRUPrivateKey_file
wolfSSL_CTX_use_certificate_chain_buffer
wolfSSL_use_certificate_buffer
wolfSSL_use_PrivateKey_buffer

wolfSSL_CTX_der_load_verify_locations

Synopsis:
#include <wolfssl/ssl.h>

```
int wolfSSL_CTX_der_load_verify_locations(WOLFSSL_CTX* ctx, const char* file,
                                          int format);
```

Description:
This function is similar to wolfSSL_CTX_load_verify_locations, but allows the loading of DER-formatted CA files into the SSL context (WOLFSSL_CTX). It may still be used to load PEM-formatted CA files as well. These certificates will be treated as trusted root certificates and used to verify certs received from peers during the SSL handshake.

The root certificate file, provided by the **file** argument, may be a single certificate or a file containing multiple certificates. If multiple CA certs are included in the same file, wolfSSL will load them in the same order they are presented in the file. The **format** argument specifies the format which the certificates are in either, SSL_FILETYPE_PEM or SSL_FILETYPE_ASN1 (DER). Unlike wolfSSL_CTX_load_verify_locations, this function does not allow the loading of CA certificates from a given directory path.

Note that this function is only available when the wolfSSL library was compiled with WOLFSSL_DER_LOAD defined.

Return Values:
If successful the call will return **SSL_SUCCESS**, otherwise **SSL_FAILURE** will be returned upon failure.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new()

file - a pointer to the name of the file containing the CA certificates to be loaded into the wolfSSL SSL context, with format as specified by **format**.

format - the encoding type of the certificates specified by **file**. Possible values include SSL_FILETYPE_PEM and SSL_FILETYPE_ASN1.

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx;

...

ret = wolfSSL_CTX_der_load_verify_locations(ctx, "./ca-cert.der",
                                           SSL_FILETYPE_ASN1);
if (ret != SSL_SUCCESS) {
      /*error loading CA certs*/
}

...
```

See Also:
wolfSSL_CTX_load_verify_locations
wolfSSL_CTX_load_verify_buffer

wolfSSL_CTX_use_NTRUPrivateKey_file

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_use_NTRUPrivateKey_file(WOLFSSL_CTX* ctx, const char* file);

Description:
This function loads an NTRU private key file into the WOLFSSL Context. It behaves like the normal version, only differing in its ability to accept an NTRU raw key file. This function is needed since the format of the file is different than the normal key file (buffer) functions. Please see the examples for proper usage.

Return Values:
If successful the call will return **SSL_SUCCESS**.

SSL_BAD_FILE will be returned if the file doesn’t exist, can’t be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

BUFFER_E will be returned if a chain buffer is bigger than the receiving buffer.

NO_PASSWORD will be returned if the key file is encrypted but no password is provided.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new()

file - a pointer to the name of the file containing the NTRU private key to be loaded into the wolfSSL SSL context.

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx;

...

ret = wolfSSL_CTX_use_NTRUPrivateKey_file(ctx, “./ntru-key.raw”);
if (ret != SSL_SUCCESS) {
      /*error loading NTRU private key*/
}

...
```

See Also:
wolfSSL_CTX_load_verify_buffer
wolfSSL_CTX_use_certificate_buffer
wolfSSL_CTX_use_PrivateKey_buffer
wolfSSL_CTX_use_certificate_chain_buffer
wolfSSL_use_certificate_buffer
wolfSSL_use_PrivateKey_buffer
wolfSSL_use_certificate_chain_buffer

wolfSSL_KeepArrays

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_KeepArrays(WOLFSSL* ssl);

Description:
Normally, at the end of the SSL handshake, wolfSSL frees temporary arrays. Calling this function before the handshake begins will prevent wolfSSL from freeing temporary arrays. Temporary arrays may be needed for things such as wolfSSL_get_keys() or PSK hints.

When the user is done with temporary arrays, either **wolfSSL_FreeArrays()** may be called to free the resources immediately, or alternatively the resources will be freed when the associated SSL object is freed.

Return Values:
This function has no return value.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL* ssl;
...
wolfSSL_KeepArrays(ssl);
```

See Also:
wolfSSL_FreeArrays

wolfSSL_FreeArrays

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_FreeArrays(WOLFSSL* ssl);

Description:
Normally, at the end of the SSL handshake, wolfSSL frees temporary arrays. If wolfSSL_KeepArrays() has been called before the handshake, wolfSSL will not free temporary arrays. This function explicitly frees temporary arrays and should be called when the user is done with temporary arrays and does not want to wait for the SSL object to be freed to free these resources.

Return Values:
This function has no return value.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL* ssl;
...
wolfSSL_FreeArrays(ssl);
```

See Also:
wolfSSL_KeepArrays

wolfSSL_UnloadCertsKeys

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_UnloadCertsKeys(WOLFSSL* ssl);

Description:
This function unloads any certificates or keys that SSL owns.

Return Values:
SSL_SUCCESS - returned if the function executed successfully.

BAD_FUNC_ARG - returned if the WOLFSSL object is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
…
int unloadKeys = wolfSSL_UnloadCertsKeys(ssl);
if(unloadKeys != SSL_SUCCESS){
      /*Failure case. */
}
```

See Also:
wolfSSL_CTX_UnloadCAs

wolfSSL_CTX_get_cert_cache_memsize

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_get_cert_cache_memsize(WOLFSSL_CTX* ctx);

Description:
Returns the size the certificate cache save buffer needs to be.

Return Values:
If the funciton is successful an **INTEGER** value is returned representing the memory size.

BAD_FUNC_ARG is returned if the WOLFSSL_CTX struct is NULL.

BAD_MUTEX_E - returned if there was a mutex lock error.

Parameters:

ctx - a pointer to a wolfSSL_CTX structure, created using wolfSSL_CTX_new().

Example:

```
WOLFSSL_CTX* ctx = WOLFSSL_CTX_new(/*protocol*/);
…
int certCacheSize = wolfSSL_CTX_get_cert_cache_memsize(ctx);

if(certCacheSize != BAD_FUNC_ARG || certCacheSize != BAD_MUTEX_E){
      /*Successfully retrieved the memory size. */
}
```

See Also:

CM_GetCertCacheMemSize

wolfSSL_X509_get_signature_type

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_X509_get_signature_type(WOLFSSL_X509* x509);

Description:

This function returns the value stored in the sigOID member of the WOLFSSL_X509 structure.

Return Values:

0 - returned if the WOLFSSL_X509 structure is NULL.

An **Integer** value is returned which was retrieved from the x509 object.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_X509 x509 = (WOLFSSL_X509*)XMALLOC(sizeof(WOLFSSL_X509), NULL,
                                          DYNAMIC_TYPE_X509);

…
int x509SigType = wolfSSL_X509_get_signature_type(x509);

if(x509SigType != EXPECTED){
      /*Deal with an unexpected value*/
}
```

See Also:
wolfSSL_X509_get_signature
wolfSSL_X509_version
wolfSSL_X509_get_der
wolfSSL_X509_get_serial_number
wolfSSL_X509_notBefore
wolfSSL_X509_notAfter
wolfSSL_X509_free

wolfSSL_X509_get_next_altname

Synopsis:
#include <wolfssl/ssl.h>

char* wolfSSL_X509_get_next_altname(WOLFSSL_X509* cert);

Description:
This function returns the next, if any, altname from the peer certificate.

Return Values:
NULL if there is not a next altname.

cert->altNamesNext->name from the WOLFSSL_X509 structure that is a string value from the altName list is returned if it exists.

Parameters:

cert - a pointer to the wolfSSL_X509 structure.

Example:

```
WOLFSSL_X509 x509 = (WOLFSSL_X509*)XMALLOC(sizeof(WOLFSSL_X509), NULL,
                                          DYNAMIC_TYPE_X509);

…
int x509NextAltName = wolfSSL_X509_get_next_altname(x509);
```

```
if(x509NextAltName == NULL){
      /*There isn’t another alt name*/
}
```

See Also:
wolfSSL_X509_get_issuer_name
wolfSSL_X509_get_subject_name

wolfSSL_X509_get_subjectCN

Synopsis:
#include <wolfssl/ssl.h>

char* wolfSSL_X509_get_subjectCN(WOLFSSL_X509* x509);

Description:
Returns the common name of the subject from the certificate.

Return Values:
NULL - returned if the x509 structure is null

A **string** representation of the subject’s common name is returned if the function executes successfully.

Parameters:

x509 - a pointer to a WOLFSSL_X509 structure containing certificate information.

Example:

```
WOLFSSL_X509 x509 = (WOLFSSL_X509*)XMALLOC(sizeof(WOLFSSL_X509), NULL,
                                           DYNAMIC_TYPE_X509);

…
int x509Cn = wolfSSL_X509_get_subjectCN(x509);

if(x509Cn == NULL){
      /*Deal with NULL case*/
} else {
      /*x509Cn contains the common name*/
}
```

See Also:
wolfSSL_X509_Name_get_entry
wolfSSL_X509_get_next_altname
wolfSSL_X509_get_issuer_name
wolfSSL_X509_get_subject_name

wolfSSL_X509_get_der

Synopsis:
#include <wolfssl/ssl.h>

const byte* wolfSSL_X509_get_der(WOLFSSL_X509* x509, int* outSz);

Description:
This function gets the DER encoded certificate in the WOLFSSL_X509 struct.

Return Values:
This function returns the DerBuffer structure’s **buffer** member, which is of type byte.

NULL - returned if the **x509** or **outSz** parameter is NULL.

Parameters:

x509 - a pointer to a WOLFSSL_X509 structure containing certificate information.

outSz - length of the derBuffer member of the WOLFSSL_X509 struct.

Example:

```
WOLFSSL_X509 x509 = (WOLFSSL_X509*)XMALLOC(sizeof(WOLFSSL_X509), NULL,
                                          DYNAMIC_TYPE_X509);
int* outSz; /*initialize*/
…
byte* x509Der = wolfSSL_X509_get_der(x509, outSz);

if(x509Der == NULL){
      /*Failure case one of the parameters was NULL */
}
```

See Also:
wolfSSL_X509_version
wolfSSL_X509_Name_get_entry
wolfSSL_X509_get_next_altname
wolfSSL_X509_get_issuer_name
wolfSSL_X509_get_subject_name

wolfSSL_X509_get_hw_type

Synopsis:
#include <wolfssl/ssl.h>

byte* wolfSSL_X509_get_hw_type(WOLFSSL_X509* x509, byte* in, int* inOutSz);

Description:
The function copies the **hwType** member of the WOLFSSL_X509 structure to the buffer.

Return Values:
The function returns a **byte type** of the data previously held in the **hwType** member of the WOLFSSL_X509 structure.

NULL - returned if **inOutSz** is NULL.

Parameters:

x509 - a pointer to a WOLFSSL_X509 structure containing certificate information.

in - pointer to type byte that represents the buffer.

inOutSz - pointer to type int that represents the size of the buffer.

Example:

```
WOLFSSL_X509* x509;  /*X509 certificate*/
byte* in;  /*initialize the buffer*/
int* inOutSz;  /*holds the size of the buffer*/
...
byte* hwType = wolfSSL_X509_get_hw_type(x509, in, inOutSz);
```

```
if(hwType == NULL){
      /*Failure case function returned NULL. */
}
```

See Also:
wolfSSL_X509_get_hw_serial_number
wolfSSL_X509_get_device_type

wolfSSL_X509_d2i_fp

Synopsis:
#include <wolfssl/ssl.h>

WOLFSSL_X509* wolfSSL_X509_d2i_fp(WOLFSSL_X509** x509, XFILE file);

Description:
If NO_STDIO_FILESYSTEM is defined this function will allocate heap memory, initialize a WOLFSSL_X509 structure and return a pointer to it.

Return Values:
WOLFSSL_X509 structure pointer is returned if the function executes successfully.

NULL - if the call to XFTELL macro returns a negative value.

Parameters:

x509 - a pointer to a WOLFSSL_X509 pointer.

file - a defined type that is a pointer to a FILE.

Example:

```
WOLFSSL_X509* x509a = (WOLFSSL_X509*)XMALLOC(sizeof(WOLFSSL_X509), NULL,
                        DYNAMIC_TYPE_X509);
WOLFSSL_X509** x509 = x509a;
XFILE file;  (mapped to struct fs_file*)
...
WOLFSSL_X509* newX509 = wolfSSL_X509_d2i_fp(x509, file);

if(newX509 == NULL){
      /*The function returned NULL */
}
```

See Also:

wolfSSL_X509_d2i
XFTELL
XREWIND
XFSEEK

wolfSSL_SetCertCbCtx

Synopsis:

#include <wolfssl/ssl.h>

void wolfSSL_SetCertCbCtx(WOLFSSL* ssl, void* ctx);

Description:

This function stores user CTX object information for verify callback.

Return Values:

This function has no return value.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

ctx - a void pointer that is set to WOLFSSL structure’s verifyCbCtx member’s value.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
(void*)ctx;
…
if(ssl != NULL){
      wolfSSL_SetCertCbCtx(ssl, ctx);
} else {
      /*Error case, the SSL is not initialized properly. */
}
```

See Also:
wolfSSL_CTX_save_cert_cache
wolfSSL_CTX_restore_cert_cache
wolfSSL_CTX_set_verify

wolfSSL_CertPemToDer

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CertPemToDer(const unsigned char* pem, int pemSz,
Unsigned char* buff, int buffSz, int type);

Description:
This function converts a PEM formatted certificate to DER format. Calls OpenSSL function PemToDer.

Return Values:
Returns the bytes written to the buffer.

Parameters:

pem - pointer PEM formatted certificate.

pemSz - size of the certificate.

buff - buffer to be copied to DER format.

buffSz - size of the buffer.

type - Certificate file type found in asn_public.h **enum CertType.**

Example:

```
const unsigned char* pem;
int pemSz;
unsigned char buff[BUFSIZE];
int buffSz = sizeof(buff)/sizeof(char);
int type;
```

```
...
if(wolfSSL_CertPemToDer(pem, pemSz, buff, buffSz, type) <= 0) {
      /*There were bytes written to buffer*/
}
```

See Also:

PemToDer (OpenSSL)

wolfSSL_X509_notAfter

Synopsis:

#include <wolfssl/ssl.h>

const byte* wolfSSL_X509_notAfter(wolfSSL_X509* x509);

Description:

This function checks to see if x509 is NULL and if it’s not, it returns the notAfter member of the x509 struct.

Return Values:

The function returns a **constant byte pointer** to the notAfter member of the x509 struct.

NULL - returned if the x509 object is NULL.

Parameters:

x509 - a pointer to the WOLFSSL_X509 struct.

Example:

```
WOLFSSL_X509* x509 = (WOLFSSL_X509)XMALOC(sizeof(WOLFSSL_X509), NULL,
                              DYNAMIC_TYPE_X509) ;
...
byte* notAfter = wolfSSL_X509_notAfter(x509);
if(notAfter == NULL){
      /*Failure case, the x509 object is null. */
}
```

See Also:

wolfssl/openssl/ssl.h
cyassl/ssl.h

wolfSSL_get_peer_certificate

Synopsis:

#include <wolfssl/ssl.h>

WOLFSSL_X509* wolfSSL_get_peer_certificate(WOLFSSL* ssl);

Description:

This function gets the peer's certificate.

Return Values:

Returns a **pointer** to the peerCert member of the WOLFSSL_X509 structure if it exists.

0 - returned if the peer certificate issuer size is not defined.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
...
WOLFSSL_X509* peerCert = wolfSSL_get_peer_certificate(ssl);

if(peerCert){
      /*You have a pointer peerCert to the peer certification*/
}
```

See Also:

wolfSSL_X509_get_issuer_name
wolfSSL_X509_get_subject_name
wolfSSL_X509_get_isCA

wolfSSL_get_peer_cert_chain

Synopsis:

#include <wolfssl/ssl.h>

STACK_OF(WOLFSSL_X509)* wolfSSL_get_peer_cert_chain(const WOLFSSL* ssl);

Description:

This function gets the peer's certificate chain.

Return Values:

Returns a **pointer** to the peer's Certificate stack.

NULL - returned if no peer certificate.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
…
         wolfSSL_connect(ssl);

STACK_OF(WOLFSSL_X509)* chain = wolfSSL_get_peer_cert_chain(ssl);

ifchain){
      /*You have a pointer to the peer certificate chain*/
}
```

See Also:

wolfSSL_X509_get_issuer_name
wolfSSL_X509_get_subject_name
wolfSSL_X509_get_isCA

wolfSSL_X509_get_isCA

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_X509_get_isCA(WOLFSSL_X509* x509);

Description:
Checks the isCa member of the WOLFSSL_X509 structure and returns the value.

Return Values:
The value in the **isCA member** of the WOLFSSL_X509 structure is returned.

0 - returned if there is not a valid x509 structure passed in.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL* ssl;
…
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
…
if(wolfSSL_X509_get_isCA(ssl)){
	/*This is the CA*/
}else {
	/*Failure case*/
}
```

See Also:
wolfSSL_X509_get_issuer_name
wolfSSL_X509_get_isCA

wolfSSL_CTX_save_cert_cache

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_save_cert_cache(WOLFSSL_CTX* ctx, const char* fname);

Description:

This function writes the cert cache from memory to file.

Return Values:

SSL_SUCCESS - if CM_SaveCertCache exits normally.

BAD_FUNC_ARG - is returned if either of the arguments are NULL.

SSL_BAD_FILE - if the cert cache save file could not be opened.

BAD_MUTEX_E - if the lock mutex failed.

MEMORY_E - the allocation of memory failed.

FWRITE_ERROR - Certificate cache file write failed.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, holding the certificate information.

fname - the cert cache buffer.

Example:

```
WOLFSSL_CTX* ctx = WOLFSSL_CTX_new(/*protocol def*/);
const char* fname;
...
if(wolfSSL_CTX_save_cert_cache(ctx, fname)){
      /*file was written. */
}
```

See Also:

CM_SaveCertCache
DoMemSaveCertCache

wolfSSL_CTX_restore_cert_cache

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_restore_cert_cache(WOLFSSL_CTX* ctx,
                                   const char* fname) ;

Description:
This function persistes certificate cache from a file.

Return Values:
SSL_SUCCESS - returned if the function, CM_RestoreCertCache, executes normally.

SSL_BAD_FILE - returned if XFOPEN returns XBADFILE. The file is corrupted.

MEMORY_E - returned if the allocated memory for the temp buffer fails.

BAD_FUNC_ARG - returned if fname or ctx have a NULL value.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, holding the certificate information.

fname - the cert cache buffer.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
const char* fname = /*path to file*/;
...
if(wolfSSL_CTX_restore_cert_cache(ctx, fname)){
      /*check to see if the execution was successful */
}
```

See Also:
CM_RestoreCertCache
XFOPEN

wolfSSL_get_chain_X509

Synopsis:
#include <wolfssl/ssl.h>

WOLFSSL_X509*
wolfSSL_get_chain_X509(WOLFSSL_X509_CHAIN* chain, int idx);

Description:
This function gets the peer's wolfSSL_X509_certificate at index (idx) from the chain of certificates.

Return Values:
The function returns a pointer to a WOLFSSL_X509 structure.

Parameters:

chain - a pointer to the WOLFSSL_X509_CHAIN used for no dynamic memory SESSION_CACHE.

idx - the index of the WOLFSSL_X509 certificate.

Example:

```
WOLFSSL_X509_CHAIN* chain = &session->chain;
int idx = /*set idx*/;
…
WOLFSSL_X509_CHAIN ptr;
prt = wolfSSL_get_chain_X509(chain, idx);

if(ptr != NULL){
/*ptr contains the cert at the index specified*/
} else {
      /*ptr is NULL*/
}
```

See Also:
InitDecodedCert
ParseCertRelative
CopyDecodedToX509

wolfSSL_wolfSSL_X509_notBefore

Synopsis:
#include <wolfssl/ssl.h>

const byte* wolfSSL_X509_notBefore(WOLFSSL_X509* x509);

Description:
The function checks to see if x509 is NULL and if it's not, it returns the notBefore member of the x509 struct.

Return Values:
This function returns a **constant byte pointer** to the x509's member notAfter.

NULL - the function returns NULL if the x509 structure is NULL.

Parameters:

x509 - a pointer to the WOLFSSL_X509 struct.

Example:

```
WOLFSSL_X509* x509 = (WOLFSSL_X509)XMALLOC(sizeof(WOLFSSL_X509), NULL,
                              DYNAMIC_TYPE_X509) ;
…
byte* notAfter = wolfSSL_X509_notAfter(x509);
if(notAfter == NULL){
      /*The x509 object was NULL */
}
```

See Also:
wolfSSL_X509_notAfter

wolfSSL_X509_get_signature

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_X509_get_signature(WOLFSSL_X509* x509,
                              unsigned char* buf, int bufSz);

Description:
Gets the X509 signature and stores it in the buffer.

Return Values:

SSL_SUCCESS - returned if the function successfully executes. The signature is loaded into the buffer.

SSL_FATAL_ERRROR - returns if the x509 struct or the bufSz member is NULL. There is also a check for the length member of the sig structure (sig is a member of x509).

Parameters:

x509 - pointer to a WOLFSSL_X509 structure..

buf - a char pointer to the buffer.

bufSz - an integer pointer to the size of the buffer.

Example:

```
WOLFSSL_X509* x509 = (WOLFSSL_X509)XMALLOC(sizeof(WOLFSSL_X509), NULL,
                                          DYNAMIC_TYPE_X509);
unsigned char* buf; /*Initialize*/
int* bufSz = sizeof(buf)/sizeof(unsigned char);
...
if(wolfSSL_X509_get_signature(x509, buf, bufSz) != SSL_SUCCESS){
      /*The function did not execute successfully. */
} else{
      /*The buffer was written to correctly. */
}
```

See Also:

wolfSSL_X509_get_serial_number
wolfSSL_X509_get_signature_type
wolfSSL_X509_get_device_type

wolfSSL_X509_get_device_type

Synopsis:

#include <wolfssl/ssl.h>

byte* wolfSSL_X509_get_device_type(WOLFSSL_X509* x509, byte* in,
int* inOutSz);

Description:

This function copies the device type from the x509 structure to the buffer.

Return Values:

Returns a **byte pointer** holding the device type from the x509 structure.

NULL - returned if the buffer size is NULL.

Parameters:

x509 - pointer to a WOLFSSL_X509 structure, created with WOLFSSL_X509_new().

in - a pointer to a byte type that will hold the device type (the buffer).

inOutSz - the minimum of either the parameter inOutSz or the deviceTypeSz member of the x509 structure.

Example:

```
WOLFSSL_X509* x509 = (WOLFSSL_X509)XMALLOC(sizeof(WOLFSSL_X509), NULL,
                            DYNAMIC_TYPE_X509);
byte* in;
int* inOutSz;
...
byte* deviceType = wolfSSL_X509_get_device_type(x509, in, inOutSz);

if(!deviceType){
      /*Failure case, NULL was returned. */
}
```

See Also:

wolfSSL_X509_get_hw_type
wolfSSL_X509_get_hw_serial_number
wolfSSL_X509_d2i

wolfSSL_CTX_memsave_cert_cache

Synopsis:

```
#include <wolfssl/ssl.h>

int  wolfSSL_CTX_memsave_cert_cache(WOLFSSL_CTX* ctx, void* mem,
                                                int sz, int* used);
```

Description:

This function persists the certificate cache to memory.

Return Values:

SSL_SUCCESS - returned on successful execution of the function. No errors were thrown.

BAD_MUTEX_E - mutex error where the WOLFSSL_CERT_MANAGER member caLock was not 0 (zero).

BAD_FUNC_ARG - returned if **ctx**, **mem**, or **used** is NULL or if **sz** is less than or equal to 0 (zero).

BUFFER_E - output buffer **mem** was too small.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

mem - a void pointer to the destination (output buffer).

sz - the size of the output buffer.

used - a pointer to size of the cert cache header.

Example:

```
WOLFSSL_CTX* ctx = WOLFSSL_CTX_new(/*protocol*/);
void* mem;  /*initialize */
int sz; /*sizeof mem*/
int* used; /*cert cache header*/
...
if(wolfSSL_CTX_memsave_cert_cache(ctx, mem, sz, used) != SSL_SUCCESS){
      /*The function returned with an error*/
}
```

See Also:

DoMemSaveCertCache
GetCertCacheMemSize
CM_MemRestoreCertCache
CM_GetCertCacheMemSize

wolfSSL_KeyPemToDer

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_KeyPemToDer(const unsigned char* pem, int pemSz,
unsigned char* buff, int buffSz, const char* pass);

Description:
Converts a key in PEM format to DER format.

Return Values:
The function returns the number of **bytes** written to the buffer on successful execution.

< 0 returned indicating an error.

Parameters:

pem - a pointer to the PEM encoded certificate.

pemSz - the size of the PEM buffer (**pem**).

buff - a pointer to the copy of the buffer member of the DerBuffer struct.

buffSz - size of the buffer space allocated in the DerBuffer struct.

pass - password passed into the function.

Example:

```
byte* loadBuf; /*Initialize */
long fileSz = 0;
byte* bufSz; /*Initialize */
static int LoadKeyFile(byte** keyBuf, word32* keyBufSz, const char* keyFile,
                              int typeKey, const char* pasword);
…
bufSz = wolfSSL_KeyPemToDer(loadBuf, (int)fileSz, saveBuf,
                              (int)fileSz, password);

if(saveBufSz > 0){
      /*Bytes were written to the buffer. */
}
```

See Also:

PemToDer
wolfssl_decrypt_buffer_key

wolfSSL_X509_load_certificate_file

Synopsis:

#include <wolfssl/ssl.h>

WOLFSSL_X509* wolfSSL_X509_load_certificate_file(const char* fname,
int format);

Description:

The function loads the x509 certificate into memory.

Return Values:

A successful execution returns **pointer** to a WOLFSSL_X509 structure.

NULL - returned if the certificate was not able to be written.

Parameters:

fname - the certificate file to be loaded.

format - the format of the certificate.

Example:

```
#define cliCert    "certs/client-cert.pem"
…
X509* x509;
…
```

```
x509 = wolfSSL_X509_load_certificate_file(cliCert, SSL_FILETYPE_PEM);
AssertNotNull(x509);
```

See Also:
InitDecodedCert
PemToDer
wolfSSL_get_certificate
AssertNotNull

wolfSSL_X509_get_issuer_name

Synopsis:
#include <wolfssl/ssl.h>

WOLFSSL_X509_NAME* wolfSSL_X509_get_issuer_name(WOLFSSL_X509* cert);

Description:
This function returns the name of the certificate issuer.

Return Values:
A **pointer** to the WOLFSSL_X509 struct’s issuer member is returned.

NULL - if the cert passed in is NULL.

Parameters:

cert - a pointer to a WOLFSSL_X509 structure.

Example:

```
WOLFSSL_X509* x509;
WOLFSSL_X509_NAME issuer;
...
issuer = wolfSSL_X509_NAME_oneline(wolfSSL_X509_get_issuer_name(x509), 0, 0);

if(!issuer){
      /*NULL was returned*/
} else {
      /*issuer hods the name of the certificate issuer. */
}
```

See Also:
wolfSSL_X509_get_subject_name
wolfSSL_X509_get_isCA
wolfSSL_get_peer_certificate
wolfSSL_X509_NAME_oneline

wolfSSL_X509_NAME_oneline

Synopsis:
#include <wolfssl/ssl.h>

char* wolfSSL_X509_NAME_oneline(WOLFSSL_X509* name, char* in, int sz);

Description:
This function copies the name of the x509 into a buffer.

Return Values:
A **char pointer** to the buffer with the WOLFSSL_X509_NAME structures name member's data is returned if the function executed normally.

Parameters:
name - a pointer to a WOLFSSL_X509 structure.

in - a buffer to hold the name copied from the WOLFSSL_X509_NAME structure.

sz - the maximum size of the buffer.

Example:

```
WOLFSSL_X509 x509; /*Initialize */
char* name;
...
name = wolfSSL_X509_NAME_oneline(wolfSSL_X509_get_issuer_name(x509), 0, 0);

if(name <= 0){
      /*There's nothing in the buffer. */
}
```

See Also:
wolfSSL_X509_get_subject_name

wolfSSL_X509_get_issuer_name
wolfSSL_X509_get_isCA
wolfSSL_get_peer_certificate
wolfSSL_X509_version

wolfSSL_X509_get_hw_serial_number

Synopsis:
#include <wolfssl/ssl.h>

byte* wolfSSL_X509_get_hw_serial_number(WOLFSSL_X509 x509, byte* in,
int* inOutSz);

Description:
This function returns the hwSerialNum member of the x509 object.

Return Values:
The function returns a **byte pointer** to the in buffer that will contain the serial number loaded from the x509 object.

Parameters:

x509 - pointer to a WOLFSSL_X509 structure containing certificate information.

in - a pointer to the buffer that will be copied to.

inOutSz - a pointer to the size of the buffer.

Example:

```
char* serial;
byte* in;  /*Initialize*/
int* inOutSz; /*Initialize to max size of buffer*/
WOLFSSL_X509 x509;
...
serial = wolfSSL_X509_get_hw_serial_number(x509, in, inOutSz);

if(serial == NULL || serial <= 0){
      /*Failure case */
}
```

See Also:

wolfSSL_X509_get_subject_name
wolfSSL_X509_get_issuer_name
wolfSSL_X509_get_isCA
wolfSSL_get_peer_certificate
wolfSSL_X509_version

wolfSSL_X509_get_subject_name

Synopsis:
#include <wolfssl/ssl.h>

WOLFSSL_X509_NAME* wolfSSL_X509_get_subject_name(WOLFSSL_X509* cert);

Description:
This function returns the **subject** member of the WOLFSSL_X509 structure.

Return Values:
A **pointer** to the WOLFSSL_X509_NAME structure. The pointer may be **NULL** if the WOLFSSL_X509 struct is NULL or if the **subject** member of the structure is NULL.

Parameters:

cert - a pointer to a WOLFSSL_X509 structure.

Example:

```
WOLFSSL_X509* cert;  /* Will be initialized */
WOLFSSL_X509_NAME name;
…
name = wolfSSL_X509_get_subject_name(cert);
if(name == NULL){
      /*Deal with the NULL case */
}
```

See Also:
wolfSSL_X509_get_issuer_name
wolfSSL_X509_get_isCA
wolfSSL_get_peer_certificate

wolfSSL_X509_version

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_X509_version(WOLFSSL_X509* x509);

Description:
This function retrieves the version of the X509 certificate.

Return Values:
0 - returned if the x509 structure is NULL.

The **version** stored in the x509 structure will be returned.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_X509* x509;  /*Initialize */
int version;
...
version = wolfSSL_X509_version(x509);
if(!version){
      /*The function returned 0, failure case. */
}
```

See Also:
wolfSSL_X509_get_subject_name
wolfSSL_X509_get_issuer_name
wolfSSL_X509_get_isCA
wolfSSL_get_peer_certificate

wolfSSL_DeriveTlsKeys

Synopsis:
#include <wolfssl/ssl.h>

```
int wolfSSL_DeriveTlsKeys(byte* key_data, word32 keyLen, const byte* ms,
                          word32  msLen, const byte* sr, const byte* cr,
                          int tls1_2, int hash_type);
```

Description:
An external facing wrapper to derive TLS Keys.

Return Values:
0 - returned on success.

BUFFER_E - returned if the sum of **labLen** and **seedLen** (computes total size) exceeds the maximum size.

MEMORY_E - returned if the allocation of memory failed.

Parameters:

key_data - a byte pointer that is allocateded in DeriveTlsKeys and passed through to PRF to hold the final hash.

keyLen - a word32 type that is derived in DeriveTlsKeys from the WOLFSSL structure's specs member.

ms - a constant pointer type holding the master secret held in the **arrays** structure within the WOLFSSL structure.

msLen - a word32 type that holds the length of the master secret in an enumerated define, SECRET_LEN.

sr - a constant byte pointer to the serverRandom member of the **arrays** structure within the WOLFSSL structure.

cr - a constant byte pointer to the clientRandom member of the **arrays** structure within the WOLFSSL structure.

tls1_2 - an integer type returned from IsAtLeastTLSv1_2().

hash_type - an integer type held in the WOLFSSL structure.

Example:

```
int DeriveTlsKeys(WOLFSSL* ssl){
int ret;
…
ret = wolfSSL_DeriveTlsKeys(key_data, length, ssl->arrays->masterSecret,
                  SECRET_LEN, ssl->arrays->clientRandom,
                  IsAtLeastTLSv1_2(ssl), ssl->specs.mac_algorithm);
…
}
```

See Also:
PRF
doPRF
DeriveTlsKeys
IsAtLeastTLSv1_2

wolfSSL_get_psk_identity

Synopsis:
#include <wolfssl/ssl.h>

const char* wolfSSL_get_psk_identity(const WOLFSSL* ssl);

Description:
The function returns a constant pointer to the client_identity member of the Arrays structure.

Return Values:
The **string** value of the client_identity member of the Arrays structure.

NULL - if the WOLFSSL structure is NULL or if the Arrays member of the WOLFSSL structure is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
const char* pskID;
```

```
...
pskID = wolfSSL_get_psk_identity(ssl);

if(pskID == NULL){
      /*There is not a value in pskID*/
}
```

See Also:
wolfSSL_get_psk_identity_hint
wolfSSL_use_psk_identity_hint

wolfSSL_SetMinEccKey_Sz

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_SetMinEccKey_Sz(WOLFSSL* ssl, short keySz);

Description:
Sets the value of the minEccKeySz member of the **options** structure. The **options** struct is a member of the WOLFSSL structure and is accessed through the **ssl** parameter.

Return Values:
SSL_SUCCESS - if the function successfully set the minEccKeySz member of the **options** structure.

BAD_FUNC_ARG - if the WOLFSSL_CTX structure is NULL or if the key size (keySz) is less than 0 (zero) or not divisible by 8.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

keySz - value used to set the minimum ECC key size. Sets value in the **options** structure.

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx); /*New session */
short keySz = /*min key size allowable*/;
```

```
...
if(wolfSSL_SetMinEccKey_Sz(ssl, keySz) != SSL_SUCCESS){
      /*Failure case.  */

}
```

See Also:
wolfSSL_CTX_SetMinEccKey_Sz
wolfSSL_CTX_SetMinRsaKey_Sz
wolfSSL_SetMinRsaKey_Sz

wolfSSL_UseClientQSHKeys

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_UseClientQSHKeys(WOLFSSL* ssl, unsigned char flag);

Description:
If the flag is **1** keys will be sent in hello. If flag is **0** then the keys will not be sent during hello.

Return Values:
0 - on success.

BAD_FUNC_ARG - if the WOLFSSL structure is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().
flag - an unsigned char input to determine if the keys will be sent during hello.

Example:

```
WOLFSSL* ssl;
unsigned char flag = 1;  /*send keys*/
...
if(!wolfSSL_UseClientQSHKeys(ssl, flag)){
      /*The keys will be sent during hello. */
}
```

See Also:
wolfSSL_UseALPN
wolfSSL_UseSupportedQSH
wolfSSL_isQSH

wolfSSL_CTX_SetMinDhKey_Sz

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_SetMinDhKey_Sz(WOLFSSL_CTX* ctx, word16 keySz);

Description:
This function sets the minimum size of the Diffie Hellman key size by accessing the minDhKeySz member in the WOLFSSL_CTX structure.

Return Values:
SSL_SUCCESS - returned if the function completes successfully.

BAD_FUNC_ARG - returned if the WOLFSSL_CTX struct is NULL or if the keySz is greater than 16,000 or not divisible by 8.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

keySz - a word16 type used to set the minimum DH key size. The WOLFSSL_CTX struct holds this information in the minDhKeySz member.

Example:

```
public static int CTX_SetMinDhKey_Sz(IntPtr ctx, short minDhKey){
…
return wolfSSL_CTX_SetMinDhKey_Sz(local_ctx, minDhKey);
```

See Also:
wolfSSL_SetMinDhKey_Sz
CTX_SetMinDhKey_Sz
wolfSSL_GetDhKey_Sz

wolfSSL_CTX_SetTMpDH_file

wolfSSL_CTX_SetTmpDH_buffer

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_SetTmpDH_buffer(WOLFSSL_CTX* ctx, const unsigned char* buf,
long sz, int format);

Description:
A wrapper function that calls wolfSSL_SetTmpDH_buffer_wrapper

Return Values:
0 - returned for a successful execution.

BAD_FUNC_ARG - returned if the **ctx** or **buf** parameters are NULL.

MEMORY_E - if there is a memory allocation error.

SSL_BAD_FILETYPE - returned if **format** is not correct.

Parameters:

ctx - a pointer to a WOLFSSL structure, created using wolfSSL_CTX_new().

buf - a pointer to a constant unsigned char type that is allocated as the buffer and passed through to wolfSSL_SetTmpDH_buffer_wrapper.

sz - a long integer type that is derived from the **fname** parameter in wolfSSL_SetTmpDH_file_wrapper().

format - an integer type passed through from wolfSSL_SetTmpDH_file_wrapper().

Example:

```
static int wolfSSL_SetTmpDH_file_wrapper(WOLFSSL_CTX* ctx, WOLFSSL* ssl,
                              Const char* fname, int format);
#ifdef WOLFSSL_SMALL_STACK
byte staticBuffer[1]; /*force heap usage*/
```

```
#else
byte* staticBuffer; /*Initialize */
long sz = 0;
…
if(ssl){
      ret = wolfSSL_SetTmpDH_buffer(ssl, myBuffer, sz, format);
} else {
      ret = wolfSSL_CTX_SetTmpDH_buffer(ctx, myBuffer, sz, format);
}
```

See Also:

wolfSSL_SetTmpDH_buffer_wrapper
wolfSSL_SetTMpDH_buffer
wolfSSL_SetTmpDH_file_wrapper
wolfSSL_CTX_SetTmpDH_file

wolfSSL_GetIVSize

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_GetIVSize(WOLFSSL* ssl);

Description:

Returns the iv_size member of the **specs** structure held in the WOLFSSL struct.

Return Values:

Returns the value held in **ssl->specs.iv_size**.

BAD_FUNC_ARG - returned if the WOLFSSL structure is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
int ivSize;
...
ivSize = wolfSSL_GetIVSize(ssl);
```

```
if(ivSize > 0){
      /*ivSize holds the specs.iv_size value. */
}
```

See Also:

wolfSSL_GetKeySize
wolfSSL_GetClientWriteIV
wolfSSL_GetServerWriteIV

wolfSSL_GetDhKey_Sz

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_GetDhKey_Sz(WOLFSSL* ssl);

Description:

Returns the value of dhKeySz that is a member of the **options** structure. This value represents the Diffie-Hellman key size in bytes.

Return Values:

Returns the value held in **ssl->options.dhKeySz** which is an integer value.

BAD_FUNC_ARG - returns if the WOLFSSL struct is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
int dhKeySz;
...
dhKeySz = wolfSSL_GetDhKey_Sz(ssl);

if(dhKeySz == BAD_FUNC_ARG || dhKeySz <= 0){
      /*Failure case */
} else {
      /*dhKeySz holds the size of the key. */
}
```

See Also:
wolfSSL_SetMinDhKey_sz
wolfSSL_CTX_SetMinDhKey_Sz
wolfSSL_CTX_SetTmpDH
wolfSSL_SetTmpDH
wolfSSL_CTX_SetTmpDH_file

wolfSSL_SetTmpDH_buffer

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_SetTmpDH_buffer(WOLFSSL* ssl, const unsigned char* buf,
long sz, int format);

Description:
The function calls the wolfSSL_SetTMpDH_buffer_wrapper, which is a wrapper for Diffie-Hellman parameters.

Return Values:
SSL_SUCCESS - on successful execution.

SSL_BAD_FILETYPE - if the file type is not PEM and is not ASN.1. It will also be returned if the wc_DhParamsLoad does not return normally.

SSL_NO_PEM_HEADER - returns from PemToDer if there is not a PEM header.

SSL_BAD_FILE - returned if there is a file error in PemToDer.

SSL_FATAL_ERROR - returned from PemToDer if there was a copy error.

MEMORY_E - if there was a memory allocation error.

BAD_FUNC_ARG - returned if the WOLFSSL struct is NULL or if there was otherwise a NULL argument passed to a subroutine.

DH_KEY_SIZE_E - is returned if their is a key size error in wolfSSL_SetTmpDH() or in wolfSSL_CTX_SetTmpDH().

SIDE_ERROR - returned if it is not the server side in wolfSSL_SetTmpDH.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

buf - allocated buffer passed in from wolfSSL_SetTMpDH_file_wrapper.

sz - a long int that holds the size of the file (fname within wolfSSL_SetTmpDH_file_wrapper).

format - an integer type passed through from wolfSSL_SetTmpDH_file_wrapper() that is a representation of the certificate format.

Example:

```
Static int wolfSSL_SetTmpDH_file_wrapper(WOLFSSL_CTX* ctx, WOLFSSL* ssl,
                              Const char* fname, int format);
long sz = 0;
byte* myBuffer = staticBuffer[FILE_BUFFER_SIZE];
…
if(ssl)
ret = wolfSSL_SetTmpDH_buffer(ssl, myBuffer, sz, format);
```

See Also:
wolfSSL_SetTmpDH_buffer_wrapper
wc_DhParamsLoad
wolfSSL_SetTmpDH
PemToDer
wolfSSL_CTX_SetTmpDH
wolfSSL_CTX_SetTmpDH_file

wolfSSL_CTX_SetMinRsaKey_Sz

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_SetMinRsaKey_Sz(WOLFSSL_CTX* ctx, short keySz);

Description:
Sets the minimum RSA key size in both the WOLFSSL_CTX structure and the WOLFSSL_CERT_MANAGER structure.

Return Values:
SSL_SUCCESS - returned on successful execution of the function.

BAD_FUNC_ARG - returned if the ctx structure is NULL or the keySz is less than zero or not divisible by 8.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

keySz - a short integer type stored in minRsaKeySz in the **ctx** structure and the **cm** structure converted to bytes.

Example:

```
WOLFSSL_CTX* ctx = SSL_CTX_new(method);
(void)minDhKeyBits;
ourCert = myoptarg;
…
minDhKeyBits = atoi(myoptarg);
…
if(wolfSSL_CTX_SetMinRsaKey_Sz(ctx, minRsaKeyBits) != SSL_SUCCESS){
…
```

See Also:
wolfSSL_SetMinRsaKey_Sz

wolfSSL_CTX_SetTmpDH_file

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_SetTmpDH(WOLFSSL_CTX* ctx, const char* fname, int format);

Description:

The function calls wolfSSL_SetTmpDH_file_wrapper to set the server Diffie-Hellman parameters.

Return Values:

SSL_SUCCESS - returned if the wolfSSL_SetTmpDH_file_wrapper or any of its subroutines return successfully.

MEMORY_E - returned if an allocation of dynamic memory fails in a subroutine.

BAD_FUNC_ARG - returned if the **ctx** or **fname** parameters are NULL or if a subroutine is passed a NULL argument.

SSL_BAD_FILE - returned if the certificate file is unable to open or if the a set of checks on the file fail from wolfSSL_SetTmpDH_file_wrapper.

SSL_BAD_FILETYPE - returned if teh format is not PEM or ASN.1 from wolfSSL_SetTmpDH_buffer_wrapper().

DH_KEY_SIZE_E - returned from wolfSSL_SetTmpDH() if the ctx minDhKeySz member exceeds maximum size allowed for DH.

SIDE_ERROR - returned in wolfSSL_SetTmpDH() if the side is not the server end.

SSL_NO_PEM_HEADER - returned from PemToDer if there is no PEM header.

SSL_FATAL_ERROR - returned from PemToDer if there is a memory copy failure.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

fname - a constant character pointer to a certificate file.

format - an integer type passed through from wolfSSL_SetTmpDH_file_wrapper() that is a representation of the certificate format.

Example:

```
#define dhParam     “certs/dh2048.pem”
#DEFINE aSSERTiNTne(x, y)     AssertInt(x, y, !=, ==)
WOLFSSL_CTX* ctx;
…
AssertNotNull(ctx = wolfSSL_CTX_new(wolfSSLv23_client_method()))
…
AssertIntNE(SSL_SUCCESS, wolfSSL_CTX_SetTmpDH_file(NULL, dhParam,
                                          SSL_FILETYPE_PEM));
```

See Also:

wolfSSL_SetTmpDH_buffer_wrapper
wolfSSL_SetTmpDH
wolfSSL_CTX_SetTmpDH
wolfSSL_SetTmpDH_buffer
wolfSSL_CTX_SetTmpDH_buffer
wolfSSL_SetTmpDH_file_wrapper
AllocDer
PemToDer

wolfSSL_get_psk_identity_hint

Synopsis:

#include <wolfssl/ssl.h>

const char* wolfSSL_get_psk_identity_hint(const WOLFSSL* ssl);

Description:

This function returns the **psk identity hint**.

Return Values:

const char pointer - the value that was stored in the **arrays** member of the WOLFSSL structure is returned.

NULL - returned if the WOLFSSL or Arrays structures are NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
char* idHint;
...
idHint = wolfSSL_get_psk_identity_hint(ssl);
if(idHint){
      /*The hint was retrieved*/
      return idHint;
} else {
      /*Hint wasn't successfully retrieved */
}
```

See Also:
wolfSSL_get_psk_identity

wolfSSL_SetMinRsaKey_Sz

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_SetMinRsaKey_Sz(WOLFSSL* ssl, short keySz);

Description:
Sets the minimum allowable key size in bytes for RSA located in the WOLFSSL structure.

Return Values:
SSL_SUCCESS - the minimum was set successfully.

BAD_FUNC_ARG - returned if the ssl structure is NULL or if the ksySz is less than zero or not divisible by 8.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

keySz - a short integer value representing the the minimum key in bits.

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
short keySz;
…

int isSet =  wolfSSL_SetMinRsaKey_Sz(ssl, keySz);
if(isSet != SSL_SUCCESS){
      /*Failed to set. */
}
```

See Also:
wolfSSL_CTX_SetMinRsaKey_Sz

wolfSSL_SetMinDhKey_Sz

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_SetMinDhKey_Sz(WOLFSSL* ssl, word16 keySz);

Description:
Sets the minimum size for a Diffie-Hellman key in the WOLFSSL structure in bytes.

Return Values:
SSL_SUCCESS - the minimum size was successfully set.

BAD_FUNC_ARG - the WOLFSSL structure was NULL or the **keySz** parameter was greater than the allowable size or not divisible by 8.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

keySz - a word16 type representing the bit size of the minimum DH key.

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
```

```
word16 keySz;
...
if(wolfSSL_SetMinDhKey(ssl, keySz) != SSL_SUCCESS){
	/*Failed to set. */
}
```

See Also:
wolfSSL_GetDhKey_Sz

wolfSSL_CTX_set_tmp_dh

Synopsis:
#include <wolfssl/ssl.h>

long wolfSSL_CTX_set_tmp_dh(WOLFSSL_CTX* ctx, WOLFSSL_DH* dh);

Description:
Initializes the WOLFSSL_CTX structure's **dh** member with the Diffie-Hellman parameters.

Return Values:
SSL_SUCCESS - returned if the function executed successfully.

BAD_FUNC_ARG - returned if the ctx or dh structures are NULL.

SSL_FATAL_ERROR - returned if there was an error setting a structure value.

MEMORY_E - returned if their was a failure to allocate memory.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

dh - a pointer to a WOLFSSL_DH structure.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL_DH* dh;
```

```
…
return wolfSSL_CTX_set_tmp_dh(ctx, dh);
```

See Also:

wolfSSL_BN_bn2bin

wolfSSL_CTX_use_psk_identity_hint

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_use_psk_identity_hint(WOLFSSL_CTX* ctx, const char* hint);

Description:

This function stores the hint argument in the **server_hint** member of the WOLFSSL_CTX structure.

Return Values:

SSL_SUCCESS - returned for successful execution of the function.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

hint - a constant char pointer that will be copied to the WOLFSSL_CTX structure.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
const char* hint;
int ret;
…
ret = wolfSSL_CTX_use_psk_identity_hint(ctx, hint);
if(ret == SSL_SUCCESS){
      /* Function was successful. */
      return ret;
} else {
      /*Failure case. */
}
```

See Also:
wolfSSL_use_psk_identity_hint

wolfSSL_use_psk_identity_hint

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_use_psk_identity_hint(WOLFSSL* ssl, const char* hint);

Description:
This function stores the hint argument in the **server_hint** member of the Arrays structure within the WOLFSSL structure.

Return Values:
SSL_SUCCESS - returned if the hint was successfully stored in the WOLFSSL structure.

SSL_FAILURE - returned if the WOLFSSL or Arrays structures are NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

hint - a constant character pointer that holds the hint to be saved in memory.

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
const char* hint; /*pass in valid hint*/
...
if(wolfSSL_use_psk_identity_hint(ssl, hint) != SSL_SUCCESS){
      /*Handle failure case. */
}
```

See Also:
wolfSSL_CTX_use_psk_identity_hint

wolfSSL_make_eap_keys

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_make_eap_keys(WOLFSSL* ssl, void* msk, unsigned int len,
                                                                                    const char* label);

Description:
This function is used by EAP_TLS and EAP-TTLS to derive keying material from the master secret.

Return Values:
BUFFER_E - returned if the actual size of the buffer exceeds the maximum size allowable.

MEMORY_E - returned if there is an error with memory allocation.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

msk - a void pointer variable that will hold the result of the p_hash function.

len - an unsigned integer that represents the length of the **msk** variable.

label - a constant char pointer that is copied from in PRF() .

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);;
void* msk;
unsigned int len;
const char* label;
…
return wolfSSL_make_eap_keys(ssl, msk, len, label);
```

See Also:
PRF
doPRF
p_hash
wc_HmacFinal
wc_HmacUpdate

wolfSSL_CTX_SetMinEccKey_Sz

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_SetMinEccKey_Sz(WOLFSSL_CTX* ctx, short keySz);

Description:

Sets the minimum size in bytes for the ECC key in the WOLF_CTX structure and the WOLFSSL_CERT_MANAGER structure.

Return Values:

SSL_SUCCESS - returned for a successful execution and the minEccKeySz member is set.

BAD_FUNC_ARG - returned if the WOLFSSL_CTX struct is NULL or if the **keySz** is negative or not divisible by 8.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

keySz - a short integer type that represents the minimum ECC key size in bits.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
short keySz; /*minimum key size*/
…
if(wolfSSL_CTX_SetMinEccKey(ctx, keySz) != SSL_SUCCESS){
      /*Failed to set min key size */
}
```

See Also:

wolfSSL_SetMinEccKey_Sz

wolfSSL_SetTmpDH_file

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_SetTmpDH_file(WOLFSSL* ssl, const char* fname, int format);

Description:
This function calls wolfSSL_SetTmpDH_file_wrapper to set server Diffie-Hellman parameters.

Return Values:
SSL_SUCCESS - returned on successful completion of this function and its subroutines.

MEMORY_E - returned if a memory allocation failed in this function or a subroutine.

SIDE_ERROR - if the **side** member of the **Options** structure found in the WOLFSSL struct is not the server side.

SSL_BAD_FILETYPE - returns if the certificate fails a set of checks.

BAD_FUNC_ARG - returns if an argument value is NULL that is not permitted such as, the WOLFSSL structure.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

fname - a constant char pointer holding the certificate.

format - an integer type that holds the format of the certification.

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
const char* dhParam;
…
AssertIntNE(SSL_SUCCESS, wolfSSL_SetTmpDH_file(ssl, dhParam,
SSL_FILETYPE_PEM));
```

See Also:
wolfSSL_CTX_SetTmpDH_file
wolfSSL_SetTmpDH_file_wrapper
wolfSSL_SetTmpDH_buffer
wolfSSL_CTX_SetTmpDH_buffer
wolfSSL_SetTmpDH_buffer_wrapper
wolfSSL_SetTmpDH
wolfSSL_CTX_SetTmpDH

wolfSSL_PubKeyPemToDer

Synopsis:
```
#include <wolfssl/ssl.h>

int wolfSSL_PubKeyPemToDer(const unsigned char* pem, int pemSz,
                           unsigned char* buff, int buffSz);
```

Description:
Converts the PEM format to DER format.

Return Values:
An **int** type representing the bytes written to buffer.

< 0 - returned for an error.

BAD_FUNC_ARG - returned if the DER length is incorrect or if the pem buff, or buffSz arguments are NULL.

Parameters:

pem - the PEM certificate.

pemSz - the size of the PEM certificate.

buff - the buffer that will be written to from the DerBuffer.

buffSz - the size of the buffer.

Example:

```
unsigned char* pem = "/*pem file*/";
int pemSz = sizeof(pem)/sizeof(char);
unsigned char* buff; /*The buffer*/
int buffSz; /*Initialize*/
...
if(wolfSSL_PubKeyPemToDer(pem, pemSz, buff, buffSz)!= SSL_SUCCESS){
      /*Conversion was not successful */
}
```

See Also:
wolfSSL_PubKeyPemToDer
wolfSSL_PemPubKeyToDer
PemToDer

wolfSSL_CTX_SetTmpDH

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_SetTmpDH(WOLFSSL_CTX* ctx, const unsigned char* p, int pSz,
Const unsigned char* g, int gSz);

Description:
Sets the parameters for the server CTX Diffie-Hellman.

Return Values:
SSL_SUCCESS - returned if the function and all subroutines return without error.

BAD_FUNC_ARG - returned if the CTX, p or g parameters are NULL.

DH_KEY_SIZE_E - returned if the minDhKeySz member of the WOLFSSL_CTX struct is not the correct size.

MEMORY_E - returned if the allocation of memory failed in this function or a subroutine.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

p - a constant unsigned char pointer loaded into the **buffer** member of the serverDH_P struct.

pSz - an int type representing the size of p, initialized to MAX_DH_SIZE.

g - a constant unsigned char pointer loaded into the **buffer** member of the serverDH_G struct.

gSz - an int type representing the size of g, initialized ot MAX_DH_SIZE.

Example:

```
WOLFSSL_CTX* ctx =  WOLFSSL_CTX_new(/*protocol def*/);
byte* p; /*Initialize / Allocate size*/
byte* g; /*Initialize / Allocate size*/
word32 pSz = (word32)sizeof(p)/sizeof(byte);
word32 gSz = (word32)sizeof(g)/sizeof(byte);
…
int ret =  wolfSSL_CTX_SetTmpDH(ctx, p, pSz, g, gSz);

if(ret != SSL_SUCCESS){
      /*Failure case*/
}
```

See Also:

wolfSSL_SetTmpDH
wc_DhParamsLoad

wolfSSL_d2i_X509_bio

Synopsis:

#include <wolfssl/ssl.h>

d2i_X509_bio ->

WOLFSSL_X509* wolfSSL_d2i_X509_bio(WOLFSSL_BIO* bio, WOLFSSL_X509** x509);

Description:

This function get the DER buffer from bio and converts it to a WOLFSSL_X509 structure.

Return Values:

Returns NULL on failure and a new WOLFSSL_X509 structure pointer on success.

Parameters:

bio - pointer to the WOLFSSL_BIO structure that has the DER certificate buffer.

x509 - pointer that get set to new WOLFSSL_X509 structure created.

Example:

```
WOLFSSL_BIO* bio;
WOLFSSL_X509* x509;

// load DER into bio

x509 = wolfSSL_d2i_X509_bio(bio, NULL);
Or
wolfSSL_d2i_X509_bio(bio, &x509);
// use x509 returned (check for NULL)
```

See Also:

wolfSSL_PEM_read_bio_DSAparams

Synopsis:

#include <wolfssl/ssl.h>

PEM_read_bio_DSAparams ->
WOLFSSL_DSA* wolfSSL_PEM_read_bio_DSAparams(WOLFSSL_BIO* bio,
WOLFSSL_DSA** x, pem_password_cb* cb, void* u);

Description:

This function get the DSA parameters from a PEM buffer in bio.

Return Values:

On successfully parsing the PEM buffer a WOLFSSL_DSA structure is created and returned. If failing to parse the PEM buffer NULL is returned.

Parameters:

bio - pointer to the WOLFSSL_BIO structure for getting PEM memory pointer.

x - pointer to be set to new WOLFSSL_DSA structure.

cb - password callback function.

u - null terminated password string.

Example:

```
WOLFSSL_BIO* bio;
WOLFSSL_DSA* dsa;

// setup bio

dsa = wolfSSL_PEM_read_bio_DSAparams(bio, NULL, NULL, NULL);

// check dsa is not NULL and then use dsa
```

See Also:

wolfSSL_PEM_read_bio_X509_AUX

Synopsis:

#include <wolfssl/ssl.h>

PEM_read_bio_X509_AUX->
WOLFSSL_X509* wolfSSL_PEM_read_bio_X509_AUX(WOLFSSL_BIO* bp, WOLFSSL_X509** x, pem_password_cb* cb, void* u);

Description:

This function behaves the same as wolfSSL_PEM_read_bio_X509. AUX signifies containing extra information such as trusted/rejected use cases and friendly name for human readability.

Return Values:

On successfully parsing the PEM buffer a WOLFSSL_X509 structure is returned. If unsuccessful NULL is returned.

Parameters:

bp - WOLFSSL_BIO structure to get PEM buffer from.

x - if setting WOLFSSL_X509 by function side effect.

cb - password callback.

u - NULL terminated user password.

Example:

```
WOLFSSL_BIO* bio;
WOLFSSL_X509* x509;
// setup bio

X509 = wolfSSL_PEM_read_bio_X509_AUX(bio, NULL, NULL, NULL);

//check x509 is not null and then use it
```

See Also:

wolfSSL_PEM_read_bio_X509

wolfSSL_PEM_write_bio_PrivateKey

Synopsis:

#include <wolfssl/ssl.h>

PEM_write_bio_PrivateKey ->
int wolfSSL_PEM_write_bio_PrivateKey(WOLFSSL_BIO* bio, WOLFSSL_EVP_PKEY* key, const WOLFSSL_EVP_CIPHER* cipher, unsigned char* passwd, int len, pem_password_cb* cb, void* arg);

Description:

This function writes a key into a WOLFSSL_BIO structure in PEM format.

Return Values:

On successfully creating the PEM buffer SSL_SUCCESS is returned. If unsuccessful SSL_FAILURE is returned.

Parameters:

bio - WOLFSSL_BIO structure to get PEM buffer from.

key - key to convert to PEM format.

cipher- EVP cipher structure.

passwd - password.

len - length of password.

cb - password callback.

arg - optional argument.

Example:

```
WOLFSSL_BIO* bio;
WOLFSSL_EVP_PKEY* key;
int ret;
// create bio and setup key
ret = wolfSSL_PEM_write_bio_PrivateKey(bio, key, NULL, NULL, 0, NULL, NULL);

//check ret value
```

See Also:

wolfSSL_PEM_read_bio_X509_AUX

wolfSSL_X509_digest

Synopsis:

#include <wolfssl/ssl.h>

X509_digest ->

int wolfSSL_X509_digest(const WOLFSSL_X509* x509, const WOLFSSL_EVP_MD* digest, unsigned char* buf, unsigned int* len)

Description:

This function returns the hash of the DER certificate.

Return Values:

SSL_SUCCESS: On successfully creating a hash.

SSL_FAILURE: Returned on bad input or unsuccessful hash.

Parameters:

x509 - certificate to get the hash of.

digest - the hash algorithm to use.

buf - buffer to hold hash.

len - length of buffer.

Example:

```
WOLFSSL_X509* x509;
unsigned char buffer[64];
unsigned int bufferSz;
int ret;

ret = wolfSSL_X509_digest(x509, wolfSSL_EVP_sha256(), buffer, &bufferSz);
//check ret value
```

See Also:

wolfSSL_X509_get_ext_d2i

Synopsis:

#include <wolfssl/ssl.h>

X509_get_ext_d2i ->
void*wolfSSL_X509_get_ext_d2i(const WOLFSSL_X509* x509, int nid, int* c, int* idx)

Description:

This function looks for and returns the extension matching the passed in NID value.

Return Values:

NULL: If extension is not found or error is encountered.

If successful a STACK_OF(WOLFSSL_ASN1_OBJECT) pointer is returned.

Parameters:

x509 - certificate to get parse through for extension.

nid - extension OID to be found.

c - if not NULL is set to -2 for multiple extensions found -1 if not found, 0 if found and not critical and 1 if found and critical.

idx - if NULL return first extension matched otherwise if not stored in x509 start at idx.

Example:

```
const WOLFSSL_X509* x509;
int c;
int idx = 0;
STACK_OF(WOLFSSL_ASN1_OBJECT)* sk;

sk = wolfSSL_X509_get_ext_d2i(x509, NID_basic_constraints, &c, &idx);

//check sk for NULL and then use it. sk needs freed after done.
```

See Also:

wolfSSL_sk_ASN1_OBJECT_free

wolfSSL_X509_NAME_get_text_by_NID

Synopsis:

#include <wolfssl/ssl.h>

X509_NAME_get_text_by_NID ->
int wolfSSL_X509_NAME_get_text_by_NID(WOLFSSL_X509_NAME* name, int nid, char* buf, int len);

Description:

This function gets the text related to the passed in NID value.

Return Values:

Returns the size of text buffer.

Parameters:

name - WOLFSSL_X509_NAME to search for text.

nid - NID to search for.

buf - buffer to hold text when found.

len - length of buffer.

Example:

```
WOLFSSL_X509_NAME* name;
char buffer[100];
int bufferSz;
int ret;
// get WOLFSSL_X509_NAME
ret = wolfSSL_X509_NAME_get_text_by_NID(name, NID_commonName, buffer,
bufferSz);

//check ret value
```

See Also:

wolfSSL_X509_STORE_add_cert

Synopsis:

#include <wolfssl/ssl.h>

X509_STORE_add_cert ->

int wolfSSL_X509_STORE_add_cert(WOLFSSL_X509_STORE* str, WOLFSSL_X509* x509);

Description:

This function adds a certificate to the WOLFSSL_X509_STRE structure.

Return Values:

SSL_SUCCESS: If certificate is added successfully.

SSL_FATAL_ERROR: If certificate is not added successfully.

Parameters:

str - certificate store to add the certificate to.

x509 - certificate to add.

Example:

```
WOLFSSL_X509_STORE* str;
WOLFSSL_X509* x509;
int ret;

ret = wolfSSL_X509_STORE_add_cert(str, x509);

//check ret value
```

See Also:

wolfSSL_X509_free

wolfSSL_X509_STORE_CTX_get_chain

Synopsis:

#include <wolfssl/ssl.h>

X509_STORE_CTX_get_chain ->
int wolfSSL_X509_STORE_CTX_get_chain(WOLFSSL_X509_STORE_CTX* ctx);

Description:

This function is a getter function for chain variable in WOLFSSL_X509_STORE_CTX structure. Currently chain is not populated.

Return Values:

If successful returns WOLFSSL_STACK (same as STACK_OF(WOLFSSL_X509)) pointer otherwise NULL.

Parameters:

ctx - certificate store ctx to get parse chain from.

Example:

```
WOLFSSL_STACK* sk;
WOLFSSL_X509_STORE_CTX* ctx;

sk = wolfSSL_X509_STORE_CTX_get_chain(ctx);

//check sk for NULL and then use it. sk needs freed after done.
```

See Also:

wolfSSL_sk_X509_free

wolfSSL_X509_STORE_set_flags

Synopsis:

#include <wolfssl/ssl.h>

X509_STORE_set_flags ->
int wolfSSL_X509_STORE_set_flags(WOLFSSL_X509_STORE* str, unsigned long flag);

Description:

This function takes in a flag to change the behavior of the WOLFSSL_X509_STORE structure passed in. An example of a flag used is WOLFSSL_CRL_CHECK.

Return Values:

SSL_SUCCESS: If no errors were encountered when setting the flag.

If unsuccessful a negative error value is returned.

Parameters:

str - certificate store to set flag in.

flag - flag for behavior.

Example:

```
WOLFSSL_X509_STORE* str;
int ret;
// create and set up str

ret = wolfSSL_X509_STORE_set_flags(str, WOLFSSL_CRL_CHECKALL);
If (ret != SSL_SUCCESS) {
      //check ret value and handle error case
}
```

See Also:

wolfSSL_X509_STORE_new, wolfSSL_X509_STORE_free

wolfSSL_DES_set_key

Synopsis:

#include <wolfssl/openssl/des.h>

DES_set_key ->

int wolfSSL_DES_set_key(WOLFSSL_const_DES_cblock* myDes, WOLFSSL_DES_key_schedule* key);

Description:

This function sets the key schedule. If the macro WOLFSSL_CHECK_DESKEY is defined then acts like wolfSSL_DES_set_key_checked if not then acts like wolfSSL_DES_set_key_unchecked.

Return Values:

If WOLFSSL_CHECK_DESKEY set then -1 if parity error, -2 for weak/null key, and 0 for success.

If macro WOLFSSL_CHECK_DESKEY is not defined then always returns 0.

Parameters:

myDes - DES key

key - key to set from myDes.

Example:

```
WOLFSSL_const_DES_cblock* myDes;
WOLFSSL_DES_key_schedule* key;
int ret;
// load DES key
ret = wolfSSL_DES_set_key(myDes, key);

// check ret value
```

See Also:

wolfSSL_DES_set_key_checked, wolfSSL_DES_set_key_unchecked

wolfSSL_DSA_dup_DH

Synopsis:

#include <wolfssl/ssl.h>

DSA_dup_DH ->

WOLFSSL_DH* wolfSSL_DSA_dup_DH(const WOLFSSL_DSA* dsa);

Description:

This function duplicates the parameters in dsa to a newly created WOLFSSL_DH structure.

Return Values:

If duplicated returns WOLFSSL_DH structure if function failed NULL is returned.

Parameters:

dsa - WOLFSSL_DSA structure to duplicate.

Example:

```
WOLFSSL_DH* dh;

WOLFSSL_DSA* dsa;

// set up dsa

dh = wolfSSL_DSA_dup_DH(dsa);


// check dh is not null
```

See Also:

17.3 Context and Session Setup

The functions in this section have to do with creating and setting up SSL/TLS context objects (WOLFSSL_CTX) and SSL/TLS session objects (WOLFSSL).

wolfSSLv3_client_method

Synopsis:

#include <wolfssl/ssl.h>

WOLFSSL_METHOD *wolfSSLv3_client_method(void);

Description:

The wolfSSLv3_client_method() function is used to indicate that the application is a client and will only support the SSL 3.0 protocol. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

Return Values:

If successful, the call will return a pointer to the newly created WOLFSSL_METHOD structure.

If memory allocation fails when calling XMALLOC, the failure value of the underlying malloc() implementation will be returned (typically NULL with errno will be set to ENOMEM).

Parameters:

This function has no parameters.

Example:

```
WOLFSSL_METHOD* method;
WOLFSSL_CTX* ctx;

method = wolfSSLv3_client_method();
if (method == NULL) {
      /*unable to get method*/
}

ctx = wolfSSL_CTX_new(method);
...
```

See Also:

wolfTLSv1_client_method
wolfTLSv1_1_client_method
wolfTLSv1_2_client_method
wolfDTLSv1_client_method
wolfSSLv23_client_method
wolfSSL_CTX_new

wolfSSLv3_server_method

Synopsis:

#include <wolfssl/ssl.h>

WOLFSSL_METHOD *wolfSSLv3_server_method(void);

Description:

The wolfSSLv3_server_method() function is used to indicate that the application is a server and will only support the SSL 3.0 protocol. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

Return Values:

If successful, the call will return a pointer to the newly created WOLFSSL_METHOD structure.

If memory allocation fails when calling XMALLOC, the failure value of the underlying malloc() implementation will be returned (typically NULL with errno will be set to ENOMEM).

Parameters:

This function has no parameters.

Example:

```
WOLFSSL_METHOD* method;
WOLFSSL_CTX* ctx;

method = wolfSSLv3_server_method();
if (method == NULL) {
      /*unable to get method*/
}

ctx = wolfSSL_CTX_new(method);
...
```

See Also:

wolfTLSv1_server_method

wolfTLSv1_1_server_method
wolfTLSv1_2_server_method
wolfDTLSv1_server_method
wolfSSLv23_server_method
wolfSSL_CTX_new

wolfSSLv23_client_method

Synopsis:
#include <wolfssl/ssl.h>

WOLFSSL_METHOD *wolfSSLv23_client_method(void);

Description:
The wolfSSLv23_client_method() function is used to indicate that the application is a client and will support the highest protocol version supported by the server between SSL 3.0 - TLS 1.2. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

Both wolfSSL clients and servers have robust version downgrade capability. If a specific protocol version method is used on either side, then only that version will be negotiated or an error will be returned. For example, a client that uses TLSv1 and tries to connect to a SSLv3 only server will fail, likewise connecting to a TLSv1.1 will fail as well.

To resolve this issue, a client that uses the wolfSSLv23_client_method() function will use the highest protocol version supported by the server and downgrade to SSLv3 if needed. In this case, the client will be able to connect to a server running SSLv3 - TLSv1.2.

Return Values:
If successful, the call will return a pointer to the newly created WOLFSSL_METHOD structure.

If memory allocation fails when calling XMALLOC, the failure value of the underlying malloc() implementation will be returned (typically NULL with errno will be set to ENOMEM).

Parameters:

This function has no parameters.

Example:

```
WOLFSSL_METHOD* method;
WOLFSSL_CTX* ctx;

method = wolfSSLv23_client_method();
if (method == NULL) {
      /*unable to get method*/
}

ctx = wolfSSL_CTX_new(method);
...
```

See Also:
wolfSSLv3_client_method
wolfTLSv1_client_method
wolfTLSv1_1_client_method
wolfTLSv1_2_client_method
wolfDTLSv1_client_method
wolfSSL_CTX_new

wolfSSLv23_server_method

Synopsis:
#include <wolfssl/ssl.h>

WOLFSSL_METHOD *wolfSSLv23_server_method(void);

Description:
The wolfSSLv23_server_method() function is used to indicate that the application is a server and will support clients connecting with protocol version from SSL 3.0 - TLS 1.2. This function allocates memory for and initializes a new WOLFSSL_METHOD structure

to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

Return Values:
If successful, the call will return a pointer to the newly created WOLFSSL_METHOD structure.

If memory allocation fails when calling XMALLOC, the failure value of the underlying malloc() implementation will be returned (typically NULL with errno will be set to ENOMEM).

Parameters:

This function has no parameters.

Example:

```
WOLFSSL_METHOD* method;
WOLFSSL_CTX* ctx;

method = wolfSSLv23_server_method();
if (method == NULL) {
      /*unable to get method*/
}

ctx = wolfSSL_CTX_new(method);
...
```

See Also:
wolfSSLv3_server_method
wolfTLSv1_server_method
wolfTLSv1_1_server_method
wolfTLSv1_2_server_method
wolfDTLSv1_server_method
wolfSSL_CTX_new

wolfTLSv1_client_method

Synopsis:
WOLFSSL_METHOD *wolfTLSv1_client_method(void);

Description:

The wolfTLSv1_client_method() function is used to indicate that the application is a client and will only support the TLS 1.0 protocol. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

Return Values:
If successful, the call will return a pointer to the newly created WOLFSSL_METHOD structure.

If memory allocation fails when calling XMALLOC, the failure value of the underlying malloc() implementation will be returned (typically NULL with errno will be set to ENOMEM).

Example:

```
WOLFSSL_METHOD* method;
WOLFSSL_CTX* ctx;

method = wolfTLSv1_client_method();
if (method == NULL) {
	/*unable to get method*/
}

ctx = wolfSSL_CTX_new(method);
...
```

See Also:
wolfSSLv3_client_method
wolfTLSv1_1_client_method
wolfTLSv1_2_client_method
wolfDTLSv1_client_method
wolfSSLv23_client_method
wolfSSL_CTX_new

wolfTLSv1_server_method

Synopsis:
WOLFSSL_METHOD *wolfTLSv1_server_method(void);

Description:
The wolfTLSv1_server_method() function is used to indicate that the application is a

server and will only support the TLS 1.0 protocol. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

Return Values:

If successful, the call will return a pointer to the newly created WOLFSSL_METHOD structure.

If memory allocation fails when calling XMALLOC, the failure value of the underlying malloc() implementation will be returned (typically NULL with errno will be set to ENOMEM).

Example:

```
WOLFSSL_METHOD* method;
WOLFSSL_CTX* ctx;

method = wolfTLSv1_server_method();
if (method == NULL) {
      /*nable to get method*/
}

ctx = wolfSSL_CTX_new(method);
...
```

See Also:

wolfSSLv3_server_method
wolfTLSv1_1_server_method
wolfTLSv1_2_server_method
wolfDTLSv1_server_method
wolfSSLv23_server_method
wolfSSL_CTX_new

wolfTLSv1_1_client_method

Synopsis:

WOLFSSL_METHOD *wolfTLSv1_1_client_method(void);

Description:

The wolfTLSv1_1_client_method() function is used to indicate that the application is a client and will only support the TLS 1.0 protocol. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

Return Values:

If successful, the call will return a pointer to the newly created WOLFSSL_METHOD structure.

If memory allocation fails when calling XMALLOC, the failure value of the underlying malloc() implementation will be returned (typically NULL with errno will be set to ENOMEM).

Example:

```
WOLFSSL_METHOD* method;
WOLFSSL_CTX* ctx;

method = wolfTLSv1_1_client_method();
if (method == NULL) {
      /*unable to get method*/
}

ctx = wolfSSL_CTX_new(method);
...
```

See Also:

wolfSSLv3_client_method
wolfTLSv1_client_method
wolfTLSv1_2_client_method
wolfDTLSv1_client_method
wolfSSLv23_client_method
wolfSSL_CTX_new

wolfTLSv1_1_server_method

Synopsis:

WOLFSSL_METHOD *wolfTLSv1_1_server_method(void);

Description:

The wolfTLSv1_1_server_method() function is used to indicate that the application is a server and will only support the TLS 1.1 protocol. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

Return Values:
If successful, the call will return a pointer to the newly created WOLFSSL_METHOD structure.

If memory allocation fails when calling XMALLOC, the failure value of the underlying malloc() implementation will be returned (typically NULL with errno will be set to ENOMEM).

Example:

```
WOLFSSL_METHOD* method;
WOLFSSL_CTX* ctx;

method = wolfTLSv1_1_server_method();
if (method == NULL) {
      /*unable to get method*/
}

ctx = wolfSSL_CTX_new(method);
...
```

See Also:
wolfSSLv3_server_method
wolfTLSv1_server_method
wolfTLSv1_2_server_method
wolfDTLSv1_server_method
wolfSSLv23_server_method
wolfSSL_CTX_new

wolfTLSv1_2_client_method

Synopsis:
WOLFSSL_METHOD *wolfTLSv1_2_client_method(void);

Description:
The wolfTLSv1_2_client_method() function is used to indicate that the application is a client and will only support the TLS 1.2 protocol. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS

context with wolfSSL_CTX_new().

Return Values:

If successful, the call will return a pointer to the newly created WOLFSSL_METHOD structure.

If memory allocation fails when calling XMALLOC, the failure value of the underlying malloc() implementation will be returned (typically NULL with errno will be set to ENOMEM).

Example:

```
WOLFSSL_METHOD* method;
WOLFSSL_CTX* ctx;

method = wolfTLSv1_2_client_method();
if (method == NULL) {
      /*unable to get method*/
}

ctx = wolfSSL_CTX_new(method);
...
```

See Also:

wolfSSLv3_client_method
wolfTLSv1_client_method
wolfTLSv1_1_client_method
wolfDTLSv1_client_method
wolfSSLv23_client_method
wolfSSL_CTX_new

wolfTLSv1_2_server_method

Synopsis:

WOLFSSL_METHOD *wolfTLSv1_2_server_method(void);

Description:

The wolfTLSv1_2_server_method() function is used to indicate that the application is a server and will only support the TLS 1.2 protocol. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

Return Values:

If successful, the call will return a pointer to the newly created WOLFSSL_METHOD structure.

If memory allocation fails when calling XMALLOC, the failure value of the underlying malloc() implementation will be returned (typically NULL with errno will be set to ENOMEM).

Example:

```
WOLFSSL_METHOD* method;
WOLFSSL_CTX* ctx;

method = wolfTLSv1_2_server_method();
if (method == NULL) {
      /*unable to get method*/
}

ctx = wolfSSL_CTX_new(method);
...
```

See Also:

wolfSSLv3_server_method
wolfTLSv1_server_method
wolfTLSv1_1_server_method
wolfDTLSv1_server_method
wolfSSLv23_server_method
wolfSSL_CTX_new

wolfSSLv3_server_method_ex; wolfSSLv3_client_method_ex; wolfTLSv1_server_method_ex; wolfTLSv1_client_method_ex; wolfTLSv1_1_server_method_ex; wolfTLSv1_1_client_method_ex; wolfTLSv1_2_server_method_ex; wolfTLSv1_2_client_method_ex; wolfSSLv23_server_method_ex; wolfSSLv23_client_method_ex;

Synopsis:

#include <wolfssl/ssl.h>

WOLFSSL_METHOD* (*wolfSSL_method_func)(void* heap)

Description:

These functions have the same behavior as their counterparts (functions with not having _ex) except that they do not create WOLFSSL_METHOD* using dynamic memory. The functions will use the heap hint passed in to create a new WOLFSSL_METHOD struct.

Return Values:

A value of WOLFSSL_METHOD pointer if success.

NULL is returned in error cases.

Parameters:

heap - a pointer to a heap hint for creating WOLFSSL_METHOD struct.

Example:

```
WOLFSSL_CTX* ctx;
int ret;

...

ctx = NULL:
ret = wolfSSL_CTX_load_static_memory(&ctx, wolfSSLv23_server_method_ex,
memory, memorySz, 0, MAX_CONCURRENT_HANDSHAKES);
if (ret != SSL_SUCCESS) {
      // handle error case
}

...
```

See Also:

wolfSSL_new
wolfSSL_CTX_new
wolfSSL_CTX_free

wolfDTLSv1_client_method

Synopsis:
WOLFSSL_METHOD *wolfDTLSv1_client_method(void);

Description:
The wolfDTLSv1_client_method() function is used to indicate that the application is a client and will only support the DTLS 1.0 protocol. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

Return Values:
If successful, the call will return a pointer to the newly created WOLFSSL_METHOD structure.

If memory allocation fails when calling XMALLOC, the failure value of the underlying malloc() implementation will be returned (typically NULL with errno will be set to ENOMEM).

Example:

```
WOLFSSL_METHOD* method;
WOLFSSL_CTX* ctx;

method = wolfDTLSv1_client_method();
if (method == NULL) {
      /*unable to get method*/
}

ctx = wolfSSL_CTX_new(method);
...
```

See Also:
wolfSSLv3_client_method
wolfTLSv1_client_method
wolfTLSv1_1_client_method
wolfTLSv1_2_client_method
wolfSSLv23_client_method
wolfSSL_CTX_new

wolfDTLSv1_server_method

Synopsis:

#include <wolfssl/ssl.h>

WOLFSSL_METHOD *wolfDTLSv1_server_method(void);

Description:

The wolfDTLSv1_server_method() function is used to indicate that the application is a server and will only support the DTLS 1.0 protocol. This function allocates memory for and initializes a new WOLFSSL_METHOD structure to be used when creating the SSL/TLS context with wolfSSL_CTX_new().

Return Values:

If successful, the call will return a pointer to the newly created WOLFSSL_METHOD structure.

If memory allocation fails when calling XMALLOC, the failure value of the underlying malloc() implementation will be returned (typically NULL with errno will be set to ENOMEM).

Example:

```
WOLFSSL_METHOD* method;
WOLFSSL_CTX* ctx;

method = wolfDTLSv1_server_method();
if (method == NULL) {
      /*unable to get method*/
}

ctx = wolfSSL_CTX_new(method);
...
```

See Also:

wolfSSLv3_server_method
wolfTLSv1_server_method
wolfTLSv1_1_server_method
wolfTLSv1_2_server_method
wolfSSLv23_server_method
wolfSSL_CTX_new

wolfSSL_new

Synopsis:
#include <wolfssl/ssl.h>

WOLFSSL* wolfSSL_new(WOLFSSL_CTX* ctx);

Description:
This function creates a new SSL session, taking an already created SSL context as input.

Return Values:
If successful the call will return a pointer to the newly-created WOLFSSL structure. Upon failure, NULL will be returned.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

Example:

```
WOLFSSL*     ssl = NULL;
WOLFSSL_CTX* ctx = 0;

ctx = wolfSSL_CTX_new(method);
if (ctx == NULL) {
      /*context creation failed*/
}

ssl = wolfSSL_new(ctx);
if (ssl == NULL) {
      /*SSL object creation failed*/
}
```

See Also:
wolfSSL_CTX_new

wolfSSL_free

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_free(WOLFSSL* ssl);

Description:
This function frees an allocated WOLFSSL object.

Return Values:
No return values are used for this function.

Parameters:

ssl - pointer to the SSL object, created with wolfSSL_new().

Example:

```
WOLFSSL* ssl = 0;
...
wolfSSL_free(ssl);
```

See Also:
wolfSSL_CTX_new
wolfSSL_new
wolfSSL_CTX_free

wolfSSL_ASN1_INTEGER_to_BN

Synopsis:
#include <wolfssl/ssl.h>

ASN1_INTEGER_to_BN ->
WOLFSSL_BIGNUM* wolfSSL_ASN1_INTEGER_to_BN(const
WOLFSSL_ASN1_INTEGER* ai, WOLFSSL_BIGNUM* bn);

Description:
This function is used to copy a WOLFSSL_ASN1_INTEGER value to a
WOLFSSL_BIGNUM structure.

Return Values:
On successfully copying the WOLFSSL_ASN1_INTEGER value a WOLFSSL_BIGNUM pointer is returned.

If a failure occured NULL is returned.

Parameters:

ai - WOLFSSL_ASN1_INTEGER structure to copy from.

bn - if wanting to copy into an already existing WOLFSSL_BIGNUM struct then pass in a pointer to it. Optionally this can be NULL and a new WOLFSSL_BIGNUM structure will be created.

Example:

```
WOLFSSL_ASN1_INTEGER* ai;

WOLFSSL_BIGNUM* bn;

// create ai

bn = wolfSSL_ASN1_INTEGER_to_BN(ai, NULL);

// or if having already created bn and wanting to reuse structure
// wolfSSL_ASN1_INTEGER_to_BN(ai, bn);

// check bn is or return value is not NULL
```

See Also:

wolfSSL_ASN1_INTEGER_get

Synopsis:

#include <wolfssl/ssl.h>

#include <l/wolfssl/openssl/asn1.h>

ASN1_INTEGER_get ->

long wolfSSL_ASN1_INTEGER_get(const WOLFSSL_ASN1_INTEGER* i)

Description:

This functions convert ASN1_INTEGER structure to the value.

Return Values:

ASN1_INTEGER_get() returns the value of i but it returns 0 if a is NULL and -1 on error.

Parameters:

i - WOLFSSL_ASN1_INTEGER structure

Example:

```
WOLFSSL_ASN1_INTEGER* i;

long a;

// create ai

a = wolfSSL_ASN1_INTEGER_get(ai);

// check a is or return value is not NULL
```

wolfSSL_BN_mod_exp

Synopsis:

#include <wolfssl/openssl/bn.h>

BN_mod_exp ->
int wolfSSL_BN_mod_exp(WOLFSSL_BIGNUM* r, const WOLFSSL_BIGNUM* a, const WOLFSSL_BIGNUM* p, const WOLFSSL_BIGNUM* m, WOLFSSL_BN_CTX* ctx);

Description:

This function performs the following math “r = (a^p) % m”.

Return Values:

SSL_SUCCESS: On successfully performing math operation.

SSL_FAILURE: If an error case was encountered.

Parameters:

r - structure to hold result.

a - value to be raised by a power.

p - power to raise a by.

m - modulus to use.

ctx -currently not used with wolfSSL can be NULL.

Example:

```
WOLFSSL_BIGNUM r,a,p,m;

int ret;

// set big number values

ret  = wolfSSL_BN_mod_exp(r, a, p, m, NULL);
// check ret value
```

See Also:

wolfSSL_BN_new, wolfSSL_BN_free

wolfSSL_BN_mod_mul

Synopsis:

#include <wolfssl/openssl/bn.h>

BN_mod_mul ->

nt wolfSSL_BN_mod_mul(WOLFSSL_BIGNUM *r, const WOLFSSL_BIGNUM *a,
    const WOLFSSL_BIGNUM *b, const WOLFSSL_BIGNUM *m,
WOLFSSL_BN_CTX *ctx)

Description:

This function performs the following math “"r=(a*b) mod m”.

Return Values:

SSL_SUCCESS: On successfully performing math operation.

SSL_FAILURE: If an error case was encountered.

Parameters:

r - structure to hold result.

a - value to be multiplied

b - value to multiply

m - modulus to use

ctx -currently not used with wolfSSL can be NULL.

Example:

```
WOLFSSL_BIGNUM r,a,b, m;

int ret;

// set big number values

ret  = wolfSSL_BN_mod_mul(r, a, b, m, NULL);
// check ret value
```

See Also:

wolfSSL_BN_new, wolfSSL_BN_free

wolfSSL_check_private_key

Synopsis:

#include <wolfssl/ssl.h>

SSL_check_private_key ->
int wolfSSL_check_private_key(const WOLFSSL* ssl);

Description:

This function checks that the private key is a match with the certificate being used.

Return Values:

SSL_SUCCESS: On successfully match.

SSL_FAILURE: If an error case was encountered.

All error cases other than SSL_FAILURE are negative values.

Parameters:

ssl - WOLFSSL structure to check.

Example:

```
WOLFSSL* ssl;

int ret;

// create and set up ssl
ret  = wolfSSL_check_private_key(ssl);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSL_get_client_random

Synopsis:

#include <wolfssl/ssl.h>

SSL_get_client_random ->
size_t wolfSSL_get_client_random(const WOLFSSL* ssl, unsigned char* out, size_t outSz);

Description:

This is used to get the random data sent by the client during the handshake.

Return Values:

On successfully getting data returns a value greater than 0. If no random data buffer or an error state returns 0. If outSz passed in is 0 then the maximum buffer size needed is returned.

Parameters:

ssl - WOLFSSL structure to get clients random data buffer from.

out -buffer to hold random data.

outSz -size of out buffer passed in. (if 0 function will return max buffer size needed)

Example:

```
WOLFSSL ssl;

unsigned char* buffer;

size_t bufferSz;

size_t ret;

bufferSz  = wolfSSL_get_client_random(ssl, NULL, 0);
buffer = malloc(bufferSz);
ret  = wolfSSL_get_client_random(ssl, buffer, bufferSz);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSL_get_server_random

Synopsis:

#include <wolfssl/ssl.h>

SSL_get_server_random ->
size_t wolfSSL_get_server_random(const WOLFSSL* ssl, unsigned char* out, size_t outSz);

Description:

This is used to get the random data sent by the server during the handshake.

Return Values:

On successfully getting data returns a value greater than 0. If no random data buffer or an error state returns 0. If outSz passed in is 0 then the maximum buffer size needed is returned.

Parameters:

ssl - WOLFSSL structure to get clients random data buffer from.

out -buffer to hold random data.

outSz -size of out buffer passed in. (if 0 function will return max buffer size needed)

Example:

```
WOLFSSL ssl;

unsigned char* buffer;

size_t bufferSz;

size_t ret;

bufferSz  = wolfSSL_get_server_random(ssl, NULL, 0);
buffer = malloc(bufferSz);
ret  = wolfSSL_get_server_random(ssl, buffer, bufferSz);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSL_SESSION_get_master_key

Synopsis:

#include <wolfssl/ssl.h>

SSL_SESSION_get_master_key ->
int wolfSSL_SESSION_get_master_key(const WOLFSSL_SESSION* ses, unsigned char* out, int outSz);

Description:

This is used to get the master key after completing a handshake.

Return Values:

On successfully getting data returns a value greater than 0. If no data or an error state is hit then the function returns 0. If the outSz passed in is 0 then the maximum buffer size needed is returned.

Parameters:

ses - WOLFSSL_SESSION structure to get master secret buffer from.

out -buffer to hold data.

outSz -size of out buffer passed in. (if 0 function will return max buffer size needed)

Example:

```
WOLFSSL_SESSION ssl;

unsigned char* buffer;

size_t bufferSz;

size_t ret;

// complete handshake and get session structure

bufferSz  = wolfSSL_SESSION_get_master_secret(ses, NULL, 0);
buffer = malloc(bufferSz);
ret  = wolfSSL_SESSION_get_master_secret(ses, buffer, bufferSz);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSL_SESSION_get_master_key_length

Synopsis:

#include <wolfssl/ssl.h>

SSL_SESSION_get_master_key_length ->
int wolfSSL_SESSION_get_master_key_length(const WOLFSSL_SESSION* ses);

Description:

This is used to get the master secret key length.

Return Values:

Returns master secret key size.

Parameters:

ses - WOLFSSL_SESSION structure to get master secret buffer from.

Example:

```
WOLFSSL_SESSION ssl;

unsigned char* buffer;

size_t bufferSz;

size_t ret;

// complete handshake and get session structure

bufferSz  = wolfSSL_SESSION_get_master_secret_length(ses);
buffer = malloc(bufferSz);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSL_get_options

Synopsis:

#include <wolfssl/ssl.h>

SSL_get_options ->

unsigned long wolfSSL_get_options(const WOLFSSL* ssl);

Description:

This function returns the current options mask.

Return Values:

Returns the mask value stored in ssl.

Parameters:

ssl - WOLFSSL structure to get options mask from.

Example:

```
WOLFSSL* ssl;

unsigned long mask;
```

```
mask  = wolfSSL_get_options(ssl);
// check mask
```

See Also:

wolfSSL_new, wolfSSL_free, wolfSSL_set_options

wolfSSL_set_options

Synopsis:

#include <wolfssl/ssl.h>

SSL_set_options ->
unsigned long wolfSSL_get_options(const WOLFSSL* ssl, unsigned long op);

Description:

This function sets the options mask in the ssl.

Some valid options are:

SSL_OP_ALL

SSL_OP_COOKIE_EXCHANGE

SSL_OP_NO_SSLv2

SSL_OP_NO_SSLv3

SSL_OP_NO_TLSv1

SSL_OP_NO_TLSv1_1

SSL_OP_NO_TLSv1_2

SSL_OP_NO_COMPRESSION

Return Values:

Returns the updated options mask value stored in ssl.

Parameters:

ssl - WOLFSSL structure to set options mask.

Example:

```
WOLFSSL* ssl;

unsigned long mask;

mask = SSL_OP_NO_TLSv1

mask  = wolfSSL_set_options(ssl, mask);
// check mask
```

See Also:

wolfSSL_new, wolfSSL_free, wolfSSL_get_options

wolfSSL_set_msg_callback

Synopsis:

#include <wolfssl/ssl.h>

SSL_set_msg_callback ->
int wolfSSL_set_msg_callback(WOLFSSL *ssl, SSL_Msg_Cb cb);

Description:

This function sets a callback in the ssl. The callback is to observe handshake messages. NULL value of cb resets the callback.

Callback function prototype:

typedef void (*SSL_Msg_Cb)(int write_p, int version, int content_type,
    const void *buf, size_t len, WOLFSSL *ssl, void *arg);

Return Values:

SSL_SUCCESS: On success.

SSL_FAILURE: If an NULL ssl passed in.

Parameters:

ssl - WOLFSSL structure to set callback argument.

Example:

```
static cb(int write_p, int version, int content_type,
    const void *buf, size_t len, WOLFSSL *ssl, void *arg)
{ … }

WOLFSSL* ssl;
ret  = wolfSSL_set_msg_callback(ssl, cb);
// check ret
```

See Also:
wolfSSL_set_msg_callback_arg

wolfSSL_set_msg_callback_arg

Synopsis:
#include <wolfssl/ssl.h>

SSL_set_msg_callback_arg ->
void wolfSSL_set_msg_callback_arg(WOLFSSL *ssl, void *arg);

Description:
This function sets associated callback context value in the ssl. The value is handed over to the callback argument.

Return Values:
None

Parameters:
ssl - WOLFSSL structure to set callback argument.

Example:

```
static cb(int write_p, int version, int content_type,
    const void *buf, size_t len, WOLFSSL *ssl, void *arg)
{ … }

WOLFSSL* ssl;
ret  = wolfSSL_set_msg_callback(ssl, cb);
// check ret
       wolfSSL_set_msg_callback(ssl, arg);
```

See Also:

wolfSSL_set_msg_callback

wolfSSL_get_verify_result

Synopsis:

#include <wolfssl/ssl.h>

SSL_get_verify_result ->
long wolfSSL_get_verify_result(const WOLFSSL* ssl);

Description:

This is used to get the results after trying to verify the peer's certificate.

Return Values:

X509_V_OK: On successful verification.

SSL_FAILURE: If an NULL ssl passed in.

Parameters:

ssl - WOLFSSL structure to get verification results from.

Example:

```
WOLFSSL* ssl;

long ret;

// attempt/complete handshake

ret  = wolfSSL_get_verify_result(ssl);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSL_get1_session

Synopsis:

#include <wolfssl/ssl.h>

SSL_get1_session ->
WOLFSSL_SESSION* wolfSSL_get1_session(WOLFSSL* ssl)

Description:

This function returns the WOLFSSL_SESSION from the WOLFSSL structure.

Return Values:

WOLFSSL_SESSION: On success return session pointer.

NULL: on failure returns NULL.

Parameters:

ssl - WOLFSSL structure to get session from.

Example:

```
WOLFSSL* ssl;

WOLFSSL_SESSION* ses;

// attempt/complete handshake

ses  = wolfSSL_get1_session(ssl);
// check ses information
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSL_set_tlsext_debug_arg

Synopsis:

#include <wolfssl/ssl.h>

SSL_set_tlsext_debug_arg ->
long wolfSSL_set_tlsext_debug_arg(WOLFSSL* ssl, void* arg);

Description:

This is used to set the debug argument passed around.

Return Values:

SSL_SUCCESS: On successful setting argument.

SSL_FAILURE: If an NULL ssl passed in.

Parameters:

ssl - WOLFSSL structure to set argument in.

arg - argument to use.

Example:

```
WOLFSSL* ssl;

void* args;

int ret;

// create ssl object

ret  = wolfSSL_set_tlsext_debug_arg(ssl, args);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSL_set_tmp_dh

Synopsis:

#include <wolfssl/ssl.h>

SSL_set_tmp_dh ->
long wolfSSL_set_tmp_dh(WOLFSSL* ssl, WOLFSSL_DH* dh);

Description:

This function sets the temporary DH to use during the handshake.

Return Values:

SSL_SUCCESS: On successful setting DH.

SSL_FAILURE, MEMORY_E, SSL_FATAL_ERROR, BAD_FUNC_ARG: in error cases

Parameters:

ssl - WOLFSSL structure to set temporary DH.

dh - DH to use.

Example:

```
WOLFSSL* ssl;

WOLFSSL_DH* dh;

int ret;

// create ssl object

ret  = wolfSSL_set_tmp_dh(ssl, dh);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSL_state

Synopsis:

#include <wolfssl/ssl.h>

SSL_state ->
int wolfSSL_state(WOLFSSL* ssl);

Description:

This is used to get the internal error state of the WOLFSSL structure.

Return Values:

Returns ssl error state or BAD_FUNC_ARG if ssl is NULL.

Parameters:

ssl - WOLFSSL structure to get state from.

Example:

```
WOLFSSL* ssl;

int ret;

// create ssl object

ret  = wolfSSL_state(ssl);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSL_use_certificate

Synopsis:

#include <wolfssl/ssl.h>

SSL_use_certificate ->
int wolfSSL_use_certificate(WOLFSSL* ssl, WOLFSSL_X509* x509);

Description:

This is used to set the certificate for WOLFSSL structure to use during a handshake.

Return Values:

SSL_SUCCESS: On successful setting argument.

SSL_FAILURE: If a NULL argument passed in.

Parameters:

ssl - WOLFSSL structure to set certificate in.

x509 - certificate to use.

Example:

```
WOLFSSL* ssl;

WOLFSSL_X509* x509

int ret;

// create ssl object and x509

ret  = wolfSSL_use_certificate(ssl, x509);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSL_use_certificate_ASN1

Synopsis:

#include <wolfssl/ssl.h>

SSL_use_certificate_ASN1 ->

int wolfSSL_use_certificate_ASN1(WOLFSSL* ssl, unsigned char* der, int derSz);

Description:

This is used to set the certificate for WOLFSSL structure to use during a handshake. A DER formatted buffer is expected.

Return Values:

SSL_SUCCESS: On successful setting argument.

SSL_FAILURE: If a NULL argument passed in.

Parameters:

ssl - WOLFSSL structure to set certificate in.

der - DER certificate to use.

derSz - size of the DER buffer passed in.

Example:

```
WOLFSSL* ssl;

unsigned char* der;

int derSz;

int ret;

// create ssl object and set DER variables

ret  = wolfSSL_use_certificate_ASN1(ssl, der, derSz);
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSLv23_method

Synopsis:

#include <wolfssl/ssl.h>

SSLv23_method ->
WOLFSSL_METHOD* wolfSSLv23_method(void);

Description:

This function returns a WOLFSSL_METHOD similar to wolfSSLv23_client_method except that it is not determined which side yet (server/client).

Return Values:

WOLFSSL_METHOD*: On successful creation returns a WOLFSSL_METHOD pointer.

NULL: NULL if memory allocation error or failure to create method.

Parameters:

None

Example:

```
WOLFSSL* ctx;

ctx  = wolfSSL_CTX_new(wolfSSLv23_method());
// check ret value
```

See Also:

wolfSSL_new, wolfSSL_free

wolfSSL_CTX_new

Synopsis:

WOLFSSL_CTX* wolfSSL_CTX_new(WOLFSSL_METHOD* method);

Description:

This function creates a new SSL context, taking a desired SSL/TLS protocol method for input.

Return Values:

If successful the call will return a pointer to the newly-created WOLFSSL_CTX. Upon failure, NULL will be returned.

Parameters:

method - pointer to the desired WOLFSSL_METHOD to use for the SSL context. This is created using one of the wolfSSLvXX_XXXX_method() functions to specify SSL/TLS/DTLS protocol level.

Example:

```
WOLFSSL_CTX*    ctx    = 0;
WOLFSSL_METHOD* method = 0;
```

```
method = wolfSSLv3_client_method();
if (method == NULL) {
      /*unable to get method*/
}

ctx = wolfSSL_CTX_new(method);
if (ctx == NULL) {
      /*context creation failed*/
}
```

See Also:
wolfSSL_new

wolfSSL_CTX_free

Synopsis:
void wolfSSL_CTX_free(WOLFSSL_CTX* ctx);

Description:
This function frees an allocated WOLFSSL_CTX object. This function decrements the CTX reference count and only frees the context when the reference count has reached 0.

Return Values:
No return values are used for this function.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

Example:

```
WOLFSSL_CTX* ctx = 0;
...
wolfSSL_CTX_free(ctx);
```

See Also:
wolfSSL_CTX_new
wolfSSL_new
wolfSSL_free

wolfSSL_CTX_clear_options

Synopsis:

long wolfSSL_CTX_clear_options(WOLFSSL_CTX* ctx, long opt);

Description:

This function resets option bits of WOLFSSL_CTX object.

Return Values:

New option bits

Parameters:

ctx - pointer to the SSL context.

Example:

```
WOLFSSL_CTX* ctx = 0;
...
wolfSSL_CTX_clear_options(ctx, SSL_OP_NO_TLSv1);
```

See Also:

wolfSSL_CTX_new
wolfSSL_new
wolfSSL_free

wolfSSL_CTX_add_extra_chain_cert

Synopsis:

#include <wolfssl/ssl.h>

SSL_CTX_add_extra_chain_cert ->

long wolfSSL_CTX_add_extra_chain_cert(WOLFSSL_CTX* ctx, WOLFSSL_X509* x509);

Description:

This function adds the certificate to the internal chain being built in the WOLFSSL_CTX structure.

Return Values:

SSL_SUCCESS: after successfully adding the certificate.

SSL_FAILURE: if failing to add the certificate to the chain.

Parameters:

ctx - WOLFSSL_CTX structure to add certificate to.

x509 - certificate to add to the chain.

Example:

```
WOLFSSL_CTX* ctx;

WOLFSSL_X509* x509;

int ret;

// create ctx

ret = wolfSSL_CTX_add_extra_chain_cert(ctx, x509);
// check ret value
```

See Also:

wolfSSL_CTX_new, wolfSSL_CTX_free

wolfSSL_CTX_get_cert_store

Synopsis:

#include <wolfssl/ssl.h>

SSL_CTX_get_cert_store ->
WOLFSSL_X509_STORE* wolfSSL_CTX_get_cert_store(WOLFSSL_CTX* ctx);

Description:

This is a getter function for the WOLFSSL_X509_STORE structure in ctx.

Return Values:

WOLFSSL_X509_STORE*: On successfully getting the pointer.

NULL: Returned if NULL arguments are passed in.

Parameters:

ctx - pointer to the WOLFSSL_CTX structure for getting cert store pointer.

Example:

```
WOLFSSL_CTX ctx;

WOLFSSL_X509_STORE* st;

// setup ctx

st = wolfSSL_CTX_get_cert_store(ctx);
//use st
```

See Also:

wolfSSL_CTX_new, wolfSSL_CTX_free, wolfSSL_CTX_set_cert_store

wolfSSL_CTX_set_cert_store

Synopsis:

#include <wolfssl/ssl.h>

SSL_CTX_set_cert_store ->
void wolfSSL_CTX_set_cert_store(WOLFSSL_CTX* ctx, WOLFSSL_X509_STORE* str);

Description:

This is a setter function for the WOLFSSL_X509_STORE structure in ctx.

Return Values:

None

Parameters:

ctx - pointer to the WOLFSSL_CTX structure for setting cert store pointer.

str - pointer to the WOLFSSL_X509_STORE to set in ctx.

Example:

```
WOLFSSL_CTX ctx;

WOLFSSL_X509_STORE* st;

// setup ctx and st

st = wolfSSL_CTX_set_cert_store(ctx, st);
//use st
```

See Also:

wolfSSL_CTX_new, wolfSSL_CTX_free, wolfSSL_CTX_get_cert_store

wolfSSL_CTX_get_default_passwd_cb

Synopsis:

#include <wolfssl/ssl.h>

SSL_CTX_get_default_passwd_cb ->
int wolfSSL_CTX_get_default_passwd_cb(WOLFSSL_CTX* ctx)

Description:

This is a getter function for the password callback set in ctx.

Return Values:

On success returns the callback function.

NULL: If ctx is NULL then NULL is returned.

Parameters:

ctx - WOLFSSL_CTX structure to get call back from.

Example:

```
WOLFSSL_CTX* ctx;
Pem_password_cb cb;
// setup ctx
cb = wolfSSL_CTX_get_default_passwd_cb(ctx);

//use cb
```

See Also:

wolfSSL_CTX_new, wolfSSL_CTX_free

wolfSSL_CTX_get_default_passwd_cb_userdata

Synopsis:

#include <wolfssl/ssl.h>

SSL_CTX_get_default_passwd_cb_userdata ->
void* wolfSSL_CTX_get_default_passwd_cb_userdata(WOLFSSL_CTX* ctx)

Description:

This is a getter function for the password callback user data set in ctx.

Return Values:

On success returns the user data pointer.

NULL: If ctx is NULL then NULL is returned.

Parameters:

ctx - WOLFSSL_CTX structure to get user data from.

Example:

```
WOLFSSL_CTX* ctx;
void* data;
// setup ctx
data = wolfSSL_CTX_get_default_passwd_cb(ctx);
```

```
//use data
```

See Also:

wolfSSL_CTX_new, wolfSSL_CTX_free

wolfSSL_CTX_get_read_ahead

Synopsis:

#include <wolfssl/ssl.h>

SSL_CTX_get_read_ahead ->
int wolfSSL_CTX_get_read_ahead(WOLFSSL_CTX* ctx);

Description:

This function returns the get read ahead flag from a WOLFSSL_CTX structure;

Return Values:

On success returns the read ahead flag.

SSL_FAILURE: If ctx is NULL then SSL_FAILURE is returned.

Parameters:

ctx - WOLFSSL_CTX structure to get read ahead flag from.

Example:

```
WOLFSSL_CTX* ctx;
int flag;
// setup ctx
flag = wolfSSL_CTX_get_read_ahead(ctx);

//check flag
```

See Also:

wolfSSL_CTX_new, wolfSSL_CTX_free, wolfSSL_CTX_set_read_ahead

wolfSSL_CTX_set_read_ahead

Synopsis:

#include <wolfssl/ssl.h>

SSL_CTX_set_read_ahead ->
int wolfSSL_CTX_set_read_ahead(WOLFSSL_CTX* ctx, int v);

Description:

This function sets the read ahead flag in the WOLFSSL_CTX structure;

Return Values:

SSL_SUCCESS: If ctx read ahead flag set.

SSL_FAILURE: If ctx is NULL then SSL_FAILURE is returned.

Parameters:

ctx - WOLFSSL_CTX structure to set read ahead flag.

Example:

```
WOLFSSL_CTX* ctx;
int flag;
int ret;
// setup ctx
ret = wolfSSL_CTX_set_read_ahead(ctx, flag);
// check return value
```

See Also:

wolfSSL_CTX_new, wolfSSL_CTX_free, wolfSSL_CTX_get_read_ahead

wolfSSL_CTX_set_tlsext_status_arg

Synopsis:

#include <wolfssl/ssl.h>

SSL_CTX_set_tlsext_status_arg ->

long wolfSSL_CTX_set_tlsext_status_arg(WOLFSSL_CTX* ctx, void* arg);

Description:

This function sets the options argument to use with OCSP.

Return Values:

SSL_FAILURE: If ctx or it's cert manager is NULL.

SSL_SUCCESS: If successfully set.

Parameters:

ctx - WOLFSSL_CTX structure to set user argument.

arg - user argument.

Example:

```
WOLFSSL_CTX* ctx;
void* data;
int ret;
// setup ctx
ret = wolfSSL_CTX_set_tlsext_status_arg(ctx, data);

//check ret value
```

See Also:

wolfSSL_CTX_new, wolfSSL_CTX_free

wolfSSL_CTX_set_tlsext_opaque_prf_input_callback_arg

Synopsis:

#include <wolfssl/ssl.h>

SSL_CTX_set_tlsext_opaque_prf_input_callback_arg ->
long wolfSSL_CTX_set_tlsext_opaque_prf_input_callback_arg(WOLFSSL_CTX* ctx, void* arg);

Description:

This function sets the optional argument to be passed to the PRF callback.

Return Values:

SSL_FAILURE: If ctx is NULL.

SSL_SUCCESS: If successfully set.

Parameters:

ctx - WOLFSSL_CTX structure to set user argument.

arg - user argument.

Example:

```
WOLFSSL_CTX* ctx;
void* data;
int ret;
// setup ctx
ret = wolfSSL_CTX_set_tlsext_opaques_prf_input_callback_arg(ctx, data);

//check ret value
```

See Also:

wolfSSL_CTX_new, wolfSSL_CTX_free

wolfSSL_SetVersion

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_SetVersion(WOLFSSL* ssl, int version);

Description:

This function sets the SSL/TLS protocol version for the specified SSL session (WOLFSSL object) using the version as specified by **version**.

This will override the protocol setting for the SSL session (**ssl**) - originally defined and set by the SSL context (wolfSSL_CTX_new()) method type.

Return Values:
If successful the call will return **SSL_SUCCESS**.

BAD_FUNC_ARG will be returned if the input SSL object is NULL or an incorrect protocol version is given for **version**.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

version - SSL/TLS protocol version. Possible values include WOLFSSL_SSLV3, WOLFSSL_TLSV1, WOLFSSL_TLSV1_1, WOLFSSL_TLSV1_2.

Example:

```
int ret = 0;
WOLFSSL* ssl;
...

ret = wolfSSL_SetVersion(ssl, WOLFSSL_TLSV1);
if (ret != SSL_SUCCESS) {
      /*failed to set SSL session protocol version*/
}
```

See Also:
wolfSSL_CTX_new

wolfSSL_use_old_poly

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_use_old_poly(WOLFSSL* ssl, int value);

Description:
Since there is some differences between the first release and newer versions of chacha-poly AEAD construction we have added an option to communicate with servers/clients using the older version. By default wolfSSL uses the new version.

Return Values:
If successful the call will return **0**.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

value - whether or not to use the older version of setting up the information for poly1305. Passing a flag value of 1 indicates yes use the old poly AEAD, to switch back to using the new version pass a flag value of 0.

Example:

```
int ret = 0;
WOLFSSL* ssl;
...

ret = wolfSSL_use_old_poly(ssl, 1);
if (ret != 0) {
      /*failed to set poly1305 AEAD version*/
}
```

wolfSSL_check_domain_name

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_check_domain_name(WOLFSSL* ssl, const char* dn);

Description:
wolfSSL by default checks the peer certificate for a valid date range and a verified signature. Calling this function before wolfSSL_connect() or wolfSSL_accept() will add a domain name check to the list of checks to perform. **dn** holds the domain name to check against the peer certificate when it's received.

Return Values:
If successful the call will return **SSL_SUCCESS**.

SSL_FAILURE will be returned if a memory error was encountered.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

dn - domain name to check against the peer certificate when received.

Example:

```
int ret = 0;
WOLFSSL* ssl;
char* domain = (char*) “www.yassl.com”;
...

ret = wolfSSL_check_domain_name(ssl, domain);
if (ret != SSL_SUCCESS) {
      /*failed to enable domain name check*/
}
```

See Also:
NA

wolfSSL_set_cipher_list

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_set_cipher_list(WOLFSSL* ssl, const char* list);

Description:
This function sets cipher suite list for a given WOLFSSL object (SSL session). The ciphers in the list should be sorted in order of preference from highest to lowest. Each call to wolfSSL_set_cipher_list() resets the cipher suite list for the specific SSL session to the provided list each time the function is called.

The cipher suite list, **list**, is a null-terminated text string, and a colon-delimited list. For example, one value for **list** may be

"DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:AES256-SHA256"

Valid cipher values are the full name values from the cipher_names[] array in src/internal.c (for a definite list of valid cipher values check src/internal.c):

RC4-SHA
RC4-MD5
DES-CBC3-SHA
AES128-SHA
AES256-SHA
NULL-SHA
NULL-SHA256
DHE-RSA-AES128-SHA
DHE-RSA-AES256-SHA
PSK-AES128-CBC-SHA256
PSK-AES128-CBC-SHA
PSK-AES256-CBC-SHA
PSK-NULL-SHA256
PSK-NULL-SHA
HC128-MD5
HC128-SHA
HC128-B2B256
AES128-B2B256
AES256-B2B256
RABBIT-SHA
NTRU-RC4-SHA
NTRU-DES-CBC3-SHA
NTRU-AES128-SHA
NTRU-AES256-SHA
QSH
AES128-CCM-8
AES256-CCM-8
ECDHE-ECDSA-AES128-CCM-8
ECDHE-ECDSA-AES256-CCM-8
ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES256-SHA
ECDHE-ECDSA-AES128-SHA
ECDHE-ECDSA-AES256-SHA
ECDHE-RSA-RC4-SHA
ECDHE-RSA-DES-CBC3-SHA
ECDHE-ECDSA-RC4-SHA
ECDHE-ECDSA-DES-CBC3-SHA
AES128-SHA256

AES256-SHA256
DHE-RSA-AES128-SHA256
DHE-RSA-AES256-SHA256
ECDH-RSA-AES128-SHA
ECDH-RSA-AES256-SHA
ECDH-ECDSA-AES128-SHA
ECDH-ECDSA-AES256-SHA
ECDH-RSA-RC4-SHA
ECDH-RSA-DES-CBC3-SHA
ECDH-ECDSA-RC4-SHA
ECDH-ECDSA-DES-CBC3-SHA
AES128-GCM-SHA256
AES256-GCM-SHA384
DHE-RSA-AES128-GCM-SHA256
DHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDH-RSA-AES128-GCM-SHA256
ECDH-RSA-AES256-GCM-SHA384
ECDH-ECDSA-AES128-GCM-SHA256
ECDH-ECDSA-AES256-GCM-SHA384
CAMELLIA128-SHA
DHE-RSA-CAMELLIA128-SHA
CAMELLIA256-SHA
DHE-RSA-CAMELLIA256-SHA
CAMELLIA128-SHA256
DHE-RSA-CAMELLIA128-SHA256
CAMELLIA256-SHA256
DHE-RSA-CAMELLIA256-SHA256
ECDHE-RSA-AES128-SHA256
ECDHE-ECDSA-AES128-SHA256
ECDH-RSA-AES128-SHA256
ECDH-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-SHA384
ECDH-RSA-AES256-SHA384
ECDH-ECDSA-AES256-SHA384

Return Values:

SSL_SUCCESS will be returned upon successful function completion, otherwise **SSL_FAILURE** will be returned on failure.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

list - null-terminated text string and a colon-delimited list of cipher suites to use with the specified SSL session.

Example:

```
int ret = 0;
WOLFSSL* ssl = 0;
...
ret = wolfSSL_set_cipher_list(ssl,
“DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:AES256-SHA256”);
if (ret != SSL_SUCCESS) {
      /*failed to set cipher suite list*/
}
```

See Also:
wolfSSL_CTX_set_cipher_list
wolfSSL_new

wolfSSL_CTX_set_cipher_list

Synopsis:
int wolfSSL_CTX_set_cipher_list(WOLFSSL_CTX* ctx, const char* list);

Description:
This function sets cipher suite list for a given WOLFSSL_CTX. This cipher suite list becomes the default list for any new SSL sessions (WOLFSSL) created using this context. The ciphers in the list should be sorted in order of preference from highest to lowest. Each call to wolfSSL_CTX_set_cipher_list() resets the cipher suite list for the specific SSL context to the provided list each time the function is called.

The cipher suite list, **list**, is a null-terminated text string, and a colon-delimited list. For example, one value for **list** may be

"DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:AES256-SHA256"

Valid cipher values are the full name values from the cipher_names[] array in src/internal.c (for a definite list of valid cipher values check src/internal.c):

RC4-SHA
RC4-MD5
DES-CBC3-SHA
AES128-SHA
AES256-SHA
NULL-SHA
NULL-SHA256
DHE-RSA-AES128-SHA
DHE-RSA-AES256-SHA
PSK-AES128-CBC-SHA256
PSK-AES128-CBC-SHA
PSK-AES256-CBC-SHA
PSK-NULL-SHA256
PSK-NULL-SHA
HC128-MD5
HC128-SHA
HC128-B2B256
AES128-B2B256
AES256-B2B256
RABBIT-SHA
QSH
AES128-CCM-8
AES256-CCM-8
ECDHE-ECDSA-AES128-CCM-8
ECDHE-ECDSA-AES256-CCM-8
ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES256-SHA
ECDHE-ECDSA-AES128-SHA
ECDHE-ECDSA-AES256-SHA
ECDHE-RSA-RC4-SHA
ECDHE-RSA-DES-CBC3-SHA
ECDHE-ECDSA-RC4-SHA
ECDHE-ECDSA-DES-CBC3-SHA
AES128-SHA256

AES256-SHA256
DHE-RSA-AES128-SHA256
DHE-RSA-AES256-SHA256
ECDH-RSA-AES128-SHA
ECDH-RSA-AES256-SHA
ECDH-ECDSA-AES128-SHA
ECDH-ECDSA-AES256-SHA
ECDH-RSA-RC4-SHA
ECDH-RSA-DES-CBC3-SHA
ECDH-ECDSA-RC4-SHA
ECDH-ECDSA-DES-CBC3-SHA
AES128-GCM-SHA256
AES256-GCM-SHA384
DHE-RSA-AES128-GCM-SHA256
DHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDH-RSA-AES128-GCM-SHA256
ECDH-RSA-AES256-GCM-SHA384
ECDH-ECDSA-AES128-GCM-SHA256
ECDH-ECDSA-AES256-GCM-SHA384
CAMELLIA128-SHA
DHE-RSA-CAMELLIA128-SHA
CAMELLIA256-SHA
DHE-RSA-CAMELLIA256-SHA
CAMELLIA128-SHA256
DHE-RSA-CAMELLIA128-SHA256
CAMELLIA256-SHA256
DHE-RSA-CAMELLIA256-SHA256
ECDHE-RSA-AES128-SHA256
ECDHE-ECDSA-AES128-SHA256
ECDH-RSA-AES128-SHA256
ECDH-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-SHA384
ECDH-RSA-AES256-SHA384
ECDH-ECDSA-AES256-SHA384

Return Values:

SSL_SUCCESS will be returned upon successful function completion, otherwise **SSL_FAILURE** will be returned on failure.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

list - null-terminated text string and a colon-delimited list of cipher suites to use with the specified SSL context.

Example:

```
WOLFSSL_CTX* ctx = 0;
...
ret = wolfSSL_CTX_set_cipher_list(ctx,
“DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:AES256-SHA256”);
if (ret != SSL_SUCCESS) {
      /*failed to set cipher suite list*/
}
```

See Also:
wolfSSL_set_cipher_list
wolfSSL_CTX_new

Synopsis:
#include <wolfssl/ssl.h>

EVP_aes_128_ecb ->

const WOLFSSL_EVP_CIPHER* wolfSSL_EVP_aes_128_ecb(void);

EVP_aes_192_ecb ->

const WOLFSSL_EVP_CIPHER* wolfSSL_EVP_aes_192_ecb(void);

EVP_aes_256_ecb ->

const WOLFSSL_EVP_CIPHER* wolfSSL_EVP_aes_256_ecb(void);

Description:

Getter functions for the respective WOLFSSL_EVP_CIPHER pointers. wolfSSL_EVP_init() must be called once in the program first to populate these cipher strings.

Return Values:

Returns a WOLFSSL_EVP_CIPHER pointer.

Parameters:

None

Example:

```
WOLFSSL_EVP_CIPHER* cipher;
cipher = wolfSSL_EVP_aes_192_ecb();

….
```

See Also:

wolfSSL_EVP_CIPHER_CTX_init

wolfSSL_EVP_CIPHER_block_size

Synopsis:

#include <wolfssl/openssl/evp.h>

EVP_CIPHER_block_size ->
int wolfSSL_EVP_CIPHER_block_size(const WOLFSSL_EVP_CIPHER* cipher);

Description:

This is a getter function for the block size of cipher.

Return Values:

Returns the block size.

Parameters:

cipher - cipher to get block size of.

Example:

```
printf(“block size = %d\n”,
wolfSSL_EVP_CIPHER_block_size(wolfSSL_EVP_aes_256_ecb()));
```

See Also:

wolfSSL_EVP_aes_256_ctr

wolfSSL_EVP_CIPHER_CTX_block_size

Synopsis:

#include <wolfssl/openssl/evp.h>

EVP_CIPHER_CTX_block_size ->
int wolfSSL_EVP_CIPHER_CTX_block_size(const WOLFSSL_EVP_CIPHER_CTX* ctx);

Description:

This is a getter function for the ctx block size.

Return Values:

Returns ctx->block_size.

Parameters:

ctx - the cipher ctx to get block size of.

Example:

```
const WOLFSSL_CVP_CIPHER_CTX* ctx;
//set up ctx
printf(“block size = %d\n”, wolfSSL_EVP_CIPHER_CTX_block_size(ctx));
```

See Also:

wolfSSL_EVP_CIPHER_block_size

wolfSSL_EVP_CIPHER_CTX_set_flags

Synopsis:

#include <wolfssl/openssl/evp.h>

EVP_CIPHER_CTX_set_flags ->
void wolfSSL_EVP_CIPHER_CTX_set_flags(WOLFSSL_EVP_CIPHER_CTX* ctx, int flags);

Description:

Setter function for WOLFSSL_EVP_CIPHER_CTX structure.

Return Values:

None

Parameters:

ctx - structure to set flag.

flag - flag to set in structure.

Example:

```
WOLFSSL_EVP_CIPHER_CTX* ctx;
int flag;
// create ctx

wolfSSL_EVP_CIPHER_CTX_set_flags(ctx, flag);
```

See Also:

wolfSSL_EVP_CIPHER_flags

wolfSSL_EVP_CIPHER_CTX_set_key_length

Synopsis:

#include <wolfssl/openssl/evp.h>

EVP_CIPHER_CTX_set_key_length ->
int wolfSSL_EVP_CIPHER_CTX_set_key_length(WOLFSSL_EVP_CIPHER_CTX* ctx, int keylen);

Description:

Setter function for WOLFSSL_EVP_CIPHER_CTX structure key length.

Return Values:

SSL_SUCCESS: If successfully set.

SSL_FAILURE: If failed to set key length/

Parameters:

ctx - structure to set key length.

keylen - key length.

Example:

```
WOLFSSL_EVP_CIPHER_CTX* ctx;
int keylen;
// create ctx

wolfSSL_EVP_CIPHER_CTX_set_key_length(ctx, keylen);
```

See Also:

wolfSSL_EVP_CIPHER_flags

wolfSSL_EVP_CIPHER_CTX_set_padding

Synopsis:

#include <wolfssl/openssl/evp.h>

EVP_CIPHER_CTX_set_padding ->
int wolfSSL_EVP_CIPHER_CTX_set_padding(WOLFSSL_EVP_CIPHER_CTX* ctx, int padding);

Description:

Setter function for WOLFSSL_EVP_CIPHER_CTX structure to use padding.

Return Values:

SSL_SUCCESS: If successfully set.

BAD_FUNC_ARG: If null argument passed in.

Parameters:

ctx - structure to set padding flag.

padding - 0 for not setting padding, 1 for setting padding.

Example:

```
WOLFSSL_EVP_CIPHER_CTX* ctx;
// create ctx
wolfSSL_EVP_CIPHER_CTX_set_padding(ctx, 1);
```

See Also:

wolfSSL_EVP_CIPHER_flags

wolfSSL_EVP_CipherFinal

Synopsis:

#include <wolfssl/openssl/evp.h>

EVP_CipherFinal ->
int wolfSSL_EVP_CipherFinal(WOLFSSL_EVP_CIPHER_CTX* ctx, unsigned char* out, int* out1);

Description:

This function performs the final cipher operations adding in padding. If WOLFSSL_EVP_CIPH_NO_PADDING flag is set in WOLFSSL_EVP_CIPHER_CTX structure then 1 is returned and no encryption/decryption is done. If padding flag is seti padding is added and encrypted when ctx is set to encrypt, padding values are checked when set to decrypt.

Return Values:

1:Returned on success

0: If encountering a failure.

Parameters:

ctx - structure to decrypt/encrypt with.

out - buffer for final decrypt/encrypt.

out1 - size of out buffer when data has been added by function.

Example:

```
WOLFSSL_EVP_CIPHER_CTX* ctx;
int out1;
unsigned char out[64];
// create ctx

wolfSSL_EVP_CipherFinal(ctx, out, &out1);
```

See Also:

wolfSSL_EVP_CIPHER_CTX_new

wolfSSL_CipherInit_ex

Synopsis:

#include <wolfssl/openssl/evp.h>

EVP_CipherInit_ex ->
int wolfSSL_CipherInit_ex(WOLFSSL_EVP_CIPHER_CTX* ctx, const WOLFSSL_EVP_CIPHER* type, WOLFSSL_ENGINE* impl, unsigned char* key, unsigned char* iv, int enc);

Description:

Function for initializing WOLFSSL_EVP_CIPHER_CTX. This function is a wrapper for wolfSSL_CipherInit() because wolfSSL does not use WOLFSSL_ENGINE.

Return Values:

SSL_SUCCESS: If successfully set.

SSL_FAILURE: If not successful.

Parameters:

ctx - structure to initialize.

type - type of encryption/decryption to do, for example AES.

impl - engine to use. N/A for wolfSSL, can be NULL.

key - key to set .

iv - iv if needed by algorithm.

enc - encryption (1) or decryption (0) flag.

Example:

```
WOLFSSL_EVP_CIPHER_CTX* ctx = NULL;
WOLFSSL_ENGINE* e = NULL;
unsigned char key[16];
unsigned char iv[12];
```

```
wolfCrypt_Init();

ctx = wolfSSL_EVP_CIPHER_CTX_new();
if (ctx == NULL) {
 printf("issue creating ctx\n");
 return -1;
}

printf("cipher init ex error ret = %d\n", wolfSSL_EVP_CipherInit_ex(NULL,
EVP_aes_128_    cbc(), e, key, iv, 1));
printf("cipher init ex success ret = %d\n", wolfSSL_EVP_CipherInit_ex(ctx,
EVP_aes_128_c    bc(), e, key, iv, 1));
// free resources
```

See Also:

wolfSSL_EVP_CIPHER_CTX_new, wolfCrypt_Init, wolfSSL_EVP_CIPHER_CTX_free

wolfSSL_EVP_CipherUpdate

Synopsis:

#include <wolfssl/openssl/evp.h>

EVP_CipherUpdate ->
int wolfSSL_EVP_CipherUpdate(WOLFSSL_EVP_CIPHER_CTX* ctx, unsigned char* out, int *outl, const unsigned char* in, int inl);

Description:

Function for encrypting/decrypting data. In buffer is added to be encrypted or decrypted and out buffer holds the results. outl will be the length of encrypted/decrypted information.

Return Values:

SSL_SUCCESS: If successfull.

SSL_FAILURE: If not successful.

Parameters:

ctx - structure to get cipher type from.

out - buffer to hold output.

outl - adjusted to be size of output.

in - buffer to perform operation on.

inl - length of input buffer.

Example:

```
WOLFSSL_EVP_CIPHER_CTX* ctx = NULL;
unsigned char out[100];
int outl;
unsigned char in[100];
int inl = 100;

ctx = wolfSSL_EVP_CIPHER_CTX_new();
// set up ctx
ret = wolfSSL_EVP_CipherUpdate(ctx, out, outl, in, inl);
// check ret value
// buffer out holds outl bytes of data
// free resources
```

See Also:

wolfSSL_EVP_CIPHER_CTX_new, wolfCrypt_Init, wolfSSL_EVP_CIPHER_CTX_free

wolfSSL_EVP_DecryptInit_ex

Synopsis:

#include <wolfssl/openssl/evp.h>

EVP_DecryptInit_ex ->
int wolfSSL_EVP_DecryptInit_ex

Description:

Function for initializing WOLFSSL_EVP_CIPHER_CTX. This function is a wrapper for wolfSSL_EVP_CipherInit() because wolfSSL does not use WOLFSSL_ENGINE. Sets encrypt flag to be decrypt.

Return Values:

SSL_SUCCESS: If successfully set.

SSL_FAILURE: If not successful.

Parameters:

ctx - structure to initialize.

type - type of encryption/decryption to do, for example AES.

impl - engine to use. N/A for wolfSSL, can be NULL.

key - key to set .

iv - iv if needed by algorithm.

enc - encryption (1) or decryption (0) flag.

Example:

```
WOLFSSL_EVP_CIPHER_CTX* ctx = NULL;
WOLFSSL_ENGINE* e = NULL;
unsigned char key[16];
unsigned char iv[12];

wolfCrypt_Init();

ctx = wolfSSL_EVP_CIPHER_CTX_new();
if (ctx == NULL) {
 printf("issue creating ctx\n");
 return -1;
}

printf("cipher init ex error ret = %d\n", wolfSSL_EVP_DecryptInit_ex(NULL,
EVP_aes_128_    cbc(), e, key, iv, 1));
printf("cipher init ex success ret = %d\n", wolfSSL_EVP_DecryptInit_ex(ctx,
EVP_aes_128_c    bc(), e, key, 1));
// free resources
```

See Also:

wolfSSL_EVP_CIPHER_CTX_new, wolfCrypt_Init, wolfSSL_EVP_CIPHER_CTX_free

wolfSSL_EVP_des_cbc, wolfSSL_EVP_des_ecb

Synopsis:

#include <wolfssl/openssl/evp.h>

EVP_des_cbc ->

const WOLFSSL_EVP_CIPHER* wolfSSL_EVP_des_cbc(void);

EVP_des_ecb ->

const WOLFSSL_EVP_CIPHER* wolfSSL_EVP_des_ecb(void);

Description:

Getter functions for the respective WOLFSSL_EVP_CIPHER pointers. wolfSSL_EVP_init() must be called once in the program first to populate these cipher strings. WOLFSSL_DES_ECB macro must be defined for wolfSSL_EVP_des_ecb().

Return Values:

Returns a WOLFSSL_EVP_CIPHER pointer for DES operations.

Parameters:

None

Example:

```
WOLFSSL_EVP_CIPHER* cipher;
cipher = wolfSSL_EVP_des_ecb();
….
```

See Also:

wolfSSL_EVP_CIPHER_CTX_init

wolfSSL_EVP_des_cbc, wolfSSL_EVP_des_ecb

Synopsis:

#include <wolfssl/openssl/evp.h>

EVP_des_ede3_cbc ->

const WOLFSSL_EVP_CIPHER* wolfSSL_EVP_des_ede_cbc(void);

EVP_des_ede3_ecb ->

const WOLFSSL_EVP_CIPHER* wolfSSL_EVP_des_ede3_ecb(void);

Description:

Getter functions for the respective WOLFSSL_EVP_CIPHER pointers. wolfSSL_EVP_init() must be called once in the program first to populate these cipher strings. WOLFSSL_DES_ECB macro must be defined for wolfSSL_EVP_des_ede3_ecb().

Return Values:

Returns a WOLFSSL_EVP_CIPHER pointer for DES EDE3 operations.

Parameters:

None

Example:

```
printf("block size des ede3 cbc = %d\n",
wolfSSL_EVP_CIPHER_block_size(wolfSSL_EVP_des_ede3_cbc()));
```

```
printf("block size des ede3 ecb = %d\n",
wolfSSL_EVP_CIPHER_block_size(wolfSSL_EVP_des_ede3_ecb()));
```

See Also:
wolfSSL_EVP_CIPHER_CTX_init

wolfSSL_EVP_DigestInit_ex

Synopsis:
#include <wolfssl/openssl/evp.h>

EVP_DigestInit_ex ->
int wolfSSL_EVP_DigestInit_ex(WOLFSSL_EVP_MD_CTX* ctx, const WOLFSSL_EVP_MD* type, WOLFSSL_ENGINE* impl);

Description:
Function for initializing WOLFSSL_EVP_MD_CTX. This function is a wrapper for wolfSSL_EVP_DigestInit() because wolfSSL does not use WOLFSSL_ENGINE.

Return Values:
SSL_SUCCESS: If successfully set.

SSL_FAILURE: If not successful.

Parameters:
ctx - structure to initialize.

type - type of hash to do, for example SHA.

impl - engine to use. N/A for wolfSSL, can be NULL.

Example:
```
WOLFSSL_EVP_MD_CTX* md = NULL;

wolfCrypt_Init();

md = wolfSSL_EVP_MD_CTX_new();
```

```
if (md == NULL) {
  printf("error setting md\n");
  return -1;
}

printf("cipher md init ret = %d\n", wolfSSL_EVP_DigestInit_ex(md,
wolfSSL_EVP_sha1(), e));

//free resources
```

See Also:

wolfSSL_EVP_MD_CTX_new, wolfCrypt_Init, wolfSSL_EVP_MD_CTX_free

wolfSSL_EVP_EncryptInit_ex

Synopsis:

#include <wolfssl/openssl/evp.h>

EVP_EncryptInit_ex ->
int wolfSSL_EVP_EncryptInit_ex(WOLFSSL_EVP_CIPHER_CTX* ctx, const WOLFSSL_EVP_Cipher* type, WOLFSSL_ENGINE* impl, unsigned char* key, unsigned char* iv);

Description:

Function for initializing WOLFSSL_EVP_CIPHER_CTX. This function is a wrapper for wolfSSL_EVP_CipherInit() because wolfSSL does not use WOLFSSL_ENGINE. Sets encrypt flag to be encrypt.

Return Values:

SSL_SUCCESS: If successfully set.

SSL_FAILURE: If not successful.

Parameters:

ctx - structure to initialize.

type - type of encryption to do, for example AES.

impl - engine to use. N/A for wolfSSL, can be NULL.

key - key to use.

iv - iv to use.

Example:

```
WOLFSSL_EVP_CIPHER_CTX* ctx = NULL;

wolfCrypt_Init();

ctx = wolfSSL_EVP_CIPHER_CTX_new();
if (ctx == NULL) {
  printf("error setting ctx\n");
  return -1;
}

printf("cipher ctx init ret = %d\n", wolfSSL_EVP_EncryptInit_ex(ctx,
wolfSSL_EVP_aes_128_cbc(), e, key, iv));

//free resources
```

See Also:

wolfSSL_EVP_CIPHER_CTX_new, wolfCrypt_Init, wolfSSL_EVP_CIPHER_CTX_free

wolfSSL_set_compression

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_set_compression(WOLFSSL* ssl);

Description:

Turns on the ability to use compression for the SSL connection. Both sides must have compression turned on otherwise compression will not be used. The zlib library performs the actual data compression. To compile into the library use **--with-libz** for the configure system and define HAVE_LIBZ otherwise.

Keep in mind that while compressing data before sending decreases the actual size of the messages being sent and received, the amount of data saved by compression usually takes longer in time to analyze than it does to send it raw on all but the slowest of networks.

Return Values:

If successful the call will return **SSL_SUCCESS**.

NOT_COMPILED_IN will be returned if compression support wasn't built into the library.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

Example:

```
int ret = 0;
WOLFSSL* ssl = 0;
...
ret = wolfSSL_set_compression(ssl);
if (ret == SSL_SUCCESS) {
      /*successfully enabled compression for SSL session*/
}
```

See Also:

NA

wolfSSL_set_fd

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_set_fd(WOLFSSL* ssl, int fd);

Description:

This function assigns a file descriptor (**fd**) as the input/output facility for the SSL connection. Typically this will be a socket file descriptor.

Return Values:

If successful the call will return **SSL_SUCCESS**, otherwise, **Bad_FUNC_ARG** will be returned.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

fd - file descriptor to use with SSL/TLS connection.

Example:

```
int sockfd;
WOLFSSL* ssl = 0;
...

ret = wolfSSL_set_fd(ssl, sockfd);
if (ret != SSL_SUCCESS) {
      /*failed to set SSL file descriptor*/
}
```

See Also:

wolfSSL_SetIOSend
wolfSSL_SetIORecv
wolfSSL_SetIOReadCtx
wolfSSL_SetIOWriteCtx

wolfSSL_set_group_messages

Synopsis:

int wolfSSL_set_group_messages(WOLFSSL* ssl);

Description:

This function turns on grouping of handshake messages where possible.

Return Values:

SSL_SUCCESS will be returned upon success.

BAD_FUNC_ARG will be returned if the input context is null.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

Example:

```
WOLFSSL* ssl = 0;
...
ret = wolfSSL_set_group_messages(ssl);
if (ret != SSL_SUCCESS) {
      // failed to set handshake message grouping
}
```

See Also:
wolfSSL_CTX_set_group_messages
wolfSSL_new

wolfSSL_CTX_set_group_messages

Synopsis:
int wolfSSL_CTX_set_group_messages(WOLFSSL_CTX* ctx);

Description:
This function turns on grouping of handshake messages where possible.

Return Values:

SSL_SUCCESS will be returned upon success.

BAD_FUNC_ARG will be returned if the input context is null.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

Example:

```
WOLFSSL_CTX* ctx = 0;
```

```
...
ret = wolfSSL_CTX_set_group_messages(ctx);
if (ret != SSL_SUCCESS) {
      /*failed to set handshake message grouping*/
}
```

See Also:

wolfSSL_set_group_messages
wolfSSL_CTX_new

wolfSSL_set_session

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_set_session(WOLFSSL* ssl, WOLFSSL_SESSION* session);

Description:

This function sets the session to be used when the SSL object, **ssl**, is used to establish a SSL/TLS connection.

For session resumption, before calling wolfSSL_shutdown() with your session object, an application should save the session ID from the object with a call to wolfSSL_get_session(), which returns a pointer to the session. Later, the application should create a new WOLFSSL object and assign the saved session with wolfSSL_set_session(). At this point, the application may call wolfSSL_connect() and wolfSSL will try to resume the session. The wolfSSL server code allows session resumption by default.

Return Values:

SSL_SUCCESS will be returned upon successfully setting the session.

SSL_FAILURE will be returned on failure. This could be caused by the session cache being disabled, or if the session has timed out.

Parameters:

ssl - pointer to the SSL object, created with wolfSSL_new().

session - pointer to the WOLFSSL_SESSION used to set the session for **ssl**.

Example:

```
int ret = 0;
WOLFSSL* ssl = 0;
WOLFSSL_SESSION* session;
...

ret = wolfSSL_get_session(ssl, session);
if (ret != SSL_SUCCESS) {
      /*failed to set the SSL session*/
}
...
```

See Also:
wolfSSL_get_session

wolfSSL_CTX_set_session_cache_mode

Synopsis:
long wolfSSL_CTX_set_session_cache_mode(WOLFSSL_CTX* ctx, long mode);

Description:
This function enables or disables SSL session caching. Behavior depends on the value used for **mode**. The following values for **mode** are available:

SSL_SESS_CACHE_OFF
 - disable session caching. Session caching is turned on by default.

SSL_SESS_CACHE_NO_AUTO_CLEAR
 - Disable auto-flushing of the session cache. Auto-flushing is turned on by default.

Return Values:

SSL_SUCCESS will be returned upon success.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

mode - modifier used to change behavior of the session cache.

Example:

```
WOLFSSL_CTX* ctx = 0;
...
ret = wolfSSL_CTX_set_session_cache_mode(ctx, SSL_SESS_CACHE_OFF);
if (ret != SSL_SUCCESS) {
      /*failed to turn SSL session caching off*/
}
```

See Also:
wolfSSL_flush_sessions
wolfSSL_get_session
wolfSSL_set_session
wolfSSL_get_sessionID
wolfSSL_CTX_set_timeout

wolfSSL_set_timeout

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_set_timeout(WOLFSSL* ssl, unsigned int to);

Description:
This function sets the SSL session timeout value in seconds.

Return Values:

SSL_SUCCESS will be returned upon successfully setting the session.

BAD_FUNC_ARG will be returned if **ssl** is NULL.

Parameters:

ssl - pointer to the SSL object, created with wolfSSL_new().

to - value, in seconds, used to set the SSL session timeout.

Example:

```
int ret = 0;
WOLFSSL* ssl = 0;
...

ret = wolfSSL_set_timeout(ssl, 500);
if (ret != SSL_SUCCESS) {
      /*failed to set session timeout value*/
}
...
```

See Also:

wolfSSL_get_session
wolfSSL_set_session

wolfSSL_CTX_set_timeout

Synopsis:

int wolfSSL_CTX_set_timeout(WOLFSSL_CTX* ctx, unsigned int to);

Description:

This function sets the timeout value for SSL sessions, in seconds, for the specified SSL context.

Return Values:

SSL_SUCCESS will be returned upon success.

BAD_FUNC_ARG will be returned when the input context (**ctx**) is null.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

to - session timeout value in seconds

Example:

```
WOLFSSL_CTX*    ctx    = 0;
...
ret = wolfSSL_CTX_set_timeout(ctx, 500);
if (ret != SSL_SUCCESS) {
      /*failed to set session timeout value*/
}
```

See Also:
wolfSSL_flush_sessions
wolfSSL_get_session
wolfSSL_set_session
wolfSSL_get_sessionID
wolfSSL_CTX_set_session_cache_mode

wolfSSL_set_using_nonblock

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_set_using_nonblock(WOLFSSL* ssl, int nonblock);

Description:
This function informs the WOLFSSL object that the underlying I/O is non-blocking.

After an application creates a WOLFSSL object, if it will be used with a non-blocking socket, call wolfSSL_set_using_nonblock() on it. This lets the WOLFSSL object know that receiving EWOULDBLOCK means that the recvfrom call would block rather than that it timed out.

Return Values:

This function does not have a return value.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

nonblock - value used to set non-blocking flag on WOLFSSL object. Use 1 to specify non-blocking, otherwise 0.

Example:

```
WOLFSSL* ssl = 0;
...

wolfSSL_set_using_nonblock(ssl, 1);
```

See Also:
wolfSSL_get_using_nonblock
wolfSSL_dtls_got_timeout
wolfSSL_dtls_get_current_timeout

wolfSSL_set_verify

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_set_verify(WOLFSSL* ssl, int mode, VerifyCallback vc);

typedef int (*VerifyCallback)(int, WOLFSSL_X509_STORE_CTX*);

Description:
This function sets the verification method for remote peers and also allows a verify callback to be registered with the SSL session. The verify callback will be called only when a verification failure has occurred. If no verify callback is desired, the NULL pointer can be used for **verify_callback**.

The verification **mode** of peer certificates is a logically OR'd list of flags. The possible flag values include:

SSL_VERIFY_NONE

> **Client mode**: the client will not verify the certificate received from the server and the handshake will continue as normal.

Server mode: the server will not send a certificate request to the client. As such, client verification will not be enabled.

SSL_VERIFY_PEER

Client mode: the client will verify the certificate received from the server during the handshake. This is turned on by default in wolfSSL, therefore, using this option has no effect.

Server mode: the server will send a certificate request to the client and verify the client certificate received.

SSL_VERIFY_FAIL_IF_NO_PEER_CERT

Client mode: no effect when used on the client side.

Server mode: the verification will fail on the server side if the client fails to send a certificate when requested to do so (when using SSL_VERIFY_PEER on the SSL server).

SSL_VERIFY_FAIL_EXCEPT_PSK

Client mode: no effect when used on the client side.

Server mode: the verification is the same as SSL_VERIFY_FAIL_IF_NO_PEER_CERT except in the case of a PSK connection. If a PSK connection is being made then the connection will go through without a peer cert.

Return Values:

This function has no return value.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

mode - session timeout value in seconds

verify_callback - callback to be called when verification fails. If no callback is desired, the NULL pointer can be used for verify_callback.

Example:

```
WOLFSSL* ssl = 0;
...

wolfSSL_set_verify(ssl, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT,
0);
```

See Also:
wolfSSL_CTX_set_verify

wolfSSL_CTX_set_verify

Synopsis:
void wolfSSL_CTX_set_verify(WOLFSSL_CTX* ctx, int mode,
VerifyCallback vc);

typedef int (*VerifyCallback)(int, WOLFSSL_X509_STORE_CTX*);

Description:
This function sets the verification method for remote peers and also allows a verify callback to be registered with the SSL context. The verify callback will be called only when a verification failure has occurred. If no verify callback is desired, the NULL pointer can be used for **verify_callback**.

The verification **mode** of peer certificates is a logically OR'd list of flags. The possible flag values include:

SSL_VERIFY_NONE

Client mode: the client will not verify the certificate received from the server and the handshake will continue as normal.

Server mode: the server will not send a certificate request to the client. As such, client verification will not be enabled.

SSL_VERIFY_PEER

Client mode: the client will verify the certificate received from the server during the handshake. This is turned on by default in wolfSSL, therefore, using this option has no effect.

Server mode: the server will send a certificate request to the client and verify the client certificate received.

SSL_VERIFY_FAIL_IF_NO_PEER_CERT

Client mode: no effect when used on the client side.

Server mode: the verification will fail on the server side if the client fails to send a certificate when requested to do so (when using SSL_VERIFY_PEER on the SSL server).

SSL_VERIFY_FAIL_EXCEPT_PSK

Client mode: no effect when used on the client side.

Server mode: the verification is the same as SSL_VERIFY_FAIL_IF_NO_PEER_CERT except in the case of a PSK connection. If a PSK connection is being made then the connection will go through without a peer cert.

Return Values:

This function has no return value.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

mode - session timeout value in seconds

verify_callback - callback to be called when verification fails. If no callback is desired, the NULL pointer can be used for verify_callback.

Example:

```
WOLFSSL_CTX*    ctx    = 0;
...
wolfSSL_CTX_set_verify(ctx, SSL_VERIFY_PEER |
                     SSL_VERIFY_FAIL_IF_NO_PEER_CERT, 0);
```

See Also:
wolfSSL_set_verify

wolfSSL_CTX_get_verify_depth

Synopsis:
#include <wolfssl/ssl.h>

long wolfSSL_CTX_get_verify_depth(WOLFSSL_CTX* ctx);

Description:
This function gets the certificate chaining depth using the CTX structure.

Return Values:
MAX_CHAIN_DEPTH - returned if the CTX struct is not NULL. The constant representation of the max certificate chain peer depth.

BAD_FUNC_ARG - returned if the CTX structure is NULL.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

Example:

```
WOLFSSL_METHOD method; /*protocol method*/
WOLFSSL_CTX* ctx = WOLFSSL_CTX_new(method);
…
long ret = wolfSSL_CTX_get_verify_depth(ctx);
```

```
if(ret == EXPECTED){
      /*You have the expected value*/
} else {
      /*Handle an unexpected depth*/
}
```

See Also:
wolfSSL_CTX_use_certificate_chain_file
wolfSSL_get_verify_depth

wolfSSL_CTX_UnloadCAs

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_UnloadCAs(WOLFSSL_CTX* ctx);

Description:
This function unloads the CA signer list and frees the whole signer table.

Return Values:
SSL_SUCCESS - returned on successful execution of the function.

BAD_FUNC_ARG - returned if the WOLFSSL_CTX struct is NULL or there are otherwise unpermitted argument values passed in a subroutine.

BAD_MUTEX_E - returned if there was a mutex error. The LockMutex() did not return 0.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

Example:

```
WOLFSSL_METHOD method = wolfTLSv1_2_client_method();
WOLFSSL_CTX* ctx = WOLFSSL_CTX_new(method);
…
```

```
if(!wolfSSL_CTX_UnloadCAs(ctx)){
      /*The function did not unload CAs*/
}
```

See Also:

wolfSSL_CertManagerUnloadCAs
LockMutex
FreeSignerTable
UnlockMutex

wolfSSL_dtls_set_timeout_init

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_dtls_set_timeout_init(WOLFSSL* ssl, int timeout);

Description:

This function sets the dtls timeout.

Return Values:

SSL_SUCCESS - returned if the function executes without an error. The **dtls_timeout_init** and the **dtls_timeout** members of SSL have been set.

BAD_FUNC_ARG - returned if the WOLFSSL struct is NULL or if the timeout is not greater than 0. It will also return if the **timeout** argument exceeds the maximum value allowed.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

timeout - an int type that will be set to the **dtls_timeout_init** member of the WOLFSSL structure.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
```

```
WOLFSSL* ssl = wolfSSL_new(ctx);
int timeout = TIMEOUT;  /*timeout value*/
...
if(wolfSSL_dtls_set_timeout_init(ssl, timeout)){
      /*the dtls timeout was set*/
} else {
      /*Failed to set DTLS timeout. */
}
```

See Also:

wolfSSL_dtls_set_timeout_max
wolfSSL_dtls_got_timeout

wolfSSL_GetCookieCtx

Synopsis:

#include <wolfssl/ssl.h>

void* wolfSSL_GetCookieCtx(WOLFSSL* ssl);

Description:

This function returns the **IOCB_CookieCtx** member of the WOLFSSL structure.

Return Values:

The function returns a **void pointer** value stored in the IOCB_CookieCtx.

NULL - if the WOLFSSL struct is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
void* cookie;
...
cookie = wolfSSL_GetCookieCtx(ssl);

if(cookie != NULL){
      /*You have the cookie */
}
```

See Also:

wolfSSL_SetCookieCtx
wolfSSL_CTX_SetGenCookie

wolfSSL_CTX_UseSessionTicket

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_UseSessionTicket(WOLFSSL_CTX* ctx)

Description:

This function sets wolfSSL context to use a session ticket.

Return Values:

SSL_SUCCESS: Function executed successfully.

BAD_FUNC_ARG: Returned if **ctx** is null.

MEMORY_E: Error allocating memory in internal function.

Parameters:

ctx - The WOLFSSL_CTX structure to use.

Example:

```
wolfSSL_Init();
WOLFSSL_CTX* ctx;
WOLFSSL_METHOD method = /* Some wolfSSL method */
ctx = wolfSSL_CTX_new(method);

if(wolfSSL_CTX_UseSessionTicket(ctx) != SSL_SUCCESS)
{
    /* Error setting session ticket */
}
```

See Also:

TLSX_UseSessionTicket

wolfSSL_UseSupportedQSH

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_UseSupportedQSH(WOLFSSL* ssl, word16 name)

Description:

This function sets the ssl session to use supported QSH provided by name.

Return Values:

SSL_SUCCESS: Successfully set supported QSH.

BAD_FUNC_ARG: ssl is null or name is invalid.

MEMORY_E: Error allocating memory for operation.

Parameters:

ssl - Pointer to ssl session to use.

name - Name of a supported QSH. Valid names are WOLFSSL_NTRU_EESS439, WOLFSSL_NTRU_EESS593, or WOLFSSL_NTRU_EESS743.

Example:

```
wolfSSL_Init();
WOLFSSL_CTX* ctx;
WOLFSSL* ssl;
WOLFSSL_METHOD method = /* Some wolfSSL method */
```

```
ctx = wolfSSL_CTX_new(method);
ssl = wolfSSL_new(ctx);

word16 qsh_name = WOLFSSL_NTRU_EESS439;

if(wolfSSL_UseSupportedQSH(ssl,qsh_name) != SSL_SUCCESS)
{
    /* Error setting QSH */
}
```

See Also:

TLSX_UseQSHScheme

wolfSSL_UseALPN

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_UseALPN(WOLFSSL* ssl, char *protocol_name_list,
                    word32 protocol_name_listSz, byte options)

Description:

Setup ALPN use for a wolfSSL session.

Return Values:

SSL_SUCCESS: Success

BAD_FUNC_ARG: Returned if **ssl** or **protocol_name_list** is null or **protocol_name_listSz** is too large or **options** contain something not supported.

MEMORY_ERROR: Error allocating memory for protocol list.

SSL_FAILURE: Error

Parameters:

ssl - The wolfSSL session to use.

protocol_name_list - List of protocol names to use. Comma delimited string is required.

protocol_name_listSz - Size of the list of protocol names.

options - WOLFSSL_ALPN_CONTINUE_ON_MISMATCH or WOLFSSL_ALPN_FAILED_ON_MISMATCH.

Example:

```
wolfSSL_Init();
WOLFSSL_CTX* ctx;
WOLFSSL* ssl;
WOLFSSL_METHOD method = /* Some wolfSSL method */
ctx = wolfSSL_CTX_new(method);
ssl = wolfSSL_new(ctx);

char alpn_list[] = { /* ALPN List */ }

if(wolfSSL_UseALPN(ssl, alpn_list, sizeof(alpn_list),
                              WOLFSSL_APN_FAILED_ON_MISMATCH) != SSL_SUCCESS)
{
    /* Error setting session ticket */
}
```

See Also:

TLSX_UseALPN

wolfSSL_CTX_trust_peer_cert

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_trust_peer_cert(WOLFSSL_CTX* ctx, const char* file,
                                                          int type);

Description:

This function loads a certificate to use for verifying a peer when performing a TLS/SSL handshake. The peer certificate sent during the handshake is compared by using the SKID when available and the signature. If these two things do not match then any loaded CAs are used.

Feature is enabled by defining the macro WOLFSSL_TRUST_PEER_CERT

Please see the examples for proper usage.

Return Values:

If successful the call will return **SSL_SUCCESS**.

SSL_FAILURE will be returned if ctx is NULL, or if both file and type are invalid.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn't exist, can't be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

file - pointer to name of the file containing certificates

type - type of certificate being loaded ie SSL_FILETYPE_ASN1 or SSL_FILETYPE_PEM.

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);

...
```

```
ret = wolfSSL_CTX_trust_peer_cert(ctx, “./peer-cert.pem”, SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
/* error loading trusted peer cert  */
}

...
```

See Also:

wolfSSL_CTX_load_verify_buffer
wolfSSL_CTX_use_certificate_file
wolfSSL_CTX_use_PrivateKey_file
wolfSSL_CTX_use_NTRUPrivateKey_file
wolfSSL_CTX_use_certificate_chain_file
wolfSSL_CTX_trust_peer_buffer
wolfSSL_CTX_Unload_trust_peers
wolfSSL_use_certificate_file
wolfSSL_use_PrivateKey_file
wolfSSL_use_certificate_chain_file

wolfSSL_CTX_trust_peer_buffer

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_trust_peer_buffer(WOLFSSL_CTX* ctx, const unsigned char* buffer, long sz, int type);

Description:

This function loads a certificate to use for verifying a peer when performing a TLS/SSL handshake. The peer certificate sent during the handshake is compared by using the SKID when available and the signature. If these two things do not match then any loaded CAs are used. Is the same functionality as wolfSSL_CTX_trust_peer_cert except is from a buffer instead of a file.

Feature is enabled by defining the macro WOLFSSL_TRUST_PEER_CERT

Please see the examples for proper usage.

Return Values:

If successful the call will return **SSL_SUCCESS**.

SSL_FAILURE will be returned if ctx is NULL, or if both file and type are invalid.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn't exist, can't be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

buffer - pointer to the buffer containing certificates

sz - length of the buffer input

type - type of certificate being loaded i.e. SSL_FILETYPE_ASN1 or SSL_FILETYPE_PEM.

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx;

...

ret = wolfSSL_CTX_trust_peer_buffer(ctx, bufferPtr, bufferSz,
SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
```

```
// error loading trusted peer cert
}

...
```

See Also:
wolfSSL_CTX_load_verify_buffer
wolfSSL_CTX_use_certificate_file
wolfSSL_CTX_use_PrivateKey_file
wolfSSL_CTX_use_NTRUPrivateKey_file
wolfSSL_CTX_use_certificate_chain_file
wolfSSL_CTX_trust_peer_cert
wolfSSL_CTX_Unload_trust_peers
wolfSSL_use_certificate_file
wolfSSL_use_PrivateKey_file
wolfSSL_use_certificate_chain_file

wolfSSL_CTX_Unload_trust_peers

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_Unload_trust_peers(WOLFSSL_CTX* ctx);

Description:
This function is used to unload all previously loaded trusted peer certificates.

Feature is enabled by defining the macro WOLFSSL_TRUST_PEER_CERT.

Return Values:
If successful the call will return **SSL_SUCCESS**.

BAD_FUNC_ARG will be returned if ctx is NULL.

SSL_BAD_FILE will be returned if the file doesn't exist, can't be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx;

...

ret = wolfSSL_CTX_Unload_trust_peers(ctx);
if (ret != SSL_SUCCESS) {
// error unloading trusted peer certs
}

...
```

See Also:

wolfSSL_CTX_trust_peer_buffer
wolfSSL_CTX_trust_peer_cert

wolfSSL_CTX_allow_anon_cipher

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_allow_anon_cipher(WOLFSSL_CTX* ctx);

Description:

This function enables the **havAnon** member of the CTX structure if HAVE_ANON is defined during compilation.

Return Values:

SSL_SUCCESS - returned if the function executed successfully and the **haveAnnon** member of the CTX is set to 1.

SSL_FAILURE - returned if the CTX structure was NULL.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
...
#ifdef HAVE_ANON
      if(cipherList == NULL){
        wolfSSL_CTX_allow_anon_cipher(ctx);
        if(wolfSSL_CTX_set_cipher_list(ctx, "ADH_AES128_SHA") !=
SSL_SUCCESS){
            /*failure case*/
        }
      }
#endif
```

See Also:

wolfSSL_CTX_memrestore_cert_cache

Synopsis:

#include <wolfssl/ssl.h>

void wolfSSL_CTX_memrestore_cert_cache(WOLFSSL* ssl);

Description:

This function restores the certificate cache from memory.

Return Values:

SSL_SUCCESS - returned if the function and subroutines executed without an error.

BAD_FUNC_ARG - returned if the **ctx** or **mem** parameters are NULL or if the **sz** parameter is less than or equal to zero.

BUFFER_E - returned if the cert cache memory buffer is too small.

CACHE_MATCH_ERROR - returned if there was a cert cache header mismatch.

BAD_MUTEX_E - returned if the lock mutex on failed.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

mem - a void pointer with a value that will be restored to the certificate cache.

sz - an int type that represents the size of the **mem** parameter.

Example:

```
WOLFSSL_CTX* ctx = WOLFSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = WOLFSSL_new(ctx);
void* mem;
int sz = (*int) sizeof(mem);
…
if(wolfSSL_CTX_memrestore_cert_cache(ssl->ctx, mem, sz)){
      /*The success case*/
}
```

See Also:
CM_MemRestoreCertCache

wolfSSL_CTX_SetMinVersion

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_SetMinVersion(WOLFSSL_CTX* ctx, int version);

Description:
This function sets the minimum downgrade version allowed. Applicable only when the connection allows downgrade using (wolfSSLv23_client_method or wolfSSLv23_server_method).

Return Values:
SSL_SUCCESS - returned if the function returned without error and the minimum version is set.

BAD_FUNC_ARG - returned if the WOLFSSL_CTX structure was NULL or if the minimum version is not supported.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

version - an integer representation of the version to be set as the minimum: WOLFSSL_SSLV3 = 0, WOLFSSL_TLSV1 = 1, WOLFSSL_TLSV1_1 = 2 or WOLFSSL_TLSV1_2 = 3.

Example:

```
WOLFSSL_CTX* ctx = WOLFSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = WOLFSSL_new(ctx);
int version;  /*macro representation */
…
if(wolfSSL_CTX_SetMinVersion(ssl->ctx, version) != SSL_SUCCESS){
      /*Failed to set min version*/
}
```

See Also:
SetMinVersionHelper

17.4 Callbacks

The functions in this section have to do with callbacks which the application is able to set in relation to wolfSSL.

wolfSSL_SetIOReadCtx

Synopsis:
void wolfSSL_SetIOReadCtx(WOLFSSL* ssl, void *rctx);

Description:
This function registers a context for the SSL session's receive callback function. By default, wolfSSL sets the file descriptor passed to wolfSSL_set_fd() as the context when wolfSSL is using the system's TCP library. If you've registered your own receive callback you may want to set a specific context for the session. For example, if you're using memory buffers the context may be a pointer to a structure describing where and how to access the memory buffers.

Return Values:
No return values are used for this function.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

rctx - pointer to the context to be registered with the SSL session's (**ssl**) receive callback function.

Example:

```
int sockfd;
WOLFSSL* ssl = 0;
...
/*Manually setting the socket fd as the receive CTX, for example*/
wolfSSL_SetIOReadCtx(ssl, &sockfd);
...
```

See Also:
wolfSSL_SetIORecv
wolfSSL_SetIOSend
wolfSSL_SetIOWriteCtx

wolfSSL_SetIOWriteCtx

Synopsis:
void wolfSSL_SetIOWriteCtx(WOLFSSL* ssl, void *wctx);

Description:

This function registers a context for the SSL session's send callback function. By default, wolfSSL sets the file descriptor passed to wolfSSL_set_fd() as the context when wolfSSL is using the system's TCP library. If you've registered your own send callback you may want to set a specific context for the session. For example, if you're using memory buffers the context may be a pointer to a structure describing where and how to access the memory buffers.

Return Values:

No return values are used for this function.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

wctx - pointer to the context to be registered with the SSL session's (**ssl**) send callback function.

Example:

```
int sockfd;
WOLFSSL* ssl = 0;
...
/*Manually setting the socket fd as the send CTX, for example*/
wolfSSL_SetIOSendCtx(ssl, &sockfd);
...
```

See Also:

wolfSSL_SetIORecv
wolfSSL_SetIOSend
wolfSSL_SetIOReadCtx

wolfSSL_SetIOReadFlags

Synopsis:

void wolfSSL_SetIOReadFlags(WOLFSSL* ssl, int flags);

Description:

This function sets the flags for the receive callback to use for the given SSL session. The receive callback could be either the default wolfSSL EmbedReceive callback, or a custom callback specified by the user (see wolfSSL_SetIORecv). The default flag value is set internally by wolfSSL to the value of 0.

The default wolfSSL receive callback uses the recv() function to receive data from the socket. From the recv() man page:

“The flags argument to a recv() function is formed by or'ing one or more of the values:

MSG_OOB	process out-of-band data
MSG_PEEK	peek at incoming message
MSG_WAITALL	wait for full request or error

The MSG_OOB flag requests receipt of out-of-band data that would not be received in the normal data stream. Some protocols place expedited data at the head of the normal data queue, and thus this flag cannot be used with such protocols. The MSG_PEEK flag causes the receive operation to return data from the beginning of the receive queue without removing that data from the queue. Thus, a subsequent receive call will return the same data. The MSG_WAITALL flag requests that the operation block until the full request is satisfied. However, the call may still return less data than requested if a signal is caught, an error or disconnect occurs, or the next data to be received is of a different type than that returned.”

Return Values:

No return values are used for this function.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

flags - value of the I/O read flags for the specified SSL session (**ssl**).

Example:

```
WOLFSSL* ssl = 0;
...
/*Manually setting recv flags to 0*/
wolfSSL_SetIOReadFlags(ssl, 0);
...
```

See Also:

wolfSSL_SetIORecv
wolfSSL_SetIOSend
wolfSSL_SetIOReadCtx

wolfSSL_SetIOWriteFlags

Synopsis:
void wolfSSL_SetIOWriteFlags(WOLFSSL* ssl, int flags);

Description:
This function sets the flags for the send callback to use for the given SSL session. The send callback could be either the default wolfSSL EmbedSend callback, or a custom callback specified by the user (see wolfSSL_SetIOSend). The default flag value is set internally by wolfSSL to the value of 0.

The default wolfSSL send callback uses the send() function to send data from the socket. From the send() man page:

“The flags parameter may include one or more of the following:

```
#define MSG_OOB        0x1  /* process out-of-band data */
#define MSG_DONTROUTE  0x4  /* bypass routing, use direct interface */
```

The flag MSG_OOB is used to send ``out-of-band'' data on sockets that support this notion (e.g.  SOCK_STREAM); the underlying protocol must also support ``out-of-band'' data. MSG_DONTROUTE is usually used only by diagnostic or routing programs.”

Return Values:
No return values are used for this function.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

flags - value of the I/O send flags for the specified SSL session (**ssl**).

Example:

```
WOLFSSL* ssl = 0;
...
```

```
/*Manually setting send flags to 0*/
wolfSSL_SetIOSendFlags(ssl, 0);
...
```

See Also:
wolfSSL_SetIORecv
wolfSSL_SetIOSend
wolfSSL_SetIOReadCtx

wolfSSL_SetIORecv

Synopsis:
void wolfSSL_SetIORecv(WOLFSSL_CTX* ctx, CallbackIORecv CBIORecv);

typedef int (*CallbackIORecv)(WOLFSSL* ssl, char* buf, int sz, void* ctx);

Description:
This function registers a receive callback for wolfSSL to get input data. By default, wolfSSL uses EmbedReceive() as the callback which uses the system's TCP recv() function. The user can register a function to get input from memory, some other network module, or from anywhere. Please see the EmbedReceive() function in **src/io.c** as a guide for how the function should work and for error codes. In particular, **IO_ERR_WANT_READ** should be returned for non blocking receive when no data is ready.

Return Values:
No return values are used for this function.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

callback - function to be registered as the receive callback for the wolfSSL context, **ctx**. The signature of this function must follow that as shown above in the Synopsis section.

Example:

```
WOLFSSL_CTX* ctx = 0;

/*Receive callback prototype*/
int MyEmbedReceive(WOLFSSL* ssl, char* buf, int sz, void* ctx);
```

```
/*Register the custom receive callback with wolfSSL*/
wolfSSL_SetIORecv(ctx, MyEmbedReceive);

int MyEmbedReceive(WOLFSSL* ssl, char* buf, int sz, void* ctx)
{
      /*custom EmbedReceive function*/
}
```

See Also:
wolfSSL_SetIOSend
wolfSSL_SetIOReadCtx
wolfSSL_SetIOWriteCtx

wolfSSL_SetIOSend

Synopsis:
void wolfSSL_SetIOSend(WOLFSSL_CTX* ctx, CallbackIOSend CBIOSend);

typedef int (*CallbackIOSend)(WOLFSSL* ssl, char* buf, int sz, void* ctx);

Description:
This function registers a send callback for wolfSSL to write output data. By default, wolfSSL uses EmbedSend() as the callback which uses the system's TCP send() function. The user can register a function to send output to memory, some other network module, or to anywhere. Please see the EmbedSend() function in **src/io.c** as a guide for how the function should work and for error codes. In particular, **IO_ERR_WANT_WRITE** should be returned for non blocking send when the action cannot be taken yet.

Return Values:
No return values are used for this function.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

callback - function to be registered as the send callback for the wolfSSL context, **ctx**. The signature of this function must follow that as shown above in the Synopsis section.

Example:

```
WOLFSSL_CTX* ctx = 0;

/*Receive callback prototype*/
int MyEmbedSend(WOLFSSL* ssl, char* buf, int sz, void* ctx);

/*Register the custom receive callback with wolfSSL*/
wolfSSL_SetIOSend(ctx, MyEmbedSend);

int MyEmbedSend(WOLFSSL* ssl, char* buf, int sz, void* ctx)
{
      /*custom EmbedSend function*/
}
```

See Also:
wolfSSL_SetIORecv
wolfSSL_SetIOReadCtx
wolfSSL_SetIOWriteCtx

wolfSSL_CTX_set_TicketEncCb

Synopsis:
#include <wolfssl/ssl.h>

```
typedef int (*SessionTicketEncCb)(WOLFSSL*,
                        unsigned char key_name[WOLFSSL_TICKET_NAME_SZ],
                        unsigned char iv[WOLFSSL_TICKET_IV_SZ],
                        unsigned char mac[WOLFSSL_TICKET_MAC_SZ],
                        int enc, unsigned char* ticket, int inLen, int* outLen, void* userCtx);

int wolfSSL_CTX_set_TicketEncCb(WOLFSSL_CTX* ctx, SessionTicketEncCb);
```

Description:
This function sets the session ticket key encrypt callback function for a server to support session tickets as specified in RFC 5077.

Return Values:

SSL_SUCCESS will be returned upon successfully setting the session.

BAD_FUNC_ARG will be returned on failure. This is caused by passing invalid arguments to the function.

Parameters:

ctx - pointer to the WOLFSSL_CTX object, created with wolfSSL_CTX_new().

cb - user callback function to encrypt/decrypt session tickets

Callback Parameters:

ssl - pointer to the WOLFSSL object, created with wolfSSL_new()

key_name - unique key name for this ticket context, should be randomly generated

iv - unique IV for this ticket, up to 128 bits, should be randomly generated

mac - up to 256 bit mac for this ticket

enc - if this encrypt parameter is true the user should fill in key_name, iv, mac, and encrypt the ticket in-place of length inLen and set the resulting output length in *outLen. Returning WOLFSSL_TICKET_RET_OK tells wolfSSL that the encryption was successful. If this encrypt parameter is false, the user should perform a decrypt of the ticket in-place of length inLen using key_name, iv, and mac. The resulting decrypt length should be set in *outLen. Returning WOLFSSL_TICKET_RET_OK tells wolfSSL to proceed using the decrypted ticket. Returning WOLFSSL_TICKET_RET_CREATE tells wolfSSL to use the decrypted ticket but also to generate a new one to send to the client, helpful if recently rolled keys and don't want to force a full handshake. Returning WOLFSSL_TICKET_RET_REJECT tells wolfSSL to reject this ticket, perform a full handshake, and create a new standard session ID for normal session resumption. Returning WOLFSSL_TICKET_RET_FATAL tells wolfSSL to end the connection attempt with a fatal error.

ticket - the input/output buffer for the encrypted ticket. See the enc parameter

inLen - the input length of the ticket parameter

outLen - the resulting output length of the ticket parameter. When entering the callback outLen will indicate the maximum size available in the ticket buffer.

userCtx - the user context set with wolfSSL_CTX_set_TicketEncCtx()

Example:

```
See wolfssl/test.h myTicketEncCb() used by the example server and example
echoserver.
```

See Also:
wolfSSL_CTX_set_TicketHint
wolfSSL_CTX_set_TicketEncCtx

wolfSSL_CTX_set_TicketEncCb

Synopsis:
#include <wolfssl/ssl.h>

```
typedef int (*SessionTicketEncCb)(WOLFSSL*,
                        unsigned char key_name[WOLFSSL_TICKET_NAME_SZ],
                        unsigned char iv[WOLFSSL_TICKET_IV_SZ],
                        unsigned char mac[WOLFSSL_TICKET_MAC_SZ],
                        int enc, unsigned char* ticket, int inLen, int* outLen, void* userCtx);

int wolfSSL_CTX_set_TicketEncCb(WOLFSSL_CTX* ctx, SessionTicketEncCb);
```

Description:
This function sets the session ticket key encrypt callback function for a server to support session tickets as specified in RFC 5077.

Return Values:

SSL_SUCCESS will be returned upon successfully setting the session.

BAD_FUNC_ARG will be returned on failure. This is caused by passing invalid arguments to the function.

Parameters:

ctx - pointer to the WOLFSSL_CTX object, created with wolfSSL_CTX_new().

cb - user callback function to encrypt/decrypt session tickets

Callback Parameters:

ssl - pointer to the WOLFSSL object, created with wolfSSL_new()

key_name - unique key name for this ticket context, should be randomly generated

iv - unique IV for this ticket, up to 128 bits, should be randomly generated

mac - up to 256 bit mac for this ticket

enc - if this encrypt parameter is true the user should fill in key_name, iv, mac, and encrypt the ticket in-place of length inLen and set the resulting output length in *outLen. Returning WOLFSSL_TICKET_RET_OK tells wolfSSL that the encryption was successful. If this encrypt parameter is false, the user should perform a decrypt of the ticket in-place of length inLen using key_name, iv, and mac. The resulting decrypt length should be set in *outLen. Returning WOLFSSL_TICKET_RET_OK tells wolfSSL to proceed using the decrypted ticket. Returning WOLFSSL_TICKET_RET_CREATE tells wolfSSL to use the decrypted ticket but also to generate a new one to send to the client, helpful if recently rolled keys and don't want to force a full handshake. Returning WOLFSSL_TICKET_RET_REJECT tells wolfSSL to reject this ticket, perform a full handshake, and create a new standard session ID for normal session resumption. Returning WOLFSSL_TICKET_RET_FATAL tells wolfSSL to end the connection attempt with a fatal error.

ticket - the input/output buffer for the encrypted ticket. See the enc parameter

inLen - the input length of the ticket parameter

outLen - the resulting output length of the ticket parameter. When entering the callback outLen will indicate the maximum size available in the ticket buffer.

userCtx - the user context set with wolfSSL_CTX_set_TicketEncCtx()

Example:

```
See wolfssl/test.h myTicketEncCb() used by the example server and example
echoserver.
```

See Also:

wolfSSL_CTX_set_TicketHint
wolfSSL_CTX_set_TicketEncCtx

wolfSSL_CTX_set_TicketHint

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_set_TicketHint(WOLFSSL_CTX* ctx, int hint);

Description:
This function sets the session ticket hint relayed to the client. For server side use.

Return Values:

SSL_SUCCESS will be returned upon successfully setting the session.

BAD_FUNC_ARG will be returned on failure. This is caused by passing invalid arguments to the function.

Parameters:

ctx - pointer to the WOLFSSL_CTX object, created with wolfSSL_CTX_new().

hint - number of seconds the ticket might be valid for. Hint to client.

See Also:
wolfSSL_CTX_set_TicketEncCb()

wolfSSL_CTX_set_TicketEncCtx

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_set_TicketEncCtx(WOLFSSL_CTX* ctx, void* userCtx);

Description:
This function sets the session ticket encrypt user context for the callback. For server

side use.

Return Values:

SSL_SUCCESS will be returned upon successfully setting the session.

BAD_FUNC_ARG will be returned on failure. This is caused by passing invalid arguments to the function.

Parameters:

ctx - pointer to the WOLFSSL_CTX object, created with wolfSSL_CTX_new().

userCtx - the user context for the callback

See Also:
wolfSSL_CTX_set_TicketEncCb()

wolfSSL_CTX_SetCACb

Synopsis:
void wolfSSL_CTX_SetCACb(WOLFSSL_CTX* ctx, CallbackCACache cb);

typedef void (*CallbackCACache)(unsigned char* der, int sz, int type);

Description:
This function registers a callback with the SSL context (WOLFSSL_CTX) to be called when a new CA certificate is loaded into wolfSSL. The callback is given a buffer with the DER-encoded certificate.

Return Values:
This function has no return value.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

callback - function to be registered as the CA callback for the wolfSSL context, **ctx**. The signature of this function must follow that as shown above in the Synopsis section.

Example:

```
WOLFSSL_CTX* ctx = 0;

/*CA callback prototype*/
int MyCACallback(unsigned char *der, int sz, int type);

/*Register the custom CA callback with the SSL context*/
wolfSSL_CTX_SetCACb(ctx, MyCACallback);

int MyCACallback(unsigned char* der, int sz, int type)
{
      /* custom CA callback function, DER-encoded cert
        located in "der" of size "sz" with type "type" */
}
```

See Also:
wolfSSL_CTX_load_verify_locations

wolfSSL_connect_ex

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_connect_ex(WOLFSSL* ssl, HandShakeCallBack hsCb,
                                   TimeoutCallBack toCb, Timeval timeout);

typedef int (*HandShakeCallBack)(HandShakeInfo*);
typedef int (*TimeoutCallBack)(TimeoutInfo*);

typedef struct timeval Timeval;

```
typedef struct handShakeInfo_st {
      char   cipherName[MAX_CIPHERNAME_SZ + 1];  /* negotiated
                                                   name */

      char
packetNames[MAX_PACKETS_HANDSHAKE][MAX_PACKETNAME_SZ+1];
```

```
                                                    /* SSL packet
                                                       names */
     int    numberPackets;          /* actual # of packets  */
     int    negotiationError;       /* cipher/parameter err */
} HandShakeInfo;


typedef struct timeoutInfo_st {
     char      timeoutName[MAX_TIMEOUT_NAME_SZ +1]; /*timeout
                                                         Name*/
     int        flags;                          /* for future
                                                         use*/
     int        numberPackets;                /* actual # of
                                                     packets */
     PacketInfo packets[MAX_PACKETS_HANDSHAKE]; /* list of
                                                     packets */
     Timeval    timeoutValue;                /* timer that caused
                                                     it */
} TimeoutInfo;


typedef struct packetInfo_st {
     char        packetName[MAX_PACKETNAME_SZ + 1]; /*SSL name*/
     Timeval     timestamp;                /*when it occured */
     unsigned char value[MAX_VALUE_SZ];    /*if fits, it's here*/
     unsigned char* bufferValue;        /*otherwise here(non 0)*/
     int         valueSz;               /*sz of value or buffer*/
} PacketInfo;
```

Description:

wolfSSL_connect_ex() is an extension that allows a HandShake Callback to be set. This can be useful in embedded systems for debugging support when a debugger isn’t available and sniffing is impractical. The HandShake Callback will be called whether or not a handshake error occurred. No dynamic memory is used since the maximum number of SSL packets is known. Packet names can be accessed through **packetNames[]**.

The connect extension also allows a Timeout Callback to be set along with a timeout value. This is useful if the user doesn’t want to wait for the TCP stack to timeout.

This extension can be called with either, both, or neither callbacks.

Return Values:
If successful the call will return **SSL_SUCCESS**.

GETTIME_ERROR will be returned if *gettimeofday()* encountered an error.

SETITIMER_ERROR will be returned if *setitimer()* encountered an error.

SIGACT_ERROR will be returned if *sigaction()* encountered an error.

SSL_FATAL_ERROR will be returned if the underlying *SSL_connect()* call encountered an error.

See Also:
wolfSSL_accept_ex

wolfSSL_accept_ex

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_accept_ex(WOLFSSL* ssl, HandShakeCallBack hsCb,
                                   TimeoutCallBack toCb, Timeval timeout);

typedef int (*HandShakeCallBack)(HandShakeInfo*);
typedef int (*TimeoutCallBack)(TimeoutInfo*);

typedef struct timeval Timeval;

```
typedef struct handShakeInfo_st {
     char   cipherName[MAX_CIPHERNAME_SZ + 1];  /*negotiated
                                                  name*/


     char
packetNames[MAX_PACKETS_HANDSHAKE][MAX_PACKETNAME_SZ+1];
/* SSL packet names */

     int    numberPackets;              /*actual # of packets */
     int    negotiationError;          /*cipher/parameter err */
} HandShakeInfo;


typedef struct timeoutInfo_st {
     char      timeoutName[MAX_TIMEOUT_NAME_SZ +1]; /*timeout
                                                       Name*/
     int        flags;                            /*for future
                                                       use*/
     int        numberPackets;              /*actual # of
                                                  packets */
     PacketInfo packets[MAX_PACKETS_HANDSHAKE]; /*list of
                                                  packets */
     Timeval    timeoutValue;              /* timer that
                                             caused it*/
} TimeoutInfo;


typedef struct packetInfo_st {
     char         packetName[MAX_PACKETNAME_SZ + 1];/*SSL name */
     Timeval      timestamp;                 /*when it occured */
     unsigned char value[MAX_VALUE_SZ];  /*if fits, it's here */
     unsigned char* bufferValue;        /*otherwise here(non 0)*/
     int          valueSz;              /*sz of value or buffer*/
} PacketInfo;
```

Description:
wolfSSL_accept_ex() is an extension that allows a HandShake Callback to be set. This can be useful in embedded systems for debugging support when a debugger isn't available and sniffing is impractical. The HandShake Callback will be called whether or not a handshake error occurred. No dynamic memory is used since the maximum number of SSL packets is known. Packet names can be accessed through **packetNames[]**.

The connect extension also allows a Timeout Callback to be set along with a timeout value. This is useful if the user doesn't want to wait for the TCP stack to timeout.

This extension can be called with either, both, or neither callbacks.

Return Values:
If successful the call will return **SSL_SUCCESS**.

GETTIME_ERROR will be returned if *gettimeofday()* encountered an error.

SETITIMER_ERROR will be returned if *setitimer()* encountered an error.

SIGACT_ERROR will be returned if *sigaction()* encountered an error.

SSL_FATAL_ERROR will be returned if the underlying *SSL_accept()* call encountered an error.

See Also:
wolfSSL_connect_ex

wolfSSL_SetLoggingCb

Synopsis:
#include <wolfssl/wolfcrypt/logging.h>

int wolfSSL_SetLoggingCb(wolfSSL_Logging_cb log_function);

typedef void (*wolfSSL_Logging_cb)(const int logLevel, const char *const logMessage);

Description:
This function registers a logging callback that will be used to handle the wolfSSL log message. By default, if the system supports it *fprintf()* to **stderr** is used but by using this function anything can be done by the user.

Return Values:
If successful this function will return 0.

BAD_FUNC_ARG is the error that will be returned if a function pointer is not provided.

Parameters:

log_function - function to register as a logging callback. Function signature must follow the above prototype.

Example:

```
int ret = 0;
```

```
/*Logging callback prototype*/
void MyLoggingCallback(const int logLevel, const char* const logMessage);

/*Register the custom logging callback with wolfSSL*/
ret = wolfSSL_SetLoggingCb(MyLoggingCallback);
if (ret != 0) {
      /*failed to set logging callback*/
}

void MyLoggingCallback(const int logLevel, const char* const logMessage)
{
      /*custom logging function*/
}
```

See Also:
wolfSSL_Debugging_ON
wolfSSL_Debugging_OFF

wolfSSL_SetTlsHmacInner

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_SetTlsHmacInner(WOLFSSL* ssl, byte* inner, word32 sz,
int content, int verify);

Description:
Allows caller to set the Hmac Inner vector for message sending/receiving. The result is written to **inner** which should be at least wolfSSL_GetHmacSize() bytes. The size of the message is specified by **sz**, **content** is the type of message, and **verify** specifies whether this is a verification of a peer message. Valid for cipher types excluding **WOLFSSL_AEAD_TYPE**.

Return Values:
If successful the call will return 1.

BAD_FUNC_ARG will be returned for an error state.

See Also:
wolfSSL_GetBulkCipher()
wolfSSL_GetHmacType()

wolfSSL_CTX_SetMacEncryptCb

Synopsis:

```
#include <wolfssl/ssl.h>

void wolfSSL_CTX_SetMacEncryptCb(WOLFSSL_CTX*, CallbackMacEncrypt);

typedef int (*CallbackMacEncrypt)(WOLFSSL* ssl, unsigned char* macOut,
      const unsigned char* macIn, unsigned int macInSz, int macContent,
      int macVerify, unsigned char* encOut, const unsigned char* encIn,
      unsigned int encSz, void* ctx);
```

Description:

Allows caller to set the Atomic User Record Processing Mac/Encrypt Callback. The callback should return 0 for success or < 0 for an error. The **ssl** and **ctx** pointers are available for the user's convenience. **macOut** is the output buffer where the result of the mac should be stored. **macIn** is the mac input buffer and **macInSz** notes the size of the buffer. **macContent** and **macVerify** are needed for wolfSSL_SetTlsHmacInner() and be passed along as is. **encOut** is the output buffer where the result on the encryption should be stored. **encIn** is the input buffer to encrypt while **encSz** is the size of the input. An example callback can be found wolfssl/test.h myMacEncryptCb().

Return Values:

NA

See Also:

wolfSSL_SetMacEncryptCtx()
wolfSSL_GetMacEncryptCtx()

wolfSSL_SetMacEncryptCtx

Synopsis:

```
#include <wolfssl/ssl.h>

void wolfSSL_SetMacEncryptCtx(WOLFSSL*, void* ctx);
```

Description:

Allows caller to set the Atomic User Record Processing Mac/Encrypt Callback Context to **ctx**.

Return Values:
NA

See Also:
wolfSSL_CTX_SetMacEncryptCb()
wolfSSL_GetMacEncryptCtx()

wolfSSL_GetMacEncryptCtx

Synopsis:
#include <wolfssl/ssl.h>

void* wolfSSL_GetMacEncryptCtx(WOLFSSL*);

Description:
Allows caller to retrieve the Atomic User Record Processing Mac/Encrypt Callback Context previously stored with wolfSSL_SetMacEncryptCtx().

Return Values:
If successful the call will return a valid pointer to the context.

NULL will be returned for a blank context.

See Also:
wolfSSL_CTX_SetMacEncryptCb()
wolfSSL_SetMacEncryptCtx()

wolfSSL_CTX_SetDecryptVerifyCb

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_CTX_SetDecryptVerifyCb(WOLFSSL_CTX*, CallbackDecryptVerify);

```
typedef int (*CallbackDecryptVerify)(WOLFSSL* ssl,
        unsigned char* decOut, const unsigned char* decIn,
        unsigned int decSz, int content, int verify, unsigned int* padSz,
        void* ctx);
```

Description:
Allows caller to set the Atomic User Record Processing Decrypt/Verify Callback. The callback should return 0 for success or < 0 for an error. The **ssl** and **ctx** pointers are available for the user's convenience. **decOut** is the output buffer where the result of the decryption should be stored. **decIn** is the encrypted input buffer and **decInSz** notes the size of the buffer. **content** and **verify** are needed for wolfSSL_SetTlsHmacInner() and be passed along as is. **padSz** is an output variable that should be set with the total value of the padding. That is, the mac size plus any padding and pad bytes. An example callback can be found wolfssl/test.h myDecryptVerifyCb().

Return Values:
NA

See Also:
wolfSSL_SetMacEncryptCtx()
wolfSSL_GetMacEncryptCtx()

wolfSSL_SetDecryptVerifyCtx

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_SetDecryptVerifyCtx(WOLFSSL*, void* ctx);

Description:
Allows caller to set the Atomic User Record Processing Decrypt/Verify Callback Context to **ctx**.

Return Values:
NA

See Also:
wolfSSL_CTX_SetDecryptVerifyCb()

wolfSSL_GetDecryptVerifyCtx()

wolfSSL_GetDecryptVerifyCtx

Synopsis:
#include <wolfssl/ssl.h>

void* wolfSSL_GetDecryptVerifyCtx(WOLFSSL*);

Description:
Allows caller to retrieve the Atomic User Record Processing Decrypt/Verify Callback Context previously stored with wolfSSL_SetDecryptVerifyCtx().

Return Values:
If successful the call will return a valid pointer to the context.

NULL will be returned for a blank context.

See Also:
wolfSSL_CTX_SetDecryptVerifyCb()
wolfSSL_SetDecryptVerifyCtx()

wolfSSL_CTX_SetEccSignCb

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_CTX_SetEccSignCb(WOLFSSL_CTX*, CallbackEccSign);

```
typedef int (*CallbackEccSign)(WOLFSSL* ssl,
      const unsigned char* in, unsigned int inSz,
      unsigned char* out, unsigned int* outSz,
      const unsigned char* keyDer, unsigned int keySz,
      void* ctx);
```

Description:
Allows caller to set the Public Key Callback for ECC Signing. The callback should return 0 for success or < 0 for an error. The **ssl** and **ctx** pointers are available for the

user’s convenience. **in** is the input buffer to sign while **inSz** denotes the length of the input. **out** is the output buffer where the result of the signature should be stored. **outSz** is an input/output variable that specifies the size of the output buffer upon invocation and the actual size of the signature should be stored there before returning. **keyDer** is the ECC Private key in ASN1 format and **keySz** is the length of the key in bytes. An example callback can be found wolfssl/test.h myEccSign().

Return Values:
NA

See Also:
wolfSSL_SetEccSignCtx()
wolfSSL_GetEccSignCtx()

wolfSSL_SetEccSignCtx

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_SetEccSignCtx(WOLFSSL*, void* ctx);

Description:
Allows caller to set the Public Key Ecc Signing Callback Context to **ctx**.

Return Values:
NA

See Also:
wolfSSL_CTX_SetEccSignCb()
wolfSSL_GetEccSignCtx()

wolfSSL_GetEccSignCtx

Synopsis:
#include <wolfssl/ssl.h>

void* wolfSSL_GetEccSignCtx(WOLFSSL*);

Description:
Allows caller to retrieve the Public Key Ecc Signing Callback Context previously stored with wolfSSL_SetEccSignCtx().

Return Values:
If successful the call will return a valid pointer to the context.

NULL will be returned for a blank context.

See Also:
wolfSSL_CTX_SetEccSignCb()
wolfSSL_SetEccSignCtx()

wolfSSL_CTX_SetEccVerifyCb

Synopsis:
```
#include <wolfssl/ssl.h>

void wolfSSL_CTX_SetEccVerifyCb(WOLFSSL_CTX*, CallbackEccVerify);

typedef int (*CallbackEccVerify)(WOLFSSL* ssl,
        const unsigned char* sig, unsigned int sigSz,
        const unsigned char* hash, unsigned int hashSz,
        const unsigned char* keyDer, unsigned int keySz,
        int* result, void* ctx);
```

Description:
Allows caller to set the Public Key Callback for ECC Verification. The callback should return 0 for success or < 0 for an error. The **ssl** and **ctx** pointers are available for the user's convenience. **sig** is the signature to verify and **sigSz** denotes the length of the signature. **hash** is an input buffer containing the digest of the message and **hashSz** denotes the length in bytes of the hash. **result** is an output variable where the result of the verification should be stored, **1** for success and **0** for failure. **keyDer** is the ECC Private key in ASN1 format and **keySz** is the length of the key in bytes. An example callback can be found wolfssl/test.h myEccVerify().

Return Values:
NA

See Also:
wolfSSL_SetEccVerifyCtx()
wolfSSL_GetEccVerifyCtx()

wolfSSL_SetEccVerifyCtx

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_SetEccVerifyCtx(WOLFSSL*, void* ctx);

Description:
Allows caller to set the Public Key Ecc Verification Callback Context to **ctx**.

Return Values:
NA

See Also:
wolfSSL_CTX_SetEccVerifyCb()
wolfSSL_GetEccVerifyCtx()

wolfSSL_GetEccVerifyCtx

Synopsis:
#include <wolfssl/ssl.h>

void* wolfSSL_GetEccVerifyCtx(WOLFSSL*);

Description:
Allows caller to retrieve the Public Key Ecc Verification Callback Context previously stored with wolfSSL_SetEccVerifyCtx().

Return Values:
If successful the call will return a valid pointer to the context.

NULL will be returned for a blank context.

See Also:
wolfSSL_CTX_SetEccVerifyCb()
wolfSSL_SetEccVerifyCtx()

wolfSSL_CTX_SetRsaSignCb

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_CTX_SetEccRsaCb(WOLFSSL_CTX*, CallbackRsaSign);

```
typedef int (*CallbackRsaSign)(WOLFSSL* ssl,
        const unsigned char* in, unsigned int inSz,
        unsigned char* out, unsigned int* outSz,
        const unsigned char* keyDer, unsigned int keySz,
        void* ctx);
```

Description:
Allows caller to set the Public Key Callback for RSA Signing. The callback should return 0 for success or < 0 for an error. The **ssl** and **ctx** pointers are available for the user's convenience. **in** is the input buffer to sign while **inSz** denotes the length of the input. **out** is the output buffer where the result of the signature should be stored. **outSz** is an input/output variable that specifies the size of the output buffer upon invocation and the actual size of the signature should be stored there before returning. **keyDer** is the RSA Private key in ASN1 format and **keySz** is the length of the key in bytes. An example callback can be found wolfssl/test.h myRsaSign().

Return Values:
NA

See Also:
wolfSSL_SetRsaSignCtx()
wolfSSL_GetRsaSignCtx()

wolfSSL_SetRsaSignCtx

Synopsis:

```
#include <wolfssl/ssl.h>

void wolfSSL_SetRsaSignCtx(WOLFSSL*, void* ctx);
```

Description:

Allows caller to set the Public Key RSA Signing Callback Context to **ctx**.

Return Values:

NA

See Also:

wolfSSL_CTX_SetRsaSignCb()
wolfSSL_GetRsaSignCtx()

wolfSSL_GetRsaSignCtx

Synopsis:

```
#include <wolfssl/ssl.h>

void* wolfSSL_GetRsaSignCtx(WOLFSSL*);
```

Description:

Allows caller to retrieve the Public Key RSA Signing Callback Context previously stored with wolfSSL_SetRsaSignCtx().

Return Values:

If successful the call will return a valid pointer to the context.

NULL will be returned for a blank context.

See Also:

wolfSSL_CTX_SetRsaSignCb()
wolfSSL_SetRsaSignCtx()

wolfSSL_CTX_SetRsaVerifyCb

Synopsis:

#include <wolfssl/ssl.h>

void wolfSSL_CTX_SetRsaVerifyCb(WOLFSSL_CTX*, CallbackRsaVerify);

```
typedef int (*CallbackRsaVerify)(WOLFSSL* ssl,
        unsigned char* sig, unsigned int sigSz,
        unsigned char** out,
        const unsigned char* keyDer, unsigned int keySz,
        void* ctx);
```

Description:
Allows caller to set the Public Key Callback for RSA Verification. The callback should return the number of plaintext bytes for success or < 0 for an error. The **ssl** and **ctx** pointers are available for the user's convenience. **sig** is the signature to verify and **sigSz** denotes the length of the signature. **out** should be set to the beginning of the verification buffer after the decryption process and any padding. **keyDer** is the RSA Public key in ASN1 format and **keySz** is the length of the key in bytes. An example callback can be found wolfssl/test.h myRsaVerify().

Return Values:
NA

Also:
wolfSSL_SetRsaVerifyCtx()
wolfSSL_GetRsaVerifyCtx()

wolfSSL_SetRsaVerifyCtx

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_SetRsaVerifyCtx(WOLFSSL*, void* ctx);

Description:
Allows caller to set the Public Key RSA Verification Callback Context to **ctx**.

Return Values:
NA

See Also:
wolfSSL_CTX_SetRsaVerifyCb()
wolfSSL_GetRsaVerifyCtx()

wolfSSL_GetRsaVerifyCtx

Synopsis:
```
#include <wolfssl/ssl.h>

void* wolfSSL_GetRsaVerifyCtx(WOLFSSL*);
```

Description:
Allows caller to retrieve the Public Key RSA Verification Callback Context previously stored with wolfSSL_SetRsaVerifyCtx().

Return Values:
If successful the call will return a valid pointer to the context.

NULL will be returned for a blank context.

See Also:
wolfSSL_CTX_SetRsaVerifyCb()
wolfSSL_SetRsaVerifyCtx()

wolfSSL_CTX_SetRsaEncCb

Synopsis:
```
#include <wolfssl/ssl.h>

void wolfSSL_CTX_SetRsaEncCb(WOLFSSL_CTX*, CallbackRsaEnc);

typedef int (*CallbackRsaEnc)(WOLFSSL* ssl,
      const unsigned char* in, unsigned int inSz,
      unsigned char* out, unsigned int* outSz,
      const unsigned char* keyDer, unsigned int keySz,
      void* ctx);
```

Description:
Allows caller to set the Public Key Callback for RSA Public Encrypt. The callback should return 0 for success or < 0 for an error. The **ssl** and **ctx** pointers are available for the user's convenience. **in** is the input buffer to encrypt while **inSz** denotes the length of the input. **out** is the output buffer where the result of the encryption should be stored. **outSz** is an input/output variable that specifies the size of the output buffer upon invocation and the actual size of the encryption should be stored there before returning. **keyDer** is the RSA Public key in ASN1 format and **keySz** is the length of the key in bytes. An example callback can be found wolfssl/test.h myRsaEnc().

Return Values:
NA

See Also:
wolfSSL_SetRsaEncCtx()
wolfSSL_GetRsaEncCtx()

wolfSSL_SetRsaEncCtx

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_SetRsaEncCtx(WOLFSSL*, void* ctx);

Description:
Allows caller to set the Public Key RSA Public Encrypt Callback Context to **ctx**.

Return Values:
NA

See Also:
wolfSSL_CTX_SetRsaEncCb()
wolfSSL_GetRsaEncCtx()

wolfSSL_GetRsaEncCtx

Synopsis:
#include <wolfssl/ssl.h>

void* wolfSSL_GetRsaEncCtx(WOLFSSL*);

Description:
Allows caller to retrieve the Public Key RSA Public Encrypt Callback Context previously stored with wolfSSL_SetRsaEncCtx().

Return Values:
If successful the call will return a valid pointer to the context.

NULL will be returned for a blank context.

See Also:
wolfSSL_CTX_SetRsaEncCb()
wolfSSL_SetRsaEncCtx()

wolfSSL_CTX_SetRsaDecCb

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_CTX_SetRsaDecCb(WOLFSSL_CTX*, CallbackRsaDec);

```
typedef int (*CallbackRsaDec)(WOLFSSL* ssl,
       unsigned char* in, unsigned int inSz,
       unsigned char** out,
       const unsigned char* keyDer, unsigned int keySz,
       void* ctx);
```

Description:
Allows caller to set the Public Key Callback for RSA Private Decrypt. The callback should return the number of plaintext bytes for success or < 0 for an error. The **ssl** and **ctx** pointers are available for the user's convenience. **in** is the input buffer to decrypt and **inSz** denotes the length of the input. **out** should be set to the beginning of the decryption buffer after the decryption process and any padding. **keyDer** is the RSA Private key in ASN1 format and **keySz** is the length of the key in bytes. An example

callback can be found wolfssl/test.h myRsaDec().

Return Values:
NA

See Also:
wolfSSL_SetRsaDecCtx()
wolfSSL_GetRsaDecCtx()

wolfSSL_SetRsaDecCtx

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_SetRsaDecCtx(WOLFSSL*, void* ctx);

Description:
Allows caller to set the Public Key RSA Private Decrypt Callback Context to **ctx**.

Return Values:
NA

See Also:
wolfSSL_CTX_SetRsaDecCb()
wolfSSL_GetRsaDecCtx()

wolfSSL_GetRsaDecCtx

Synopsis:
#include <wolfssl/ssl.h>

void* wolfSSL_GetRsaDecCtx(WOLFSSL*);

Description:
Allows caller to retrieve the Public Key RSA Private Decrypt Callback Context previously stored with wolfSSL_SetRsaDecCtx().

Return Values:
If successful the call will return a valid pointer to the context.

NULL will be returned for a blank context.

See Also:
wolfSSL_CTX_SetRsaDecCb()
wolfSSL_SetRsaDecCtx()

wolfSSL_set_SessionTicket_cb

Synopsis:
#include <wolfssl/ssl.h>

```
int wolfSSL_set_SessionTicket_cb(WOLFSSL* ssl, CallbackSessionTicket cb,
                                                        void* ctx);
```

Description:
This function sets the session ticket callback. The type CallbackSessionTicket is a function pointer with the signature of:

```
        int (*CallbackSessionTicket)(WOLFSSL*, const unsigned char*, int, void*)
```

Return Values:
SSL_SUCCESS - returned if the function executed without error.

BAD_FUNC_ARG - returned if the WOLFSSL structure is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

cb - a function pointer to the type CallbackSessionTicket.

ctx - a void pointer to the session_ticket_ctx member of the WOLFSSL structure.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
…
int sessionTicketCB(WOLFSSL* ssl, const unsigned char* ticket, int ticketSz,
                        void* ctx){ … }
wolfSSL_set_SessionTicket_cb(ssl, sessionTicketCB, (void*)"initial session");
```

See Also:
wolfSSL_set_SessionTicket
CallbackSessionTicket
sessionTicketCB

wolfSSL_set_session_secret_cb

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_set_session_secret_cb(WOLFSSL* ssl, SessionSecretCb cb, void* ctx);

Description:
This function sets the session secret callback function. The SessionSecretCb type has the signature:
int (*SessionSecretCb)(WOLFSSL* ssl, void* secret, int* secretSz, void* ctx).
The **sessionSecretCb** member of the WOLFSSL struct is set to the parameter **cb**.

Return Values:
SSL_SUCCESS - returned if the execution of the function did not return an error.

SSL_FATAL_ERROR - returned if the WOLFSSL structure is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

cb - a SessionSecretCb type that is a function pointer with the above signature.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
int SessionSecretCB (WOLFSSL* ssl, void* secret, int* secretSz, void* ctx) =
SessionSecretCb;  /*Signature of SessionSecretCb*/
…
int wolfSSL_set_session_secret_cb(ssl, SessionSecretCB, (void*)ssl->ctx){
      /*Function body. */
}
```

See Also:
SessionSecretCb

wolfSSL_CTX_SetGenCookie

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_CTX_SetGenCookie(WOLFSSL_CTX* ctx, CallbackGenCookie cb);

Description:
This function sets the callback for the CBIOCookie member of the WOLFSSL_CTX structure. The CallbackGenCookie type is a function pointer and has the signature:
int (*CallbackGenCookie)(WOLFSSL* ssl, unsigned char* buf, int sz, void* ctx);

Return Values:
This function has no return value.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

cb - a CallbackGenCookie type function pointer with the signature of CallbackGenCookie.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
…
int SetGenCookieCB(WOLFSSL* ssl, unsigned char* buf, int sz, void* ctx){
      /*Callback function body. */
}
…
wolfSSL_CTX_SetGenCookie(ssl->ctx, SetGenCookieCB);
```

See Also:
CallbackGenCookie

wolfSSL_SetHsDoneCb

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_SetHsDoneCb(WOLFSSL* ssl, HandShakeDoneCb cb, void* user_ctx);

Description:
This function sets the handshake done callback. The **hsDoneCb** and **hsDoneCtx** members of the WOLFSSL structure are set in this function.

Return Values:
SSL_SUCCESS - returned if the function executed without an error. The hsDoneCb and hsDoneCtx members of the WOLFSSL struct are set.

BAD_FUNC_ARG - returned if the WOLFSSL struct is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

cb - a function pointer of type HandShakeDoneCb with the signature of the form:
    int (*HandShakeDoneCb)(WOLFSSL*, void*);

user_ctx - a void pointer to the user registered context.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
…
int myHsDoneCb(WOLFSSL* ssl, void* user_ctx){
      /*callback function */
 }
…
wolfSSL_SetHsDoneCb(ssl, myHsDoneCb, NULL);
```

See Also:

HandShakeDoneCb

wolfSSL_SetFuzzerCb

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_SetFuzzerCb(WOLFSSL* ssl, CallbackFuzzer cbf, void* fCtx);

Description:
This function sets the fuzzer callback.

Return Values:
This function has no return value.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

cbf - a CallbackFuzzer type that is a function pointer of the form:

```
int (*CallbackFuzzer)(WOLFSSL* ssl, const unsigned char* buf, int sz,
                      int type, void* fuzzCtx);
```

fCtx - a void pointer type that will be set to the fuzzerCtx member of the WOLFSSL structure.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
void* fCtx;

int callbackFuzzerCB(WOLFSSL* ssl, const unsigned char* buf, int sz,
                        int type, void* fuzzCtx){
/*function definition*/
 }
…
wolfSSL_SetFuzzerCb(ssl, callbackFuzzerCB, fCtx);
```

See Also:
CallbackFuzzer

wolfSSL_CertManagerSetCRL_Cb

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CertManagerSetCRL_Cb(WOLFSSL_CERT_MANAGER* cm,
                                                        CbMissingCRL cb);

Description:

This function sets the CRL Certificate Manager callback. If HAVE_CRL is defined and a matching CRL record is not found then the cbMissingCRL is called (set via wolfSSL_CertManagerSetCRL_Cb). This allows you to externally retrieve the CRL and load it.

Return Values:

SSL_SUCCESS - returned upon successful execution of the function and subroutines.

BAD_FUNC_ARG - returned if the WOLFSSL_CERT_MANAGER structure is NULL.

Parameters:

cm - the WOLFSSL_CERT_MANAGER structure holding the information for the certificate.

cb - a function pointer to (*CbMissingCRL) that is set to the cbMissingCRL member of the WOLFSSL_CERT_MANAGER.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
…
void cb(const char* url){
      /*Function body. */
}
…
CbMissingCRL cb = CbMissingCRL;
…
if(ctx){
      return wolfSSL_CertManagerSetCRL_Cb(ssl->ctx->cm, cb);
}
```

See Also:

CbMissingCRL
wolfSSL_SetCRL_Cb

wolfSSL_SetOCSP_Cb

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_SetOCSP_Cb(WOLFSSL* ssl, CbOCSPIO ioCb, CbOCSPRespFree
respFreeCb, void* ioCbCtx);

Description:
This function sets the OCSP callback in the WOLFSSL_CERT_MANAGER structure.

Return Values:
SSL_SUCCESS - returned if the function executes without error. The ocspIOCb, ocspRespFreeCb, and ocspIOCtx memebers of the CM are set.

BAD_FUNC_ARG - returned if the WOLFSSL or WOLFSSL_CERT_MANAGER structures are NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

ioCb - a function pointer to type CbOCSPIO.

respFreeCb - a function pointer to type CbOCSPRespFree which is the call to free the response memory.

ioCbCtx - a void pointer that will be held in the ocspIOCtx member of the CM.

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
…
int OCSPIO_CB(void* , const char*, int , unsigned char* , int,
                  unsigned char**){  /*must have this signature*/
            /*Function Body*/
}
…
```

```
void OCSPRespFree_CB(void* , unsigned char* ){ /*must have this signature*/
      /*function body*/
}
…
void* ioCbCtx;
CbOCSPRespFree CB_OCSPRespFree;

if(wolfSSL_SetOCSP_Cb(ssl, OCSPIO_CB(/*Pass args*/), CB_OCSPRespFree,
                        ioCbCtx) != SSL_SUCCESS){
      /*Callback not set */
}
```

See Also:

wolfSSL_CertManagerSetOCSP_Cb
CbOCSPIO
CbOCSPRespFree

wolfSSL_SetCRL_Cb

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_SetCRL_Cb(WOLFSSL* ssl, CbMissingCRL cb);

Description:

Sets the CRL callback in the WOLFSSL_CERT_MANAGER structure.

Return Values:

SSL_SUCCESS - returned if the function or subroutine executes without error. The cbMissingCRL member of the WOLFSSL_CERT_MANAGER is set.

BAD_FUNC_ARG - returned if the WOLFSSL or WOLFSSL_CERT_MANAGER structure is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

cb - a function pointer to CbMissingCRL.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
…
void cb(const char* url)/*required signature */
{
      /*Function body */
}
…
int crlCb = wolfSSL_SetCRL_Cb(ssl, cb);
if(crlCb != SSL_SUCCESS){
      /*The callback was not set properly */
}
```

See Also:

CbMissingCRL
wolfSSL_CertManagerSetCRL_Cb

wolfSSL_CTX_SetOCSP_Cb

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_SetOCSP_Cb(WOLFSSL_CTX* ctx, CbOCSPIO ioCb,
CbOCSPRespFree respFreeCb, void* ioCbCtx);

Description:

Sets the callback for the OCSP in the WOLFSSL_CTX structure.

Return Values:

SSL_SUCCESS - returned if the function executed successfully. The ocspIOCb, ocspRespFreeCb, and ocspIOCtx members in the CM were successfully set.

BAD_FUNC_ARG - returned if the WOLFSSL_CTX or WOLFSSL_CERT_MANAGER structure is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

ioCb - a CbOCSPIO type that is a function pointer.

respFreeCb - a CbOCSPRespFree type that is a function pointer.

ioCbCtx - a void pointer that will be held in the WOLFSSL_CERT_MANAGER.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
…
CbOCSPIO ocspIOCb;
CbOCSPRespFree ocspRespFreeCb;
…
void* ioCbCtx;

int isSetOCSP = wolfSSL_CTX_SetOCSP_Cb(ctx, ocspIOCb, ocspRespFreeCb,
ioCbCtx);

if(isSetOCSP != SSL_SUCCESS){
      /*The function did not return successfully. */
}
```

See Also:
wolfSSL_CertManagerSetOCSP_Cb
CbOCSPIO
CbOCSPRespFree

wolfSSL_CertManagerSetOCSP_Cb

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CertManagerSetOCSP_Cb(WOLFSSL_CERT_MANAGER* cm,
                CbOCSPIO ioCb, CbOCSPRespFree respFreeCb, void* ioCbCtx);

Description:
The function sets the OCSP callback in the WOLFSSL_CERT_MANAGER.

Return Values:
SSL_SUCCESS - returned on successful execution. The arguments are saved in the WOLFSSL_CERT_MANAGER structure.

BAD_FUNC_ARG - returned if the WOLFSSL_CERT_MANAGER is NULL.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure.

ioCb - a function pointer of type CbOCSPIO.

respFreeCb - a function pointer of type CbOCSPRespFree.

ioCbCtx - a void pointer variable to the I/O callback user registered context.

Example:

```
wolfSSL_SetOCSP_Cb(WOLFSSL* ssl, CbOCSPIO ioCb,
                        CbOCSPRespFree respFreeCb, void* ioCbCtx){

…
return wolfSSL_CertManagerSetOCSP_Cb(ssl->ctx->cm, ioCb, respFreeCb,
ioCbCtx);
```

See Also:
wolfSSL_CertManagerSetOCSPOverrideURL
wolfSSL_CertManagerCheckOCSP
wolfSSL_CertManagerEnableOCSPStapling
wolfSSL_ENableOCSP
wolfSSL_DisableOCSP
wolfSSL_SetOCSP_Cb

wolfSSL_set_psk_client_callback

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_set_psk_client_callback(WOLFSSL* ssl, wc_psk_client_callback cb);

Description:
Sets the PSK client side callback.

Return Values:
This function has no return value.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

cb - a function pointer to type wc_psk_client_callback.

Example:

```
WOLFSSL* ssl;
unsigned int cb(WOLFSSL*, const char*, char*)/*Header of function*
{
      /*Funciton body */
}
…
cb = wc_psk_client_callback;
if(ssl){
      wolfSSL_set_psk_client_callback(ssl, cb);
} else {
      /*could not set callback */
}
```

See Also:

wolfSSL_CTX_set_psk_client_callback
wolfSSL_CTX_set_psk_server_callback
wolfSSL_set_psk_server_callback

wolfSSL_CTX_SetCRL_Cb

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_SetCRL_Cb(WOLFSSL_CTX* ctx, CbMissingCRL cb);

Description:

This function will set the callback argument to the cbMissingCRL member of the WOLFSSL_CERT_MANAGER structure by calling wolfSSL_CertManagerSetCRL_Cb.

Return Values:

SSL_SUCCESS - returned for a successful execution. The WOLFSSL_CERT_MANAGER structure's member cbMssingCRL was successfully set to **cb.**

BAD_FUNC_ARG - returned if WOLFSSL_CTX or WOLFSSL_CERT_MANAGER are

NULL.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created with wolfSSL_CTX_new().

cb - a pointer to a callback function of type CbMissingCRL. Signature requirement:
void (*CbMissingCRL)(const char* url);

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
…
void cb(const char* url)/*Required signature*/
{
      /*Function body*/
}
…
if (wolfSSL_CTX_SetCRL_Cb(ctx, cb) != SSL_SUCCESS){
      /*Failure case, cb was not set correctly. */
}
```

See Also:
wolfSSL_CertManagerSetCRL_Cb
CbMissingCRL

wolfSSL_CTX_set_psk_server_callback

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_CTX_set_psk_server_callback(WOLFSSL_CTX* ctx,
wc_psk_server_callback cb);

Description:
This function sets the psk callback for the server side in the WOLFSSL_CTX structure.

Return Values:
This function has no return value.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

cb - a function pointer for the callback and will be stored in the WOLFSSL_CTX structure.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
…
unsigned int cb(WOLFSSL*, const char*, unsigned char*, unsigned int)
/*signature requirement*/
{
      /*Function body. */
}
…
if(ctx != NULL){
wolfSSL_CTX_set_psk_server_callback(ctx, cb);
} else {
      /*The CTX object was not properly initialized. */
}
```

See Also:

wc_psk_server_callback
wolfSSL_set_psk_client_callback
wolfSSL_set_psk_server_callback
wolfSSL_CTX_set_psk_client_callback

wolfSSL_set_psk_server_callback

Synopsis:

#include <wolfssl/ssl.h>

void wolfSSL_set_psk_server_callback(WOLFSSL* ssl, wc_psk_server_callback cb);

Description:

Sets the psk callback for the server side by setting the WOLFSSL structure **options** members.

Return Values:

This function has no return value.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

cb - a function pointer for the callback and will be stored in the WOLFSSL structure.

Example:

```
WOLFSSL_CTX* ctx;
WOLFSSL* ssl;
…
int cb(WOLFSSL*, const char*, unsigned char*, unsigned int)/*Required sig. */
{
      /*Function body. */
}
…
if(ssl != NULL && cb != NULL){
      wolfSSL_set_psk_server_callback(ssl, cb);
}
```

See Also:
wolfSSL_set_psk_client_callback
wolfSSL_set_psk_server_callback
wolfSSL_CTX_set_psk_server_callback
wolfSSL_CTX_set_psk_client_callback
wolfSSL_get_psk_identity_hint
wc_psk_server_callback
InitSuites

wolfSSL_CTX_set_psk_client_callback

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_CTX_set_psk_client_callback(WOLFSSL_CTX* ctx,
                                                                                   wc_psk_client_callback cb);

Description:
The function sets the client_psk_cb member of the WOLFSSL_CTX structure.

Return Values:

This function has no return value.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

cb - wc_psk_client_callback is a function pointer that will be stored in the WOLFSSL_CTX structure.

Example:

```
WOLFSSL_CTX* ctx = WOLFSSL_CTX_new(/*protocol def*/);
…
static INLINE unsigned int my_psk_client_cb(WOLFSSL* ssl, const char* hint,
            char* identity, unsigned int id_max_len, unsigned char* key,
            Unsigned int key_max_len){
…
wolfSSL_CTX_set_psk_client_callback(ctx, my_psk_client_cb);
```

See Also:

(*wc_psk_client_callback)
wolfSSL_set_psk_client_callback
wolfSSL_set_psk_server_callback
wolfSSL_CTX_set_psk_server_callback
wolfSSL_CTX_set_psk_client_callback

EmbedReceiveFrom

Synopsis:

#include <wolfssl/ssl.h>

int EmbedReceiveFrom(WOLFSSL* ssl, char* buf, int sz, void* ctx);

Description:

This function is the receive embedded callback.

Return Values:

This function returns the nb **bytes read** if the execution was successful.

WOLFSSL_CBIO_ERR_WANT_READ - if the connection refused or if a ‘would block’ error was thrown in the function.

WOLFSSL_CBIO_ERR_TIMEOUT - returned if the socket timed out.

WOLFSSL_CBIO_ERR_CONN_RST - returned if the connection reset.

WOLFSSL_CBIO_ERR_ISR - returned if the socket was interrupted.

WOLFSSL_CBIO_ERR_GENERAL - returned if there was a general error.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

buf - a constant char pointer to the buffer.

sz - an int type representing the size of the buffer.

ctx - a void pointer to the WOLFSSL_CTX context.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = WOLFSSL_new(ctx);
char* buf; /*Allocate / Initialize */
int sz = sizeof(buf)/sizeof(char);
(void*)ctx;
…

int nb = EmbedReceiveFrom(ssl, buf, sz, ctx);

if(nb > 0){
      /*nb is the number of bytes written and is positive*/
}
```

See Also:

TranslateReturnCode
RECVFROM_FUNCTION
Setsockopt

EmbedReceive

Synopsis:
#include <wolfssl/ssl.h>

int EmbedReceive(WOLFSSL* ssl, char* buf, int sz, void* ctx);

Description:
This function is the receive embedded callback.

Return Values:
This function returns the **number of bytes** read.

WOLFSSL_CBIO_ERR_WANT_READ - returned with a “Would block” message if the last error was SOCKET_EWOULDBLCOK or SOCKET_EAGAIN.

WOLFSSL_CBIO_ERR_TIMEOUT - returned with a “Socket timeout” message.

WOLFSSL_CBIO_ERR_CONN_RST - returned with a “Connection reset” message if the last error was SOCKET_ECONNRESET.

WOLFSSL_CBIO_ERR_ISR - returned with a “Socket interrupted” message if the last error was SOCKET_EINTR.

WOLFSSL_CBIO_ERR_WANT_READ - returned with a “Connection refused” messag if the last error was SOCKET_ECONNREFUSED.

WOLFSSL_CBIO_ERR_CONN_CLOSE - returned with a “Connection aborted” message if the last error was SOCKET_ECONNABORTED.

WOLFSSL_CBIO_ERR_GENERAL - returned with a “General error” message if the last error was not specified.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

buf - a char pointer representation of the buffer.

sz - the size of the buffer.

ctx - a void pointer to user registered context. In the default case the ctx is a socket descriptor pointer.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
char* buf; /*The buffer. */
int sz; /* Size of buf */
void* ctx;

int bytesRead = EmbedReceive(ssl, buf, sz, ctx);
if(bytesRead <= 0){
      /*There were no bytes read. Failure case. */
}
```

See Also:
wolfSSL_dtls_get_current_timeout
TranslateReturnCode
RECV_FUNCTION

EmbedSend

Synopsis:
#include <wolfssl/ssl.h>

int EmbedSend(WOLFSSL* ssl, char* buf, int sz, void* ctx);

Description:
This function is the send embedded callback.

Return Values:
This function returns the **number of bytes** sent.

WOLFSSL_CBIO_ERR_WANT_WRITE - returned with a “Would block” message if the last error was SOCKET_EWOULDBLOCK or SOCKET_EAGAIN.

WOLFSSL_CBIO_ERR_CONN_RST - returned with a “Connection reset” message if the last error was SOCKET_ECONNRESET.

WOLFSSL_CBIO_ERR_ISR - returned with a “Socket interrupted” message if the last error was SOCKET_EINTR.

WOLFSSL_CBIO_ERR_CONN_CLOSE - returned with a “Socket EPIPE” message if the last error was SOCKET_EPIPE.

WOLFSSL_CBIO_ERR_GENERAL - returned with a “General error” message if the last error was not specified.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

buf - a char pointer representing the buffer.

sz - the size of the buffer.

ctx - a void pointer to user registered context.

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
char* buf; /*buffer*/
int sz; /*size of buffer*/
void* ctx;

int dSent = EmbedSend(ssl, buf, sz, ctx);
if(dSent <= 0){
      /*No byes sent. Failure case. */
}
```

See Also:
TranslateReturnCode
SEND_FUNCTION
LastError
InitSSL_Ctx
LastError

EmbedSendTo

Synopsis:
#include <wolfssl/ssl.h>

int EmbedSendTo(WOLFSSL* ssl, char8 buf, int sz, void* ctx);

Description:
This function is the send embedded callback.

Return Values:
This function returns the **number of bytes** sent.

WOLFSSL_CBIO_ERR_WANT_WRITE - returned with a “Would Block” message if the last error was either SOCKET_EWOULDBLOCK or SOCKET_EAGAIN error.

WOLFSSL_CBIO_ERR_CONN_RST - returned with a “Connection reset” message if the last error was SOCKET_ECONNRESET.

WOLFSSL_CBIO_ERR_ISR - returned with a “Socket interrupted” message if the last error was SOCKET_EINTR.

WOLFSSL_CBIO_ERR_CONN_CLOSE - returned with a “Socket EPIPE” message if the last error was WOLFSSL_CBIO_ERR_CONN_CLOSE.

WOLFSSL_CBIO_ERR_GENERAL - returned with a “General error” message if the last error was not specified.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

buf - a char pointer representing the buffer.

sz - the size of the buffer.

ctx - a void pointer to the user registered context. The default case is a WOLFSSL_DTLS_CTX sructure.

Example:

```
WOLFSSL* ssl;
```

```
…
char* buf;
int sz; /*Size of buffer */
void* ctx;

int sEmbed = EmbedSendto(ssl, buf, sz, ctx);
if(sEmbed <= 0){
      /*No bytes sent. Failure case. */
}
```

See Also:

LastError
EmbedSend
EmbedReceive

EmbedGenerateCookie

Synopsis:

#include <wolfssl/ssl.h>

int EmbedGenerateCookie(WOLFSSL* ssl, byte* buf, int sz, void* ctx);

Description:

This function is the DTLS Generate Cookie callback.

Return Values:

This function returns the number of **bytes** copied into the buffer.

GEN_COOKIE_E - returned if the getpeername failed in EmbedGenerateCookie.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

buf - byte pointer representing the buffer. It is the destination from XMEMCPY().

sz - the size of the buffer.

ctx - a void pointer to user registered context.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
byte buffer[BUFFER_SIZE];
int sz = sizeof(buffer)/sizeof(byte);
void* ctx;
…
int ret = EmbedGenerateCookie(ssl, buffer, sz, ctx);

if(ret > 0){
      /*EmbedGenerateCookie code block for success*/
}
```

See Also:
wc_ShaHash
EmbedGenerateCookie
XMEMCPY
XMEMSET

EmbedOcspRespFree

Synopsis:
#include <wolfssl/ssl.h>

void EmbedOcspRespFree(void* ctx, byte* resp);

Description:
This function frees the response buffer.

Return Values:
This function has no return value.

Parameters:

ctx - a void pointer to heap hint.

resp - a byte pointer representing the response.

Example:

```
void* ctx;
byte* resp; /*Response buffer. */

…
EmbedOcspRespFree(ctx, resp);
```

See Also:
XFREE

17.5 Error Handling and Debugging

The functions in this section have to do with printing and handling errors as well as enabling and disabling debugging in wolfSSL.

wolfSSL_ERR_error_string

Synopsis:
#include <wolfssl/ssl.h>

char* wolfSSL_ERR_error_string(unsigned long errNumber, char* data);

Description:
This function converts an error code returned by wolfSSL_get_error() into a more human-readable error string. **errNumber** is the error code returned by wolfSSL_get_error() and **data** is the storage buffer which the error string will be placed in.

The maximum length of **data** is 80 characters by default, as defined by MAX_ERROR_SZ is wolfssl/wolfcrypt/error.h.

Return Values:
On successful completion, this function returns the same string as is returned in **data**. Upon failure, this function returns a string with the appropriate failure reason, **msg**.

Parameters:

errNumber - error code returned by wolfSSL_get_error().

data - output buffer containing human-readable error string matching **errNumber**.

Example:

```
int err = 0;
WOLFSSL* ssl;
char buffer[80];
...
err = wolfSSL_get_error(ssl, 0);
wolfSSL_ERR_error_string(err, buffer);
printf("err = %d, %s\n", err, buffer);
```

See Also:
wolfSSL_get_error
wolfSSL_ERR_error_string_n
wolfSSL_ERR_print_errors_fp
wolfSSL_load_error_strings

wolfSSL_ERR_error_string_n

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_ERR_error_string_n(unsigned long e, char* buf, unsigned long len);

Description:
This function is a version of wolfSSL_ERR_error_string() where **len** specifies the maximum number of characters that may be written to **buf**. Like wolfSSL_ERR_error_string(), this function converts an error code returned from wolfSSL_get_error() into a more human-readable error string. The human-readable string is placed in **buf**.

Return Values:
This function has no return value.

Parameters:

e - error code returned by wolfSSL_get_error().

buff - output buffer containing human-readable error string matching **e**.

len - maximum length in characters which may be written to **buf**.

Example:

```
int err = 0;
WOLFSSL* ssl;
char buffer[80];
...
err = wolfSSL_get_error(ssl, 0);
wolfSSL_ERR_error_string_n(err, buffer, 80);
printf("err = %d, %s\n", err, buffer);
```

See Also:
wolfSSL_get_error
wolfSSL_ERR_error_string
wolfSSL_ERR_print_errors_fp
wolfSSL_load_error_strings

wolfSSL_ERR_peek_last_error

Synopsis:

#include <wolfssl/ssl.h>

ERR_peek_last_error ->

unsigned long wolfSSL_ERR_peek_last_error(void);

Description:

This function returns the absolute value of the last error from WOLFSSL_ERROR encountered.

Return Values:

Returns absolute value of last error.

Parameters:

None

Example:

```
unsigned long err;

...

err = wolfSSL_ERR_peek_last_error();


// inspect err value
```

See Also:

wolfSSL_ERR_print_errors_fp

wolfSSL_ERR_print_errors_fp

Synopsis:

#include <wolfssl/ssl.h>

void wolfSSL_ERR_print_errors_fp(FILE* fp, int err);

Description:

This function converts an error code returned by wolfSSL_get_error() into a more human-readable error string and prints that string to the output file - **fp**. **err** is the error code returned by wolfSSL_get_error() and **fp** is the file which the error string will be placed in.

Return Values:

This function has no return value.

Parameters:

fp - output file for human-readable error string to be written to.

err - error code returned by wolfSSL_get_error().

Example:

```
int err = 0;
WOLFSSL* ssl;
FILE* fp = ...
...
```

```
err = wolfSSL_get_error(ssl, 0);
wolfSSL_ERR_print_errors_fp(fp, err);
```

See Also:
wolfSSL_get_error
wolfSSL_ERR_error_string
wolfSSL_ERR_error_string_n
wolfSSL_load_error_strings

wolfSSL_get_error

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_get_error(WOLFSSL* ssl, int ret);

Description:
This function returns a unique error code describing why the previous API function call (wolfSSL_connect, wolfSSL_accept, wolfSSL_read, wolfSSL_write, etc.) resulted in an error return code (SSL_FAILURE). The return value of the previous function is passed to wolfSSL_get_error through **ret**.

After wolfSSL_get_error is called and returns the unique error code, wolfSSL_ERR_error_string() may be called to get a human-readable error string. See wolfSSL_ERR_error_string() for more information.

Return Values:
On successful completion, this function will return the unique error code describing why the previous API function failed.

SSL_ERROR_NONE will be returned if **ret** > 0.

Parameters:

ssl - pointer to the SSL object, created with wolfSSL_new().

ret - return value of the previous function that resulted in an error return code.

Example:

```
int err = 0;
WOLFSSL* ssl;
char buffer[80];
...
err = wolfSSL_get_error(ssl, 0);
wolfSSL_ERR_error_string(err, buffer);
printf(“err = %d, %s\n”, err, buffer);
```

See Also:

wolfSSL_ERR_error_string
wolfSSL_ERR_error_string_n
wolfSSL_ERR_print_errors_fp
wolfSSL_load_error_strings

wolfSSL_load_error_strings

Synopsis:

#include <wolfssl/ssl.h>

void wolfSSL_load_error_strings(void);

Description:

This function is for OpenSSL compatibility (SSL_load_error_string) only and takes no action.

Return Values:

This function has no return value.

Parameters:

This function takes no parameters.

Example:

```
wolfSSL_load_error_strings();
```

See Also:

wolfSSL_get_error
wolfSSL_ERR_error_string
wolfSSL_ERR_error_string_n

wolfSSL_ERR_print_errors_fp
wolfSSL_load_error_strings

wolfSSL_want_read

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_want_read(WOLFSSL* ssl)

Description:
This function is similar to calling wolfSSL_get_error() and getting SSL_ERROR_WANT_READ in return. If the underlying error state is SSL_ERROR_WANT_READ, this function will return 1, otherwise, 0.

Return Values:

1 - wolfSSL_get_error() would return SSL_ERROR_WANT_READ, the underlying I/O has data available for reading.

0 - There is no SSL_ERROR_WANT_READ error state.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

Example:

```
int ret;
WOLFSSL* ssl = 0;
...

ret = wolfSSL_want_read(ssl);
if (ret == 1) {
      // underlying I/O has data available for reading (SSL_ERROR_WANT_READ)
}
```

See Also:
wolfSSL_want_write
wolfSSL_get_error

wolfSSL_want_write

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_want_write(WOLFSSL* ssl)

Description:

This function is similar to calling wolfSSL_get_error() and getting SSL_ERROR_WANT_WRITE in return. If the underlying error state is SSL_ERROR_WANT_WRITE, this function will return 1, otherwise, 0.

Return Values:

1 - wolfSSL_get_error() would return SSL_ERROR_WANT_WRITE, the underlying I/O needs data to be written in order for progress to be made in the underlying SSL connection.

0 - There is no SSL_ERROR_WANT_WRITE error state.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

Example:

```
int ret;
WOLFSSL* ssl = 0;
...

ret = wolfSSL_want_write(ssl);
if (ret == 1) {
      // underlying I/O needs data to be written (SSL_ERROR_WANT_WRITE)
}
```

See Also:

wolfSSL_want_read
wolfSSL_get_error

wolfSSL_Debugging_ON

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_Debugging_ON(void);

Description:

If logging has been enabled at build time this function turns on logging at runtime. To enable logging at build time use *--enable-debug* or define **DEBUG_WOLFSSL**

Return Values:

If successful this function will return 0.

NOT_COMPILED_IN is the error that will be returned if logging isn't enabled for this build.

Parameters:

This function has no parameters.

Example:

```
wolfSSL_Debugging_ON();
```

See Also:

wolfSSL_Debugging_OFF
wolfSSL_SetLoggingCb

wolfSSL_Debugging_OFF

Synopsis:

#include <wolfssl/ssl.h>

void wolfSSL_Debugging_OFF(void);

Description:

This function turns off runtime logging messages. If they're already off, no action is taken.

Return Values:

No return values are returned by this function.

Parameters:

This function has no parameters.

Example:

```
wolfSSL_Debugging_OFF();
```

See Also:
wolfSSL_Debugging_ON
wolfSSL_SetLoggingCb

17.6 OCSP and CRL

The functions in this section have to do with using OCSP (Online Certificate Status Protocol) and CRL (Certificate Revocation List) with wolfSSL.

wolfSSL_CTX_EnableOCSP

Synopsis:
long wolfSSL_CTX_EnableOCSP(WOLFSSL_CTX* ctx, int options);

Description:
This function sets options to configure behavior of OCSP functionality in wolfSSL. The value of **options** if formed by or'ing one or more of the following options:

WOLFSSL_OCSP_ENABLE
- enable OCSP lookups

WOLFSSL_OCSP_URL_OVERRIDE
- use the override URL instead of the URL in certificates.

The override URL is specified using the wolfSSL_CTX_SetOCSP_OverrideURL() function.

This function only sets the OCSP options when wolfSSL has been compiled with OCSP support (--enable-ocsp, #define HAVE_OCSP).

Return Values:

SSL_SUCCESS is returned upon success

SSL_FAILURE is returned upon failure

NOT_COMPILED_IN is returned when this function has been called, but OCSP support was not enabled when wolfSSL was compiled.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

options - value used to set the OCSP options.

Example:

```
WOLFSSL_CTX* ctx = 0;
...
wolfSSL_CTX_OCSP_set_options(ctx, WOLFSSL_OCSP_ENABLE);
```

See Also:
wolfSSL_CTX_OCSP_set_override_url

wolfSSL_CTX_SetOCSP_OverrideURL

Synopsis:
int wolfSSL_CTX_SetOCSP_OverrideURL(WOLFSSL_CTX* ctx, const char* url);

Description:
This function manually sets the URL for OCSP to use. By default, OCSP will use the URL found in the individual certificate unless the WOLFSSL_OCSP_URL_OVERRIDE option is set using the wolfSSL_CTX_EnableOCSP.

Return Values:

SSL_SUCCESS is returned upon success

SSL_FAILURE is returned upon failure

NOT_COMPILED_IN is returned when this function has been called, but OCSP support was not enabled when wolfSSL was compiled.

Parameters:

ctx - pointer to the SSL context, created with wolfSSL_CTX_new().

url - pointer to the OCSP URL for wolfSSL to use.

Example:

```
WOLFSSL_CTX* ctx = 0;
...
wolfSSL_CTX_OCSP_set_override_url(ctx, “custom-url-here”);
```

See Also:
wolfSSL_CTX_OCSP_set_options

wolfSSL_EnableCRL

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_EnableCRL(WOLFSSL* ssl, int options);

Description:
Enables CRL certificate revocation.

Return Values:
SSL_SUCCESS - the function and subroutines returned with no errors.

BAD_FUNC_ARG - returned if the WOLFSSL structure is NULL.

MEMORY_E - returned if the allocation of memory failed.

SSL_FAILURE - returned if the InitCRL function does not return successfully.

NOT_COMPILED_IN - HAVE_CRL was not enabled during the compiling.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

options - an integer that is used to determine the setting of crlCheckAll member of the WOLFSSL_CERT_MANAGER structure.

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
…
if (wolfSSL_EnableCRL(ssl, WOLFSSL_CRL_CHECKALL) != SSL_SUCCESS){
      /*Failure case. SSL_SUCCESS was not returned by this function or a
subroutine */
}
```

See Also:

wolfSSL_CertManagerEnableCRL
InitCRL

wolfSSL_DisableOCSP

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_DisableOCSP(WOLFSSL* ssl);

Description:
Disables the OCSP certificate revocation option.

Return Values:
SSL_SUCCESS - returned if the function and its subroutine return with no errors. The ocspEnabled member of the WOLFSSL_CERT_MANAGER structure was successfully

set.

BAD_FUNC_ARG - returned if the WOLFSSL structure is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
…
if(wolfSSL_DisableOCSP(ssl) != SSL_SUCCESS){
      /*Returned with an error. Failure case in this block. */
}
```

See Also:
wolfSSL_CertManagerDisableOCSP

wolfSSL_UseOCSPStapling

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_UseOCSPStapling(WOLFSSL* ssl, byte status_type, byte options);

Description:
Stapling eliminates the need to contact the CA. Stapling lowers the cost of certificate revocation check presented in OCSP.

Return Values:
SSL_SUCCESS - returned if TLSX_UseCertificateStatusRequest executes without error.

MEMORY_E - returned if there is an error with the allocation of memory.

BAD_FUNC_ARG - returned if there is an argument that has a NULL or otherwise unacceptable value passed into the function.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

status_type - a byte type that is passed through to TLSX_UseCertificateStatusRequest() and stored in the CertificateStatusRequest structure.

options - a byte type that is passed through to TLSX_UseCertificateStatusRequest() and stored in the CertificateStatusRequest structure.

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
…
if (wolfSSL_UseOCSPStapling(ssl, WOLFSSL_CSR2_OCSP,
      WOLFSSL_CSR2_OCSP_USE_NONCE) != SSL_SUCCESS){

      /*Failed case. */
}
```

See Also:
TLSX_UseCertificateStatusRequest
wolfSSL_CTX_UseOCSPStapling

EmbedOcspLookup

Synopsis:
#include <wolfssl/ssl.h>

int EmbedOcspLookup(void* ctx, const char* url, int urlSz, byte* ocspReqBuf,
                    int ocspReqSz, byte** ocspRespBuf);

Description:
This function retrieves the OCSP response from an OCSP responder URL given an input request.

Return Values:
>0 - OCSP Response Size

-1 - Error returned.

Parameters:

ctx - a void pointer representing the heap pointer.

url - a char pointer for the OCSP url for certificate verification.

urlSz - a byte pointer for the url size.

ocspReqBuf - a byte pointer for the OCSP request buffer.

ocspReqSz - an int type representing the size of the request buffer.

ocspRespBuf - a byte pointer that holds the OCSP response.

Example:

```
WOLFSSL_CERT_MANAGER* cm;
int options;
int wolfSSL_CertManagerEnableOCSP(WOLFSSL_CERT_MANAGER* cm, int options){
…
#ifndef WOLFSSL_USER_IO
      cm->ocspIOCb = EmbedOcspLookup;
      cm->ocspRespFreeCb = EmbedOcspRespFree;
#endif
```

See Also:
Process_http_response
build_http_request
wolfSSL_CertManagerEnableOCSPStapling

wolfSSL_CTX_UseOCSPStaplingV2

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_UseOCSPStaplingV2(WOLFSSL_CTX* ctx, bute status_type,
                                                  byte options);

Description:
Creates and initializes the certificate status request for OCSP Stapling.

Return Values:

SSL_SUCCESS - if the function and subroutines executed without error.

BAD_FUNC_ARG - returned if the WOLFSSL_CTX structure is NULL or if the side variable is not client side.

MEMORY_E - returned if the allocation of memory failed.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

status_type - a byte type that is located in the CertificatStatusRequest structure and must be either WOLFSSL_CSR2_OCSP or WOLFSSL_CSR2_OCSP_MULTI.

options - a byte type that will be held in CertificateStatusRequestItemV2 struct.

Example:

```
WOLFSSL_CTX* ctx  = wolfSSL_CTX_new(/*protocol method*/);
byte status_type;
byte options;
...
if(wolfSSL_CTX_UseOCSPStaplingV2(ctx, status_type, options); != SSL_SUCCESS){
      /*Failure case. */
}
```

See Also:

TLSX_UseCertificateStatusRequestV2
wc_RNG_GenerateBlock
TLSX_Push

wolfSSL_UseOCSPStaplingV2

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_UseOCSPStaplingV2(WOLFSSL* ssl, byte status_type, byte options);

Description:
The function sets the status type and options for OCSP.

Return Values:
SSL_SUCCESS - returned if the function and subroutines executed without error.

MEMORY_E - returned if there was an allocation of memory error.

BAD_FUNC_ARG - returned if a NULL or otherwise unaccepted argument was passed to the function or a subroutine.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

status_type - a byte type that loads the OCSP status type.

options - a byte type that holds the OCSP options, set in wolfSSL_SNI_SetOptions() and wolfSSL_CTX_SNI_SetOptions().

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
...
if (wolfSSL_UseOCSPStaplingV2(ssl, WOLFSSL_CSR2_OCSP_MULTI, 0) !=
SSL_SUCCESS){
      /*Did not execute properly. Failure case code block. */
}
```

See Also:
TLSX_UseCertificatStatusRequestV2
wolfSSL_SNI_SetOptions
wolfSSL_CTX_SNI_SetOptions

wolfSSL_CTX_LoadCRL

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_LoadCRL(WOLFSSL_CTX* ctx, const char* path, int type,
int monitor);

Description:

This function loads CRL into the WOLFSSL_CTX structure through wolfSSL_CertManagerLoadCRL().

Return Values:

SSL_SUCCESS - returned if the function and its subroutines execute without error.

BAD_FUNC_ARG - returned if this function or any subroutines are passed NULL structures.

BAD_PATH_ERROR - returned if the **path** variable opens as NULL.

MEMORY_E - returned if an allocation of memory failed.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

path - the path to the certificate.

type - an integer variable holding the type of certificate.

monitor - an integer variable used to determine if the monitor path is requested.

Example:

```
WOLFSSL_CTX* ctx;
const char* path;
…
return wolfSSL_CTX_LoadCRL(ctx, path, SSL_FILETYPE_PEM, 0);
```

See Also:

wolfSSL_CertManagerLoadCRL
LoadCRL

wolfSSL_CertManagerLoadCRLBuffer

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CertManagerLoadCRLBuffer(WOLFSSL_CERT_MANAGER* cm,
const unsigned char* buff, long sz, int type);

Description:

The function loads the CRL file by calling BufferLoadCRL.

Return Values:

SSL_SUCCESS - returned if the function completed without errors.

BAD_FUNC_ARG - returned if the WOLFSSL_CERT_MANAGER is NULL .

SSL_FATAL_ERROR - returned if there is an error associated with the WOLFSSL_CERT_MANAGER.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure.

buff - a constant byte type and is the buffer.

sz - a long int representing the size of the buffer.

type - a long integer that holds the certificate type.

Example:

```
WOLFSSL_CERT_MANAGER* cm;
const unsigned char* buff;
long sz; /*size of buffer*/
int type;  /*cert type*/
...
int ret = wolfSSL_CertManagerLoadCRLBuffer(cm, buff, sz, type);
if(ret == SSL_SUCCESS){
      return ret;
} else {
      /*Failure case. */
}
```

See Also:
BufferLoadCRL
wolfSSL_CertManagerEnableCRL

wolfSSL_LoadCRL

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_LoadCRL(WOLFSSL* ssl, const char* path, int type, int monitor);

Description:
A wrapper function that ends up calling LoadCRL to load the certificate for revocation checking.

Return Values:
WOLFSSL_SUCCESS - returned if the function and all of the subroutines executed without error.

SSL_FATAL_ERROR - returned if one of the subroutines does not return successfully.

BAD_FUNC_ARG - f the WOLFSSL_CERT_MANAGER or the WOLFSSL structure are NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

path - a constant character pointer that holds the path to the crl file.

type - an integer representing the type of certificate.

monitor - an integer variable used to verify the monitor path if requested.

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
const char* crlPemDir;
…
if(wolfSSL_LoadCRL(ssl, crlPemDir, SSL_FILETYPE_PEM, 0) != SSL_SUCCESS){
      /*Failure case. Did not return SSL_SUCCESS. */
}
```

See Also:
wolfSSL_CertManagerLoadCRL
wolfSSL_CertManagerEnableCRL
LoadCRL

wolfSSL_DisableCRL

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_DisableCRL(WOLFSSL* ssl);

Description:
Disables CRL certificate revocation.

Return Values:
SSL_SUCCESS - wolfSSL_CertMangerDisableCRL successfully disabled the crlEnabled member of the WOLFSSL_CERT_MANAGER structure.

BAD_FUNC_ARG - the WOLFSSL structure was NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
...
if(wolfSSL_DisableCRL(ssl) != SSL_SUCCESS){
      /*Failure case*/
}
```

See Also:

wolfSSL_CertManagerDisableCRL
wolfSSL_CertManagerDisableOCSP

wolfSSL_CertManagerDisableOCSP

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CertManagerDisableOCSP(WOLFSSL* ssl);

Description:

Disables OCSP certificate revocation.

Return Values:

SSL_SUCCESS - wolfSSL_CertMangerDisableCRL successfully disabled the crlEnabled member of the WOLFSSL_CERT_MANAGER structure.

BAD_FUNC_ARG - the WOLFSSL structure was NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
...
if(wolfSSL_CertManagerDisableOCSP(ssl) != SSL_SUCCESS){
      /*Fail case. */
}
```

See Also:

wolfSSL_DisableCRL

wolfSSL_CertManagerCheckCRL

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CertManagerCheckCRL(WOLFSSL_CERT_MANAGER* cm, byte* der,
int sz);

Description:
Check CRL if the option is enabled and compares the cert to the CRL list.

Return Values:
SSL_SUCCESS - returns if the function returned as expected. If the crlEnabled member of the WOLFSSL_CERT_MANAGER struct is turned on.

MEMORY_E - returns if the allocated memory failed.

BAD_FUNC_ARG - if the WOLFSSL_CERT_MANAGER is NULL.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER struct.

der - pointer to a DER formatted certificate.

sz - size of the certificate.

Example:

```
WOLFSSL_CERT_MANAGER* cm;
byte* der;
int sz; /*size of der */
...
if(wolfSSL_CertManagerCheckCRL(cm, der, sz) != SSL_SUCCESS){
      /*Error returned. Deal with failure case. */
}
```

See Also:
CheckCertCRL

ParseCertRelative
wolfSSL_CertManagerSetCRL_CB
InitDecodedCert

wolfSSL_CTX_EnableCRL

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_EnableCRL(WOLFSSL_CTX* ctx, int options);

Description:

Enables CRL certificate verification through the CTX.

Return Values:

SSL_SUCCESS - returned if this function and it's subroutines execute without errors.

BAD_FUNC_ARG - returned if the CTX struct is NULL or there was otherwise an invalid argument passed in a subroutine.

MEMORY_E - returned if there was an error allocating memory during execution of the function.

SSL_FAILURE - returned if the crl member of the WOLFSSL_CERT_MANAGER fails to initialize correctly.

NOT_COMPILED_IN - wolfSSL was not compiled with the HAVE_CRL option.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
```

```
...
if(wolfSSL_CTX_EnableCRL(ssl->ctx, options) != SSL_SUCCESS){
      /*The function failed*/
}
```

See Also:

wolfSSL_CertManagerEnableCRL
InitCRL
wolfSSL_CTX_DisableCRL

wolfSSL_CTX_DisableCRL

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_DisableCRL(WOLFSSL_CTX* ctx);

Description:

This function disables CRL verification in the CTX structure.

Return Values:

SSL_SUCCESS - returned if the function executes without error. The crlEnabled member of the WOLFSSL_CERT_MANAGER struct is set to 0.

BAD_FUNC_ARG - returned if either the CTX struct or the CM struct has a NULL value.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);

...
if(wolfSSL_CTX_DisableCRL(ssl->ctx) != SSL_SUCCESS){
      /*Failure case.*/
}
```

See Also:

wolfSSL_CertManagerDisableCRL

wolfSSL_EnableOCSP

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_EnableOCSP(WOLFSSL* ssl, int options);

Description:
This function enables OCSP certificate verification.

Return Values:
SSL_SUCCESS - returned if the function and subroutines executes without errors.

BAD_FUNC_ARG - returned if an argument in this function or any subroutine receives an invalid argument value.

MEMORY_E - returned if there was an error allocating memory for a structure or other variable.

NOT_COMPILED_IN - returned if wolfSSL was not compiled with the HAVE_OCSP option.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

options - an integer type passed to wolfSSL_CertMangerENableOCSP() used for settings check.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
int options; /*initialize to option constant*/
…
int ret = wolfSSL_EnableOCSP(ssl, options);
```

```
if(ret != SSL_SUCCESS){
      /*OCSP is not enabled*/
}
```

See Also:
wolfSSL_CertManagerEnableOCSP

wolfSSL_CTX_UseOCSPStapling

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_UseOCSPStapling(WOLFSSL_CTX* ctx, byte status_type,
byte options);

Description:
This function requests the certificate status during the handshake.

Return Values:
SSL_SUCCESS - returned if the function and subroutines execute without error.

BAD_FUNC_ARG - returned if the WOLFSSL_CTX structure is NULL or otherwise if a unpermitted value is passed to a subroutine.

MEMORY_E - returned if the function or subroutine failed to properly allocate memory.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

status_type - a byte type that is passed through to TLSX_UseCertificateStatusRequest() and stored in the CertificateStatusRequest structure.

options - a byte type that is passed through to TLSX_UseCertificateStatusRequest() and stored in the CertificateStatusRequest structure.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
```

```
WOLFSSL* ssl = wolfSSL_new(ctx);
byte statusRequest = 0; /*Initialize status request*/
…
switch(statusRequest){
      case WOLFSSL_CSR_OCSP:
            if(wolfSSL_CTX_UseOCSPStapling(ssl->ctx, WOLFSSL_CSR_OCSP,
                        WOLF_CSR_OCSP_USE_NONCE) != SSL_SUCCESS){
                        /*UseCertificateStatusRequest failed*/
                        }
                  /*Continue switch cases*/
```

See Also:
wolfSSL_UseOCSPStaplingV2
wolfSSL_UseOCSPStapling
TLSX_UseCertificateStatusRequest

wolfSSL_CTX_DisableOCSP

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_CTX_DisableOCSP(WOLFSSL_CTX* ctx);

Description:
This function disables OCSP certificate revocation checking by affecting the ocspEnabled member of the WOLFSSL_CERT_MANAGER structure.

Return Values:
SSL_SUCCESS - returned if the function executes without error. The ocspEnabled member of the CM has been disabled.

BAD_FUNC_ARG - returned if the WOLFSSL_CTX structure is NULL.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
```

```
WOLFSSL* ssl = wolfSSL_new(ctx);
...
if(!wolfSSL_CTX_DisableOCSP(ssl->ctx)){
      /*OCSP is not disabled*/
}
```

See Also:
wolfSSL_DisableOCSP
wolfSSL_CertManagerDisableOCSP

wolfSSL_CTX_EnableOCSPStapling

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_CTX_EnableOCSPStapling(WOLFSSL_CTX* ctx);

Description:
This function enables OCSP stapling by calling wolfSSL_CertManagerEnableOCSPStapling().

Return Values:
SSL_SUCCESS - returned if there were no errors and the function executed successfully.

BAD_FUNC_ARG - returned if the WOLFSSL_CTX structure is NULL or otherwise if there was a unpermitted argument value passed to a subroutine.

MEMORY_E - returned if there was an issue allocating memory.

SSL_FAILURE - returned if the initialization of the OCSP structure failed.

NOT_COMPILED_IN - returned if wolfSSL was not compiled with HAVE_CERTIFICATE_STATUS_REQUEST option.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

Example:

```
WOLFSSL* ssl = WOLFSSL_new();
ssl->method.version; /*set to desired protocol*/
...
if(!wolfSSL_CTX_EnableOCSPStapling(ssl->ctx)){
      /*OCSP stapling is not enabled*/
}
```

See Also:
wolfSSL_CertManagerEnableOCSPStapling
InitOCSP

wolfSSL_CertManagerEnableOCSPStapling

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_CertManagerEnableOCSPStapling(WOLFSSL_CERT_MANAGER* cm);

Description:
This function turns on OCSP stapling if it is not turned on as well as set the options.

Return Values:
SSL_SUCCESS - returned if there were no errors and the function executed successfully.

BAD_FUNC_ARG - returned if the WOLFSSL_CERT_MANAGER structure is NULL or otherwise if there was a unpermitted argument value passed to a subroutine.

MEMORY_E - returned if there was an issue allocating memory.

SSL_FAILURE - returned if the initialization of the OCSP structure failed.

NOT_COMPILED_IN - returned if wolfSSL was not compiled with HAVE_CERTIFICATE_STATUS_REQUEST option.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, a member of the WOLFSSL_CTX structure.

Example:

```
int wolfSSL_CTX_EnableOCSPStapling(WOLFSSL_CTX* ctx){
…
return wolfSSL_CertManagerEnableOCSPStapling(ctx->cm);
```

See Also:
wolfSSL_CTX_EnableOCSPStapling

wolfSSL_SetOCSP_OverrideURL

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_SetOCSP_OverrideURL(WOLFSSL* ssl, const char* url);

Description:
This function sets the ocspOverrideURL member in the WOLFSSL_CERT_MANAGER structure.

Return Values:
SSL_SUCCESS - returned on successful execution of the function.

BAD_FUNC_ARG - returned if the WOLFSSL struct is NULL or if a unpermitted argument was passed to a subroutine.

MEMORY_E - returned if there was an error allocating memory in the subroutine.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

url - a constant char pointer to the url that will be stored in the ocspOverrideURL member of the WOLFSSL_CERT_MANAGER structure.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
char url[URLSZ];
...
if(wolfSSL_SetOCSP_OverrideURL(ssl, url)){
      /*The override url is set to the new value*/
}
```

See Also:
wolfSSL_CertManagerSetOCSPOverrideURL

17.7 Informational

The functions in this section are informational. They allow the application to gather some kind of information about the current status or setup of wolfSSL.

wolfSSL_GetObjectSize

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_GetObjectSize(void);

Description:
This function returns the size of the WOLFSSL object and will be dependent on build options and settings. If SHOW_SIZES has been defined when building wolfSSL, this function will also print the sizes of individual objects within the WOLFSSL object (Suites, Ciphers, etc.) to stdout.

Return Values:
This function returns the size of the WOLFSSL object.

Parameters:

This function has no parameters.

Example:

```
int size = 0;
size = wolfSSL_GetObjectSize();
printf("sizeof(WOLFSSL) = %d\n", size);
```

See Also:
wolfSSL_new();

wolfSSL_GetMacSecret

Synopsis:
#include <wolfssl/ssl.h>

const byte* wolfSSL_GetMacSecret(WOLFSSL* ssl, int verify);

Description:
Allows retrieval of the Hmac/Mac secret from the handshake process. The **verify** parameter specifies whether this is for verification of a peer message.

Return Values:
If successful the call will return a valid pointer to the secret. The size of the secret can be obtained from wolfSSL_GetHmacSize().

NULL will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

verify - specifies whether this is for verification of a peer message.

See Also:
wolfSSL_GetHmacSize()

wolfSSL_GetClientWriteKey

Synopsis:
#include <wolfssl/ssl.h>

const byte* wolfSSL_GetClientWriteKey(WOLFSSL* ssl);

Description:
Allows retrieval of the client write key from the handshake process.

Return Values:
If successful the call will return a valid pointer to the key. The size of the key can be obtained from wolfSSL_GetKeySize().

NULL will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetKeySize()
wolfSSL_GetClientWriteIV()

wolfSSL_GetClientWriteIV

Synopsis:
#include <wolfssl/ssl.h>

const byte* wolfSSL_GetClientWriteIV(WOLFSSL* ssl);

Description:
Allows retrieval of the client write IV (initialization vector) from the handshake process.

Return Values:
If successful the call will return a valid pointer to the IV. The size of the IV can be obtained from wolfSSL_GetCipherBlockSize().

NULL will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetCipherBlockSize()
wolfSSL_GetClientWriteKey()

wolfSSL_GetServerWriteKey

Synopsis:
#include <wolfssl/ssl.h>

const byte* wolfSSL_GetServerWriteKey(WOLFSSL* ssl);

Description:
Allows retrieval of the server write key from the handshake process.

Return Values:
If successful the call will return a valid pointer to the key. The size of the key can be obtained from wolfSSL_GetKeySize().

NULL will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetKeySize()
wolfSSL_GetServerWriteIV()

wolfSSL_GetServerWriteIV

Synopsis:
#include <wolfssl/ssl.h>

const byte* wolfSSL_GetServerWriteIV(WOLFSSL* ssl);

Description:
Allows retrieval of the server write IV (initialization vector) from the handshake process.

Return Values:
If successful the call will return a valid pointer to the IV. The size of the IV can be obtained from wolfSSL_GetCipherBlockSize().

NULL will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetCipherBlockSize()
wolfSSL_GetClientWriteKey()

wolfSSL_GetKeySize

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_GetKeySize(WOLFSSL* ssl);

Description:
Allows retrieval of the key size from the handshake process.

Return Values:
If successful the call will return the key size in bytes.

BAD_FUNC_ARG will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetClientWriteKey()

wolfSSL_GetServerWriteKey()

wolfSSL_GetSide

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_GetSide(WOLFSSL* ssl);

Description:
Allows retrieval of the side of this WOLFSSL connection.

Return Values:
If successful the call will return either **WOLFSSL_SERVER_END** or **WOLFSSL_CLIENT_END** depending on the side of WOLFSSL object.

BAD_FUNC_ARG will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetClientWriteKey()
wolfSSL_GetServerWriteKey()

wolfSSL_IsTLSv1_1

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_IsTLSV1_1(WOLFSSL* ssl);

Description:
Allows caller to determine if the negotiated protocol version is at least TLS version 1.1 or greater.

Return Values:
If successful the call will return **1** for true or **0** for false.

BAD_FUNC_ARG will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetSide()

wolfSSL_GetBulkCipher

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_GetBulkCipher(WOLFSSL* ssl);

Description:
Allows caller to determine the negotiated bulk cipher algorithm from the handshake.

Return Values:
If successful the call will return one of the following:

wolfssl_cipher_null
wolfssl_des
wolfssl_triple_des
wolfssl_aes
wolfssl_aes_gcm
wolfssl_aes_ccm
wolfssl_camellia
wolfssl_hc128
wolfssl_rabbit

BAD_FUNC_ARG will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetCipherBlockSize()
wolfSSL_GetKeySize()

wolfSSL_GetCipherBlockSize

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_GetCipherBlockSize(WOLFSSL* ssl);

Description:
Allows caller to determine the negotiated cipher block size from the handshake.

Return Values:
If successful the call will return the size in bytes of the cipher block size.

BAD_FUNC_ARG will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetBulkCipher()
wolfSSL_GetKeySize()

wolfSSL_GetAeadMacSize

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_GetAeadMacSize(WOLFSSL* ssl);

Description:

Allows caller to determine the negotiated aead mac size from the handshake. For cipher type **WOLFSSL_AEAD_TYPE**.

Return Values:
If successful the call will return the size in bytes of the aead mac size.

BAD_FUNC_ARG will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetBulkCipher()
wolfSSL_GetKeySize()

wolfSSL_GetHmacSize

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_GetHmacSize(WOLFSSL* ssl);

Description:
Allows caller to determine the negotiated (h)mac size from the handshake. For cipher types except **WOLFSSL_AEAD_TYPE**.

Return Values:
If successful the call will return the size in bytes of the (h)mac size.

BAD_FUNC_ARG will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetBulkCipher()
wolfSSL_GetHmacType()

wolfSSL_GetHmacType

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_GetHmacType(WOLFSSL* ssl);

Description:
Allows caller to determine the negotiated (h)mac type from the handshake. For cipher types except **WOLFSSL_AEAD_TYPE**.

Return Values:
If successful the call will return one of the following:

MD5
SHA
SHA256
SHA384

BAD_FUNC_ARG or **SSL_FATAL_ERROR** will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetBulkCipher()
wolfSSL_GetHmacSize()

wolfSSL_GetCipherType

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_GetCipherType(WOLFSSL* ssl);

Description:
Allows caller to determine the negotiated cipher type from the handshake.

Return Values:
If successful the call will return one of the following:

WOLFSSL_BLOCK_TYPE
WOLFSSL_STREAM_TYPE
WOLFSSL_AEAD_TYPE

BAD_FUNC_ARG will be returned for an error state.

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetBulkCipher()
wolfSSL_GetHmacType()

wolfSSL_GetOutputSize

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_GetOutputSize(WOLFSSL* ssl, int inSz);

Description:
Returns the record layer size of the plaintext input. This is helpful when an application wants to know how many bytes will be sent across the Transport layer, given a specified plaintext input size.

This function must be called after the SSL/TLS handshake has been completed.

Return Values:
Upon success, the requested size will be returned. Upon error, one of the following will be returned:

INPUT_SIZE_E will be returned if the input size is greater than the maximum TLS fragment size (see wolfSSL_GetMaxOutputSize())

BAD_FUNC_ARG will be returned upon invalid function argument, or if the SSL/TLS handshake has not been completed yet

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

inSz - size of plaintext data

See Also:
wolfSSL_GetMaxOutputSize()

wolfSSL_GetMaxOutputSize

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_GetMaxOutputSize(WOLFSSL* ssl);

Description:
Returns the maximum record layer size for plaintext data. This will correspond to either the maximum SSL/TLS record size as specified by the protocol standard, the maximum TLS fragment size as set by the TLS Max Fragment Length extension.

This function is helpful when the application has called wolfSSL_GetOutputSize() and received a INPUT_SIZE_E error.

This function must be called after the SSL/TLS handshake has been completed.

Return Values:
Upon success, the maximum output size will be returned. Upon error, one of the following will be returned:

BAD_FUNC_ARG will be returned upon invalid function argument, or if the SSL/TLS handshake has not been completed yet

Parameters:

ssl - a pointer to a WOLFSSL object, created using wolfSSL_new().

See Also:
wolfSSL_GetOutputSize()

17.8 Connection, Session, and I/O

The functions in this section deal with setting up the SSL/TLS connection, managing SSL sessions, and input/output.

wolfSSL_accept

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_accept(WOLFSSL* ssl);

Description:
This function is called on the server side and waits for an SSL client to initiate the SSL/TLS handshake. When this function is called, the underlying communication channel has already been set up.

wolfSSL_accept() works with both blocking and non-blocking I/O. When the underlying I/O is non-blocking, wolfSSL_accept() will return when the underlying I/O could not satisfy the needs of wolfSSL_accept to continue the handshake. In this case, a call to wolfSSL_get_error() will yield either **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**. The calling process must then repeat the call to wolfSSL_accept when data is available to read and wolfSSL will pick up where it left off. When using a non-blocking socket, nothing needs to be done, but select() can be used to check for the required condition.

If the underlying I/O is blocking, wolfSSL_accept() will only return once the handshake has been finished or an error occurred.

Return Values:
If successful the call will return **SSL_SUCCESS**.

SSL_FATAL_ERROR will be returned if an error occurred. To get a more detailed error code, call wolfSSL_get_error().

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
int ret = 0;
int err = 0;
WOLFSSL* ssl;
char buffer[80];
...

ret = wolfSSL_accept(ssl);
if (ret != SSL_SUCCESS) {
      err = wolfSSL_get_error(ssl, ret);
      printf("error = %d, %s\n", err, wolfSSL_ERR_error_string(err, buffer));
}
```

See Also:
wolfSSL_get_error
wolfSSL_connect

wolfSSL_connect

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_connect(WOLFSSL* ssl);

Description:
This function is called on the client side and initiates an SSL/TLS handshake with a server. When this function is called, the underlying communication channel has already been set up.

wolfSSL_connect() works with both blocking and non-blocking I/O. When the underlying I/O is non-blocking, wolfSSL_connect() will return when the underlying I/O could not satisfy the needs of wolfSSL_connect to continue the handshake. In this case, a call to wolfSSL_get_error() will yield either **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**. The calling process must then repeat the call to wolfSSL_connect() when the underlying I/O is ready and wolfSSL will pick up where it left off. When using a non-blocking socket, nothing needs to be done, but select() can be used to check for the required condition.

If the underlying I/O is blocking, wolfSSL_connect() will only return once the handshake has been finished or an error occurred.

wolfSSL takes a different approach to certificate verification than OpenSSL does. The default policy for the client is to verify the server, this means that if you don't load CAs to verify the server you'll get a connect error, unable to verify (-155). It you want to mimic OpenSSL behavior of having SSL_connect succeed even if verifying the server fails and reducing security you can do this by calling:

SSL_CTX_set_verify(ctx, SSL_VERIFY_NONE, 0);

before calling SSL_new(); Though it's not recommended.

Return Values:

If successful the call will return **SSL_SUCCESS**.

SSL_FATAL_ERROR will be returned if an error occurred. To get a more detailed error code, call wolfSSL_get_error().

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
int ret = 0;
int err = 0;
WOLFSSL* ssl;
char buffer[80];
...

ret = wolfSSL_connect(ssl);
if (ret != SSL_SUCCESS) {
      err = wolfSSL_get_error(ssl, ret);
      printf("error = %d, %s\n", err, wolfSSL_ERR_error_string(err, buffer));
}
```

See Also:

wolfSSL_get_error

wolfSSL_accept

wolfSSL_connect_cert

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_connect_cert(WOLFSSL* ssl);

Description:
This function is called on the client side and initiates an SSL/TLS handshake with a server only long enough to get the peer's certificate chain. When this function is called, the underlying communication channel has already been set up.

wolfSSL_connect_cert() works with both blocking and non-blocking I/O. When the underlying I/O is non-blocking, wolfSSL_connect_cert() will return when the underlying I/O could not satisfy the needs of wolfSSL_connect_cert() to continue the handshake. In this case, a call to wolfSSL_get_error() will yield either **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**. The calling process must then repeat the call to wolfSSL_connect_cert() when the underlying I/O is ready and wolfSSL will pick up where it left off. When using a non-blocking socket, nothing needs to be done, but select() can be used to check for the required condition.

If the underlying I/O is blocking, wolfSSL_connect_cert() will only return once the peer's certificate chain has been received.

Return Values:
If successful the call will return **SSL_SUCCESS**.

SSL_FAILURE will be returned if the SSL session parameter is NULL.

SSL_FATAL_ERROR will be returned if an error occurred. To get a more detailed error code, call wolfSSL_get_error().

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
int ret = 0;
int err = 0;
WOLFSSL* ssl;
char buffer[80];
...

ret = wolfSSL_connect_cert(ssl);
if (ret != SSL_SUCCESS) {
      err = wolfSSL_get_error(ssl, ret);
      printf(“error = %d, %s\n”, err, wolfSSL_ERR_error_string(err, buffer));
}
```

See Also:
wolfSSL_get_error
wolfSSL_connect
wolfSSL_accept

wolfSSL_get_fd

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_get_fd(const WOLFSSL* ssl);

Description:
This function returns the file descriptor (**fd**) used as the input/output facility for the SSL connection. Typically this will be a socket file descriptor.

Return Values:
If successful the call will return the SSL session file descriptor.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

Example:

```
int sockfd;
WOLFSSL* ssl = 0;
...
```

```
sockfd = wolfSSL_get_fd(ssl);
...
```

See Also:
wolfSSL_set_fd

wolfSSL_get_session

Synopsis:
#include <wolfssl/ssl.h>

WOLFSSL_SESSION* wolfSSL_get_session(WOLFSSL* ssl);

Description:
This function returns a pointer to the current session (WOLFSSL_SESSION) used in **ssl**. The WOLFSSL_SESSION pointed to contains all the necessary information required to perform a session resumption and reestablish the connection without a new handshake.

For session resumption, before calling wolfSSL_shutdown() with your session object, an application should save the session ID from the object with a call to wolfSSL_get_session(), which returns a pointer to the session. Later, the application should create a new WOLFSSL object and assign the saved session with wolfSSL_set_session(). At this point, the application may call wolfSSL_connect() and wolfSSL will try to resume the session. The wolfSSL server code allows session resumption by default.

Return Values:
If successful the call will return a pointer to the the current SSL session object.

NULL will be returned if **ssl** is NULL, the SSL session cache is disabled, wolfSSL doesn’t have the Session ID available, or mutex functions fail.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

Example:

```
WOLFSSL* ssl = 0;
```

```
WOLFSSL_SESSION* session = 0;
...
session = wolfSSL_get_session(ssl);
if (session == NULL) {
      /*failed to get session pointer*/
}
...
```

See Also:
wolfSSL_set_session

wolfSSL_get_using_nonblock

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_get_using_nonblock(WOLFSSL* ssl);

Description:
This function allows the application to determine if wolfSSL is using non-blocking I/O. If wolfSSL is using non-blocking I/O, this function will return 1, otherwise 0.

After an application creates a WOLFSSL object, if it will be used with a non-blocking socket, call wolfSSL_set_using_nonblock() on it. This lets the WOLFSSL object know that receiving EWOULDBLOCK means that the recvfrom call would block rather than that it timed out.

Return Values:

0 - underlying I/O is blocking.

1 - underlying I/O is non-blocking.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

Example:

```
int ret = 0;
WOLFSSL* ssl = 0;
...
ret = wolfSSL_get_using_nonblock(ssl);
```

```
if (ret == 1) {
      /*underlying I/O is non-blocking*/
}
...
```

See Also:
wolfSSL_set_session

wolfSSL_flush_sessions

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_flush_sessions(WOLFSSL_CTX *ctx, long tm);

Description:
This function flushes session from the session cache which have expired. The time, **tm**, is used for the time comparison.

Note that wolfSSL currently uses a static table for sessions, so no flushing is needed. As such, this function is currently just a stub. This function provides OpenSSL compatibility (SSL_flush_sessions) when wolfSSL is compiled with the OpenSSL compatibility layer.

Return Values:
This function does not have a return value.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

tm - time used in session expiration comparison.

Example:

```
WOLFSSL_CTX* ssl;
...

wolfSSL_flush_sessions(ctx, time(0));
```

See Also:

wolfSSL_get_session
wolfSSL_set_session

wolfSSL_negotiate

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_negotiate(WOLFSSL* ssl);

Description:
Performs the actual connect or accept based on the side of the SSL method. If called from the client side then an *wolfSSL_connect()* is done while a *wolfSSL_accept()* is performed if called from the server side.

Return Values:
SSL_SUCCESS will be returned if successful. (Note, older versions will return 0.)

SSL_FATAL_ERROR will be returned if the underlying call resulted in an error. Use wolfSSL_get_error() to get a specific error code.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

Example:
```
int ret = SSL_FATAL_ERROR;
WOLFSSL* ssl = 0;
...
ret = wolfSSL_negotiate(ssl);
if (ret == SSL_FATAL_ERROR) {
      /*SSL establishment failed*/
      int error_code = wolfSSL_get_error(ssl);
      ...
}
...
```

See Also:
SSL_connect
SSL_accept

wolfSSL_peek

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_peek(WOLFSSL* ssl, void* data, int sz);

Description:

This function copies **sz** bytes from the SSL session (**ssl**) internal read buffer into the buffer **data**. This function is identical to wolfSSL_read() except that the data in the internal SSL session receive buffer is not removed or modified.

If necessary, like wolfSSL_read(), wolfSSL_peek() will negotiate an SSL/TLS session if the handshake has not already been performed yet by wolfSSL_connect() or wolfSSL_accept().

The SSL/TLS protocol uses SSL records which have a maximum size of 16kB (the max record size can be controlled by the MAX_RECORD_SIZE define in <wolfssl_root>/wolfssl/internal.h). As such, wolfSSL needs to read an entire SSL record internally before it is able to process and decrypt the record. Because of this, a call to wolfSSL_peek() will only be able to return the maximum buffer size which has been decrypted at the time of calling. There may be additional not-yet-decrypted data waiting in the internal wolfSSL receive buffer which will be retrieved and decrypted with the next call to wolfSSL_peek() / wolfSSL_read().

If **sz** is larger than the number of bytes in the internal read buffer, SSL_peek() will return the bytes available in the internal read buffer. If no bytes are buffered in the internal read buffer yet, a call to wolfSSL_peek() will trigger processing of the next record.

Return Values:

>0 - the number of bytes read upon success.

0 - will be returned upon failure. This may be caused by a either a clean (close notify alert) shutdown or just that the peer closed the connection. Call wolfSSL_get_error() for the specific error code.

SSL_FATAL_ERROR - will be returned upon failure when either an error occurred or, when using non-blocking sockets, the SSL_ERROR_WANT_READ or SSL_ERROR_WANT_WRITE error was received and and the application needs to call wolfSSL_peek() again. Use wolfSSL_get_error() to get a specific error code.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

data - buffer where wolfSSL_peek() will place data read.

sz - number of bytes to read into **data**.

Example:

```
WOLFSSL* ssl = 0;
char reply[1024];
...

input = wolfSSL_peek(ssl, reply, sizeof(reply));
if (input > 0) {
      /*"input" number of bytes returned into buffer "reply"*/
}
```

See Also:
wolfSSL_read

wolfSSL_pending

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_pending(WOLFSSL* ssl);

Description:
This function returns the number of bytes which are buffered and available in the SSL object to be read by wolfSSL_read().

Return Values:

This function returns the number of bytes pending.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

Example:

```
int pending = 0;
WOLFSSL* ssl = 0;
...

pending = wolfSSL_pending(ssl);
printf("There are %d bytes buffered and available for reading", pending);
```

See Also:
wolfSSL_recv
wolfSSL_read
wolfSSL_peek

wolfSSL_read

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_read(WOLFSSL* ssl, void* data, int sz);

Description:
This function reads **sz** bytes from the SSL session (**ssl**) internal read buffer into the buffer **data**. The bytes read are removed from the internal receive buffer.

If necessary wolfSSL_read() will negotiate an SSL/TLS session if the handshake has not already been performed yet by wolfSSL_connect() or wolfSSL_accept().

The SSL/TLS protocol uses SSL records which have a maximum size of 16kB (the max record size can be controlled by the MAX_RECORD_SIZE define in <wolfssl_root>/wolfssl/internal.h). As such, wolfSSL needs to read an entire SSL record internally before it is able to process and decrypt the record. Because of this, a call to wolfSSL_read() will only be able to return the maximum buffer size which has been decrypted at the time of calling. There may be additional not-yet-decrypted data waiting in the internal wolfSSL receive buffer which will be retrieved and decrypted with the next call to wolfSSL_read().

If **sz** is larger than the number of bytes in the internal read buffer, SSL_read() will return the bytes available in the internal read buffer. If no bytes are buffered in the internal read buffer yet, a call to wolfSSL_read() will trigger processing of the next record.

Return Values:

>0 - the number of bytes read upon success.

0 - will be returned upon failure. This may be caused by a either a clean (close notify alert) shutdown or just that the peer closed the connection. Call wolfSSL_get_error() for the specific error code.

SSL_FATAL_ERROR - will be returned upon failure when either an error occurred or, when using non-blocking sockets, the SSL_ERROR_WANT_READ or SSL_ERROR_WANT_WRITE error was received and and the application needs to call wolfSSL_read() again. Use wolfSSL_get_error() to get a specific error code.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

data - buffer where wolfSSL_read() will place data read.

sz - number of bytes to read into **data**.

Example:

```
WOLFSSL* ssl = 0;
char reply[1024];
...

input = wolfSSL_read(ssl, reply, sizeof(reply));
if (input > 0) {
      /*"input" number of bytes returned into buffer "reply"*/
}
```

See wolfSSL examples (client, server, echoclient, echoserver) for more complete examples of wolfSSL_read().

See Also:
wolfSSL_recv
wolfSSL_write
wolfSSL_peek

wolfSSL_pending

wolfSSL_recv

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_recv(WOLFSSL* ssl, void* data, int sz, int flags);

Description:

This function reads **sz** bytes from the SSL session (**ssl**) internal read buffer into the buffer **data** using the specified **flags** for the underlying recv operation. The bytes read are removed from the internal receive buffer. This function is identical to wolfSSL_read() except that it allows the application to set the recv flags for the underlying read operation.

If necessary wolfSSL_recv() will negotiate an SSL/TLS session if the handshake has not already been performed yet by wolfSSL_connect() or wolfSSL_accept().

The SSL/TLS protocol uses SSL records which have a maximum size of 16kB (the max record size can be controlled by the MAX_RECORD_SIZE define in <wolfssl_root>/wolfssl/internal.h). As such, wolfSSL needs to read an entire SSL record internally before it is able to process and decrypt the record. Because of this, a call to wolfSSL_recv() will only be able to return the maximum buffer size which has been decrypted at the time of calling. There may be additional not-yet-decrypted data waiting in the internal wolfSSL receive buffer which will be retrieved and decrypted with the next call to wolfSSL_recv().

If **sz** is larger than the number of bytes in the internal read buffer, SSL_recv() will return the bytes available in the internal read buffer. If no bytes are buffered in the internal read buffer yet, a call to wolfSSL_recv() will trigger processing of the next record.

Return Values:

>0 - the number of bytes read upon success.

0 - will be returned upon failure. This may be caused by a either a clean (close notify alert) shutdown or just that the peer closed the connection. Call wolfSSL_get_error() for the specific error code.

SSL_FATAL_ERROR - will be returned upon failure when either an error occurred or, when using non-blocking sockets, the SSL_ERROR_WANT_READ or SSL_ERROR_WANT_WRITE error was received and and the application needs to call wolfSSL_recv() again. Use wolfSSL_get_error() to get a specific error code.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

data - buffer where wolfSSL_recv() will place data read.

sz - number of bytes to read into **data**.

flags - the recv flags to use for the underlying recv operation.

Example:

```
WOLFSSL* ssl = 0;
char reply[1024];
int flags = ... ;
...

input = wolfSSL_recv(ssl, reply, sizeof(reply), flags);
if (input > 0) {
      /*"input" number of bytes returned into buffer "reply"*/
}
```

See Also:
wolfSSL_read
wolfSSL_write
wolfSSL_peek
wolfSSL_pending

wolfSSL_send

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_send(WOLFSSL* ssl, const void* data, int sz, int flags);

Description:

This function writes **sz** bytes from the buffer, **data**, to the SSL connection, **ssl**, using the specified **flags** for the underlying write operation.

If necessary wolfSSL_send() will negotiate an SSL/TLS session if the handshake has not already been performed yet by wolfSSL_connect() or wolfSSL_accept().

wolfSSL_send() works with both blocking and non-blocking I/O. When the underlying I/O is non-blocking, wolfSSL_send() will return when the underlying I/O could not satisfy the needs of wolfSSL_send to continue. In this case, a call to wolfSSL_get_error() will yield either **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**. The calling process must then repeat the call to wolfSSL_send() when the underlying I/O is ready.

If the underlying I/O is blocking, wolfSSL_send() will only return once the buffer **data** of size **sz** has been completely written or an error occurred.

Return Values:

>0 - the number of bytes written upon success.

0 - will be returned upon failure. Call wolfSSL_get_error() for the specific error code.

SSL_FATAL_ERROR - will be returned upon failure when either an error occurred or, when using non-blocking sockets, the SSL_ERROR_WANT_READ or SSL_ERROR_WANT_WRITE error was received and and the application needs to call wolfSSL_send() again. Use wolfSSL_get_error() to get a specific error code.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

data - data buffer to send to peer.

sz - size, in bytes, of **data** to be sent to peer.

flags - the send flags to use for the underlying send operation.

Example:

```
WOLFSSL* ssl = 0;
char msg[64] = “hello wolfssl!”;
int msgSz = (int)strlen(msg);
int flags = ... ;
...

input = wolfSSL_send(ssl, msg, msgSz, flags);
if (input != msgSz) {
      // wolfSSL_send() failed
}
```

See Also:
wolfSSL_write
wolfSSL_read
wolfSSL_recv

wolfSSL_write

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_write(WOLFSSL* ssl, const void* data, int sz);

Description:
This function writes **sz** bytes from the buffer, **data**, to the SSL connection, **ssl**.

If necessary, wolfSSL_write() will negotiate an SSL/TLS session if the handshake has not already been performed yet by wolfSSL_connect() or wolfSSL_accept().

wolfSSL_write() works with both blocking and non-blocking I/O. When the underlying I/O is non-blocking, wolfSSL_write() will return when the underlying I/O could not satisfy the needs of wolfSSL_write() to continue. In this case, a call to wolfSSL_get_error() will yield either **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**. The calling process must then repeat the call to wolfSSL_write() when the underlying I/O is ready.

If the underlying I/O is blocking, wolfSSL_write() will only return once the buffer **data** of size **sz** has been completely written or an error occurred.

Return Values:

>0 - the number of bytes written upon success.

0 - will be returned upon failure. Call wolfSSL_get_error() for the specific error code.

SSL_FATAL_ERROR - will be returned upon failure when either an error occurred or, when using non-blocking sockets, the SSL_ERROR_WANT_READ or SSL_ERROR_WANT_WRITE error was received and and the application needs to call wolfSSL_write() again. Use wolfSSL_get_error() to get a specific error code.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

data - data buffer which will be sent to peer.

sz - size, in bytes, of data to send to the peer (**data**).

Example:

```
WOLFSSL* ssl = 0;
char msg[64] = “hello wolfssl!”;
int msgSz = (int)strlen(msg);
int flags;
int ret;
...

ret = wolfSSL_write(ssl, msg, msgSz);
if (ret <= 0) {
      /*wolfSSL_write() failed, call wolfSSL_get_error()*/
}
```

See wolfSSL examples (client, server, echoclient, echoserver) for more more detailed examples of wolfSSL_write().

See Also:
wolfSSL_send
wolfSSL_read
wolfSSL_recv

wolfSSL_writev

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_writev(WOLFSSL* ssl, const struct iovec* iov, int iovcnt);

Description:

Simulates writev semantics but doesn't actually do block at a time because of SSL_write() behavior and because front adds may be small. Makes porting into software that uses writev easier.

Return Values:

>0 - the number of bytes written upon success.

0 - will be returned upon failure. Call wolfSSL_get_error() for the specific error code.

MEMORY_ERROR will be returned if a memory error was encountered.

SSL_FATAL_ERROR - will be returned upon failure when either an error occurred or, when using non-blocking sockets, the SSL_ERROR_WANT_READ or SSL_ERROR_WANT_WRITE error was received and and the application needs to call wolfSSL_write() again. Use wolfSSL_get_error() to get a specific error code.

Parameters:

ssl - pointer to the SSL session, created with wolfSSL_new().

iov - array of I/O vectors to write

iovcnt - number of vectors in **iov** array.

Example:

```
WOLFSSL* ssl = 0;
char *bufA = “hello\n”;
char *bufB = “hello world\n”;
int iovcnt;
struct iovec iov[2];

iov[0].iov_base = buffA;
```

```
iov[0].iov_len = strlen(buffA);
iov[1].iov_base = buffB;
iov[1].iov_len = strlen(buffB);
iovcnt = 2;
...

ret = wolfSSL_writev(ssl, iov, iovcnt);
/*wrote "ret" bytes, or error if <= 0.*/
```

See Also:
wolfSSL_write

wolfSSL_SESSION_get_peer_chain

Synopsis:
#include <wolfssl/ssl.h>

WOLFSSL_X509_CHAIN*
wolfSSL_SESSION_get_peer_chain(WOLFSSL_SESSION* session);

Description:
Returns the peer certificate chain from the WOLFSSL_SESSION struct.

Return Values:
A **pointer** to a WOLFSSL_X509_CHAIN structure that contains the peer certification chain.

Parameters:

session - a pointer to a WOLFSSL_SESSION structure.

Example:

```
WOLFSSL_SESSION* session;
WOLFSSL_X509_CHAIN* chain;
...
chain = wolfSSL_SESSION_get_peer_chain(session);
if(!chain){
      /*There was no chain. Failure case. */
}
```

See Also:
get_locked_session_stats

wolfSSL_GetSessionAtIndex
wolfSSL_GetSessionIndex
AddSession

wolfSSL_get_session_cache_memsize

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_get_session_cache_memsize(void);

Description:
This function returns how large the session cache save buffer should be.

Return Values:
This function returns an **integer** that represents the size of the session cache save buffer.

Parameters:

This function has no parameters.

Example:

```
int sz = /*Minimum size for error checking*/;
...
if(sz < wolfSSL_get_session_cache_memsize()){
      /*Memory buffer is too small*/
}
```

See Also:
wolfSSL_memrestore_session_cache

wolfSSL_set_SessionTicket

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_set_SessionTicket(WOLFSSL* ssl, byte* buf, word32 bufSz);

Description:

This function sets the **ticket** member of the **WOLFSSL_SESSION** structure within the WOLFSSL struct. The buffer passed into the function is copied to memory.

Return Values:

SSL_SUCCESS - returned on successful execution of the function. The function returned without errors.

BAD_FUNC_ARG - returned if the WOLFSSL structure is NULL. This will also be thrown if the **buf** argument is NULL but the **bufSz** argument is not zero.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

buf - a byte pointer that gets loaded into the ticket member of the session structure.

bufSz - a word32 type that represents the size of the buffer.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
byte* buffer; /*File to load*/
word32 bufSz; /*size of buffer*/
...
if(wolfSSL_KeepArrays(ssl, buffer, bufSz) != SSL_SUCCESS){
      /*There was an error loading the buffer to memory. */
}
```

See Also:

wolfSSL_set_SessionTicket_cb

nwolfSSL_GetSessionAtIndex

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_GetSessionAtIndex(int idx, WOLFSSL_SESSION* session);

Description:

This function gets the session at specified index of the session cache and copies it into memory. The WOLFSSL_SESSION structure holds the session information.

Return Values:

SSL_SUCCESS - returned if the function executed successfully and no errors were thrown.

BAD_MUTEX_E - returned if there was an unlock or lock mutex error.

SSL_FAILURE - returned if the function did not execute successfully.

Parameters:

idx - an int type representing the session index.

session - a pointer to the WOLFSSL_SESSION structure.

Example:

```
int idx; /*The index to locate the session. */
WOLFSSL_SESSION* session;  /*Buffer to copy to. */
...
if(wolfSSL_GetSessionAtIndex(idx, session) != SSL_SUCCESS){
      /*Failure case. */
}
```

See Also:

UnLockMutex
LockMutex
wolfSSL_GetSessionIndex

wolfSSL_GetSessionIndex

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_GetSessionIndex(WOLFSSL* ssl);

Description:

This function gets the session index of the WOLFSSL structure.

Return Values:
The function returns an int type representing the **sessionIndex** within the WOLFSSL struct.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = WOLFSSL_new(ctx);
...
int sesIdx = wolfSSL_GetSessionIndex(ssl);

if(sesIdx < 0 || sesIdx > sizeof(ssl->sessionIndex)/sizeof(int)){
      /* You have an out of bounds index number and something is not
            right. */
}
```

See Also:
wolfSSL_GetSessionAtIndex

wolfSSL_save_session_cache

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_save_session_cache(const char* fname);

Description:
This function persists the session cache to file. It doesn't use memsave because of additional memory use.

Return Values:
SSL_SUCCESS - returned if the function executed without error. The session cache has been written to a file.

SSL_BAD_FILE - returned if **fname** cannot be opened or is otherwise corrupt.

FWRITE_ERROR - returned if XFWRITE failed to write to the file.

BAD_MUTEX_E - returned if there was a mutex lock failure.

Parameters:

fname - is a constant char pointer that points to a file for writing.

Example:

```
const char* fname;
...
if(wolfSSL_save_session_cache(fname) != SSL_SUCCESS){
      /*Fail to write to file. */
}
```

See Also:

XFWRITE
wolfSSL_restore_session_cache
wolfSSL_memrestore_session_cache

wolfSSL_memrestore_session_cache

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_memrestore_session_cache(const void* mem, int sz);

Description:

This function restores the persistent session cache from memory.

Return Values:

SSL_SUCCESS - returned if the function executed without an error.

BUFFER_E - returned if the memory buffer is too small.

BAD_MUTEX_E - returned if the session cache mutex lock failed.

CACHE_MATCH_ERROR - returned if the session cache header match failed.

Parameters:

mem - a constant void pointer containing the source of the restoration.

sz - an integer representing the size of the memory buffer.

Example:

```
const void* memoryFile;
int szMf;
...
if(wolfSSL_memrestore_session_cache(memoryFile, szMf) != SSL_SUCCESS){
      /*Failure case. SSL_SUCCESS was not returned. */
}
```

See Also:
wolfSSL_save_session_cache

wolfSSL_PrintSessionStats

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_PrintSessionStats(void);

Description:
This function prints the statistics from the session.

Return Values:
SSL_SUCCESS - returned if the function and subroutines return without error. The session stats have been successfully retrieved and printed.

BAD_FUNC_ARG - returned if the subroutine wolfSSL_get_session_stats() was passed an unacceptable argument.

BAD_MUTEX_E - returned if there was a mutex error in the subroutine.

Parameters:
This function takes no parameters.

Example:

```
/*You will need to have a session object to retrieve stats from. */
if(wolfSSL_PrintSessionStats(void) != SSL_SUCCESS     ){
      /*Did not print session stats*/
      }
```

See Also:
wolfSSL_get_session_stats

wolfSSL_restore_session_cache

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_restore_session_cache(WOLFSSL* ssl);

Description:
This function restores the persistent session cache from file. It does not use memstore because of additional memory use.

Return Values:
SSL_SUCCESS - returned if the function executed without error.

SSL_BAD_FILE - returned if the file passed into the function was corrupted and could not be opened by XFOPEN.

FREAD_ERROR - returned if the file had a read error from XFREAD.

CACHE_MATCH_ERROR - returned if the session cache header match failed.

BAD_MUTEX_E - returned if there was a mutex lock failure.

Parameters:

fname - a constant char pointer file input that will be read.

Example:

```
const char *fname;
...
if(wolfSSL_restore_session_cache(fname) != SSL_SUCCESS){
/*Failure case. The function did not return SSL_SUCCESS. */
}
```

See Also:
XFREAD
XFOPEN

wolfSSL_get_session_stats

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_get_session_stats(word32* active, word32 *total, word32* peak,
word32* maxSessions);

Description:
This function gets the statistics for the session.

Return Values:
SSL_SUCCESS - returned if the function and subroutines return without error. The session stats have been successfully retrieved and printed.

BAD_FUNC_ARG - returned if the subroutine wolfSSL_get_session_stats() was passed an unacceptable argument.

BAD_MUTEX_E - returned if there was a mutex error in the subroutine.

Parameters:

active - a word32 pointer representing the total current sessions.

total - a word32 pointer representing the total sessions.

peak - a word32 pointer representing the peak sessions.

maxSessions - a word32 pointer representing the maximum sessions.

Example:

```
int wolfSSL_PrintSessionStats(void){
…
ret = wolfSSL_get_session_stats(&totalSessionsNow, &totalSessionsSeen, &peak,
                                    &maxSessions);
…
return ret;
```

See Also:
get_locked_session_stats
wolfSSL_PrintSessionStats

wolfSSL_session_reused

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_session_reused(WOLFSSL* ssl);

Description:
This function returns the **resuming** member of the **options** struct. The flag indicates whether or not to reuse a session. If not, a new session must be established.

Return Values:
This function returns an int type held in the **Options** structure representing the flag for session reuse.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
…
if(!wolfSSL_session_reused(sslResume)){
      /*No session reuse allowed. */
}
```

See Also:

wolfSSL_SESSION_free
wolfSSL_GetSessionIndex
wolfSSL_memsave_session_cache

wolfSSL_memsave_session_cache

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_memsave_session_cache(void* mem, int sz);

Description:
This function persists session cache to memory.

Return Values:
SSL_SUCCESS - returned if the function executed without error. The session cache has been successfully persisted to memory.

BAD_MUTEX_E - returned if there was a mutex lock error.

BUFFER_E - returned if the buffer size was too small.

Parameters:

mem - a void pointer representing the destination for the memory copy, XMEMCPY().

sz - an int type representing the size of **mem**.

Example:

```
void* mem;
int sz; /*Max size of the memory buffer. */
…
if(wolfSSL_memsave_session_cache(mem, sz) != SSL_SUCCESS){
      /*Failure case, you did not persist the session cache to memory */
}
```

See Also:
XMEMCPY
wolfSSL_get_session_cache_memsize

wolfSSL_SetIO_NetX

Synopsis:

#include <wolfssl/ssl.h>

void wolfSSL_SetIO_NetX(WOLFSSL* ssl, NX_TCP_SOCKET* nxSocket,
ULONG waitOption);

Description:

This function sets the **nxSocket** and **nxWait** members of the **nxCtx** struct within the WOLFSSL structure.

Return Values:

This function has no return value.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

nxSocket - a pointer to type NX_TCP_SOCKET that is set to the **nxSocket** member of the **nxCTX** structure.

waitOption - a ULONG type that is set to the **nxWait** member of the **nxCtx** structure.

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
NX_TCP_SOCKET* nxSocket; /*Initialize */
ULONG waitOption; /*Initialize */
…
if(ssl != NULL || nxSocket != NULL || waitOption <= 0){
      wolfSSL_SetIO_NetX(ssl, nxSocket, waitOption);
} else {
      /*You need to pass in good parameters. */
}
```

See Also:

set_fd
NetX_Send
NetX_Receive

wolfSSL_GetIOReadCtx

Synopsis:

#include <wolfssl/ssl.h>

void* wolfSSL_GetIOReadCtx(WOLFSSL* ssl);

Description:

This function returns the IOCB_ReadCtx member of the WOLFSSL struct.

Return Values:

This function returns a void pointer to the **IOCB_ReadCtx** member of the WOLFSSL structure.

NULL - returned if the WOLFSSL struct is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
void* ioRead;
...
ioRead = wolfSSL_GetIOReadCtx(ssl);
if(ioRead == NULL){
      /*Failure case. The ssl object was NULL. */
}
```

See Also:

wolfSSL_GetIOWriteCtx
wolfSSL_SetIOReadFlags
wolfSSL_SetIOWriteCtx
wolfSSL_SetIOReadCtx
wolfSSL_SetIOSend

wolfSSL_GetIOWriteCtx

Synopsis:

#include <wolfssl/ssl.h>

void* wolfSSL_GetIOWriteCtx(WOLFSSL* ssl);

Description:
This function returns the IOCB_WriteCtx member of the WOLFSSL structure.

Return Values:
This function returns a void pointer to the **IOCB_WriteCtx** member of the WOLFSSL structure.

NULL - returned if the WOLFSSL struct is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL* ssl;
void* ioWrite;
...
ioWrite = wolfSSL_GetIOWriteCtx(ssl);
if(ioWrite == NULL){
      /*The funciton returned NULL. */
}
```

See Also:
wolfSSL_GetIOReadCtx
wolfSSL_SetIOWriteCtx
wolfSSL_SetIOReadCtx
wolfSSL_SetIOSend

wolfSSL_Rehandshake

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_Rehandshake(WOLFSSL* ssl);

Description:
This function executes a secure renegotiation handshake; this is user forced as wolfSSL

discourages this functionality.

Return Values:

SSL_SUCCESS - returned if the function executed without error.

BAD_FUNC_ARG - returned if the WOLFSSL structure was NULL or otherwise if an unacceptable argument was passed in a subroutine.

SECURE_RENEGOTIATION_E - returned if there was an error with renegotiating the handshake.

SSL_FATAL_ERROR - returned if there was an error with the server or client configuration and the renegotiation could not be completed. See wolfSSL_negotiate().

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL* ssl = wolfSSL_new(ctx);
...
if(wolfSSL_Rehandshake(ssl) != SSL_SUCCESS){
      /*There was an error and the rehandshake is not successful. */
 }
```

See Also:

wolfSSL_negotiate
wc_InitSha512
wc_InitSha384
wc_InitSha256
wc_InitSha
wc_InitMd5

wolfSSL_UseSecureRenegotiation

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_UseSecureRenegotiation(WOLFSSL* ssl)

Description:

This function forces secure renegotiation for the supplied WOLFSSL structure. This is not recommended.

Return Values:

SSL_SUCCESS: Successfully set secure renegotiation.

BAD_FUNC_ARG: Returns error if ssl is null.

MEMORY_E: Returns error if unable to allocate memory for secure renegotiation.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
wolfSSL_Init();
WOLFSSL_CTX* ctx;
WOLFSSL* ssl;
WOLFSSL_METHOD method = /* Some wolfSSL method */
ctx = wolfSSL_CTX_new(method);
ssl = wolfSSL_new(ctx);

if(wolfSSL_UseSecureRenegotiation(ssl) != SSL_SUCCESS)
{
    /* Error setting secure renegotiation */
}
```

See Also:

TLSX_Find
TLSX_UseSecureRenegotiation

wolfSSL_UseSessionTicket

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_UseSessionTicket(WOLFSSL* ssl)

Description:

Force provided **WOLFSSL** structure to use session ticket. The constant HAVE_SESSION_TICKET should be defined and the constant NO_WOLFSSL_CLIENT should not be defined to use this function.

Return Values:

SSL_SUCCESS: Successfully set use session ticket.

BAD_FUNC_ARG: Returned if *ssl* is null.

MEMORY_E: Error allocating memory for setting session ticket.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
wolfSSL_Init();

WOLFSSL_CTX* ctx;

WOLFSSL* ssl;

WOLFSSL_METHOD method = /* Some wolfSSL method */

ctx = wolfSSL_CTX_new(method);
ssl = wolfSSL_new(ctx);

if(wolfSSL_UseSessionTicket(ssl) != SSL_SUCCESS)

{

    /* Error setting session ticket */

}
```

See Also:

TLSX_UseSessionTicket

wolfSSL_get_current_cipher_suite

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_get_current_cipher_suite(WOLFSSL* ssl)

Description:

Returns the current cipher suit an ssl session is using.

Return Values:

ssl->options.cipherSuite: An integer representing the current cipher suite.

0: The ssl session provided is null.

Parameters:

ssl - The SSL session to check.

Example:

```
wolfSSL_Init();

WOLFSSL_CTX* ctx;

WOLFSSL* ssl;

WOLFSSL_METHOD method = /* Some wolfSSL method */

ctx = wolfSSL_CTX_new(method);
ssl = wolfSSL_new(ctx);

if(wolfSSL_get_current_cipher_suite(ssl) == 0)

{

    /* Error getting cipher suite */

}
```

See Also:

wolfSSL_CIPHER_get_name
wolfSSL_get_current_cipher
wolfSSL_get_cipher_list

wolfSSL_get_cipher_list

Synopsis:

#include <wolfssl/ssl.h>

char* wolfSSL_get_cipher_list(int priority)

Description:

Get the name of cipher at priority level passed in.

Return Values:

string: Success

0: Priority is either out of bounds or not valid.

Parameters:

priority - Integer representing the priority level of a cipher.

Example:

```
printf("The cipher at 1 is %s", wolfSSL_get_cipher_list(1));
```

See Also:

wolfSSL_CIPHER_get_name
wolfSSL_get_current_cipher

wolfSSL_isQSH

Synopsis:

#include <wolfssl/ssl.h>

wolfSSL_isQSH(WOLFSSL* ssl)

Description:

Checks if QSH is used in the supplied SSL session.

Return Values:

0: Not used

1: Is used

Parameters:

ssl - Pointer to the SSL session to check.

Example:

```
wolfSSL_Init();
WOLFSSL_CTX* ctx;
WOLFSSL* ssl;
WOLFSSL_METHOD method = /* Some wolfSSL method */
ctx = wolfSSL_CTX_new(method);
ssl = wolfSSL_new(ctx);

if(wolfSSL_isQSH(ssl) == 1)
{
    /* SSL is using QSH. */
}
```

See Also:

wolfSSL_UseSupportedQSH

wolfSSL_get_version

Synopsis:

#include <wolfssl/ssl.h>

const char* wolfSSL_get_version(WOLFSSL* ssl)

Description:

Returns the SSL version being used as a string.

Return Values:

"SSLv3": Using SSLv3

"TLSv1": Using TLSv1

"TLSv1.1": Using TLSv1.1

"TLSv1.2": Using TLSv1.2

"TLSv1.3": Using TLSv1.3

"DTLS": Using DTLS

"DTLSv1.2": Using DTLSv1.2

"unknown": There was a problem determining which version of TLS being used.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
wolfSSL_Init();

WOLFSSL_CTX* ctx;

WOLFSSL* ssl;

WOLFSSL_METHOD method = /* Some wolfSSL method */

ctx = wolfSSL_CTX_new(method);
ssl = wolfSSL_new(ctx);

printf(wolfSSL_get_version("Using version: %s", ssl));
```

See Also:

wolfSSL_lib_version

wolfSSL_get_ciphers

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_get_ciphers(char* buf, int len);

Description:
This function gets the ciphers enabled in wolfSSL.

Return Values:
SSL_SUCCESS - returned if the function executed without error.

BAD_FUNC_ARG - returned if the **buf** parameter was NULL or if the **len** argument was less than or equal to zero.

BUFFER_E - returned if the buffer is not large enough and will overflow.

Parameters:

buf - a char pointer representing the buffer.

len - the length of the buffer.

Example:

```
static void ShowCiphers(void){
      char* ciphers;  /*initialize*/
      int ret = wolfSSL_get_ciphers(ciphers, (int)sizeof(ciphers));

      if(ret == SSL_SUCCES){
            printf("%s\n", ciphers);
      }
}
```

See Also:
GetCipherNames
wolfSSL_get_cipher_list
ShowCiphers

wolfSSL_get_verify_depth

Synopsis:

#include <wolfssl/ssl.h>

long wolfSSL_get_verify_depth(WOLFSSL* ssl);

Description:

This function returns the maximum chain depth allowed, which is 9 by default, for a valid session i.e. there is a non-null session object (ssl).

Return Values:

MAX_CHAIN_DEPTH - returned if the WOLFSSL_CTX structure is not NULL. By default the value is 9.

BAD_FUNC_ARG - returned if the WOLFSSL_CTX structure is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
...
long sslDep = wolfSSL_get_verify_depth(ssl);

if(sslDep > EXPECTED){
      /*The verified depth is greater than what was expected*/
} else {
      /*The verified depth is smaller or equal to the expected value */
}
```

See Also:

wolfSSL_CTX_get_verify_depth

wolfSSL_get_cipher

Synopsis:

#include <wolfssl/ssl.h>

const char* wolfSSL_get_cipher(WOLFSSL* ssl);

Description:
This function matches the cipher suite in the SSL object with the available suites.

Return Values:
This function returns the **string** value of the suite matched. It will return “None” if there are no suites matched.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
#ifdef WOLFSSL_DTLS
…

/*make sure a valid suite is used */
if(wolfSSL_get_cipher(ssl) == NULL){
      WOLFSSL_MSG(“Can not match cipher suite imported”);
      return MATCH_SUITE_ERROR;
}
…
#endif /*WOLFSSL_DTLS */
```

See Also:
wolfSSL_CIPHER_get_name
wolfSSL_get_current_cipher

wolfSSL_CIPHER_get_name

Synopsis:
#include <wolfssl/ssl.h>

const char* wolfSSL_CIPHER_get_name(const WOLFSSL_CIPHER* cipher);

Description:
This function matches the cipher suite in the SSL object with the available suites and returns the string representation.

Return Values:

This function returns the **string** representation of the matched cipher suite. It will return "None" if there are no suites matched.

Parameters:

cipher - a constant pointer to a WOLFSSL_CIPHER structure.

Example:

```
WOLFSSL* ssl;

/*gets cipher name in the format DHE_RSA ...*/
const char* wolfSSL_get_cipher_name_internal(WOLFSSL* ssl){
      WOLFSSL_CIPHER* cipher;
      const char* fullName;
…
      cipher = wolfSSL_get_curent_cipher(ssl);
      fullName = wolfSSL_CIPHER_get_name(cipher);

      if(fullName){
            /*sanity check on returned cipher*/
      }
```

See Also:

wolfSSL_get_cipher
wolfSSL_get_current_cipher
wolfSSL_get_cipher_name_internal
wolfSSL_get_cipher_name

wolfSSL_get_cipher_name

Synopsis:

#include <wolfssl/ssl.h>

const char* wolfSSL_get_cipher_name(WOLFSSL* ssl);

Description:

This function gets the cipher name in the format DHE-RSA by passing through argument to wolfSSL_get_cipher_name_internal.

Return Values:
This function returns the **string** representation of the cipher suite that was matched.

NULL - error or cipher not found.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
…
char* cipherS = wolfSSL_get_cipher_name(ssl);

if(cipher == NULL){
      /*There was not a cipher suite matched */
} else {
      /*There was a cipher suite matched*/
      printf(“%s\n”, cipherS);
}
```

See Also:
wolfSSL_CIPHER_get_name
wolfSSL_get_current_cipher
wolfSSL_get_cipher_name_internal

wolfSSL_get_current_cipher

Synopsis:
#include <wolfssl/ssl.h>

WOLFSSL_CIPHER* wolfSSL_get_current_cipher(WOLFSSL* ssl);

Description:
This function returns a pointer to the current cipher in the ssl session.

Return Values:
The function returns the **address** of the cipher member of the WOLFSSL struct. This is a pointer to the WOLFSSL_CIPHER structure.

NULL - returned if the WOLFSSL structure is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
…
WOLFSSL_CIPHER* cipherCurr = wolfSSL_get_current_cipher;

if(!cipherCurr){
      /*Failure case. */
} else {
      /*The cipher was returned to cipherCurr */
}
```

See Also:
wolfSSL_get_cipher
wolfSSL_get_cipher_name_internal
wolfSSL_get_cipher_name

wolfSSL_get_SessionTicket

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_get_SessionTicket(WOLFSSL* ssl, byte* buf, word32* bufSz);

Description:
This function copies the ticket member of the Session structure to the buffer.

Return Values:
SSL_SUCCESS - returned if the function executed without error.

BAD_FUNC_ARG - returned if one of the arguments was NULL or if the bufSz argument was 0.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

buf - a byte pointer representing the memory buffer.

bufSz - a word32 pointer representing the buffer size.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
byte* buf;
word32 bufSz;  /*Initialize with buf size*/
…
if(wolfSSL_get_SessionTicket(ssl, buf, bufSz) <= 0){
      /*Nothing was written to the buffer*/
} else {
      /*the buffer holds the content from ssl->session.ticket */
}
```

See Also:
wolfSSL_UseSessionTicket
wolfSSL_set_SessionTicket

wolfSSL_lib_version_hex

Synopsis:
#include <wolfssl/ssl.h>

word32 wolfSSL_lib_version_hex(void);

Description:
This function returns the current library version in hexadecimal notation.

Return Values:
LILBWOLFSSL_VERSION_HEX - returns the hexidecimal version defined in wolfssl/version.h.

Parameters:

This function does not take any parameters.

Example:

```
word32 libV;
libV = wolfSSL_lib_version_hex();

if(libV != EXPECTED_HEX){
      /*How to handle an unexpected value*/
} else {
      /*The expected result for libV */
}
```

See Also:
wolfSSL_lib_version

wolfSSL_SNI_Status

Synopsis:
#include <wolfssl/ssl.h>

byte wolfSSL_SNI_Status(WOLFSSL* ssl, byte type);

Description:
This function gets the status of an SNI object.

Return Values:
This function returns the **byte** value of the SNI struct's status member if the SNI is not NULL.

0 - if the SNI object is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

type - the SNI type.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
```

```
WOLFSSL* ssl = wolfSSL_new(ctx);
…
#define AssertIntEQ(x, y) AssertInt(x, y, ==, !=)
…
Byte type = WOLFSSL_SNI_HOST_NAME;
char* request = (char*)&type;
AssertIntEQ(WOLFSSL_SNI_NO_MATCH, wolfSSL_SNI_Status(ssl, type));
…
```

See Also:

TLSX_SNI_Status
TLSX_SNI_find
TLSX_Find

wolfSSL_get_alert_history

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_get_alert_history(WOLFSSL* ssl, WOLFSSL_ALERT_HISTORY *h);

Description:

This function gets the alert history.

Return Values:

SSL_SUCCESS - returned when the function completed successfully. Either there was alert history or there wasn't, either way, the return value is SSL_SUCCESS.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

h - a pointer to a WOLFSSL_ALERT_HISTORY structure that will hold the WOLFSSL struct's **alert_history** member's value.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
WOLFSSL_ALERT_HISTORY* h;
...
```

```
wolfSSL_get_alert_history(ssl, h);
/*  h now has a copy of the ssl->alert_history  contents  */
```

See Also:

wolfSSL_get_error

wolfSSL_lib_version

Synopsis:

#include <wolfssl/ssl.h>

const char* wolfSSL_KeepArrays(void);

Description:

This function returns the current library version.

Return Values:

LIBWOLFSSL_VERSION_STRING - a const char pointer defining the version.

Parameters:

This function takes no parameters.

Example:

```
char version[MAXSIZE];
version = wolfSSL_KeepArrays();
…
if(version != ExpectedVersion){
      /*Handle the mismatch case*/
}
```

See Also:

word32_wolfSSL_lib_version_hex

wolfSSL_CTX_UseCavium

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_UseCavium(WOLFSSL_CTX* ctx, int devId)

Description:

Forces provided **WOLFSSL_CTX** to use cavium.

Return Values:

SSL_SUCCESS: Successfully set cavium.

BAD_FUNC_ARG: Returns if *ctx* is null.

Parameters:

ctx - Pointer to **WOLFSSL_CTX** to use.

devId - The value to set the **ctx->devId** to.

Example:

```
wolfSSL_Init();
WOLFSSL_CTX* ctx;
WOLFSSL_METHOD method = /* Some wolfSSL method */
ctx = wolfSSL_CTX_new(method);

if(wolfSSL_CTX_UseCavium(ctx, CAVIUM_DEV_ID) != SSL_SUCCESS)
{
    /* Error setting session ticket */
}
```

See Also:

wolfSSL_UseCavium

wolfSSL_UseCavium

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_UseCavium(WOLFSSL* ssl, int devId)

Description:

Forces provided **WOLFSSL** structure to use cavium.

Return Values:

SSL_SUCCESS: Success

BAD_FUNC_ARG: Returned if **ssl** is null.

Parameters:

ssl - Pointer to the **WOLFSSL** session. Created with wolfSSL_new()

devId - Value to set **ssl->devId** to.

Example:

```
wolfSSL_Init();

WOLFSSL_CTX* ctx;

WOLFSSL* ssl;

WOLFSSL_METHOD method = /* Some wolfSSL method */

ctx = wolfSSL_CTX_new(method);
ssl = wolfSSL_new(ctx);

if(wolfSSL_UseCavium(ssl, CAVIUM_DEV_ID) != SSL_SUCCESS)

{

    /* Error setting session ticket */

}
```

See Also:

wolfSSL_CTX_UseCavium

wolfSSL_set_jobject

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_set_jobject(WOLFSSL* ssl, void* objPtr);

Description:
This function sets the **jObjectRef** member of the WOLFSSL structure.

Return Values:
SSL_SUCCESS - returned if jObjectRef is properly set to objPtr.

SSL_FAILURE - returned if the function did not properly execute and jObjectRef is not set.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

objPtr - a void pointer that will be set to **jObjectRef.**

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = WOLFSSL_new();
void* objPtr = &obj;
...
if(wolfSSL_set_jobject(ssl, objPtr)){
      /*The success case*/
 }
```

See Also:
wolfSSL_get_jobject

wolfSSL_get_jobject

Synopsis:
#include <wolfssl/ssl.h>

void* wolfSSL_get_jobject(WOLFSSL* ssl);

Description:

This function returns the **jObjectRef** member of the WOLFSSL structure.

Return Values:

If the WOLFSSL struct is not NULL, the function returns the **jObjectRef** value.

NULL - returned if the WOLFSSL struct is NULL.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL(ctx);
...
void* jobject = wolfSSL_get_jobject(ssl);

if(jobject != NULL){
      /*Success case*/
}
```

See Also:

wolfSSL_set_jobject

wolfSSL_BIO_ctrl_pending

Synopsis:

#include <wolfssl/ssl.h>

size_t wolfSSL_BIO_ctrl_pending(WOLFSSL_BIO* bio);

Description:

Gets the number of pending bytes to read. If BIO type is BIO_BIO then is the number to read from pair. If BIO contains an SSL object then is pending data from SSL object (wolfSSL_pending(ssl)). If is BIO_MEMORY type then returns the size of memory buffer.

Return Values:

0 or greater: number of pending bytes.

Parameters:

bio - pointer to the WOLFSSL_BIO structure that has already been created

Example:

```
WOLFSSL_BIO* bio;
int pending;

bio = wolfSSL_BIO_new();


….


pending = wolfSSL_BIO_ctrl_pending(bio);
```

See Also:

wolfSSL_BIO_make_bio_pair, wolfSSL_BIO_new

wolfSSL_BIO_get_mem_ptr

Synopsis:

#include <wolfssl/ssl.h>

BIO_get_mem_ptr ->
long wolfSSL_BIO_get_mem_ptr(WOLFSSL_BIO* bio, WOLFSSL_BUF_MEM** ptr);

Description:

This is a getter function for WOLFSSL_BIO memory pointer.

Return Values:

SSL_SUCCESS: On successfully getting the pointer SSL_SUCCESS is returned (currently value of 1).

SSL_FAILURE: Returned if NULL arguments are passed in (currently value of 0).

Parameters:

bio - pointer to the WOLFSSL_BIO structure for getting memory pointer.

ptr - structure that is currently a char*. Is set to point to bio's memory.

Example:

```
WOLFSSL_BIO* bio;
WOLFSSL_BUF_MEM* pt;

// setup bio

wolfSSL_BIO_get_mem_ptr(bio, &pt);


//use pt
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_s_mem

wolfSSL_BIO_reset

Synopsis:

#include <wolfssl/ssl.h>

BIO_reset ->

int wolfSSL_BIO_reset(WOLFSSL_BIO* bio);

Description:

Resets bio to an initial state. As an example for type BIO_BIO this resets the read and write index.

Return Values:

0: On successfully resetting the bio.

-1 (WOLFSSL_BIO_ERROR): Returned on bad input or unsuccessful reset.

Parameters:

bio - WOLFSSL_BIO structure to reset.

Example:

```
WOLFSSL_BIO* bio;
// setup bio
```

```
wolfSSL_BIO_reset(bio);

//use pt
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_free

wolfSSL_ERR_load_BIO_strings

Synopsis:

#include <wolfssl/ssl.h>

ERR_load_BIO_strings ->
void wolfSSL_ERR_load_BIO_strings(void)

Description:

Do nothing. wolfSSL error string is statically defined.

Return Values:

None

Parameters:

none

Example:

```
    wolfSSL_ERR_load_BIO_strings();
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_free

wolfSSL_BIO_s_socket

Synopsis:

#include <wolfssl/ssl.h>

BIO_s_socket ->
WOLFSSL_BIO_METHOD* wolfSSL_BIO_s_socket(void);

Description:

This is used to get a BIO_SOCKET type WOLFSSL_BIO_METHOD.

Return Values:

WOLFSSL_BIO_METHOD*: pointer to a WOLFSSL_BIO_METHOD structure that is a socket type

Parameters:

None

Example:

```
WOLFSSL_BIO* bio;

bio = wolfSSL_BIO_new(wolfSSL_BIO_s_socket);
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_s_mem

wolfSSL_BIO_set_fd

Synopsis:

#include <wolfssl/ssl.h>

BIO_set_fd ->
long wolfSSL_BIO_set_fd(WOLFSSL_BIO* bio, int fd, int closeF);

Description:

Sets the file descriptor for bio to use.

Return Values:

Returns SSL_SUCCESS (1).

Parameters:

bio - WOLFSSL_BIO structure to set fd.

fd - file descriptor to use.

closeF - flag for behavior when closing fd.

Example:

```
WOLFSSL_BIO* bio;
int fd;
// setup bio
wolfSSL_BIO_set_fd(bio, fd, BIO_NOCLOSE);
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_free

wolfSSL_BIO_set_write_buf_size

Synopsis:

#include <wolfssl/ssl.h>

BIO_set_write_buf_size ->

int wolfSSL_BIO_set_write_buf_size(WOLFSSL_BIO* bio, long size;

Description:

This is used to set the size of write buffer for a WOLFSSL_BIO. If write buffer has been previously set this function will free it when resetting the size. It is similar to wolfSSL_BIO_reset in that it resets read and write indexes to 0.

Return Values:

SSL_SUCCESS: On successfully setting the write buffer.
SSL_FAILURE: If an error case was encountered.

Parameters:

bio - WOLFSSL_BIO structure to set fd.

size - size of buffer to allocate.

Example:

```
WOLFSSL_BIO* bio;
int ret;

bio = wolfSSL_BIO_new(wolfSSL_BIO_s_mem());
ret = wolfSSL_BIO_set_write_buf_size(bio, 15000);
// check return value
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_s_mem
wolfSSL_BIO_new, wolfSSL_BIO_free

wolfSSL_BIO_make_bio_pair

Synopsis:

#include <wolfssl/ssl.h>

BIO_make_bio_pair ->
int wolfSSL_BIO_make_bio_pair(WOLFSSL_BIO* b1, WOLFSSL_BIO* b2);

Description:

This is used to pair two bios together. A pair of bios acts similar to a two way pipe writing to one can be read by the other and vice versa. It is expected that both bios be in the same thread, this function is not thread safe. Freeing one of the two bios removes both from being paired. If a write buffer size was not previously set for either of the bios it is set to a default size of 17000 (WOLFSSL_BIO_SIZE) before being paired.

Return Values:

SSL_SUCCESS: On successfully pairing the two bios.
SSL_FAILURE: If an error case was encountered.

Parameters:

b1 - WOLFSSL_BIO structure to set pair.

b2 - second WOLFSSL_BIO structure to complete pair.

Example:

```
WOLFSSL_BIO* bio;
WOLFSSL_BIO* bio2;
int ret;

bio  = wolfSSL_BIO_new(wolfSSL_BIO_s_bio());
bio2 = wolfSSL_BIO_new(wolfSSL_BIO_s_bio());
ret = wolfSSL_BIO_make_bio_pair(bio, bio2);
// check ret value
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_s_mem

wolfSSL_BIO_new, wolfSSL_BIO_free

wolfSSL_BIO_ctrl_reset_read_request

Synopsis:

#include <wolfssl/ssl.h>

BIO_ctrl_reset_read_request ->

int wolfSSL_BIO_ctrl_reset_read_request(WOLFSSL_BIO* bio);

Description:

This is used to set the read request flag back to 0.

Return Values:

SSL_SUCCESS: On successfully setting value.

SSL_FAILURE: If an error case was encountered.

Parameters:

bio - WOLFSSL_BIO structure to set read request flag.

Example:

```
WOLFSSL_BIO* bio;
int ret;

...
ret = wolfSSL_BIO_ctrl_reset_read_request(bio);
// check ret value
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_s_mem

wolfSSL_BIO_new, wolfSSL_BIO_free

wolfSSL_BIO_nread

Synopsis:

#include <wolfssl/ssl.h>

BIO_nread ->
int wolfSSL_BIO_nread(WOLFSSL_BIO* bio, char** buf, int num);

Description:

This is used to get a buffer pointer for reading from. The internal read index is advanced by the number returned from the function call with buf being pointed to the beginning of the buffer to read from. In the case that less bytes are in the read buffer than the value requested with num the lesser value is returned. Reading past the value returned can result in reading out of array bounds.

Return Values:

0 or greater: on success return the number of bytes to read

-1: on error case with nothing to read return -1 (WOLFSSL_BIO_ERROR)

Parameters:

bio - WOLFSSL_BIO structure to read from.

buf - pointer to set at beginning of read array.

num -number of bytes to try and read.

Example:

```
WOLFSSL_BIO* bio;

char* bufPt;

int ret;


// set up bio

ret = wolfSSL_BIO_nread(bio, &bufPt, 10); // try to read 10 bytes
// handle negative ret check
// read ret bytes from bufPt
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_nwrite

wolfSSL_BIO_nread0

Synopsis:

#include <wolfssl/ssl.h>

BIO_nread ->
int wolfSSL_BIO_nread0(WOLFSSL_BIO* bio, char** buf);

Description:

This is used to get a buffer pointer for reading from. Unlike wolfSSL_BIO_nread the internal read index is not advanced by the number returned from the function call. Reading past the value returned can result in reading out of array bounds.

Return Values:

Greater than 0: on success return the number of bytes to read

Parameters:

bio - WOLFSSL_BIO structure to read from.

buf - pointer to set at beginning of read array.

Example:

```
WOLFSSL_BIO* bio;

char* bufPt;

int ret;


// set up bio

ret = wolfSSL_BIO_nread0(bio, &bufPt); // read as many bytes as possible
// handle negative ret check
// read ret bytes from bufPt
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_nwrite0

wolfSSL_BIO_nwrite

Synopsis:

#include <wolfssl/ssl.h>

BIO_nwrite ->
int wolfSSL_BIO_nwrite(WOLFSSL_BIO* bio, char** buf, int num);

Description:

Gets a pointer to the buffer for writing as many bytes as returned by the function. Writing more bytes to the pointer returned then the value returned can result in writing out of bounds.

Return Values:

Returns the number of bytes that can be written to the buffer pointer returned.

WOLFSSL_BIO_UNSET: -2 in the case that is not part of a bio pair

WOLFSSL_BIO_ERROR: -1 in the case that there is no more room to write to

Parameters:

bio - WOLFSSL_BIO structure to write to.

buf - pointer to buffer to write to.

num - number of bytes desired to be written.

Example:

```
WOLFSSL_BIO* bio;
char* bufPt;
int ret;

// set up bio
ret = wolfSSL_BIO_nwrite(bio, &bufPt, 10); // try to write 10 bytes
// handle negative ret check
// write ret bytes to bufPt
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_free, wolfSSL_BIO_nread

wolfSSL_BIO_puts

Synopsis:

#include <wolfssl/ssl.h>

BIO_puts ->
int wolfSSL_BIO_puts(WOLFSSL_BIO* bio, const char* data)

Description:

BIO_puts() tries to write a NUL-terminated string data to BIO bio.

Return Values:

Return the number of bytes that is successfully written.

SSL_FAILURE: o data was successfully written.

Parameters:

bio - WOLFSSL_BIO structure to write to.

data - pointer to buffer to write to.

Example:

```
WOLFSSL_BIO* bio;
char* data;
int ret;

// set up bio
ret = wolfSSL_BIO_puts (bio, &data, 10);

// handle negative ret check
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_free, wolfSSL_BIO_read

wolfSSL_BIO_set_fp

Synopsis:

#include <wolfssl/ssl.h>

BIO_set_fp ->
long wolfSSL_BIO_set_fp(WOLFSSL_BIO* bio, XFILE fp, int c);

Description:

This is used to set the internal file pointer for a BIO.

Return Values:

SSL_SUCCESS: On successfully setting file pointer.

SSL_FAILURE: If an error case was encountered.

Parameters:

bio - WOLFSSL_BIO structure to set pair.

fp - file pointer to set in bio.

c - close file behavior flag.

Example:

```
WOLFSSL_BIO* bio;

XFILE fp;

int ret;

bio  = wolfSSL_BIO_new(wolfSSL_BIO_s_file());
ret  = wolfSSL_BIO_set_fp(bio, fp, BIO_CLOSE);
// check ret value
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_s_mem, wolfSSL_BIO_get_fp
wolfSSL_BIO_new, wolfSSL_BIO_free

wolfSSL_BIO_get_fp

Synopsis:

#include <wolfssl/ssl.h>

BIO_get_fp ->
long wolfSSL_BIO_get_fp(WOLFSSL_BIO* bio, XFILE fp);

Description:

This is used to get the internal file pointer for a BIO.

Return Values:

SSL_SUCCESS: On successfully getting file pointer.

SSL_FAILURE: If an error case was encountered.

Parameters:

bio - WOLFSSL_BIO structure to set pair.

fp - file pointer to set in bio.

Example:

```
WOLFSSL_BIO* bio;

XFILE fp;

int ret;

bio  = wolfSSL_BIO_new(wolfSSL_BIO_s_file());
ret  = wolfSSL_BIO_get_fp(bio, &fp);
// check ret value
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_s_mem, wolfSSL_BIO_set_fp
wolfSSL_BIO_new, wolfSSL_BIO_free

wolfSSL_BIO_seek

Synopsis:

#include <wolfssl/ssl.h>

BIO_seek ->
int wolfSSL_BIO_seek(WOLFSSL_BIO* bio, int ofs);

Description:

This function adjusts the file pointer to the offset given. This is the offset from the head of the file.

Return Values:

0: On successfully seeking.

-1: If an error case was encountered.

Parameters:

bio - WOLFSSL_BIO structure to set.

ofs - offset into file.

Example:

```
WOLFSSL_BIO* bio;

XFILE fp;

int ret;

bio  = wolfSSL_BIO_new(wolfSSL_BIO_s_file());
ret  = wolfSSL_BIO_set_fp(bio, &fp);
// check ret value
ret  = wolfSSL_BIO_seek(bio, 3);
// check ret value
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_s_mem, wolfSSL_BIO_set_fp
wolfSSL_BIO_new, wolfSSL_BIO_free

wolfSSL_BIO_write_filename

Synopsis:

#include <wolfssl/ssl.h>

BIO_write_filename ->
int wolfSSL_BIO_write_filename(WOLFSSL_BIO* bio, char* name);

Description:

This is used to set and write to a file. WIll overwrite any data currently in the file and is set to close the file when the bio is freed.

Return Values:

SSL_SUCCESS: On successfully opening and setting file.

SSL_FAILURE: If an error case was encountered.

Parameters:

bio - WOLFSSL_BIO structure to set file.

name - name of file to write to.

Example:

```
WOLFSSL_BIO* bio;

int ret;

bio  = wolfSSL_BIO_new(wolfSSL_BIO_s_file());
ret  = wolfSSL_BIO_write_filename(bio, “test.txt”);
// check ret value
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_s_file, wolfSSL_BIO_set_fp
wolfSSL_BIO_new, wolfSSL_BIO_free

wolfSSL_BIO_get_mem_data

Synopsis:

#include <wolfssl/ssl.h>

BIO_get_mem_data ->
int wolfSSL_BIO_get_mem_data(WOLFSSL_BIO* bio, const byte** p);

Description:

This is used to set a byte pointer to the start of the internal memory buffer.

Return Values:

On success the size of the buffer is returned

SSL_FATAL_ERROR: If an error case was encountered.

Parameters:

bio - WOLFSSL_BIO structure to get memory buffer of.

p - byte pointer to set to memory buffer.

Example:

```
WOLFSSL_BIO* bio;

const byte* p;

int ret;

bio  = wolfSSL_BIO_new(wolfSSL_BIO_s_mem());
```

```
ret  = wolfSSL_BIO_get_mem_data(bio, &p);
// check ret value
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_s_mem, wolfSSL_BIO_set_fp
wolfSSL_BIO_new, wolfSSL_BIO_free

wolfSSL_BIO_get_mem_ptr

Synopsis:

#include <wolfssl/ssl.h>

BIO_get_mem_ptr ->
long wolfSSL_BIO_get_mem_ptr(WOLFSSL_BIO* bio, WOLFSSL_BUF_MEM** ptr);

Description:

This is used to get the internal memory pointer from a BIO.

Return Values:

SSL_SUCCESS: On successfully getting memory pointer.

SSL_FAILURE: If an error case was encountered.

Parameters:

bio - WOLFSSL_BIO structure to get memory pointer from.

ptr - pointer to WOLFSSL_BUF_MEM structure.

Example:

```
WOLFSSL_BIO* bio;

WOLFSSL_BUF_MEM* p;

int ret;

bio  = wolfSSL_BIO_new(wolfSSL_BIO_s_mem());
ret  = wolfSSL_BIO_get_mem_ptr(bio, &p);
// check ret value
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_s_mem, wolfSSL_BIO_set_fp
wolfSSL_BIO_new, wolfSSL_BIO_free

wolfSSL_BIO_set_mem_eof_return

Synopsis:

#include <wolfssl/ssl.h>

BIO_set_mem_eof_return ->
long wolfSSL_BIO_set_mem_eof_return(WOLFSSL_BIO* bio, int v);

Description:

This is used to set the end of file value. Common value is -1 so as not to get confused with expected positive values.

Return Values:

Returns 0

Parameters:

bio - WOLFSSL_BIO structure to set end of file value.

v - value to set in bio.

Example:

```
WOLFSSL_BIO* bio;

int ret;

bio  = wolfSSL_BIO_new(wolfSSL_BIO_s_mem());
ret  = wolfSSL_BIO_set_mem_eof_return(bio, -1);
// check ret value
```

See Also:

wolfSSL_BIO_new, wolfSSL_BIO_s_mem, wolfSSL_BIO_set_fp
wolfSSL_BIO_new, wolfSSL_BIO_free

17.9 DTLS Specific

The functions in this section are specific to using DTLS with wolfSSL.

wolfSSL_dtls

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_dtls(WOLFSSL* ssl);

Description:
This function is used to determine if the SSL session has been configured to use DTLS.

Return Values:
If the SSL session (**ssl**) has been configured to use DTLS, this function will return 1, otherwise 0.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
int ret = 0;
WOLFSSL* ssl;
...

ret = wolfSSL_dtls(ssl);
if (ret) {
      // SSL session has been configured to use DTLS
}
```

See Also:
wolfSSL_dtls_get_current_timeout
wolfSSL_dtls_get_peer
wolfSSL_dtls_got_timeout
wolfSSL_dtls_set_peer

wolfSSL_dtls_get_current_timeout

Synopsis:
#include <wolfssl/ssl.h>

wolfSSL_dtls_get_current_timeout(WOLFSSL* ssl);

Description:
This function returns the current timeout value in seconds for the WOLFSSL object. When using non-blocking sockets, something in the user code needs to decide when to check for available recv data and how long it has been waiting. The value returned by this function indicates how long the application should wait.

Return Values:
The current DTLS timeout value in seconds, or **NOT_COMPILED_IN** if wolfSSL was not built with DTLS support.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
int timeout = 0;
WOLFSSL* ssl;
...

timeout = wolfSSL_get_dtls_current_timeout(ssl);
printf("DTLS timeout (sec) = %d\n", timeout);
```

See Also:
wolfSSL_dtls
wolfSSL_dtls_get_peer
wolfSSL_dtls_got_timeout
wolfSSL_dtls_set_peer

wolfSSL_dtls_get_peer

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_dtls_get_peer(WOLFSSL* ssl, void* peer, unsigned int* peerSz);

Description:

This function gets the sockaddr_in (of size **peerSz**) of the current DTLS peer. The function will compare peerSz to the actual DTLS peer size stored in the SSL session. If the peer will fit into **peer**, the peer's sockaddr_in will be copied into **peer**, with peerSz set to the size of **peer**.

Return Values:

SSL_SUCCESS will be returned upon success.

SSL_FAILURE will be returned upon failure.

SSL_NOT_IMPLEMENTED will be returned if wolfSSL was not compiled with DTLS support.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

peer - pointer to memory location to store peer's sockaddr_in structure.

peerSz - input/output size. As input, the size of the allocated memory pointed to by **peer**. As output, the size of the actual sockaddr_in structure pointed to by **peer**.

Example:

```
int ret = 0;
WOLFSSL* ssl;
sockaddr_in addr;
...

ret = wolfSSL_dtls_get_peer(ssl, &addr, sizeof(addr));
if (ret != SSL_SUCCESS) {
      // failed to get DTLS peer
}
```

See Also:

wolfSSL_dtls_get_current_timeout
wolfSSL_dtls_got_timeout
wolfSSL_dtls_set_peer

wolfSSL_dtls

wolfSSL_dtls_got_timeout

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_dtls_got_timeout(WOLFSSL* ssl);

Description:
When using non-blocking sockets with DTLS, this function should be called on the WOLFSSL object when the controlling code thinks the transmission has timed out. It performs the actions needed to retry the last transmit, including adjusting the timeout value. If it has been too long, this will return a failure.

Return Values:

SSL_SUCCESS will be returned upon success

SSL_FATAL_ERROR will be returned if there have been too many retransmissions/timeouts without getting a response from the peer.

NOT_COMPILED_IN will be returned if wolfSSL was not compiled with DTLS support.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

See the following files for usage examples:
<wolfssl_root>/examples/client/client.c
<wolfssl_root>/examples/server/server.c

See Also:
wolfSSL_dtls_get_current_timeout
wolfSSL_dtls_get_peer
wolfSSL_dtls_set_peer
wolfSSL_dtls

wolfSSL_dtls_set_peer

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_dtls_set_peer(WOLFSSL* ssl, void* peer, unsigned int peerSz);

Description:
This function sets the DTLS peer, **peer** (sockaddr_in) with size of **peerSz**.

Return Values:

SSL_SUCCESS will be returned upon success.

SSL_FAILURE will be returned upon failure.

SSL_NOT_IMPLEMENTED will be returned if wolfSSL was not compiled with DTLS support.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

peer - pointer to peer's sockaddr_in structure.

peerSz - size of the sockaddr_in structure pointed to by **peer**.

Example:

```
int ret = 0;
WOLFSSL* ssl;
sockaddr_in addr;
...

ret = wolfSSL_dtls_set_peer(ssl, &addr, sizeof(addr));
if (ret != SSL_SUCCESS) {
      // failed to set DTLS peer
}
```

See Also:

wolfSSL_dtls_get_current_timeout
wolfSSL_dtls_get_peer
wolfSSL_dtls_got_timeout
wolfSSL_dtls

wolfSSL_dtls_set_timeout_max

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_dtls_set_timeout_max(WOLFSSL* ssl, int timeout);

Description:

This function sets the maximum dtls timeout.

Return Values:

SSL_SUCCESS - returned if the function executed without an error.

BAD_FUNC_ARG - returned if the WOLFSSL struct is NULL or if the **timeout** argument is not greater than zero or is less than the **dtls_timeout_init** member of the WOLFSSL structure.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

timeout - an int type representing the dtls maximum timeout.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
int timeout = TIMEOUTVAL;
...
int ret = wolfSSL_dtls_set_timeout_max(ssl);

if(!ret){
      /*Failed to set the max timeout*/
}
```

See Also:

wolfSSL_dtls_set_timeout_init
wolfSSL_dtls_got_timeout

wolfSSL_DTLS_SetCookieSecret

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_DTLS_SetCookieSecret(WOLFSSL* ssl, const byte* secret,
word32 secretSz);

Description:

This function sets a new dtls cookie secret.

Return Values:

0 - returned if the function executed without an error.

BAD_FUNC_ARG - returned if there was an argument passed to the function with an unacceptable value.

COOKIE_SECRET_SZ - returned if the secret size is 0.

MEMORY_ERROR - returned if there was a problem allocating memory for a new cookie secret.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

secret - a constant byte pointer representing the secret buffer.

secretSz - the size of the buffer.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
const* byte secret;
word32 secretSz; /*size of secret*/
…
```

```
if(!wolfSSL_DTLS_SetCookieSecret(ssl, secret, secretSz)){
      /*Code block for failure to set DTLS cookie secret*/
} else {
      /*Success! Cookie secret is set. */
}
```

See Also:

ForceZero
wc_RNG_GenerateBlock
XMEMCPY

wolfDTLSv1_2_client_method

Synopsis:

#include <wolfssl/ssl.h>

WOLFSSL_METHOD* wolfDTLSv1_2_client_method(void);

Description:

This function initializes the DTLS v1.2 client method.

Return Values:

This function returns a pointer to a new **WOLFSSL_METHOD** structure.

Parameters:

This function has no parameters.

Example:

```
wolfSSL_Init();

WOLFSSL_CTX* ctx = wolfSSL_CTX_new(wolfDTLSv1_2_client_method());
…

WOLFSSL* ssl = wolfSSL_new(ctx);
…
```

See Also:

wolfSSL_Init
wolfSSL_CTX_new

wolfSSL_dtls_export

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_dtls_export(WOLFSSL* ssl, unsigned char* buf, unsigned int* sz);

Description:

The wolfSSL_dtls_export() function is used to serialize a WOLFSSL session into the provided buffer. Allows for less memory overhead than using a function callback for sending a session and choice over when the session is serialized. If buffer is NULL when passed to function then sz will be set to the size of buffer needed for serializing the WOLFSSL session.

Return Values:

If successful, the amount of the buffer used will be returned.

All unsuccessful return values will be less than 0.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

buf - buffer to hold serialized session.

sz - size of buffer.

Example:

```
WOLFSSL* ssl;
int ret;
unsigned char buf[MAX];
```

```
bufSz = MAX;

...

ret = wolfSSL_dtls_export(ssl, buf, bufSz);
if (ret < 0) {
// handle error case
}
...
```

See Also:

wolfSSL_new
wolfSSL_CTX_new
wolfSSL_CTX_dtls_set_export
wolfSSL_dtls_import

wolfSSL_dtls_import

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_dtls_import(WOLFSSL* ssl, unsigned char* buf, unsigned int sz);

Description:

The wolfSSL_dtls_import() function is used to parse in a serialized session state. This allows for picking up the connection after the handshake has been completed.

Return Values:

If successful, the amount of the buffer read will be returned.

All unsuccessful return values will be less than 0.

If a version mismatch is found ie DTLS v1 and ctx was set up for DTLS v1.2 then VERSSION_ERROR is returned.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

buf - serialized session to import.

sz - size of serialized session buffer.

Example:

```
WOLFSSL* ssl;
int ret;
unsigned char buf[MAX];
bufSz = MAX;

...
//get information sent from wc_dtls_export function and place it in buf
fread(buf, 1, bufSz, input);

ret = wolfSSL_dtls_import(ssl, buf, bufSz);
if (ret < 0) {
// handle error case
}

// no wolfSSL_accept needed since handshake was already done

...
ret = wolfSSL_write(ssl) and wolfSSL_read(ssl);
...
```

See Also:
wolfSSL_new
wolfSSL_CTX_new
wolfSSL_CTX_dtls_set_export

wolfSSL_dtls_set_export

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_dtls_set_export(WOLFSSL* ssl, wc_dtls_export func);

Description:

The wolfSSL_dtls_set_export() function is used to set the callback function for exporting a session. It is allowed to pass in NULL as the parameter func to clear the export function previously stored. Used on the server side and is called immediately after handshake is completed.

Return Values:

If successful, the call will return **SSL_SUCCESS**.

If null or not expected arguments are passed in **BAD_FUNC_ARG** is returned.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

func - wc_dtls_export function to use when exporting a session.

Example:

```
int send_session(WOLFSSL* ssl, byte* buf, word32 sz, void* userCtx);

// body of send session (wc_dtls_export) that passses buf (serialized
session) to destination

WOLFSSL* ssl;
int ret;

...

ret = wolfSSL_dtls_set_export(ssl, send_session);
if (ret != SSL_SUCCESS) {
// handle error case
}

...
ret = wolfSSL_accept(ssl);
```

```
...
```

See Also:

wolfSSL_new

wolfSSL_CTX_new

wolfSSL_CTX_dtls_set_export

wolfSSL_CTX_dtls_set_export

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_dtls_set_export(WOLFSSL_CTX* ctx, wc_dtls_export func);

Description:

The wolfSSL_CTX_dtls_set_export() function is used to set the callback function for exporting a session. It is allowed to pass in NULL as the parameter func to clear the export function previously stored. Used on the server side and is called immediately after handshake is completed.

Return Values:

If successful, the call will return **SSL_SUCCESS**.

If null or not expected arguments are passed in **BAD_FUNC_ARG** is returned.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created with wolfSSL_CTX_new().

func - wc_dtls_export function to use when exporting a session.

Example:

```
int send_session(WOLFSSL* ssl, byte* buf, word32 sz, void* userCtx);

// body of send session (wc_dtls_export) that passses buf (serialized
session) to destination

WOLFSSL_CTX* ctx;
int ret;

...

ret = wolfSSL_CTX_dtls_set_export(ctx, send_session);
if (ret != SSL_SUCCESS) {
// handle error case
}

...
ret = wolfSSL_accept(ssl);
...
```

See Also:

wolfSSL_new
wolfSSL_CTX_new
wolfSSL_dtls_set_export
Static buffer use

wolfSSL_CTX_load_static_memory

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_load_static_memory(WOLFSSL_CTX** ctx, wolfSSL_method_func method, unsigned char* buf, unsigned int sz, int flag, int max);

Description:

This function is used to set aside static memory for a CTX. Memory set aside is then used for the CTX's lifetime and for any SSL objects created from the CTX. By passing in a NULL ctx pointer and a wolfSSL_method_func function the creation of the CTX itself will also use static memory. wolfSSL_method_func has the function signature of WOLFSSL_METHOD* (*wolfSSL_method_func)(void* heap);.
Passing in 0 for max makes it behave as if not set and no max concurrent use

restrictions is in place.
The flag value passed in determines how the memory is used and behavior while operating. Available flags are the following.

0 - default general memory
WOLFMEM_IO_POOL - used for input/output buffer when sending receiving messages. Overrides general memory, so all memory in buffer passed in is used for IO.
WOLFMEM_IO_FIXED - same as WOLFMEM_IO_POOL but each SSL now keeps two buffers to themselves for their lifetime.
WOLFMEM_TRACK_STATS - each SSL keeps track of memory stats while running.

Return Values:

If successful, **SSL_SUCCESS** will be returned.

All unsuccessful return values will be less than 0 or equal to **SSL_FAILURE**.

Parameters:

ctx - address of pointer to a WOLFSSL_CTX structure.

method - function to create protocol. (should be NULL if ctx is not also NULL)

buf - memory to use for all operations.

sz - size of memory buffer being passed in.

flag - type of memory.

max - max concurrent operations.

Example:

```
WOLFSSL_CTX* ctx;
WOLFSSL* ssl;
int ret;
unsigned char memory[MAX];
```

```
int memorySz = MAX;
unsigned char IO[MAX];
int IOSz = MAX;
int flag = WOLFMEM_IO_FIXED | WOLFMEM_TRACK_STATS;

...
// create ctx also using static memory, start with general memory to use
ctx = NULL:
ret = wolfSSL_CTX_load_static_memory(&ctx, wolfSSLv23_server_method_ex,
memory, memorySz, 0, MAX_CONCURRENT_HANDSHAKES);
if (ret != SSL_SUCCESS) {
      // handle error case
}


// load in memory for use with IO
ret = wolfSSL_CTX_load_static_memory(&ctx, NULL, IO, IOSz, flag,
MAX_CONCURRENT_IO);
if (ret != SSL_SUCCESS) {
      // handle error case
}
...
```

See Also:

wolfSSL_CTX_new
wolfSSL_CTX_is_static_memory
wolfSSL_is_static_memory

wolfSSL_CTX_is_static_memory

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_is_static_memory(WOLFSSL_CTX* ctx, WOLFSSL_MEM_STATS* mem_stats);

Description:

This function does not change any of the connections behavior and is used only for gathering information about the static memory usage.

Return Values:

A value of **1** is returned if using static memory for the CTX is true.

0 is returned if not using static memory.

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new().

mem_stats - structure to hold information about static memory usage.

Example:

```
WOLFSSL_CTX* ctx;
int ret;
WOLFSSL_MEM_STATS mem_stats;

...
//get information about static memory with CTX
ret = wolfSSL_CTX_is_static_memory(ctx, &mem_stats);
if (ret == 1) {
// handle case of is using static memory
 // print out or inspect elements of mem_stats
}

if (ret == 0) {
//handle case of ctx not using static memory
}
…
```

See Also:

wolfSSL_CTX_new
wolfSSL_CTX_load_static_memory
wolfSSL_is_static_memory

wolfSSL_is_static_memory

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_is_static_memory(WOLFSSL* ssl, WOLFSSL_MEM_CONN_STATS* mem_stats);

Description:

wolfSSL_is_static_memory is used to gather information about a SSL's static memory usage. The return value indicates if static memory is being used and WOLFSSL_MEM_CONN_STATS will be filled out if and only if the flag WOLFMEM_TRACK_STATS was passed to the parent CTX when loading in static memory.

Return Values:

A value of **1** is returned if using static memory for the CTX is true.

0 is returned if not using static memory.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

mem_stats - structure to contain static memory usage.

Example:

```
WOLFSSL* ssl;
int ret;
WOLFSSL_MEM_CONN_STATS mem_stats;

...

ret = wolfSSL_is_static_memory(ssl, mem_stats);
if (ret == 1) {
// handle case when is static memory
// investigate elements in mem_stats if WOLFMEM_TRACK_STATS flag
}

...
```

See Also:

wolfSSL_new

wolfSSL_CTX_is_static_memory

wolfDTLSv1_2_server_method

Synopsis:

#include <wolfssl/ssl.h>

WOLFSSL_METHOD* wolfDTLSv1_2_server_method(void);

Description:

This function creates and initializes a WOLFSSL_METHOD for the server side.

Return Values:

This function returns a WOLFSSL_METHOD pointer.

Parameters:

This function takes no parameters.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(wolfDTLSv1_2_server_method());
WOLFSSL* ssl = WOLFSSL_new(ctx);
…
```

See Also:

wolfSSL_CTX_new

17.10 Memory Abstraction Layer

The functions in this section are used when an application sets its own memory handling functions by using the wolfSSL memory abstraction layer.

wolfSSL_Malloc

Synopsis:

#include <wolfssl/wolfcrypt/memory.h>

void* wolfSSL_Malloc(size_t size)

Description:

This function is similar to malloc(), but calls the memory allocation function which wolfSSL has been configured to use. By default, wolfSSL uses malloc(). This can be changed using the wolfSSL memory abstraction layer - see wolfSSL_SetAllocators().

Return Values:

If successful, this function returns a pointer to allocated memory. If there is an error, NULL will be returned. Specific return values may be dependent on the underlying memory allocation function being used (if not using the default malloc()).

Parameters:

size - number of bytes to allocate.

Example:

```
char* buffer;

buffer = (char*) wolfSSL_Malloc(20);
if (buffer == NULL) {
      // failed to allocate memory
}
```

See Also:

wolfSSL_Free
wolfSSL_Realloc

wolfSSL_SetAllocators

wolfSSL_Realloc

Synopsis:
#include <wolfssl/wolfcrypt/memory.h>

void* wolfSSL_Realloc(void *ptr, size_t size)

Description:
This function is similar to realloc(), but calls the memory re-allocation function which wolfSSL has been configured to use. By default, wolfSSL uses realloc(). This can be changed using the wolfSSL memory abstraction layer - see wolfSSL_SetAllocators().

Return Values:
If successful, this function returns a pointer to re-allocated memory. This may be the same pointer as **ptr**, or a new pointer location. If there is an error, NULL will be returned. Specific return values may be dependent on the underlying memory re-allocation function being used (if not using the default realloc()).

Parameters:

ptr - pointer to the previously-allocated memory, to be reallocated.

size - number of bytes to allocate.

Example:

```
char* buffer;

buffer = (char*) wolfSSL_Realloc(30);
if (buffer == NULL) {
      // failed to re-allocate memory
}
```

See Also:
wolfSSL_Free
wolfSSL_Malloc
wolfSSL_SetAllocators

wolfSSL_Free

Synopsis:

#include <wolfssl/wolfcrypt/memory.h>

void wolfSSL_Free(void* ptr)

Description:

This function is similar to free(), but calls the memory free function which wolfSSL has been configured to use. By default, wolfSSL uses free(). This can be changed using the wolfSSL memory abstraction layer - see wolfSSL_SetAllocators().

Return Values:

This function does not have a return value.

Parameters:

ptr - pointer to the memory to be freed.

Example:

```
char* buffer;
...

wolfSSL_Free(buffer);
```

See Also:

wolfSSL_Alloc
wolfSSL_Realloc
wolfSSL_SetAllocators

wolfSSL_SetAllocators

Synopsis:

#include <wolfssl/wolfcrypt/memory.h>

```
int wolfSSL_SetAllocators(wolfSSL_Malloc_cb  malloc_function,
                          wolfSSL_Free_cb free_function,
                          wolfSSL_Realloc_cb realloc_function);
```

typedef void *(*wolfSSL_Malloc_cb)(size_t size);
typedef void (*wolfSSL_Free_cb)(void *ptr);
typedef void *(*wolfSSL_Realloc_cb)(void *ptr, size_t size);

Description:

This function registers the allocation functions used by wolfSSL. By default, if the system supports it, malloc/free and realloc are used. Using this function allows the user at runtime to install their own memory handlers.

Return Values:

If successful this function will return 0.

BAD_FUNC_ARG is the error that will be returned if a function pointer is not provided.

Parameters:

malloc_function - memory allocation function for wolfSSL to use. Function signature must match wolfSSL_Malloc_cb prototype, above.

free_function - memory free function for wolfSSL to use. Function signature must match wolfSSL_Free_cb prototype, above.

realloc_function - memory re-allocation function for wolfSSL to use. Function signature must match wolfSSL_Realloc_cb prototype, above.

Example:

```
int ret = 0;

// Memory function prototypes
void* MyMalloc(size_t size);
void  MyFree(void* ptr);
void* MyRealloc(void* ptr, size_t size);

// Register custom memory functions with wolfSSL
ret = wolfSSL_SetAllocators(MyMalloc, MyFree, MyRealloc);
if (ret != 0) {
      // failed to set memory functions
}

void* MyMalloc(size_t size)
```

```
{
      // custom malloc function
}

void MyFree(void* ptr)
{
      // custom free function
}

void* MyRealloc(void* ptr, size_t size)
{
      // custom realloc function
}
```

See Also:
NA

17.11 Certificate Manager

The functions in this section are part of the wolfSSL Certificate Manager. The Certificate Manager allows applications to load and verify certificates external to the SSL/TLS connection.

wolfSSL_CertManagerDisableCRL

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CertManagerDisableCRL(WOLFSSL_CERT_MANGER* cm);

Description:
Turns off Certificate Revocation List checking when verifying certificates with the Certificate Manager. By default, CRL checking is off. You can use this function to temporarily or permanently disable CRL checking with this Certificate Manager context that previously had CRL checking enabled.

Return Values:
If successful the call will return **SSL_SUCCESS**.

BAD_FUNC_ARG is the error that will be returned if a function pointer is not provided.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, created using wolfSSL_CertManagerNew().

Example:

```
int ret = 0;
WOLFSSL_CERT_MANAGER* cm;

...

ret = wolfSSL_CertManagerDisableCRL(cm);
if (ret != SSL_SUCCESS) {
      // error disabling cert manager
}

...
```

See Also:
wolfSSL_CertManagerEnableCRL

wolfSSL_CertManagerEnableCRL

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CertManagerEnableCRL(WOLFSSL_CERT_MANGER* cm, int options);

Description:
Turns on Certificate Revocation List checking when verifying certificates with the Certificate Manager. By default, CRL checking is off. options include WOLFSSL_CRL_CHECKALL which performs CRL checking on each certificate in the chain versus the Leaf certificate only which is the default.

Return Values:
If successful the call will return **SSL_SUCCESS**.

NOT_COMPILED_IN will be returned if wolfSSL was not built with CRL enabled.

MEMORY_E will be returned if an out of memory condition occurs.

BAD_FUNC_ARG is the error that will be returned if a pointer is not provided.

SSL_FAILURE will be returned if the CRL context cannot be initialized properly.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, created using wolfSSL_CertManagerNew().

options - options to use when enabling the Certification Manager, **cm**.

Example:

```
int ret = 0;
WOLFSSL_CERT_MANAGER* cm;
...

ret = wolfSSL_CertManagerEnableCRL(cm, 0);
if (ret != SSL_SUCCESS) {
      // error enabling cert manager
}

...
```

See Also:
wolfSSL_CertManagerDisableCRL

wolfSSL_CertManagerFree

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_CertManagerFree(WOLFSSL_CERT_MANGER* cm);

Description:
Frees all resources associated with the Certificate Manager context. Call this when you no longer need to use the Certificate Manager.

Return Values:
No return value is used.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, created using wolfSSL_CertManagerNew().

Example:

```
WOLFSSL_CERT_MANAGER* cm;
...

wolfSSL_CertManagerFree(cm);
```

See Also:
wolfSSL_CertManagerNew

wolfSSL_CertManagerLoadCA

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CertManagerLoadCA(WOLFSSL_CERT_MANGER* cm,
const char* file, const char* path);

Description:
Specifies the locations for CA certificate loading into the manager context. The PEM certificate CAfile may contain several trusted CA certificates. If CApath is not NULL it specifies a directory containing CA certificates in PEM format.

Return Values:
If successful the call will return **SSL_SUCCESS**.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn't exist, can't be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

BAD_FUNC_ARG is the error that will be returned if a pointer is not provided.

SSL_FATAL_ERROR - will be returned upon failure.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, created using wolfSSL_CertManagerNew().

file - pointer to the name of the file containing CA certificates to load.

path - pointer to the name of a directory path containing CA certificates to load. The NULL pointer may be used if no certificate directory is desired.

Example:

```
int ret = 0;
WOLFSSL_CERT_MANAGER* cm;
...

ret = wolfSSL_CertManagerLoadCA(cm, "path/to/cert-file.pem", 0);
if (ret != SSL_SUCCESS) {
      // error loading CA certs into cert manager
}
```

See Also:
wolfSSL_CertManagerVerify

wolfSSL_CertManagerNew

Synopsis:
#include <wolfssl/ssl.h>

WOLFSSL_CERT_MANAGER* wolfSSL_CertManagerNew(void);

Description:
Allocates and initializes a new Certificate Manager context. This context may be used independent of SSL needs. It may be used to load certificates, verify certificates, and check the revocation status.

Return Values:
If successful the call will return a valid WOLFSSL_CERT_MANAGER pointer.

NULL will be returned for an error state.

Parameters:

There are no parameters for this function.

Example:

```
WOLFSSL_CERT_MANAGER* cm;

cm = wolfSSL_CertManagerNew();
if (cm == NULL) {
      // error creating new cert manager
}
```

See Also:
wolfSSL_CertManagerFree

wolfSSL_CertManagerVerify

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CertManagerVerify(WOLFSSL_CERT_MANGER* cm, const char* fname,
int fomat);

Description:
Specifies the certificate to verify with the Certificate Manager context. The format can be SSL_FILETYPE_PEM or SSL_FILETYPE_ASN1.

Return Values:
If successful the call will return **SSL_SUCCESS**.

ASN_SIG_CONFIRM_E will be returned if the signature could not be verified.

ASN_SIG_OID_E will be returned if the signature type is not supported.

CRL_CERT_REVOKED is an error that is returned if this certificate has been revoked.

CRL_MISSING is an error that is returned if a current issuer CRL is not available.

ASN_BEFORE_DATE_E will be returned if the current date is before the before date.

ASN_AFTER_DATE_E will be returned if the current date is after the after date.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn't exist, can't be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

BAD_FUNC_ARG is the error that will be returned if a pointer is not provided.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, created using wolfSSL_CertManagerNew().

fname - pointer to the name of the file containing the certificates to verify.

format - format of the certificate to verify - either SSL_FILETYPE_ASN1 or SSL_FILETYPE_PEM.

Example:

```
int ret = 0;
WOLFSSL_CERT_MANAGER* cm;
...

ret = wolfSSL_CertManagerVerify(cm, "path/to/cert-file.pem",
SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      // error verifying certificate
}
```

See Also:
wolfSSL_CertManagerLoadCA
wolfSSL_CertManagerVerifyBuffer

wolfSSL_CertManagerVerifyBuffer

Synopsis:
#include <wolfssl/ssl.h>

```
int wolfSSL_CertManagerVerifyBuffer(WOLFSSL_CERT_MANGER* cm,
                                    const byte* buff, long sz, int format);
```

Description:
Specifies the certificate buffer to verify with the Certificate Manager context. The format can be SSL_FILETYPE_PEM or SSL_FILETYPE_ASN1.

Return Values:
If successful the call will return **SSL_SUCCESS**.

ASN_SIG_CONFIRM_E will be returned if the signature could not be verified.

ASN_SIG_OID_E will be returned if the signature type is not supported.

CRL_CERT_REVOKED is an error that is returned if this certificate has been revoked.

CRL_MISSING is an error that is returned if a current issuer CRL is not available.

ASN_BEFORE_DATE_E will be returned if the current date is before the before date.

ASN_AFTER_DATE_E will be returned if the current date is after the after date.

SSL_BAD_FILETYPE will be returned if the file is the wrong format.

SSL_BAD_FILE will be returned if the file doesn’t exist, can’t be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

ASN_INPUT_E will be returned if Base16 decoding fails on the file.

BAD_FUNC_ARG is the error that will be returned if a pointer is not provided.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, created using wolfSSL_CertManagerNew().

buff - buffer containing the certificates to verify.

sz - size of the buffer, **buf**.

format - format of the certificate to verify, located in **buf** - either SSL_FILETYPE_ASN1 or SSL_FILETYPE_PEM.

Example:

```
int ret = 0;
int sz = 0;
WOLFSSL_CERT_MANAGER* cm;
byte certBuff[...];
...

ret = wolfSSL_CertManagerVerifyBuffer(cm, certBuff, sz, SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      // error verifying certificate
}
```

See Also:
wolfSSL_CertManagerLoadCA
wolfSSL_CertManagerVerify

wolfSSL_CertManagerCheckOCSP

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CertManagerCheckOCSP(WOLFSSL_CERT_MANAGER* cm,
                                                   byte* der, int sz);

Description:

The function enables the WOLFSSL_CERT_MANAGER's member, ocspEnabled to signify that the OCSP check option is enabled.

Return Values:

SSL_SUCCESS - returned on successful execution of the function. The ocspEnabled member of the WOLFSSL_CERT_MANAGER is enabled.

BAD_FUNC_ARG - returned if the WOLFSSL_CERT_MANAGER structure is NULL or if an argument value that is not allowed is passed to a subroutine.

MEMORY_E - returned if there is an error allocating memory within this function or a subroutine.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, created using wolfSSL_CertManagerNew().

der - a byte pointer to the certificate.

sz - an int type representing the size of the DER cert.

Example:
```
WOLFSSL* ssl = wolfSSL_new(ctx);
byte* der;
int sz; /*size of der */
...
if(wolfSSL_CertManagerCheckOCSP(cm, der, sz) != SSL_SUCCESS){
      /*Failure case. */
}
```

See Also:

ParseCertRelative
CheckCertOCSP

wolfSSL_CertManagerUnloadCAs

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CertManagerUnloadCAs(WOLFSSL_CERT_MANAGER* cm);

Description:
This function unloads the CA signer list.

Return Values:
SSL_SUCCESS - returned on successful execution of the function.

BAD_FUNC_ARG - returned if the WOLFSSL_CERT_MANAGER is NULL.

BAD_MUTEX_E - returned if there was a mutex error.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, created using wolfSSL_CertManagerNew().

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL_CERT_MANAGER* cm = wolfSSL_CertManagerNew();
...
if(wolfSSL_CertManagerUnloadCAs(ctx->cm) != SSL_SUCCESS){
      /*Failure case. */
}
```

See Also:
FreeSignerTable
UnlockMutex

wolfSSL_CertManagerSetOCSPOverrideURL

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CertManagerSetOCSPOverrideURL(WOLFSSL_CERT_MANAGER* cm,
                                          const char* url);

Description:
The function copies the url to the ocspOverrideURL member of the

WOLFSSL_CERT_MANAGER structure.

Return Values:

SSL_SUCCESS - the function was able to execute as expected.

BAD_FUNC_ARG - the WOLFSSL_CERT_MANAGER struct is NULL.

MEMEORY_E - Memory was not able to be allocated for the ocspOverrideURL member of the certificate manager.

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new().

Example:

```
WOLFSSL_CERT_MANAGER* cm = wolfSSL_CertManagerNew();
const char* url;
…
int wolfSSL_SetOCSP_OverrideURL(WOLFSSL* ssl, const char* url)
…
if(wolfSSL_CertManagerSetOCSPOverrideURL(ssl->ctx->cm, url) != SSL_SUCCESS){
      /*Failure case. */
}
```

See Also:

ocspOverrideURL
wolfSSL_SetOCSP_OverrideURL

wolfSSL_CertManagerLoadCRL

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CertManagerLoadCRL(WOLFSSL_CERT_MANAGER* cm,
                  const char* path, int type, int monitor);

Description:

Error checks and passes through to LoadCRL() in order to load the cert into the CRL for revocation checking.

Return Values:

SSL_SUCCESS - if there is no error in wolfSSL_CertManagerLoadCRL and if LoadCRL returns successfully.

BAD_FUNC_ARG - if the WOLFSSL_CERT_MANAGER struct is NULL.

SSL_FATAL_ERROR - if wolfSSL_CertManagerEnableCRL returns anything other than SSL_SUCCESS.

BAD_PATH_ERROR - if the path is NULL.

MEMORY_E - if LoadCRL fails to allocate heap memory.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, created using wolfSSL_CertManagerNew().

path - a constant char pointer holding the CRL path.

type - type of certificate to be loaded.

monitor - requests monitoring in LoadCRL().

Example:

```
int wolfSSL_LoadCRL(WOLFSSL* ssl, const char* path, int type,
                        int monitor);
…
wolfSSL_CertManagerLoadCRL(ssl->ctx->cm, path, type, monitor);
```

See Also:

wolfSSL_CertManagerEnableCRL
wolfSSL_LoadCRL

wolfSSL_CertManagerLoadCABuffer

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CertManagerLoadCABuffer(WOLFSSL_CERT_MANAGER* cm,
                                  const unsigned char* in, long sz, int format);

Description:
Loads the CA Buffer by calling wolfSSL_CTX_load_verify_buffer and returning that result using a temporary cm so as not to lose the information in the cm passed into the function.

Return Values:

SSL_FATAL_ERROR is returned if the WOLFSSL_CERT_MANAGER struct is NULL or if wolfSSL_CTX_new() returns NULL.

SSL_SUCCESS is returned for a successful execution.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, created using wolfSSL_CertManagerNew().

in - buffer for cert information.

sz - length of the buffer.

format - certificate format, either PEM or DER.

Example:

```
WOLFSSL_CERT_MANAGER* cm = (WOLFSSL_CERT_MANAGER*)vp;
…
const unsigned char* in;
long sz;
int format;
…
if(wolfSSL_CertManagerLoadCABuffer(vp, sz, format) != SSL_SUCCESS){
      /*Error returned. Failure case code block. */
}
```

See Also:
wolfSSL_CTX_load_verify_buffer
ProcessChainBuffer
ProcessBuffer
cm_pick_method

wolfSSL_CertManagerUnload_trust_peers

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CertManagerUnload_trust_peers(WOLFSSL_CERT_MANAGER* cm);

Description:
The function will free the Trusted Peer linked list and unlocks the trusted peer list.

Return Values:
SSL_SUCCESS if the function completed normally.

BAD_FUNC_ARG if the WOLFSSL_CERT_MANAGER is NULL.

BAD_MUTEX_E mutex error if tpLock, a member of the WOLFSSL_CERT_MANAGER struct, is 0 (nill).

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, created using wolfSSL_CertManagerNew().

Example:

```
WOLFSSL_CTX* ctx = WOLFSSL_CTX_new(/*Protocol define*/);
WOLFSSL_CERT_MANAGER* cm = wolfSSL_CertManagerNew();
...
if(wolfSSL_CertManagerUnload_trust_peers(cm) != SSL_SUCCESS){
      /*The function did not execute successfully. */
}
```

See Also:

UnLockMutex

wolfSSL_CertManagerEnableOCSP

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CertManagerEnableOCSP(WOLFSSL_CERT_MANAGER* cm,
int options) ;

Description:
Turns on OCSP if it’s turned off and if compiled with the set option available.

Return Values:
SSL_SUCCESS returned if the function call is successful.

BAD_FUNC_ARG if cm struct is NULL.

MEMORY_E if WOLFSSL_OCSP struct value is NULL.

SSL_FAILURE initialization of WOLFSSL_OCSP struct fails to initialize.

NOT_COMPILED_IN build not compiled with correct feature enabled.

Parameters:

cm - a pointer to a WOLFSSL_CERT_MANAGER structure, created using wolfSSL_CertManagerNew().

options - used to set values in WOLFSSL_CERT_MANAGER struct.

Example:

```
WOLFSSL_CTX* ctx = wolfSSL_CTX_new(/*protocol method*/);
WOLFSSL* ssl = wolfSSL_new(ctx);
WOLFSSL_CERT_MANAGER* cm = wolfSSL_CertManagerNew();
int options;
```

```
…
if(wolfSSL_CertManagerEnableOCSP(ssl->ctx->cm, options) != SSL_SUCCESS){
      /*Failure case. */
}
```

See Also:

wolfSSL_CertManagerNew

wolfSSL_BN_div

Synopsis:

#include <wolfssl/ssl.h>
#include <wolfssl/openssl/bn.h>

int wolfSSL_BN_div(WOLFSSL_BIGNUM* r, WOLFSSL_BIGNUM* m,

const WOLFSSL_BIGNUM* a, const WOLFSSL_BIGNUM* b,

const WOLFSSL_BN_CTX* c)

Description:

This function divides a by b and returns the quotient in r and the remainder in m (r=a/b, m=a%b). Either r or m may be NULL, in such case the value is not returned respectively. The quotient is rounded towards zero.

Return Values:

SSL_SUCCESS returned if the function call is successful.

Parameters:

r - the quotient (a/b)

m - the remainder (a%b)

a- the dividend

b - the divisor

c - pointer to a WOLFSSL_BN_CTX structure

Example:

```
BIGNUM *r, *m, *a,*b;
BN_CTX *c;
unsigned long wa,wb;

a = BN_new();
b = BN_new();
r = BN_new();
m = BN_new();

wa = 100;
wb = 30;
BN_set_word(a, wa);
BN_set_word(b, wb);
c = NULL;

if(BN_div(r, m, a, b, c)!= SSL_SUCCESS){
      /*Failure case. */
};

BN_free(a);
BN_free(b);
```

wolfSSL_BN_mod_inverse

Synopsis:

#include <wolfssl/openssl/bn.h>

WOLFSSL_BIGNUM *wolfSSL_BN_mod_inverse(WOLFSSL_BIGNUM* r, WOLFSSL_BIGNUM* a, const WOLFSSL_BIGNUM* n, WOLFSSL_BN_CTX *ctx);

Description:

This function compute the inverse of a modulo n places the results in r ((a*r)%n == 1).If r is NULL, a new BIGNUM is created.

Return Values:

Returns a pointer to computed bignum value and **NULL** on failure.

Parameters:

r - placeholder for computed mod inverse bignum value
a - bignum argument to compute mod inverse in (a*r)%n == 1
n - bignum argument to compute mod inverse in (a*r)%n == 1
ctx - bignum context

Example:

```
unsigned char value[1];
WOLFSSL_BIGNUM* r,a,n,val;

value[0] = 0x02;
wolfSSL_BN_bin2bn(value, sizeof(value), a);
value[0] = 0x05;
wolfSSL_BN_bin2bn(value, sizeof(value), n);

r = wolfSSL_BN_new();
val = wolfSSL_mod_inverse(r,a,n);
printf("mod inverse = %x\n",wolfSSL_BN_bn2bin(r,value));

wolfSSL_BN_free(a);
wolfSSL_BN_free(n);
wolfSSL_BN_free(r);
```

17.12 OpenSSL Compatibility Layer

The functions in this section are part of wolfSSL's OpenSSL Compatibility Layer. These functions are only available when wolfSSL has been compiled with the OPENSSL_EXTRA define.

wolfSSL_X509_get_serial_number

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_X509_get_serial_number(WOLFSSL_X509* x509, byte* in,
int* inOutSz);

Description:
Retrieves the peer's certificate serial number. The serial number buffer (**in**) should be at least 32 bytes long and be provided as the ***inOutSz** argument as input. After calling

the function ***inOutSz** will hold the actual length in bytes written to the **in** buffer.

Return Values:
If successful the call will return **SSL_SUCCESS**.

BAD_FUNC_ARG will be returned if a bad function argument was encountered.

See Also:
SSL_get_peer_certificate

wolfSSL_get_sessionID

Synopsis:
#include <wolfssl/ssl.h>

const unsigned char* wolfSSL_get_sessionID(const WOLFSSL_SESSION* session);

Description:
Retrieves the session's ID. The session ID is always 32 bytes long.

Return Values:
The session ID.

See Also:
SSL_get_session()

wolfSSL_get_peer_chain

Synopsis:
#include <wolfssl/ssl.h>

X509_CHAIN* wolfSSL_get_peer_chain(WOLFSSL* ssl);

Description:
Retrieves the peer's certificate chain.

Return Values:
If successful the call will return the peer's certificate chain.

0 will be returned if an invalid WOLFSSL pointer is passed to the function.

See Also:
wolfSSL_get_chain_count
wolfSSL_get_chain_length
wolfSSL_get_chain_cert
wolfSSL_get_chain_cert_pem

wolfSSL_get_chain_count

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_get_chain_count(WOLFSSL_X509_CHAIN* chain);

Description:
Retrieves the peer's certificate chain count.

Return Values:
If successful the call will return the peer's certificate chain count.

0 will be returned if an invalid chain pointer is passed to the function.

See Also:
wolfSSL_get_peer_chain
wolfSSL_get_chain_length
wolfSSL_get_chain_cert
wolfSSL_get_chain_cert_pem

wolfSSL_get_chain_length

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_get_chain_length(WOLFSSL_X509_CHAIN* chain, int idx);

Description:
Retrieves the peer's ASN1.DER certificate length in bytes at index (**idx**).

Return Values:
If successful the call will return the peer's certificate length in bytes by index.

0 will be returned if an invalid chain pointer is passed to the function.

See Also:
wolfSSL_get_peer_chain
wolfSSL_get_chain_count
wolfSSL_get_chain_cert
wolfSSL_get_chain_cert_pem

wolfSSL_get_chain_cert

Synopsis:
#include <wolfssl/ssl.h>

byte* wolfSSL_get_chain_cert(WOLFSSL_X509_CHAIN* chain, int idx);

Description:
Retrieves the peer's ASN1.DER certificate at index (**idx**).

Return Values:
If successful the call will return the peer's certificate by index.

0 will be returned if an invalid chain pointer is passed to the function.

See Also:
wolfSSL_get_peer_chain
wolfSSL_get_chain_count
wolfSSL_get_chain_length
wolfSSL_get_chain_cert_pem

wolfSSL_get_chain_cert_pem

Synopsis:
#include <wolfssl/ssl.h>

unsigned char* wolfSSL_get_chain_cert_pem(WOLFSSL_X509_CHAIN* chain, int idx);

Description:
Retrieves the peer's PEM certificate at index (**idx**).

Return Values:
If successful the call will return the peer's certificate by index.

0 will be returned if an invalid chain pointer is passed to the function.

See Also:
wolfSSL_get_peer_chain
wolfSSL_get_chain_count
wolfSSL_get_chain_length
wolfSSL_get_chain_cert

wolfSSL_PemCertToDer

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_PemCertToDer(const char* fileName, unsigned char* derBuf, int derSz);

Description:
Loads the PEM certificate from **fileName** and converts it into DER format, placing the result into **derBuffer** which is of size **derSz**.

Return Values:
If successful the call will return the number of bytes written to **derBuffer**.

SSL_BAD_FILE will be returned if the file doesn't exist, can't be read, or is corrupted.

MEMORY_E will be returned if an out of memory condition occurs.

SSL_NO_PEM_HEADER will be returned if the PEM certificate header can't be found.

BUFFER_E will be returned if a chain buffer is bigger than the receiving buffer.

Parameters:

filename - pointer to the name of the PEM-formatted certificate for conversion.

derBuffer - the buffer for which the converted PEM certificate will be placed in DER format.

derSz - size of derBuffer.

Example:

```
int derSz;
byte derBuf[...];

derSz = wolfSSL_PemCertToDer("./cert.pem", derBuf, sizeof(derBuf));
```

See Also:
SSL_get_peer_certificate

wolfSSL_CTX_use_RSAPrivateKey_file

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_use_RSAPrivateKey_file(WOLFSSL_CTX* ctx,const char* file,
int format);

Description:
This function loads the private RSA key used in the SSL connection into the SSL context (WOLFSSL_CTX). This function is only available when wolfSSL has been compiled with the OpenSSL compatibility layer enabled (--enable-opensslExtra, #define OPENSSL_EXTRA), and is identical to the more-typically used wolfSSL_CTX_use_PrivateKey_file() function.

The **file** argument contains a pointer to the RSA private key file, in the format specified by **format**.

Return Values:
If successful the call will return **SSL_SUCCESS**, otherwise **SSL_FAILURE** will be returned. If the function call fails, possible causes might include:

- The input key file is in the wrong format, or the wrong format has been given using the "format" argument
- file doesn't exist, can't be read, or is corrupted
- an out of memory condition occurs

Parameters:

ctx - a pointer to a WOLFSSL_CTX structure, created using wolfSSL_CTX_new()

file - a pointer to the name of the file containing the RSA private key to be loaded into the wolfSSL SSL context, with format as specified by **format**.

format - the encoding type of the RSA private key specified by **file**. Possible values include SSL_FILETYPE_PEM and SSL_FILETYPE_ASN1.

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx;

...

ret = wolfSSL_CTX_use_RSAPrivateKey_file(ctx, "./server-key.pem",
                                       SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      // error loading private key file
}

...
```

See Also:
wolfSSL_CTX_use_PrivateKey_buffer
wolfSSL_CTX_use_PrivateKey_file
wolfSSL_use_RSAPrivateKey_file
wolfSSL_use_PrivateKey_buffer
wolfSSL_use_PrivateKey_file

wolfSSL_use_certificate_file

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_use_certificate_file(WOLFSSL* ssl, const char* file, int format);

Description:

This function loads a certificate file into the SSL session (WOLFSSL structure). The certificate file is provided by the **file** argument. The **format** argument specifies the format type of the file - either **SSL_FILETYPE_ASN1** or **SSL_FILETYPE_PEM**.

Return Values:

If successful the call will return **SSL_SUCCESS**, otherwise **SSL_FAILURE** will be returned. If the function call fails, possible causes might include:

- The file is in the wrong format, or the wrong format has been given using the “format” argument
- file doesn’t exist, can’t be read, or is corrupted
- an out of memory condition occurs
- Base16 decoding fails on the file

Parameters:

ssl - a pointer to a WOLFSSL structure, created with wolfSSL_new().

file - a pointer to the name of the file containing the certificate to be loaded into the wolfSSL SSL session, with format as specified by **format**.

format - the encoding type of the certificate specified by **file**. Possible values include SSL_FILETYPE_PEM and SSL_FILETYPE_ASN1.

Example:

```
int ret = 0;
WOLFSSL* ssl;

...

ret = wolfSSL_use_certificate_file(ssl, “./client-cert.pem”,
                                 SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      // error loading cert file
}

...
```

See Also:
wolfSSL_CTX_use_certificate_buffer
wolfSSL_CTX_use_certificate_file
wolfSSL_use_certificate_buffer

wolfSSL_use_PrivateKey_file

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_use_PrivateKey_file(WOLFSSL* ssl, const char* file, int format);

Description:
This function loads a private key file into the SSL session (WOLFSSL structure). The key file is provided by the **file** argument. The **format** argument specifies the format type of the file - **SSL_FILETYPE_ASN1** or **SSL_FILETYPE_PEM**.

Return Values:
If successful the call will return **SSL_SUCCESS**, otherwise **SSL_FAILURE** will be returned. If the function call fails, possible causes might include:

- The file is in the wrong format, or the wrong format has been given using the "format" argument
- The file doesn't exist, can't be read, or is corrupted
- An out of memory condition occurs
- Base16 decoding fails on the file
- The key file is encrypted but no password is provided

Parameters:

ssl - a pointer to a WOLFSSL structure, created with wolfSSL_new().

file - a pointer to the name of the file containing the key file to be loaded into the wolfSSL SSL session, with format as specified by **format**.

format - the encoding type of the key specified by **file**. Possible values include SSL_FILETYPE_PEM and SSL_FILETYPE_ASN1.

Example:

```
int ret = 0;
WOLFSSL* ssl;

...

ret = wolfSSL_use_PrivateKey_file(ssl, "./server-key.pem",
                                SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      // error loading key file
}

...
```

See Also:
wolfSSL_CTX_use_PrivateKey_buffer
wolfSSL_CTX_use_PrivateKey_file
wolfSSL_use_PrivateKey_buffer

wolfSSL_use_certificate_chain_file

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_use_certificate_chain_file(WOLFSSL* ssl, const char* file);

Description:
This function loads a chain of certificates into the SSL session (WOLFSSL structure). The file containing the certificate chain is provided by the **file** argument, and must contain PEM-formatted certificates. This function will process up to MAX_CHAIN_DEPTH (default = 9, defined in internal.h) certificates, plus the subject certificate.

Return Values:
If successful the call will return **SSL_SUCCESS**, otherwise **SSL_FAILURE** will be returned. If the function call fails, possible causes might include:

- The file is in the wrong format, or the wrong format has been given using the "format" argument
- file doesn't exist, can't be read, or is corrupted
- an out of memory condition occurs

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new()

file - a pointer to the name of the file containing the chain of certificates to be loaded into the wolfSSL SSL session. Certificates must be in PEM format.

Example:

```
int ret = 0;
WOLFSSL* ctx;

...

ret = wolfSSL_use_certificate_chain_file(ssl, “./cert-chain.pem”);
if (ret != SSL_SUCCESS) {
      // error loading cert file
}

...
```

See Also:
wolfSSL_CTX_use_certificate_chain_file
wolfSSL_CTX_use_certificate_chain_buffer
wolfSSL_use_certificate_chain_buffer

wolfSSL_use_RSAPrivateKey_file

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_use_RSAPrivateKey_file(WOLFSSL* ssl,const char* file, int format);

Description:
This function loads the private RSA key used in the SSL connection into the SSL session (WOLFSSL structure). This function is only available when wolfSSL has been compiled with the OpenSSL compatibility layer enabled (--enable-opensslExtra, #define OPENSSL_EXTRA), and is identical to the more-typically used wolfSSL_use_PrivateKey_file() function.

The **file** argument contains a pointer to the RSA private key file, in the format specified by **format**.

Return Values:

If successful the call will return **SSL_SUCCESS**, otherwise **SSL_FAILURE** will be returned. If the function call fails, possible causes might include:

- The input key file is in the wrong format, or the wrong format has been given using the "format" argument
- file doesn't exist, can't be read, or is corrupted
- an out of memory condition occurs

Parameters:

ssl - a pointer to a WOLFSSL structure, created using wolfSSL_new()

file - a pointer to the name of the file containing the RSA private key to be loaded into the wolfSSL SSL session, with format as specified by **format**.

format - the encoding type of the RSA private key specified by **file**. Possible values include SSL_FILETYPE_PEM and SSL_FILETYPE_ASN1.

Example:

```
int ret = 0;
WOLFSSL* ssl;

...

ret = wolfSSL_use_RSAPrivateKey_file(ssl, "./server-key.pem",
                                   SSL_FILETYPE_PEM);
if (ret != SSL_SUCCESS) {
      // error loading private key file
}

...
```

See Also:

wolfSSL_CTX_use_RSAPrivateKey_file
wolfSSL_CTX_use_PrivateKey_buffer
wolfSSL_CTX_use_PrivateKey_file
wolfSSL_use_PrivateKey_buffer
wolfSSL_use_PrivateKey_file

wolfSSL_PKCS5_PBKDF2_HMAC_SHA1

Synopsis:

#include <wolfssl/openssl/evp.h>

int wolfSSL_PKCS5_PBKDF2_HMAC_SHA1(const char *pass, int passlen, const unsigned char *salt, int saltlen, int iter, int keylen, unsigned char *out);

Description:

This function derives a key from a password using a salt and iteration count as specified in RFC2898.

Return Values:

Return **1** on success or **0** on error.

Parameters:

pass - password
passlen - password length
salt - salt
saltlen - salt length
iter - iteration count
keylen - key length

Example:

```
const char *pass = "pass";
const unsigned char *salt = (unsigned char *)"salt";
unsigned char *out = malloc(256);
int iter = 100;
int ret = 0;
int pass_len = 0;
int salt_len = 0;

pass_len = strlen(pass);
salt_len = strlen(salt);

ret =
WolfSSL_PBKDF2_HMAC_SHA1(passwd,pass_len,salt,salt_len,iter,SHA_DIGEST_SIZE,o
ut);

free(out);
```

See Also:

wolfSSL_PKCS12_parse

Synopsis:

#include <wolfssl/ssl.h>

PKCS12_parse ->
int wolfSSL_PKCS12_parse(WC_PKCS12* pkcs12, const char* paswd,
WOLFSSL_EVP_PKEY** pkey, WOLFSSL_X509** cert,
STACK_OF(WOLFSSL_X509)** stack);

Description:

PKCS12 can be enabled with adding –enable-opensslextra to the configure command. It can use triple DES and RC4 for decryption so would recommend also enabling these features when enabling opensslextra (--enable-des3 –enable-arc4). wolfSSL does not currently support RC2 so decryption with RC2 is currently not available. This may be noticeable with default encryption schemes used by OpenSSL command line to create .p12 files.

wolfSSL_PKCS12_parse (PKCS12_parse). The first thing this function does is check the MAC is correct if present. If the MAC fails then the function returns and does not try to decrypt any of the stored Content Infos.

This function then parses through each Content Info looking for a bag type, if the bag type is known it is decrypted as needed and either stored in the list of certificates being built or as a key found. After parsing through all bags the key found is then compared with the certificate list until a matching pair is found. This matching pair is then returned as the key and certificate, optionally the certificate list found is returned as a STACK_OF certificates.

At the moment a CRL, Secret or SafeContents bag will be skipped over and not parsed. It can be seen if these or other “Unknown” bags are skipped over by viewing the debug print out. Additional attributes such as friendly name are skipped over when parsing a PKCS12 file.

Return Values:

SSL_SUCCESS: On successfully parsing PKCS12.

SSL_FAILURE: If an error case was encountered.

Parameters:

pkcs12 - WC_PKCS12 structure to parse.

paswd - password for decrypting PKCS12.

pkey - structure to hold private key decoded from PKCS12.

cert - structure to hold certificate decoded from PKCS12.

stack - optional stack of extra certificates.

Example:

```
WC_PKCS12* pkcs;
WOLFSSL_BIO* bio;
WOLFSSL_X509* cert;
WOLFSSL_EVP_PKEY* pkey;
STACK_OF(X509) certs;

//bio loads in PKCS12 file

wolfSSL_d2i_PKCS12_bio(bio, &pkcs);
wolfSSL_PKCS12_parse(pkcs, “a password”, &pkey, &cert, &certs)
wc_PKCS12_free(pkcs)
```

```
//use cert, pkey, and optionally certs stack
```

See Also:

wolfSSL_d2i_PKCS12_bio, wc_PKCS12_free

wolfSSL_d2i_PKCS12_bio

Synopsis:

#include <wolfssl/ssl.h>

d2i_PKCS12_bio ->
WC_PKCS12* wolfSSL_d2i_PKCS12_bio(WOLFSSL_BIO* bio, WC_PKCS12** pkcs12);

Description:

wolfSSL_d2i_PKCS12_bio (d2i_PKCS12_bio) copies in the PKCS12 information from WOLFSSL_BIO to the structure WC_PKCS12. The information is divided up in the structure as a list of Content Infos along with a structure to hold optional MAC information. After the information has been divided into chunks (but not decrypted) in the structure WC_PKCS12, it can then be parsed and decrypted by calling

Return Values:

WC_PKCS12*: pointer to a WC_PKCS12 structure. If function failed it will return NULL.

Parameters:

bio - WOLFSSL_BIO structure to read PKCS12 buffer from.
pkcs12 - WC_PKCS12 structure pointer for new PKCS12 structure created. Can be NULL

Example:

```
WC_PKCS12* pkcs;

WOLFSSL_BIO* bio;

WOLFSSL_X509* cert;

WOLFSSL_EVP_PKEY* pkey;

STACK_OF(X509) certs;
```

```
//bio loads in PKCS12 file

wolfSSL_d2i_PKCS12_bio(bio, &pkcs);
wolfSSL_PKCS12_parse(pkcs, “a password”, &pkey, &cert, &certs)
wc_PKCS12_free(pkcs)

//use cert, pkey, and optionally certs stack
```

See Also:

wolfSSL_PKCS12_parse, wc_PKCS12_free

wolfSSL_set_tlsext_status_type

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_set_tlsext_status_type(WOLFSSL *s int type);

Description:

This function is called when the client application request that a server send back an OCSP status response (also known as OCSP stapling).Currently, the only supported type is TLSEXT_STATUSTYPE_ocsp.

Return Values:

Return **1** on success or **0** on error.

Parameters:

s - pointer to WolfSSL struct which is created by SSL_new() function
type - ssl extension type which TLSEXT_STATUSTYPE_ocsp is only supported .

Example:

```
WOLFSSL *ssl;
```

```
WOLFSSL_CTX *ctx;
int ret;

ctx = wolfSSL_CTX_new(wolfSSLv23_server_method());
ssl = wolfSSL_new(ctx);
ret = WolfSSL_set_tlsext_status_type(ssl,TLSEXT_STATUSTYPE_ocsp);

wolfSSL_free(ssl);
wolfSSL_CTX_free(ctx);
```

See Also:

wolfSSL_new
wolfSSL_CTX_new
wolfSSL_free
wolfSSL_CTX_free

wolfSSL_ASN1_TIME_adj

Synopsis:

# include <wolfssl/ssl.h>

WOLFSSL_ASN1_TIME* wolfSSL_ASN1_TIME_adj(WOLFSSL_ASN1_TIME *s, time_t t, int offset_day, long offset_sec)

Description:

This function sets the ASN1_TIME structure s to the time represented by the time offset_day and offset_sec after the time_t value t.
if s is NULL, a new ASN1_TIME structure s is allocated and returned.

Return Values:

Returns a pointer to WOLFSSL_ASN1_TIME structure.

Parameters:

s - pointer to WOLFSSL_ASN1_TIME structure.
t - time_t time information to adjust.
offset_day - a number of days to adjust time_t t.
offset_sec - a number of secs to dajust timet_t t.

Example:

```
#include <wolfssl/ssl.h>

WOLFSSL_ASN_TIME *s,*adj_ret;
time_t t = 30 * years + 45 * days;
int offset_day = 10;
long offset_sec = 1200;

s = (WOLFSSL_ASN1_TIME *)malloc(sizeof(WOLFSSL_ASN1_TIME));
adj_ret = wolfSSL_ASN1_TIME_adj(s, t, offset_day, offset_sec);
```

See Also:

wolfSSL_X509_STORE_CTX_set_time

Synopsis:

#include <wolfssl/ssl.h>
void wolfSSL_X509_STORE_CTX_set_time(WOLFSSL_X509_STORE_CTX
*ctx,unsigned long flags, time_t t)

Description:

This function sets certificate validation date.

Return Values:

No value returned.

Parameters:

ctx - pointer to WOLFSSL_X509_STORE_CTX.if NULL is passed, function allocate memory and return it.
flags - not used
time_t - time to validate certificate.

Example:

```
WOLFSSL_X509_STORE_CTX*  ctx;
time_t ctime;

ctx = XMALLOC(sizeof(WOLFSSL_X509_STORE_CTX),
NULL,DYNAMIC_TYPE_TMP_BUFFER);
ctx->param = XMALLOC(sizeof(WOLFSSL_X509_VERIFY_PARAM), NULL,
DYNAMIC_TYPE_TMP_BUFFER);
ctime = time_to validate;
wolfSSL_X509_STORE_CTX_set_time(ctx, 0, ctime);
```

See Also:

wolfSSL_X509_STORE_CTX_set_verify_cb

Synopsis:

```
#include <wolfssl/ssl.h>
void wolfSSL_X509_STORE_CTX_set_verify_cb(WOLFSSL_X509_STORE_CTX *ctx,
WOLFSSL_X509_STORE_CTX_verify_cb verify_cb)
```

Description:

This function sets the verification callback of ctx.

Callback prototype:
typedef void *WOLFSSL_X509_STORE_CTX_verify_cb;

Return Values:

No value returned.

Parameters:

ctx - pointer to WOLFSSL_X509_STORE_CTX
cb - verification callback function.

Example:

```
static int cb(int v, WOLFSSL_X509_STORE_CTX*ctx)
{ …
    return 1;
}


WOLFSSL_X509_STORE_CTX *ctx;

wolfSSL_X509_STORE_CTX_set_verify_cb(ctx, cb) ;
```

See Also:

wolfSSL_CTX_add_client_CA

Synopsis:

```
#include <wolfssl/ssh.h>
int wolfSSL_CTX_add_client_CA(WOLFSSL_CTX *ctx, WOLFSSL_X509 *x509)
```

Description:

This function adds client certificates to WOLFSSL_CTX context structure.

Return Values:

On success a SSL_SUCCESS is returned, on failure SSL_FAILURE is returned.

Parameters:

ctx - pointer to WOLFSSL_CTX structure to set client certificate in.
x509 - pointer to WOLFSSL_X509 structure which is client certificate information.

Example:

```
WOLFSSL_CTX* ctx;
WOLFSSL_X509* x509;
```

```
int ret;

ctx = wolfSSL_CTX_new(wolfSSLv23_client_method());
x509 = wolfSSL_X509_load_certificate_file(certfile, SSL_FILETYPE_PEM);
ret = wolfSSL_CTX_add_client_CA(ctx, x509);
```

See Also:

wolfSSL_X509_load_certificate_file
wolfSSL_SSL_CTX_get_cliet_CA_list

wolfSSL_CTX_set_srp_username

Synopsys:

```
#include <wolfssl/ssl.h>
int wolfSSL_CTX_set_srp_username(WOLFSSL_CTX* ctx, char* password)
```

Description:

This function set user name for SRP in WOLFSSL_CTX structure.

Return Values:

On success a SSL_SUCCESS is returned, on failure SSL_FAILURE is returned.

Parameters:

ctx - pointer to WOLFSSL_CTX_structure.
username - user name for SRP.

Examples:

```
WOLFSSL_CTX *ctx;
const char *username = "TESTUSER";
int r;

ctx = wolfSSL_CTX_new(wolfSSLv23_client_method());
r = wolfSSL_CTX_set_srp_username(ctx, (char *)username);
```

See Also:

wolfSSL_CTX_new
wolfSSL_CTX_set_srp_password

wolfSSL_CTX_set_srp_password

Synopsis:

```
#include <wolfssl/ssl.h>
int wolfSSL_CTX_set_srp_password(WOLFSSL_CTX* ctx, char* password)
```

Description:

This function sets password for SRP in WOLFSSL_CTX structure.

Return Values:

On success a SSL_SUCCESS is returned, on failure SSL_FAILURE is returned.

Parameters:

ctx - pointer to WOLFSSL_CTX structure.
password - password for SRP.

Example:

```
WOLFSSL_CTX *ctx;
const char *password = "TESTPASS";
int r;

ctx = wolfSSL_CTX_new(wolfSSLv23_client_method());
r = wolfSSL_CTX_set_srp_password(ctx, (char *)password);
```

See Also:

wolfSSL_CTX_new
wolfSSL_CTX_set_srp_username

wolfSS_SSL_CTX_set_alpn_protos

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_set_alpn_protos(WOLFSSL_CTX *ctx, const unsigned char *p,
unsigned int p_len)

Description:

This function is used by the client to set the list of protocols available to be negotiated.
.

Parameters:

ctx - pointer to WOLFSSL_CTX structure.
P - protocol list in protocol-list format
P_len - list length

Example:

```
WOLFSSL_CTX *ctx;
unsigned char protos[] = {
     7, 't', 'l', 's', '/', '1', '.', '3',
     8, 'h', 't', 't', 'p', '/', '2', '.', '0'
};
unsigned int len = sizeof(protos);

SSL_CTX_set_alpn_protos(ctx, protos, len);
```

See Also:

wolfSSL_CTX_new

17.13 TLS Extensions

The functions in this section are specific to supported TLS extensions.

wolfSSL_CTX_UseSNI

Synopsis:

```
#include <wolfssl/ssl.h>

int wolfSSL_CTX_UseSNI(WOLFSSL_CTX* ctx, byte type,
                                    const void* data, word16 size);
```

Description:

This function enables the use of Server Name Indication for SSL objects created from the SSL context passed in the 'ctx' parameter. It means that the SNI extension will be sent on ClientHello by wolfSSL clients and wolfSSL servers will respond ClientHello + SNI with either ServerHello + blank SNI or alert fatal in case of SNI mismatch.

Return Values:

If successful the call will return **SSL_SUCCESS**.

BAD_FUNC_ARG is the error that will be returned in one of these cases:
  * ctx is NULL
  * data is NULL
  * type is a unknown value. (see below)

MEMORY_E is the error returned when there is not enough memory.

Parameters:

ctx - pointer to a SSL context, created with wolfSSL_CTX_new().

type - indicates which type of server name is been passed in data. The known types are:

```
enum {
    WOLFSSL_SNI_HOST_NAME = 0
};
```

data - pointer to the server name data.

size - size of the server name data.

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx = 0;

ctx = wolfSSL_CTX_new(method);

if (ctx == NULL) {
    // context creation failed
}

ret = wolfSSL_CTX_UseSNI(ctx, WOLFSSL_SNI_HOST_NAME, "www.yassl.com",
strlen("www.yassl.com"));

if (ret != 0) {
    // sni usage failed
}
```

See Also:

wolfSSL_CTX_new
wolfSSL_UseSNI

wolfSSL_UseSNI

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_UseSNI(WOLFSSL* ssl, unsigned char type,
                                    const void* data, word16 size);

Description:

This function enables the use of Server Name Indication in the SSL object passed in the 'ssl' parameter. It means that the SNI extension will be sent on ClientHello by wolfSSL client and wolfSSL server will respond ClientHello + SNI with either ServerHello + blank SNI or alert fatal in case of SNI mismatch.

Return Values:

If successful the call will return **SSL_SUCCESS**.

BAD_FUNC_ARG is the error that will be returned in one of these cases:

 * ssl is NULL
 * data is NULL
 * type is a unknown value. (see below)

MEMORY_E is the error returned when there is not enough memory.

Parameters:

ssl - pointer to a SSL object, created with wolfSSL_new().

type - indicates which type of server name is been passed in data. The known types are:

```
enum {
    WOLFSSL_SNI_HOST_NAME = 0
};
```

data - pointer to the server name data.

size - size of the server name data.

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx = 0;
WOLFSSL* ssl = 0;

ctx = wolfSSL_CTX_new(method);

if (ctx == NULL) {
    // context creation failed
}

ssl = wolfSSL_new(ctx);

if (ssl == NULL) {
    // ssl creation failed
}
```

```
ret = wolfSSL_UseSNI(ssl, WOLFSSL_SNI_HOST_NAME, "www.yassl.com",
strlen("www.yassl.com"));

if (ret != 0) {
    // sni usage failed
}
```

See Also:

wolfSSL_new
wolfSSL_CTX_UseSNI

wolfSSL_CTX_SNI_SetOptions

Synopsis:

#include <wolfssl/ssl.h>

void wolfSSL_CTX_SNI_SetOptions(WOLFSSL_CTX* ctx, byte type,
unsigned char options);

Description:

This function is called on the server side to configure the behavior of the SSL sessions using Server Name Indication for SSL objects created from the SSL context passed in the 'ctx' parameter. The options are explained below.

Return Values:

This function does not have a return value.

Parameters:

ctx - pointer to a SSL context, created with wolfSSL_CTX_new().

type - indicates which type of server name is been passed in data. The known types are:

```
enum {
    WOLFSSL_SNI_HOST_NAME = 0
};
```

options - a bitwise semaphore with the chosen options. The available options are:

```
enum {
        WOLFSSL_SNI_CONTINUE_ON_MISMATCH = 0x01,
        WOLFSSL_SNI_ANSWER_ON_MISMATCH   = 0x02
    };
```

Normally the server will abort the handshake by sending a fatal-level unrecognized_name(112) alert if the hostname provided by the client mismatch with the servers.

WOLFSSL_SNI_CONTINUE_ON_MISMATCH - With this option set, the server will not send a SNI response instead of aborting the session.

WOLFSSL_SNI_ANSWER_ON_MISMATCH - With this option set, the server will send a SNI response as if the host names match instead of aborting the session.

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx = 0;

ctx = wolfSSL_CTX_new(method);

if (ctx == NULL) {
    // context creation failed
}

ret = wolfSSL_CTX_UseSNI(ctx, 0, "www.yassl.com", strlen("www.yassl.com"));

if (ret != 0) {
    // sni usage failed
}

wolfSSL_CTX_SNI_SetOptions(ctx, WOLFSSL_SNI_HOST_NAME,
WOLFSSL_SNI_CONTINUE_ON_MISMATCH);
```

See Also:
wolfSSL_CTX_new
wolfSSL_CTX_UseSNI
wolfSSL_SNI_SetOptions

wolfSSL_SNI_SetOptions

Synopsis:
#include <wolfssl/ssl.h>

void wolfSSL_SNI_SetOptions(WOLFSSL* ssl, unsigned char type, unsigned char options);

Description:

This function is called on the server side to configure the behavior of the SSL session using Server Name Indication in the SSL object passed in the 'ssl' parameter. The options are explained below.

Return Values:

This function does not have a return value.

Parameters:

ssl - pointer to a SSL object, created with wolfSSL_new().

type - indicates which type of server name is been passed in data. The known types are:

```
enum {
    WOLFSSL_SNI_HOST_NAME = 0
};
```

options - a bitwise semaphore with the chosen options. The available options are:

```
enum {
    WOLFSSL_SNI_CONTINUE_ON_MISMATCH = 0x01,
    WOLFSSL_SNI_ANSWER_ON_MISMATCH   = 0x02
};
```

Normally the server will abort the handshake by sending a fatal-level unrecognized_name(112) alert if the hostname provided by the client mismatch with the servers.

WOLFSSL_SNI_CONTINUE_ON_MISMATCH - With this option set, the server will not send a SNI response instead of aborting the session.

WOLFSSL_SNI_ANSWER_ON_MISMATCH - With this option set, the server will send a SNI response as if the host names match instead of aborting the session.

Example:

```
int ret = 0;
```

```
WOLFSSL_CTX* ctx = 0;
WOLFSSL* ssl = 0;

ctx = wolfSSL_CTX_new(method);

if (ctx == NULL) {
    // context creation failed
}

ssl = wolfSSL_new(ctx);

if (ssl == NULL) {
    // ssl creation failed
}

ret = wolfSSL_UseSNI(ssl, 0, "www.yassl.com", strlen("www.yassl.com"));

if (ret != 0) {
    // sni usage failed
}

wolfSSL_SNI_SetOptions(ssl, WOLFSSL_SNI_HOST_NAME,
WOLFSSL_SNI_CONTINUE_ON_MISMATCH);
```

See Also:

wolfSSL_new
wolfSSL_UseSNI
wolfSSL_CTX_SNI_SetOptions

wolfSSL_SNI_GetRequest

Synopsis:

#include <wolfssl/ssl.h>

word16 wolfSSL_SNI_GetRequest(WOLFSSL *ssl, byte type, void** data);

Description:

This function is called on the server side to retrieve the Server Name Indication provided by the client in a SSL session.

Return Values:

The size of the provided SNI data.

Parameters:

ssl - pointer to a SSL object, created with wolfSSL_new().

type - indicates which type of server name is been retrieved in data. The known types are:

```
enum {
    WOLFSSL_SNI_HOST_NAME = 0
};
```

data - pointer to the data provided by the client.

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx = 0;
WOLFSSL* ssl = 0;

ctx = wolfSSL_CTX_new(method);

if (ctx == NULL) {
    // context creation failed
}

ssl = wolfSSL_new(ctx);

if (ssl == NULL) {
    // ssl creation failed
}

ret = wolfSSL_UseSNI(ssl, 0, "www.yassl.com", strlen("www.yassl.com"));

if (ret != 0) {
    // sni usage failed
}

if (wolfSSL_accept(ssl) == SSL_SUCCESS) {
    void *data = NULL;
    unsigned short size = wolfSSL_SNI_GetRequest(ssl, 0, &data);
}
```

See Also:

wolfSSL_UseSNI
wolfSSL_CTX_UseSNI

wolfSSL_SNI_GetFromBuffer

Synopsis:
#include <wolfssl/ssl.h>

WOLFSSL_API int wolfSSL_SNI_GetFromBuffer(const byte* clientHello, word32 helloSz, byte type, byte* sni, word32* inOutSz);

Description:
This function is called on the server side to retrieve the Server Name Indication provided by the client from the Client Hello message sent by the client to start a session. It does not requires context or session setup to retrieve the SNI.

Return Values:
If successful the call will return **SSL_SUCCESS**;
If there is no SNI extension in the client hello, the call will return **0**.

BAD_FUNC_ARG is the error that will be returned in one of this cases:
  * buffer is NULL
  * bufferSz <= 0
  * sni is NULL
  * inOutSz is NULL or <= 0

BUFFER_ERROR is the error returned when there is a malformed Client Hello message.

INCOMPLETE_DATA is the error returned when there is not enough data to complete the extraction.

Parameters:

buffer - pointer to the data provided by the client (Client Hello).

bufferSz - size of the Client Hello message.

type - indicates which type of server name is been retrieved from the buffer. The known types are:

```
enum {
    WOLFSSL_SNI_HOST_NAME = 0
```

};

sni - pointer to where the output is going to be stored.

inOutSz - pointer to the output size, this value will be updated to MIN("SNI's length", inOutSz).

Example:

```
unsigned char buffer[1024] = {0};
unsigned char result[32]   = {0};
int           length       = 32;

// read Client Hello to buffer...

ret = wolfSSL_SNI_GetFromBuffer(buffer, sizeof(buffer), 0, result, &length));
if (ret != SSL_SUCCESS) {
    // sni retrieve failed
}
```

See Also:
wolfSSL_UseSNI
wolfSSL_CTX_UseSNI
wolfSSL_SNI_GetRequest

wolfSSL_CTX_UseMaxFragment

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_UseMaxFragment(WOLFSSL_CTX* ctx, byte mfl);

Description:
This function is called on the client side to enable the use of Maximum Fragment Length for SSL objects created from the SSL context passed in the 'ctx' parameter. It means that the Maximum Fragment Length extension will be sent on ClientHello by wolfSSL clients.

Return Values:
If successful the call will return **SSL_SUCCESS**.

BAD_FUNC_ARG is the error that will be returned in one of these cases:
* ctx is NULL
* mfl is out of range.

MEMORY_E is the error returned when there is not enough memory.

Parameters:

ctx - pointer to a SSL context, created with wolfSSL_CTX_new().

mfl - indicates which is the Maximum Fragment Length requested for the session. The available options are:

```
enum {
    WOLFSSL_MFL_2_9  = 1, /*  512 bytes */
    WOLFSSL_MFL_2_10 = 2, /* 1024 bytes */
    WOLFSSL_MFL_2_11 = 3, /* 2048 bytes */
    WOLFSSL_MFL_2_12 = 4, /* 4096 bytes */
    WOLFSSL_MFL_2_13 = 5  /* 8192 bytes *//* wolfSSL ONLY!!! */
};
```

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx = 0;

ctx = wolfSSL_CTX_new(method);

if (ctx == NULL) {
    // context creation failed
}

ret = wolfSSL_CTX_UseMaxFragment(ctx, WOLFSSL_MFL_2_11);

if (ret != 0) {
    // max fragment usage failed
}
```

See Also:
wolfSSL_CTX_new
wolfSSL_UseMaxFragment

wolfSSL_UseMaxFragment

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_UseMaxFragment(WOLFSSL* ssl, byte mfl);

Description:
This function is called on the client side to enable the use of Maximum Fragment Length in the SSL object passed in the 'ssl' parameter. It means that the Maximum Fragment Length extension will be sent on ClientHello by wolfSSL clients.

Return Values:
If successful the call will return **SSL_SUCCESS**.

BAD_FUNC_ARG is the error that will be returned in one of these cases:
 * ssl is NULL
 * mfl is out of range.

MEMORY_E is the error returned when there is not enough memory.

Parameters:

ssl - pointer to a SSL object, created with wolfSSL_new().

mfl - indicates witch is the Maximum Fragment Length requested for the session. The available options are:

```
    enum {
        WOLFSSL_MFL_2_9  = 1, /*  512 bytes */
        WOLFSSL_MFL_2_10 = 2, /* 1024 bytes */
        WOLFSSL_MFL_2_11 = 3, /* 2048 bytes */
        WOLFSSL_MFL_2_12 = 4, /* 4096 bytes */
        WOLFSSL_MFL_2_13 = 5  /* 8192 bytes *//* wolfSSL ONLY!!! */
    };
```

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx = 0;
WOLFSSL* ssl = 0;
```

```
ctx = wolfSSL_CTX_new(method);

if (ctx == NULL) {
    // context creation failed
}

ssl = wolfSSL_new(ctx);

if (ssl == NULL) {
    // ssl creation failed
}

ret = wolfSSL_UseMaxFragment(ssl, WOLFSSL_MFL_2_11);

if (ret != 0) {
    // max fragment usage failed
}
```

See Also:
wolfSSL_new
wolfSSL_CTX_UseMaxFragment

wolfSSL_CTX_UseTruncatedHMAC

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_CTX_UseTruncatedHMAC(WOLFSSL_CTX* ctx);

Description:
This function is called on the client side to enable the use of Truncated HMAC for SSL objects created from the SSL context passed in the 'ctx' parameter. It means that the Truncated HMAC extension will be sent on ClientHello by wolfSSL clients.

Return Values:
If successful the call will return **SSL_SUCCESS**.

BAD_FUNC_ARG is the error that will be returned in one of these cases:
  * ctx is NULL

MEMORY_E is the error returned when there is not enough memory.

Parameters:

ctx - pointer to a SSL context, created with wolfSSL_CTX_new().

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx = 0;

ctx = wolfSSL_CTX_new(method);

if (ctx == NULL) {
    // context creation failed
}

ret = wolfSSL_CTX_UseTruncatedHMAC(ctx);

if (ret != 0) {
    // truncated HMAC usage failed
}
```

See Also:
wolfSSL_CTX_new
wolfSSL_UseMaxFragment

wolfSSL_UseTruncatedHMAC

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_UseTruncatedHMAC(WOLFSSL* ssl);

Description:
This function is called on the client side to enable the use of Truncated HMAC in the SSL object passed in the 'ssl' parameter. It means that the Truncated HMAC extension will be sent on ClientHello by wolfSSL clients.

Return Values:
If successful the call will return **SSL_SUCCESS**.

BAD_FUNC_ARG is the error that will be returned in one of these cases:

* ssl is NULL

MEMORY_E is the error returned when there is not enough memory.

Parameters:

ssl - pointer to a SSL object, created with wolfSSL_new().

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx = 0;
WOLFSSL* ssl = 0;

ctx = wolfSSL_CTX_new(method);

if (ctx == NULL) {
    // context creation failed
}

ssl = wolfSSL_new(ctx);

if (ssl == NULL) {
    // ssl creation failed
}

ret = wolfSSL_UseTruncatedHMAC(ssl);

if (ret != 0) {
    // truncated HMAC usage failed
}
```

See Also:

wolfSSL_new
wolfSSL_CTX_UseMaxFragment

wolfSSL_CTX_UseSupportedCurve

Synopsis:

#include <wolfssl/ssl.h>

int wolfSSL_CTX_UseSupportedCurve(WOLFSSL_CTX* ctx, word16 name);

Description:
This function is called on the client side to enable the use of Supported Elliptic Curves Extension for SSL objects created from the SSL context passed in the 'ctx' parameter. It means that the supported curves enabled will be sent on ClientHello by wolfSSL clients. This function can be called more than one time to enable multiple curves.
Return Values:
If successful the call will return **SSL_SUCCESS**.

BAD_FUNC_ARG is the error that will be returned in one of these cases:
* ctx is NULL
* name is a unknown value. (see below)

MEMORY_E is the error returned when there is not enough memory.

Parameters:

ctx - pointer to a SSL context, created with wolfSSL_CTX_new().

name - indicates which curve will be supported for the session. The available options are:
```
    enum {
        WOLFSSL_ECC_SECP160R1 = 0x10,
        WOLFSSL_ECC_SECP192R1 = 0x13,
        WOLFSSL_ECC_SECP224R1 = 0x15,
        WOLFSSL_ECC_SECP256R1 = 0x17,
        WOLFSSL_ECC_SECP384R1 = 0x18,
        WOLFSSL_ECC_SECP521R1 = 0x19
    };
```

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx = 0;

ctx = wolfSSL_CTX_new(method);

if (ctx == NULL) {
    // context creation failed
}
```

```
ret = wolfSSL_CTX_UseSupportedCurve(ctx, WOLFSSL_ECC_SECP256R1);

if (ret != 0) {
    // Elliptic Curve Extension usage failed
}
```

See Also:
wolfSSL_CTX_new
wolfSSL_UseSupportedCurve

wolfSSL_UseSupportedCurve

Synopsis:
#include <wolfssl/ssl.h>

int wolfSSL_UseSupportedCurve(WOLFSSL* ssl, word16 name);

Description:
This function is called on the client side to enable the use of Supported Elliptic Curves Extension in the SSL object passed in the 'ssl' parameter. It means that the supported curves enabled will be sent on ClientHello by wolfSSL clients. This function can be called more than one time to enable multiple curves.

Return Values:
If successful the call will return **SSL_SUCCESS**.

BAD_FUNC_ARG is the error that will be returned in one of these cases:
* ssl is NULL
* name is a unknown value. (see below)

MEMORY_E is the error returned when there is not enough memory.

Parameters:

ssl - pointer to a SSL object, created with wolfSSL_new().

name - indicates which curve will be supported for the session. The available options are:
enum {

```
        WOLFSSL_ECC_SECP160R1 = 0x10,
        WOLFSSL_ECC_SECP192R1 = 0x13,
        WOLFSSL_ECC_SECP224R1 = 0x15,
        WOLFSSL_ECC_SECP256R1 = 0x17,
        WOLFSSL_ECC_SECP384R1 = 0x18,
        WOLFSSL_ECC_SECP521R1 = 0x19
   };
```

Example:

```
int ret = 0;
WOLFSSL_CTX* ctx = 0;
WOLFSSL* ssl = 0;

ctx = wolfSSL_CTX_new(method);

if (ctx == NULL) {
    // context creation failed
}

ssl = wolfSSL_new(ctx);

if (ssl == NULL) {
    // ssl creation failed
}

ret = wolfSSL_UseSupportedCurve(ssl, WOLFSSL_ECC_SECP256R1);

if (ret != 0) {
    // Elliptic Curve Extension usage failed
}
```

See Also:

wolfSSL_CTX_new
wolfSSL_CTX_UseSupportedCurve