C# Wrapper

API Guide

December 7th 2015, version 1.1

1.0 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.

Init

Synopsis:
int 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 **SUCCESS**.

Parameters:

This function has no parameters.

Example:

```
int ret = 0;
ret = wolfssl.Init();
if (ret != wolfssl.SUCCESS) {
      // failed to initialize wolfssl library
}
```

See Also:
Cleanup

Cleanup

Synopsis:
void 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:
No return value for this function.

Parameters:

There are no parameters for this function.

Example:

```
...
wolfssl.Cleanup();
```

See Also:
Init

shutdown

Synopsis:
int shutdown(IntPtr 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 shutdown (to save resources). Either option is allowed by the TLS specification.

Return Values:

SUCCESS - will be returned upon success.

Parameters:

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

Example:

```
int ret = 0;
IntPtr ssl;
...

ret = wolfssl.shutdown(ssl);
if (ret != wolfssl.SUCCESS) {
      // failed to shut down SSL connection
}
```

See Also:
free
CTX_free

1.1 Certificates and Keys

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

SetTmpDH_file

Synopsis:
int SetTmpDH_file(IntPtr ssl, StringBuilder dhparam, int format);

Description:
This function loads a DH parameter key file into the SSL context. The file is provided by the **dhparam** 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 **SUCCESS**, otherwise **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;
IntPtr ssl;
StringBuilder dhparam = new StringBuilder(“dh2048.pem”);

...

ret = wolfssl.SetTmpDH_file(ssl, dhparam,
                                    wolfssl.SSL_FILETYPE_PEM);
if (ret != wolfssl.SUCCESS) {
      // error loading param file
}
```

```
...
```

CTX_SetMinDhKey_Sz

Synopsis:
int SetMinDhKey_Sz(IntPtr ctx, short sz);

Description:
This function sets the minimum DH key size into the SSL context(CTX context).

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

Example:

```
int ret = 0;
IntPtr ctx;
short sz = 512;

...

ret = wolfssl.SetMinDhKey_Sz(ctx, sz);
if (ret != wolfssl.SUCCESS) {
      // error setting key size
}

...
```

CTX_use_PrivateKey_file

Synopsis:
int CTX_use_PrivateKey_file(IntPtr ctx, string 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 **SUCCESS**, otherwise **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;
IntPtr ctx;

...

ret = wolfssl.CTX_use_PrivateKey_file(ctx, “server-key.pem”,
                                      wolfssl.SSL_FILETYPE_PEM);
if (ret != wolfssl.SUCCESS) {
      // error loading key file
}

...
```

CTX_use_certificate_file

Synopsis:

int CTX_use_certificate_file(IntPtr ctx, string 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 **SUCCESS**, otherwise **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 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;
IntPtr ctx;

...

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

...
```

1.2 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).

usev23_client

Synopsis:
IntPtr usev23_client(void);

Description:
The usev23_client() 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 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 usev23_client() 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:

```
IntPtr method;
IntPtr ctx;

method = wolfssl.usev23_client();
if (method == IntPtr.Zero) {
      // unable to get method
}

ctx = wolfssl.CTX_new(method);
...
```

See Also:
useTLSv1_2_client
useDTLSv1_2_client
CTX_new

usev23_server

Synopsis:
IntPtr usev23_server(void);

Description:
The usev23_server() 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 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:

```
IntPtr method;
IntPtr ctx;

method = wolfssl.usev23_server();
if (method == IntPtr.Zero) {
      // unable to get method
}

ctx = wolfssl.CTX_new(method);
...
```

See Also:
useTLSv1_2_server
useDTLSv1_server
CTX_new

useTLSv1_2_client

Synopsis:
IntPtr useTLSv1_2_client(void);

Description:
The useTLSv1_2_client() 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 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:

```
IntPtr method;
```

```
IntPtr ctx;

method = wolfssl.useTLSv1_2_client();
if (method == IntPtr.Zero) {
      // unable to get method
}

ctx = wolfssl.CTX_new(method);
...
```

See Also:
usev23_client
CTX_new

useTLSv1_2_server

Synopsis:
IntPtr useTLSv1_2_server(void);

Description:
The useTLSv1_2_server() 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 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:

```
IntPtr method;
IntPtr ctx;

method = wolfssl.useTLSv1_2_server();
if (method == IntPtr.Zero) {
      // unable to get method
```

```
}

ctx = wolfssl.CTX_new(method);
...
```

See Also:
usev23_server_method
CTX_new

useDTLSv1_2_client

Synopsis:
IntPtr useDTLSv1_2_client(void);

Description:
The useDTLSv1_2_client() function is used to indicate that the application is a client and will only support the DTLS 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 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:

```
IntPtr method;
IntPtr ctx;

method = wolfssl.useDTLSv1_2_client();
if (method == IntPtr.Zero) {
      // unable to get method
}

ctx = wolfssl.CTX_new(method);
...
```

See Also:
useTLSv1_2_client
usev23_client
CTX_new

useDTLSv1_2_server

Synopsis:
IntPtr useDTLSv1_2_server(void);

Description:
The .useDTLSv1_2_server() function is used to indicate that the application is a server and will only support the DTLS 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 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:

```
IntPtr method;
IntPtr ctx;

method = wolfssl.useDTLSv1_2_server();
if (method == IntPtr.Zero) {
      // unable to get method
}

ctx = wolfssl.CTX_new(method);
...
```

See Also:
useTLSv1_2_server
usev23_server

CTX_new

new_ssl

Synopsis:
IntPtr new_ssl(IntPtr 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 CTX_new().

Example:

```
IntPtr ssl;
IntPtr ctx;

ctx = wolfssl.CTX_new(method);
if (ctx == IntPtr.Zero) {
      // context creation failed
}

ssl = wolfssl.new_ssl(ctx);
if (ssl == NULL) {
      // SSL object creation failed
}
```

See Also:
CTX_new

free

Synopsis:
void free(IntPtr 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 new_ssl().

Example:

```
IntPtr ssl;
...
wolfssl.free(ssl);
```

See Also:
CTX_new
new_ssl
CTX_free

CTX_new

Synopsis:
IntPtr CTX_new(IntPtr 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 usevXX_XXXX() functions to specify SSL/TLS protocol level.

Example:

```
IntPtr ctx;
IntPtr method;

method = wolfssl.usev23_server
if (method == IntPtr.Zero) {
      // unable to get method
}

ctx = wolfssl.CTX_new(method);
if (ctx == IntPtr.Zero) {
      // context creation failed
}
```

See Also:
new_ssl
CTX_dtls_new

CTX_dtls_new

Synopsis:
IntPtr CTX_dtls_new(IntPtr method);

Description:
This function creates a new SSL context, taking a desired DTLS 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 useDTLSvXX_XXXX() functions to specify DTLS protocol level.

Example:

```
IntPtr ctx;
IntPtr method;

method = wolfssl.useDTLSv1_2_server
if (method == IntPtr.Zero) {
      // unable to get method
}

ctx = wolfssl.CTX_dtls_new(method);
if (ctx == IntPtr.Zero) {
      // context creation failed
}
```

See Also:
new_ssl
CTX_new

CTX_free

Synopsis:
void CTX_free(IntPtr ctx);

Description:
This function frees an allocated wolfSSL CTX object. Frees memory used for keeping alive any callbacks used such as PSK and IO delegates.

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

Parameters:

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

Example:

```
IntPtr ctx;
...
wolfssl.CTX_free(ctx);
```

See Also:
CTX_new
CTX_dtls_new
new_ssl
free

set_cipher_list

Synopsis:
int set_cipher_list(IntPtr ssl, StringBuilder 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:

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:

SUCCESS will be returned upon successful function completion, otherwise **FAILURE** will be returned on failure.

Parameters:

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

list - null-terminated text string and a colon-delimited list of cipher suites to use with the specified SSL session.

Example:

```
int ret = 0;
IntPtr ssl;
StingBuilder list = new
StringBuilder("DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:AES256-SHA256");
...
ret = wolfssl.set_cipher_list(ssl, list);
if (ret != wolfssl.SUCCESS) {
      // failed to set cipher suite list
}
```

See Also:
CTX_set_cipher_list
new_ssl

CTX_set_cipher_list

Synopsis:
int CTX_set_cipher_list(IntPtr ctx, StringBuilder 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 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:

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:

SUCCESS will be returned upon successful function completion, otherwise **FAILURE** will be returned on failure.

Parameters:

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

list - null-terminated text string and a colon-delimited list of cipher suites to use with the

specified SSL context.

Example:

```
IntPtr ctx;
StringBuilder list = new
StringBuilder("DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:AES256-SHA256");
...
ret = wolfssl.CTX_set_cipher_list(ctx, list);
if (ret != wolfssl.SUCCESS) {
      // failed to set cipher suite list
}
```

See Also:
set_cipher_list
CTX_new
CTX_dtls_new

get_ciphers

Synopsis:
int get_ciphers(StringBuilder list, int sz);

Description:
Creates a list of all available cipher suites.

Return Values:
If successful the call will return **SUCCESS**, otherwise, **FAILURE** will be returned.

Parameters:

list - buffer to fill with cipher suite names.

sz - size of buffer available to fill with cipher suite names.

Example:

```
int sz = 4096;
StringBuilder list = new StringBuilder(sz);
...
```

```
ret = wolfssl.get_ciphers(list, sz);
if (ret != wolfssl.SUCCESS) {
      // failed to set SSL file descriptor
}
```

get_current_cipher

Synopsis:
int get_current_cipher(IntPtr ssl);

Description:
Gets the current cipher suite being used in the SSL/TLS connection

Return Values:
Returns a string that has the current cipher suite being used by the ssl structure.

Parameters:

ssl - ssl context structure for connection.

Example:

```
string cipher;
IntPtr ssl;
...

cipher = wolfssl.get_ciphers(ssl);
```

get_version

Synopsis:
int get_version(IntPtr ssl);

Description:
Gets the current version being used in the SSL/TLS connection ie TLSv1.2.

Return Values:
Returns a string that has the current version being used by the ssl structure.

Parameters:

ssl - ssl context structure for connection.

Example:

```
string version;
IntPtr ssl;
...

version = wolfssl.get_version(ssl);
```

set_fd

Synopsis:
int set_fd(IntPtr ssl, Socket fd);

Description:
This function assigns a Socket (**fd**) as the input/output facility for the SSL connection.

Return Values:
If successful the call will return **SUCCESS**, otherwise, **FAILURE** will be returned.

Parameters:

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

fd - Socket to use with SSL/TLS connection.

Example:

```
Socket sockfd;
IntPtr ssl;
...

ret = wolfssl.set_fd(ssl, sockfd);
if (ret != wolfssl.SUCCESS) {
      // failed to set SSL file descriptor
}
```

See Also:
SetIOSend
SetIORecv

set_dtls_fd

Synopsis:
int set_dtls_fd(IntPtr ssl, UdpClient fd, IPEndPoint ep);

Description:
This function assigns a UdpCleint (**fd**) and IPEndPoint (**ep**) as the input/output facility for the SSL connection.

Return Values:
If successful the call will return **SUCCESS**, otherwise, **FAILURE** will be returned.

Parameters:

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

fd - UdpClient to use with DTLS connection.

ep - IPEndPoint to use with DTLS connection.

Example:

```
UdpClient sockfd;
IPEndPoint ep;
IntPtr ssl;
...

ret = wolfssl.set_dtls_fd(ssl, sockfd, ep);
if (ret != wolfssl.SUCCESS) {
      // failed to set SSL file descriptor
}
```

See Also:
SetIOSend
SetIORecv

1.3 Callbacks

The functions in this section have to do with callbacks which the application is able to set in relation to wolfSSL.

CallbackIORecv_delegate

Synopsis:
int CallbackIORecv_delegate(IntPtr ssl, IntPtr buf, int sz, IntPtr rctx);

Description:
This delegate is used for IO receive call back.

Return Values:
Size of message received.

Parameters:

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

buf - pointer buffer for storing the received message.

sz - the size of buffer available for message.

rctx - pointer to the context passed along during SSL receive.

See Also:
SetIORecv
SetIOSend

CallbackIOSend_delegate

Synopsis:
int CallbackIOSend_delegate(IntPtr ssl, IntPtr buf, int sz, IntPtr sctx);

Description:

This delegate is used for IO send call back.

Return Values:

Size of message sent.

Parameters:

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

buf - pointer to the message to send.

sz - size of message to send.

sctx - pointer to the context passed along during SSL send.

See Also:
wolfssl.SetIORecv
wolfssl.SetIOSend

SetIORecv

Synopsis:
void SetIORecv(IntPtr ctx, CallbackIORecv_delegate CBIORecv);

Description:
This function registers a receive callback for wolfssl to get input data. It is kept alive in memory by the CTX to avoid the garbage collector.

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 delegate

CallbackIORecv_delegate.

Example:

```
private int MyEmbedReceive(IntPtr ssl, IntPtr buf, int sz, IntPtr ctx)
{
      // custom EmbedReceive function
}

...
IntPtr ctx;

CallbackIORecv_delegate myCb = new CallbackIORecv_delegate(MyEmbedReceive);
// Register the custom receive callback with wolfssl
wolfssl.SetIORecv(ctx, myCB);

...
```

See Also:
SetIOSend

SetIOSend

Synopsis:
void SetIOSend(IntPtr ctx, CallbackIOSend_delegate CBIOSend);

Description:
This function registers a send callback for wolfssl to write output data. Memory used for delegate is kept alive by the CTX.

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

Parameters:

ctx - pointer to the SSL context, created with 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

inCallbackIOSend_delegate.

Example:

```
private int MyEmbedSend(IntPtr ssl, IntPtr buf, int sz, IntPtr ctx)
{
      // custom EmbedSend function
}

...
IntPtr ctx;

CallbackIOSend_delegate myCB = new CallbackIOSend(MyEmbedSend);
// Register the custom receive callback with wolfssl
wolfssl.SetIOSend(ctx, myCb);

...
```

See Also:
SetIORecv

SetLogging

Synopsis:
int SetLogging(wolfssl.loggingCB func);

delegate loggingCB(int logLevel, StringBuilder logMessage);

Description:
This function registers a logging callback that will be used to handle the C# wolfssl log message. By default, no logging messages are printed out this is for security so the user can direct where logging messages are displayed at.

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

Parameters:

wolfssl.loggingCB- function to register as a logging callback. Function signature must follow the above prototype.

Example:

```
int ret = 0;

// Logging callback prototype
void MyLoggingCallback(int logLevel, StringBuilder logMessage);

// Register the custom logging callback with wolfssl
ret = wolfssl.SetLogging(myLogCallback);
if (ret != 0) {
      // failed to set logging callback
}

void MyLoggingCallback(int logLevel, StringBuilder logMessage)
{
      // custom logging function
}
```

See Also:

1.4 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.

get_error

Synopsis:
string get_error(IntPtr ssl);

Description:
This function converts an error into a more human-readable error string. Providing the error number concatenated with the error description.

Return Values:
On successful completion, this function returns the string containing the error value and description.

Example:

```
int err = 0;
IntPtr ssl;
string error;
...
error = wolfssl.get_error(ssl);
Console.WriteLine(error);
```

See Also:

1.5 Connection, Session, and I/O

The functions in this section deal with setting up the SSL/TLS connection, managing SSL sessions, and input/output.

accept

Synopsis:
int accept(IntPtr 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.

With the underlying I/O being blocking, accept() will only return once the handshake has been finished or an error occurred.

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

To get a more detailed error code if SUCCESS is not returned, call get_error().

Parameters:

ssl - a pointer to a wolfssl structure, created using new_ssl().

Example:

```
int ret = 0;
string err;
IntPtr ssl;
string buffer;
...

ret = wolfssl.accept(ssl);
if (ret != wolfssl.SUCCESS) {
      err = wolfssl.get_error(ssl);
      Console.WriteLine(err);
}
```

See Also:
get_error
connect

connect

Synopsis:
int connect(IntPtr 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.

With the underlying I/O is blocking, 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).

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

To get a more detailed error code, call get_error().

Parameters:

ssl - a pointer to a wolfssl structure, created using new_ssl().

Example:

```
int ret = 0;
string err;
IntPtr ssl;
char buffer;
...

ret = wolfssl.connect(ssl);
if (ret != wolfssl.SUCCESS) {
      err = wolfssl.get_error(ssl);
      Console.WriteLine(err);
}
```

See Also:
get_error
accept

get_fd

Synopsis:
int get_fd(IntPtr ssl);

Description:
This function returns the Socket information used as the input/output facility for the SSL connection.

Return Values:
If successful the call will return the SSL session Socket information.

Parameters:

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

Example:

```
Socket sockfd;
IntPtr ssl;
...
sockfd = wolfssl.get_fd(ssl);
...
```

See Also:
set_fd
set_dtls_fd
get_dtls_fd

get_dtls_fd

Synopsis:
intget_dtls_fd(IntPtr ssl);

Description:
This function returns the DTLS_con class with information used as the input/output facility for the DTLS connection.

Return Values:
If successful the call will return the DTLS session connection information.

Parameters:

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

Example:

```
wolfssl.DTLS_con sockfd;
IntPtr ssl;
...
sockfd = wolfssl.get_dtls_fd(ssl);
...
```

See Also:
set_fd
set_dtls_fd

read

Synopsis:

int read(IntPtr ssl, StringBuilder 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 necessaryread() will negotiate an SSL/TLS session if the handshake has not already been performed yet by connect() or 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 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 read().

If **sz** is larger than the number of bytes in the internal read buffer,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 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 get_error() for the specific error code.

Use get_error() to get a specific error code.

Parameters:

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

data - buffer where read() will place data read.

sz - number of bytes to read into **data**.

Example:

```
IntPtr ssl;
StringBuilder reply;
...

input = wolfssl.read(ssl, reply, reply.Length);
if (input > 0) {
      // "input" number of bytes returned into buffer "reply"
}
```

See wolfssl examples for more complete examples of read().

See Also:
write

read (using byte[])

Synopsis:
int read(IntPtr ssl, byte[] 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. To better facilitate working with raw bytes this function allows for using a byte array.

If necessary read() will negotiate an SSL/TLS session if the handshake has not already been performed yet by connect() or 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

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 read().

If **sz** is larger than the number of bytes in the internal read buffer,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 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 get_error() for the specific error code.

Use get_error() to get a specific error code.

Parameters:

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

data - byte array buffer where read() will place data read.

sz - number of bytes to read into **data**.

Example:

```
IntPtr ssl;
byte[] reply = new byte[1024];
...

input = wolfssl.read(ssl, reply, reply.Length);
if (input > 0) {
      // "input" number of bytes returned into buffer "reply"
}
```

See wolfssl examples for more complete examples of read().

See Also:
write

write

Synopsis:
int write(IntPtr ssl, const StringBuilder data, int sz);

Description:
This function writes **sz** bytes from the buffer, **data**, to the SSL connection, **ssl**.

If necessary, write() will negotiate an SSL/TLS session if the handshake has not already been performed yet by connect() or accept().

With the underlying I/O is blocking, 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 get_error() for the specific error code.

Use 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:

```
IntPtr ssl;
StringBuilder msg = new StringBuilder("hello wolfssl!");
```

```
int msgSz = msg.Length;
int flags;
int ret;
...

ret = wolfssl.write(ssl, msg, msgSz);
if (ret <= 0) {
      // wolfssl.write() failed, call wolfssl.get_error()
}
```

See wolfssl examples for more more detailed examples of wolfssl.write().

See Also:
read

write (using byte[])

Synopsis:
int write(IntPtr ssl, const byte[] data, int sz);

Description:
This function writes **sz** bytes from the buffer, **data**, to the SSL connection, **ssl**. Is to provided easier use when working with raw bytes, allowing for users to pass the raw bytes to the wolfssl write function.

If necessary, write() will negotiate an SSL/TLS session if the handshake has not already been performed yet by connect() or accept().

With the underlying I/O is blocking, 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 get_error() for the specific error code.

Use get_error() to get a specific error code.

Parameters:

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

data - byte array data buffer which will be sent to peer.

sz - size, in bytes, of data to send to the peer (**data**).

Example:

```
IntPtr ssl;
byte[] msg = new byte[] {0 ,1 ,7, 255};
int msgSz = msg.Length;
int ret;
...

ret = wolfssl.write(ssl, msg, msgSz);
if (ret <= 0) {
      // wolfssl.write() failed, call wolfssl.get_error()
}
```

See wolfssl examples for more more detailed examples of wolfssl.write().

See Also:
read

1.6 PSK

The functions in this section deal with setting up a PSK connection.

CTX_use_psk_identity_hint

Synopsis:
int CTX_use_psk_identity_hint(IntPtr ctx, StingBuilder hint);

Description:
This function is called to set the hint to be sent across during a PSK connection.

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

Parameters:

ctx - a pointer to a wolfssl structure, created using wolfssl.CTX_new().

hint - the hint to send across.

Example:

```
IntPtr ctx;
int ret;
StringBuffer hint = new StringBuilder("wolfssl server");
...

ret = wolfssl.CTX_use_psk_identity_hint(ctx, hint);
if (ret != wolfssl.SUCCESS) {
      // handle error
}
```

See Also:
CTX_set_psk_server_callback

CTX_set_psk_server_callback

Synopsis:
void CTX_set_psk_server_callback(IntPtr ctx, wolfssl.psk_delegate psk_cb);

uint psk_delegate(IntPtr ssl, string identity, IntPtr key, uint max_sz);

Description:
This function is used to set the callback function for when performing a PSK handshake.

Return Values:
Does not return anything.

Parameters:

ctx - a pointer to a wolfssl structure, created using wolfssl.CTX_new().

psk_delegate - a pointer to a psk callback function.

Example:

```
int ret = 0;
IntPtr ctx;
wolfssl.psk_delegate psk_cb = new wolfssl.psk_delegate(my_psk_server_cb);
...

ret = wolfssl.CTX_set_psk_server_callback(ctx, psk_cb);
...
```

See Also:
CTX_use_psk_identity_hint
set_psk_server_callback

set_psk_server_callback

Synopsis:
void set_psk_server_callback(IntPtr ctx, wolfssl.psk_delegate psk_cb);

uint psk_delegate(IntPtr ssl, string identity, IntPtr key, uint max_sz);

Description:
This function is used to set the callback function for when performing a PSK handshake.

Return Values:
Does not return anything.

Parameters:

ssl - a pointer to a wolfssl structure, created using wolfssl.new_ssl().

psk_delegate - a pointer to a psk callback function.

Example:

```
IntPtr ssl;
wolfssl.psk_delegate psk_cb = new wolfssl.psk_delegate(my_psk_server_cb);
```

```
...

wolfssl.set_psk_server_callback(ssl, psk_cb);
...
```

See Also:
CTX_use_psk_identity_hint
CTX_set_psk_server_callback

1.7 Appendix

A list of preprocessor flags and what they do

1.7.1 Removing Features

The following defines can be used to remove features from wolfSSL. This can be helpful if you are trying to reduce the overall library footprint size. In addition to defining a **NO_<feature-name>** define, you can also remove the respective source file as well from the build (but not the header file).

> **NO_WOLFSSL_CLIENT** removes calls specific to the client and is for a server-only builds. You should only use this if you want to remove a few calls for the sake of size.
>
> **NO_WOLFSSL_SERVER** likewise removes calls specific to the server side.
>
> **NO_DES3** removes the use of DES3 encryptions. DES3 is built-in by default because some older servers still use it and it's required by SSL 3.0.
>
> **NO_DH** and **NO_AES** are the same as the two above, they are widely used.
>
> **NO_DSA** removes DSA since it's being phased out of popular use.
>
> **NO_ERROR_STRINGS** disables error strings. Error strings are located in src/internal.c for wolfSSL or wolfcrypt/src/asn.c for wolfCrypt.

NO_HMAC removes HMAC from the build.

NO_MD4 removes MD4 from the build, MD4 is broken and shouldn't be used.

NO_MD5 removes MD5 from the build.

NO_SHA256 removes SHA-256 from the build.

NO_PSK turns off the use of the pre-shared key extension. It is built-in by default.

NO_PWDBASED disables password-based key derivation functions such as PBKDF1, PBKDF2, and PBKDF from PKCS #12.

NO_RC4 removes the use of the ARC4 stream cipher from the build. ARC4 is built-in by default because it is still popular and widely used.

NO_RABBIT and **NO_HC128** remove stream cipher extensions from the build.

NO_SESSION_CACHE can be defined when a session cache is not needed. This should reduce memory use by nearly 3 kB.

NO_TLS turns off TLS. We don’t recommend turning off TLS.

SMALL_SESSION_CACHE can be defined to limit the size of the SSL session cache used by wolfSSL. This will reduce the default session cache from 33 sessions to 6 sessions and save approximately 2.5 kB.

1.7.2 Enabling Features Disabled by Default

WOLFSSL_DTLS turns on the use of DTLS, or datagram TLS. This isn't widely supported or used so it is off by default.

WOLFSSL_RIPEMD enables RIPEMD-160 support.

WOLFSSL_SHA384 enables SHA-384 support.

WOLFSSL_SHA512 enables SHA-512 support.

HAVE_AESCCM enables AES-CCM support.

HAVE_AESGCM enables AES-GCM support.

HAVE_CAMELLIA enables Camellia support.

HAVE_CHACHA enables ChaCha20 support.

HAVE_POLY1305 enables Poly1305 support.

HAVE_ECC enables Elliptical Curve Cryptography (ECC) support.

HAVE_CSHARP turns on configuration options needed for C# wrapper. Will enable psk and dtls overriding any NO_PSK flag preset.

1.7.3 Customizing or Porting wolfSSL

WOLFSSL_USER_SETTINGS if defined allows a user specific settings file to be used. The file must be named “user_settings.h” and exist in the include path. This is included prior to the standard “settings.h” file, so default settings can be overridden.

NO_INLINE disables the automatic inlining of small, heavily used functions. Turning this on will slow down wolfSSL and actually make it bigger since these are small functions, usually much smaller than function call setup/return. You’ll also need to add wolfcrypt/src/misc.c to the list of compiled files if you’re not using autoconf.

NO_DEV_RANDOM disables the use of the default /dev/random random number generator. If defined, the user needs to write an OS-specific GenerateSeed() function (found in “wolfcrypt/src/random.c”).

SINGLE_THREADED is a switch that turns off the use of mutexes. wolfSSL currently only uses one for the session cache. If your use of wolfSSL is always single threaded you can turn this on.

USER_TICKS allows the user to define their own clock tick function if time(0) is

not wanted. Custom function needs second accuracy, but doesn't have to be correlated to EPOCH. See LowResTimer() function in "wolfssl_int.c".

USER_TIME disables the use of time.h structures in the case that the user wants (or needs) to use their own. See "wolfcrypt/src/asn.c" for implementation details. The user will need to define and/or implement XTIME, XGMTIME, and XVALIDATE_DATE.

1.7.4 Reducing Memory Usage

TFM_TIMING_RESISTANT can be defined when using fast math (USE_FAST_MATH) on systems with a small stack size. This will get rid of the large static arrays.

WOLFSSL_SMALL_STACK can be used for devices which have a small stack size. This increases the use of dynamic memory in wolfcrypt/src/integer.c, but can lead to slower performance.

1.7.5 Increasing Performance

WOLFSSL_AESNI enables use of AES accelerated operations which are built into some Intel chipsets. When using this define, the aes_asm.s file must be added to the wolfSSL build sources.

USE_FAST_MATH switches the big integer library to a faster one that uses assembly if possible. fastmath will speed up public key operations like RSA, DH, and DSA. The big integer library is generally the most portable and generally easiest to get going with, but the negatives to the normal big integer library are that it is slower and it uses a lot of dynamic memory. Because the stack memory usage can be larger when using fastmath, we recommend defining TFM_TIMING_RESISTANT as well when using this option.