SSLRef 3.0 API Details

SSLRef 3.0 Final -- 11/19/96

Copyright (c)1996 by Netscape Communications Corp.

By retrieving this software you are bound by the licensing terms disclosed in the file "LICENSE.txt". Please read it, and if you don't accept the terms, delete this software.

SSLRef 3.0 was codeveloped by Netscape Communications Corp. of Mountain View, California <http://home.netscape.com/> and Consensus Development Corporation of Berkeley, California <http://www.consensus.com>.

Contents

Preface

This document was last updated on November 22, 1996 by Eric Gundrum (Consensus Development) <eric@macgroup.com>. It is based on original documents written by Tim Dierks (Consensus Development) <timd@consensus.com>.

This document describes the programmers interface of the SSLRef 3.0 library. SSLRef 3.0 implements the SSL 3.0 protocol.

Before reading this document, you should be familiar with the SSL 3.0 protocol. You can find more information about SSL 3.0 from <http://home.netscape.com/newsref/std/SSL.html>.

You may also find it helpful to read about how SSLRef works, described in API_Introduction.txt, and what SSLRef does, described in Implementation_Notes.txt.

Definitions

Throughout this document these terms are used exclusively with these definitions:

- **Client** refers to any software implementing the client side of the SSL protocol.
- **Server** refers to any software implementing the server side of the SSL software.
- **Peer** refers to the client or server to which you are connecting.
- **User** refers to the software that is making use of the SSLRef library.

Public API

This section describes each function call of the public API. These calls are declared in ssl.h. These descriptions include the purpose of the call, how it is normally used, and how it affects the state of the SSL connection.

Before reading this section, you should have read API_Introduction.txt to learn about the architecture of SSLRef 3.0, and you should review the sample code for SSLRef 3.0 to learn how the library is typically used.

Summary of API functions

- **Data Structures** describes the public data structures used by the library.
- **Connection Creation and Deletion** represents a group of calls used to create and destroy whole SSL connection contexts.
- **Connection Configuration** represents a group of calls used to setup and configure the state of the SSL connection context.
- **Connection Status** represents a group of calls used to examine the state of the SSL connection context.
- **Data Exchange** functions are used to communicate over the physical connection using the SSL protocol. These include SSLHandshake, SSLRead and SSLWrite.
- **Application Callback** functions identify several groups of functions implemented by the user to provide platform-dependent capabilities, such as memory allocation and data exchange over the physical connection. Additional callbacks are used to provide capabilities best tailored to the specific needs of the user.

Data Structures

General Data Types

The library generally uses ANSI C data types wherever possible. However, when the specific size of the data is critical to the application, these general data types are used: sint8, uint8, sint16, uint16, sint32, uint32.

SSLBuffer

Many functions require a block of data upon which to act. These data blocks are passed as an SSLBuffer structure specifying the length of the data block and a pointer to the data block.

SSLContext

For each SSL connection, a connection instance (context) is created. This context serves as a constant reference to that connection supplied by the user. The context parameter, of type SSLContext, is used by nearly all functions in the public API. (Internally it is labeled as **ctx**.)

A SSLContext instance stores the cryptographic data associated with the connection, certificates, references to the I/O callbacks, and various other data used by the library.

SSLContext Is an Opaque Data Structure

SSLContext , like all other structures internal to the SSLRef implementation, is treated by the API as an opaque value.

The details of this structure are not exposed to the user. The user must call through the API to modify or view fields of an SSLContext instance.

This separation of interface and implementation serves three purposes:

1. This separation creates a clear distinction between data members which are part of the public interface and those which are not.
2. This separation allows the library developer to modify the SSLRef implementation without requiring modification of the library's clients.
3. This separation allows the library to be notified when a value changes, allowing for greater flexibility in adding features to the SSLRef library.

SSLRef Is Thread-safe, But SSLContext Is Not Reentrant

The library relies on the user to maintain an independent context for each connection instead of using static or global data. This allows the library to remain thread-safe, but the SSLContext data structure is not reentrant.

The library assumes that exactly one thread owns a specific SSLContext instance. There is no library-based protection against multiple library threads attempting to update the same SSLContext instance. If multiple threads are sharing a particular SSLContext instance, the user is responsible for resolving contention for access to fields within the SSLContext instance.

For the library to be used in a threaded environment, the user implemented callbacks must be thread-safe. Note that all active sessions can simultaneously use callbacks to access the same database information about resumable sessions, for example.

Reference Parameters

Included in the parameter list of most callback functions is a reference parameter. This reference parameter can be used as a pointer by which data can be passed through the library to the callback functions.

For example, an I/O callback used to implement receiving data might use its reference parameter as a pointer to the socket number or other identifier required to identify the stream for a particular connection.

The reference parameter is stored as type void*. However, the library will never dereference or otherwise access a reference parameter, so they need not be pointers.

Reference parameters are stored in the same SSLContext structure as the callback function pointers. Because the reference parameter is how the callback functions are expected to distinguish between different physical connections, the SSLContext structures must be unique for each SSL connection. Typically a generic context structure is created and held as a template. When an SSL connection instance is created, the template structure is duplicated and its unique fields, such as the I/O reference parameter, are set as needed.

An individual callback may share its reference parameter with other similar callbacks. For example, the Read and Write callbacks will both receive the same reference parameter value when called for a particular connection. Review the list of application callbacks to see how the reference parameters are shared.

SSLRef Error Return Values

Most SSLRef routines can return an error if the routine does not complete as expected. SSLRef expects the user-defined callback functions to return similar errors where appropriate. Errors from the callback functions are generally passed up the call chain and ultimately are returned to the caller of the SSL API function. These SSLErrors are declared in sslerrors.h:

SSLNoErr	SSLMemoryErr
SSLUnsupportedErr	SSLOverflowErr
SSLUnknownErr	SSLProtocolErr
SSLNegotiationErr	SSLFatalAlert
SSLWouldBlockErr	SSLIOErr
SSLConnectionClosedGraceful	SSLConnectionClosedError
ASNBadEncodingErr	ASNIntegerTooBigErr
X509CertChainInvalidErr	X509NamesNotEqualErr

Connection Creation and Deletion

Calls used to create and delete the SSL connection context.

uint32 SSLContextSize(void);

The Purpose Of This Function

Determine the amount of memory required by a SSLContext structure.

How the Parameters are Used

The return value indicates how many bytes should be allocated for use by the SSLContext structure.

How This Function Affects the Library

SSLContextSize() does not affect the library.

How Errors Are Handled

SSLContextSize() cannot encounter an error.

SSLErr SSLInitContext(SSLContext *ctx);

The Purpose Of This Function

Initialize the internal values of the SSLContext structure. SSLInitContext() must be called before other SSLSet... functions.

How the Parameters are Used

`ctx` points to a memory buffer to be initialized as a SSLContext with default values.

How This Function Affects the Library

The internal values of `ctx` are initialized.

If `ctx` points to a SSLContext currently in use, its values will be initialized without regard to its associated SSL connection.

SSLInitContext() does not affect any internal values set by calls to SSLSet.... SSLInitContext and the various SSLSet... calls can be made in any order; however, the memory allocation callbacks should be set before others that will allocate memory.

How Errors Are Handled

SSLInitContext() cannot encounter an error.

SSLErr SSLDeleteContext(SSLContext *ctx);

The Purpose Of This Function

Release all resources held by a SSLContext structure. The SSLContext structure is not deallocated.

How the Parameters are Used

`ctx` points to the context to be released. The user is still responsible for deallocating the SSLContext structure after the SSLDeleteContext call.

How This Function Affects the Library

Heap memory and other resources of `ctx` are released. Memory occupied by `ctx` is not released.

If the SSL connection has not been closed, it should be closed by calling SSLClose before deleting the SSLContext.

How Errors Are Handled

SSLDeleteContext() cannot encounter an error.

SSLErr SSLDuplicateContext (SSLContext *src, SSLContext *dest, void *ioRef);

The Purpose Of This Function

Initialize a SSLContext structure using another as a template.

How the Parameters are Used

`src` points to the template context.

`dest` points to the context to be initialized.

`ioRef` points to the I/O reference parameter to be used when initializing the new SSLContext (`dest`).

How This Function Affects the Library

The contents of `src` are unaffected by this call.

The contents of `dest` are initialized by the library using the information in `src` where appropriate. `ioRef` is also added to dest as if SSLSetIORef(dest, ioRef) had been called.

Even if `src` represents an active SSL connection, `dest` is set as an inactive SSL connection as if `dest` were created from scratch using the SSLSet... calls. SSLDuplicateContext() is merely a short cut to avoid repeated SSLSet... calls when most contexts will be initialized with the same data.

How Errors Are Handled

Errors returned by the allocation callbacks can be returned. Typically this will be SSLMemoryErr.

X509CertChainInvalidErr and X509NamesNotEqualErr also can be returned if there is a problem with the certificate chain, but these are unlikely.

Connection Configuration

Calls used to configure the SSL connection context.

SSLErr SSLSetProtocolSide (SSLContext *ctx, SSLProtocolSide side);

The Purpose Of This Function

Specify whether to connect as an SSL server or SSL client.

How the Parameters are Used

`side` must be either SSL_ServerSide or SSL_ClientSide.

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

SSL_ServerSide specifies that the library will act as a SSL server for all communications using this context.

SSL_ClientSide specifies that the library will act as a SSL client for all communications using this context.

How Errors Are Handled

SSLSetProtocolSide() cannot encounter an error.

SSLErr SSLSetProtocolVersion (SSLContext *ctx, SSLProtocolVersion version);

The Purpose Of This Function

Specify what versions of the SSL protocol can be used for the connection.

How the Parameters are Used

`version` specifies what versions of the SSL protocol can be used for the connection as described below.

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

The library negotiates connections differently based on the value of version:

SSL_Version_Undetermined

Use this option to connect to SSL 2.0 and SSL 3.0 peers.

The library will negotiate either SSL 2.0 or SSL 3.0.

The library will send initial Hello messages using SSL 2.0, then attempt to negotiate up to SSL 3.0. If the peer does not support SSL 3.0, then communication will continue using SSL 2.0.

SSL_Version_3_0_With_2_0_Hello

Use this option to force an SSL 3.0 connection, but allow a meaningful error message when SSL 2.0-only peers attempt a connection.

The library will negotiate only SSL 3.0, but it will support an initiating SSL 2.0 Hello.

The library will send initial Hello messages using SSL 2.0, then attempt to negotiate up to SSL 3.0. If the peer does not support SSL 3.0, then handshaking will fail and further communication will cease.

SSL 2.0 communications is not permitted in this mode.

SSL_Version_3_0_Only

Use this option for the highest security when you always know that the SSL peer supports SSL 3.0.

The library will send initial Hello messages using SSL 3.0 and all communications will occur using SSL 3.0. If the peer does not support SSL 3.0, the handshake will fail without a meaningful indication of why it failed.

This setting can cause problems with some SSL 2.0 servers that will block against a read that will never complete because the server does not understand the SSL 3.0 Hello.

SSL_Version_2_0

Use this option to restrict all communications to SSL 2.0.

The library will send initial Hello messages using SSL 2.0. If the peer supports SSL 2.0, further communication will take place using SSL 2.0.

Version Handshaking Summary

Client Setting	Server Setting			
	Undetermined	3.0 W/2.0 Hello	3.0 Only	2.0 Only
Undetermined	3.0	3.0	(a)	2.0
3.0 W/2.0 Hello	3.0	3.0	(a)	(b)
3.0 Only	3.0	3.0	3.0	(c)
2.0 Only	2.0	(d)	(e)	2.0

Each entry specifies the negotiated protocol version. Alphabetic notes indicate why negotiation is impossible:

(a) These protocols support SSL 3.0, but the SSL 3.0 setting on the server prevents the SSL 2.0 Hello message they send from being recognized.

(b) The SSL 2.0 Hello message sent by the client is recognized, but the server will respond with a 2.0 response. The client rejects this response; it is set to communicate using only SSL 3.0.

(c) The SSL 3.0 Hello message sent by the client will not be understood by the SSL 2.0 server.

(d) The server understands the SSL 2.0 message, but it is configured to not communicate with a SSL 2.0-only client.

(e) The server will not be able to parse the SSL 2.0 Hello message sent by the client.

How Errors Are Handled

SSLSetProtocolVersion() does not check for errors.

SSLErr SSLSetPrivateKey (SSLContext *ctx, SSLRSAPrivateKey *privKey);

The Purpose Of This Function

Specify the private key used for signing and non-export key exchange. This private key must match the installed public key.

How the Parameters are Used

`privKey` points to a valid RSAREF or BSAFE formatted private key. The desired format is specified by setting BSAFE or RSAREF compile flag to a non-zero value. The library will free this block after closing the SSL connection.

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

This SSLContext structure's private key field is set to the specified pointer.

How Errors Are Handled

SSLSetPrivateKey() does not check for errors.

SSLErr SSLSetExportPrivateKey (SSLContext *ctx, SSLRSAPrivateKey *privKey);

The Purpose Of This Function

Specify the export private key used for this SSL connection. This key should be less than 512 bits for compatibility with exportable SSL implementations. Note that this key is required when using the `RSA_EXPORT` Cipher Suite.

How the Parameters are Used

`privKey` points to a valid RSAREF or BSAFE formatted private key. The desired format is specified by setting BSAFE or RSAREF compile flag to a non-zero value. The library will free this block after closing the SSL connection.

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

This SSLContext structure's export private key field is set to the specified pointer.

How Errors Are Handled

SSLSetExportPrivateKey() does not check for errors.

SSLErr SSLSetDHAnonParams (SSLContext *ctx, SSLDHParams *dhAnonParams);

The Purpose Of This Function

Specify the parameters used for anonymous Diffie-Hellman calculations.

How the Parameters are Used

`dhAnonParams` points to a valid RSAREF or BSAFE formatted set of Diffie-Hellman parameters. The desired format is specified by setting BSAFE or RSAREF compile flag to a non-zero value. The library will free this block after closing the SSL connection.

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

This SSLContext structure's Diffie-Hellman parameters field is set to the specified pointer. This value is passed to RSAREF or BSAFE as needed by the libraries.

How Errors Are Handled

SSLSetDHAnonParams() does not check for errors.

SSLErr SSLSetRequestClientCert (SSLContext *ctx, int requestClientCert);

The Purpose Of This Function

Specify if certificates should be requested from connecting SSL clients.

How the Parameters are Used

`requestClientCert` specifies that the peer is asked for a certificate if the peer is acting as an SSL client.

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

This SSLContext structure's `requestClientCert` field is set to the specified value. A non-zero value directs the library to request a certificate when handshaking with an SSL client.

How Errors Are Handled

SSLSetRequestClientCert() does not check for errors.

SSLErr SSLAddCertificate (SSLContext *ctx, SSLBuffer derCert, int parent, int complete);

The Purpose Of This Function

Add a certificate to the chain of certificates used when authenticating to an SSL peer. This is necessary when using in a server that supports RSA key exchange. This is optional for clients.

How the Parameters are Used

`derCert` is a DER-encoded X.509 (v1, 2 or 3) certificate.

`parent` set to a non-zero value specifies that this certificate is a parent to certificates previously added.

`complete` set to a non-zero value specifies that this is the last certificate to be added.

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

Certificates are added to the chain as parents or children of other certificates previously added. When `complete` is specified, SSLRef will attempt to verify the certificate chain.

How Errors Are Handled

Possible errors returned include allocation errors, `X509CertChainInvalidErr` and `X509NamesNotEqualErr`.

SSLErr SSLAddDistinguishedName (SSLContext *ctx, SSLBuffer derDN);

The Purpose Of This Function

Add a distinguished name to the list of acceptable distinguished names used when acting as a server and requesting client authentication. This list is transmitted to the client when requesting a client certificate.

How the Parameters are Used

`derDN` is a DER-encoded X.509 (v1, 2 or 3) Distinguished Name.

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

The distinguished name is added to the SSLContext structure.

How Errors Are Handled

Errors returned by the allocation callbacks can be returned. Typically this will be SSLMemoryErr.

SSLErr SSLSetPeerID(SSLContext *ctx, SSLBuffer peerID);

The Purpose Of This Function

Uniquely identify the SSL peer associated with this SSL connection. Setting a peer ID activates the session resumption capabilities for this session.

How the Parameters are Used

`peerID` is an opaque buffer. When acting as a client, the library passes it to the session database callback functions as the key and without regard for its contents; it should be a representation of the peer's network address and port, or some other uniquely identifying value. When acting as a server, the library uses the existence of a `peerID` buffer to indicate that this session might be resumed; the library uses the session ID provided by the client as the key passed to the session database callback functions. When acting as a server, the contents of `peerID` are not used, but the buffer must exist. In all cases, the library will free this buffer when it is done with it.

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

The library uses callback functions to retrieve existing peer information to resume the SSL session. If the peer is a server and is not in the resumable connection database, the library uses this value as a key when adding the resumption data for this session. If the peer is a client, the library uses the client's session ID as the key when adding the resumption data for this session.

If this function is not called for an SSLContext, the library will not attempt to resume a session with this peer, nor will the library attempt to store any session data.

How Errors Are Handled

Errors returned by the allocation callbacks can be returned. Typically this will be SSLMemoryErr.

Connection Status

Calls used to examine the SSL connection context. These functions provide read access to the SSLContext fields that reflect the status of the connection. There are no functions for retrieving fields set by the user.

SSLErr SSLGetProtocolVersion (SSLContext *ctx, SSLProtocolVersion *version);

The Purpose Of This Function

Examine what version of the SSL protocol is being used for the connection. (The protocol version is normally negotiated between the SSL peers. See the description for SSLSetProtocolVersion.)

How the Parameters are Used

`version` points to a container into which the version identifier is copied as described below.

`ctx` points to the context structure defining the SSL connection being examined.

How This Function Affects the Library

Before the protocol is negotiated, `version` will contain whatever value is set by the user.

After the protocol is negotiated, `version` will contain one of these values:

SSL_Version_2_0

The library is communicating with the peer using the SSL 2.0 protocol.

SSL_Version_3_0

The library is communicating with the peer using the SSL 3.0 protocol.

How Errors Are Handled

SSLGetProtocolVersion() does not check for errors.

SSLErr SSLGetPeerCertificateChainLength (SSLContext *ctx, int *chainLen);

The Purpose Of This Function

Examine the number of certificates in the peer certificate chain.

How the Parameters are Used

`chainLen` points to a container into which the number of certificates in the peer certificate chain is copied.

`ctx` points to the context structure defining the SSL connection being examined.

How This Function Affects the Library

SSLGetPeerCertificateChainLength() counts the number of certificates in the peer certificate chain and copies the result to the container pointed to by `chainLength`.

How Errors Are Handled

SSLGetPeerCertificateChainLength() does not check for errors.

SSLErr SSLGetPeerCertificate (SSLContext *ctx, int index, SSLBuffer *derCert);

The Purpose Of This Function

Examine a certificate from the peer certificate chain.

How the Parameters are Used

`derCert` points to an empty buffer structure which will be updated to specify a copy of the requested certificate. The memory pointed to by the updated buffer is allocated using the AllocFunc() callback and should be freed by the user.

`index` specifies which certificate to retrieve by its position in the certificate chain.

`ctx` points to the context structure defining the SSL connection being examined.

How This Function Affects the Library

The requested certificate is found in the peer certificate chain. Memory is allocated for a copy of the certificate using the AllocFunc() callback function and the certificate is copied to the memory. The buffer structure is updated to point to the new certificate.

How Errors Are Handled

SSLOverflowErr can be returned if the requested certificate does not exist.

Any error returned by the AllocFunc() callback function can also be returned.

SSLErr SSLGetNegotiatedCipher (SSLContext *ctx, uint16 *cipherSuite);

The Purpose Of This Function

Examine what Cipher Suite was selected during the handshake process.

How the Parameters are Used

`cipherSuite` points to a container into which the identifier of the negotiated Cipher Suite will be copied. Possible identifiers are specified in Appendix A of the SSL Protocol Version 3.0 Specification document.

`ctx` points to the context structure defining the SSL connection being examined.

How This Function Affects the Library

After a Cipher Suite has been chosen, `cipherSuite` will contain its identifier as specified in cryptype.h.

How Errors Are Handled

SSLGetNegotiatedCipher() does not check for errors.

SSLErr SSLGetWritePendingSize (SSLContext *ctx, uint32 *waitingBytes);

The Purpose Of This Function

Examine how many encrypted bytes are waiting in the write queue to be sent to the peer.

How the Parameters are Used

`waitingBytes` points to a container into which the number of bytes in the write queue will be copied.

`ctx` points to the context structure defining the SSL connection being examined.

How This Function Affects the Library

`waitingBytes` will specify how many bytes have been encrypted but not yet sent to the peer. These bytes are held in the write queue until the queue is serviced through a call to SSLWrite or SSLServiceWriteQueue.

How Errors Are Handled

SSLGetWritePendingSize() does not check for errors.

SSLErr SSLGetReadPendingSize (SSLContext *ctx, uint32 *waitingBytes);

The Purpose Of This Function

Examine how many decrypted bytes of application data are in the read buffer available to be read by the user.

How the Parameters are Used

`waitingBytes` points to a container into which the number of bytes of application data in the read buffer will be copied.

`ctx` points to the context structure defining the SSL connection being examined.

How This Function Affects the Library

`waitingBytes` will specify how many bytes of application data have been decrypted but not yet read by the user (or the library during handshaking). These bytes are held in the read buffer until read through a call to SSLRead.

How Errors Are Handled

SSLGetReadPendingSize() does not check for errors.

Data Exchange

Calls used to establish an SSL connection and exchange data with an SSL peer.

SSLErr SSLHandshake(SSLContext *ctx);

The Purpose Of This Function

Initiate and progress handshake negotiation for an SSL connection.

How the Parameters are Used

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

SSLHandshake initiates the handshake process with the peer. In the absence of any errors, SSLHandshake will not return until the handshake process is complete. Note that the handshake process generally will cause data to be written to the connection as well as read using the I/O callbacks.

How Errors Are Handled

`SSLWouldBlockError` is returned if the read or write callback is unable to fully satisfy the request made by the SSL engine. The user should continue by calling SSLHandshake after taking any desired action during the block.

`SSLProtocolError` is returned if there is an error in the handshake process. `SSLProtocolError` typically indicates the peer is not compliant with the SSL protocol and will not support SSL communications. This error is fatal.

This function can return any of the defined SSL errors including `any` error returned by the allocation and I/O callback functions.

SSLErr SSLRead (void *data, uint32 *length, SSLContext *ctx);

The Purpose Of This Function

Read data from an SSL connection.

How the Parameters are Used

`data` points to the buffer where the read data is will be stored.

`length` points to the size of the buffer in bytes. `*length` will be replaced with the number of bytes actually read.

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

If a secure SSL connection has already been negotiated, SSLRead first attempts to satisfy the read request with any data already decrypted in its internal buffer. If the buffer contains insufficient data, the ReadFunc() callback function is called to retrieve enough data from the physical connection to decrypt a complete record. The decrypted data is moved to the internal buffer and the requested amount is returned to the user.

If a secure SSL connection has not yet been negotiated, SSLRead initiates the handshake process with the peer. In the absence of any errors, SSLRead will not return until the handshake process is complete. Note that the handshake process generally will cause data to be written to the connection as well as read.

How Errors Are Handled

`SSLWouldBlockError` is returned if the read or write callback is unable to fully satisfy the request made by the SSL engine. The user should attempt to read again by calling SSLRead after taking any desired action during the block.

If the block occurred during handshaking and the user calls SSLRead again, it will attempt to continue the handshake process.

`SSLProtocolError` is returned if there is an error in the handshake process. `SSLProtocolError` typically indicates the peer is not compliant with the SSL protocol and will not support SSL communications. This error is fatal.

This function can return any of the defined SSL errors including `SSLConnectionClosedGraceful`, `SSLConnectionClosedError` and any error returned by the allocation and I/O callback functions.

SSLErr SSLWrite (void *data, uint32 *length, SSLContext *ctx);

The Purpose Of This Function

Write data to an SSL connection.

How the Parameters are Used

`data` points to the block of data to be written. The user is responsible for freeing the buffer passed to SSLWrite.

`length` points to the number of bytes to be written. `*length` will be replaced with the number of bytes actually written.

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

If an SSL connection has already been established, SSLWrite causes the specified data to be encrypted, added to the write queue, and the write queue is serviced.

If an SSL connection has not yet been established, SSLWrite initiates the handshake process with the peer. In the absence of any errors, SSLWrite will not return until the handshake process is complete. This process also causes data to be read using the ReadFunc callback.

How Errors Are Handled

`SSLWouldBlockError` is returned if the read or write callback is unable to fully satisfy the request made by the SSL engine. The user should attempt to write again by calling SSLWrite after taking any desired action during the block, and accounting for the portion of data that was successfully written. If the block occurred during handshaking and the user calls SSLWrite again, it will attempt to continue the handshake process.

Typically `SSLWouldBlockError` will return only if the write callback returned `SSLWouldBlockError` while processing the write queue. However, the data buffer passed to SSLWrite will have been processed into the write queue before the block. This means that all data is processed, but SSLWrite must be called again with new data or no data, or SSLProcessWriteQueue must be called to continue processing the write queue.

`SSLProtocolError` is returned if there is an error in the handshake process. `SSLProtocolError` typically indicates the peer is not compliant with the SSL protocol and will not support SSL communications. This error is fatal.

This function can return any of the defined SSL errors including `SSLConnectionClosedGraceful`, `SSLConnectionClosedError` and any error returned by the allocation and I/O callback functions.

SSLErr SSLServiceWriteQueue (SSLConntext *ctx);

The Purpose Of This Function

Attempt to process the internal write queue, writing all available data to the connection. SSLProcessWriteQueue is typically called after SSLWrite has returned SSLWouldBlockErr and the user has no additional data to write.

How the Parameters are Used

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects the Library

SSLServiceWriteQueue attempts to write the contents of the write queue using the WriteFunc() callback function. In the absence of any errors, SSLServiceWriteQueue will not return until the write queue is empty.

Directing SSLWrite to write zero bytes of data will have a similar affect, but SSLWrite performs a number of other tasks in addition to processing the write queue.

How Errors Are Handled

`SSLWouldBlockError` is returned if the WriteFunc() callback is unable to fully satisfy the request made by the SSL engine. The user should continue by calling SSLProcessWriteQueue after taking any desired action during the block.

This function can return any error returned by the allocation and I/O callback functions.

SSLErr SSLClose(SSLContext *ctx);

The Purpose Of This Function

SSLClose attempts to close the SSL connection with the peer. If the connection is not properly closed, it cannot be resumed.

How Parameters are Used

`ctx` points to the context structure defining the SSL connection. The library updates it as necessary for its internal use.

How This Function Affects The Library

A finished message is sent to the peer, indicating that no further communications should be expected.

Note that SSLClose calls the WriteFunc() callback function.

How Errors Are Handled

SSLWouldBlockError is returned if the WriteFunc() callback is unable to process the write request. The user should continue by calling SSLProcessWriteQueue after taking any desired action during the block.

Assigning Application Callbacks

Specify the various functions used by the library to obtain platform-specific services for the specified context.

SSLErr SSLSetAllocFunc (SSLContext *ctx, SSLAllocFunc alloc);

The Purpose Of This Function

Specify the function used by the library to allocate memory for internal use.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`alloc` points to a SSLAllocFunc() function as specified in the section **Platform-specific Callbacks**.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetFreeFunc (SSLContext *ctx, SSLFreeFunc free);

The Purpose Of This Function

Specify the function used by the library to free memory allocated using the function specified by SSLSetAllocFunc().

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`free` points to a SSLFreeFunc() function as specified in the section **Platform-specific Callbacks**.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetReallocFunc (SSLContext *ctx, SSLAllocFunc realloc);

The Purpose Of This Function

Specify the function used by the library to adjust the size of a memory block previously allocated using the function specified by SSLSetAllocFunc().

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`realloc` points to a SSLReallocFunc() function as specified in the section **Platform-specific Callbacks**.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetAllocRef(SSLContext *ctx, void *allocRef);

The Purpose Of This Function

Specify the reference parameter passed by the library to the memory allocation callback functions.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`allocRef` is passed through to the memory allocation callback functions. It is shared by SSLAllocFunc(), SSLFreeFunc(), and SSLReallocFunc().

`allocRef` is not used by the library. It is up to the callback functions to determine how `allocRef` is used.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetTimeFunc (SSLContext *ctx, SSLTimeFunc time);

The Purpose Of This Function

Specify the function used by the library to determine the current time.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`time` points to a SSLTimeFunc() function as specified in the section **Platform-specific Callbacks**.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetConvertTimeFunc (SSLContext *ctx, SSLTimeFunc convertTime);

The Purpose Of This Function

Specify the function used by the library to convert time.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`converTime` points to a SSLConvertTimeFunc() function as specified in the section **Platform-specific Callbacks**.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetTimeRef(SSLContext *ctx, void *timeRef);

The Purpose Of This Function

Specify the reference parameter passed by the library to the time callback functions.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`timeRef` is passed through to the time callback functions. It is shared by SSLTimeFunc() and SSLConvertTimeFunc().

`timeRef` is not used by the library. It is up to the callback functions to determine how `timeRef` is used.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetRandomFunc (SSLContext *ctx, SSLRandomFunc random);

The Purpose Of This Function

Specify the function used by the library to obtain a random number.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`random` points to a SSLRandomFunc() function as specified in the section **Platform-specific Callbacks**.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetRandomRef (SSLContext *ctx, void *randomRef);

The Purpose Of This Function

Specify the reference parameter passed by the library to the random callback function.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`randomRef` is passed through to the random callback function.

`randomRef` is not used by the library. It is up to the callback function to determine how `randomRef` is used.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetReadFunc (SSLContext *ctx, SSLIOFunc read);

The Purpose Of This Function

Specify the function used by the library to retrieve data from the physical connection.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`read` points to a SSLIOFunc() function as specified in the section **I/O Callbacks**.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetWriteFunc (SSLContext *ctx, SSLIOFunc write);

The Purpose Of This Function

Specify the function used by the library to write data to the physical connection.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`write` points to a SSLIOFunc() function as specified in the section **I/O Callbacks**.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetIORef(SSLContext *ctx, void *ioRef);

The Purpose Of This Function

Specify the reference parameter passed by the library to the I/O callback functions.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`ioRef` is passed through to the I/O callback functions. It is shared by SSLReadFunc() and SSLWriteFunc().

`ioRef` is not used by the library. It is up to the callback functions to determine how `ioRef` is used.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetAddSessionFunc (SSLContext *ctx, SSLAddSessionFunc addSession);

The Purpose Of This Function

Specify the function used by the library to add a session's resume information to the database of resumable sessions.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`addSession` points to a SSLAddSessionFunc() function as specified in the section **Session Resumption Callbacks**.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetGetSessionFunc (SSLContext *ctx, SSLGetSessionFunc getSession);

The Purpose Of This Function

Specify the function used by the library to retrieve a session's resume information from the database of resumable sessions.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`getSession` points to a SSLGetSessionFunc() function as specified in the section **Session Resumption Callbacks**.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetDeleteSessionFunc (SSLContext *ctx, SSLDeleteSessionFunc deleteSession);

The Purpose Of This Function

Specify the function used by the library to remove a session's resume information from the database of resumable sessions.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`deleteSession` points to a SSLDeleteSessionFunc() function as specified in the section **Session Resumption Callbacks**.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetSessionRef (SSLContext *ctx, void *sessionRef);

The Purpose Of This Function

Specify the reference parameter passed by the library to the Session Resumption callback functions.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`sessionRef` is passed through to the Session Resumption callback functions. It is shared by SSLAddSessionFunc(), SSLGetSessionFunc(), and SSLDeleteSessionFunc().

`sessionRef` is not used by the library. It is up to the callback functions to determine how `sessionRef` is used.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetCheckCertificateFunc (SSLContext *ctx, SSLCheckCertificateFunc checkCertificate);

The Purpose Of This Function

Specify the function used by the library to check the validity of a certificate.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`checkCertificate` points to a SSLCheckCertificateFunc() function as specified in the section **Certificate Verification Callbacks**.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

SSLErr SSLSetCheckCertificateRef (SSLContext *ctx, void *checkCertificateRef);

The Purpose Of This Function

Specify the reference parameter passed by the library to the Certificate Verification callback functions.

How the Parameters are Used

`ctx` points to the context structure into which this parameter is installed.

`checkCertificateRef` is passed through to the Certificate Verification callback functions. It is used by SSLCheckCertificateFunc().

`checkCertificateRef` is not used by the library. It is up to the callback functions to determine how `checkCertificateRef` is used.

How This Function Affects the Library

This function configures the SSLContext structure for later use.

How Errors Are Handled

This functions does not check for errors.

Platform-specific Callbacks

These functions must be supplied by the user to provide memory allocation, random numbers, and time access in a platform-independent manner.

If the library is used in a threaded manner, all callbacks provided by the application must be thread-safe.

typedef SSLErr (*SSLAllocFunc) (SSLBuffer *buf, void *allocRef);

The Purpose Of This Function

Allocate memory to be used internally by the library.

How the Parameters are Used

`buf` points to a SSLBuffer structure identifying the memory to be allocated.

`buf->length` specifies how many bytes to allocate.

`buf->data` must contain a pointer to the allocated space when the function returns. It must be NULL if the allocation fails.

`allocRef` is the value specified by SetAllocRef() for this SSLContext. It is up to the callback function to determine how to use allocRef.

The return value must be zero if the allocation succeeds. All other values are considered an error.

How This Function Affects the Library

The library calls *SSLAllocFunc() whenever it requires more memory, such as for read and write buffers.

Any `buf` structure returned from *SSLAllocFunc() must be a valid parameter to *SSLFreeFunc(), even if the allocation failed.

How Errors Are Handled

The user determines what errors this function can generate.

The user is encouraged to return one of the defined SSL errors where appropriate.

All errors generated by this function are passed back to the user through the calling functions. The library will not act upon any of these errors.

typedef SSLErr (*SSLFreeFunc) (SSLBuffer *buf, void *allocRef);

The Purpose Of This Function

Free memory used internally by the library and allocated with *SSLAllocFunc().

How the Parameters are Used

`buf` points to a SSLBuffer structure identifying the memory to be freed.

`buf->length` specifies how many bytes this buffer uses. It should not be used by this function.

`buf->data` contains a pointer to the memory to be freed. It may be NULL if the allocation had failed.

`allocRef` is the value specified by SetAllocRef() for this SSLContext. It is up to the callback function to determine how to use `allocRef`.

The return value must be zero if the free succeeds. All other values are considered an error.

How This Function Affects the Library

The library calls *SSLFreeFunc() whenever it is done with the memory it had received from a previous call to *SSLAllocFunc().

*SSLFreeFunc() must be able to handle any `buf` structure returned from *SSLAllocFunc(), including one where the allocation failed (which should have `buf->data` set to NULL).

How Errors Are Handled

The user determines what errors this function can generate.

The user is encouraged to return one of the defined SSL errors where appropriate.

All errors generated by this function are passed back to the user through the calling functions. The library will not act upon any of these errors.

typedef SSLErr (*SSLReallocFunc) (SSLBuffer *buf, uint32 newSize, void *allocRef);

Attempt to reallocate the buffer, copying all data possible.

The Purpose Of This Function

Attempt to resize a block of memory used internally by the library and allocated with *SSLAllocFunc().

How the Parameters are Used

`buf` points to a SSLBuffer structure identifying the memory to be resized.

`buf->length` specifies how many bytes this buffer uses. It should be set to the new size when this function returns.

`buf->data` contains a pointer to the memory to be resized. It should be set to point to the new block when this function returns. All data from the original buffer must be copied to the new block if a new block is allocated.

`buf->data` should be set to NULL if the realloc fails and destroys the original buffer. Memory previously pointed to by `buf->data` must be freed.

`newSize` specifies the desired size of the new buffer.

`allocRef` is the value specified by SetAllocRef() for this SSLContext. It is up to the callback function to determine how to use `allocRef`.

The return value must be zero if the realloc succeeds. All other values are considered an error.

How This Function Affects the Library

The library can call *SSLReallocFunc() to resize a block of memory it had received from a previous call to *SSLAllocFunc().

*SSLReallocFunc() must be able to handle any `buf` structure returned from *SSLAllocFunc().

How Errors Are Handled

The user determines what errors this function can generate.

The user is encouraged to return one of the defined SSL errors where appropriate.

All errors generated by this function are passed back to the user through the calling functions. The library will not act upon any of these errors.

typedef SSLErr (*SSLRandomFunc) (SSLBuffer buf, void *randomRef);

The Purpose Of This Function

Provide a buffer of cryptographically secure random bytes.

How the Parameters are Used

`buf` is a SSLBuffer structure identifying where to copy the random data.

`buf.length` specifies how many bytes of random data is requested.

`buf.data` contains a pointer to the memory where the bytes are to be copied.

`randomRef` is the value specified by SetRandomRef() for this SSLContext. It is up to the callback function to determine how to use `randomRef` .

The return value must be zero if the function succeeds. All other values are considered an error.

How This Function Affects the Library

The random data provided by this function is used for a variety of cryptographic purposes for the specified SSL connection. It is up to the user to ensure that this data is cryptographically secure for the specified SSLContext.

Seeding this generator is the responsibility of the user.

How Errors Are Handled

The user determines what errors this function can generate.

The user is encouraged to return one of the defined SSL errors where appropriate.

All errors generated by this function are passed back to the user through the calling functions. The library will not act upon any of these errors.

typedef SSLErr (*SSLTimeFunc) (uint32 *time, void *timeRef);

The Purpose Of This Function

Provide the current time in seconds GMT since Midnight, Jan 1, 1970, GMT.

How the Parameters are Used

`time` points to a 32 bit unsigned integer into which the user must place the current time in seconds GMT since Midnight, Jan 1, 1970, GMT. This value must be GMT, not a local reference.

`timeRef` is the value specified by SetTimeRef() for this SSLContext. It is up to the callback function to determine how to use `timeRef` .

The return value must be zero if the function succeeds. All other values are considered an error.

How This Function Affects the Library

The time data provided by this function is used for a variety of purposes for the specified SSL connection.

How Errors Are Handled

The user determines what errors this function can generate.

The user is encouraged to return one of the defined SSL errors where appropriate.

All errors generated by this function are passed back to the user through the calling functions. The library will not act upon any of these errors.

typedef SSLErr (*SSLConvertTimeFunc) (uint32 *time, void *timeRef);

The Purpose Of This Function

Convert the input time from the format returned by the user's ANSI C function mktime() to seconds GMT since Midnight, Jan 1, 1970, GMT.

How the Parameters are Used

`time` points to a 32 bit unsigned integer containing a time value in the format provided by the user's ANSI C function mktime().(mktime() is an ANSI C library function linked to the SSLRef library when it is built.)

Upon return, `time` must point to the input value converted to seconds GMT since Midnight, Jan 1, 1970, GMT. This value must be GMT, not a local reference.

`timeRef` is the value specified by SetTimeRef() for this SSLContext. It is up to the callback function to determine how to use `timeRef`.

The return value must be zero if the function succeeds. All other values are considered an error.

How This Function Affects the Library

The time data provided by this function is used for a variety of purposes for the specified SSL connection.

How Errors Are Handled

The user determines what errors this function can generate.

The user is encouraged to return one of the defined SSL errors where appropriate.

All errors generated by this function are passed back to the user through the calling functions. The library will not act upon any of these errors.

I/O Callbacks

These functions are used to give the SSLRef library access to data transport over a particular physical connection. The user is responsible for opening and closing the physical connection; hence, no such functions are provided. This allows clients to begin an SSL connection after some unencrypted communication for protocols which require it, or vary the client/server mode of the communication.

If the library is used in a threaded manner, all callbacks provided by the application must be thread-safe.

typedef SSLErr (*SSLReadFunc) (SSLBuffer data, uint32 *processed, void *connRef);

The Purpose Of This Function

Retrieve data from the specified physical connection.

How the Parameters are Used

`data` is an SSLBuffer into which the retrieved data should be copied.

`data.length` specifies how much data to read.

Upon return, `processed` must point to the number of bytes that were read.

`ioRef` is the value specified by SetIORef() for this SSLContext. It must provide enough information to distinguish this physical connection from others. It is up to the callback function to determine exactly how to use `ioRef`. Typically it will contain a pointer to data describing the physical connection.

The return value must be zero if the function succeeds. All other values are considered an error.

How This Function Affects the Library

The data read is decrypted by the library and, if application data, passed to the user through SSLRead(). Otherwise the data is processed by the library as appropriate.

How Errors Are Handled

The user determines what errors this function can generate.

The user is encouraged to return one of the defined SSL errors where appropriate.

All errors generated by this function are passed back to the user through the calling functions. The library will not act upon any of these errors.

This function may at the user's discretion return SSLWouldBlockErr if fulfilling the read request would cause the library to block further processing.

typedef SSLErr (*SSLWriteFunc) (SSLBuffer data, uint32 *processed, void *connRef);

The Purpose Of This Function

Write data to the specified physical connection.

How the Parameters are Used

`data` is an SSLBuffer from which the data should be copied.

`data.length` specifies how much data to write.

Upon return, `processed` must point to the number of bytes that were written.

`ioRef` is the value specified by SetIORef() for this SSLContext. It must provide enough information to distinguish this physical connection from others. It is up to the callback function to determine exactly how to use `ioRef`. Typically it will contain a pointer to the physical connection.

The return value must be zero if the function succeeds. All other values are considered an error.

How This Function Affects the Library

The library prepares data from SSLWrite(), or SSL protocol messages, and passes the data to this function to be sent through the physical connection.

How Errors Are Handled

The user determines what errors this function can generate.

The user is encouraged to return one of the defined SSL errors where appropriate.

All errors generated by this function are passed back to the user through the calling functions. The library will not act upon any of these errors.

This function may at the user's discretion return SSLWouldBlockErr if fulfilling the write request would cause the library to block further processing.

Session Resumption Callbacks

These functions are used to allow SSLRef to maintain a database of resumable SSL sessions as identified by the value set by the SSLSetPeerID() function. The user is responsible for aging and deleting obsolete sessions from this database.

The stored information is persistent across processes. It consists of

- session ID (used as a key for retrieval)
- protocol version of the session
- cipher spec used for the session
- master secret for the session
- certificates of the peer for the session

Note that this is an opaque data structure. Users should not rely on this information being stored in this way in future versions.

If the library is used in a threaded manner, all callbacks provided by the application must be thread-safe.

typedef SSLErr (*SSLAddSessionFunc) (SSLBuffer sessionKey, SSLBuffer sessionData, void *sessionRef);

The Purpose Of This Function

Add the session information to the database.

How the Parameters are Used

`sessionKey` is an SSLBuffer identifying this peer. The value is originally specified by the user through the SSLSetPeerID() function. The library does not access this data in any way.

`sessionData` is an SSLBuffer identifying information needed by the library to resume this session. The library retains ownership of this buffer. The user must copy the data before returning.

`sessionData.length` specifies the size of the data to be stored.

`sessionData.data` points to the data to be stored.

`sessionRef` is the value specified by SetSessionRef() for this SSLContext. It is up to the callback function to determine exactly how to use `sessionRef`. Typically it will contain a pointer to the database structures.

The return value must be zero if the function succeeds. All other values are considered an error.

How This Function Affects the Library

When the library establishes a session with a peer, it requests this function store enough information about the session that the session can be later resumed. This stored data is opaque to the user, but must be retrievable by the session key.

If the database already contains an entry for this session key that entry should be replaced with this new one.

The user is responsible for aging and deleting obsolete sessions from its database.

How Errors Are Handled

The user determines what errors this function can generate.

All errors generated by this function are ignored.

typedef SSLErr (*SSLGetSessionFunc) (SSLBuffer sessionKey, SSLBuffer *sessionData, void *sessionRef);

The Purpose Of This Function

Retrieve the session information specified by the sessionKey from the database.

How the Parameters are Used

`sessionKey` is an SSLBuffer identifying this peer. The value is originally specified by the user through the SSLSetPeerID() function. The library does not access this data in any way.

`sessionData` must be filled with a pointer to an SSLBuffer allocated by this function and into which the session resumption data is copied. This buffer must be allocated such that it can be freed with the FreeFunc() callback. Note that ownership of the allocated memory is transferred from this function to the library. The library becomes responsible for freeing that memory.

`sessionRef` is the value specified by SetSessionRef() for this SSLContext. It is up to the callback function to determine exactly how to use `sessionRef`. Typically it will contain a pointer to the physical connection.

The return value must be zero if the function succeeds. All other values are considered an error.

How This Function Affects the Library

Before the library establishes a session with the peer, it requests this function return any stored information about previous sessions with this peer. This information, which typically contains authentication and cryptographic data, will be used to resume the session. This stored data is opaque to the user, but must be retrievable by the session key.

If the database does not contain an entry for this session key, SSLNoMatchingSessionKeyErr should be returned.

How Errors Are Handled

Return SSLNoMatchingSessionKeyErr if no sessionKey match is found.

The user determines what other errors this function can generate.

The user is encouraged to return one of the defined SSL errors where appropriate.

All errors generated by this function cause the session to not be resumed. Otherwise, errors are ignored.

typedef SSLErr (*SSLDeleteSessionFunc) (SSLBuffer sessionKey, void *connRef);

The Purpose Of This Function

Delete the session information specified by the sessionKey from the database.

How the Parameters are Used

`sessionKey` is an SSLBuffer identifying this peer. The value is originally specified by the user through the SSLSetPeerID() function. The library does not access this data in any way.

`sessionRef` is the value specified by SetSessionRef() for this SSLContext. It is up to the callback function to determine exactly how to use `sessionRef`. Typically it will contain a pointer to the physical connection.

The return value must be zero if the function succeeds. All other values are considered an error.

How This Function Affects the Library

If the library detects that the security of a session might be compromised, such as by proper close messages not being received, the library will request that session resumption data associated with that peer be removed from the database so that the session can not be resumed.

How Errors Are Handled

Return SSLNoMatchingSessionKeyErr if no sessionKey match is found.

The user determines what other errors this function can generate.

The user is encouraged to return one of the defined SSL errors where appropriate.

All errors generated by this function are ignored.

Certificate Verification Callbacks

These functions are used to allow SSLRef to allow the user to perform additional verification of certificates. Before a certificate is added to a certificate chain, SSLRef verifies it against other certificates in the chain. However, SSLRef does not perform any other verification of certificate.

If the library is used in a threaded manner, all callbacks provided by the application must be thread-safe.

typedef SSLErr (*SSLCheckCertificateFunc) (int certCount, SSLBuffer *derCerts, void *checkCertificateRef);

The Purpose Of This Function

Perform any additional certificate verification procedures desired by the user. This callback function is typically used to enable whatever model of trust is appropriate for the user.

How the Parameters are Used

`certCount` is the number of certificates to be processed.

`derCerts` points to an array of buffer structures specifying the certificates to be processed. The library retains ownership of this array and the certificates.

`checkCertificateRef` is the value specified by SSLSetCheckCertificateRef() for this SSLContext. It is up to the callback function to determine exactly how to use `checkCertificateRef`.

The return value must be zero if the function succeeds. All other values are considered an error.

How This Function Affects the Library

When the library is building a chain of certificates, each certificate is verified with the other certificates in the chain. No other verification or authenticity checks are performed. Instead, this callback function is called, allowing the user to perform any additional checks desired.

How Errors Are Handled

The user determines what errors this function can generate.

All errors generated by this function are passed to the calling function.