NAME

TLSv1_2_method, TLSv1_2_server_method, TLSv1_2_client_method, SSL_CTX_new, SSL_CTX_new_ex, SSL_CTX_up_ref, SSLv3_method, SSLv3_server_method, SSLv3_client_method, TLSv1_method, TLSv1_server_method, TLSv1_client_method, TLSv1_1_method, TLSv1_1_server_method, TLSv1_1_client_method, TLS_method, TLS_server_method, TLS_client_method, SSLv23_method, SSLv23_server_method, SSLv23_client_method, DTLS_method, DTLS_server_method, DTLS_client_method, DTLSv1_method, DTLSv1_server_method, DTLSv1_client_method, DTLSv1_2_method, DTLSv1_2_server_method, DTLSv1_2_client_method - create a new SSL_CTX object as framework for TLS/SSL or DTLS enabled functions

SYNOPSIS

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

SSL_CTX *SSL_CTX_new_ex(OSSL_LIB_CTX *libctx, const char *propq,
                        const SSL_METHOD *method);
SSL_CTX *SSL_CTX_new(const SSL_METHOD *method);
int SSL_CTX_up_ref(SSL_CTX *ctx);

const SSL_METHOD *TLS_method(void);
const SSL_METHOD *TLS_server_method(void);
const SSL_METHOD *TLS_client_method(void);

const SSL_METHOD *SSLv23_method(void);
const SSL_METHOD *SSLv23_server_method(void);
const SSL_METHOD *SSLv23_client_method(void);

#ifndef OPENSSL_NO_SSL3_METHOD
const SSL_METHOD *SSLv3_method(void);
const SSL_METHOD *SSLv3_server_method(void);
const SSL_METHOD *SSLv3_client_method(void);
#endif

#ifndef OPENSSL_NO_TLS1_METHOD
const SSL_METHOD *TLSv1_method(void);
const SSL_METHOD *TLSv1_server_method(void);
const SSL_METHOD *TLSv1_client_method(void);
#endif

#ifndef OPENSSL_NO_TLS1_1_METHOD
```

```
const SSL_METHOD *TLSv1_1_method(void);
const SSL_METHOD *TLSv1_1_server_method(void);
const SSL_METHOD *TLSv1_1_client_method(void);
#endif

#ifndef OPENSSL_NO_TLS1_2_METHOD
const SSL_METHOD *TLSv1_2_method(void);
const SSL_METHOD *TLSv1_2_server_method(void);
const SSL_METHOD *TLSv1_2_client_method(void);
#endif

const SSL_METHOD *DTLS_method(void);
const SSL_METHOD *DTLS_server_method(void);
const SSL_METHOD *DTLS_client_method(void);

#ifndef OPENSSL_NO_DTLS1_METHOD
const SSL_METHOD *DTLSv1_method(void);
const SSL_METHOD *DTLSv1_server_method(void);
const SSL_METHOD *DTLSv1_client_method(void);
#endif

#ifndef OPENSSL_NO_DTLS1_2_METHOD
const SSL_METHOD *DTLSv1_2_method(void);
const SSL_METHOD *DTLSv1_2_server_method(void);
const SSL_METHOD *DTLSv1_2_client_method(void);
#endif
```

DESCRIPTION

SSL_CTX_new_ex() creates a new **SSL_CTX** object, which holds various configuration and data relevant to SSL/TLS or DTLS session establishment. These are later inherited by the **SSL** object representing an active session. The *method* parameter specifies whether the context will be used for the client or server side or both - for details see the "NOTES" below. The library context *libctx* (see **OSSL_LIB_CTX**(3)) is used to provide the cryptographic algorithms needed for the session. Any cryptographic algorithms that are used by any **SSL** objects created from this **SSL_CTX** will be fetched from the *libctx* using the property query string *propq* (see "ALGORITHM FETCHING" in **crypto**(7). Either or both the *libctx* or *propq* parameters may be NULL.

SSL_CTX_new() does the same as **SSL_CTX_new_ex()** except that the default library context is used and no property query string is specified.

An **SSL_CTX** object is reference counted. Creating an **SSL_CTX** object for the first time increments the reference count. Freeing the **SSL_CTX** (using SSL_CTX_free) decrements it. When the reference count drops to zero, any memory or resources allocated to the **SSL_CTX** object are freed. **SSL_CTX_up_ref()** increments the reference count for an existing **SSL_CTX** structure.

An **SSL_CTX** object should not be changed after it is used to create any **SSL** objects or from multiple threads concurrently, since the implementation does not provide serialization of access for these cases.

NOTES

On session establishment, by default, no peer credentials verification is done. This must be explicitly requested, typically using **SSL_CTX_set_verify**(3). For verifying peer certificates many options can be set using various functions such as **SSL_CTX_load_verify_locations**(3) and **SSL_CTX_set1_param**(3). The **X509_VERIFY_PARAM_set_purpose**(3) function can be used, also in conjunction with **SSL_CTX_get0_param**(3), to set the intended purpose of the session. The default is **X509_PURPOSE_SSL_SERVER** on the client side and **X509_PURPOSE_SSL_CLIENT** on the server side.

The SSL_CTX object uses *method* as the connection method. Three method variants are available: a generic method (for either client or server use), a server-only method, and a client-only method.

The *method* parameter of **SSL_CTX_new_ex()** and **SSL_CTX_new()** can be one of the following:

TLS_method(), **TLS_server_method()**, **TLS_client_method()**

These are the general-purpose *version-flexible* SSL/TLS methods. The actual protocol version used will be negotiated to the highest version mutually supported by the client and the server. The supported protocols are SSLv3, TLSv1, TLSv1.1, TLSv1.2 and TLSv1.3. Applications should use these methods, and avoid the version-specific methods described below, which are deprecated.

SSLv23_method(), **SSLv23_server_method()**, **SSLv23_client_method()**

These functions do not exist anymore, they have been renamed to **TLS_method()**, **TLS_server_method()** and **TLS_client_method()** respectively. Currently, the old function calls are renamed to the corresponding new ones by preprocessor macros, to ensure that existing code which uses the old function names still compiles. However, using the old function names is deprecated and new code should call the new functions instead.

TLSv1_2_method(), **TLSv1_2_server_method()**, **TLSv1_2_client_method()**

A TLS/SSL connection established with these methods will only understand the TLSv1.2 protocol. These methods are deprecated.

TLSv1_1_method(), **TLSv1_1_server_method()**, **TLSv1_1_client_method()**

A TLS/SSL connection established with these methods will only understand the TLSv1.1 protocol. These methods are deprecated.

TLSv1_method(), **TLSv1_server_method()**, **TLSv1_client_method()**
A TLS/SSL connection established with these methods will only understand the TLSv1 protocol. These methods are deprecated.

SSLv3_method(), **SSLv3_server_method()**, **SSLv3_client_method()**
A TLS/SSL connection established with these methods will only understand the SSLv3 protocol. The SSLv3 protocol is deprecated and should not be used.

DTLS_method(), **DTLS_server_method()**, **DTLS_client_method()**
These are the version-flexible DTLS methods. Currently supported protocols are DTLS 1.0 and DTLS 1.2.

DTLSv1_2_method(), **DTLSv1_2_server_method()**, **DTLSv1_2_client_method()**
These are the version-specific methods for DTLSv1.2. These methods are deprecated.

DTLSv1_method(), **DTLSv1_server_method()**, **DTLSv1_client_method()**
These are the version-specific methods for DTLSv1. These methods are deprecated.

SSL_CTX_new() initializes the list of ciphers, the session cache setting, the callbacks, the keys and certificates and the options to their default values.

TLS_method(), **TLS_server_method()**, **TLS_client_method()**, **DTLS_method()**, **DTLS_server_method()** and **DTLS_client_method()** are the *version-flexible* methods. All other methods only support one specific protocol version. Use the *version-flexible* methods instead of the version specific methods.

If you want to limit the supported protocols for the version flexible methods you can use **SSL_CTX_set_min_proto_version**(3), **SSL_set_min_proto_version**(3), **SSL_CTX_set_max_proto_version**(3) and **SSL_set_max_proto_version**(3) functions. Using these functions it is possible to choose e.g. **TLS_server_method()** and be able to negotiate with all possible clients, but to only allow newer protocols like TLS 1.0, TLS 1.1, TLS 1.2 or TLS 1.3.

The list of protocols available can also be limited using the **SSL_OP_NO_SSLv3**, **SSL_OP_NO_TLSv1**, **SSL_OP_NO_TLSv1_1**, **SSL_OP_NO_TLSv1_3**, **SSL_OP_NO_TLSv1_2** and **SSL_OP_NO_TLSv1_3** options of the **SSL_CTX_set_options**(3) or **SSL_set_options**(3) functions, but this approach is not recommended. Clients should avoid creating "holes" in the set of protocols they support. When disabling a protocol, make sure that you also disable either all previous or all

subsequent protocol versions. In clients, when a protocol version is disabled without disabling *all* previous protocol versions, the effect is to also disable all subsequent protocol versions.

The SSLv3 protocol is deprecated and should generally not be used. Applications should typically use **SSL_CTX_set_min_proto_version**(3) to set the minimum protocol to at least **TLS1_VERSION**.

RETURN VALUES

The following return values can occur:

NULL
: The creation of a new SSL_CTX object failed. Check the error stack to find out the reason.

Pointer to an SSL_CTX object
: The return value points to an allocated SSL_CTX object.

SSL_CTX_up_ref() returns 1 for success and 0 for failure.

SEE ALSO

SSL_CTX_set_options(3), **SSL_CTX_free**(3), **SSL_CTX_set_verify**(3), **SSL_CTX_set1_param**(3), **SSL_CTX_get0_param**(3), **SSL_connect**(3), **SSL_accept**(3), **SSL_CTX_set_min_proto_version**(3), **ssl**(7), **SSL_set_connect_state**(3)

HISTORY

Support for SSLv2 and the corresponding **SSLv2_method()**, **SSLv2_server_method()** and **SSLv2_client_method()** functions where removed in OpenSSL 1.1.0.

SSLv23_method(), **SSLv23_server_method()** and **SSLv23_client_method()** were deprecated and the preferred **TLS_method()**, **TLS_server_method()** and **TLS_client_method()** functions were added in OpenSSL 1.1.0.

All version-specific methods were deprecated in OpenSSL 1.1.0.

SSL_CTX_new_ex() was added in OpenSSL 3.0.

COPYRIGHT