NAME

SSL_get_error - obtain result code for TLS/SSL I/O operation

SYNOPSIS

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

 int SSL_get_error(const SSL *ssl, int ret);
```

DESCRIPTION

SSL_get_error() returns a result code (suitable for the C ‘‘switch’’ statement) for a preceding call to *SSL_connect()*, *SSL_accept()*, *SSL_do_handshake()*, *SSL_read()*, *SSL_peek()*, or *SSL_write()* on **ssl**. The value returned by that TLS/SSL I/O function must be passed to *SSL_get_error()* in parameter **ret**.

In addition to **ssl** and **ret**, *SSL_get_error()* inspects the current thread’s OpenSSL error queue. Thus, *SSL_get_error()* must be used in the same thread that performed the TLS/SSL I/O operation, and no other OpenSSL function calls should appear in between. The current thread’s error queue must be empty before the TLS/SSL I/O operation is attempted, or *SSL_get_error()* will not work reliably.

RETURN VALUES

The following return values can currently occur:

SSL_ERROR_NONE

The TLS/SSL I/O operation completed. This result code is returned if and only if **ret > 0**.

SSL_ERROR_ZERO_RETURN

The TLS/SSL connection has been closed. If the protocol version is SSL 3.0 or higher, this result code is returned only if a closure alert has occurred in the protocol, i.e. if the connection has been closed cleanly. Note that in this case **SSL_ERROR_ZERO_RETURN** does not necessarily indicate that the underlying transport has been closed.

SSL_ERROR_WANT_READ, SSL_ERROR_WANT_WRITE

The operation did not complete; the same TLS/SSL I/O function should be called again later. If, by then, the underlying **BIO** has data available for reading (if the result code is **SSL_ERROR_WANT_READ**) or allows writing data (**SSL_ERROR_WANT_WRITE**), then some TLS/SSL protocol progress will take place, i.e. at least part of an TLS/SSL record will be read or written. Note that the retry may again lead to a **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE** condition. There is no fixed upper limit for the number of iterations that may be necessary until progress becomes visible at application protocol level.

For socket **BIO**s (e.g. when *SSL_set_fd()* was used), *select()* or *poll()* on the underlying socket can be used to find out when the TLS/SSL I/O function should be retried.

Caveat: Any TLS/SSL I/O function can lead to either of **SSL_ERROR_WANT_READ** and **SSL_ERROR_WANT_WRITE**. In particular, *SSL_read()* or *SSL_peek()* may want to write data and *SSL_write()* may want to read data. This is mainly because TLS/SSL handshakes may occur at any time during the protocol (initiated by either the client or the server); *SSL_read()*, *SSL_peek()*, and *SSL_write()* will handle any pending handshakes.

SSL_ERROR_WANT_CONNECT, SSL_ERROR_WANT_ACCEPT

The operation did not complete; the same TLS/SSL I/O function should be called again later. The underlying BIO was not connected yet to the peer and the call would block in *connect()*/*accept()*. The SSL function should be called again when the connection is established. These messages can only appear with a *BIO_s_connect()* or *BIO_s_accept()* BIO, respectively. In order to find out, when the connection has been successfully established, on many platforms *select()* or *poll()* for writing on the socket file descriptor can be used.

SSL_ERROR_WANT_X509_LOOKUP

The operation did not complete because an application callback set by *SSL_CTX_set_client_cert_cb()* has asked to be called again. The TLS/SSL I/O function should be called again later. Details depend on the application.

SSL_ERROR_WANT_ASYNC
: The operation did not complete because an asynchronous engine is still processing data. This will only occur if the mode has been set to SSL_MODE_ASYNC using *SSL_CTX_set_mode(3)* or *SSL_set_mode(3)* and an asynchronous capable engine is being used. An application can determine whether the engine has completed its processing using *select()* or *poll()* on the asynchronous wait file descriptor. This file descriptor is available by calling *SSL_get_all_async_fds(3)* or *SSL_get_changed_async_fds(3)*. The TLS/SSL I/O function should be called again later. The function **must** be called from the same thread that the original call was made from.

SSL_ERROR_WANT_ASYNC_JOB
: The asynchronous job could not be started because there were no async jobs available in the pool (see *ASYNC_init_thread(3)*). This will only occur if the mode has been set to SSL_MODE_ASYNC using *SSL_CTX_set_mode(3)* or *SSL_set_mode(3)* and a maximum limit has been set on the async job pool through a call to *ASYNC_init_thread(3)*. The application should retry the operation after a currently executing asynchronous operation for the current thread has completed.

SSL_ERROR_SYSCALL
: Some non-recoverable, fatal I/O error occurred. The OpenSSL error queue may contain more information on the error. For socket I/O on Unix systems, consult **errno** for details. If this error occurs then no further I/O operations should be performed on the connection and *SSL_shutdown()* must not be called.

SSL_ERROR_SSL
: A non-recoverable, fatal error in the SSL library occurred, usually a protocol error. The OpenSSL error queue contains more information on the error. If this error occurs then no further I/O operations should be performed on the connection and *SSL_shutdown()* must not be called.

SEE ALSO

ssl(3), *err(3)*

HISTORY

SSL_ERROR_WANT_ASYNC was added in OpenSSL 1.1.0.

COPYRIGHT