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_ex()**, **SSL_read()**, **SSL_peek_ex()**, **SSL_peek()**, **SSL_shutdown()**, **SSL_write_ex()** 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.

NOTES

Some TLS implementations do not send a close_notify alert on shutdown.

On an unexpected EOF, versions before OpenSSL 3.0 returned **SSL_ERROR_SYSCALL**, nothing was added to the error stack, and errno was 0. Since OpenSSL 3.0 the returned error is **SSL_ERROR_SSL** with a meaningful error on the error stack.

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 peer has closed the connection for writing by sending the close_notify alert. No more data can be read. Note that **SSL_ERROR_ZERO_RETURN** does not necessarily indicate that the underlying transport has been closed.

This error can also appear when the option **SSL_OP_IGNORE_UNEXPECTED_EOF** is set. See **SSL_CTX_set_options**(3) for more details.

SSL_ERROR_WANT_READ, SSL_ERROR_WANT_WRITE

The operation did not complete and can be retried later.

SSL_ERROR_WANT_READ is returned when the last operation was a read operation from a nonblocking **BIO**. It means that not enough data was available at this time to complete the operation. If at a later time the underlying **BIO** has data available for reading the same function can be called again.

SSL_read() and **SSL_read_ex()** can also set **SSL_ERROR_WANT_READ** when there is still unprocessed data available at either the **SSL** or the **BIO** layer, even for a blocking **BIO**. See **SSL_read**(3) for more information.

SSL_ERROR_WANT_WRITE is returned when the last operation was a write to a nonblocking **BIO** and it was unable to sent all data to the **BIO**. When the **BIO** is writable again, the same function can be called again.

Note that the retry may again lead to an **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.

It is safe to call **SSL_read()** or **SSL_read_ex()** when more data is available even when the call that set this error was an **SSL_write()** or **SSL_write_ex()**. However, if the call was an **SSL_write()** or **SSL_write_ex()**, it should be called again to continue sending the application data. If you get **SSL_ERROR_WANT_WRITE** from **SSL_write()** or **SSL_write_ex()** then you should not do any other operation that could trigger **IO** other than to repeat the previous **SSL_write()** call.

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_ex()**, **SSL_read()**, **SSL_peek_ex()**, or **SSL_peek()** may want to write data and **SSL_write()** or **SSL_write_ex()** 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_ex()**, **SSL_read()**, **SSL_peek_ex()**, **SSL_peek()**, **SSL_write_ex()**, 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_WANT_CLIENT_HELLO_CB
: The operation did not complete because an application callback set by **SSL_CTX_set_client_hello_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_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.

  This value can also be returned for other errors, check the error queue for details.

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(7)

HISTORY

The SSL_ERROR_WANT_ASYNC error code was added in OpenSSL 1.1.0. The SSL_ERROR_WANT_CLIENT_HELLO_CB error code was added in OpenSSL 1.1.1.

COPYRIGHT