



*Full credit is given to the above companies including the OS that this PDF file was generated!*

### ***Rocky Enterprise Linux 9.2 Manual Pages on command 'SSL\_CTX\_add\_client\_custom\_ext.3ossl'***

***\$ man SSL\_CTX\_add\_client\_custom\_ext.3ossl***

SSL\_EXTENSION\_SUPPORTED(3ossl) OpenSSL SSL\_EXTENSION\_SUPPORTED(3ossl)

#### NAME

SSL\_extension\_supported, SSL\_custom\_ext\_add\_cb\_ex,  
SSL\_custom\_ext\_free\_cb\_ex, SSL\_custom\_ext\_parse\_cb\_ex,  
SSL\_CTX\_add\_custom\_ext, SSL\_CTX\_add\_client\_custom\_ext,  
SSL\_CTX\_add\_server\_custom\_ext, custom\_ext\_add\_cb, custom\_ext\_free\_cb,  
custom\_ext\_parse\_cb - custom TLS extension handling

#### SYNOPSIS

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

```
typedef int (*SSL_custom_ext_add_cb_ex)(SSL *s, unsigned int ext_type,  
    unsigned int context,  
    const unsigned char **out,  
    size_t *outlen, X509 *x,  
    size_t chainidx, int *al,  
    void *add_arg);
```

```
typedef void (*SSL_custom_ext_free_cb_ex)(SSL *s, unsigned int ext_type,  
                                         unsigned int context,  
                                         const unsigned char *out,  
                                         void *add_arg);
```

```
typedef int (*SSL_custom_ext_parse_cb_ex)(SSL *s, unsigned int ext_type,  
                                         unsigned int context,  
                                         const unsigned char *in,  
                                         size_t inlen, X509 *x,  
                                         size_t chainidx, int *al,  
                                         void *parse_arg);
```

```
int SSL_CTX_add_custom_ext(SSL_CTX *ctx, unsigned int ext_type,  
                          unsigned int context,  
                          SSL_custom_ext_add_cb_ex add_cb,  
                          SSL_custom_ext_free_cb_ex free_cb,  
                          void *add_arg,  
                          SSL_custom_ext_parse_cb_ex parse_cb,  
                          void *parse_arg);
```

```
typedef int (*custom_ext_add_cb)(SSL *s, unsigned int ext_type,  
                                 const unsigned char **out,  
                                 size_t *outlen, int *al,  
                                 void *add_arg);
```

```
typedef void (*custom_ext_free_cb)(SSL *s, unsigned int ext_type,  
                                   const unsigned char *out,  
                                   void *add_arg);
```

```
typedef int (*custom_ext_parse_cb)(SSL *s, unsigned int ext_type,  
                                   const unsigned char *in,  
                                   size_t inlen, int *al,
```

```
void *parse_arg);
```

```
int SSL_CTX_add_client_custom_ext(SSL_CTX *ctx, unsigned int ext_type,  
    custom_ext_add_cb add_cb,  
    custom_ext_free_cb free_cb, void *add_arg,  
    custom_ext_parse_cb parse_cb,  
    void *parse_arg);
```

```
int SSL_CTX_add_server_custom_ext(SSL_CTX *ctx, unsigned int ext_type,  
    custom_ext_add_cb add_cb,  
    custom_ext_free_cb free_cb, void *add_arg,  
    custom_ext_parse_cb parse_cb,  
    void *parse_arg);
```

```
int SSL_extension_supported(unsigned int ext_type);
```

## DESCRIPTION

`SSL_CTX_add_custom_ext()` adds a custom extension for a TLS/DTLS client or server for all supported protocol versions with extension type `ext_type` and callbacks `add_cb`, `free_cb` and `parse_cb` (see the "EXTENSION CALLBACKS" section below). The context value determines which messages and under what conditions the extension will be added/parsed (see the "EXTENSION CONTEXTS" section below).

`SSL_CTX_add_client_custom_ext()` adds a custom extension for a TLS/DTLS client with extension type `ext_type` and callbacks `add_cb`, `free_cb` and `parse_cb`. This function is similar to `SSL_CTX_add_custom_ext()` except it only applies to clients, uses the older style of callbacks, and implicitly sets the context value to:

```
SSL_EXT_TLS1_2_AND_BELOW_ONLY | SSL_EXT_CLIENT_HELLO  
| SSL_EXT_TLS1_2_SERVER_HELLO | SSL_EXT_IGNORE_ON_RESUMPTION
```

SSL\_CTX\_add\_server\_custom\_ext() adds a custom extension for a TLS/DTLS server with extension type ext\_type and callbacks add\_cb, free\_cb and parse\_cb. This function is similar to SSL\_CTX\_add\_custom\_ext() except it only applies to servers, uses the older style of callbacks, and implicitly sets the context value to the same as for SSL\_CTX\_add\_client\_custom\_ext() above.

The ext\_type parameter corresponds to the extension\_type field of RFC5246 et al. It is not a NID. In all cases the extension type must not be handled by OpenSSL internally or an error occurs.

SSL\_extension\_supported() returns 1 if the extension ext\_type is handled internally by OpenSSL and 0 otherwise.

## EXTENSION CALLBACKS

The callback add\_cb is called to send custom extension data to be included in various TLS messages. The ext\_type parameter is set to the extension type which will be added and add\_arg to the value set when the extension handler was added. When using the new style callbacks the context parameter will indicate which message is currently being constructed e.g. for the ClientHello it will be set to SSL\_EXT\_CLIENT\_HELLO.

If the application wishes to include the extension ext\_type it should set \*out to the extension data, set \*outlen to the length of the extension data and return 1.

If the add\_cb does not wish to include the extension it must return 0.

If add\_cb returns -1 a fatal handshake error occurs using the TLS alert value specified in \*al.

When constructing the ClientHello, if add\_cb is set to NULL a zero

length extension is added for ext\_type. For all other messages if add\_cb is set to NULL then no extension is added.

When constructing a Certificate message the callback will be called for each certificate in the message. The x parameter will indicate the current certificate and the chainidx parameter will indicate the position of the certificate in the message. The first certificate is always the end entity certificate and has a chainidx value of 0. The certificates are in the order that they were received in the Certificate message.

For all messages except the ServerHello and EncryptedExtensions every registered add\_cb is always called to see if the application wishes to add an extension (as long as all requirements of the specified context are met).

For the ServerHello and EncryptedExtension messages every registered add\_cb is called once if and only if the requirements of the specified context are met and the corresponding extension was received in the ClientHello. That is, if no corresponding extension was received in the ClientHello then add\_cb will not be called.

If an extension is added (that is add\_cb returns 1) free\_cb is called (if it is set) with the value of out set by the add callback. It can be used to free up any dynamic extension data set by add\_cb. Since out is constant (to permit use of constant data in add\_cb) applications may need to cast away const to free the data.

The callback parse\_cb receives data for TLS extensions. The callback is only called if the extension is present and relevant for the context (see "EXTENSION CONTEXTS" below).

The extension data consists of inlen bytes in the buffer in for the

extension ext\_type.

If the message being parsed is a TLSv1.3 compatible Certificate message then parse\_cb will be called for each certificate contained within the message. The x parameter will indicate the current certificate and the chainidx parameter will indicate the position of the certificate in the message. The first certificate is always the end entity certificate and has a chainidx value of 0.

If the parse\_cb considers the extension data acceptable it must return 1. If it returns 0 or a negative value a fatal handshake error occurs using the TLS alert value specified in \*al.

The buffer in is a temporary internal buffer which will not be valid after the callback returns.

## EXTENSION CONTEXTS

An extension context defines which messages and under which conditions an extension should be added or expected. The context is built up by performing a bitwise OR of multiple pre-defined values together. The valid context values are:

### SSL\_EXT\_TLS\_ONLY

The extension is only allowed in TLS

### SSL\_EXT\_DTLS\_ONLY

The extension is only allowed in DTLS

### SSL\_EXT\_TLS\_IMPLEMENTATION\_ONLY

The extension is allowed in DTLS, but there is only a TLS implementation available (so it is ignored in DTLS).

### SSL\_EXT\_SSL3\_ALLOWED

Extensions are not typically defined for SSLv3. Setting this value will allow the extension in SSLv3. Applications will not typically need to use this.

#### SSL\_EXT\_TLS1\_2\_AND\_BELOW\_ONLY

The extension is only defined for TLSv1.2/DTLSv1.2 and below. Servers will ignore this extension if it is present in the ClientHello and TLSv1.3 is negotiated.

#### SSL\_EXT\_TLS1\_3\_ONLY

The extension is only defined for TLS1.3 and above. Servers will ignore this extension if it is present in the ClientHello and TLSv1.2 or below is negotiated.

#### SSL\_EXT\_IGNORE\_ON\_RESUMPTION

The extension will be ignored during parsing if a previous session is being successfully resumed.

#### SSL\_EXT\_CLIENT\_HELLO

The extension may be present in the ClientHello message.

#### SSL\_EXT\_TLS1\_2\_SERVER\_HELLO

The extension may be present in a TLSv1.2 or below compatible ServerHello message.

#### SSL\_EXT\_TLS1\_3\_SERVER\_HELLO

The extension may be present in a TLSv1.3 compatible ServerHello message.

#### SSL\_EXT\_TLS1\_3\_ENCRYPTED\_EXTENSIONS

The extension may be present in an EncryptedExtensions message.

#### SSL\_EXT\_TLS1\_3\_HELLO\_RETRY\_REQUEST

The extension may be present in a HelloRetryRequest message.

#### SSL\_EXT\_TLS1\_3\_CERTIFICATE

The extension may be present in a TLSv1.3 compatible Certificate message.

#### SSL\_EXT\_TLS1\_3\_NEW\_SESSION\_TICKET

The extension may be present in a TLSv1.3 compatible NewSessionTicket message.

#### SSL\_EXT\_TLS1\_3\_CERTIFICATE\_REQUEST

The extension may be present in a TLSv1.3 compatible CertificateRequest message.

The context must include at least one message value (otherwise the extension will never be used).

## NOTES

The `add_arg` and `parse_arg` parameters can be set to arbitrary values which will be passed to the corresponding callbacks. They can, for example, be used to store the extension data received in a convenient structure or pass the extension data to be added or freed when adding extensions.

If the same custom extension type is received multiple times a fatal `decode_error` alert is sent and the handshake aborts. If a custom extension is received in a ServerHello/EncryptedExtensions message which was not sent in the ClientHello a fatal `unsupported_extension` alert is sent and the handshake is aborted. The ServerHello/EncryptedExtensions `add_cb` callback is only called if the corresponding extension was received in the ClientHello. This is compliant with the TLS specifications. This behaviour ensures that each callback is called at most once and that an application can never send

unsolicited extensions.

## RETURN VALUES

`SSL_CTX_add_custom_ext()`, `SSL_CTX_add_client_custom_ext()` and `SSL_CTX_add_server_custom_ext()` return 1 for success and 0 for failure.

A failure can occur if an attempt is made to add the same `ext_type` more than once, if an attempt is made to use an extension type handled internally by OpenSSL or if an internal error occurs (for example a memory allocation failure).

`SSL_extension_supported()` returns 1 if the extension `ext_type` is handled internally by OpenSSL and 0 otherwise.

## SEE ALSO

`ssl(7)`

## HISTORY

The `SSL_CTX_add_custom_ext()` function was added in OpenSSL 1.1.1.

## COPYRIGHT

Copyright 2014-2020 The OpenSSL Project Authors. All Rights Reserved.

Licensed under the Apache License 2.0 (the "License"). You may not use this file except in compliance with the License. You can obtain a copy in the file LICENSE in the source distribution or at <https://www.openssl.org/source/license.html>.

3.0.7                    2023-07-13    `SSL_EXTENSION_SUPPORTED(3ossl)`