NAME

X509_STORE_CTX_new_ex, X509_STORE_CTX_new, X509_STORE_CTX_cleanup, X509_STORE_CTX_free, X509_STORE_CTX_init, X509_STORE_CTX_set0_trusted_stack, X509_STORE_CTX_set_cert, X509_STORE_CTX_set0_crls, X509_STORE_CTX_get0_param, X509_STORE_CTX_set0_param, X509_STORE_CTX_get0_untrusted, X509_STORE_CTX_set0_untrusted, X509_STORE_CTX_get_num_untrusted, X509_STORE_CTX_get0_chain, X509_STORE_CTX_set0_verified_chain, X509_STORE_CTX_set_default, X509_STORE_CTX_set_verify, X509_STORE_CTX_verify_fn, X509_STORE_CTX_set_purpose, X509_STORE_CTX_set_trust, X509_STORE_CTX_purpose_inherit - X509_STORE_CTX initialisation

SYNOPSIS

```
#include <openssl/x509_vfy.h>

X509_STORE_CTX *X509_STORE_CTX_new_ex(OSSL_LIB_CTX *libctx, const char *propq);
X509_STORE_CTX *X509_STORE_CTX_new(void);
void X509_STORE_CTX_cleanup(X509_STORE_CTX *ctx);
void X509_STORE_CTX_free(X509_STORE_CTX *ctx);

int X509_STORE_CTX_init(X509_STORE_CTX *ctx, X509_STORE *trust_store,
                        X509 *target, STACK_OF(X509) *untrusted);

void X509_STORE_CTX_set0_trusted_stack(X509_STORE_CTX *ctx, STACK_OF(X509) *sk);

void X509_STORE_CTX_set_cert(X509_STORE_CTX *ctx, X509 *target);
void X509_STORE_CTX_set0_crls(X509_STORE_CTX *ctx, STACK_OF(X509_CRL) *sk);

X509_VERIFY_PARAM *X509_STORE_CTX_get0_param(const X509_STORE_CTX *ctx);
void X509_STORE_CTX_set0_param(X509_STORE_CTX *ctx, X509_VERIFY_PARAM *param);

STACK_OF(X509)* X509_STORE_CTX_get0_untrusted(const X509_STORE_CTX *ctx);
void X509_STORE_CTX_set0_untrusted(X509_STORE_CTX *ctx, STACK_OF(X509) *sk);

int X509_STORE_CTX_get_num_untrusted(const X509_STORE_CTX *ctx);
STACK_OF(X509) *X509_STORE_CTX_get0_chain(const X509_STORE_CTX *ctx);
void X509_STORE_CTX_set0_verified_chain(X509_STORE_CTX *ctx, STACK_OF(X509) *chain);

int X509_STORE_CTX_set_default(X509_STORE_CTX *ctx, const char *name);
typedef int (*X509_STORE_CTX_verify_fn)(X509_STORE_CTX *);
void X509_STORE_CTX_set_verify(X509_STORE_CTX *ctx, X509_STORE_CTX_verify_fn verify);
```

```
int X509_STORE_CTX_set_purpose(X509_STORE_CTX *ctx, int purpose);
int X509_STORE_CTX_set_trust(X509_STORE_CTX *ctx, int trust);
int X509_STORE_CTX_purpose_inherit(X509_STORE_CTX *ctx, int def_purpose,
                                   int purpose, int trust);
```

DESCRIPTION

These functions initialise an **X509_STORE_CTX** structure for subsequent use by **X509_verify_cert**(3) or **X509_STORE_CTX_verify**(3).

X509_STORE_CTX_new_ex() returns a newly initialised **X509_STORE_CTX** structure associated with the specified library context *libctx* and property query string *propq*. Any cryptographic algorithms fetched while performing processing with the X509_STORE_CTX will use that library context and property query string.

X509_STORE_CTX_new() is the same as **X509_STORE_CTX_new_ex()** except that the default library context and a NULL property query string are used.

X509_STORE_CTX_cleanup() internally cleans up an **X509_STORE_CTX** structure. It is used by **X509_STORE_CTX_init()** and **X509_STORE_CTX_free()**.

X509_STORE_CTX_free() completely frees up *ctx*. After this call *ctx* is no longer valid. If *ctx* is NULL nothing is done.

It must be called before each call to **X509_verify_cert**(3) or **X509_STORE_CTX_verify**(3), i.e., a context is only good for one verification. If you want to verify a further certificate or chain with the same *ctx* then you must call **X509_STORE_CTX_init()** again. The trusted certificate store is set to *trust_store* of type **X509_STORE**. This may be NULL because there are no trusted certificates or because they are provided simply as a list using **X509_STORE_CTX_set0_trusted_stack()**. The certificate to be verified is set to *target*, and a list of additional certificates may be provided in *untrusted*, which will be untrusted but may be used to build the chain. Each of the *trust_store*, *target* and *untrusted* parameters can be NULL. Yet note that **X509_verify_cert**(3) and **X509_STORE_CTX_verify**(3) will need a verification target. This can also be set using **X509_STORE_CTX_set_cert()**. For **X509_STORE_CTX_verify**(3), which takes by default the first element of the list of untrusted certificates as its verification target, this can be also set indirectly using **X509_STORE_CTX_set0_untrusted()**.

X509_STORE_CTX_set0_trusted_stack() sets the set of trusted certificates of *ctx* to *sk*. This is an alternative way of specifying trusted certificates instead of using an **X509_STORE** where its complexity is not needed or to make sure that only the given set *sk* of certificates are trusted.

X509_STORE_CTX_set_cert() sets the target certificate to be verified in *ctx* to *target*.

X509_STORE_CTX_set0_verified_chain() sets the validated chain to *chain*. Ownership of the chain is transferred to *ctx*, and so it should not be free'd by the caller.

X509_STORE_CTX_get0_chain() returns the internal pointer used by the *ctx* that contains the constructed (output) chain.

X509_STORE_CTX_set0_crls() sets a set of CRLs to use to aid certificate verification to *sk*. These CRLs will only be used if CRL verification is enabled in the associated **X509_VERIFY_PARAM** structure. This might be used where additional "useful" CRLs are supplied as part of a protocol, for example in a PKCS#7 structure.

X509_STORE_CTX_get0_param() retrieves an internal pointer to the verification parameters associated with *ctx*.

X509_STORE_CTX_set0_param() sets the internal verification parameter pointer to *param*. After this call **param** should not be used.

X509_STORE_CTX_get0_untrusted() retrieves an internal pointer to the stack of untrusted certificates associated with *ctx*.

X509_STORE_CTX_set0_untrusted() sets the internal pointer to the stack of untrusted certificates associated with *ctx* to *sk*. **X509_STORE_CTX_verify()** will take the first element, if any, as its default target if the target certificate is not set explicitly.

X509_STORE_CTX_get_num_untrusted() returns the number of untrusted certificates that were used in building the chain. This is can be used after calling **X509_verify_cert**(3) and similar functions. With **X509_STORE_CTX_verify**(3), this does not count the first chain element.

X509_STORE_CTX_get0_chain() returns the internal pointer used by the *ctx* that contains the validated chain.

Details of the chain building and checking process are described in "Certification Path Building" in **openssl-verification-options**(1) and "Certification Path Validation" in **openssl-verification-options**(1).

X509_STORE_CTX_set0_verified_chain() sets the validated chain used by *ctx* to be *chain*. Ownership of the chain is transferred to *ctx*, and so it should not be free'd by the caller.

X509_STORE_CTX_set_default() looks up and sets the default verification method to *name*. This uses

the function **X509_VERIFY_PARAM_lookup()** to find an appropriate set of parameters from the purpose identifier *name*. Currently defined purposes are "sslclient", "sslserver", "nssslserver", "smimesign", "smimeencrypt", "crlsign", "ocsphelper", "timestampsign", and "any".

X509_STORE_CTX_set_verify() provides the capability for overriding the default verify function. This function is responsible for verifying chain signatures and expiration times.

A verify function is defined as an X509_STORE_CTX_verify type which has the following signature:

```
int (*verify)(X509_STORE_CTX *);
```

This function should receive the current X509_STORE_CTX as a parameter and return 1 on success or 0 on failure.

X509 certificates may contain information about what purposes keys contained within them can be used for. For example "TLS WWW Server Authentication" or "Email Protection". This "key usage" information is held internally to the certificate itself. In addition the trust store containing trusted certificates can declare what purposes we trust different certificates for. This "trust" information is not held within the certificate itself but is "meta" information held alongside it. This "meta" information is associated with the certificate after it is issued and could be determined by a system administrator. For example a certificate might declare that it is suitable for use for both "TLS WWW Server Authentication" and "TLS Client Authentication", but a system administrator might only trust it for the former. An X.509 certificate extension exists that can record extended key usage information to supplement the purpose information described above. This extended mechanism is arbitrarily extensible and not well suited for a generic library API; applications that need to validate extended key usage information in certificates will need to define a custom "purpose" (see below) or supply a nondefault verification callback (**X509_STORE_set_verify_cb_func**(3)).

X509_STORE_CTX_set_purpose() sets the purpose for the target certificate being verified in the *ctx*. Built-in available values for the *purpose* argument are **X509_PURPOSE_SSL_CLIENT**, **X509_PURPOSE_SSL_SERVER**, **X509_PURPOSE_NS_SSL_SERVER**, **X509_PURPOSE_SMIME_SIGN**, **X509_PURPOSE_SMIME_ENCRYPT**, **X509_PURPOSE_CRL_SIGN**, **X509_PURPOSE_ANY**, **X509_PURPOSE_OCSP_HELPER** and **X509_PURPOSE_TIMESTAMP_SIGN**. It is also possible to create a custom purpose value. Setting a purpose will ensure that the key usage declared within certificates in the chain being verified is consistent with that purpose as well as, potentially, other checks. Every purpose also has an associated default trust value which will also be set at the same time. During verification this trust setting will be verified to check it is consistent with the trust set by the system administrator for certificates in the chain.

X509_STORE_CTX_set_trust() sets the trust value for the target certificate being verified in the *ctx*. Built-in available values for the *trust* argument are **X509_TRUST_COMPAT**, **X509_TRUST_SSL_CLIENT**, **X509_TRUST_SSL_SERVER**, **X509_TRUST_EMAIL**, **X509_TRUST_OBJECT_SIGN**, **X509_TRUST_OCSP_SIGN**, **X509_TRUST_OCSP_REQUEST** and **X509_TRUST_TSA**. It is also possible to create a custom trust value. Since **X509_STORE_CTX_set_purpose()** also sets the trust value it is normally sufficient to only call that function. If both are called then **X509_STORE_CTX_set_trust()** should be called after **X509_STORE_CTX_set_purpose()** since the trust setting of the last call will be used.

It should not normally be necessary for end user applications to call **X509_STORE_CTX_purpose_inherit()** directly. Typically applications should call **X509_STORE_CTX_set_purpose()** or **X509_STORE_CTX_set_trust()** instead. Using this function it is possible to set the purpose and trust values for the *ctx* at the same time. Both *ctx* and its internal verification parameter pointer must not be NULL. The *def_purpose* and *purpose* arguments can have the same purpose values as described for **X509_STORE_CTX_set_purpose()** above. The *trust* argument can have the same trust values as described in **X509_STORE_CTX_set_trust()** above. Any of the *def_purpose*, *purpose* or *trust* values may also have the value 0 to indicate that the supplied parameter should be ignored. After calling this function the purpose to be used for verification is set from the *purpose* argument unless the purpose was already set in *ctx* before, and the trust is set from the *trust* argument unless the trust was already set in *ctx* before. If *trust* is 0 then the trust value will be set from the default trust value for *purpose*. If the default trust value for the purpose is *X509_TRUST_DEFAULT* and *trust* is 0 then the default trust value associated with the *def_purpose* value is used for the trust setting instead.

NOTES

The certificates and CRLs in a store are used internally and should **not** be freed up until after the associated **X509_STORE_CTX** is freed.

BUGS

The certificates and CRLs in a context are used internally and should **not** be freed up until after the associated **X509_STORE_CTX** is freed. Copies should be made or reference counts increased instead.

RETURN VALUES

X509_STORE_CTX_new() returns a newly allocated context or NULL if an error occurred.

X509_STORE_CTX_init() returns 1 for success or 0 if an error occurred.

X509_STORE_CTX_get0_param() returns a pointer to an **X509_VERIFY_PARAM** structure or NULL if an error occurred.

X509_STORE_CTX_cleanup(), **X509_STORE_CTX_free()**, **X509_STORE_CTX_set0_trusted_stack()**, **X509_STORE_CTX_set_cert()**, **X509_STORE_CTX_set0_crls()** and **X509_STORE_CTX_set0_param()** do not return values.

X509_STORE_CTX_set_default() returns 1 for success or 0 if an error occurred.

X509_STORE_CTX_get_num_untrusted() returns the number of untrusted certificates used.

SEE ALSO

X509_verify_cert(3), **X509_STORE_CTX_verify**(3), **X509_VERIFY_PARAM_set_flags**(3)

HISTORY

The **X509_STORE_CTX_set0_crls()** function was added in OpenSSL 1.0.0. The **X509_STORE_CTX_get_num_untrusted()** function was added in OpenSSL 1.1.0. The **X509_STORE_CTX_new_ex()** function was added in OpenSSL 3.0.

There is no need to call **X509_STORE_CTX_cleanup()** explicitly since OpenSSL 3.0.

COPYRIGHT