NAME

X509V3_get_d2i, X509V3_add1_i2d, X509V3_EXT_d2i, X509V3_EXT_i2d, X509_get_ext_d2i, X509_add1_ext_i2d, X509_CRL_get_ext_d2i, X509_CRL_add1_ext_i2d, X509_REVOKED_get_ext_d2i, X509_REVOKED_add1_ext_i2d, X509_get0_extensions, X509_CRL_get0_extensions, X509_REVOKED_get0_extensions - X509 extension decode and encode functions

SYNOPSIS

```
#include <openssl/x509v3.h>

void *X509V3_get_d2i(const STACK_OF(X509_EXTENSION) *x, int nid, int *crit,
                     int *idx);
int X509V3_add1_i2d(STACK_OF(X509_EXTENSION) **x, int nid, void *value,
                    int crit, unsigned long flags);

void *X509V3_EXT_d2i(X509_EXTENSION *ext);
X509_EXTENSION *X509V3_EXT_i2d(int ext_nid, int crit, void *ext_struc);

void *X509_get_ext_d2i(const X509 *x, int nid, int *crit, int *idx);
int X509_add1_ext_i2d(X509 *x, int nid, void *value, int crit,
                      unsigned long flags);

void *X509_CRL_get_ext_d2i(const X509_CRL *crl, int nid, int *crit, int *idx);
int X509_CRL_add1_ext_i2d(X509_CRL *crl, int nid, void *value, int crit,
                          unsigned long flags);

void *X509_REVOKED_get_ext_d2i(const X509_REVOKED *r, int nid, int *crit, int *idx);
int X509_REVOKED_add1_ext_i2d(X509_REVOKED *r, int nid, void *value, int crit,
                              unsigned long flags);

const STACK_OF(X509_EXTENSION) *X509_get0_extensions(const X509 *x);
const STACK_OF(X509_EXTENSION) *X509_CRL_get0_extensions(const X509_CRL *crl);
const STACK_OF(X509_EXTENSION) *X509_REVOKED_get0_extensions(const X509_REVOKED *r);
```

DESCRIPTION

X509V3_get_d2i() looks for an extension with OID *nid* in the extensions *x* and, if found, decodes it. If *idx* is NULL then only one occurrence of an extension is permissible, otherwise the first extension after index **idx* is returned and **idx* updated to the location of the extension. If *crit* is not NULL then **crit* is set to a status value: -2 if the extension occurs multiple times (this is only returned if *idx* is NULL), -1 if the extension could not be found, 0 if the extension is found and is not critical and 1 if critical. A

pointer to an extension specific structure or NULL is returned.

X509V3_add1_i2d() adds extension *value* to STACK **x* (allocating a new STACK if necessary) using OID *nid* and criticality *crit* according to *flags*.

X509V3_EXT_d2i() attempts to decode the ASN.1 data contained in extension *ext* and returns a pointer to an extension specific structure or NULL if the extension could not be decoded (invalid syntax or not supported).

X509V3_EXT_i2d() encodes the extension specific structure *ext_struc* with OID *ext_nid* and criticality *crit*.

X509_get_ext_d2i() and **X509_add1_ext_i2d()** operate on the extensions of certificate *x*. They are otherwise identical to **X509V3_get_d2i()** and **X509V3_add1_i2d()**.

X509_CRL_get_ext_d2i() and **X509_CRL_add1_ext_i2d()** operate on the extensions of CRL *crl*. They are otherwise identical to **X509V3_get_d2i()** and **X509V3_add1_i2d()**.

X509_REVOKED_get_ext_d2i() and **X509_REVOKED_add1_ext_i2d()** operate on the extensions of **X509_REVOKED** structure *r* (i.e for CRL entry extensions). They are otherwise identical to **X509V3_get_d2i()** and **X509V3_add1_i2d()**.

X509_get0_extensions(), **X509_CRL_get0_extensions()** and **X509_REVOKED_get0_extensions()** return a STACK of all the extensions of a certificate, a CRL or a CRL entry respectively.

NOTES

In almost all cases an extension can occur at most once and multiple occurrences is an error. Therefore, the *idx* parameter is usually NULL.

The *flags* parameter may be one of the following values.

X509V3_ADD_DEFAULT appends a new extension only if the extension does not exist. An error is returned if the extension exists.

X509V3_ADD_APPEND appends a new extension, ignoring whether the extension exists.

X509V3_ADD_REPLACE replaces an existing extension. If the extension does not exist, appends a new extension.

X509V3_ADD_REPLACE_EXISTING replaces an existing extension. If the extension does not exist,

returns an error.

X509V3_ADD_KEEP_EXISTING appends a new extension only if the extension does not exist. An error is **not** returned if the extension exists.

X509V3_ADD_DELETE deletes and frees an existing extension. If the extension does not exist, returns an error. No new extension is added.

If **X509V3_ADD_SILENT** is bitwise ORed with *flags*: any error returned will not be added to the error queue.

The function **X509V3_get_d2i()** and its variants will return NULL if the extension is not found, occurs multiple times or cannot be decoded. It is possible to determine the precise reason by checking the value of **crit*.

The function **X509V3_add1_i2d()** and its variants allocate **X509_EXTENSION** objects on STACK **x* depending on *flags*. The **X509_EXTENSION** objects must be explicitly freed using **X509_EXTENSION_free()**.

SUPPORTED EXTENSIONS

The following sections contain a list of all supported extensions including their name and NID.

PKIX Certificate Extensions

The following certificate extensions are defined in PKIX standards such as RFC5280.

Basic Constraints	NID_basic_constraints
Key Usage	NID_key_usage
Extended Key Usage	NID_ext_key_usage
Subject Key Identifier	NID_subject_key_identifier
Authority Key Identifier	NID_authority_key_identifier
Private Key Usage Period	NID_private_key_usage_period
Subject Alternative Name	NID_subject_alt_name
Issuer Alternative Name	NID_issuer_alt_name
Authority Information Access	NID_info_access
Subject Information Access	NID_sinfo_access

Name Constraints	NID_name_constraints
Certificate Policies	NID_certificate_policies
Policy Mappings	NID_policy_mappings
Policy Constraints	NID_policy_constraints
Inhibit Any Policy	NID_inhibit_any_policy
TLS Feature	NID_tlsfeature

Netscape Certificate Extensions

The following are (largely obsolete) Netscape certificate extensions.

Netscape Cert Type	NID_netscape_cert_type
Netscape Base Url	NID_netscape_base_url
Netscape Revocation Url	NID_netscape_revocation_url
Netscape CA Revocation Url	NID_netscape_ca_revocation_url
Netscape Renewal Url	NID_netscape_renewal_url
Netscape CA Policy Url	NID_netscape_ca_policy_url
Netscape SSL Server Name	NID_netscape_ssl_server_name
Netscape Comment	NID_netscape_comment

Miscellaneous Certificate Extensions

Strong Extranet ID	NID_sxnet
Proxy Certificate Information	NID_proxyCertInfo

PKIX CRL Extensions

The following are CRL extensions from PKIX standards such as RFC5280.

CRL Number	NID_crl_number
CRL Distribution Points	NID_crl_distribution_points
Delta CRL Indicator	NID_delta_crl
Freshest CRL	NID_freshest_crl
Invalidity Date	NID_invalidity_date
Issuing Distribution Point	NID_issuing_distribution_point

The following are CRL entry extensions from PKIX standards such as RFC5280.

CRL Reason Code	NID_crl_reason
Certificate Issuer	NID_certificate_issuer

OCSP Extensions

OCSP Nonce	NID_id_pkix_OCSP_Nonce
OCSP CRL ID	NID_id_pkix_OCSP_CrlID
Acceptable OCSP Responses	NID_id_pkix_OCSP_acceptableResponses
OCSP No Check	NID_id_pkix_OCSP_noCheck
OCSP Archive Cutoff	NID_id_pkix_OCSP_archiveCutoff
OCSP Service Locator	NID_id_pkix_OCSP_serviceLocator
Hold Instruction Code	NID_hold_instruction_code

Certificate Transparency Extensions

The following extensions are used by certificate transparency, RFC6962

CT Precertificate SCTs	NID_ct_precert_scts
CT Certificate SCTs	NID_ct_cert_scts

RETURN VALUES

X509V3_get_d2i(), its variants, and **X509V3_EXT_d2i()** return a pointer to an extension specific structure or NULL if an error occurs.

X509V3_add1_i2d() and its variants return 1 if the operation is successful and 0 if it fails due to a non-fatal error (extension not found, already exists, cannot be encoded) or -1 due to a fatal error such as a memory allocation failure.

X509V3_EXT_i2d() returns a pointer to an **X509_EXTENSION** structure or NULL if an error occurs.

X509_get0_extensions(), **X509_CRL_get0_extensions()** and **X509_REVOKED_get0_extensions()** return a stack of extensions. They return NULL if no extensions are present.

SEE ALSO

d2i_X509(3), **ERR_get_error**(3), **X509_CRL_get0_by_serial**(3), **X509_get0_signature**(3), **X509_get_ext_d2i**(3), **X509_get_extension_flags**(3), **X509_get_pubkey**(3), **X509_get_subject_name**(3), **X509_get_version**(3), **X509_NAME_add_entry_by_txt**(3), **X509_NAME_ENTRY_get_object**(3), **X509_NAME_get_index_by_NID**(3), **X509_NAME_print_ex**(3), **X509_new**(3), **X509_sign**(3), **X509_verify_cert**(3)

COPYRIGHT