NAME

d2i_X509, i2d_X509, d2i_X509_bio, d2i_X509_fp, i2d_X509_bio, i2d_X509_fp - X509 encode and decode functions

SYNOPSIS

```
#include <openssl/x509.h>

X509 *d2i_X509(X509 **px, const unsigned char **in, int len);
int i2d_X509(X509 *x, unsigned char **out);

X509 *d2i_X509_bio(BIO *bp, X509 **x);
X509 *d2i_X509_fp(FILE *fp, X509 **x);

int i2d_X509_bio(BIO *bp, X509 *x);
int i2d_X509_fp(FILE *fp, X509 *x);

int i2d_re_X509_tbs(X509 *x, unsigned char **out);
```

DESCRIPTION

The X509 encode and decode routines encode and parse an **X509** structure, which represents an X509 certificate.

d2i_X509() attempts to decode **len** bytes at ***in**. If successful a pointer to the **X509** structure is returned. If an error occurred then **NULL** is returned. If **px** is not **NULL** then the returned structure is written to ***px**. If ***px** is not **NULL** then it is assumed that ***px** contains a valid **X509** structure and an attempt is made to reuse it. This ''reuse'' capability is present for historical compatibility but its use is **strongly discouraged** (see BUGS below, and the discussion in the RETURN VALUES section).

If the call is successful ***in** is incremented to the byte following the parsed data.

i2d_X509() encodes the structure pointed to by **x** into DER format. If **out** is not **NULL** is writes the DER encoded data to the buffer at ***out**, and increments it to point after the data just written. If the return value is negative an error occurred, otherwise it returns the length of the encoded data.

For OpenSSL 0.9.7 and later if ***out** is **NULL** memory will be allocated for a buffer and the encoded data written to it. In this case ***out** is not incremented and it points to the start of the data just written.

d2i_X509_bio() is similar to *d2i_X509()* except it attempts to parse data from BIO **bp**.

d2i_X509_fp() is similar to *d2i_X509()* except it attempts to parse data from FILE pointer **fp**.

i2d_X509_bio() is similar to *i2d_X509()* except it writes the encoding of the structure **x** to BIO **bp** and it returns 1 for success and 0 for failure.

i2d_X509_fp() is similar to *i2d_X509()* except it writes the encoding of the structure **x** to BIO **bp** and it returns 1 for success and 0 for failure.

i2d_re_X509_tbs() is similar to *i2d_X509()* except it encodes only the TBSCertificate portion of the certificate.

NOTES

The letters **i** and **d** in for example **i2d_X509** stand for ''internal'' (that is an internal C structure) and ''DER''. So **i2d_X509** converts from internal to DER. The ''re'' in **i2d_re_X509_tbs** stands for ''re-encode'', and ensures that a fresh encoding is generated in case the object has been modified after creation (see the BUGS section).

The functions can also understand **BER** forms.

The actual X509 structure passed to *i2d_X509()* must be a valid populated **X509** structure it can **not** simply be fed with an empty structure such as that returned by *X509_new()*.

The encoded data is in binary form and may contain embedded zeroes. Therefore any FILE pointers or BIOs should be opened in binary mode. Functions such as ***strlen()*** will **not** return the correct length of the encoded structure.

The ways that ***in** and ***out** are incremented after the operation can trap the unwary. See the **WARNINGS** section for some common errors.

The reason for the auto increment behaviour is to reflect a typical usage of ASN1 functions: after one structure is encoded or decoded another will processed after it.

EXAMPLES

Allocate and encode the DER encoding of an X509 structure:

```
int len;
unsigned char *buf, *p;

len = i2d_X509(x, NULL);

buf = OPENSSL_malloc(len);

if (buf == NULL)
/* error */

p = buf;

i2d_X509(x, &p);
```

If you are using OpenSSL 0.9.7 or later then this can be simplified to:

```
int len;
unsigned char *buf;

buf = NULL;

len = i2d_X509(x, &buf);

if (len < 0)
/* error */
```

Attempt to decode a buffer:

```
X509 *x;

unsigned char *buf, *p;

int len;

/* Something to setup buf and len */

p = buf;

x = d2i_X509(NULL, &p, len);

if (x == NULL)
/* Some error */
```

Alternative technique:

```
X509 *x;
```

```
unsigned char *buf, *p;

int len;

/* Something to setup buf and len */

p = buf;

x = NULL;

if(!d2i_X509(&x, &p, len))
/* Some error */
```

WARNINGS

The use of temporary variable is mandatory. A common mistake is to attempt to use a buffer directly as follows:

```
int len;
unsigned char *buf;

len = i2d_X509(x, NULL);

buf = OPENSSL_malloc(len);

if (buf == NULL)
/* error */

i2d_X509(x, &buf);

/* Other stuff ... */

OPENSSL_free(buf);
```

This code will result in **buf** apparently containing garbage because it was incremented after the call to point after the data just written. Also **buf** will no longer contain the pointer allocated by ***OPENSSL_malloc()*** and the subsequent call to ***OPENSSL_free()*** may well crash.

The auto allocation feature (setting buf to NULL) only works on OpenSSL 0.9.7 and later. Attempts to use it on earlier versions will typically cause a segmentation violation.

Another trap to avoid is misuse of the **xp** argument to ***d2i_X509()***:

```
X509 *x;

if (!d2i_X509(&x, &p, len))
/* Some error */
```

This will probably crash somewhere in ***d2i_X509()***. The reason for this is that the variable **x** is uninitialized and an attempt will be made to interpret its (invalid) value as an **X509** structure, typically causing a segmentation violation. If **x** is set to NULL first then this will not happen.

BUGS

In some versions of OpenSSL the "reuse" behaviour of *d2i_X509()* when ***px** is valid is broken and some parts of the reused structure may persist if they are not present in the new one. As a result the use of this "reuse" behaviour is strongly discouraged.

i2d_X509() will not return an error in many versions of OpenSSL, if mandatory fields are not initialized due to a programming error then the encoded structure may contain invalid data or omit the fields entirely and will not be parsed by *d2i_X509()*. This may be fixed in future so code should not assume that *i2d_X509()* will always succeed.

The encoding of the TBSCertificate portion of a certificate is cached in the **X509** structure internally to improve encoding performance and to ensure certificate signatures are verified correctly in some certificates with broken (non-DER) encodings.

Any function which encodes an X509 structure such as *i2d_X509()*, *i2d_X509_fp()* or *i2d_X509_bio()* may return a stale encoding if the **X509** structure has been modified after deserialization or previous serialization.

If, after modification, the **X509** object is re-signed with *X509_sign()*, the encoding is automatically renewed. Otherwise, the encoding of the TBSCertificate portion of the **X509** can be manually renewed by calling *i2d_re_X509_tbs()*.

RETURN VALUES

d2i_X509(), *d2i_X509_bio()* and *d2i_X509_fp()* return a valid **X509** structure or **NULL** if an error occurs. The error code that can be obtained by *ERR_get_error(3)*. If the "reuse" capability has been used with a valid X509 structure being passed in via **px** then the object is not freed in the event of error but may be in a potentially invalid or inconsistent state.

i2d_X509() returns the number of bytes successfully encoded or a negative value if an error occurs. The error code can be obtained by *ERR_get_error(3)*.

i2d_X509_bio() and *i2d_X509_fp()* return 1 for success and 0 if an error occurs The error code can be obtained by *ERR_get_error(3)*.

SEE ALSO

ERR_get_error(3)

HISTORY

d2i_X509, i2d_X509, d2i_X509_bio, d2i_X509_fp, i2d_X509_bio and i2d_X509_fp are available in all versions of SSLeay and OpenSSL.