NAME

d2i_ACCESS_DESCRIPTION, d2i_ADMISSIONS, d2i_ADMISSION_SYNTAX, d2i_ASIdOrRange, d2i_ASIdentifierChoice, d2i_ASIdentifiers, d2i_ASN1_BIT_STRING, d2i_ASN1_BMPSTRING, d2i_ASN1_ENUMERATED, d2i_ASN1_GENERALIZEDTIME, d2i_ASN1_GENERALSTRING, d2i_ASN1_IA5STRING, d2i_ASN1_INTEGER, d2i_ASN1_NULL, d2i_ASN1_OBJECT, d2i_ASN1_OCTET_STRING, d2i_ASN1_PRINTABLE, d2i_ASN1_PRINTABLESTRING, d2i_ASN1_SEQUENCE_ANY, d2i_ASN1_SET_ANY, d2i_ASN1_T61STRING, d2i_ASN1_TIME, d2i_ASN1_TYPE, d2i_ASN1_UINTEGER, d2i_ASN1_UNIVERSALSTRING, d2i_ASN1_UTCTIME, d2i_ASN1_UTF8STRING, d2i_ASN1_VISIBLESTRING, d2i_ASRange, d2i_AUTHORITY_INFO_ACCESS, d2i_AUTHORITY_KEYID, d2i_BASIC_CONSTRAINTS, d2i_CERTIFICATEPOLICIES, d2i_CMS_ContentInfo, d2i_CMS_ReceiptRequest, d2i_CMS_bio, d2i_CRL_DIST_POINTS, d2i_DHxparams, d2i_DIRECTORYSTRING, d2i_DISPLAYTEXT, d2i_DIST_POINT, d2i_DIST_POINT_NAME, d2i_DSAPrivateKey, d2i_DSAPrivateKey_bio, d2i_DSAPrivateKey_fp, d2i_DSAPublicKey, d2i_DSA_PUBKEY, d2i_DSA_PUBKEY_bio, d2i_DSA_PUBKEY_fp, d2i_DSA_SIG, d2i_DSAparams, d2i_ECDSA_SIG, d2i_ECPKParameters, d2i_ECParameters, d2i_ECPrivateKey, d2i_ECPrivateKey_bio, d2i_ECPrivateKey_fp, d2i_EC_PUBKEY, d2i_EC_PUBKEY_bio, d2i_EC_PUBKEY_fp, d2i_EDIPARTYNAME, d2i_ESS_CERT_ID, d2i_ESS_ISSUER_SERIAL, d2i_ESS_SIGNING_CERT, d2i_EXTENDED_KEY_USAGE, d2i_GENERAL_NAME, d2i_GENERAL_NAMES, d2i_IPAddressChoice, d2i_IPAddressFamily, d2i_IPAddressOrRange, d2i_IPAddressRange, d2i_ISSUING_DIST_POINT, d2i_NAMING_AUTHORITY, d2i_NETSCAPE_CERT_SEQUENCE, d2i_NETSCAPE_SPKAC, d2i_NETSCAPE_SPKI, d2i_NOTICEREF, d2i_OCSP_BASICRESP, d2i_OCSP_CERTID, d2i_OCSP_CERTSTATUS, d2i_OCSP_CRLID, d2i_OCSP_ONEREQ, d2i_OCSP_REQINFO, d2i_OCSP_REQUEST, d2i_OCSP_RESPBYTES, d2i_OCSP_RESPDATA, d2i_OCSP_RESPID, d2i_OCSP_RESPONSE, d2i_OCSP_REVOKEDINFO, d2i_OCSP_SERVICELOC, d2i_OCSP_SIGNATURE, d2i_OCSP_SINGLERESP, d2i_OTHERNAME, d2i_PBE2PARAM, d2i_PBEPARAM, d2i_PBKDF2PARAM, d2i_PKCS12, d2i_PKCS12_BAGS, d2i_PKCS12_MAC_DATA, d2i_PKCS12_SAFEBAG, d2i_PKCS12_bio, d2i_PKCS12_fp, d2i_PKCS7, d2i_PKCS7_DIGEST, d2i_PKCS7_ENCRYPT, d2i_PKCS7_ENC_CONTENT, d2i_PKCS7_ENVELOPE, d2i_PKCS7_ISSUER_AND_SERIAL, d2i_PKCS7_RECIP_INFO, d2i_PKCS7_SIGNED, d2i_PKCS7_SIGNER_INFO, d2i_PKCS7_SIGN_ENVELOPE, d2i_PKCS7_bio, d2i_PKCS7_fp, d2i_PKCS8_PRIV_KEY_INFO, d2i_PKCS8_PRIV_KEY_INFO_bio, d2i_PKCS8_PRIV_KEY_INFO_fp, d2i_PKCS8_bio, d2i_PKCS8_fp, d2i_PKEY_USAGE_PERIOD, d2i_POLICYINFO, d2i_POLICYQUALINFO, d2i_PROFESSION_INFO, d2i_PROXY_CERT_INFO_EXTENSION, d2i_PROXY_POLICY, d2i_RSAPrivateKey, d2i_RSAPrivateKey_bio, d2i_RSAPrivateKey_fp, d2i_RSAPublicKey, d2i_RSAPublicKey_bio, d2i_RSAPublicKey_fp, d2i_RSA_OAEP_PARAMS, d2i_RSA_PSS_PARAMS, d2i_RSA_PUBKEY, d2i_RSA_PUBKEY_bio, d2i_RSA_PUBKEY_fp, d2i_SCRYPT_PARAMS, d2i_SCT_LIST, d2i_SXNET, d2i_SXNETID, d2i_TS_ACCURACY, d2i_TS_MSG_IMPRINT, d2i_TS_MSG_IMPRINT_bio, d2i_TS_MSG_IMPRINT_fp, d2i_TS_REQ, d2i_TS_REQ_bio, d2i_TS_REQ_fp, d2i_TS_RESP, d2i_TS_RESP_bio, d2i_TS_RESP_fp, d2i_TS_STATUS_INFO, d2i_TS_TST_INFO, d2i_TS_TST_INFO_bio, d2i_TS_TST_INFO_fp, d2i_USERNOTICE, d2i_X509, d2i_X509_bio, d2i_X509_fp, d2i_X509_ALGOR, d2i_X509_ALGORS, d2i_X509_ATTRIBUTE, d2i_X509_CERT_AUX, d2i_X509_CINF, d2i_X509_CRL, d2i_X509_CRL_INFO, d2i_X509_CRL_bio, d2i_X509_CRL_fp, d2i_X509_EXTENSION, d2i_X509_EXTENSIONS, d2i_X509_NAME, d2i_X509_NAME_ENTRY, d2i_X509_PUBKEY, d2i_X509_REQ, d2i_X509_REQ_INFO, d2i_X509_REQ_bio, d2i_X509_REQ_fp, d2i_X509_REVOKED, d2i_X509_SIG, d2i_X509_VAL, i2d_ACCESS_DESCRIPTION, i2d_ADMISSIONS, i2d_ADMISSION_SYNTAX, i2d_ASIdOrRange, i2d_ASIdentifierChoice, i2d_ASIdentifiers, i2d_ASN1_BIT_STRING, i2d_ASN1_BMPSTRING, i2d_ASN1_ENUMERATED, i2d_ASN1_GENERALIZEDTIME, i2d_ASN1_GENERALSTRING, i2d_ASN1_IA5STRING, i2d_ASN1_INTEGER, i2d_ASN1_NULL, i2d_ASN1_OBJECT, i2d_ASN1_OCTET_STRING, i2d_ASN1_PRINTABLE, i2d_ASN1_PRINTABLESTRING, i2d_ASN1_SEQUENCE_ANY, i2d_ASN1_SET_ANY, i2d_ASN1_T61STRING, i2d_ASN1_TIME, i2d_ASN1_TYPE, i2d_ASN1_UNIVERSALSTRING, i2d_ASN1_UTCTIME, i2d_ASN1_UTF8STRING,

i2d_ASN1_VISIBLESTRING, i2d_ASN1_bio_stream, i2d_ASRange, i2d_AUTHORITY_INFO_ACCESS, i2d_AUTHORITY_KEYID, i2d_BASIC_CONSTRAINTS, i2d_CERTIFICATEPOLICIES, i2d_CMS_ContentInfo, i2d_CMS_ReceiptRequest, i2d_CMS_bio, i2d_CRL_DIST_POINTS, i2d_DHxparams, i2d_DIRECTORYSTRING, i2d_DISPLAYTEXT, i2d_DIST_POINT, i2d_DIST_POINT_NAME, i2d_DSAPrivateKey, i2d_DSAPrivateKey_bio, i2d_DSAPrivateKey_fp, i2d_DSAPublicKey, i2d_DSA_PUBKEY, i2d_DSA_PUBKEY_bio, i2d_DSA_PUBKEY_fp, i2d_DSA_SIG, i2d_DSAparams, i2d_ECDSA_SIG, i2d_ECPKParameters, i2d_ECParameters, i2d_ECPrivateKey, i2d_ECPrivateKey_bio, i2d_ECPrivateKey_fp, i2d_EC_PUBKEY, i2d_EC_PUBKEY_bio, i2d_EC_PUBKEY_fp, i2d_EDIPARTYNAME, i2d_ESS_CERT_ID, i2d_ESS_ISSUER_SERIAL, i2d_ESS_SIGNING_CERT, i2d_EXTENDED_KEY_USAGE, i2d_GENERAL_NAME, i2d_GENERAL_NAMES, i2d_IPAddressChoice, i2d_IPAddressFamily, i2d_IPAddressOrRange, i2d_IPAddressRange, i2d_ISSUING_DIST_POINT, i2d_NAMING_AUTHORITY, i2d_NETSCAPE_CERT_SEQUENCE, i2d_NETSCAPE_SPKAC, i2d_NETSCAPE_SPKI, i2d_NOTICEREF, i2d_OCSP_BASICRESP, i2d_OCSP_CERTID, i2d_OCSP_CERTSTATUS, i2d_OCSP_CRLID, i2d_OCSP_ONEREQ, i2d_OCSP_REQINFO, i2d_OCSP_REQUEST, i2d_OCSP_RESPBYTES, i2d_OCSP_RESPDATA, i2d_OCSP_RESPID, i2d_OCSP_RESPONSE, i2d_OCSP_REVOKEDINFO, i2d_OCSP_SERVICELOC, i2d_OCSP_SIGNATURE, i2d_OCSP_SINGLERESP, i2d_OTHERNAME, i2d_PBE2PARAM, i2d_PBEPARAM, i2d_PBKDF2PARAM, i2d_PKCS12, i2d_PKCS12_BAGS, i2d_PKCS12_MAC_DATA, i2d_PKCS12_SAFEBAG, i2d_PKCS12_bio, i2d_PKCS12_fp, i2d_PKCS7, i2d_PKCS7_DIGEST, i2d_PKCS7_ENCRYPT, i2d_PKCS7_ENC_CONTENT, i2d_PKCS7_ENVELOPE, i2d_PKCS7_ISSUER_AND_SERIAL, i2d_PKCS7_NDEF, i2d_PKCS7_RECIP_INFO, i2d_PKCS7_SIGNED, i2d_PKCS7_SIGNER_INFO, i2d_PKCS7_SIGN_ENVELOPE, i2d_PKCS7_bio, i2d_PKCS7_fp, i2d_PKCS8PrivateKeyInfo_bio, i2d_PKCS8PrivateKeyInfo_fp, i2d_PKCS8_PRIV_KEY_INFO, i2d_PKCS8_PRIV_KEY_INFO_bio, i2d_PKCS8_PRIV_KEY_INFO_fp, i2d_PKCS8_bio, i2d_PKCS8_fp, i2d_PKEY_USAGE_PERIOD, i2d_POLICYINFO, i2d_POLICYQUALINFO, i2d_PROFESSION_INFO, i2d_PROXY_CERT_INFO_EXTENSION, i2d_PROXY_POLICY, i2d_RSAPrivateKey, i2d_RSAPrivateKey_bio, i2d_RSAPrivateKey_fp, i2d_RSAPublicKey, i2d_RSAPublicKey_bio, i2d_RSAPublicKey_fp, i2d_RSA_OAEP_PARAMS, i2d_RSA_PSS_PARAMS, i2d_RSA_PUBKEY, i2d_RSA_PUBKEY_bio, i2d_RSA_PUBKEY_fp, i2d_SCRYPT_PARAMS, i2d_SCT_LIST, i2d_SXNET, i2d_SXNETID, i2d_TS_ACCURACY, i2d_TS_MSG_IMPRINT, i2d_TS_MSG_IMPRINT_bio, i2d_TS_MSG_IMPRINT_fp, i2d_TS_REQ, i2d_TS_REQ_bio, i2d_TS_REQ_fp, i2d_TS_RESP, i2d_TS_RESP_bio, i2d_TS_RESP_fp, i2d_TS_STATUS_INFO, i2d_TS_TST_INFO, i2d_TS_TST_INFO_bio, i2d_TS_TST_INFO_fp, i2d_USERNOTICE, i2d_X509, i2d_X509_bio, i2d_X509_fp, i2d_X509_ALGOR, i2d_X509_ALGORS, i2d_X509_ATTRIBUTE, i2d_X509_CERT_AUX, i2d_X509_CINF, i2d_X509_CRL, i2d_X509_CRL_INFO, i2d_X509_CRL_bio, i2d_X509_CRL_fp, i2d_X509_EXTENSION, i2d_X509_EXTENSIONS, i2d_X509_NAME, i2d_X509_NAME_ENTRY, i2d_X509_PUBKEY, i2d_X509_REQ, i2d_X509_REQ_INFO, i2d_X509_REQ_bio, i2d_X509_REQ_fp, i2d_X509_REVOKED, i2d_X509_SIG, i2d_X509_VAL, – convert objects from/to ASN.1/DER representation

SYNOPSIS

```
TYPE *d2i_TYPE(TYPE **a, const unsigned char **ppin, long length);
TYPE *d2i_TYPE_bio(BIO *bp, TYPE **a);
TYPE *d2i_TYPE_fp(FILE *fp, TYPE **a);

int i2d_TYPE(TYPE *a, unsigned char **ppout);
int i2d_TYPE_fp(FILE *fp, TYPE *a);
int i2d_TYPE_bio(BIO *bp, TYPE *a);
```

DESCRIPTION

In the description here, *TYPE* is used a placeholder for any of the OpenSSL datatypes, such as *X509_CRL*. The function parameters *ppin* and *ppout* are generally either both named *pp* in the headers, or *in* and *out*.

These functions convert OpenSSL objects to and from their ASN.1/DER encoding. Unlike the C structures

which can have pointers to sub-objects within, the DER is a serialized encoding, suitable for sending over the network, writing to a file, and so on.

d2i_TYPE() attempts to decode **len** bytes at ***ppin**. If successful a pointer to the **TYPE** structure is returned and ***ppin** is incremented to the byte following the parsed data. If **a** is not **NULL** then a pointer to the returned structure is also written to ***a**. If an error occurred then **NULL** is returned.

On a successful return, if ***a** is not **NULL** then it is assumed that ***a** contains a valid **TYPE** 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).

d2i_TYPE_bio() is similar to **d2i_TYPE()** except it attempts to parse data from BIO **bp**.

d2i_TYPE_fp() is similar to **d2i_TYPE()** except it attempts to parse data from FILE pointer **fp**.

i2d_TYPE() encodes the structure pointed to by **a** into DER format. If **ppout** is not **NULL**, it writes the DER encoded data to the buffer at ***ppout**, 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.

If ***ppout** is **NULL** memory will be allocated for a buffer and the encoded data written to it. In this case ***ppout** is not incremented and it points to the start of the data just written.

i2d_TYPE_bio() is similar to **i2d_TYPE()** except it writes the encoding of the structure **a** to BIO **bp** and it returns 1 for success and 0 for failure.

i2d_TYPE_fp() is similar to **i2d_TYPE()** except it writes the encoding of the structure **a** to BIO **bp** and it returns 1 for success and 0 for failure.

These routines do not encrypt private keys and therefore offer no security; use **PEM_write_PrivateKey(3)** or similar for writing to files.

NOTES

The letters **i** and **d** in **i2d_TYPE** stand for ‘‘internal’’ (that is, an internal C structure) and ‘‘DER’’ respectively. So **i2d_TYPE** converts from internal to DER.

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

The actual TYPE structure passed to **i2d_TYPE()** must be a valid populated **TYPE** structure — it **cannot** simply be fed with an empty structure such as that returned by **TYPE_new()**.

The encoded data is in binary form and may contain embedded zeros. 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 ***ppin** and ***ppout** are incremented after the operation can trap the unwary. See the **WARNINGS** section for some common errors. The reason for this-auto increment behaviour is to reflect a typical usage of ASN1 functions: after one structure is encoded or decoded another will be processed after it.

The following points about the data types might be useful:

ASN1_OBJECT
: Represents an ASN1 OBJECT IDENTIFIER.

DHparams
: Represents a PKCS#3 DH parameters structure.

DHxparams
: Represents an ANSI X9.42 DH parameters structure.

DSA_PUBKEY
: Represents a DSA public key using a **SubjectPublicKeyInfo** structure.

DSAPublicKey, DSAPrivateKey
: Use a non-standard OpenSSL format and should be avoided; use **DSA_PUBKEY**, **PEM_write_PrivateKey(3)** , or similar instead.

ECDSA_SIG
Represents an ECDSA signature.

RSAPublicKey
Represents a PKCS#1 RSA public key structure.

X509_ALGOR
Represents an **AlgorithmIdentifier** structure as used in IETF RFC 6960 and elsewhere.

X509_Name
Represents a **Name** type as used for subject and issuer names in IETF RFC 6960 and elsewhere.

X509_REQ
Represents a PKCS#10 certificate request.

X509_SIG
Represents the **DigestInfo** structure defined in PKCS#1 and PKCS#7.

RETURN VALUES

d2i_TYPE(), **d2i_TYPE_bio()** and **d2i_TYPE_fp()** return a valid **TYPE** structure or **NULL** if an error occurs. If the ''reuse'' capability has been used with a valid structure being passed in via **a**, then the object is freed in the event of error and ***a** is set to NULL.

i2d_TYPE() returns the number of bytes successfully encoded or a negative value if an error occurs.

i2d_TYPE_bio() and **i2d_TYPE_fp()** return 1 for success and 0 if an error occurs.

EXAMPLES

Allocate and encode the DER encoding of an X509 structure:

```
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;
const unsigned char *p;
int len;

/* Set up buf and len to point to the input buffer. */
p = buf;
x = d2i_X509(NULL, &p, len);
if (x == NULL)
    /* error */
```

Alternative technique:

```
X509 *x;
unsigned char *buf;
const unsigned char *p;
int len;

/* Set up buf and len to point to the input buffer. */
p = buf;
x = NULL;

if (d2i_X509(&x, &p, len) == NULL)
```

```
    /* error */
```

WARNINGS

Using a 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);
...
i2d_X509(x, &buf);
...
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()** is likely to crash.

Another trap to avoid is misuse of the **a** argument to **d2i_TYPE()**:

```
X509 *x;

if (d2i_X509(&x, &p, len) == NULL)
    /* 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_TYPE()** when ***a** is valid is broken and some parts of the reused structure may persist if they are not present in the new one. Additionally, in versions of OpenSSL prior to 1.1.0, when the "reuse" behaviour is used and an error occurs the behaviour is inconsistent. Some functions behaved as described here, while some did not free ***a** on error and did not set ***a** to NULL.

As a result of the above issues the "reuse" behaviour is strongly discouraged.

i2d_TYPE() 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_TYPE()**. This may be fixed in future so code should not assume that **i2d_TYPE()** will always succeed.

Any function which encodes a structure (**i2d_TYPE()**, **i2d_TYPE()** or **i2d_TYPE()**) may return a stale encoding if the structure has been modified after deserialization or previous serialization. This is because some objects cache the encoding for efficiency reasons.

COPYRIGHT