

**NAME**

i2t\_ASN1\_OBJECT, OBJ\_length, OBJ\_get0\_data, OBJ\_nid2obj, OBJ\_nid2ln, OBJ\_nid2sn, OBJ\_obj2nid, OBJ\_txt2nid, OBJ\_ln2nid, OBJ\_sn2nid, OBJ\_cmp, OBJ\_dup, OBJ\_txt2obj, OBJ\_obj2txt, OBJ\_create, OBJ\_cleanup, OBJ\_add\_sigid - ASN1 object utility functions

**SYNOPSIS**

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

```
ASN1_OBJECT *OBJ_nid2obj(int n);
```

```
const char *OBJ_nid2ln(int n);
```

```
const char *OBJ_nid2sn(int n);
```

```
int OBJ_obj2nid(const ASN1_OBJECT *o);
```

```
int OBJ_ln2nid(const char *ln);
```

```
int OBJ_sn2nid(const char *sn);
```

```
int OBJ_txt2nid(const char *s);
```

```
ASN1_OBJECT *OBJ_txt2obj(const char *s, int no_name);
```

```
int OBJ_obj2txt(char *buf, int buf_len, const ASN1_OBJECT *a, int no_name);
```

```
int i2t_ASN1_OBJECT(char *buf, int buf_len, const ASN1_OBJECT *a);
```

```
int OBJ_cmp(const ASN1_OBJECT *a, const ASN1_OBJECT *b);
```

```
ASN1_OBJECT *OBJ_dup(const ASN1_OBJECT *o);
```

```
int OBJ_create(const char *oid, const char *sn, const char *ln);
```

```
size_t OBJ_length(const ASN1_OBJECT *obj);
```

```
const unsigned char *OBJ_get0_data(const ASN1_OBJECT *obj);
```

```
int OBJ_add_sigid(int signid, int dig_id, int pkey_id);
```

The following function has been deprecated since OpenSSL 1.1.0, and can be hidden entirely by defining **OPENSSL\_API\_COMPAT** with a suitable version value, see **openssl\_user\_macros(7)**:

```
void OBJ_cleanup(void);
```

**DESCRIPTION**

The ASN1 object utility functions process ASN1\_OBJECT structures which are a representation of the

ASN1 OBJECT IDENTIFIER (OID) type. For convenience, OIDs are usually represented in source code as numeric identifiers, or **NIDs**. OpenSSL has an internal table of OIDs that are generated when the library is built, and their corresponding NIDs are available as defined constants. For the functions below, application code should treat all returned values -- OIDs, NIDs, or names -- as constants.

**OBJ\_nid2obj()**, **OBJ\_nid2ln()** and **OBJ\_nid2sn()** convert the NID *n* to an ASN1\_OBJECT structure, its long name and its short name respectively, or **NULL** if an error occurred.

**OBJ\_obj2nid()**, **OBJ\_ln2nid()**, **OBJ\_sn2nid()** return the corresponding NID for the object *o*, the long name *ln* or the short name *sn* respectively or NID\_undef if an error occurred.

**OBJ\_txt2nid()** returns NID corresponding to text string *s*. *s* can be a long name, a short name or the numerical representation of an object.

**OBJ\_txt2obj()** converts the text string *s* into an ASN1\_OBJECT structure. If *no\_name* is 0 then long names and short names will be interpreted as well as numerical forms. If *no\_name* is 1 only the numerical form is acceptable.

**OBJ\_obj2txt()** converts the ASN1\_OBJECT *a* into a textual representation. Unless *buf* is NULL, the representation is written as a NUL-terminated string to *buf*, where at most *buf\_len* bytes are written, truncating the result if necessary. In any case it returns the total string length, excluding the NUL character, required for non-truncated representation, or -1 on error. If *no\_name* is 0 then if the object has a long or short name then that will be used, otherwise the numerical form will be used. If *no\_name* is 1 then the numerical form will always be used.

**i2t\_ASN1\_OBJECT()** is the same as **OBJ\_obj2txt()** with the *no\_name* set to zero.

**OBJ\_cmp()** compares *a* to *b*. If the two are identical 0 is returned.

**OBJ\_dup()** returns a copy of *o*.

**OBJ\_create()** adds a new object to the internal table. *oid* is the numerical form of the object, *sn* the short name and *ln* the long name. A new NID is returned for the created object in case of success and NID\_undef in case of failure.

**OBJ\_length()** returns the size of the content octets of *obj*.

**OBJ\_get0\_data()** returns a pointer to the content octets of *obj*. The returned pointer is an internal pointer which **must not** be freed.

**OBJ\_add\_sigid()** creates a new composite "Signature Algorithm" that associates a given NID with two other NIDs - one representing the underlying signature algorithm and the other representing a digest algorithm to be used in conjunction with it. *signid* represents the NID for the composite "Signature Algorithm", *dig\_id* is the NID for the digest algorithm and *pkey\_id* is the NID for the underlying signature algorithm. As there are signature algorithms that do not require a digest, NID\_undef is a valid *dig\_id*.

**OBJ\_cleanup()** releases any resources allocated by creating new objects.

## NOTES

Objects in OpenSSL can have a short name, a long name and a numerical identifier (NID) associated with them. A standard set of objects is represented in an internal table. The appropriate values are defined in the header file **objects.h**.

For example the OID for commonName has the following definitions:

```
#define SN_commonName      "CN"
#define LN_commonName      "commonName"
#define NID_commonName    13
```

New objects can be added by calling **OBJ\_create()**.

Table objects have certain advantages over other objects: for example their NIDs can be used in a C language switch statement. They are also static constant structures which are shared: that is there is only a single constant structure for each table object.

Objects which are not in the table have the NID value NID\_undef.

Objects do not need to be in the internal tables to be processed, the functions **OBJ\_txt2obj()** and **OBJ\_obj2txt()** can process the numerical form of an OID.

Some objects are used to represent algorithms which do not have a corresponding ASN.1 OBJECT IDENTIFIER encoding (for example no OID currently exists for a particular algorithm). As a result they **cannot** be encoded or decoded as part of ASN.1 structures. Applications can determine if there is a corresponding OBJECT IDENTIFIER by checking **OBJ\_length()** is not zero.

These functions cannot return **const** because an **ASN1\_OBJECT** can represent both an internal, constant, OID and a dynamically-created one. The latter cannot be constant because it needs to be freed after use.

**RETURN VALUES**

**OBJ\_nid2obj()** returns an **ASN1\_OBJECT** structure or **NULL** if an error occurred.

**OBJ\_nid2ln()** and **OBJ\_nid2sn()** returns a valid string or **NULL** on error.

**OBJ\_obj2nid()**, **OBJ\_ln2nid()**, **OBJ\_sn2nid()** and **OBJ\_txt2nid()** return a NID or **NID\_undef** on error.

**OBJ\_add\_sigid()** returns 1 on success or 0 on error.

**i2t\_ASN1\_OBJECT()** and **OBJ\_obj2txt()** return -1 on error. On success, they return the length of the string written to *buf* if *buf* is not **NULL** and *buf\_len* is big enough, otherwise the total string length.

Note that this does not count the trailing NUL character.

**EXAMPLES**

Create an object for **commonName**:

```
ASN1_OBJECT *o = OBJ_nid2obj(NID_commonName);
```

Check if an object is **commonName**

```
if (OBJ_obj2nid(obj) == NID_commonName)
    /* Do something */
```

Create a new NID and initialize an object from it:

```
int new_nid = OBJ_create("1.2.3.4", "NewOID", "New Object Identifier");
ASN1_OBJECT *obj = OBJ_nid2obj(new_nid);
```

Create a new object directly:

```
obj = OBJ_txt2obj("1.2.3.4", 1);
```

**BUGS**

Neither **OBJ\_create()** nor **OBJ\_add\_sigid()** do any locking and are thus not thread safe. Moreover, none of the other functions should be called while concurrent calls to these two functions are possible.

**SEE ALSO**

**ERR\_get\_error(3)**

**HISTORY**

**OBJ\_cleanup()** was deprecated in OpenSSL 1.1.0 by **OPENSSL\_init\_crypto(3)** and should not be used.

## **COPYRIGHT**

Copyright 2002-2021 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>.