NAME

ASN1_generate_nconf, ASN1_generate_v3 - ASN1 string generation functions

SYNOPSIS

```
 #include <openssl/asn1.h>

 ASN1_TYPE *ASN1_generate_nconf(const char *str, CONF *nconf);
 ASN1_TYPE *ASN1_generate_v3(const char *str, X509V3_CTX *cnf);
```

DESCRIPTION

These functions generate the ASN1 encoding of a string in an **ASN1_TYPE** structure.

str contains the string to encode. *nconf* or *cnf* contains the optional configuration information where additional strings will be read from. *nconf* will typically come from a config file whereas *cnf* is obtained from an **X509V3_CTX** structure, which will typically be used by X509 v3 certificate extension functions. *cnf* or *nconf* can be set to NULL if no additional configuration will be used.

GENERATION STRING FORMAT

The actual data encoded is determined by the string *str* and the configuration information. The general format of the string is:

[*modifier*,]*type*[:*value*]

That is zero or more comma separated modifiers followed by a type followed by an optional colon and a value. The formats of *type*, *value* and *modifier* are explained below.

Supported Types

The supported types are listed below. Case is not significant in the type names. Unless otherwise specified only the **ASCII** format is permissible.

BOOLEAN, **BOOL**

This encodes a boolean type. The *value* string is mandatory and should be **TRUE** or **FALSE**. Additionally **TRUE**, **true**, **Y**, **y**, **YES**, **yes**, **FALSE**, **false**, **N**, **n**, **NO** and **no** are acceptable.

NULL

Encode the **NULL** type, the *value* string must not be present.

INTEGER, **INT**

Encodes an ASN1 **INTEGER** type. The *value* string represents the value of the integer, it can be prefaced by a minus sign and is normally interpreted as a decimal value unless the prefix **0x** is included.

ENUMERATED, **ENUM**

Encodes the ASN1 **ENUMERATED** type, it is otherwise identical to **INTEGER**.

OBJECT, **OID**

Encodes an ASN1 **OBJECT IDENTIFIER**, the *value* string can be a short name, a long name or numerical format.

UTCTIME, **UTC**

Encodes an ASN1 **UTCTime** structure, the value should be in the format **YYMMDDHHMMSSZ**.

GENERALIZEDTIME, **GENTIME**

Encodes an ASN1 **GeneralizedTime** structure, the value should be in the format **YYYYMMDDHHMMSSZ**.

OCTETSTRING, **OCT**

Encodes an ASN1 **OCTET STRING**. *value* represents the contents of this structure, the format strings **ASCII** and **HEX** can be used to specify the format of *value*.

BITSTRING, **BITSTR**

Encodes an ASN1 **BIT STRING**. *value* represents the contents of this structure, the format strings **ASCII**, **HEX** and **BITLIST** can be used to specify the format of *value*.

If the format is anything other than **BITLIST** the number of unused bits is set to zero.

UNIVERSALSTRING, **UNIV**, **IA5**, **IA5STRING**, **UTF8**, **UTF8String**, **BMP**, **BMPSTRING**, **VISIBLESTRING**, **VISIBLE**, **PRINTABLESTRING**, **PRINTABLE**, **T61**, **T61STRING**, **TELETEXSTRING**, **GeneralString**, **NUMERICSTRING**, **NUMERIC**

These encode the corresponding string types. *value* represents the contents of this structure. The format can be **ASCII** or **UTF8**.

SEQUENCE, **SEQ**, **SET**

Formats the result as an ASN1 **SEQUENCE** or **SET** type. *value* should be a section name which will contain the contents. The field names in the section are ignored and the values are in the generated string format. If *value* is absent then an empty SEQUENCE will be encoded.

Modifiers

Modifiers affect the following structure, they can be used to add EXPLICIT or IMPLICIT tagging, add wrappers or to change the string format of the final type and value. The supported formats are documented below.

EXPLICIT, **EXP**

Add an explicit tag to the following structure. This string should be followed by a colon and the tag value to use as a decimal value.

By following the number with **U**, **A**, **P** or **C** UNIVERSAL, APPLICATION, PRIVATE or CONTEXT SPECIFIC tagging can be used, the default is CONTEXT SPECIFIC.

IMPLICIT, **IMP**

This is the same as **EXPLICIT** except IMPLICIT tagging is used instead.

OCTWRAP, **SEQWRAP**, **SETWRAP**, **BITWRAP**

The following structure is surrounded by an OCTET STRING, a SEQUENCE, a SET or a BIT STRING respectively. For a BIT STRING the number of unused bits is set to zero.

FORMAT

This specifies the format of the ultimate value. It should be followed by a colon and one of the strings **ASCII**, **UTF8**, **HEX** or **BITLIST**.

If no format specifier is included then **ASCII** is used. If **UTF8** is specified then the value string must be a valid **UTF8** string. For **HEX** the output must be a set of hex digits. **BITLIST** (which is only valid for a BIT STRING) is a comma separated list of the indices of the set bits, all other bits are zero.

RETURN VALUES

ASN1_generate_nconf() and **ASN1_generate_v3()** return the encoded data as an **ASN1_TYPE** structure or NULL if an error occurred.

The error codes that can be obtained by **ERR_get_error**(3).

EXAMPLES

A simple IA5String:

```
 IA5STRING:Hello World
```

An IA5String explicitly tagged:

```
 EXPLICIT:0,IA5STRING:Hello World
```

An IA5String explicitly tagged using APPLICATION tagging:

```
EXPLICIT:0A,IA5STRING:Hello World
```

A BITSTRING with bits 1 and 5 set and all others zero:

```
FORMAT:BITLIST,BITSTRING:1,5
```

A more complex example using a config file to produce a SEQUENCE consisting of a BOOL an OID and a UTF8String:

```
asn1 = SEQUENCE:seq_section

[seq_section]

field1 = BOOLEAN:TRUE
field2 = OID:commonName
field3 = UTF8:Third field
```

This example produces an RSAPrivateKey structure, this is the key contained in the file client.pem in all OpenSSL distributions (note: the field names such as ’coeff’ are ignored and are present just for clarity):

```
asn1=SEQUENCE:private_key
[private_key]
version=INTEGER:0

n=INTEGER:0xBB6FE79432CC6EA2D8F970675A5A87BFBE1AFF0BE63E879F2AFFB93644\
D4D2C6D000430DEC66ABF47829E74B8C5108623A1C0EE8BE217B3AD8D36D5EB4FCA1D9

e=INTEGER:0x010001

d=INTEGER:0x6F05EAD2F27FFAEC84BEC360C4B928FD5F3A9865D0FCAAD291E2A52F4A\
F810DC6373278C006A0ABBA27DC8C63BF97F7E666E27C5284D7D3B1FFFE16B7A87B51D

p=INTEGER:0xF3929B9435608F8A22C208D86795271D54EBDFB09DDEF539AB083DA912\
D4BD57

q=INTEGER:0xC50016F89DFF2561347ED1186A46E150E28BF2D0F539A1594BBD7FE467\
46EC4F
```

```
exp1=INTEGER:0x9E7D4326C924AFC1DEA40B45650134966D6F9DFA3A7F9D698CD4ABEA\
9C0A39B9

exp2=INTEGER:0xBA84003BB95355AFB7C50DF140C60513D0BA51D637272E355E397779\
E7B2458F

coeff=INTEGER:0x30B9E4F2AFA5AC679F920FC83F1F2DF1BAF1779CF989447FABC2F5\
628657053A
```

This example is the corresponding public key in a SubjectPublicKeyInfo structure:

```
# Start with a SEQUENCE
asn1=SEQUENCE:pubkeyinfo

# pubkeyinfo contains an algorithm identifier and the public key wrapped
# in a BIT STRING
[pubkeyinfo]
algorithm=SEQUENCE:rsa_alg
pubkey=BITWRAP,SEQUENCE:rsapubkey

# algorithm ID for RSA is just an OID and a NULL
[rsa_alg]
algorithm=OID:rsaEncryption
parameter=NULL

# Actual public key: modulus and exponent
[rsapubkey]
n=INTEGER:0xBB6FE79432CC6EA2D8F970675A5A87BFBE1AFF0BE63E879F2AFFB93644\
D4D2C6D000430DEC66ABF47829E74B8C5108623A1C0EE8BE217B3AD8D36D5EB4FCA1D9

e=INTEGER:0x010001
```

SEE ALSO

ERR_get_error(3)

COPYRIGHT