

# **GNU Libtasn1 API Reference Manual**

---

**COLLABORATORS**

	<i>TITLE :</i> GNU Libtasn1 API Reference Manual		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		December 6, 2010	

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>GNU Libtasn1 API Reference Manual</b>	<b>1</b>
1.1	libtasn1 . . . . .	1
<b>2</b>	<b>Index</b>	<b>22</b>

## Chapter 1

# GNU Libtasn1 API Reference Manual

This document describes the GNU Libtasn1 library developed for ASN.1 (Abstract Syntax Notation One) structures management and DER (Distinguished Encoding Rules) encoding functions.

More up to date information can be found at <http://www.gnu.org/software/libtasn1/>.

### 1.1 libtasn1

libtasn1 —

#### Synopsis

```
#define          ASN1_API
#define          ASN1_VERSION
typedef         asn1_retCode;
#define          ASN1_SUCCESS
#define          ASN1_FILE_NOT_FOUND
#define          ASN1_ELEMENT_NOT_FOUND
#define          ASN1_IDENTIFIER_NOT_FOUND
#define          ASN1_DER_ERROR
#define          ASN1_VALUE_NOT_FOUND
#define          ASN1_GENERIC_ERROR
#define          ASN1_VALUE_NOT_VALID
#define          ASN1_TAG_ERROR
#define          ASN1_TAG_IMPLICIT
#define          ASN1_ERROR_TYPE_ANY
#define          ASN1_SYNTAX_ERROR
#define          ASN1_MEM_ERROR
#define          ASN1_MEM_ALLOC_ERROR
#define          ASN1_DER_OVERFLOW
#define          ASN1_NAME_TOO_LONG
#define          ASN1_ARRAY_ERROR
#define          ASN1_ELEMENT_NOT_EMPTY
#define          ASN1_PRINT_NAME
#define          ASN1_PRINT_NAME_TYPE
#define          ASN1_PRINT_NAME_TYPE_VALUE
#define          ASN1_PRINT_ALL
#define          ASN1_CLASS_UNIVERSAL
#define          ASN1_CLASS_APPLICATION
```

```
#define ASN1_CLASS_CONTEXT_SPECIFIC
#define ASN1_CLASS_PRIVATE
#define ASN1_CLASS_STRUCTURED
#define ASN1_TAG_BOOLEAN
#define ASN1_TAG_INTEGER
#define ASN1_TAG_SEQUENCE
#define ASN1_TAG_SET
#define ASN1_TAG_OCTET_STRING
#define ASN1_TAG_BIT_STRING
#define ASN1_TAG_UTCTime
#define ASN1_TAG_GENERALIZEDTime
#define ASN1_TAG_OBJECT_ID
#define ASN1_TAG_ENUMERATED
#define ASN1_TAG_NULL
#define ASN1_TAG_GENERALSTRING
typedef node_asn;
typedef ASN1_TYPE;
#define ASN1_TYPE_EMPTY
typedef ASN1_ARRAY_TYPE;
#define ASN1_MAX_NAME_SIZE
#define ASN1_MAX_ERROR_DESCRIPTION_SIZE
asn1_retCode asn1_parser2tree (const char *file_name,
                                ASN1_TYPE *definitions,
                                char *errorDescription);
asn1_retCode asn1_parser2array (const char *inputFileName,
                                 const char *outputFileName,
                                 const char *vectorName,
                                 char *errorDescription);
asn1_retCode asn1_array2tree (const ASN1_ARRAY_TYPE *array,
                               ASN1_TYPE *definitions,
                               char *errorDescription);
void asn1_print_structure (FILE *out,
                           ASN1_TYPE structure,
                           const char *name,
                           int mode);
asn1_retCode asn1_create_element (ASN1_TYPE definitions,
                                  const char *source_name,
                                  ASN1_TYPE *element);
asn1_retCode asn1_delete_structure (ASN1_TYPE *structure);
asn1_retCode asn1_delete_element (ASN1_TYPE structure,
                                   const char *element_name);
asn1_retCode asn1_write_value (ASN1_TYPE node_root,
                                const char *name,
                                const void *ivalue,
                                int len);
asn1_retCode asn1_read_value (ASN1_TYPE root,
                              const char *name,
                              void *ivalue,
                              int *len);
asn1_retCode asn1_number_of_elements (ASN1_TYPE element,
                                       const char *name,
                                       int *num);
asn1_retCode asn1_der_coding (ASN1_TYPE element,
                              const char *name,
                              void *ider,
                              int *len,
                              char *ErrorDescription);
```

---

asn1_retCode	asn1_der_decoding	(ASN1_TYPE *element, const void *ider, int len, char *errorDescription);
asn1_retCode	asn1_der_decoding_element	(ASN1_TYPE *structure, const char *elementName, const void *ider, int len, char *errorDescription);
asn1_retCode	asn1_der_decoding_startEnd	(ASN1_TYPE element, const void *ider, int len, const char *name_element, int *start, int *end);
asn1_retCode	asn1_expand_any_defined_by	(ASN1_TYPE definitions, ASN1_TYPE *element);
asn1_retCode	asn1_expand_octet_string	(ASN1_TYPE definitions, ASN1_TYPE *element, const char *octetName, const char *objectName);
asn1_retCode	asn1_read_tag	(ASN1_TYPE root, const char *name, int *tagValue, int *classValue);
const char *	asn1_find_structure_from_oid	(ASN1_TYPE definitions, const char *oidValue);
const char *	asn1_check_version	(const char *req_version);
const char *	asn1_strerror	(asn1_retCode error);
void	asn1_perror	(asn1_retCode error);
int	asn1_get_tag_der	(unsigned char *der, int der_len, unsigned char *cls, int *len, unsigned long *tag);
void	asn1_octet_der	(unsigned char *str, int str_len, unsigned char *der, int *der_len);
asn1_retCode	asn1_get_octet_der	(unsigned char *der, int der_len, int *ret_len, unsigned char *str, int str_size, int *str_len);
void	asn1_bit_der	(unsigned char *str, int bit_len, unsigned char *der, int *der_len);
asn1_retCode	asn1_get_bit_der	(unsigned char *der, int der_len, int *ret_len, unsigned char *str, int str_size, int *bit_len);
signed long	asn1_get_length_der	(unsigned char *der, int der_len,

---

signed long	asn1_get_length_ber	int *len); (unsigned char *ber, int ber_len, int *len);
void	asn1_length_der	(unsigned long int len, unsigned char *ans, int *ans_len);
ASN1_TYPE	asn1_find_node	(ASN1_TYPE pointer, const char *name);
asn1_retCode	asn1_copy_node	(ASN1_TYPE dst, const char *dst_name, ASN1_TYPE src, const char *src_name);
#define	LIBTASN1_VERSION	
#define	MAX_NAME_SIZE	
#define	MAX_ERROR_DESCRIPTION_SIZE	
const char *	libtasn1_strerror	(asn1_retCode error);
void	libtasn1_perror	(asn1_retCode error);

## Description

## Details

### ASN1\_API

```
#define ASN1_API
```

### ASN1\_VERSION

```
#define ASN1_VERSION "2.9"
```

### asn1\_retCode

```
typedef int asn1_retCode; /* type returned by libtasn1 functions */
```

### ASN1\_SUCCESS

```
#define ASN1_SUCCESS 0
```

### ASN1\_FILE\_NOT\_FOUND

```
#define ASN1_FILE_NOT_FOUND 1
```

### ASN1\_ELEMENT\_NOT\_FOUND

```
#define ASN1_ELEMENT_NOT_FOUND 2
```

**ASN1\_IDENTIFIER\_NOT\_FOUND**

```
#define ASN1_IDENTIFIER_NOT_FOUND~3
```

**ASN1\_DER\_ERROR**

```
#define ASN1_DER_ERROR 4
```

**ASN1\_VALUE\_NOT\_FOUND**

```
#define ASN1_VALUE_NOT_FOUND 5
```

**ASN1\_GENERIC\_ERROR**

```
#define ASN1_GENERIC_ERROR 6
```

**ASN1\_VALUE\_NOT\_VALID**

```
#define ASN1_VALUE_NOT_VALID 7
```

**ASN1\_TAG\_ERROR**

```
#define ASN1_TAG_ERROR 8
```

**ASN1\_TAG\_IMPLICIT**

```
#define ASN1_TAG_IMPLICIT 9
```

**ASN1\_ERROR\_TYPE\_ANY**

```
#define ASN1_ERROR_TYPE_ANY 10
```

**ASN1\_SYNTAX\_ERROR**

```
#define ASN1_SYNTAX_ERROR 11
```

**ASN1\_MEM\_ERROR**

```
#define ASN1_MEM_ERROR 12
```

**ASN1\_MEM\_ALLOC\_ERROR**

```
#define ASN1_MEM_ALLOC_ERROR 13
```

---



**ASN1\_DER\_OVERFLOW**

```
#define ASN1_DER_OVERFLOW 14
```

**ASN1\_NAME\_TOO\_LONG**

```
#define ASN1_NAME_TOO_LONG 15
```

**ASN1\_ARRAY\_ERROR**

```
#define ASN1_ARRAY_ERROR 16
```

**ASN1\_ELEMENT\_NOT\_EMPTY**

```
#define ASN1_ELEMENT_NOT_EMPTY 17
```

**ASN1\_PRINT\_NAME**

```
#define ASN1_PRINT_NAME 1
```

**ASN1\_PRINT\_NAME\_TYPE**

```
#define ASN1_PRINT_NAME_TYPE 2
```

**ASN1\_PRINT\_NAME\_TYPE\_VALUE**

```
#define ASN1_PRINT_NAME_TYPE_VALUE~3
```

**ASN1\_PRINT\_ALL**

```
#define ASN1_PRINT_ALL 4
```

**ASN1\_CLASS\_UNIVERSAL**

```
#define ASN1_CLASS_UNIVERSAL 0x00~/* old: 1 */
```

**ASN1\_CLASS\_APPLICATION**

```
#define ASN1_CLASS_APPLICATION 0x40~/* old: 2 */
```

**ASN1\_CLASS\_CONTEXT\_SPECIFIC**

```
#define ASN1_CLASS_CONTEXT_SPECIFIC~0x80~/* old: 3 */
```

---

**ASN1\_CLASS\_PRIVATE**

```
#define ASN1_CLASS_PRIVATE    0xC0~/* old: 4 */
```

**ASN1\_CLASS\_STRUCTURED**

```
#define ASN1_CLASS_STRUCTURED    0x20
```

**ASN1\_TAG\_BOOLEAN**

```
#define ASN1_TAG_BOOLEAN    0x01
```

**ASN1\_TAG\_INTEGER**

```
#define ASN1_TAG_INTEGER    0x02
```

**ASN1\_TAG\_SEQUENCE**

```
#define ASN1_TAG_SEQUENCE    0x10
```

**ASN1\_TAG\_SET**

```
#define ASN1_TAG_SET    0x11
```

**ASN1\_TAG\_OCTET\_STRING**

```
#define ASN1_TAG_OCTET_STRING    0x04
```

**ASN1\_TAG\_BIT\_STRING**

```
#define ASN1_TAG_BIT_STRING    0x03
```

**ASN1\_TAG\_UTCTime**

```
#define ASN1_TAG_UTCTime    0x17
```

**ASN1\_TAG\_GENERALIZEDTime**

```
#define ASN1_TAG_GENERALIZEDTime~0x18
```

**ASN1\_TAG\_OBJECT\_ID**

```
#define ASN1_TAG_OBJECT_ID    0x06
```

**ASN1\_TAG\_ENUMERATED**

```
#define ASN1_TAG_ENUMERATED    0x0A
```

**ASN1\_TAG\_NULL**

```
#define ASN1_TAG_NULL          0x05
```

**ASN1\_TAG\_GENERALSTRING**

```
#define ASN1_TAG_GENERALSTRING  0x1B
```

**node\_asn**

```
typedef struct node_asn_struct node_asn;
```

**ASN1\_TYPE**

```
typedef node_asn *ASN1_TYPE;
```

**ASN1\_TYPE\_EMPTY**

```
#define ASN1_TYPE_EMPTY    NULL
```

**ASN1\_ARRAY\_TYPE**

```
typedef struct static_struct_asn ASN1_ARRAY_TYPE;
```

**ASN1\_MAX\_NAME\_SIZE**

```
#define ASN1_MAX_NAME_SIZE 128
```

**ASN1\_MAX\_ERROR\_DESCRIPTION\_SIZE**

```
#define ASN1_MAX_ERROR_DESCRIPTION_SIZE 128
```

---

**asn1\_parser2tree ()**

```
asn1_retCode      asn1_parser2tree      (const char *file_name,
                                         ASN1_TYPE *definitions,
                                         char *errorDescription);
```

Function used to start the parse algorithm. Creates the structures needed to manage the definitions included in *file\_name* file.

**file\_name** : specify the path and the name of file that contains ASN.1 declarations.

**definitions** : return the pointer to the structure created from "file\_name" ASN.1 declarations.

**errorDescription** : return the error description or an empty string if success.

**Returns** : **ASN1\_SUCCESS** if the file has a correct syntax and every identifier is known, **ASN1\_ELEMENT\_NOT\_EMPTY** if *definitions* not **ASN1\_TYPE\_EMPTY**, **ASN1\_FILE\_NOT\_FOUND** if an error occurred while opening *file\_name*, **ASN1\_SYNTAX\_ERROR** if the syntax is not correct, **ASN1\_IDENTIFIER\_NOT\_FOUND** if in the file there is an identifier that is not defined, **ASN1\_NAME\_TOO\_LONG** if in the file there is an identifier which more than **ASN1\_MAX\_NAME\_SIZE** characters.

**asn1\_parser2array ()**

```
asn1_retCode      asn1_parser2array      (const char *inputFileName,
                                         const char *outputFileName,
                                         const char *vectorName,
                                         char *errorDescription);
```

Function that generates a C structure from an ASN1 file. Creates a file containing a C vector to use to manage the definitions included in *inputFileName* file. If *inputFileName* is "/aa/bb/xx.yy" and *outputFileName* is **NULL**, the file created is "/aa/bb/xx\_asn1\_tab.c". If *vectorName* is **NULL** the vector name will be "xx\_asn1\_tab".

**inputFileName** : specify the path and the name of file that contains ASN.1 declarations.

**outputFileName** : specify the path and the name of file that will contain the C vector definition.

**vectorName** : specify the name of the C vector.

**errorDescription** : return the error description or an empty string if success.

**Returns** : **ASN1\_SUCCESS** if the file has a correct syntax and every identifier is known, **ASN1\_FILE\_NOT\_FOUND** if an error occurred while opening *inputFileName*, **ASN1\_SYNTAX\_ERROR** if the syntax is not correct, **ASN1\_IDENTIFIER\_NOT\_FOUND** if in the file there is an identifier that is not defined, **ASN1\_NAME\_TOO\_LONG** if in the file there is an identifier which more than **ASN1\_MAX\_NAME\_SIZE** characters.

**asn1\_array2tree ()**

```
asn1_retCode      asn1_array2tree      (const ASN1_ARRAY_TYPE *array,
                                         ASN1_TYPE *definitions,
                                         char *errorDescription);
```

Creates the structures needed to manage the ASN.1 definitions. *array* is a vector created by **asn1\_parser2array()**.

**array** : specify the array that contains ASN.1 declarations

**definitions** : return the pointer to the structure created by \*ARRAY ASN.1 declarations

**errorDescription** : return the error description.

**Returns** : **ASN1\_SUCCESS** if structure was created correctly, **ASN1\_ELEMENT\_NOT\_EMPTY** if *\*definitions* not **ASN1\_TYPE\_EMPTY**, **ASN1\_IDENTIFIER\_NOT\_FOUND** if in the file there is an identifier that is not defined (see *errorDescription* for more information), **ASN1\_ARRAY\_ERROR** if the array pointed by *array* is wrong.

**asn1\_print\_structure ()**

```
void                asn1_print_structure                (FILE *out,
                                                         ASN1_TYPE structure,
                                                         const char *name,
                                                         int mode);
```

Prints on the *out* file descriptor the structure's tree starting from the *name* element inside the structure *structure*.

**out** : pointer to the output file (e.g. stdout).

**structure** : pointer to the structure that you want to visit.

**name** : an element of the structure

**mode** : specify how much of the structure to print, can be **ASN1\_PRINT\_NAME**, **ASN1\_PRINT\_NAME\_TYPE**, **ASN1\_PRINT\_NAME\_VALUE** or **ASN1\_PRINT\_ALL**.

**asn1\_create\_element ()**

```
asn1_retCode        asn1_create_element                (ASN1_TYPE definitions,
                                                         const char *source_name,
                                                         ASN1_TYPE *element);
```

Creates a structure of type *source\_name*. Example using "pkix.asn":

```
rc = asn1_create_element(cert_def, "PKIX1.Certificate", certptr);
```

**definitions** : pointer to the structure returned by "parser\_asn1" function

**source\_name** : the name of the type of the new structure (must be inside p\_structure).

**element** : pointer to the structure created.

**Returns** : **ASN1\_SUCCESS** if creation OK, **ASN1\_ELEMENT\_NOT\_FOUND** if *source\_name* is not known.

**asn1\_delete\_structure ()**

```
asn1_retCode        asn1_delete_structure                (ASN1_TYPE *structure);
```

Deletes the structure *\*structure*. At the end, *\*structure* is set to **ASN1\_TYPE\_EMPTY**.

**structure** : pointer to the structure that you want to delete.

**Returns** : **ASN1\_SUCCESS** if successful, **ASN1\_ELEMENT\_NOT\_FOUND** if *\*structure* was **ASN1\_TYPE\_EMPTY**.

**asn1\_delete\_element ()**

```
asn1_retCode        asn1_delete_element                (ASN1_TYPE structure,
                                                         const char *element_name);
```

Deletes the element named *\*element\_name* inside *\*structure*.

**structure** : pointer to the structure that contains the element you want to delete.

**element\_name** : element's name you want to delete.

**Returns** : **ASN1\_SUCCESS** if successful, **ASN1\_ELEMENT\_NOT\_FOUND** if the *element\_name* was not found.

**asn1\_write\_value ()**

```
asn1_retCode      asn1_write_value      (ASN1_TYPE node_root,
                                         const char *name,
                                         const void *ivalue,
                                         int len);
```

Set the value of one element inside a structure.

If an element is OPTIONAL and you want to delete it, you must use the value=NULL and len=0. Using "pkix.asn":

```
result=asn1_write_value(cert, "tbsCertificate.issuerUniqueID", NULL, 0);
```

Description for each type:

INTEGER: VALUE must contain a two's complement form integer.

value[0]=0xFF, len=1 -> integer=-1. value[0]=0xFF value[1]=0xFF, len=2 -> integer=-1. value[0]=0x01, len=1 -> integer= 1. value[0]=0x00 value[1]=0x01, len=2 -> integer= 1. value="123", len=0 -> integer= 123.

ENUMERATED: As INTEGER (but only with not negative numbers).

BOOLEAN: VALUE must be the null terminated string "TRUE" or "FALSE" and LEN != 0.

value="TRUE", len=1 -> boolean=TRUE. value="FALSE", len=1 -> boolean=FALSE.

OBJECT IDENTIFIER: VALUE must be a null terminated string with each number separated by a dot (e.g. "1.2.3.543.1"). LEN != 0.

value="1 2 840 10040 4 3", len=1 -> OID=dsa-with-sha.

UTCTime: VALUE must be a null terminated string in one of these formats: "YYMMDDhhmmssZ", "YYMMDDhhmmssZ", "YYMMDDhhmmss+hh'mm'", "YYMMDDhhmmss-hh'mm'", "YYMMDDhhmm+hh'mm'", or "YYMMDDhhmm-hh'mm'". LEN != 0.

value="9801011200Z", len=1 -> time=January 1st, 1998 at 12h 00m Greenwich Mean Time

GeneralizedTime: VALUE must be in one of this format: "YYYYMMDDhhmmss.sZ", "YYYYMMDDhhmmss.sZ", "YYYYMMDDhhmmss.s+hh'mm'", "YYYYMMDDhhmmss.s-hh'mm'", "YYYYMMDDhhmm+hh'mm'", or "YYYYMMDDhhmm-hh'mm'". where ss.s indicates the seconds with any precision like "10.1" or "01.02". LEN != 0

value="2001010112001.12-0700", len=1 -> time=January 1st, 2001 at 12h 00m 01.12s Pacific Daylight Time

OCTET STRING: VALUE contains the octet string and LEN is the number of octets.

value="\backslashx01\backslashx02\backslashx03", len=3 -> three bytes octet string

GeneralString: VALUE contains the generalstring and LEN is the number of octets.

value="\backslashx01\backslashx02\backslashx03", len=3 -> three bytes generalstring

BIT STRING: VALUE contains the bit string organized by bytes and LEN is the number of bits.

value="\backslashxCF", len=6 -> bit string="110011" (six bits)

CHOICE: if NAME indicates a choice type, VALUE must specify one of the alternatives with a null terminated string. LEN != 0. Using "pkix.asn":

```
result=asn1_write_value(cert, "certificate1.tbsCertificate.subject", "rdnSequence", 1);
```

ANY: VALUE indicates the der encoding of a structure. LEN != 0.

SEQUENCE OF: VALUE must be the null terminated string "NEW" and LEN != 0. With this instruction another element is appended in the sequence. The name of this element will be "?1" if it's the first one, "?2" for the second and so on.

Using "pkix.asn":

```
result=asn1_write_value(cert, "certificate1.tbsCertificate.subject.rdnSequence", "NEW", 1);
```

SET OF: the same as SEQUENCE OF. Using "pkix.asn":

```
result=asn1_write_value(cert, "tbsCertificate.subject.rdnSequence.?LAST", "NEW", 1);
```

**node\_root** : pointer to a structure

**name** : the name of the element inside the structure that you want to set.

**ivalue** : vector used to specify the value to set. If len is >0, VALUE must be a two's complement form integer. if len=0 \*VALUE must be a null terminated string with an integer value.

**len** : number of bytes of \*value to use to set the value: value[0]..value[len-1] or 0 if value is a null terminated string

**Returns** : **ASN1\_SUCCESS** if the value was set, **ASN1\_ELEMENT\_NOT\_FOUND** if *name* is not a valid element, and **ASN1\_VALUE\_NO** if *ivalue* has a wrong format.

### asn1\_read\_value ()

```
asn1_retCode      asn1_read_value      (ASN1_TYPE root,
                                         const char *name,
                                         void *ivalue,
                                         int *len);
```

Returns the value of one element inside a structure.

If an element is OPTIONAL and the function "read\_value" returns **ASN1\_ELEMENT\_NOT\_FOUND**, it means that this element wasn't present in the der encoding that created the structure. The first element of a SEQUENCE\_OF or SET\_OF is named "?1". The second one "?2" and so on.

INTEGER: VALUE will contain a two's complement form integer.

integer=-1 -> value[0]=0xFF , len=1. integer=1 -> value[0]=0x01 , len=1.

ENUMERATED: As INTEGER (but only with not negative numbers).

BOOLEAN: VALUE will be the null terminated string "TRUE" or "FALSE" and LEN=5 or LEN=6.

OBJECT IDENTIFIER: VALUE will be a null terminated string with each number separated by a dot (i.e. "1.2.3.543.1").

LEN = strlen(VALUE)+1

UTCTime: VALUE will be a null terminated string in one of these formats: "YYMMDDhhmmss+hh'mm'" or "YYMMDDhhmmss-hh'mm'". LEN=strlen(VALUE)+1.

GeneralizedTime: VALUE will be a null terminated string in the same format used to set the value.

OCTET STRING: VALUE will contain the octet string and LEN will be the number of octets.

GeneralString: VALUE will contain the generalstring and LEN will be the number of octets.

BIT STRING: VALUE will contain the bit string organized by bytes and LEN will be the number of bits.

CHOICE: If NAME indicates a choice type, VALUE will specify the alternative selected.

ANY: If NAME indicates an any type, VALUE will indicate the DER encoding of the structure actually used.

**root** : pointer to a structure.

**name** : the name of the element inside a structure that you want to read.

**ivalue** : vector that will contain the element's content, must be a pointer to memory cells already allocated.

**len** : number of bytes of \*value: value[0]..value[len-1]. Initially holds the sizeof value.

**Returns** : **ASN1\_SUCCESS** if value is returned, **ASN1\_ELEMENT\_NOT\_FOUND** if *name* is not a valid element, **ASN1\_VALUE\_NO** if there isn't any value for the element selected, and **ASN1\_MEM\_ERROR** if The value vector isn't big enough to store the result, and in this case *len* will contain the number of bytes needed.

**asn1\_number\_of\_elements ()**

```
asn1_retCode      asn1_number_of_elements      (ASN1_TYPE element,
                                                const char *name,
                                                int *num);
```

Counts the number of elements of a sub-structure called NAME with names equal to "?1", "?2", ...

**element** : pointer to the root of an ASN1 structure.

**name** : the name of a sub-structure of ROOT.

**num** : pointer to an integer where the result will be stored

**Returns** : **ASN1\_SUCCESS** if successful, **ASN1\_ELEMENT\_NOT\_FOUND** if *name* is not known, **ASN1\_GENERIC\_ERROR** if pointer *num* is **NULL**.

**asn1\_der\_coding ()**

```
asn1_retCode      asn1_der_coding              (ASN1_TYPE element,
                                                const char *name,
                                                void *ider,
                                                int *len,
                                                char *ErrorDescription);
```

Creates the DER encoding for the NAME structure (inside \*POINTER structure).

**element** : pointer to an ASN1 element

**name** : the name of the structure you want to encode (it must be inside \*POINTER).

**ider** : vector that will contain the DER encoding. DER must be a pointer to memory cells already allocated.

**len** : number of bytes of \*ider: *ider*[0]..*ider*[len-1], Initially holds the sizeof of der vector.

**ErrorDescription** : return the error description or an empty string if success.

**Returns** : **ASN1\_SUCCESS** if DER encoding OK, **ASN1\_ELEMENT\_NOT\_FOUND** if *name* is not a valid element, **ASN1\_VALUE\_N** if there is an element without a value, **ASN1\_MEM\_ERROR** if the *ider* vector isn't big enough and in this case *len* will contain the length needed.

**asn1\_der\_decoding ()**

```
asn1_retCode      asn1_der_decoding            (ASN1_TYPE *element,
                                                const void *ider,
                                                int len,
                                                char *errorDescription);
```

Fill the structure \*ELEMENT with values of a DER encoding string. The structure must just be created with function **asn1\_create\_element()**. If an error occurs during the decoding procedure, the \*ELEMENT is deleted and set equal to **ASN1\_TYPE\_EMPTY**.

**element** : pointer to an ASN1 structure.

**ider** : vector that contains the DER encoding.

**len** : number of bytes of \*ider: *ider*[0]..*ider*[len-1].

**errorDescription** : null-terminated string contains details when an error occurred.

**Returns** : **ASN1\_SUCCESS** if DER encoding OK, **ASN1\_ELEMENT\_NOT\_FOUND** if ELEMENT is **ASN1\_TYPE\_EMPTY**, and **ASN1\_TAG\_ERROR** or **ASN1\_DER\_ERROR** if the der encoding doesn't match the structure name (\*ELEMENT deleted).



**asn1\_der\_decoding\_element ()**

```
asn1_retCode      asn1_der_decoding_element      (ASN1_TYPE *structure,
                                                  const char *elementName,
                                                  const void *ider,
                                                  int len,
                                                  char *errorDescription);
```

Fill the element named *ELEMENTNAME* with values of a DER encoding string. The structure must just be created with function **asn1\_create\_element()**. The DER vector must contain the encoding string of the whole *STRUCTURE*. If an error occurs during the decoding procedure, the *\*STRUCTURE* is deleted and set equal to **ASN1\_TYPE\_EMPTY**.

**structure** : pointer to an ASN1 structure

**elementName** : name of the element to fill

**ider** : vector that contains the DER encoding of the whole structure.

**len** : number of bytes of *\*der*: *der*[0]..*der*[len-1]

**errorDescription** : null-terminated string contains details when an error occurred.

**Returns** : **ASN1\_SUCCESS** if DER encoding OK, **ASN1\_ELEMENT\_NOT\_FOUND** if ELEMENT is **ASN1\_TYPE\_EMPTY** or *elementName* == NULL, and **ASN1\_TAG\_ERROR** or **ASN1\_DER\_ERROR** if the der encoding doesn't match the structure *structure* (\*ELEMENT deleted).

**asn1\_der\_decoding\_startEnd ()**

```
asn1_retCode      asn1_der_decoding_startEnd      (ASN1_TYPE element,
                                                  const void *ider,
                                                  int len,
                                                  const char *name_element,
                                                  int *start,
                                                  int *end);
```

Find the start and end point of an element in a DER encoding string. I mean that if you have a der encoding and you have already used the function **asn1\_der\_decoding()** to fill a structure, it may happen that you want to find the piece of string concerning an element of the structure.

One example is the sequence "tbsCertificate" inside an X509 certificate.

**element** : pointer to an ASN1 element

**ider** : vector that contains the DER encoding.

**len** : number of bytes of *\*ider*: *ider*[0]..*ider*[len-1]

**name\_element** : an element of NAME structure.

**start** : the position of the first byte of NAME\_ELEMENT decoding (*ider*[\*start])

**end** : the position of the last byte of NAME\_ELEMENT decoding (*ider*[\*end])

**Returns** : **ASN1\_SUCCESS** if DER encoding OK, **ASN1\_ELEMENT\_NOT\_FOUND** if ELEMENT is **ASN1\_TYPE\_EMPTY** or *name\_element* is not a valid element, **ASN1\_TAG\_ERROR** or **ASN1\_DER\_ERROR** if the der encoding doesn't match the structure ELEMENT.

**asn1\_expand\_any\_defined\_by ()**

```
asn1_retCode      asn1_expand_any_defined_by      (ASN1_TYPE definitions,
                                                  ASN1_TYPE *element);
```

Expands every "ANY DEFINED BY" element of a structure created from a DER decoding process (`asn1_der_decoding` function). The element ANY must be defined by an OBJECT IDENTIFIER. The type used to expand the element ANY is the first one following the definition of the actual value of the OBJECT IDENTIFIER.

**definitions** : ASN1 definitions

**element** : pointer to an ASN1 structure

**Returns** : **ASN1\_SUCCESS** if Substitution OK, **ASN1\_ERROR\_TYPE\_ANY** if some "ANY DEFINED BY" element couldn't be expanded due to a problem in OBJECT\_ID -> TYPE association, or other error codes depending on DER decoding.

**asn1\_expand\_octet\_string ()**

```
asn1_retCode      asn1_expand_octet_string      (ASN1_TYPE definitions,
                                                  ASN1_TYPE *element,
                                                  const char *octetName,
                                                  const char *objectName);
```

Expands an "OCTET STRING" element of a structure created from a DER decoding process (the `asn1_der_decoding()` function). The type used for expansion is the first one following the definition of the actual value of the OBJECT IDENTIFIER indicated by OBJECTNAME.

**definitions** : ASN1 definitions

**element** : pointer to an ASN1 structure

**octetName** : name of the OCTET STRING field to expand.

**objectName** : name of the OBJECT IDENTIFIER field to use to define the type for expansion.

**Returns** : **ASN1\_SUCCESS** if substitution OK, **ASN1\_ELEMENT\_NOT\_FOUND** if *objectName* or *octetName* are not correct, **ASN1\_VALUE\_NOT\_VALID** if it wasn't possible to find the type to use for expansion, or other errors depending on DER decoding.

**asn1\_read\_tag ()**

```
asn1_retCode      asn1_read_tag      (ASN1_TYPE root,
                                       const char *name,
                                       int *tagValue,
                                       int *classValue);
```

Returns the TAG and the CLASS of one element inside a structure. CLASS can have one of these constants: **ASN1\_CLASS\_APPLICATION**, **ASN1\_CLASS\_UNIVERSAL**, **ASN1\_CLASS\_PRIVATE** or **ASN1\_CLASS\_CONTEXT\_SPECIFIC**.

**root** : pointer to a structure

**name** : the name of the element inside a structure.

**tagValue** : variable that will contain the TAG value.

**classValue** : variable that will specify the TAG type.

**Returns** : **ASN1\_SUCCESS** if successful, **ASN1\_ELEMENT\_NOT\_FOUND** if *name* is not a valid element.

**asn1\_find\_structure\_from\_oid ()**

```
const char *      asn1_find_structure_from_oid      (ASN1_TYPE definitions,  
                                                    const char *oidValue);
```

Search the structure that is defined just after an OID definition.

**definitions** : ASN1 definitions

**oidValue** : value of the OID to search (e.g. "1.2.3.4").

**Returns** : **NULL** when *oidValue* not found, otherwise the pointer to a constant string that contains the element name defined just after the OID.

**asn1\_check\_version ()**

```
const char *      asn1_check_version              (const char *req_version);
```

Check that the version of the library is at minimum the requested one and return the version string; return **NULL** if the condition is not satisfied. If a **NULL** is passed to this function, no check is done, but the version string is simply returned.

See **ASN1\_VERSION** for a suitable *req\_version* string.

**req\_version** : Required version number, or **NULL**.

**Returns** : Version string of run-time library, or **NULL** if the run-time library does not meet the required version number.

**asn1\_strerror ()**

```
const char *      asn1_strerror                  (asn1_retCode error);
```

Returns a string with a description of an error. This function is similar to `strerror`. The only difference is that it accepts an error (number) returned by a libtasn1 function.

This function replaces `libtasn1_strerror()` in older libtasn1.

**error** : is an error returned by a libtasn1 function.

**Returns** : Pointer to static zero-terminated string describing error code.

Since 1.6

**asn1\_perror ()**

```
void              asn1_perror                    (asn1_retCode error);
```

Prints a string to `stderr` with a description of an error. This function is like `perror()`. The only difference is that it accepts an error returned by a libtasn1 function.

This function replaces `libtasn1_perror()` in older libtasn1.

**error** : is an error returned by a libtasn1 function.

Since 1.6

**asn1\_get\_tag\_der ()**

```
int                asn1_get_tag_der                (unsigned char *der,
                                                    int der_len,
                                                    unsigned char *cls,
                                                    int *len,
                                                    unsigned long *tag);
```

Decode the class and TAG from DER code.

**der** : DER data to decode.

**der\_len** : Length of DER data to decode.

**cls** : Output variable containing decoded class.

**len** : Output variable containing the length of the DER TAG data.

**tag** : Output variable containing the decoded tag.

**Returns** : Returns **ASN1\_SUCCESS** on success, or an error.

**asn1\_octet\_der ()**

```
void                asn1_octet_der                (unsigned char *str,
                                                    int str_len,
                                                    unsigned char *der,
                                                    int *der_len);
```

Creates the DER coding for an OCTET type (length included).

**str** : OCTET string.

**str\_len** : STR length (str[0]..str[str\_len-1]).

**der** : string returned.

**der\_len** : number of meaningful bytes of DER (der[0]..der[ans\_len-1]).

**asn1\_get\_octet\_der ()**

```
asn1_retCode        asn1_get_octet_der            (unsigned char *der,
                                                    int der_len,
                                                    int *ret_len,
                                                    unsigned char *str,
                                                    int str_size,
                                                    int *str_len);
```

Extract an OCTET SEQUENCE from DER data.

**der** : DER data to decode containing the OCTET SEQUENCE.

**der\_len** : Length of DER data to decode.

**ret\_len** : Output variable containing the length of the DER data.

**str** : Pre-allocated output buffer to put decoded OCTET SEQUENCE in.

**str\_size** : Length of pre-allocated output buffer.

**str\_len** : Output variable containing the length of the OCTET SEQUENCE.

**Returns** : Returns **ASN1\_SUCCESS** on success, or an error.

**asn1\_bit\_der ()**

```
void                asn1_bit_der                (unsigned char *str,
                                                int bit_len,
                                                unsigned char *der,
                                                int *der_len);
```

Creates the DER coding for a BIT STRING type (length and pad included).

**str** : BIT string.

**bit\_len** : number of meaningful bits in STR.

**der** : string returned.

**der\_len** : number of meaningful bytes of DER (der[0]..der[ans\_len-1]).

**asn1\_get\_bit\_der ()**

```
asn1_retCode        asn1_get_bit_der           (unsigned char *der,
                                                int der_len,
                                                int *ret_len,
                                                unsigned char *str,
                                                int str_size,
                                                int *bit_len);
```

Extract a BIT SEQUENCE from DER data.

**der** : DER data to decode containing the BIT SEQUENCE.

**der\_len** : Length of DER data to decode.

**ret\_len** : Output variable containing the length of the DER data.

**str** : Pre-allocated output buffer to put decoded BIT SEQUENCE in.

**str\_size** : Length of pre-allocated output buffer.

**bit\_len** : Output variable containing the size of the BIT SEQUENCE.

**Returns** : Return **ASN1\_SUCCESS** on success, or an error.

**asn1\_get\_length\_der ()**

```
signed long         asn1_get_length_der        (unsigned char *der,
                                                int der_len,
                                                int *len);
```

Extract a length field from DER data.

**der** : DER data to decode.

**der\_len** : Length of DER data to decode.

**len** : Output variable containing the length of the DER length field.

**Returns** : Return the decoded length value, or -1 on indefinite length, or -2 when the value was too big.

**asn1\_get\_length\_ber ()**

```
signed long      asn1_get_length_ber      (unsigned char *ber,
                                           int ber_len,
                                           int *len);
```

Extract a length field from BER data. The difference to **asn1\_get\_length\_der()** is that this function will return a length even if the value has indefinite encoding.

**ber** : BER data to decode.

**ber\_len** : Length of BER data to decode.

**len** : Output variable containing the length of the BER length field.

**Returns** : Return the decoded length value, or negative value when the value was too big.

Since 2.0

**asn1\_length\_der ()**

```
void            asn1_length_der          (unsigned long int len,
                                           unsigned char *ans,
                                           int *ans_len);
```

Creates the DER coding for the LEN parameter (only the length). The *ans* buffer is pre-allocated and must have room for the output.

**len** : value to convert.

**ans** : string returned.

**ans\_len** : number of meaningful bytes of ANS (ans[0]..ans[ans\_len-1]).

**asn1\_find\_node ()**

```
ASN1_TYPE      asn1_find_node           (ASN1_TYPE pointer,
                                           const char *name);
```

Searches for an element called *name* starting from *pointer*. The name is composed by different identifiers separated by dots. When *\*pointer* has a name, the first identifier must be the name of *\*pointer*, otherwise it must be the name of one child of *\*pointer*.

**pointer** : NODE\_ASN element pointer.

**name** : null terminated string with the element's name to find.

**Returns** : the search result, or **NULL** if not found.

**asn1\_copy\_node ()**

```
asn1_retCode    asn1_copy_node          (ASN1_TYPE dst,
                                           const char *dst_name,
                                           ASN1_TYPE src,
                                           const char *src_name);
```

Create a deep copy of a ASN1\_TYPE variable.

**dst** : Destination ASN1\_TYPE node.

**dst\_name** : Field name in destination node.

**src** : Source ASN1\_TYPE node.

**src\_name** : Field name in source node.

**Returns** : Return **ASN1\_SUCCESS** on success.

## LIBTASN1\_VERSION

```
#define LIBTASN1_VERSION ASN1_VERSION
```



### Warning

LIBTASN1\_VERSION is deprecated and should not be used in newly-written code.

---

## MAX\_NAME\_SIZE

```
# define MAX_NAME_SIZE ASN1_MAX_NAME_SIZE
```



### Warning

MAX\_NAME\_SIZE is deprecated and should not be used in newly-written code.

---

## MAX\_ERROR\_DESCRIPTION\_SIZE

```
# define MAX_ERROR_DESCRIPTION_SIZE ASN1_MAX_ERROR_DESCRIPTION_SIZE
```



### Warning

MAX\_ERROR\_DESCRIPTION\_SIZE is deprecated and should not be used in newly-written code.

---

## libtasn1\_strerror ()

```
const char *      libtasn1_strerror      (asn1_retCode error);
```



### Warning

libtasn1\_strerror is deprecated and should not be used in newly-written code. Use **asn1\_strerror()** instead.

---

Returns a string with a description of an error. This function is similar to strerror. The only difference is that it accepts an error (number) returned by a libtasn1 function.

**error** : is an error returned by a libtasn1 function.

**Returns** : Pointer to static zero-terminated string describing error code.

---

**libtasn1\_perror ()**

```
void libtasn1_perror (asn1_retCode error);
```

**Warning**

`libtasn1_perror` is deprecated and should not be used in newly-written code. Use `asn1_perror()` instead.

---

Prints a string to stderr with a description of an error. This function is like  `perror()`. The only difference is that it accepts an error returned by a libtasn1 function.

**error** : is an error returned by a libtasn1 function.

---



## Chapter 2

## Index

### A

ASN1\_API, 4  
asn1\_array2tree, 9  
ASN1\_ARRAY\_ERROR, 6  
ASN1\_ARRAY\_TYPE, 8  
asn1\_bit\_der, 18  
asn1\_check\_version, 16  
ASN1\_CLASS\_APPLICATION, 6  
ASN1\_CLASS\_CONTEXT\_SPECIFIC, 6  
ASN1\_CLASS\_PRIVATE, 7  
ASN1\_CLASS\_STRUCTURED, 7  
ASN1\_CLASS\_UNIVERSAL, 6  
asn1\_copy\_node, 19  
asn1\_create\_element, 10  
asn1\_delete\_element, 10  
asn1\_delete\_structure, 10  
asn1\_der\_coding, 13  
asn1\_der\_decoding, 13  
asn1\_der\_decoding\_element, 14  
asn1\_der\_decoding\_startEnd, 14  
ASN1\_DER\_ERROR, 5  
ASN1\_DER\_OVERFLOW, 6  
ASN1\_ELEMENT\_NOT\_EMPTY, 6  
ASN1\_ELEMENT\_NOT\_FOUND, 4  
ASN1\_ERROR\_TYPE\_ANY, 5  
asn1\_expand\_any\_defined\_by, 15  
asn1\_expand\_octet\_string, 15  
ASN1\_FILE\_NOT\_FOUND, 4  
asn1\_find\_node, 19  
asn1\_find\_structure\_from\_oid, 16  
ASN1\_GENERIC\_ERROR, 5  
asn1\_get\_bit\_der, 18  
asn1\_get\_length\_ber, 19  
asn1\_get\_length\_der, 18  
asn1\_get\_octet\_der, 17  
asn1\_get\_tag\_der, 17  
ASN1\_IDENTIFIER\_NOT\_FOUND, 5  
asn1\_length\_der, 19  
ASN1\_MAX\_ERROR\_DESCRIPTION\_SIZE, 8  
ASN1\_MAX\_NAME\_SIZE, 8  
ASN1\_MEM\_ALLOC\_ERROR, 5  
ASN1\_MEM\_ERROR, 5  
ASN1\_NAME\_TOO\_LONG, 6  
asn1\_number\_of\_elements, 13  
asn1\_octet\_der, 17  
asn1\_parser2array, 9  
asn1\_parser2tree, 9  
asn1\_perror, 16  
ASN1\_PRINT\_ALL, 6  
ASN1\_PRINT\_NAME, 6  
ASN1\_PRINT\_NAME\_TYPE, 6  
ASN1\_PRINT\_NAME\_TYPE\_VALUE, 6  
asn1\_print\_structure, 10  
asn1\_read\_tag, 15  
asn1\_read\_value, 12  
asn1\_retCode, 4  
asn1\_strerror, 16  
ASN1\_SUCCESS, 4  
ASN1\_SYNTAX\_ERROR, 5  
ASN1\_TAG\_BIT\_STRING, 7  
ASN1\_TAG\_BOOLEAN, 7  
ASN1\_TAG\_ENUMERATED, 8  
ASN1\_TAG\_ERROR, 5  
ASN1\_TAG\_GENERALIZEDTime, 7  
ASN1\_TAG\_GENERALSTRING, 8  
ASN1\_TAG\_IMPLICIT, 5  
ASN1\_TAG\_INTEGER, 7  
ASN1\_TAG\_NULL, 8  
ASN1\_TAG\_OBJECT\_ID, 7  
ASN1\_TAG\_OCTET\_STRING, 7  
ASN1\_TAG\_SEQUENCE, 7  
ASN1\_TAG\_SET, 7  
ASN1\_TAG\_UTCTime, 7  
ASN1\_TYPE, 8  
ASN1\_TYPE\_EMPTY, 8  
ASN1\_VALUE\_NOT\_FOUND, 5  
ASN1\_VALUE\_NOT\_VALID, 5  
ASN1\_VERSION, 4  
asn1\_write\_value, 11

### L

libtasn1\_perror, 21  
libtasn1\_strerror, 20  
LIBTASN1\_VERSION, 20

### M

MAX\_ERROR\_DESCRIPTION\_SIZE, [20](#)

MAX\_NAME\_SIZE, [20](#)

## **N**

node\_asn, [8](#)

---