

GNU Libtasn1 Reference Manual

COLLABORATORS

	<i>TITLE :</i> GNU Libtasn1 Reference Manual		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		August 24, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	GNU Libtasn1 API Reference Manual	1
1.1	libtasn1	1
2	Index	40

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 <https://www.gnu.org/software/libtasn1/>.

1.1 libtasn1

libtasn1 — GNU ASN.1 library

Functions

int	asn1_parser2tree ()
int	asn1_parser2array ()
int	asn1_array2tree ()
void	asn1_print_structure ()
int	asn1_create_element ()
int	asn1_delete_structure ()
int	asn1_delete_structure2 ()
int	asn1_delete_element ()
int	asn1_write_value ()
int	asn1_read_value ()
int	asn1_read_value_type ()
int	asn1_read_node_value ()
int	asn1_number_of_elements ()
int	asn1_der_coding ()
int	asn1_der_decoding2 ()
int	asn1_der_decoding ()
int	asn1_der_decoding_element ()
int	asn1_der_decoding_startEnd ()
int	asn1_expand_any_defined_by ()
int	asn1_expand_octet_string ()
int	asn1_read_tag ()
const char *	asn1_find_structure_from_oid ()
const char *	asn1_check_version ()
void	asn1_perror ()
long	asn1_get_length_der ()
long	asn1_get_length_ber ()
void	asn1_length_der ()

int	asn1_decode_simple_der ()
int	asn1_decode_simple_ber ()
int	asn1_encode_simple_der ()
asn1_node	asn1_find_node ()
int	asn1_copy_node ()
asn1_node	asn1_dup_node ()
int	asn1_get_tag_der ()
void	asn1_octet_der ()
int	asn1_get_octet_der ()
void	asn1_bit_der ()
int	asn1_get_bit_der ()
int	asn1_get_object_id_der ()
int	asn1_object_id_der ()

Types and Values

#define	ASN1_API
#define	ASN1_VERSION
#define	ASN1_VERSION_MAJOR
#define	ASN1_VERSION_MINOR
#define	ASN1_VERSION_PATCH
#define	ASN1_VERSION_NUMBER
#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_TIME_ENCODING_ERROR
#define	ASN1_RECURSION
#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
#define	ASN1_TAG_NUMERIC_STRING
#define	ASN1_TAG_IA5_STRING
#define	ASN1_TAG_TELETEX_STRING
#define	ASN1_TAG_PRINTABLE_STRING
#define	ASN1_TAG_UNIVERSAL_STRING
#define	ASN1_TAG_BMP_STRING
#define	ASN1_TAG_UTF8_STRING
#define	ASN1_TAG_VISIBLE_STRING
typedef	asn1_node_st
typedef	asn1_node
typedef	asn1_node_const
#define	ASN1_MAX_NAME_SIZE
#define	ASN1_ETYPE_INVALID
#define	ASN1_ETYPE_CONSTANT
#define	ASN1_ETYPE_IDENTIFIER
#define	ASN1_ETYPE_INTEGER
#define	ASN1_ETYPE_BOOLEAN
#define	ASN1_ETYPE_SEQUENCE
#define	ASN1_ETYPE_BIT_STRING
#define	ASN1_ETYPE_OCTET_STRING
#define	ASN1_ETYPE_TAG
#define	ASN1_ETYPE_DEFAULT
#define	ASN1_ETYPE_SIZE
#define	ASN1_ETYPE_SEQUENCE_OF
#define	ASN1_ETYPE_OBJECT_ID
#define	ASN1_ETYPE_ANY
#define	ASN1_ETYPE_SET
#define	ASN1_ETYPE_SET_OF
#define	ASN1_ETYPE_DEFINITIONS
#define	ASN1_ETYPE_CHOICE
#define	ASN1_ETYPE_IMPORTS
#define	ASN1_ETYPE_NULL
#define	ASN1_ETYPE_ENUMERATED
#define	ASN1_ETYPE_GENERALSTRING
#define	ASN1_ETYPE_NUMERIC_STRING
#define	ASN1_ETYPE_IA5_STRING
#define	ASN1_ETYPE_TELETEX_STRING
#define	ASN1_ETYPE_PRINTABLE_STRING
#define	ASN1_ETYPE_UNIVERSAL_STRING
#define	ASN1_ETYPE_BMP_STRING
#define	ASN1_ETYPE_UTF8_STRING
#define	ASN1_ETYPE_VISIBLE_STRING
#define	ASN1_ETYPE_UTC_TIME
#define	ASN1_ETYPE_GENERALIZED_TIME
#define	ASN1_DELETE_FLAG_ZEROIZE
#define	ASN1_DECODE_FLAG_ALLOW_PADDING
#define	ASN1_DECODE_FLAG_STRICT_DER
#define	ASN1_DECODE_FLAG_ALLOW_INCORRECT_TIME
struct	asn1_data_node_st

#define	ASN1_MAX_ERROR_DESCRIPTION_SIZE
#define	ASN1_MAX_TAG_SIZE
#define	ASN1_MAX_LENGTH_SIZE
#define	ASN1_MAX_TL_SIZE
typedef	_ASN1_GCC_ATTR_DEPRECATED
#define	node_asn_struct
#define	node_asn
#define	ASN1_TYPE
#define	ASN1_TYPE_EMPTY
#define	static_struct_asn
#define	ASN1_ARRAY_TYPE
#define	asn1_static_node_t
#define	node_data_struct
#define	ASN1_DATA_NODE

Description

The Libtasn1 library provides Abstract Syntax Notation One (ASN.1, as specified by the X.680 ITU-T recommendation) parsing and structures management, and Distinguished Encoding Rules (DER, as per X.690) encoding and decoding functions.

Functions

asn1_parser2tree ()

```
int
asn1_parser2tree (const char *file,
                 asn1_node *definitions,
                 char *error_desc);
```

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

Parameters

file	specify the path and the name of file that contains ASN.1 declarations.
definitions	return the pointer to the structure created from "file" ASN.1 declarations.
error_desc	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 **NULL**, **ASN1_FILE_NOT_FOUND** if an error occurred while opening *file*, **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 with more than **ASN1_MAX_NAME_SIZE** characters.

asn1_parser2array ()

```
int
asn1_parser2array (const char *inputFileName,
                  const char *outputFileName,
                  const char *vectorName,
                  char *error_desc);
```

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".

Parameters

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.
error_desc	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 with more than **ASN1_MAX_NAME_SIZE** characters.

asn1_array2tree ()

```
int
asn1_array2tree (const asn1_static_node *array,
                 asn1_node *definitions,
                 char *errorDescription);
```

Creates the structures needed to manage the ASN.1 definitions. *array* is a vector created by **asn1_parser2array()**.

Parameters

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 NULL, **ASN1_IDENTIFIER** 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_node_const 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*.

Parameters

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_TYPE_VALUE , or ASN1_PRINT_ALL .

asn1_create_element ()

```
int
asn1_create_element (asn1_node_const definitions,
                   const char *source_name,
                   asn1_node *element);
```

Creates a structure of type *source_name*. Example using "pkix.asn":

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

Parameters

definitions	pointer to the structure returned by "parser_asn1" function
source_name	the name of the type of the new structure (must be inside <i>p_structure</i>).
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 ()

```
int
asn1_delete_structure (asn1_node *structure);
```

Deletes the structure **structure*. At the end, **structure* is set to NULL.

Parameters

structure	pointer to the structure that you want to delete.
-----------	---

Returns

ASN1_SUCCESS if successful, **ASN1_ELEMENT_NOT_FOUND** if **structure* was NULL.

asn1_delete_structure2 ()

```
int
asn1_delete_structure2 (asn1_node *structure,
                       unsigned int flags);
```

Deletes the structure **structure*. At the end, **structure* is set to NULL.

Parameters

structure	pointer to the structure that you want to delete.
flags	additional flags (see ASN1_DELETE_FLAG_ZEROIZE)

Returns

ASN1_SUCCESS if successful, **ASN1_ELEMENT_NOT_FOUND** if **structure* was NULL.

asn1_delete_element ()

```
int
asn1_delete_element (asn1_node structure,
                    const char *element_name);
```

Deletes the element named **element_name* inside **structure*.

Parameters

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 ()

```
int
asn1_write_value (asn1_node 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);
```

Parameters

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 <i>len</i> is >0, <i>VALUE</i> must be a two's complement form integer. if <i>len</i> =0 <i>*VALUE</i> must be a null terminated string with an integer value.
len	number of bytes of <i>*value</i> to use to set the value: <i>value</i> [0].. <i>value</i> [<i>len</i> -1] or 0 if <i>value</i> 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_NOT_VALID** if *ivalue* has a wrong format.

asn1_read_value ()

```
int
asn1_read_value (asn1_node_const root,
                const char *name,
                void *ivalue,
                int *len);
```

Returns the value of one element inside a structure. If an element is **OPTIONAL** and this 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. If the *root* provided is a node to specific sequence element, then the keyword "?CURRENT" is also acceptable and indicates the current sequence element of this node.

Note that there can be valid values with length zero. In these case this function will succeed and *len* will be zero.

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.

Parameters

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 (may be NULL).
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_NOT_FOUND** 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. On the occasion that the stored data are of zero-length this function may return **ASN1_SUCCESS** even if the provided *len* is zero.

asn1_read_value_type ()

```
int
asn1_read_value_type (asn1_node_const root,
                     const char *name,
                     void *ivalue,
                     int *len,
                     unsigned int *etype);
```

Returns the type and value of one element inside a structure. If an element is OPTIONAL and this 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. If the *root* provided is a node to specific sequence element, then the keyword "?CURRENT" is also acceptable and indicates the current sequence element of this node.

Note that there can be valid values with length zero. In these case this function will succeed and *len* will be zero.

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.

Parameters

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 (may be NULL).	
len	number of bytes of *value: value[0]..value[len-1]. Initially holds the sizeof value.	
etype	The type of the value read (ASN1_ETYPE)	

Returns

ASN1_SUCCESS if value is returned, **ASN1_ELEMENT_NOT_FOUND** if *name* is not a valid element, **ASN1_VALUE_NOT_FOUND** 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. On the occasion that the stored data are of zero-length this function may return **ASN1_SUCCESS** even if the provided *len* is zero.

asn1_read_node_value ()

```
int
asn1_read_node_value (asn1_node_const node,
                    asn1_data_node_st *data);
```

Returns the value a data node inside a `asn1_node` structure. The data returned should be handled as constant values.

Parameters

node	pointer to a node.
data	a point to a asn1_data_node_st

Returns

ASN1_SUCCESS if the node exists.

asn1_number_of_elements ()

```
int
asn1_number_of_elements (asn1_node_const element,
                        const char *name,
                        int *num);
```

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

Parameters

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 ()

```
int
asn1_der_coding (asn1_node_const element,
                const char *name,
                void *ider,
                int *len,
                char *ErrorDescription);
```

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

Parameters

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 <i>*ider</i> : <i>ider</i> [0].. <i>ider</i> [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_NOT_FOUND** 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_decoding2 ()

```
int
asn1_der_decoding2 (asn1_node *element,
                   const void *ider,
                   int *max_ider_len,
                   unsigned int flags,
                   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 **ASN1_DECODE_FLAG_ALLOW_PADDING** flag is set then the function will ignore padding after the decoded DER data. Upon a successful return the value of **max_ider_len* will be set to the number of bytes decoded.

If **ASN1_DECODE_FLAG_STRICT_DER** flag is set then the function will not decode any BER-encoded elements.

Parameters

element	pointer to an ASN1 structure.
ider	vector that contains the DER encoding.
max_ider_len	pointer to an integer giving the information about the maximal number of bytes occupied by <i>*ider</i> . The real size of the DER encoding is returned through this pointer.
flags	flags controlling the behaviour of the function.
errorDescription	null-terminated string contains details when an error occurred.

Returns

ASN1_SUCCESS if DER encoding OK, **ASN1_ELEMENT_NOT_FOUND** if *ELEMENT* is **NULL**, and **ASN1_TAG_ERROR** or **ASN1_DER_ERROR** if the der encoding doesn't match the structure name (**ELEMENT* deleted).

asn1_der_decoding ()

```
int
asn1_der_decoding (asn1_node *element,
                  const void *ider,
                  int ider_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()`.

Note that the **element* variable is provided as a pointer for historical reasons.

Parameters

element	pointer to an ASN1 structure.	
ider	vector that contains the DER encoding.	
ider_len	number of bytes of <i>*ider</i> : <i>ider</i> [0].. <i>ider</i> [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 **NULL**, and **ASN1_TAG_ERROR** or **ASN1_DER_ERROR** if the der encoding doesn't match the structure name (**ELEMENT* deleted).

asn1_der_decoding_element ()

```
int
asn1_der_decoding_element (asn1_node *structure,
                          const char *elementName,
                          const void *ider,
                          int len,
                          char *errorDescription);
```



Warning

`asn1_der_decoding_element` is deprecated and should not be used in newly-written code.

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 **NULL**.

This function is deprecated and may just be an alias to `asn1_der_decoding` in future versions. Use `asn1_der_decoding()` instead.

Parameters

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 **NULL** 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 ()

```
int
asn1_der_decoding_startEnd (asn1_node element,
                           const void *ider,
                           int ider_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.

Note that since libtasn1 3.7 the *ider* and *ider_len* parameters can be omitted, if the element is already decoded using **asn1_der_decoding()**.

Parameters

element	pointer to an ASN1 element
ider	vector that contains the DER encoding.
ider_len	number of bytes of *ider: <i>ider</i> [0].. <i>ider</i> [len-1]
name_element	an element of NAME structure.
start	the position of the first byte of NAME_ELEMENT decoding (<i>ider</i> [*start])
end	the position of the last byte of NAME_ELEMENT decoding (<i>ider</i> [*end])

Returns

ASN1_SUCCESS if DER encoding OK, **ASN1_ELEMENT_NOT_FOUND** if ELEMENT is **asn1_node** 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 ()

```
int
asn1_expand_any_defined_by (asn1_node_const definitions,
                           asn1_node *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.

Parameters

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 ()

```
int
asn1_expand_octet_string (asn1_node_const definitions,
                          asn1_node *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.

Parameters

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** if it wasn't possible to find the type to use for expansion, or other errors depending on DER decoding.

asn1_read_tag ()

```
int
```

```
asn1_read_tag (asn1_node_const 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](#).

Parameters

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_node_const definitions,
                             const char *oidValue);
```

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

Parameters

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.

Parameters

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_perror ()

```
void
asn1_perror (int 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.

Parameters

error	is an error returned by a libtasn1 function.
-------	---

Since: 1.6

asn1_get_length_der ()

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

Extract a length field from DER data.

Parameters

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 to fit in a int, or -4 when the decoded length value plus `len` would exceed `der_len`.

asn1_get_length_ber ()

```
long
asn1_get_length_ber (const 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.

Parameters

<code>ber</code>	BER data to decode.	
<code>ber_len</code>	Length of BER data to decode.	
<code>len</code>	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 *der,
                int *der_len);
```

Creates the DER encoding of the provided length value. The `der` buffer must have enough room for the output. The maximum length this function will encode is `ASN1_MAX_LENGTH_SIZE`.

To know the size of the DER encoding use a `NULL` value for `der`.

Parameters

<code>len</code>	value to convert.	
<code>der</code>	buffer to hold the returned encoding (may be <code>NULL</code>).	
<code>der_len</code>	number of meaningful bytes of ANS (<code>der[0]..der[der_len-1]</code>).	

`asn1_decode_simple_der ()`

```
int
asn1_decode_simple_der (unsigned int etype,
                       const unsigned char *der,
                       unsigned int _der_len,
                       const unsigned char **str,
                       unsigned int *str_len);
```

Decodes a simple DER encoded type (e.g. a string, which is not constructed). The output is a pointer inside the `der`.

Parameters

etype	The type of the string to be encoded (ASN1_ETYPE_)
der	the encoded string
_der_len	the bytes of the encoded string
str	a pointer to the data
str_len	the length of the data

Returns

ASN1_SUCCESS if successful or an error value.

asn1_decode_simple_ber ()

```
int
asn1_decode_simple_ber (unsigned int etype,
                        const unsigned char *der,
                        unsigned int _der_len,
                        unsigned char **str,
                        unsigned int *str_len,
                        unsigned int *ber_len);
```

Decodes a BER encoded type. The output is an allocated value of the data. This decodes BER STRINGS only. Other types are decoded as DER.

Parameters

etype	The type of the string to be encoded (ASN1_ETYPE_)
der	the encoded string
_der_len	the bytes of the encoded string
str	a pointer to the data
str_len	the length of the data
ber_len	the total length occupied by BER (may be NULL)

Returns

ASN1_SUCCESS if successful or an error value.

asn1_encode_simple_der ()

```
int
asn1_encode_simple_der (unsigned int etype,
                        const unsigned char *str,
                        unsigned int str_len,
                        unsigned char *tl,
                        unsigned int *tl_len);
```

Creates the DER encoding for various simple ASN.1 types like strings etc. It stores the tag and length in *tl*, which should have space for at least **ASN1_MAX_TL_SIZE** bytes. Initially *tl_len* should contain the size of *tl*.

The complete DER encoding should consist of the value in *tl* appended with the provided *str*.

Parameters

etype	The type of the string to be encoded (ASN1_ETYPE_)
str	the string data.
str_len	the string length
tl	the encoded tag and length
tl_len	the bytes of the <i>tl</i> field

Returns

ASN1_SUCCESS if successful or an error value.

asn1_find_node ()

```
asn1_node
asn1_find_node (asn1_node_const 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*.

Parameters

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 ()

```
int
asn1_copy_node (asn1_node dst,
               const char *dst_name,
               asn1_node_const src,
               const char *src_name);
```

Create a deep copy of a *asn1_node* variable. That function requires *dst* to be expanded using [asn1_create_element\(\)](#).

Parameters

dst	Destination <i>asn1</i> node.
dst_name	Field name in destination node.
src	Source <i>asn1</i> node.
src_name	Field name in source node.

Returns

Return **ASN1_SUCCESS** on success.

asn1_dup_node ()

```
asn1_node
asn1_dup_node (asn1_node_const src,
               const char *src_name);
```

Create a deep copy of a `asn1_node` variable. This function will return an exact copy of the provided structure.

Parameters

<code>src</code>	Source <code>asn1</code> node.
<code>src_name</code>	Field name in source node.

Returns

Return **NULL** on failure.

asn1_get_tag_der ()

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

Decode the class and TAG from DER code.

Parameters

<code>der</code>	DER data to decode.
<code>der_len</code>	Length of DER data to decode.
<code>cls</code>	Output variable containing decoded class.
<code>len</code>	Output variable containing the length of the DER TAG data.
<code>tag</code>	Output variable containing the decoded tag (may be NULL).

Returns

Returns **ASN1_SUCCESS** on success, or an error.

asn1_octet_der ()

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

Creates a length-value DER encoding for the input data. The DER encoding of the input data will be placed in the *der* variable.

Note that the OCTET STRING tag is not included in the output.

This function does not return any value because it is expected that *der_len* will contain enough bytes to store the string plus the DER encoding. The DER encoding size can be obtained using [asn1_length_der\(\)](#).

Parameters

<i>str</i>	the input data.	
<i>str_len</i>	STR length (<i>str</i> [0].. <i>str</i> [* <i>str_len</i> -1]).	
<i>der</i>	encoded string returned.	
<i>der_len</i>	number of meaningful bytes of DER (<i>der</i> [0].. <i>der</i> [* <i>der_len</i> -1]).	

asn1_get_octet_der ()

```
int
asn1_get_octet_der (const 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. Note that this function expects the DER data past the tag field, i.e., the length and content octets.

Parameters

<i>der</i>	DER data to decode containing the OCTET SEQUENCE.	
<i>der_len</i>	The length of the <i>der</i> data to decode.	
<i>ret_len</i>	Output variable containing the encoded length of the DER data.	
<i>str</i>	Pre-allocated output buffer to put decoded OCTET SEQUENCE in.	
<i>str_size</i>	Length of pre-allocated output buffer.	
<i>str_len</i>	Output variable containing the length of the contents of the OCTET SEQUENCE.	

Returns

Returns `ASN1_SUCCESS` on success, or an error.

asn1_bit_der ()

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

Creates a length-value DER encoding for the input data as it would have been for a BIT STRING. The DER encoded data will be copied in *der*.

Note that the BIT STRING tag is not included in the output.

This function does not return any value because it is expected that *der_len* will contain enough bytes to store the string plus the DER encoding. The DER encoding size can be obtained using `asn1_length_der()`.

Parameters

<i>str</i>	BIT string.	
<i>bit_len</i>	number of meaningful bits in STR.	
<i>der</i>	string returned.	
<i>der_len</i>	number of meaningful bytes of DER (der[0]..der[ans_len-1]).	

asn1_get_bit_der ()

```
int
asn1_get_bit_der (const 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.

Parameters

<i>der</i>	DER data to decode containing the BIT SEQUENCE.	
<i>der_len</i>	Length of DER data to decode.	
<i>ret_len</i>	Output variable containing the length of the DER data.	
<i>str</i>	Pre-allocated output buffer to put decoded BIT SEQUENCE in.	
<i>str_size</i>	Length of pre-allocated output buffer.	

bit_len	Output variable containing the size of the BIT SEQUENCE.
---------	--

Returns

ASN1_SUCCESS on success, or an error.

asn1_get_object_id_der ()

```
int
asn1_get_object_id_der (const unsigned char *der,
                       int der_len,
                       int *ret_len,
                       char *str,
                       int str_size);
```

Converts a DER encoded object identifier to its textual form. This function expects the DER object identifier without the tag.

Parameters

der	DER data to decode containing the OBJECT IDENTIFIER
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 the textual object id in.
str_size	Length of pre-allocated output buffer.

Returns

ASN1_SUCCESS on success, or an error.

asn1_object_id_der ()

```
int
asn1_object_id_der (const char *str,
                   unsigned char *der,
                   int *der_len,
                   unsigned flags);
```

Creates the DER encoding of the provided object identifier.

Parameters

str	An object identifier in numeric, dot format.
-----	--

<code>der</code>	buffer to hold the returned encoding (may be NULL).
<code>der_len</code>	initially the size of <code>der</code> ; will hold the final size.
<code>flags</code>	must be zero

Returns

ASN1_SUCCESS if DER encoding was OK, **ASN1_VALUE_NOT_VALID** if `str` is not a valid OID, **ASN1_MEM_ERROR** if the `der` vector isn't big enough and in this case `der_len` will contain the length needed.

Types and Values

ASN1_API

```
# define ASN1_API __attribute__((__visibility__("default")))
```

ASN1_VERSION

```
# define ASN1_VERSION "4.19.0.3-62b1-dirty"
```

Version of the library as a string.

ASN1_VERSION_MAJOR

```
# define ASN1_VERSION_MAJOR 4
```

Major version number of the library.

ASN1_VERSION_MINOR

```
# define ASN1_VERSION_MINOR 19
```

Minor version number of the library.

ASN1_VERSION_PATCH

```
# define ASN1_VERSION_PATCH 0
```

Patch version number of the library.

ASN1_VERSION_NUMBER

```
# define ASN1_VERSION_NUMBER 0x041300
```

Version number of the library as a number.

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_TIME_ENCODING_ERROR

```
# define ASN1_TIME_ENCODING_ERROR~18
```

ASN1_RECURSION

```
# define ASN1_RECURSION 19
```

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
```

ASN1_TAG_NUMERIC_STRING

```
# define ASN1_TAG_NUMERIC_STRING 0x12
```

ASN1_TAG_IA5_STRING

```
# define ASN1_TAG_IA5_STRING    0x16
```

ASN1_TAG_TELETEX_STRING

```
# define ASN1_TAG_TELETEX_STRING 0x14
```

ASN1_TAG_PRINTABLE_STRING

```
# define ASN1_TAG_PRINTABLE_STRING~0x13
```

ASN1_TAG_UNIVERSAL_STRING

```
# define ASN1_TAG_UNIVERSAL_STRING~0x1C
```

ASN1_TAG_BMP_STRING

```
# define ASN1_TAG_BMP_STRING 0x1E
```

ASN1_TAG_UTF8_STRING

```
# define ASN1_TAG_UTF8_STRING 0x0C
```

ASN1_TAG_VISIBLE_STRING

```
# define ASN1_TAG_VISIBLE_STRING 0x1A
```

asn1_node_st

```
typedef struct asn1_node_st asn1_node_st;
```

asn1_node

```
typedef asn1_node_st *asn1_node;
```

Structure definition used for the node of the tree that represents an ASN.1 DEFINITION.

asn1_node_const

```
typedef const asn1_node_st *asn1_node_const;
```

ASN1_MAX_NAME_SIZE

```
# define ASN1_MAX_NAME_SIZE 64
```

Maximum number of characters of a name inside a file with ASN1 definitions.

ASN1_ETYPE_INVALID

```
# define ASN1_ETYPE_INVALID 0
```

ASN1_ETYPE_CONSTANT

```
# define ASN1_ETYPE_CONSTANT 1
```

ASN1_ETYPE_IDENTIFIER

```
# define ASN1_ETYPE_IDENTIFIER 2
```

ASN1_ETYPE_INTEGER

```
# define ASN1_ETYPE_INTEGER 3
```

ASN1_ETYPE_BOOLEAN

```
# define ASN1_ETYPE_BOOLEAN 4
```

ASN1_ETYPE_SEQUENCE

```
# define ASN1_ETYPE_SEQUENCE 5
```

ASN1_ETYPE_BIT_STRING

```
# define ASN1_ETYPE_BIT_STRING 6
```

ASN1_ETYPE_OCTET_STRING

```
# define ASN1_ETYPE_OCTET_STRING 7
```

ASN1_ETYPE_TAG

```
# define ASN1_ETYPE_TAG 8
```

ASN1_ETYPE_DEFAULT

```
# define ASN1_ETYPE_DEFAULT 9
```

ASN1_ETYPE_SIZE

```
# define ASN1_ETYPE_SIZE 10
```

ASN1_ETYPE_SEQUENCE_OF

```
# define ASN1_ETYPE_SEQUENCE_OF 11
```

ASN1_ETYPE_OBJECT_ID

```
# define ASN1_ETYPE_OBJECT_ID 12
```

ASN1_ETYPE_ANY

```
# define ASN1_ETYPE_ANY 13
```

ASN1_ETYPE_SET

```
# define ASN1_ETYPE_SET 14
```

ASN1_ETYPE_SET_OF

```
# define ASN1_ETYPE_SET_OF 15
```

ASN1_ETYPE_DEFINITIONS

```
# define ASN1_ETYPE_DEFINITIONS 16
```

ASN1_ETYPE_CHOICE

```
# define ASN1_ETYPE_CHOICE 18
```

ASN1_ETYPE_IMPORTS

```
# define ASN1_ETYPE_IMPORTS 19
```

ASN1_ETYPE_NULL

```
# define ASN1_ETYPE_NULL 20
```

ASN1_ETYPE_ENUMERATED

```
# define ASN1_ETYPE_ENUMERATED 21
```

ASN1_ETYPE_GENERALSTRING

```
# define ASN1_ETYPE_GENERALSTRING 27
```

ASN1_ETYPE_NUMERIC_STRING

```
# define ASN1_ETYPE_NUMERIC_STRING 28
```

ASN1_ETYPE_IA5_STRING

```
# define ASN1_ETYPE_IA5_STRING 29
```

ASN1_ETYPE_TELETEX_STRING

```
# define ASN1_ETYPE_TELETEX_STRING 30
```

ASN1_ETYPE_PRINTABLE_STRING

```
# define ASN1_ETYPE_PRINTABLE_STRING 31
```

ASN1_ETYPE_UNIVERSAL_STRING

```
# define ASN1_ETYPE_UNIVERSAL_STRING 32
```

ASN1_ETYPE_BMP_STRING

```
# define ASN1_ETYPE_BMP_STRING 33
```

ASN1_ETYPE_UTF8_STRING

```
# define ASN1_ETYPE_UTF8_STRING 34
```

ASN1_ETYPE_VISIBLE_STRING

```
# define ASN1_ETYPE_VISIBLE_STRING 35
```

ASN1_ETYPE_UTC_TIME

```
# define ASN1_ETYPE_UTC_TIME 36
```

ASN1_ETYPE_GENERALIZED_TIME

```
# define ASN1_ETYPE_GENERALIZED_TIME 37
```

ASN1_DELETE_FLAG_ZEROIZE

```
# define ASN1_DELETE_FLAG_ZEROIZE 1
```

Used by: [asn1_delete_structure2\(\)](#)

Zeroize values prior to deinitialization.

ASN1_DECODE_FLAG_ALLOW_PADDING

```
# define ASN1_DECODE_FLAG_ALLOW_PADDING 1
```

Used by: [asn1_der_decoding2\(\)](#)

This flag would allow arbitrary data past the DER data.

ASN1_DECODE_FLAG_STRICT_DER

```
# define ASN1_DECODE_FLAG_STRICT_DER (1<<1)
```

Used by: [asn1_der_decoding2\(\)](#)

This flag would ensure that no BER decoding takes place.

ASN1_DECODE_FLAG_ALLOW_INCORRECT_TIME

```
# define ASN1_DECODE_FLAG_ALLOW_INCORRECT_TIME (1<<2)
```

Used by: [asn1_der_decoding2\(\)](#)

This flag will tolerate Time encoding errors when in strict DER.

struct asn1_data_node_st

```
struct asn1_data_node_st {
    const char *name;      /* Node name */
    const void *value;     /* Node value */
    unsigned int value_len; /* Node value size */
    unsigned int type;     /* Node value type (ASN1_ETYPE_*) */
};
```

Data node inside a [asn1_node](#) structure.

Members

<code>const char *name;</code>	Node name
<code>const void *value;</code>	Node value
<code>unsigned int value_len;</code>	Node value size
<code>unsigned int type;</code>	Node value type (ASN1_ETYPE_*)

ASN1_MAX_ERROR_DESCRIPTION_SIZE

```
# define ASN1_MAX_ERROR_DESCRIPTION_SIZE 128
```

Maximum number of characters of a description message (null character included).

ASN1_MAX_TAG_SIZE

```
# define ASN1_MAX_TAG_SIZE 4
```

ASN1_MAX_LENGTH_SIZE

```
# define ASN1_MAX_LENGTH_SIZE 9
```

ASN1_MAX_TL_SIZE

```
# define ASN1_MAX_TL_SIZE (ASN1_MAX_TAG_SIZE+ASN1_MAX_LENGTH_SIZE)
```

_ASN1_GCC_ATTR_DEPRECATED

```
typedef int asn1_retCode _ASN1_GCC_ATTR_DEPRECATED;
```



Warning

`_ASN1_GCC_ATTR_DEPRECATED` is deprecated and should not be used in newly-written code.

node_asn_struct

```
# define node_asn_struct _Pragma ("GCC warning \"'node_asn_struct' macro is deprecated, ←  
use 'asn1_node' instead.\"") asn1_node_st
```



Warning

`node_asn_struct` has been deprecated since version 3.0 and should not be used in newly-written code.
Use `asn1_node` instead.

Compat #define.

node_asn

```
# define node_asn _Pragma ("GCC warning \"'node_asn' macro is deprecated, use 'asn1_node' ←  
instead.\"") asn1_node_st
```



Warning

`node_asn` has been deprecated since version 3.0 and should not be used in newly-written code.
Use `asn1_node` instead.

Compat #define.

ASN1_TYPE

```
# define ASN1_TYPE _Pragma ("GCC warning \"'ASN1_TYPE' macro is deprecated, use ' ↵  
asn1_node' instead.\") asn1_node
```



Warning

ASN1_TYPE has been deprecated since version 3.0 and should not be used in newly-written code.
Use `asn1_node` instead.

Compat #define.

ASN1_TYPE_EMPTY

```
# define ASN1_TYPE_EMPTY _Pragma ("GCC warning \"'ASN1_TYPE_EMPTY' macro is deprecated, ↵  
use 'NULL' instead.\") NULL
```



Warning

ASN1_TYPE_EMPTY has been deprecated since version 3.0 and should not be used in newly-written code.
Use NULL instead.

Compat #define.

static_struct_asn

```
# define static_struct_asn _Pragma ("GCC warning \"'static_struct_asn' macro is ↵  
deprecated, use 'asn1_static_node_st' instead.\") asn1_static_node_st
```



Warning

static_struct_asn has been deprecated since version 3.0 and should not be used in newly-written code.
Use `asn1_static_node` instead.

Compat #define.

ASN1_ARRAY_TYPE

```
# define ASN1_ARRAY_TYPE _Pragma ("GCC warning \"'ASN1_ARRAY_TYPE' macro is deprecated, ↵  
use 'asn1_static_node' instead.\") asn1_static_node
```



Warning

ASN1_ARRAY_TYPE has been deprecated since version 3.0 and should not be used in newly-written code.
Use `asn1_static_node` instead.

Compat #define.

asn1_static_node_t

```
# define asn1_static_node_t _Pragma ("GCC warning \"'asn1_static_node_t' macro is deprecated, use 'asn1_static_node' instead.\"") asn1_static_node
```

**Warning**

asn1_static_node_t has been deprecated since version 3.0 and should not be used in newly-written code. Use [asn1_static_node](#) instead.

Compat #define.

node_data_struct

```
# define node_data_struct _Pragma ("GCC warning \"'node_data_struct' macro is deprecated, use 'asn1_data_node_st' instead.\"") asn1_data_node_st
```

**Warning**

node_data_struct has been deprecated since version 3.0 and should not be used in newly-written code. Use [asn1_data_node_st](#) instead.

Compat #define.

ASN1_DATA_NODE

```
# define ASN1_DATA_NODE _Pragma ("GCC warning \"'asn1_static_node_t' macro is deprecated, use 'asn1_static_node' instead.\"") asn1_data_node_st
```

**Warning**

ASN1_DATA_NODE has been deprecated since version 3.0 and should not be used in newly-written code. Use [asn1_data_node_st](#) instead.

Compat #define.

Chapter 2

Index

-
- _ASN1_GCC_ATTR_DEPRECATED, 37
- A**
- ASN1_API, 27
- asn1_array2tree, 5
- ASN1_ARRAY_ERROR, 29
- ASN1_ARRAY_TYPE, 38
- asn1_bit_der, 25
- asn1_check_version, 18
- ASN1_CLASS_APPLICATION, 30
- ASN1_CLASS_CONTEXT_SPECIFIC, 30
- ASN1_CLASS_PRIVATE, 30
- ASN1_CLASS_STRUCTURED, 30
- ASN1_CLASS_UNIVERSAL, 30
- asn1_copy_node, 22
- asn1_create_element, 6
- ASN1_DATA_NODE, 39
- asn1_data_node_st, 36
- ASN1_DECODE_FLAG_ALLOW_INCORRECT_TIME, 36
- ASN1_DECODE_FLAG_ALLOW_PADDING, 35
- ASN1_DECODE_FLAG_STRICT_DER, 36
- asn1_decode_simple_ber, 21
- asn1_decode_simple_der, 20
- asn1_delete_element, 7
- ASN1_DELETE_FLAG_ZEROIZE, 35
- asn1_delete_structure, 7
- asn1_delete_structure2, 7
- asn1_der_coding, 13
- asn1_der_decoding, 14
- asn1_der_decoding2, 14
- asn1_der_decoding_element, 15
- asn1_der_decoding_startEnd, 16
- ASN1_DER_ERROR, 28
- ASN1_DER_OVERFLOW, 29
- asn1_dup_node, 23
- ASN1_ELEMENT_NOT_EMPTY, 29
- ASN1_ELEMENT_NOT_FOUND, 28
- asn1_encode_simple_der, 21
- ASN1_ERROR_TYPE_ANY, 28
- ASN1_ETYPE_ANY, 34
- ASN1_ETYPE_BIT_STRING, 33
- ASN1_ETYPE_BMP_STRING, 35
- ASN1_ETYPE_BOOLEAN, 33
- ASN1_ETYPE_CHOICE, 34
- ASN1_ETYPE_CONSTANT, 32
- ASN1_ETYPE_DEFAULT, 33
- ASN1_ETYPE_DEFINITIONS, 34
- ASN1_ETYPE_ENUMERATED, 34
- ASN1_ETYPE_GENERALIZED_TIME, 35
- ASN1_ETYPE_GENERALSTRING, 34
- ASN1_ETYPE_IA5_STRING, 34
- ASN1_ETYPE_IDENTIFIER, 33
- ASN1_ETYPE_IMPORTS, 34
- ASN1_ETYPE_INTEGER, 33
- ASN1_ETYPE_INVALID, 32
- ASN1_ETYPE_NULL, 34
- ASN1_ETYPE_NUMERIC_STRING, 34
- ASN1_ETYPE_OBJECT_ID, 33
- ASN1_ETYPE_OCTET_STRING, 33
- ASN1_ETYPE_PRINTABLE_STRING, 35
- ASN1_ETYPE_SEQUENCE, 33
- ASN1_ETYPE_SEQUENCE_OF, 33
- ASN1_ETYPE_SET, 34
- ASN1_ETYPE_SET_OF, 34
- ASN1_ETYPE_SIZE, 33
- ASN1_ETYPE_TAG, 33
- ASN1_ETYPE_TELETEX_STRING, 35
- ASN1_ETYPE_UNIVERSAL_STRING, 35
- ASN1_ETYPE_UTC_TIME, 35
- ASN1_ETYPE_UTF8_STRING, 35
- ASN1_ETYPE_VISIBLE_STRING, 35
- asn1_expand_any_defined_by, 17
- asn1_expand_octet_string, 17
- ASN1_FILE_NOT_FOUND, 28
- asn1_find_node, 22
- asn1_find_structure_from_oid, 18
- ASN1_GENERIC_ERROR, 28
- asn1_get_bit_der, 25
- asn1_get_length_ber, 19
- asn1_get_length_der, 19
- asn1_get_object_id_der, 26
- asn1_get_octet_der, 24
- asn1_get_tag_der, 23
- ASN1_IDENTIFIER_NOT_FOUND, 28
- asn1_length_der, 20

ASN1_MAX_ERROR_DESCRIPTION_SIZE, 36
ASN1_MAX_LENGTH_SIZE, 36
ASN1_MAX_NAME_SIZE, 32
ASN1_MAX_TAG_SIZE, 36
ASN1_MAX_TL_SIZE, 37
ASN1_MEM_ALLOC_ERROR, 29
ASN1_MEM_ERROR, 29
ASN1_NAME_TOO_LONG, 29
asn1_node, 32
asn1_node_const, 32
asn1_node_st, 32
asn1_number_of_elements, 13
asn1_object_id_der, 26
asn1_octet_der, 23
asn1_parser2array, 4
asn1_parser2tree, 4
asn1_perror, 19
ASN1_PRINT_ALL, 30
ASN1_PRINT_NAME, 29
ASN1_PRINT_NAME_TYPE, 29
ASN1_PRINT_NAME_TYPE_VALUE, 29
asn1_print_structure, 6
asn1_read_node_value, 11
asn1_read_tag, 17
asn1_read_value, 9
asn1_read_value_type, 10
ASN1_RECURSION, 29
asn1_static_node_t, 39
ASN1_SUCCESS, 27
ASN1_SYNTAX_ERROR, 28
ASN1_TAG_BIT_STRING, 31
ASN1_TAG_BMP_STRING, 32
ASN1_TAG_BOOLEAN, 30
ASN1_TAG_ENUMERATED, 31
ASN1_TAG_ERROR, 28
ASN1_TAG_GENERALIZEDTime, 31
ASN1_TAG_GENERALSTRING, 31
ASN1_TAG_IA5_STRING, 31
ASN1_TAG_IMPLICIT, 28
ASN1_TAG_INTEGER, 30
ASN1_TAG_NULL, 31
ASN1_TAG_NUMERIC_STRING, 31
ASN1_TAG_OBJECT_ID, 31
ASN1_TAG_OCTET_STRING, 30
ASN1_TAG_PRINTABLE_STRING, 31
ASN1_TAG_SEQUENCE, 30
ASN1_TAG_SET, 30
ASN1_TAG_TELETEX_STRING, 31
ASN1_TAG_UNIVERSAL_STRING, 32
ASN1_TAG_UTCTime, 31
ASN1_TAG_UTF8_STRING, 32
ASN1_TAG_VISIBLE_STRING, 32
ASN1_TIME_ENCODING_ERROR, 29
ASN1_TYPE, 38
ASN1_TYPE_EMPTY, 38
ASN1_VALUE_NOT_FOUND, 28
ASN1_VALUE_NOT_VALID, 28

ASN1_VERSION, 27
ASN1_VERSION_MAJOR, 27
ASN1_VERSION_MINOR, 27
ASN1_VERSION_NUMBER, 27
ASN1_VERSION_PATCH, 27
asn1_write_value, 8

N

node_asn, 37
node_asn_struct, 37
node_data_struct, 39

S

static_struct_asn, 38