Libtasn1

Abstract Syntax Notation One (ASN.1) library for the GNU system
for version 4.13, 4 April 2016

Fabio Fiorina
Simon Josefsson
Nikos Mavrogiannopoulos (help-libtasn1@gnu.org)

This manual is for GNU Libtasn1 (version 4.13, 4 April 2016), which is a library for Abstract Syntax Notation One (ASN.1) and Distinguished Encoding Rules (DER) manipulation.

Table of Contents

1 Introduction

This document describes the Libtasn1 library that 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.

The main features of this library are:

- On-line ASN.1 structure management that doesn't require any C code file generation.
- Off-line ASN.1 structure management with C code file generation containing an array.
- Distinguished Encoding Rules (DER) encoding support.
- No limits for INTEGER and ENUMERATED values.
- It's Free Software. Anybody can use, modify, and redistribute the library under the terms of the GNU Lesser General Public License version 2.1 or later. The command line tools, self-tests and build infrastructure are licensed under the GNU General Public License version 3.0 or later.
- Thread-safety. No global variables are used and multiple library handles and session handles may be used in parallel.
- Portability. The code should work on all Unix like operating systems, and Windows. The library itself should be portable to any C89 system, not even POSIX is required.

2 ASN.1 structure handling

2.1 ASN.1 syntax

The parser is case sensitive. The comments begin with `--` and end either with another `--`, or at the end of the respective line, whichever comes first. The C-style `/*`, `*/` comments are not supported.

For an example of the syntax, check the `pkix.asn` file distributed with the library.

ASN.1 definitions must follow the syntax below:

```
definitions_name {<object definition>}

DEFINITIONS <EXPLICIT or IMPLICIT> TAGS ::=

BEGIN

<type and constants definitions>

END
```

The `::=` token must be separate from other elements, so the following declaration is invalid:

```
-- INCORRECT
Version ::=INTEGER
```

The correct form is:

```
Version ::= INTEGER
```

Here is the list of types that the parser can manage:

- `INTEGER`;
- `ENUMERATED`;
- `BOOLEAN`;
- `OBJECT IDENTIFIER`;
- `NULL`;
- `BIT STRING`;
- `OCTET STRING`;
- `UTCTime`;
- `GeneralizedTime`;
- `GeneralString`;
- `NumericString`;
- `IA5String`;
- `TeletexString`;
- `PrintableString`;
- `UniversalString`;
- `BMPString`;

- `UTF8String`;
- `VisibleString`;
- `SEQUENCE`;
- `SEQUENCE OF`;
- `SET`;
- `SET OF`;
- `CHOICE`;
- `ANY`;
- `ANY DEFINED BY`.

This version doesn't handle the `REAL` type. It doesn't support the `AUTOMATIC TAGS` option, and the `EXPORT` and `IMPORT` sections, either.

The `SIZE` constraints are allowed, but no check is done on them.

2.2 Naming

Consider this definition:

```
Example { 1 2 3 4 }

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

Group ::= SEQUENCE {
   id   OBJECT IDENTIFIER,
   value  Value
}

Value ::= SEQUENCE {
   value1  INTEGER,
   value2  BOOLEAN
}

END
```

The notation to access the '`Group`' type of the '`Example`' definition above is '`Example.Group`' (as a NUL-terminated string.) Such strings are used in the functions described below.

Others examples:

- field '`id`' of the '`Group`' type: '`Example.Group.id`';
- field '`value1`' of the '`value`' field of the '`Group`' type: '`Example.Group.value.value1`'.

Elements of structured types unnamed by the respective definition receive the names `?1`, `?2`, and so on.

The `?LAST` name indicates the last element of a `SET OF` or `SEQUENCE OF`.

2.3 Simple parsing

For simple types like `OCTET STRING` the simple parsing functions listed below may be used instead.

- [asn1_decode_simple_der], page 20
- [asn1_encode_simple_der], page 15

2.4 Library Notes

The header file of this library is `libtasn1.h`.

The main type used in it is `asn1_node`, and it's used to store the ASN.1 definitions and structures (instances).

The `NULL` constant can be used for the variable initialization. For example:

```
asn1_node definitions = NULL;
```

Some functions require an `errorDescription` argument of type `char *`, pointing to a pre-allocated buffer of at least `ASN1_MAX_ERROR_DESCRIPTION_SIZE` bytes size (e.g., as in '`char description[ASN1_MAX_ERROR_DESCRIPTION_SIZE];`').

`ASN1_MAX_NAME_SIZE` is the maximum number of characters allowed for an ASN.1 identifier.

2.5 Future developments

- Add functions for a C code file generation containing equivalent data structures (not a single array like now).
- The `REAL` type.

3 Utilities

3.1 Invoking asn1Parser

`asn1Parser` reads a single file with ASN.1 definitions and generates a file with an array to use with libtasn1 functions.

```
Usage:  asn1Parser [options] file

Options:
 -h : shows the help message.
 -v : shows version information and exit.
 -c : checks the syntax only.
 -o file : output file.
 -n name : array name.
```

3.2 Invoking asn1Coding

`asn1Coding` generates a DER encoding from a file with ASN.1 definitions and another one with assignments.

The file with assignments must have this syntax:

```
InstanceName  Asn1Definition

nameString  value

nameString  value
...
```

To specify the field of a `CHOICE` to be used, specify its name as a value to the `CHOICE` element itself. Use `''` to denote the root element itself. (as in the example below.)

The output file is a binary file with the DER encoding.

```
Usage:  asn1Coding [options] file1 file2
 file1 : file with ASN1 definitions.
 file2 : file with assignments.
Options:
 -h : shows the help message.
 -v : shows version information and exit.
 -c : checks the syntax only.
 -o file : output file.
```

For example, consider an ASN.1 definitions file as follows:

```
MYPKIX1 { }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

OtherStruct := SEQUENCE {
```

```
     x       INTEGER,
     y       CHOICE {
         y1 INTEGER,
         y2 OCTET STRING },
}

Dss-Sig-Value ::= SEQUENCE {
     r       INTEGER,
     s       INTEGER,
     other   OtherStruct
     z       INTEGER OPTIONAL,
}

END
```

And a assignments file as follows:

```
dp MYPKIX1.Dss-Sig-Value

r 42
s 47
other.x 66
other.y y1
other.y.y1 15
z (NULL)
```

Running the command below will generate a `assign.out` file, containing the DER encoding of `PKIX1.Dss-Sig-Value`.

```
$ asn1Coding pkix.asn assign.asn1
```

If the root element is of the `CHOICE` type, the assignment file may be like (using the types defined in `pkix.asn`):

```
elt PKIX1Implicit88.GeneralName

''      dNSName
dNSName example.org
```

3.3 Invoking asn1Decoding

`asn1Decoding` generates an ASN.1 structure from a file with ASN.1 definitions and a binary file with a DER encoding.

```
Usage:  asn1Decoding [options] file1 file2 type
 file1 : file with ASN1 definitions.
 file2 : binary file with a DER encoding.
 type : ASN1 definition name.
Options:
 -h : shows the help message.
 -v : shows version information and exit.
 -o file : output file.
```

For example, after generating the `assign.out` file from the example section of the `asn1Coding` command above, the following invocation will decode the DER data.

```
$ asn1Decoding pkix.asn assign.out PKIX1.Dss-Sig-Value
```

4 Function reference

4.1 ASN.1 schema functions

asn1_parser2tree

`int asn1_parser2tree` (*const char* * ***file***, *asn1_node* * ***definitions***, *char* * ***error_desc***) [Function]

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.

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

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 occured 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 whith 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]

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.

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".

Returns: `ASN1_SUCCESS` if the file has a correct syntax and every identifier is known, `ASN1_FILE_NOT_FOUND` if an error occured 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 whith more than `ASN1_MAX_NAME_SIZE` characters.

4.2 ASN.1 field functions

asn1_array2tree

`int asn1_array2tree` (*const asn1_static_node ** ***array***, *asn1_node ** ***definitions***, *char ** ***errorDescription***) [Function]

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.

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

Returns: `ASN1_SUCCESS` if structure was created correctly, `ASN1_ELEMENT_NOT_EMPTY` if * `definitions` not NULL, `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_delete_structure

`int asn1_delete_structure` (*asn1_node ** ***structure***) [Function]

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

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

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***) [Function]

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

flags: additional flags (see `ASN1_DELETE_FLAG`)

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

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***) [Function]

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

element_name: element's name you want to delete.

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

Returns: `ASN1_SUCCESS` if successful, `ASN1_ELEMENT_NOT_FOUND` if the `element_name` was not found.

asn1_create_element

`int asn1_create_element` (*asn1_node* ***definitions***, *const char ** ***source_name***, *asn1_node ** ***element***) [Function]

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.

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

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

Returns: `ASN1_SUCCESS` if creation OK, `ASN1_ELEMENT_NOT_FOUND` if `source_name` is not known.

asn1_print_structure

`void asn1_print_structure` (*FILE ** ***out***, *asn1_node* ***structure***, *const char ** ***name***, *int* ***mode***) [Function]

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` .

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

asn1_number_of_elements

`int asn1_number_of_elements` (*asn1_node* ***element***, *const char ** ***name***, *int ** ***num***) [Function]

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

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

Returns: `ASN1_SUCCESS` if successful, `ASN1_ELEMENT_NOT_FOUND` if `name` is not known, `ASN1_GENERIC_ERROR` if pointer `num` is `NULL` .

asn1_find_structure_from_oid

`const char * asn1_find_structure_from_oid` (*asn1_node* ***definitions***, *const char ** ***oidValue***) [Function]

definitions: ASN1 definitions

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

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

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

asn1_copy_node

`int asn1_copy_node` (*asn1_node* **`dst`**, *const char* * **`dst_name`**, *asn1_node* **`src`**, *const char* * **`src_name`**) [Function]

dst: Destination asn1 node.

dst_name: Field name in destination node.

src: Source asn1 node.

src_name: Field name in source node.

Create a deep copy of a asn1_node variable. That function requires `dst` to be expanded using `asn1_create_element()` .

Returns: Return `ASN1_SUCCESS` on success.

asn1_dup_node

`asn1_node asn1_dup_node` (*asn1_node* **`src`**, *const char* * **`src_name`**) [Function]

src: Source asn1 node.

src_name: Field name in source node.

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

Returns: Return `NULL` on failure.

asn1_write_value

`int asn1_write_value` (*asn1_node* **`node_root`**, *const char* * **`name`**, *const void* * **`ivalue`**, *int* **`len`**) [Function]

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

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=Jannuary 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=Jannuary 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="$\backslash$x01$\backslash$x02$\backslash$x03" , len=3 -> three bytes octet string

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

value="$\backslash$x01$\backslash$x02$\backslash$x03" , len=3 -> three bytes generalstring

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

value="$\backslash$xCF" , 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);

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* **`root`**, *const char ** **`name`**, *void ** **`ivalue`**, *int ** **`len`**) [Function]

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]. Initialy holds the sizeof value.

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.

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.

asn1_read_value_type

`int asn1_read_value_type (asn1_node root, const char * name, void * ivalue, int * len, unsigned int * etype)` [Function]

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]. Initialy holds the sizeof value.

etype: The type of the value read (ASN1_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.

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.

asn1_read_tag

`int asn1_read_tag` (*asn1_node* **`root`**, *const char ** **`name`**, *int ** **`tagValue`**, *int ** **`classValue`**) [Function]

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 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` .

Returns: `ASN1_SUCCESS` if successful, `ASN1_ELEMENT_NOT_FOUND` if `name` is not a valid element.

asn1_read_node_value

`int asn1_read_node_value` (*asn1_node* **`node`**, *asn1_data_node_st ** **`data`**) [Function]

node: pointer to a node.

data: a point to a asn1_data_node_st

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

Returns: `ASN1_SUCCESS` if the node exists.

4.3 DER functions

asn1_length_der

`void asn1_length_der` (*unsigned long int* **`len`**, *unsigned char ** **`der`**, *int ** **`der_len`**) [Function]

len: value to convert.

der: buffer to hold the returned encoding (may be `NULL`).

der_len: number of meaningful bytes of ANS (der[0]..der[der_len-1]).

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` .

asn1_octet_der

`void asn1_octet_der` (*const unsigned char ** **`str`**, *int* **`str_len`**, *unsigned char ** **`der`**, *int ** **`der_len`**) [Function]

str: the input data.

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

der: encoded string returned.

der_len: number of meaningful bytes of DER (der[0]..der[der_len-1]).

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()` .

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`**) [Function]

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 `tl` field

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` .

Returns: `ASN1_SUCCESS` if successful or an error value.

asn1_bit_der

`void asn1_bit_der` (*const unsigned char ** **`str`**, *int* **`bit_len`**, *unsigned char ** **`der`**, *int ** **`der_len`**) [Function]

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]).

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()` .

asn1_der_coding

`int asn1_der_coding` (*asn1_node* **`element`**, *const char ** **`name`**, *void ** **`ider`**, *int ** **`len`**, *char ** **`ErrorDescription`**) [Function]

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], Initialy holds the sizeof of der vector.

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

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

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_get_length_der

`long asn1_get_length_der` (*const unsigned char ** `der`, *int* `der_len`, *int ** `len`) [Function]

der: DER data to decode.

der_len: Length of DER data to decode.

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

Extract a length field from DER data.

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_tag_der

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

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.

Decode the class and TAG from DER code.

Returns: Returns `ASN1_SUCCESS` on success, or an error.

asn1_get_length_ber

`long asn1_get_length_ber` (*const unsigned char ** `ber`, *int* `ber_len`, *int ** `len`) [Function]

ber: BER data to decode.

ber_len: Length of BER data to decode.

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

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.

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

Since: 2.0

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***) [Function]

der: DER data to decode containing the OCTET SEQUENCE.

der_len: The length of the `der` data to decode.

ret_len: Output variable containing the encoded 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 contents of the OCTET SEQUENCE.

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.

Returns: 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***) [Function]

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.

Converts a DER encoded object identifier to its textual form.

Returns: `ASN1_SUCCESS` on success, or an error.

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***) [Function]

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.

Extract a BIT SEQUENCE from DER data.

Returns: `ASN1_SUCCESS` on success, or an error.

asn1_der_decoding2

int asn1_der_decoding2 (*asn1_node * **element**, const void * **ider**, int * **max_ider_len**, unsigned int **flags**, char * **errorDescription***) [Function]

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 * `ider` . 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.

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.

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***) [Function]

element: pointer to an ASN1 structure.

ider: vector that contains the DER encoding.

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

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

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.

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***) [Function]

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.

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.

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`**) [Function]

element: pointer to an ASN1 element

ider: vector that contains the DER encoding.

ider_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])

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()` .

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* **`definitions`**, *asn1_node ** **`element`**) [Function]

definitions: ASN1 definitions

element: pointer to an ASN1 structure

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.

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* **`definitions`**, *asn1_node * * **`element`**, *const char * * **`octetName`**, *const char * * **`objectName`**) [Function]

definitions: ASN1 definitions

element: pointer to an ASN1 structure

octetName: name of the OCTECT STRING field to expand.

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

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.

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_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`**) [Function]

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

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

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`**) [Function]

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`)

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.

Returns: `ASN1_SUCCESS` if successful or an error value.

4.4 Error handling functions

asn1_perror

`void asn1_perror` (*int* ***error***) [Function]

error: is an error returned by a libtasn1 function.

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.

Since: 1.6

asn1_strerror

`const char * asn1_strerror` (*int* ***error***) [Function]

error: is an error returned by a libtasn1 function.

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.

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

Since: 1.6

4.5 Auxilliary functions

asn1_find_node

`asn1_node asn1_find_node` (*asn1_node* ***pointer***, *const char ** ***name***) [Function]

pointer: NODE_ASN element pointer.

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

Searches for an element called `name` starting from `pointer` . The name is composed by differents 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` .

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

asn1_check_version

`const char * asn1_check_version` (*const char ** ***req_version***) [Function]

req_version: Required version number, or `NULL` .

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.

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

Appendix A Copying Information

A.1 GNU Free Documentation License

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Concept Index

Function and Data Index