NAME

asn_get_header, **asn_put_header**, **asn_put_temp_header**, **asn_commit_header**, **asn_get_integer_raw**, **asn_get_integer**, **asn_put_integer**, **asn_get_octetstring_raw**, **asn_get_octetstring**, **asn_put_octetstring**, **asn_get_null_raw**, **asn_get_null**, **asn_put_null**, **asn_put_exception**, **asn_get_objid_raw**, **asn_get_objid**, **asn_put_objid**, **asn_get_sequence**, **asn_get_ipaddress_raw**, **asn_get_ipaddress**, **asn_put_ipaddress**, **asn_get_uint32_raw**, **asn_put_uint32**, **asn_get_counter64_raw**, **asn_put_counter64**, **asn_get_timeticks**, **asn_put_timeticks**, **asn_skip**, **asn_slice_oid**, **asn_append_oid**, **asn_compare_oid**, **asn_is_suboid**, **asn_oid2str_r**, **asn_oid2str** - ASN.1 library for SNMP

LIBRARY

Begemot SNMP library (libbsnmp, -lbsnmp)

SYNOPSIS

#include <bsnmp/asn1.h>

enum asn_err
asn_get_header(*struct asn_buf *buf, u_char *type, asn_len_t *lenp*);

enum asn_err
asn_put_header(*struct asn_buf *buf, u_char type, asn_len_t len*);

enum asn_err
asn_put_temp_header(*struct asn_buf *buf, u_char type, u_char **ptr*);

enum asn_err
asn_commit_header(*struct asn_buf *buf, u_char *ptr*);

enum asn_err
asn_get_integer_raw(*struct asn_buf *buf, asn_len_t len, int32_t *res*);

enum asn_err
asn_get_integer(*struct asn_buf *buf, int32_t *res*);

enum asn_err
asn_put_integer(*struct asn_buf *buf, int32_t arg*);

enum asn_err
asn_get_octetstring_raw(*struct asn_buf *buf, asn_len_t len, u_char *out, u_int *outsize*);

enum asn_err

asn_get_octetstring(*struct asn_buf *buf, u_char *out, u_int *outsize*);

enum asn_err
asn_put_octetstring(*struct asn_buf *buf, const u_char *str, u_int strsize*);

enum asn_err
asn_get_null_raw(*struct asn_buf *buf, asn_len_t len*);

enum asn_err
asn_get_null(*struct asn_buf *buf*);

enum asn_err
asn_put_null(*struct asn_buf *buf*);

enum asn_err
asn_put_exception(*struct asn_buf *buf, u_int type*);

enum asn_err
asn_get_objid_raw(*struct asn_buf *buf, asn_len_t len, struct asn_oid *oid*);

enum asn_err
asn_get_objid(*struct asn_buf *buf, struct asn_oid *oid*);

enum asn_err
asn_put_objid(*struct asn_buf *buf, const struct asn_oid *oid*);

enum asn_err
asn_get_sequence(*struct asn_buf *buf, asn_len_t *lenp*);

enum asn_err
asn_get_ipaddress_raw(*struct asn_buf *buf, asn_len_t len, u_char *ipa*);

enum asn_err
asn_get_ipaddress(*struct asn_buf *buf, u_char *ipa*);

enum asn_err
asn_put_ipaddress(*struct asn_buf *buf, const u_char *ipa*);

enum asn_err
asn_get_uint32_raw(*struct asn_buf *buf, asn_len_t len, u_int32_t *res*);

enum asn_err
asn_put_uint32(*struct asn_buf *buf, u_char type, u_int32_t val*);

enum asn_err
asn_get_counter64_raw(*struct asn_buf *buf, asn_len_t len, u_int64_t *res*);

enum asn_err
asn_put_counter64(*struct asn_buf *buf, u_int64_t val*);

enum asn_err
asn_get_timeticks(*struct asn_buf *buf, u_int32_t *valp*);

enum asn_err
asn_put_timeticks(*struct asn_buf *buf, u_int32_t val*);

enum asn_err
asn_skip(*struct asn_buf *buf, asn_len_t len*);

void
asn_slice_oid(*struct asn_oid *dest, const struct asn_oid *src, u_int from, u_int to*);

void
asn_append_oid(*struct asn_oid *to, const struct asn_oid *from*);

int
asn_compare_oid(*const struct asn_oid *oid1, const struct asn_oid *oid2*);

int
asn_is_suboid(*const struct asn_oid *oid1, const struct asn_oid *oid2*);

*char **
asn_oid2str_r(*const struct asn_oid *oid, char *buf*);

*char **
asn_oid2str(*const struct asn_oid *oid*);

DESCRIPTION

The ASN.1 library contains routines to handle ASN.1 encoding for SNMP. It supports only the restricted form of ASN.1 as required by SNMP. There are two basic structures used throughout the library:

```
/* these restrictions are in the SMI */
#define ASN_MAXID       0xffffffff
#define ASN_MAXOIDLEN 128

/* type of subidentifiers */
typedef u_int32_t asn_subid_t;

struct asn_oid {
        u_int       len;
        asn_subid_t subs[ASN_MAXOIDLEN];
};
```

This structure represents an OID with the restrictions defined in the SNMP SMI. *len* holds the current length of the OID and *subs* holds the elements of the OID.

```
struct asn_buf {
        union {
                u_char     *ptr;
                const u_char *cptr;
        }          asn_u;
        size_t     asn_len;
};
#define asn_cptr    asn_u.cptr
#define asn_ptr     asn_u.ptr
```

This structure is used to encode and decode ASN.1. It describes the output buffer for encoding routines and the input buffer for decoding routines. For encoding *asn_len* holds the number of remaining free octets in the buffer. The first free byte is pointed to by *asn_ptr*. For decoding *asn_len* holds the number of remaining bytes to decode. The next byte to decode is pointed to by *asn_cptr*.

Most of the functions return an error code *enum asn_error*:

```
enum asn_err {
        /* conversion was ok */
        ASN_ERR_OK      = 0,
        /* conversion failed and stopped */
        ASN_ERR_FAILED          = 1 | 0x1000,
        /* length field bad, value skipped */
        ASN_ERR_BADLEN          = 2,
        /* out of buffer, stopped */
```

```
        ASN_ERR_EOBUF       = 3 | 0x1000,
        /* length ok, but value is out of range */
        ASN_ERR_RANGE       = 4,
        /* not the expected tag, stopped */
        ASN_ERR_TAG    = 5 | 0x1000,
};
#define ASN_ERR_STOPPED(E) (((E) & 0x1000) != 0)
```

If **ASN_ERR_STOPPED**() returns true, the error was fatal and processing has stopped at the point of error.

The function **asn_get_header**() reads the next header from the input octet stream. It returns the tag in the variable pointed to by *type* (note that only single byte tags are supported) and the decoded length field in the value pointed to by *lenp* (this is restricted to a unsigned 32-bit value). All errors in this function are fatal and stop processing.

The function **asn_put_header**() writes an ASN.1 header. *type* is the tag to write and is restricted to one byte tags (i.e., tags lesser or equal than 0x30). *len* is the length of the value and is restricted to 16-bit.

The functions **asn_put_temp_header**() and **asn_commit_header**() are used to write a header when the length of the value is not known in advance, for example, for sequences. **asn_put_temp_header**() writes a header with the given tag *type* and space for the maximum supported length field and sets the pointer pointed to by *ptr* to the begin of this length field. This pointer must then be fed into **asn_commit_header**() directly after writing the value to the buffer. The function will compute the length, insert it into the right place and shift the value if the resulting length field is shorter than the estimated one.

The function **asn_get_integer_raw**() is used to decode a signed integer value (32-bit). It assumes, that the header of the integer has been decoded already. *len* is the length obtained from the ASN.1 header and the integer will be returned in the value pointed to by *res*.

The function **asn_get_integer**() decodes a complete 32-bit signed integer including the header. If the tag is wrong ASN_ERR_TAG is returned. The function **asn_put_integer**() encodes a 32-bit signed integer.

The function **asn_get_octetstring_raw**() decodes the value field of an ASN.1 octet string. The length obtained from the header must be fed into the *len* argument and *out* must point to a buffer to receive the octet string. On entry to the function *outsize* must point to the size of the buffer. On exit *outsize* will point to the number of octets decoded (if no error occurs this will be equal to *len)*. The function **asn_get_octetstring**() decodes an octetstring including the header. *out* must point to a buffer to receive the string, *outsize* must point to the size of the buffer. On exit of the function *outsize* will point to the

number of octets decoded. The function **asn_put_octetstring**() encodes an octetstring (including the header). *str* points to the string to encode and *strsize* is the length of the string (the string may contain embedded NULs).

The function **asn_get_null_raw**() decodes a null value. *len* is the length obtained from the header and must be 0. The function **asn_get_null**() decodes a null including the header and the function **asn_put_null**() encodes a null.

The function **asn_put_exception**() is used to encode an SNMPv2 exception. The exception type is *type*.

The function **asn_get_objid_raw**() is used to decode an OID value. *len* must be the value length obtained from the header and *oid* will receive the decoded OID. The function **asn_get_objid**() decodes a complete OID (including the header) and the function **asn_put_objid**() encodes a complete OID.

The function **asn_get_sequence**() decodes a sequence header. The length of the sequence value will be stored in the value pointed to by *lenp*.

The function **asn_get_ipaddress_raw**() decodes an IP address value. *len* is the length from the header and must be 4. *ipa* will receive the decoded IP address and must point to a buffer of at least four bytes. The function **asn_get_ipaddress**() decodes a complete IP address (including the header) and **asn_put_ipaddress**() encodes an IP address.

The function **asn_get_uint32_raw**() decodes an unsigned 32-bit integer value. *len* is the length from the header and *res* will get the decoded value. The function **asn_put_uint32**() encodes an unsigned 32-bit integer value and inserts the tag given in *type* into the header.

The function **asn_get_counter64_raw**() decodes an unsigned 64-bit integer value. *len* must be the value length from the header. The resulting value is stored into the variable pointed to by *res*. The function **asn_put_counter64**() encodes a complete unsigned 64-bit value.

The function **asn_get_timeticks**() decodes an ASN.1 object of type TIMETICKS and the function **asn_put_timeticks**() encodes such an object.

The function **asn_skip**() can be used to skip *len* bytes in the input buffer.

The function **asn_slice_oid**() splits a part out from an OID. It takes all the subids from the OID pointed to by *src* starting with the subid at position *from* (the first subid being subid 0) up to, but not including, subid *to* and generates a new OID in *dest*. If *to* is less or equal to *from* the resulting OID will have a length of zero.

The function **asn_append_oid**() appends the OID *from* to the OID *to* given that the resulting OID is not too long. If the maximum length is exceeded the result is undefined.

The function **asn_compare_oid**() compares two oids and returns the values -1, 0 or +1 when *oid1* is lesser than, equal, or larger than *oid2* resp.

The function **asn_is_suboid**() returns 1 if *oid1* is equal to the leading part of *oid2*. It returns 0 otherwise.

The function **asn_oid2str_r**() makes a printable string from *oid*. The buffer pointed to by *str* must be large enough to hold the result. The constant ASN_OIDSTRLEN is defined to be the length of the maximum string generated by this function (including the trailing NUL). The function **asn_oid2str**() makes a printable string from *oid* into a private buffer that is overwritten by each call.

DIAGNOSTICS

When an error occurs in any of the function the function pointed to by the global pointer

```
extern void (*asn_error)(const struct asn_buf *, const char *, ...);
```

is called with the current buffer (this may be NULL) and a printf(3) style format string. There is a default error handler in the library that prints a message starting with ‘ASN.1:’ followed by the error message and an optional dump of the buffer.

SEE ALSO

gensnmptree(1), bsnmpd(1), bsnmpagent(3), bsnmpclient(3), bsnmplib(3)

STANDARDS

This implementation conforms to the applicable IETF RFCs and ITU-T recommendations.

AUTHORS

Hartmut Brandt <harti@FreeBSD.org>