NAME

ber_alloc_t, ber_flush, ber_flush2, ber_printf, ber_put_int, ber_put_enum, ber_put_ostring, ber_put_string, ber_put_null, ber_put_boolean, ber_put_bitstring, ber_start_seq, ber_start_set, ber_put_seq, ber_put_set - OpenLDAP LBER simplified Basic Encoding Rules library routines for encoding

LIBRARY

OpenLDAP LBER (liblber, -llber)

SYNOPSIS

#include <lber.h>

BerElement *ber_alloc_t(int *options***);**

int ber_flush(Sockbuf ***sb*, BerElement ***ber***, int** *freeit***);**

int ber_flush2(Sockbuf ***sb*, BerElement ***ber***, int** *freeit***);**

int ber_printf(BerElement ***ber*, const char ***fmt***, ...);**

int ber_put_int(BerElement ***ber*, ber_int_t** *num***, ber_tag_t** *tag***);**

int ber_put_enum(BerElement ***ber*, ber_int_t** *num***, ber_tag_t** *tag***);**

int ber_put_ostring(BerElement ***ber*, const char ***str***, ber_len_t** *len***, ber_tag_t** *tag***);**

int ber_put_string(BerElement ***ber*, const char ***str***, ber_tag_t** *tag***);**

int ber_put_null(BerElement ***ber*, ber_tag_t** *tag***);**

int ber_put_boolean(BerElement ***ber*, ber_int_t** *bool***, ber_tag_t** *tag***);**

int ber_put_bitstring(BerElement ***ber*, const char ***str***, ber_len_t** *blen***, ber_tag_t** *tag***);**

int ber_start_seq(BerElement ***ber*, ber_tag_t** *tag***);**

int ber_start_set(BerElement ***ber*, ber_tag_t** *tag***);**

int ber_put_seq(BerElement ***ber*);**

int ber_put_set(BerElement ****ber*);**

DESCRIPTION

These routines provide a subroutine interface to a simplified implementation of the Basic Encoding Rules of ASN.1. The version of BER these routines support is the one defined for the LDAP protocol. The encoding rules are the same as BER, except that only definite form lengths are used, and bitstrings and octet strings are always encoded in primitive form. This man page describes the encoding routines in the lber library. See **lber-decode**(3) for details on the corresponding decoding routines. Consult **lber-types**(3) for information about types, allocators, and deallocators.

Normally, the only routines that need to be called by an application are **ber_alloc_t**() to allocate a BER element for encoding, **ber_printf**() to do the actual encoding, and **ber_flush2**() to actually write the element. The other routines are provided for those applications that need more control than **ber_printf**() provides. In general, these routines return the length of the element encoded, or -1 if an error occurred.

The **ber_alloc_t**() routine is used to allocate a new BER element. It should be called with an argument of LBER_USE_DER.

The **ber_flush2**() routine is used to actually write the element to a socket (or file) descriptor, once it has been fully encoded (using **ber_printf**() and friends). See **lber-sockbuf**(3) for more details on the Sockbuf implementation of the *sb* parameter. If the *freeit* parameter is non-zero, the supplied *ber* will be freed. If *LBER_FLUSH_FREE_ON_SUCCESS* is used, the *ber* is only freed when successfully flushed, otherwise it is left intact; if *LBER_FLUSH_FREE_ON_ERROR* is used, the *ber* is only freed when an error occurs, otherwise it is left intact; if *LBER_FLUSH_FREE_ALWAYS* is used, the *ber* is freed anyway. This function differs from the original **ber_flush**(3) function, whose behavior corresponds to that indicated for *LBER_FLUSH_FREE_ON_SUCCESS*. Note that in the future, the behavior of **ber_flush**(3) with *freeit* non-zero might change into that of **ber_flush2**(3) with *freeit* set to *LBER_FLUSH_FREE_ALWAYS*.

The **ber_printf**() routine is used to encode a BER element in much the same way that **sprintf**(3) works. One important difference, though, is that some state information is kept with the *ber* parameter so that multiple calls can be made to **ber_printf**() to append things to the end of the BER element. **Ber_printf**() writes to *ber*, a pointer to a BerElement such as returned by **ber_alloc_t**(). It interprets and formats its arguments according to the format string *fmt*. The format string can contain the following characters:

b Boolean. An ber_int_t parameter should be supplied. A boolean element is output.

e Enumeration. An ber_int_t parameter should be supplied. An enumeration element is output.

i Integer. An ber_int_t parameter should be supplied. An integer element is output.

B Bitstring. A char * pointer to the start of the bitstring is supplied, followed by the number of bits in the bitstring. A bitstring element is output.

n Null. No parameter is required. A null element is output.

o Octet string. A char * is supplied, followed by the length of the string pointed to. An octet string element is output.

O Octet string. A struct berval * is supplied. An octet string element is output.

s Octet string. A null-terminated string is supplied. An octet string element is output, not including the trailing NULL octet.

t Tag. A ber_tag_t specifying the tag to give the next element is provided. This works across calls.

v Several octet strings. A null-terminated array of char *'s is supplied. Note that a construct like '{v}' is required to get an actual SEQUENCE OF octet strings.

V Several octet strings. A null-terminated array of struct berval *'s is supplied. Note that a construct like '{V}' is required to get an actual SEQUENCE OF octet strings.

W Several octet strings. An array of struct berval's is supplied. The array is terminated by a struct berval with a NULL bv_val. Note that a construct like '{W}' is required to get an actual SEQUENCE OF octet strings.

{ Begin sequence. No parameter is required.

} End sequence. No parameter is required.

[Begin set. No parameter is required.

] End set. No parameter is required.

The **ber_put_int**() routine writes the integer element *num* to the BER element *ber*.

The **ber_put_enum**() routine writes the enumeration element *num* to the BER element *ber*.

The **ber_put_boolean**() routine writes the boolean value given by *bool* to the BER element.

The **ber_put_bitstring**() routine writes *blen* bits starting at *str* as a bitstring value to the given BER element. Note that *blen* is the length *in bits* of the bitstring.

The **ber_put_ostring**() routine writes *len* bytes starting at *str* to the BER element as an octet string.

The **ber_put_string**() routine writes the null-terminated string (minus the terminating ’ ’) to the BER element as an octet string.

The **ber_put_null**() routine writes a NULL element to the BER element.

The **ber_start_seq**() routine is used to start a sequence in the BER element. The **ber_start_set**() routine works similarly. The end of the sequence or set is marked by the nearest matching call to **ber_put_seq**() or **ber_put_set**(), respectively.

EXAMPLES

Assuming the following variable declarations, and that the variables have been assigned appropriately, an lber encoding of the following ASN.1 object:

```
AlmostASearchRequest := SEQUENCE {
    baseObject      DistinguishedName,
    scope           ENUMERATED {
        baseObject    (0),
        singleLevel   (1),
        wholeSubtree  (2)
    },
    derefAliases    ENUMERATED {
        neverDerefaliases   (0),
        derefInSearching    (1),
        derefFindingBaseObj (2),
        alwaysDerefAliases  (3)
    },
    sizelimit       INTEGER (0 .. 65535),
    timelimit       INTEGER (0 .. 65535),
    attrsOnly       BOOLEAN,
    attributes      SEQUENCE OF AttributeType
}
```

can be achieved like so:

```
int rc;
ber_int_t    scope, ali, size, time, attrsonly;
char   *dn, **attrs;
BerElement *ber;

/* ... fill in values ... */

ber = ber_alloc_t( LBER_USE_DER );

if ( ber == NULL ) {
        /* error */
}

rc = ber_printf( ber, "{siiiib{v}}", dn, scope, ali,
    size, time, attrsonly, attrs );

if( rc == -1 ) {
        /* error */
} else {
        /* success */
}
```

ERRORS

If an error occurs during encoding, generally these routines return -1.

NOTES

The return values for all of these functions are declared in the <lber.h> header file.

SEE ALSO

lber-decode(3), **lber-memory**(3), **lber-sockbuf**(3), **lber-types**(3)

ACKNOWLEDGEMENTS

OpenLDAP Software is developed and maintained by The OpenLDAP Project <http://www.openldap.org/>. **OpenLDAP Software** is derived from the University of Michigan LDAP 3.3 Release.