NAME

ber_get_next, ber_skip_tag, ber_peek_tag, ber_scanf, ber_get_int, ber_get_enum, ber_get_stringb, ber_get_stringa, ber_get_stringal, ber_get_stringbv, ber_get_null, ber_get_boolean, ber_get_bitstring, ber_first_element, ber_next_element - OpenLDAP LBER simplified Basic Encoding Rules library routines for decoding

LIBRARY

OpenLDAP LBER (liblber, -llber)

SYNOPSIS

#include <lber.h>

ber_tag_t ber_get_next(Sockbuf ****sb*, ber_len_t ****len***, BerElement ****ber***);**

ber_tag_t ber_skip_tag(BerElement ****ber*, ber_len_t ****len***);**

ber_tag_t ber_peek_tag(BerElement ****ber*, ber_len_t ****len***);**

ber_tag_t ber_scanf(BerElement ****ber*, const char ****fmt***, ...);**

ber_tag_t ber_get_int(BerElement ****ber*, ber_int_t ****num***);**

ber_tag_t ber_get_enum(BerElement ****ber*, ber_int_t ****num***);**

ber_tag_t ber_get_stringb(BerElement ****ber*, char ****buf***, ber_len_t ****len***);**

ber_tag_t ber_get_stringa(BerElement ****ber*, char *****buf***);**

ber_tag_t ber_get_stringal(BerElement ****ber*, struct berval *****bv***);**

ber_tag_t ber_get_stringbv(BerElement ****ber*, struct berval ****bv***, int** *alloc***);**

ber_tag_t ber_get_null(BerElement ****ber*);**

ber_tag_t ber_get_boolean(BerElement ****ber*, ber_int_t ****bool***);**

ber_tag_t ber_get_bitstringa(BerElement ****ber*, char *****buf***, ber_len_t ****blen***);**

ber_tag_t ber_first_element(BerElement ****ber*, ber_len_t ****len***, char *****cookie***);**

ber_tag_t ber_next_element(BerElement ****ber*, ber_len_t ****len***, const char ****cookie***);**

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 decoding routines in the lber library. See **lber-encode**(3) for details on the corresponding encoding 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_get_next**() to get the next BER element and **ber_scanf**() to do the actual decoding. In some cases, **ber_peek_tag**() may also need to be called in normal usage. The other routines are provided for those applications that need more control than **ber_scanf**() provides. In general, these routines return the tag of the element decoded, or LBER_ERROR if an error occurred.

The **ber_get_next**() routine is used to read the next BER element from the given Sockbuf, *sb*. It strips off and returns the leading tag, strips off and returns the length of the entire element in *len*, and sets up *ber* for subsequent calls to **ber_scanf**() et al to decode the element. See **lber-sockbuf**(3) for details of the Sockbuf implementation of the *sb* parameter.

The **ber_scanf**() routine is used to decode a BER element in much the same way that **scanf**(3) works. It reads from *ber*, a pointer to a BerElement such as returned by **ber_get_next**(), interprets the bytes according to the format string *fmt*, and stores the results in its additional arguments. The format string contains conversion specifications which are used to direct the interpretation of the BER element. The format string can contain the following characters.

a Octet string. A char ** should be supplied. Memory is allocated, filled with the contents of the octet string, null-terminated, and returned in the parameter. The caller should free the returned string using **ber_memfree**().

A Octet string. A variant of "**a**". A char ** should be supplied. Memory is allocated, filled with the contents of the octet string, null-terminated, and returned in the parameter, unless a zero-length string would result; in that case, the arg is set to NULL. The caller should free the returned string using **ber_memfree**().

s Octet string. A char * buffer should be supplied, followed by a pointer to a ber_len_t initialized to the size of the buffer. Upon return, the null-terminated octet string is put into the buffer, and the ber_len_t is set to the actual size of the octet string.

O Octet string. A struct ber_val ** should be supplied, which upon return points to a dynamically allocated struct berval containing the octet string and its length. The caller should free the returned structure using **ber_bvfree**().

o Octet string. A struct ber_val * should be supplied, which upon return contains the dynamically allocated octet string and its length. The caller should free the returned octet string using **ber_memfree**().

m Octet string. A struct ber_val * should be supplied, which upon return contains the octet string and its length. The string resides in memory assigned to the BerElement, and must not be freed by the caller.

b Boolean. A pointer to a ber_int_t should be supplied.

e Enumeration. A pointer to a ber_int_t should be supplied.

i Integer. A pointer to a ber_int_t should be supplied.

B Bitstring. A char ** should be supplied which will point to the dynamically allocated bits, followed by a ber_len_t *, which will point to the length (in bits) of the bitstring returned.

n Null. No parameter is required. The element is simply skipped if it is recognized.

v Sequence of octet strings. A char *** should be supplied, which upon return points to a dynamically allocated null-terminated array of char *'s containing the octet strings. NULL is returned if the sequence is empty. The caller should free the returned array and octet strings using **ber_memvfree**().

V Sequence of octet strings with lengths. A struct berval *** should be supplied, which upon return points to a dynamically allocated null-terminated array of struct berval *'s containing the octet strings and their lengths. NULL is returned if the sequence is empty. The caller should free the returned structures using **ber_bvecfree**().

W Sequence of octet strings with lengths. A BerVarray * should be supplied, which upon return points to a dynamically allocated array of struct berval's containing the octet strings and their lengths. The array is terminated by a struct berval with a NULL bv_val string pointer. NULL is returned if the sequence is empty. The caller should free the returned structures using **ber_bvarray_free**().

M Sequence of octet strings with lengths. This is a generalized form of the previous three

formats. A void ** (ptr) should be supplied, followed by a ber_len_t * (len) and a ber_len_t (off). Upon return (ptr) will point to a dynamically allocated array whose elements are all of size (*len). A struct berval will be filled starting at offset (off) in each element. The strings in each struct berval reside in memory assigned to the BerElement and must not be freed by the caller. The array is terminated by a struct berval with a NULL bv_val string pointer. NULL is returned if the sequence is empty. The number of elements in the array is also stored in (*len) on return. The caller should free the returned array using **ber_memfree**().

l Length of the next element. A pointer to a ber_len_t should be supplied.

t Tag of the next element. A pointer to a ber_tag_t should be supplied.

T Skip element and return its tag. A pointer to a ber_tag_t should be supplied.

x Skip element. The next element is skipped.

{ Begin sequence. No parameter is required. The initial sequence tag and length are skipped.

} End sequence. No parameter is required and no action is taken.

[Begin set. No parameter is required. The initial set tag and length are skipped.

] End set. No parameter is required and no action is taken.

The **ber_get_int**() routine tries to interpret the next element as an integer, returning the result in *num*. The tag of whatever it finds is returned on success, LBER_ERROR (-1) on failure.

The **ber_get_stringb**() routine is used to read an octet string into a preallocated buffer. The *len* parameter should be initialized to the size of the buffer, and will contain the length of the octet string read upon return. The buffer should be big enough to take the octet string value plus a terminating NULL byte.

The **ber_get_stringa**() routine is used to dynamically allocate space into which an octet string is read. The caller should free the returned string using **ber_memfree().**

The **ber_get_stringal**() routine is used to dynamically allocate space into which an octet string and its length are read. It takes a struct berval **, and returns the result in this parameter. The caller should free the returned structure using **ber_bvfree().**

The **ber_get_stringbv**() routine is used to read an octet string and its length into the provided struct

berval *. If the *alloc* parameter is zero, the string will reside in memory assigned to the BerElement, and must not be freed by the caller. If the *alloc* parameter is non-zero, the string will be copied into dynamically allocated space which should be returned using **ber_memfree**().

The **ber_get_null**() routine is used to read a NULL element. It returns the tag of the element it skips over.

The **ber_get_boolean**() routine is used to read a boolean value. It is called the same way that **ber_get_int**() is called.

The **ber_get_enum**() routine is used to read a enumeration value. It is called the same way that **ber_get_int**() is called.

The **ber_get_bitstringa**() routine is used to read a bitstring value. It takes a char ** which will hold the dynamically allocated bits, followed by an ber_len_t *, which will point to the length (in bits) of the bitstring returned. The caller should free the returned string using **ber_memfree**().

The **ber_first_element**() routine is used to return the tag and length of the first element in a set or sequence. It also returns in *cookie* a magic cookie parameter that should be passed to subsequent calls to ber_next_element(), which returns similar information.

EXAMPLES

Assume the variable *ber* contains a lightweight BER 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
}
```

The element can be decoded using **ber_scanf**() as follows.

```
ber_int_t    scope, deref, size, time, attrsonly;
char   *dn, **attrs;
ber_tag_t tag;

tag = ber_scanf( ber, "{aeeiib{v}}",
   &dn, &scope, &deref,
   &size, &time, &attrsonly, &attrs );

if( tag == LBER_ERROR ) {
      /* error */
} else {
      /* success */
}

ber_memfree( dn );
ber_memvfree( attrs );
```

ERRORS

If an error occurs during decoding, generally these routines return LBER_ERROR ((ber_tag_t)-1).

NOTES

The return values for all of these functions are declared in the **<lber.h>** header file. Some routines may dynamically allocate memory which must be freed by the caller using supplied deallocation routines.

SEE ALSO

lber-encode(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.