

NAME

snmp_value_free, **snmp_value_parse**, **snmp_value_copy**, **snmp_pdu_free**, **snmp_pdu_decode**, **snmp_pdu_encode**, **snmp_pdu_decode_header**, **snmp_pdu_decode_scoped**, **snmp_pdu_decode_secmode**, **snmp_pdu_init_secparams**, **snmp_pdu_dump**, **snmp_passwd_to_keys**, **snmp_get_local_keys**, **snmp_calc_keychange**, **TRUTH_MK**, **TRUTH_GET**, **TRUTH_OK** - SNMP decoding and encoding library

LIBRARY

Begemot SNMP library (libbsnmp, -lbsnmp)

SYNOPSIS

```
#include <bsnmp/asn1.h>
```

```
#include <bsnmp/snmp.h>
```

void

```
snmp_value_free(struct snmp_value *value);
```

int

```
snmp_value_parse(const char *buf, enum snmp_syntax, union snmp_values *value);
```

int

```
snmp_value_copy(struct snmp_value *to, const struct snmp_value *from);
```

void

```
snmp_pdu_free(struct snmp_pdu *value);
```

enum snmp_code

```
snmp_pdu_decode(struct asn_buf *buf, struct snmp_pdu *pdu, int32_t *ip);
```

enum snmp_code

```
snmp_pdu_encode(struct snmp_pdu *pdu, struct asn_buf *buf);
```

enum snmp_code

```
snmp_pdu_decode_header(struct snmp_pdu *pdu, struct asn_buf *buf);
```

enum snmp_code

```
snmp_pdu_decode_scoped(struct asn_buf *buf, struct snmp_pdu *pdu, int32_t *ip);
```

enum snmp_code

```
snmp_pdu_decode_secmode(struct asn_buf *buf, struct snmp_pdu *pdu);
```

void

snmp_pdu_init_secparams(*struct snmp_pdu *pdu*);

void

snmp_pdu_dump(*const struct snmp_pdu *pdu*);

enum snmp_code

snmp_passwd_to_keys(*struct snmp_user *user, char *passwd*);

enum snmp_code

snmp_get_local_keys(*struct snmp_user *user, uint8_t *eid, uint32_t elen*);

enum snmp_code

snmp_calc_keychange(*struct snmp_user *user, uint8_t *keychange*);

int

TRUTH_MK(*F*);

int

TRUTH_GET(*T*);

int

TRUTH_OK(*T*);

DESCRIPTION

The SNMP library contains routines to handle SNMP version 1, 2 and 3 PDUs. There are several basic structures used throughout the library:

```

struct snmp_value {
    struct asn_oid          var;
    enum snmp_syntax syntax;
    union snmp_values {
        int32_t          integer; /* also integer32 */
        struct {
            u_int          len;
            u_char          *octets;
        }                octetstring;
        struct asn_oid    oid;
        u_char            ipaddress[4];
        uint32_t          uint32; /* also gauge32, counter32,

```

```

                                unsigned32, timeticks */
        uint64_t      counter64;
    }                v;
};

```

This structure represents one variable binding from an SNMP PDU. The field *var* is the ASN.1 of the variable that is bound. *syntax* contains either the syntax code of the value or an exception code for SNMPv2 and may be one of:

```

enum snmp_syntax {
    SNMP_SYNTAX_NULL      = 0,
    SNMP_SYNTAX_INTEGER, /* == INTEGER32 */
    SNMP_SYNTAX_OCTETSTRING,
    SNMP_SYNTAX_OID,
    SNMP_SYNTAX_IPADDRESS,
    SNMP_SYNTAX_COUNTER,
    SNMP_SYNTAX_GAUGE, /* == UNSIGNED32 */
    SNMP_SYNTAX_TIMETICKS,

    /* v2 additions */
    SNMP_SYNTAX_COUNTER64,
    /* exceptions */
    SNMP_SYNTAX_NOSUCHOBJECT,
    SNMP_SYNTAX_NOSUCHINSTANCE,
    SNMP_SYNTAX_ENDOFMIBVIEW,
};

```

The field *v* holds the actual value depending on *syntax*. Note, that if *syntax* is `SNMP_SYNTAX_OCTETSTRING` and *v.octetstring.len* is not zero, *v.octetstring.octets* points to a string allocated by `malloc(3)`.

```

#define  SNMP_ENGINE_ID_SIZ      32

struct snmp_engine {
    uint8_t      engine_id[SNMP_ENGINE_ID_SIZ];
    uint32_t     engine_len;
    int32_t      engine_boots;
    int32_t      engine_time;
    int32_t      max_msg_size;
};

```

This structure represents an SNMP engine as specified by the SNMP Management Architecture described in RFC 3411.

```
#define  SNMP_ADM_STR32_SIZ      (32 + 1)
#define  SNMP_AUTH_KEY_SIZ     40
#define  SNMP_PRIV_KEY_SIZ     32

enum snmp_usm_level {
    SNMP_noAuthNoPriv = 1,
    SNMP_authNoPriv = 2,
    SNMP_authPriv = 3
};

struct snmp_user {
    char                sec_name[SNMP_ADM_STR32_SIZ];
    enum snmp_authentication  auth_proto;
    enum snmp_privacy        priv_proto;
    uint8_t             auth_key[SNMP_AUTH_KEY_SIZ];
    uint8_t             priv_key[SNMP_PRIV_KEY_SIZ];
};
```

This structure represents an SNMPv3 user as specified by the User-based Security Model (USM) described in RFC 3414. The field *sec_name* is a human readable string containing the security user name. *auth_proto* contains the id of the authentication protocol in use by the user and may be one of:

```
enum snmp_authentication {
    SNMP_AUTH_NOAUTH = 0,
    SNMP_AUTH_HMAC_MD5,
    SNMP_AUTH_HMAC_SHA
};
```

priv_proto contains the id of the privacy protocol in use by the user and may be one of:

```
enum snmp_privacy {
    SNMP_PRIV_NOPRIV = 0,
    SNMP_PRIV_DES = 1,
    SNMP_PRIV_AES
};
```

auth_key and *priv_key* contain the authentication and privacy keys for the user.

```
#define  SNMP_COMMUNITY_MAXLEN  128
```

```

#define SNMP_MAX_BINDINGS          100
#define  SNMP_CONTEXT_NAME_SIZ      (32 + 1)
#define  SNMP_TIME_WINDOW          150

#define  SNMP_USM_AUTH_SIZE         12
#define  SNMP_USM_PRIV_SIZE         8

#define  SNMP_MSG_AUTH_FLAG         0x1
#define  SNMP_MSG_PRIV_FLAG         0x2
#define  SNMP_MSG_REPORT_FLAG       0x4

#define  SNMP_MPM_SNMP_V1           0
#define  SNMP_MPM_SNMP_V2c          1
#define  SNMP_MPM_SNMP_V3           3

struct snmp_pdu {
    char                community[SNMP_COMMUNITY_MAXLEN + 1];
    enum snmp_version   version;
    u_int               type;

    /* SNMPv3 PDU header fields */
    int32_t             identifier;
    uint8_t             flags;
    int32_t             security_model;
    struct snmp_engine  engine;

    /* Associated USM user parameters */
    struct snmp_user    user;
    uint8_t             msg_digest[SNMP_USM_AUTH_SIZE];
    uint8_t             msg_salt[SNMP_USM_PRIV_SIZE];

    /* View-based Access Model */
    uint32_t            context_engine_len;
    uint8_t             context_engine[SNMP_ENGINE_ID_SIZ];
    char               context_name[SNMP_CONTEXT_NAME_SIZ];

    /* trap only */
    struct asn_oid      enterprise;
    u_char              agent_addr[4];
    int32_t            generic_trap;

```

```

int32_t          specific_trap;
uint32_t        time_stamp;

/* others */
int32_t          request_id;
int32_t          error_status;
int32_t          error_index;

/* fixes for encoding */
size_t          outer_len;
size_t          scoped_len;
u_char          *outer_ptr;
u_char          *digest_ptr;
u_char          *encrypted_ptr;
u_char          *scoped_ptr;
u_char          *pdu_ptr;
u_char          *vars_ptr;

struct snmp_value bindings[SNMP_MAX_BINDINGS];
u_int           nbindings;
};

```

This structure contains a decoded SNMP PDU. *version* is one of

```

enum snmp_version {
    SNMP_Verr = 0,
    SNMP_V1 = 1,
    SNMP_V2c,
    SNMP_V3
};

```

and *type* is the type of the PDU. *security_model* is the security model used for SNMPv3 PDUs. The only supported value currently is 3 (User-based Security Model). Additional values for any, unknown, SNMPv1 and SNMPv2c security models are also enumerated

```

enum snmp_secmodel {
    SNMP_SECMODEL_ANY = 0,
    SNMP_SECMODEL_SNMPv1 = 1,
    SNMP_SECMODEL_SNMPv2c = 2,
    SNMP_SECMODEL_USM = 3,
    SNMP_SECMODEL_UNKNOWN
};

```

```
};
```

The function **snmp_value_free()** is used to free all the dynamic allocated contents of an SNMP value. It does not free the structure pointed to by *value* itself.

The function **snmp_value_parse()** parses the ASCII representation of an SNMP value into its binary form. This function is mainly used by the configuration file reader of `bsnmpd(1)`.

The function **snmp_value_copy()** makes a deep copy of the value pointed to by *from* to the structure pointed to by *to*. It assumes that *to* is uninitialized and will overwrite its previous contents. It does not itself allocate the structure pointed to by *to*.

The function **snmp_pdu_free()** frees all the dynamically allocated components of the PDU. It does not itself free the structure pointed to by *pdu*.

The function **snmp_pdu_decode()** decodes the PDU pointed to by *buf* and stores the result into *pdu*. If an error occurs in a variable binding the (1 based) index of this binding is stored in the variable pointed to by *ip*.

The function **snmp_pdu_encode()** encodes the PDU *pdu* into the an octetstring in buffer, and if authentication and privacy are used, calculates a message digest and encrypts the PDU data in the buffer *buf*.

The function **snmp_pdu_decode_header()** decodes the header of the PDU pointed to by *buf*. The uncoded PDU contents remain in the buffer.

The function **snmp_pdu_decode_scoped()** decodes the scoped PDU pointed to by *buf*.

The function **snmp_pdu_decode_secmode()** verifies the authentication parameter contained in the PDU (if present) and if the PDU is encrypted, decrypts the PDU contents pointed to by *buf*. If successful, a plain text scoped PDU is stored in the buffer.

The function **snmp_pdu_init_secparams()** calculates the initialization vector for the privacy protocol in use before the PDU pointed to by *pdu* may be encrypted or decrypted.

The function **snmp_pdu_dump()** dumps the PDU in a human readable form by calling **snmp_printf()**.

The function **snmp_passwd_to_keys()** calculates a binary private authentication key corresponding to a plain text human readable password string. The calculated key is placed in the *auth_key* field of the *user*.

The function **snmp_get_local_keys()** calculates a localized authentication and privacy keys for a specified SNMPv3 engine. The calculated keys are placed in the *auth_key* and *priv_key* fields of the *user*.

The function **snmp_calc_keychange()** calculates a binary key change octet string based on the contents of an old and a new binary localized key. The result is placed in the buffer pointer to by *keychange* and may be used by an SNMPv3 user who wishes to change his/her password or localized key.

The function **TRUTH_MK()** takes a C truth value (zero or non-zero) and makes an SNMP truth value (2 or 1). The function **TRUTH_GET()** takes an SNMP truth value and makes a C truth value (0 or 1). The function **TRUTH_OK()** checks, whether its argument is a legal SNMP truth value.

DIAGNOSTICS

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

```
extern void (*snmp_error)(const char *, ...);
```

with a `printf(3)` style format string. There is a default error handler in the library that prints a message starting with 'SNMP:' followed by the error message to standard error.

The function pointed to by

```
extern void (*snmp_printf)(const char *, ...);
```

is called by the **snmp_pdu_dump()** function. The default handler is `printf(3)`.

ERRORS

snmp_pdu_decode() will return one of the following return codes:

[SNMP_CODE_OK]

Success.

[SNMP_CODE_FAILED]

The ASN.1 coding was wrong.

[SNMP_CODE_BADLEN]

A variable binding value had a wrong length field.

[SNMP_CODE_OORANGE]

A variable binding value was out of the allowed range.

[SNMP_CODE_BADVERS]

The PDU is of an unsupported version.

[SNMP_CODE_BADENQ]

There was an ASN.1 value with an unsupported tag.

[SNMP_CODE_BADSECLEVEL]

The requested securityLevel contained in the PDU is not supported.

[SNMP_CODE_BADDIGEST]

The PDU authentication parameter received in the PDU did not match the calculated message digest.

[SNMP_CODE_EDECRYPT]

Error occurred while trying to decrypt the PDU.

snmp_pdu_encode() will return one of the following return codes:

[SNMP_CODE_OK]

Success.

[SNMP_CODE_FAILED]

Encoding failed.

SEE ALSO

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

CAVEAT

The SNMPv3 message digests, encryption and decryption, and key routines use the cryptographic functions from `crypto(3)`. The library may optionally be built without references to the `crypto(3)` library. In such case only plain text SNMPv3 PDUs without message digests may be processed correctly.

STANDARDS

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

AUTHORS

The Begemot SNMP library was originally written by Hartmut Brandt <harti@FreeBSD.org>

Shteryana Shopova <syrix@FreeBSD.org> added support for the SNMPv3 message processing and User-Based Security model message authentication and privacy.