Improved Error Reporting and Thread-Safe Use of the SNMP Library

There is a need in some environments to support multiple threads in a single application. The SNMP Library provides the Single Session functions which support thread-safe operation when certain precautions are taken. This document describes the operation of the SNMP Library with a focus on its session management functions. The Traditional API and the Single API functions are compared and contrasted.
A working understanding of the CMU or UCD SNMP Library API is recommended to fully appreciate the concepts discussed.
The document ends with a list of restrictions for using the Single API in a multi-threaded application.

Unfortunately, the SNMPv3 support was added about the same time as the thread support and since they occurred in parallel the SNMPv3 support was never checked for multi-threading correctness. It is most likely that it is not thread-safe at this time.

***** IMPORTANT ANNOUNCEMENT *****

To the point, no resource locks are applied within the SNMP Library.
The APDU encoding and some session management functions can be used in thread-safe manners. The MIB file parsing is not thread-safe.
The Single Session API was made available in November 1998.
Existing applications use the Traditional API, which is not thread-safe.
The thread-safe considerations are discussed throughout this document.

The research and development of the Single Session API that I've completed was wholly funded by my employer, Internet Security Systems, Inc. and is distributed freely to the Internet community.

-Mike Slifcak, 23 April 1999

09 July 1999 Removed references to snmp_synch_setup and snmp_synch_reset

Availability

The Single Session API is integrated into the currently available versions of the CMU SNMP library and the UC-Davis SNMP package.

ftp://ftp.net.cmu.edu/pub/snmp/cmu-snmp-V1.13.tar.gz and later
Read: snmp_sess_api.3, Changes.SingleSession

ftp://ucd-snmp.ucdavis.edu/ucd-snmp-3.6.tar.gz and later
Read: snmp_sess_api.3, README.thread (after version 3.6.1)

Both libraries work equally well in Windows NT and various UNIX platforms. Please read this document and refer to the snmp_sess_api section 3 manual page.

Glossary of Terms

APDU	Application Protocol Data Unit
API	Application Programming Interface
CMU	Carnegie-Mellon University, Pittsburgh, PA.
Library	The SNMP library; Both CMU and UCD versions are applicable.
Session	Concept embodying the management of transacting SNMP APDUS.
SNMP	Simple Network Management Protocol
UCD	University of California at Davis, CA.

Introduction

The Library extends the UNIX file concept (open, close, read, write) to a Session.
Opening a Session binds a local socket to a well-known port and creates internal structures to help with controlling the transaction of SNMP APDUs. Closing a Session releases the memory and system resources used for these purposes.

Since the mid-1980s, many SNMP applications have used the Traditional Session API to transact SNMP APDUs between the local host and SNMP-enabled devices.

The Traditional Session API does not support multi-threaded applications:

1) There are no resource locks to prevent exposing the Library's global data resources to corruption in a multi-threaded application;

2) The Traditional API functions that receive SNMP APDUs do not provide an interface for one of many sessions;

3) Errors discovered by the Library are communicated through global data structures and are not associated with the session in which the error occurred.

The Single Session API provides these capabilities:

1) Manage a single SNMP session safely, in multi-threaded or non-threaded applications, by avoiding access to data structures that the Traditional Session API may share between Sessions;

2) Associate errors with the session context for threaded and non-threaded applications.

Contrasting and Comparing Traditional API and Single API

The Traditional API uses the struct snmp_session pointer returned from snmp_open() to identify one SNMP session. The Single API uses the opaque pointer returned from snmp_sess_open() to identify one SNMP session.

Helpful Hint: The Library copies the contents of the structure which is input to snmp_open() and snmp_sess_open().
Once copied, changing that input structure's data has no effect on the opened SNMP Session.

The Traditional API uses the snmp_error() function to identify any library and system errors that occurred during the processing for one SNMP session. The Single API uses snmp_sess_error() for the same purpose.

The Traditional API manages the private Sessions list structure; adding to the list during snmp_open(), removing during snmp_close.

With few exceptions, the Traditional API calls the Single API for each session that appears on the Sessions list.

The Traditional API reads from all Sessions on the Sessions list;
The Single API does not use the Sessions list.
The Single API can read from only one Session.

> Helpful Hint:
> This is the basis for thread-safe-ness of the Library.
> There is no resource lock applied.

Using the Single API

A multi-threaded application that deploys the SNMP Library should complete all MIB file parsing before additional threads are activated. Drawing from the parsed contents of the MIB does not incur any data corruption exposure once the internal MIB structures are initialized.

The application may create threads such that a single thread may manage a single SNMP session. The thread should call snmp_sess_init() to prepare a struct snmp_session structure. The thread can adjust session parameters such as the remote UDP port or the local UDP port, which must be set prior to invoking snmp_sess_open().

The first call to snmp_sess_init() initializes the SNMP Library, including the MIB parse trees, before any SNMP sessions are created. Applications that call snmp_sess_init() do not need to read MIBs nor setup environment variables to utilize the Library.

After the struct snmp_session is setup, the thread must call snmp_sess_open() to create an SNMP session. If at any time the thread must change the Session configuration, snmp_sess_session() returns the pointer to the internal configuration structure (a struct snmp_session, copied from snmp_sess_open).
The thread can adjust parameters such as the session timeout or the community string with this returned struct snmp_session pointer. Changes to the remote or local port values have no effect on an opened Session.

The thread can build PDUs and bind variables to PDUs, as it performs its duties.
The thread then calls snmp_sess_send() or snmp_sess_async_send() to build and send an SNMP APDU to the remote device. If a Get-Response-PDU is expected, the thread should call snmp_sess_synch_response() instead.

When the thread is finished using the session, it must free the resources that the Library used to manage the session.

Finally, the thread must call snmp_sess_close() to end the Session.

Snmp_sess_init(), snmp_open(), and snmp_sess_open() must use the same calling parameter for a given Session.
Other methods should use only the returned parameter from snmp_open() and snmp_sess_open() to access the opened SNMP Session.

Error Processing

Two calls were added: snmp_error() and snmp_sess_error() return the "errno" and "snmp_errno" values from the per session data, and a string that describes the errors that they represent. **The string must be freed by the caller.**

Use snmp_error() to process failures after Traditional API calls, or snmp_sess_error() to process failure after Single API calls.
In the case where an SNMP session could not be opened, call snmp_error() using the struct snmp_session supplied to either snmp_open() or snmp_sess_open().

The following variables and functions are obsolete and may create problems in a multi-threaded application:

```
int    snmp_errno
char * snmp_detail
snmp_set_detail()
snmp_api_errstring()
```

Function Summary

The functions in the following table are functionally equivalent, with the exception of these behaviors:
- The Traditional API manages many sessions
- The Traditional API passes a struct snmp_session pointer, and touches the Sessions list
- The Single API manages only one session
- The Single API passes an opaque pointer, and does not use Sessions list

Traditional	Single	Comment
snmp_sess_init	snmp_sess_init	Call before either open
snmp_open	snmp_sess_open	Single not on Sessions list
	snmp_sess_session	Exposes snmp_session pointer
snmp_send	snmp_sess_send	Send one APDU
snmp_async_send	snmp_sess_async_send	Send one APDU with callback
snmp_select_info	snmp_sess_select_info	Which session(s) have input
snmp_read	snmp_sess_read	Read APDUs
snmp_timeout	snmp_sess_timeout	Check for timeout

snmp_close	snmp_sess_close	Single not on Sessions list
snmp_synch_response	snmp_sess_synch_response	Send/receive one APDU
snmp_error	snmp_sess_error	Get library,system errno

Example 1 : Traditional API use.

```
    #include "snmp_api.h"
      ...
      int liberr, syserr;
      char *errstr;
      struct snmp_session Session, *sptr;
      ...
      snmp_sess_init(&Session);
      Session.peername = "foo.bar.net";
      sptr = snmp_open(&Session);
      if (sptr == NULL) {
          /* Error codes found in open calling argument */
          snmp_error(&Session, &liberr, &syserr, &errstr);
          printf("SNMP create error %s.\n", errstr);
          free(errstr);
          return 0;
      }
      /* Pass sptr to snmp_error from here forward */
      ...
      /* Change the community name */
      free(sptr->community);
      sptr->community = strdup("public");
      sptr->community_len = strlen("public");
      ...
      if (0 == snmp_send(sptr, pdu)) {
          snmp_error(sptr, &liberr, &syserr, &errstr);
          printf("SNMP write error %s.\n", errstr);
          free(errstr);
          return 0;
      }
      snmp_close(sptr);
```

Example 2 : Single API use.

```
    #include "snmp_api.h"
      ...
      int liberr, syserr;
      char *errstr;
      void *sessp;  /* <-- an opaque pointer, not a struct pointer */
      struct snmp_session Session, *sptr;
      ...
      snmp_sess_init(&Session);
      Session.peername = "foo.bar.net";
      sessp = snmp_sess_open(&Session);
      if (sessp == NULL) {
          /* Error codes found in open calling argument */
          snmp_error(&Session, &liberr, &syserr, &errstr);
          printf("SNMP create error %s.\n", errstr);
          free(errstr);
```

```
          return 0;
      }
      sptr = snmp_sess_session(sessp); /* <-- get the snmp_session
pointer */

      /* Pass sptr to snmp_sess_error from here forward */
      ...
      /* Change the community name */
      free(sptr->community);
      sptr->community = strdup("public");
      sptr->community_len = strlen("public");
      ...
      if (0 == snmp_sess_send(sessp, pdu)) {
          snmp_sess_error(sessp, &liberr, &syserr, &errstr);
          printf("SNMP write error %s.\n", errstr);
          free(errstr);
          return 0;
      }
      snmp_sess_close(sessp);
```

Example 3. Differences Between Traditional API and Single API Usage

```
5a6
>       void *sessp;  /* <-- an opaque pointer, not a struct pointer */
11,13c12,14
<       sptr = snmp_open(&Session);
<       if (sptr == NULL) {
---
>       sessp = snmp_sess_open(&Session);
>       if (sessp == NULL) {
19c20,22
<       /* Pass sptr to snmp_error from here forward */
---
>       sptr = snmp_sess_session(sessp); /* <-- get the snmp_session
pointer */
>
>       /* Pass sptr to snmp_sess_error from here forward */
26,27c29,30
<       if (0 == snmp_send(sptr, pdu)) {
<           snmp_error(sptr, &liberr, &syserr, &errstr);
---
>       if (0 == snmp_sess_send(sessp, pdu)) {
>           snmp_sess_error(sessp, &liberr, &syserr, &errstr);
33c36
<       snmp_close(sptr);
---
>       snmp_sess_close(sessp);
```

Restrictions on Multi-threaded Use of the SNMP Library

1. Invoke SOCK_STARTUP or SOCK_CLEANUP from the main thread only.

2. The MIB parsing functions use global shared data and are not multi-thread safe when the MIB tree is under construction. Once the tree is built, the data can be safely referenced from any thread. There is no provision for freeing the MIB tree. Suggestion: Read the MIB files before an SNMP session is created.

This can be accomplished by invoking snmp_sess_init from the main thread and discarding the buffer which is initialized.

3. Invoke the SNMPv2p initialization before an SNMP session is created,for reasons similar to reading the MIB file.
   The SNMPv2p structures should be available to all SNMP sessions.
   CAUTION: These structures have not been tested in a multi-threaded application.

4. Sessions created using the Single API do not interact with other SNMP sessions. If you choose to use Traditional API calls, call them from a single thread. The Library cannot reference an SNMP session using both Traditional and Single API calls.

5. Using the callback mechanism for asynchronous response PDUs requires additional caution in a multi-threaded application. This means a callback function probably should probably not use Single API calls to further process the session.

6. Each call to snmp_sess_open() creates an IDS. Only a call to snmp_sess_close() releases the resources used by the IDS.