Laser-Scan Ltd.

Table Monitor

Technical Reference

Issue 1.8 - 16-December-1989

Science Park, Milton Road, Cambridge, England CB4 4BH tel: (0223) 420414

Document "Table Monitor Technical"		Category "TECHNICAL"
Document Issue 1.3	Tony J Ibbs	3-September-1984
.		
.		
.		
Document Issue 1.7	Ron Russell	12-February-1988
Document Issue 1.8	Clarke Brunt	16-December-1989

1 *Introduction*

This is the user documentation for the LSL table monitor system.

The table monitor system is formed of 3 parts

* STARTMON

  This is a privileged image which is intended to be run during system startup. It creates a detached image which performs the actual work of the system. This image is either called TABLE_MONITOR or MONITOR_<terminal> depending on whether the "named monitor" option is selected.

* TABLE MONITOR

  The table monitor itself watches the digitising table in real time, processes the table strings it receives, and updates the relevant fields in the global section that it uses to communicate with the library. It also 'debounces' buttons and handles streamed mode.

  Whilst the library is not using the table, the table monitor process hibernates, thus not loading the system any more than necessary.

* TABLIB

  This is the table monitor library. It contains routines for waking up and putting to sleep the TABLE MONITOR, and routines for interrogating the current state of the digitising table.

2 *STARTMON - starting up the TABLE MONITOR*

2.1 *Use*

The STARTMON image is LSL$EXE:STARTMON.EXE and must be installed with the following privileges

* DETACH - to allow it to create the table monitor as a detached process. The table monitor will thus not stop when the process running STARTMON exits

* ALTPRI - to allow it to create the table monitor process with a high priority, so that it can easily deal with digitising table input as it arrives in streamed mode

STARTMON expects the following arguments

1. the first is the device name of the digitising table the user is interested in. This device name is upper-cased and fully logically translated by STARTMON before use.

   The table monitor is created with its input from the digitising table and its outputs are directed to an error file in the directory LSL$MGMT.

2. the second argument is the priority at which the table monitor should run. For a normal system, this should be given as 16 - the lowest real-time priority. This means that the table monitor will not be competing with user processes for processor time.

3. the third argument is the (octal) UIC group within which the table monitor should run. This is important because the process(es) using TABLIB must be in the same group as the table monitor itself. This will normally be 100

4. the fourth argument is optional; entering "YES" or just "Y" allows the use of a named table monitor. The use of this option allows multiple table monitors to be used.

For example

```
$!
$! set up the START image
$!
$ STARTMON := "$LSL$EXE:STARTMON"
$!
$! now start the table monitor on the line TXA3
$! - at priority 16 and in group 100
$!
$ STARTMON  TXA3  16  100 Y
$!
```

The STARTMON program produces the following messages

```
Creating TABLE_MONITOR or Creating MONITOR_<terminal>
In UIC [<group>,001], at priority <priority>
Input is <digitising table>, output is <error file>
Waiting
   .
   .
Created table monitor, process ID <hex ID number>
```

The "Waiting" messages occur as the program waits for the table monitor to initialise and reach its stable hibernating state.

2.2 *Error messages*

The following errors may be produced by STARTMON

No table input name found

> There were no arguments on the STARTMON command line. STARTMON exits - the table monitor is not created.

No priority found

> No priority number was given. STARTMON exits - the table monitor is not created.

Priority out of range (0-31)

> The priority number was given, but is unacceptable. STARTMON exits - the table monitor is not created.

No UIC group number found

> The third argument is missing, or is not octal. STARTMON exits - the table monitor is not created.

Error upper-casing "<input device name>"
<system interpretation of error>

> STARTMON upper-cases the input device name before logical name translating it. This message is given if that fails - this should never happen. The table monitor is not created.

Error translating <input device name>
<system interpretation of error>

> Some error occurred in translating the input device name. The table monitor is not created.

Error creating the table monitor process
<system interpretation of error>

> As the message says. The system version of the error should give more information.

Error running the table monitor process
Check:-

1. there is an image LSL$EXE:TABMON

2. the terminal name was specified correctly

3. the logical name LSL$MGMT is set up in the group or system logical name table

4. you have adequate quotas/privilege

As the message says. The table monitor has been aborted.

3 *TABMON - the table monitor*

The table monitor image is LSL$EXE:TABMON.EXE, and the actual process is created with name TABLE_MONITOR or MONITOR_<terminal> by the STARTMON program.

3.1 *How the table monitor works*

3.1.1 *Communication with TABLIB -*

The table monitor communicates with TABLIB by three means

* Shared global section - LSL$MGMT:LSLTABLE.SEC or LSLMGMT:TABLE_<terminal>.SEC

  The actual information about cursor position, and so on, is passed via a shared global section. This is a piece of the program's storage which is mapped onto an ordinary disk file.

  Information may be written to, or read from, the section by any process which has mapped to it.

* Lock management - resource LSL_TABLE_LOCK or T_LOCK_<terminal>

  The reading/writing of the global section is managed by use of the lock manager. A 'resource' called LSL_TABLE_LOCK or T_LOCK_<terminal>is used - the processes mapped to the global section use a lock on this resource to signify that they are reading or writing the section. The other process cannot then lock the resource for writing or reading (respectively), so is forced not to access the section.

* Lock management - resource TABLIB_TABLE_NAME or TAB_LOCK_<terminal>

  TABLIB gets an exclusive lock on this resource to ensure that another process cannot also gain control of the table monitor. This lock is also used to signal from TABMON to TABLIB that an event has happened.

* Common event flags - cluster LSL_TABLE_EFC or T_EFC_<terminal>

  Various event flags are held in common by the table monitor and TABLIB. These are the first few event flags in the cluster LSL_TABLE_EFC or T_EFC_<terminal>. They are used to tell TABLIB when a new button has been pressed, when the streamed coordinate has changed, to tell the table monitor that it should HIBERNATE again, and that the library is ready for the next button press.

* Decode routine

  If the logical name LSL$TABMON_ROUTINE (or LSL$TABMON_ROUTINE_<terminal> if the named version of the table monitor is being used) has been set up at group or system level before the table monitor was started with STARTMON, and a shared image with an UNIVERSAL entry point "DECODE" exists where it points to, then this shared routine will be used to decode the strings that are received from the table. Otherwise the table monitor will assume that an ALTEK table is being used which produces strings in the format

$n,xxxxx,yyyyy.

The specification for the decoding routine is as follows:

```
       LOGICAL FUNCTION DECODE(BUFF,BUFLEN,BUT,X,Y)
C
       IMPLICIT NONE
C
C TABMON function to decode string from table
C Return TRUE if fail, FALSE if succeed
C
C arguments
       BYTE            BUFF(*)          ! (input) string from table
       INTEGER*2       BUFLEN           ! (input) number of chars in
string
       INTEGER         BUT              ! (output) button number
       INTEGER         X                ! (output) x coordinate
       INTEGER         Y                ! (output) y coordinate
```

If the table monitor is to be asked by TABLIB to probe the table (see below), then it requires a prompt string to send to the table. By default this is the string 'V' - the prompt required by ALTEK tables.

If the external decoding routine is being used, then it is possible to define another entry point in the shared image that allows this prompt string to be specified. This entry point is "GET_TABLE_PROMPT", and the name GET_TABLE_PROMPT must be defined as UNIVERSAL in the shared image.

The specification for the routine is as follows:

```
C
       SUBROUTINE GETTABLEPROMPT(PBUF,PBUFMAX,PBUFL)
C
       IMPLICIT NONE
C
       INTEGER         PBUFMAX          ! maximum length of buffer
       BYTE            PBUF(PBUFMAX)
       INTEGER*4       PBUFL
C
```

3.1.2 *What the table monitor does* -

After waking up, the table monitor assigns the digitising table as input. It then places a read request on the table, and waits for one of three events to happen -

* The user may press a button on the puck. This itself causes one of the following:

  1. If this is a 'new' button press then the transition from no button pressed to streaming is noted. One of the following actions is taken, to deal with the <button> (the data generated by the button press)

a. If there are no <button>s in the table monitor's type-ahead buffer, and if TABLIB has signalled that it is ready for a new button, then the <button> is written to the global section, and the 'received a new button' event flag is set.

b. If there are no <button>s in the table monitor's type-ahead buffer, and TABLIB is not yet ready for a new button, then the <button> is added to the buffer.

c. If there are <button>s in the buffer, and TABLIB is ready for a new button, then the first <button> is popped from the buffer and written to the global section, the 'received a new button' event flag is set, and the latest <button> is added to the buffer.

d. If there are <button>s in the type-ahead buffer, but TABLIB is not ready, then the latest <button> is added to the buffer.

2. Whilst a button is depressed, the digitising table produces a stream of output, effectively multiple button presses. The table monitor does not stack this up, but whenever the coordinates of this streamed data changes, it writes the new X,Y position to the global section, and sets the 'streamed data changed' event flag. Note that this is regardless of whether TABLIB has read the last X,Y position.

3. If a gap of more than <time-out> (a value which defaults to 1/10 second) intervenes between two (notional) button presses, or if the button number changes, then a new button is considered to have been depressed, and the 'new button' sequence at (1) above is entered again.

NOTE

> The mechanism described above means that TABLIB will receive all <button>s, but will only receive the latest streamed position, since no type-ahead is performed for streamed output.

* TABLIB may set the 'require a new button' flag. The actions are similar to those for pressing a button:

  a. If there are <button>s in the table monitor's typeahead buffer, then the first <button> is popped from the buffer and written to the global section.

  b. If the table monitor's type-ahead buffer is empty, then no action is taken.

* TABLIB may set the 'probe the table' event flag. The actions in this case are:

a. Clears the 'probe the table' event flag and the 'had an error' event flag

b. Throws away any outstanding button presses, and purges the buffer

c. reads the table with the required prompt, and a time out of 10 seconds.

d. If the read succeeds, then

   1. the string returned from the table is interpreted and the result written into the global section

   2. the 'wait for table' table event flag is set

   If the read fails then

   1. the 'had an error' event flag is set

   2. an error message is written to the global section

* Finally, TABLIB may set the 'go back to sleep' event flag, and this causes the table monitor to relinquish the digitising table (so that other programs may use it) and to HIBERNATE again.

3.1.3 *Errors and error messages* -

The table monitor may produce three sorts of errors:

1. an ABANDON aborts the table monitor process. The message text is written to the error output file, LSL$MGMT:TABMON.ERR, or LSL$MGMT:TABLE_<terminal>.ERR

2. system tracebacks (which notionally should never occur) are also written to the error output file

3. run time errors are written to the global section for TABLIB to detect (using HAD_ERROR and READ_ERROR)

3.1.3.1 *ABANDON errors* -

These errors are produced during the initial startup of the table monitor, and are fatal. The error messages are written to the error output file.

Abandoning TABLE_MONITOR - no global section
<system interpretation of error>

The table monitor was unable to map to the global section file LSLTABLE.SEC, or possibly unable to create it if it didn't already exist.

Abandoning TABLE_MONITOR - no lock
<system interpretation of error>

The table monitor was unable to establish a null lock on either LSL_TABLE_LOCK (T_LOCK_<terminal>) or TABLIB_TABLE_NAME (TAB_LOCK_<terminal>)

Abandoning TABLE_MONITOR - no common event flag cluster
<system interpretation of error>

The table monitor was unable to associate the common event flag cluster LSL_TABLE_EFC (T_EFC_<terminal>)

3.1.3.2 *Run-time errors* -

These errors are non-fatal, and are written into the global section as a byte string, a length and an error number. This error number will be either the system error code (which will give a more precise indication of what went wrong), or possibly -1 if no error code is applicable.

The TABLIB routine HAD_ERROR is used to check for an error, and READ_ERROR to read it back.

Bad table string "<bad string>"

Occasionally the digitising table returns a garbled coordinate string. If this happens frequently then it should be reported to LSL. This error message allows the user to detect such an event, and returns the <bad string> that was received.

The error number is returned as -1 in this case.

Button push event flag not cleared?

When a new button press is detected, and the 'Ready for a new button' event flag is set, the table monitor checks that the 'New button press' event flag is unset before writing to the global section. This message occurs if that event flag was indeed not set. This error should never occur. The table monitor attempts to continue.

Corrupt type-ahead - invalid length record

The table monitor's type-ahead buffer has become corrupt. The buffer is reinitialised, and thus any type-ahead is lost.

The error number is returned as -1 in this case.

Error assigning table input

On waking up (after a TABLIB call of INIT_TAB or RESUME_TAB) the table monitor attempts to assign the digitising table. This message indicates some failure in that. The table monitor attempts to continue, but a TABLIB call of CLOSE_TAB is the only sensible action.

Error clearing button request event flag

After writing a new button press to the global section, the table monitor clears the event flag which TABLIB sets to indicate readiness for another button press. This error indicates something went wrong with that process.

Error deassigning table input

After a TABLIB call of STOP_TAB or CLOSE_TAB, the table monitor deassigns the digitising table, before HIBERNATEing. This error indicates some difficulty in that (although there is no way of detecting it after CLOSE_TAB!). The table monitor will attempt to HIBERNATE, anyway.

Error during lock

The table monitor and TABLIB synchronises access to the global section via the lock manager. This error is given if something goes wrong in a call to the lock management system routines. The table monitor will attempt to continue. (It will not write to the global section if it cannot gain an exclusive lock before doing so, though).

Error hibernating TABLE_MONITOR

This error is given if the table monitor is unable to HIBERNATE after a TABLIB call of STOP_TAB or CLOSE_TAB. The table monitor will continue, as if INIT_TAB had been called again. Note that if this error was provoked by a call of STOP_TAB, then this message MAY be perceived by the process using TABLIB, since STOP_TAB doesn't unmap from the global section or event flag cluster. Regardless, it should never happen.

Error in reading from table

This indicates that an error occurred in the QIO used to read the digitising table input. This error should be reported to LSL, but unless it recurs is unlikely to be serious. One possible cause is that the VAX is heavily loaded, and the table monitor is getting insufficient time to be able to 'keep up' with the streamed output from the table.

The table monitor will attempt to continue.

Error purging TABLE_MONITOR's working set

Before hibernating, the table monitor attempts to shrink its working set, and this error reflects some problem with that. The table monitor will attempt to continue (ie to HIBERNATE).

Error setting button push event flag

After a new button has been written to the global section, the table monitor sets an event flag so that TABLIB can detect this. This error shows that something went wrong with that process. The table monitor will attempt to continue.

Error submitting lock request

This error is similar to the error 'Error during lock' described above, except that in this case the request for a change in the lock could not be submitted. The table process will attempt to continue. (It will not write to the global section if it cannot gain an exclusive lock before doing so, though).

Error waiting for table input

This indicates that an error occurred whilst the table monitor was waiting for something to happen (either an input from the digitising table, or an indication from TABLIB that it should HIBERNATE again, or a request for another button press from TABLIB's READ_PUCK). The table monitor will attempt to continue.

Type-ahead buffer is full

This should not occur, as the type-ahead currently has room for around 20 button presses.

Want another button flag not set?

This error occurs in the same situation as 'Error clearing button request event flag', if the event flag was already clear. This should never happen.

Error submitting position inquiry to table

This error occurs when submitting a QIO to probe the table. This error should not occur, but is not fatal if it does. The table monitor sets the 'had an error' event flag and continues.

Error while inquiring position from table

This error occurs when receiving the result of the QIO used to to probe the table. This error should not occur, but is not fatal if it does. The table monitor sets the 'had an error' event flag and continues

Error submitting table lock request

The request for a change in the lock TABLIB_TABLE_NAME (TAB_LOCK_<terminal>) to inform TABLIB of an event could not be submitted. The table process will attempt to continue.

Error dequeuing table lock

Similar to 'Error submitting lock request' above, except the error occurred while trying to cancel the request.

Error during table lock

Similar to 'Error submitting lock request' above. The lock request was submitted successfully, but subsequently gave an error.

Unknown error

The error generated internally by the table monitor is unknown. This should never occur - please report it to LSL. Note that there may or may not be a system error number associated with this error.

4 *TABLIB - the table monitor library*

This section describes the routines available to the user via TABLIB, the library used by user processes to interact with the table monitor itself.

TABLIB is to be found in LSL$LIBRARY

4.1 *Restrictions*

The following restrictions are associated with use of the table monitor and TABLIB

* The user process must be in the same group as the table monitor - the various communication methods (specifically the common event flags) will not function across group boundaries.

* Only one user process may be using TABLIB for any particular table monitor at any one time - a call of INIT_TAB 'locks' the library, and CLOSE_TAB frees it again.

4.2 *The routines*

Note that in the routines, the following conventions are observed:

* declarations - the following notations are observed in describing variables

  long   xxx - xxx is a longword, a Fortran integer*4

  word   xxx - xxx is a word, a Fortran integer*2

  logical xxx - xxx is a logical

  byte   xxx - xxx is a byte

* also, values passed to the routine are declared as IN, and values returned are declared as OUT

* arguments surrounded by square brackets ([and]) are optional.

* the following standard variables are referred to

  logical error - OUT - this is a logical return from many of the routines. It is set TRUE if an error occurred in the routine, FALSE otherwise.

  long   ierr - OUT - this is a further error return. In the event of an error in a routine, this contains an appropriate system error code.

4.2.1 Starting things up, and closing them down -

In these routines, 'ierr' may return with the special error values -2 or -4, as well as the more normal system error codes (which are positive). An error of -2 means that the TABLE_MONITOR was in an unexpected state, and -4 means that someone else is already using TABLIB, and has thus locked the system.

```
error = INIT_TAB( [timeout], ierr, [ast] )
```

long timeout - IN - the minimum time between two button 'presses' of the same button for them to be regarded as different presses. This is in units of hundredths of a second, and defaults to one tenth of a second.

external ast - IN - a subroutine (declared external) which will be called whenever a table event (button, stream, or error) occurs. Once called, the routine will not be called again until one of READ_PUCK, READ_STREAM, or READ_ERROR is called. This provides an alternative to waiting for the event flags when awaiting the next event.

INIT_TAB is called to initialise the table monitor system, and must be called before any other TABLIB routines are used. It performs the following actions

1. checks whether the logical name LSL$MONITOR_TABLE has been defined. If it has, it uses its translation to use the named version of the table monitor.

2. checks whether the TABLE MONITOR is hibernating. Note that the routine will attempt to carry on, even if the table monitor process was not hibernating

3. locks the table monitor system - this prevents any other process using TABLIB at the same time

4. maps to the shared global section - it can now read and write it

5. checks that the table monitor and library share a common interface version number - that is that the format of the global section, the event flags in use, etc, are from the same version of the interface definition - if not, fails with 'ierr' = -4

6. establishes a null lock on the global section

7. associates to the common event flag cluster which is used to signal button presses, etc

8. establishes a write lock on the global section

9. writes the time out value into the global section

10. unlocks the section again (back to a null lock)

11. sets the 'ready for a button' event flag

12. if the table monitor was hibernating, then issues a wake request - this causes it to come out of hibernation. If the table monitor was not hibernating then it tries to set it hibernating, and then wakes it up as above. It attempts this 5 times, then it sets 'ierr' = -2 (but note that this does not cause it to fail)

NOTE that although the routine will succeed if the TABLE MONITOR was not hibernating, this is still an error condition. Specifically, the various state flags and the current positions are only cleared when the TABLE MONITOR is woken up. This situation should only occur if a previous program using the library exited without calling CLOSE_TAB

error = STOP_TAB(ierr)

STOP_TAB is used to put the TABLE MONITOR back to sleep, when the current process expects to use the library again later. Note that it does not unmap the global section, disassociate from the common event flag cluster, or release the locks on the section and the use of the table monitor system.

It DOES perform the following actions

1. checks that TABLE MONITOR is not hibernating - if it is, fails with 'ierr' = -2

2. sets the event flag which requests TABLE MONITOR to hibernate once again.

error = RESUME_TAB([timeout], ierr)

long timeout - IN - the minimum time between two button 'presses' of the same button for them to be regarded as different presses. This is in units of hundredths of a second, and defaults to one tenth of a second.

RESUME_TAB is used to wake the TABLE MONITOR again, after a call of STOP_TAB

It performs the following actions

1. checks that the TABLE MONITOR is hibernating - if not, fails with 'ierr' = -2

2. wakes the TABLE MONITOR up again

error = CLOSE_TAB(ierr)

CLOSE_TAB is the final shutdown routine for TABLIB. To use the library again after this, a new call to INIT_TAB is required.

It performs the following actions

1. call STOP_TAB (results may thus be as from STOP_TAB above)

2. unmaps from the global section

3. disassociates from the common event flag cluster

4. releases the (null) lock on the global section

5. releases the lock on the table monitor system - other processes are now free to use TABLIB

4.2.2 Wait routines -

In all of these routines, if the relevant event flag(s) are already set, then the routine returns immediately.

error = WAIT_TAB(ierr)

WAIT_TAB waits for the next event - it waits for the table monitor to set any one of the event flags to which the library is sensitive -

* a new puck button has been depressed

* the streamed coordinate has changed

* an error has occurred

error = WAIT_PUCK(ierr)

WAIT_PUCK waits for a new puck button to be depressed, or for an error to occur.

error = WAIT_STREAM(ierr)

WAIT_STREAM waits for the streamed coordinate to change, or for an error to occur.

4.2.3 Event flag routines -

These routines are used to enquire what it was that happened at the table

The following return is made from all of the routines

logical was_set - OUT - TRUE if the relevant event flag was set, FALSE if it was not.

Note that in general the routines may be assumed to have succeeded, but that the 'ierr' argument will always be set to SS$_NORMAL for success, and a relevant system error code otherwise.

was_set = HAD_PUCK(ierr)

HAD_PUCK tests whether the PUCK event flag was set - that is whether a new button press is waiting to be read.

was_set = HAD_STREAM(ierr)

HAD_STREAM tests whether the STREAM event flag was set - that is whether the streamed coordinate has changed.

was_set = HAD_ERROR(ierr)

HAD_ERROR tests whether the ERROR event flag was set - that is whether a new error message has been signalled by the table monitor.

4.2.4 Asking for a response -

This routine is used ask the table monitor to probe the table. It waits until either the data is available in the global section, or the error flag has been set.

Before probing the table the table monitor clears the error event flag, so after a call to ASKTAB if HAD_ERROR is .FALSE., then a call of READ_TABLE will give the probed coordinates

SUBROUTINE ASKTAB

4.2.5 Reading the responses -

These routines are used to read the relevant data from the global section. Note that each routine will unset the relevant event flag (although it will not complain if the flag is not set), and that repeated calls are allowed.

error = READ_TABLE(x, y, ierr)

```
long x      - OUT - the X coordinate at which the button was pressed
long y      - OUT - the equivalent Y coordinate
```

READ_TABLE reads the coordinates of the point that was returned when a call to ASKTAB has been made.

error = READ_PUCK(button, x, y, ierr)

```
byte button - OUT - the hexadecimal button number
long x      - OUT - the X coordinate at which the button was pressed
long y      - OUT - the equivalent Y coordinate
```

READ_PUCK reads the next button press - it returns the number of the button (in the range 0 to F) and the X and Y coordinates of the table position at which the button was pressed.

Since the table monitor provides its own type-ahead, this is not necessarily the latest button pressed.

error = READ_STREAM(x, y, ierr)

long x - OUT - the X coordinate that the streamed button has reached
long y - OUT - the Y coordinate for the same

READ_STREAM returns the current position of the streamed button (or the final position, if no button is currently depressed).

error = READ_ERROR(errbuf, errlen, errnum, ierr)

byte errbuf - OUT - the buffer into which the error message will be written. This should be at least 80 bytes long.
word errlen - OUT - the length of the message placed into 'errbuf'
long errnum - OUT - the system code of the error, or -1 if the error is a bad table string.

READ_ERROR is used to read the error message and system error code of an error in the table monitor. For a description of the possible error messages, see the section on the table monitor itself.

4.2.6 Corner point routines -

error = WRITE_CPTS(cpts_array, ierr)

long cpts_array(8) - IN - an array of data to be written

WRITE_CPTS is used to store the corner points of the map in the global section, so that they may be used by later programs. The data is assumed to be an array of eight longwords, presumably integer table coordinates for X,Y at each corner. This sets a logical in the global section to indicate that the field has been set up.

error = READ_CPTS(cpts_array, ierr)

long cpts_array(8) - OUT - an array of data to be read

READ_CPTS is used to read the corner points of the map from the global section, as stored by a call of WRITE_CPTS. No check is made that they exist before reading them, so garbage can result.

was_set = HAD_CPTS(ierr)

logical was_set - OUT - true is returned if WRITE_CPTS has been called for this global section.

HAD_CPTS is used to check whether corner points have actually been written to the global section (although it doesn't check WHEN).

4.2.7 Other routines -

RETURN_EFN(puck, stream, error, free, last)

long puck - OUT - event flag number used to signal a new button press
long stream - OUT - event flag number used to signal that the streamed coordinate has changed

long error - OUT - event flag number used to signal a new error message
long free - OUT - the first free event flag number in the common event flag cluster containing the above flags
long last - OUT - the last event flag number contained in the common event flag cluster

RETURN_EFN is used to determine which event flags in the TABLIB cluster are set when an event occurs, and which event flags the process using TABLIB may safely use within that cluster. The latter allows the process to wait for the logical OR of the table monitor events with events which are not related to the table (for instance something typed at a terminal).

The table monitor itself will never use event flags above 'free'-1