NAME

mt_start_element, **mt_end_element**, **mt_char_handler**, **mt_status_tree_sbuf**, **mt_status_tree_print**, **mt_status_entry_free**, **mt_status_free**, **mt_entry_sbuf**, **mt_param_parent_print**, **mt_param_entry_print**, **mt_protect_print**, **mt_param_list**, **mt_density_name**, **mt_density_bp**, **mt_density_num**, **mt_get_xml_str**, **mt_get_status** - Magnetic Tape library

LIBRARY

library "libmt"

SYNOPSIS

#include <sys/sbuf.h>
#include <bsdxml.h>
#include <mtlib.h>

void
mt_start_element(*void *user_data, const char *name, const char **attr*);

void
mt_end_element(*void *user_data, const char *name*);

void
mt_char_handler(*void *user_data, const XML_Char *str, int len*);

void
mt_status_tree_sbuf(*struct sbuf *sb, struct mt_status_entry *entry, int indent,*
*void (*sbuf_func)(struct sbuf *sb, struct mt_status_entry *entry, void *arg), void *arg*);

void
mt_status_tree_print(*struct mt_status_entry *entry, int indent,*
*void (*print_func)(struct mt_status_entry *entry, void *arg), void *arg*);

*struct mt_status_entry **
mt_entry_find(*struct mt_status_entry *entry, char *name*);

*struct mt_status_entry **
mt_status_entry_find(*struct mt_status_data *status_data, char *name*);

void
mt_status_entry_free(*struct mt_status_entry *entry)*);

void
mt_status_free(*struct mt_status_data *status_data*);

void
mt_entry_sbuf(*struct sbuf *sb, struct mt_status_entry *entry, char *fmt*);

void
mt_param_parent_sbuf(*struct sbuf *sb, struct mt_status_entry *entry,*
*struct mt_print_params *print_params*);

void
mt_param_parent_print(*struct mt_status_entry *entry, struct mt_print_params *print_params*);

void
mt_param_entry_sbuf(*struct sbuf *sb, struct mt_status_entry *entry, void *arg*);

void
mt_param_entry_print(*struct mt_status_entry *entry, void *arg*);

int
mt_protect_print(*struct mt_status_data *status_data, int verbose*);

int
mt_param_list(*struct mt_status_data *status_data, char *param_name, int quiet*);

*const char **
mt_density_name(*int density_num*);

int
mt_density_bp(*int density_num, int bpi*);

int
mt_density_num(*const char *density_name*);

int
mt_get_status(*char *xml_str, struct mt_status_data *status_data*);

DESCRIPTION

The MT library consists of a number of functions designed to aid in interacting with the sa(4) driver. The sa(4) driver returns some status data as XML-formatted strings, and the primary purpose of this

library is to make it easier for the software developer to parse those strings and extract the status values.

The **mt_start_element**(), **mt_end_element**(), and **mt_char_handler**() functions are designed to work with the libbsdxml(3) library, which is an XML parsing library. The user data for the XML parser should be set with **XML_SetUserData**() to a zeroed struct mt_status_data with the entries list initialized. The element handlers for the XML parser should be set to **mt_start_element**() and **mt_end_element**() with **XML_SetElementHandler**(). The character data handler should be set to **mt_char_handler**() with the **XML_SetCharacterDataHandler**() function. The error member of the status_data structure will be set to 0 if parsing is successful, and non-zero if parsing failed. In the event of a failure, the error_str member will contain an error message describing the failure. These functions will build a tree of tape driver status data that can be searched and printed using the other functions in this library.

mt_status_tree_sbuf() takes the root node of a tree of sa(4) driver status information, and displays it in an sbuf(9). The *sb* argument is the destination sbuf. The *entry* argument is the root of the tree. The *indent* argument is the number of characters to indent the output. Each recursive call to **mt_status_tree_sbuf**() will have the indent level incremented by 2. The *sbuf_func* argument is for a user-supplied alternate printing function. If it is non-NULL, it will be called instead of the default output printing code. The *arg* argument is an argument for the *sbuf_func* function.

The **mt_status_tree_print**() function is the same as the **mt_status_tree_sbuf**() function, except that the tree is printed to standard out instead of to a sbuf.

The **mt_entry_find**() function returns the first entry in the tree starting at *entry* that matches *name*. The supplied node name can be a single level name like "foo", or it can specify multiple node names that must be matched, for instance "foo.bar.baz". In the case of a single level name, it will match any node beneath *entry* that matches *name*. In the case of a multi-level name like "foo.bar.baz", it will return the first entry named "baz" whose immediate parent is "bar" and where the parent of "bar" is named "foo".

The **mt_status_entry_find**() is the same as **mt_entry_find**(), except that it operates on the top level mt_status_data and all mt_status_entry nodes below it instead of just an mt_status_entry structure.

The **mt_status_entry_free**() function frees the tree of status data underneath *entry*.

The **mt_status_free**() function frees the tree of status data underneath *status_data*.

The **mt_entry_sbuf**() function prints *entry* to the supplied sbuf *sb*, optionally using the printf(3) format *fmt*. If *fmt* is NULL, then **mt_entry_sbuf**() will render integer types in base 10 without special formatting and all other types as they were rendered in the XML.

mt_param_parent_sbuf() prints the parents of the given *entry* to the supplied sbuf *sb* subject to the print

parameters *print_params*. The result will be formatted with a period between each level, like "foo.bar.baz".

mt_param_parent_print() is like **mt_param_parent_sbuf**() except that it prints the results to standard output instead of an sbuf.

mt_param_entry_sbuf() prints the *entry* to the given sbuf *sb*. The argument *arg* is a pointer to struct mt_print_params, which allows the caller to control the printing output. This function is intended to be supplied as an argument to **mt_status_tree_sbuf**().

mt_param_entry_print() is like **mt_param_entry_sbuf**() except that it prints to standard output instead of an sbuf. It is intended to be used as an argument to **mt_status_tree_print**().

mt_protect_print() prints tape drive protection information from the supplied *status_data* beginning at the node name defined as the root node for protection data. If the *verbose* argument is non-zero, protection entry descriptions will be printed. If it is zero, protection entry descriptions will not be printed.

mt_param_list() prints tape driver parameters information from the supplied *status_data*. If the *param_name* is non-NULL, only the named parameter will be printed. If *quiet* is non-zero, parameter descriptions will be omitted in the output.

mt_density_name() Returns a text identifier for the supplied numeric *density_num*. The *density_num* should currently be a value between 0 and 255 inclusive, since that is the valid range for SCSI density code values. See below for notes on the return values.

mt_density_bp() Returns the bits per inch or bits per mm values for a given density entry specified by the *density_num*. If the *bpi* argument is non-zero, the bits per inch value is returned. Otherwise, the bits per mm value is returned.

mt_density_num() returns a numeric value for a text density description. It does a case-insensitive comparison of density names in the density table to the supplied density name.

mt_get_xml_str() gets the current XML status / parameter string from the sa(4) driver instance referenced by the open file descriptor *mtfd*. The mtio(4) ioctl(2) to be used is supplied as the *cmd* argument. Currently the **mt_get_xml_str**() function will work with the MTIOCEXTGET and MTIOCPARAMGET ioctls. The supplied *xml_str* will be filled in with a pointer to the complete XML status string. Multiple calls to the given ioctl(2) are made and more space is malloced until all of the XML string is fetched. The string returned in the *xml_str* argument should be freed when it is no longer in use.

RETURN VALUES

mt_entry_find() returns the first matching entry, or NULL if it fails to find a match.

mt_status_entry_find() returns the first matching entry, or NULL if it fails to find a match.

mt_protect_print() Returns 0 for success, and non-zero for failure. **mt_protect_print**() can only fail if it cannot find protection information in the supplied status data.

mt_param_list() Returns 0 for success and non-zero for failure. **mt_param_list**() can only fail if it cannot find parameter information in the supplied status data.

mt_density_name() returns a text description of a numeric density. The special density value 0 is decoded as "default". The special density value 0x7f is decoded as "same". If the density is not known, **mt_density_name**() will return "UNKNOWN".

mt_density_bp() returns the bits per inch value for the given density (if the *bpi* field is non-zero), the bits per mm value otherwise, or 0 if the supplied *density_num* is not in the density table or the table entry does not include bpi / bpmm values.

mt_density_num() returns a numeric density value between 0 and 255 for the supplied density name. It returns 0 if the density name is not recognized.

mt_get_xml_str() returns 0 for success, and -1 for failure.

SEE ALSO

mt(1), mtio(4), sa(4)

HISTORY

The MT library first appeared in FreeBSD 10.1.

AUTHORS

Ken Merry <ken@FreeBSD.org>

BUGS

The library interface is not complete, and may change in the future. Application authors should not rely on the library interface to be consistent in the immediate future.