NAME

LCP: ALI_alloc_cmd, ALI_alloc_attr, ALI_alloc_string, ALI_receive, ALI_acknowledge, ALI_free,
ALIR_initiate_session, ALIR_alloc_cmd, ALIR_alloc_response, ALIR_alloc_ready, ALIR_alloc_message,
ALIR_alloc_slotinfo, ALIR_alloc_bayinfo, ALIR_alloc_driveinfo, ALIR_alloc_perfinfo, ALIR_send,
ALIR_free, lcp_init - SmartMedia library control functions

SYNOPSIS

#include <lcp.h>
#include <ali.h>
#include <ov_lib.h>

struct ALI_command *ALI_alloc_cmd(enum ALI_cmd command);

void ALI_alloc_attr(struct ALI_attrlist **list, char *name, char *value);

void ALI_alloc_string(struct ALI_stringlist **list, char *string);

struct ALI_command *ALI_receive(char *buffer, time_t timeout);

void ALI_acknowledge(struct ALI_command *cmd);

void ALI_free(struct ALI_command *cmd);

int ALIR_initiate_session(char *hostname, char *instance, char *client,
int flags);

struct ALIR_command *ALIR_alloc_cmd(enum ALI_cmd command);

struct ALIR_command *ALIR_alloc_response(enum ALI_response_type type,
char *errtoken, struct ALI_command *cmd);

struct ALIR_command *ALIR_alloc_ready(enum ALIR_ready_type type,
char *reason);

struct ALIR_command *ALIR_alloc_message(enum ALIR_msg_severity sever,
enum ALIR_msg_whom who);

void ALIR_alloc_slotinfo(struct ALIR_command *cmd, char *slotid,
char *bayid, char *PCL, char *formfactor, int occupied, int access);

void ALIR_alloc_bayinfo(struct ALIR_command *cmd, char *name, int access);

void ALIR_alloc_driveinfo(struct ALIR_command *cmd, char *name,
char *PCL, char *bayid, int occupied, int access);

void ALIR_alloc_perfinfo(struct ALIR_command *cmd, char *name,
char *value);

void ALIR_send(struct ALIR_command *cmd);

void ALIR_free(struct ALIR_command *cmd);

void lcp_init(struct libinfo *libi, lcp_init_private(), lcp_deactivate(),
slot_private(), drive_private(), bay_private(), port_private());
DESCRIPTION

A library control program (LCP) uses abstract library interface (ALI) routines to receive commands from the MLM server, and ALI response (ALI/R) routines to send responses and requests back to the MLM server. ALI and ALI/R routines are described below.

**ALI_alloc_cmd()** allocates memory for an ALI_command structure for the task specified by *command*, and returns a pointer to that ALI_command structure in memory.

**ALI_alloc_attr()** allocates memory for an attribute *name* and *value* pair, and links the attribute-value pair into the ALI_attrlist structure *list*.

**ALI_alloc_string()** allocates memory for *string* and links this into the ALI_stringlist structure *list*.

**ALI_receive()** parses the ALI command string in *buffer*, provided the command arrives before *timeout* expires, and returns a pointer to an ALI_command structure. Usually timeout is set to the polling interval or OV_TIMEOUT_STANDARD (120 seconds).

**ALI_acknowledge()** informs the server that the LCP received the incoming ALI_command structure *cmd* previously returned by **ALI_receive()**.

**ALI_free()** frees up memory allocated for the ALI_command structure for the *cmd* previously returned by **ALI_receive()**.

**ALIR_initiate_session()** begins a session with the MLM server running on *hostname* for a specific *instance* of the LCP running on *client*, with optional *flag* settings.

**ALIR_alloc_cmd()** allocates memory for an ALIR_command structure for the task specified by *command*, and returns a pointer to that ALIR_command structure in memory.

**ALIR_alloc_response()** allocates memory for an ALIR_response structure of ALI_response_type *type* (accepted, unacceptable, success, error, cancelled), possibly with error indication *errtoken*, in reply to the ALI_command structure specified by *cmd*, and returns a pointer to the resulting ALIR_command structure in memory.

**ALIR_alloc_ready()** allocates memory for an ALIR_ready structure of ALI_ready_type *type* (yes, no, lost, broken), with a device-dependent *reason* string if not ready, and returns a pointer to the resulting ALIR_command structure in memory.

**ALIR_alloc_message()** allocates memory for an ALIR_message with a given *severity* (emergency, alert, critical, error, warning, notice, information, debug), intended for somebody *who* (operator, admin, all) reads the message, and returns a pointer to the resulting ALIR_command structure in memory.

Given an existing **ALIR_config** command structure as the *cmd* parameter, **ALIR_alloc_slotinfo()** inserts slotmap information, including *slotid*, *bayid*, *PCL*, *formfactor*, *occupied*, and whether *access* is allowed, into an existing ALIR_command structure.

Given an existing **ALIR_config** command structure as the *cmd* parameter, **ALIR_alloc_bayinfo()** inserts bay information, including *name* and whether *access* is allowed, into an existing ALIR_command structure.

Given an existing **ALIR_config** command structure as the *cmd* parameter, **ALIR_alloc_driveinfo()** inserts drive information, including drive *name*, *PCL*, *bayid*, *occupied*, and whether *access* is allowed, into an existing ALIR_command structure.

Given an existing **ALIR_config** command structure as the *cmd* parameter, **ALIR_alloc_perfinfo()** inserts performance information, including trait *name* and *value*, into an existing ALIR_command structure.

**ALIR_send()** transmits an ALIR_command structure back to the MLM server.

**ALIR_free()** frees up memory allocated for the ALIR_command structure for the *cmd* previously returned by **ALIR_alloc_cmd()** or equivalent.

**lcp_init()** sets up library information structures. The first three parameters are mandatory: *libi* pointing to a libinfo structure, and the **lcp_init_private()** and **lcp_deactivate()** procedures to initialize and deactivate library hardware. The remaining parameters may be NULL, or may specify procedures for handling slots, drives,
bays, and ports.

**DIAGNOSTICS**
In the event of failure, these routines generally return null values and set system error codes as defined in the `<errno.h>` include file.

**SEE ALSO**
- umsh(1M), AAPI(3), CAPI(3), DCP(3)