

# Aculab Resource management API guide

MAN1774 Revision 6.4.8



## PROPRIETARY INFORMATION

The information contained in this document is the property of Aculab plc and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission. It should not be used for commercial purposes without prior agreement in writing.

All trademarks recognised and acknowledged.

Aculab plc endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission.

The development of Aculab's products and services is continuous and published information may not be up to date. It is important to check the current position with Aculab plc.

Copyright © Aculab plc. 2002-2010 all rights reserved.

## Document Revision

Rev	Date	By	Detail
6.0	06.12.02	DJL	Interim release
6.1	04.06.03	DJL	Controlled release
6.1.1	15.01.04	DJL	Review updates
6.2.2	08.09.04	DJL	Beta release
6.2.2	15.09.04	DJL	Full release
6.2.3	14.10.04	DJL	Addition of Linux information
6.3.0	24.12.04	DJL	Various updates including support for new hardware
6.3.1	25.02.05	DJL	Correction to code example
6.3.2	11.04.05	DJL	Update to <code>resources_available</code> parameter table
6.4.0	19.09.05	DJL	Addition of a number of new functions
6.4.1	18.01.06	DJL	Small changes plus addition of some new parameter values
6.4.2	15.03.06	DJL	Additional information added to <code>acu_open_prosody()</code>
6.4.3	18.05.06	DJL	<code>acu_error_desc()</code> return string information updated
6.4.4	17.01.07	DJL	Linux and Solaris updates
6.4.5	14.12.07	MJW	IP conflict and <code>acu_prosody_ip_get_local_card_info</code>
6.4.6	12.10.10	DF	Removed reference to EOL cards
6.4.7	11.11.10	EBJ	Updated to corporate fonts.
6.4.8	05.01.11	EBJ	Removal of Hyperlinks

## CONTENTS

1	Introduction.....	5
2	Using the Resource Management API .....	6
2.1	Linking with the Resource Management Library.....	6
2.2	Finding cards.....	6
2.3	Opening cards .....	7
2.4	Determining a card’s capabilities .....	8
3	API call summary and description .....	9
3.1	acu_get_system_snapshot() .....	10
3.2	acu_open_card() .....	11
3.3	acu_close_card() .....	13
3.4	acu_open_call() .....	14
3.5	acu_close_call() .....	16
3.6	acu_open_switch() .....	16
3.7	acu_close_switch() .....	17
3.8	acu_open_mg().....	17
3.9	acu_close_mg() .....	18
3.10	acu_open_ipt().....	18
3.11	acu_close_ipt() .....	19
3.12	acu_open_prosody() .....	20
3.13	acu_close_prosody() .....	21
3.14	acu_identify_card() .....	21
3.15	acu_eject_card().....	22
3.16	acu_set_card_name().....	24
3.17	acu_get_card_info() .....	25
3.18	acu_set_system_notification_queue() .....	27
3.19	acu_get_system_notification().....	28
3.20	acu_get_system_notification_wait_object() .....	30
3.21	acu_set_card_notification_queue() .....	31
3.22	acu_get_card_notification() .....	32
3.23	acu_get_card_notification_wait_object() .....	33
3.24	acu_allocate_event_queue() .....	35
3.25	acu_free_event_queue() .....	36
3.26	acu_get_event_from_queue() .....	37
3.27	acu_get_event_queue_wait_object().....	38
3.28	acu_get_card_app_context_token() .....	39
3.29	acu_set_card_app_context_token() .....	40
3.30	acu_get_res_api_version() .....	41
3.31	acu_get_res_mgr_version() .....	42
3.32	acu_get_card_configuration_state() .....	43
3.33	acu_get_card_state_desc() .....	44
3.34	acu_error_desc() .....	44
3.35	acu_get_aculab_directory().....	45
3.36	acu_register_prosody_ip_card().....	46
3.37	acu_configure_prosody_ip_card() .....	47
3.38	acu_get_prosody_ip_card_config().....	49
3.39	acu_unregister_prosody_ip_card() .....	50
3.40	acu_prosody_ip_get_device_info() .....	51
3.41	acu_prosody_ip_get_registered_cards() .....	52
3.42	acu_prosody_ip_card_restart().....	53

3.43 acu_prosody_ip_card_status() .....	54
3.44 acu_prosody_ip_get_local_card_info() .....	55
Appendix A: Error codes .....	56

## 1 Introduction

This document describes the Aculab Resource Management API.

The Resource Management API is used to enumerate the cards found in a system, to manage and detect hot-swap events, and to open cards for use with other Aculab APIs.

This API guide should be read in conjunction with the following Aculab version 6.\* documents:

- Call control API guide
- Switch API guide
- Prosody API guide

## 2 Using the Resource Management API

This section describes the general usage of the Resource Management API.

### 2.1 Linking with the Resource Management Library

All V6 applications must `#include` the file `acu_type.h`. This contains types and error codes that are common to all the Aculab APIs. It must be `#included` before any of the other API header files.

#### NOTE

**When using Linux, you must define `ACU_LINUX` before including `acu_type.h`**

To use the Resource Management API, the file `res_lib.h` needs to be included. This contains the structure definitions and function prototypes for the Resource Management API. On Windows, the implementation for the Resource Management API resides in the dynamic link library `res_lib.dll`. To use it, you must link the file `res_lib.lib` with your application.

### 2.2 Finding cards

In V6, cards are initially identified by their serial number. Using a serial number, an application can obtain a card id to represent the card, which can be used in API calls to affect that card.

Applications that are expected to share resources with other applications on the same system, will typically need to know the serial numbers of the cards to use. An application can find out which cards are available in the system using the `acu_get_system_snapshot()` API call. This function returns the number of cards in the system and an array of their serial numbers.

```
ACU_SNAPSHOT_PARMS snapshot_parms;
ACU_ERR result;
ACU_UINT c;

INIT_ACU_STRUCT(&snapshot_parms);
result = acu_get_system_snapshot(&snapshot_parms);
if (result == 0)
{
    for (c = 0; c < snapshot_parms.count; c++)
    {
        printf("Found card %s\n",
            snapshot_parms.serial_no[c]);
    }
}
```

## 2.3 Opening cards

Once the serial number of a card is known, the card can be opened for use. This is achieved using the `acu_open_card()` API call. Opening a card allocates an id for that card. The card id can be used in API calls that operate on the card. Obtaining a card id is the first step towards opening a resource such as a network port or a Prosody module.

```
char* serial_no = "1234567";
ACU_OPEN_CARD_PARMS open_card_parms;
ACU_CLOSE_CARD_PARMS close_card_parms;
ACU_ERR result;

INIT_ACU_STRUCT(&open_card_parms);
strncpy(open_card_parms.serial_no, serial_no,
ACU_MAX_SERIAL);

result = acu_open_card(&open_card_parms);
if (result == 0)
{
    printf("Opened card %s and got card id 0x%X\n",
        open_card_parms.serial_no,
        open_card_parms.card_id);
    ...
    /* use card here */
    ...
    INIT_ACU_STRUCT(&close_card_parms);
    close_card_parms.card_id =
    open_card_parms.card_id;
    result = acu_close_card(&close_card_parms);
}
```

Acquiring a card uses system resources, and in hot swap systems, will prevent the clean removal of that card. When an application has finished with a card it must close the associated card id, which will release the resources allocated for that card.

### NOTE

**It is not possible to close a card while there are still resources allocated on it (e.g. call handles, Prosody modules, etc.).**

A card is closed using the `acu_close_card()` API call.

## 2.4 Determining a card's capabilities

Once you have identified the cards in the system, you can go on to determine what capabilities of each card. The API call for this is `acu_get_card_info()` which can only be called using a valid card id. Using this function, you can find out whether a card can be used for calls and/or speech.

```

ACU_CARD_ID card_id; /* id of a previously opened card */
ACU_CARD_INFO_PARMS card_info_parms;
ACU_ERR result;
INIT_ACU_STRUCT(&card_info_parms);

card_info_parms.card_id = card_id;
result = acu_get_card_info(&card_info_parms);

if (result == 0)
{
    printf("Card capabilities:\n");
    if (card_info_parms.resources_available & ACU_RESOURCE_SWITCH)
    {
        printf("\tSWITCH\n");
    }
    if (card_info_parms.resources_available & ACU_RESOURCE_CALL)
    {
        printf("\tCALL\n");
    }
    if (card_info_parms.resources_available & ACU_RESOURCE_SPEECH)
    {
        printf("\tSPEECH\n");
    }
}

```

Before it can make use of a card's capabilities, an application must open the card for each capability it wants to use. The API calls available for this are:

```

acu_open_switch()
acu_open_call()
acu_open_prosody()

```

To open a card for switch, call control, and Prosody you would do the following:

```

ACU_CARD_ID card_id; /* a previously opened card */
ACU_OPEN_SWITCH_PARMS open_switch_parms;
ACU_OPEN_CALL_PARMS open_call_parms;
ACU_OPEN_PROSODY_PARMS open_prosody_parms;
ACU_ERR result;

INIT_ACU_STRUCT(&open_switch_parms);
INIT_ACU_STRUCT(&open_call_parms);
INIT_ACU_STRUCT(&open_prosody_parms);

open_switch_parms.card_id = card_id;
result = acu_open_switch(&open_switch_parms);
if (result != 0)
{
    printf("Error %d opening switch API\n", result);
    exit(EXIT_FAILURE);
}

open_call_parms.card_id = card_id;
result = acu_open_call(&open_call_parms);
if (result != 0)
{
    printf("Error %d opening call API\n", result);
    exit(EXIT_FAILURE);
}

open_prosody_parms.card_id = card_id;
result = acu_open_prosody(&open_prosody_parms);
if (result != 0)
{
    printf("Error %d opening Prosody API\n", result);
    exit(EXIT_FAILURE);
}

```

### 3 API call summary and description

This section describes the structures for the following calls:

#### Enumeration and opening cards

API	Description
<code>acu_get_system_snapshot()</code>	Returns details of cards currently present in the system
<code>acu_open_card()</code>	Makes a card available for use by the application
<code>acu_close_card()</code>	Detaches from a card
<code>acu_open_call()</code>	Opens a card for use with the Aculab Call API
<code>acu_close_call()</code>	Closes the call driver for a given card
<code>acu_open_switch()</code>	Opens the switch driver for the specified card
<code>acu_close_switch()</code>	Detaches from the Switch API for a given card
<code>acu_open_prosody()</code>	Opens the Prosody driver for a given card
<code>acu_close_prosody()</code>	Detaches from the Prosody driver for a given card
<code>acu_open_mg()</code>	Opens the Media Gateway API for a given card
<code>acu_close_mg()</code>	Detaches from the Media Gateway API for a given card.
<code>acu_open_ipt()</code>	Opens the IP Telephony API for a given card
<code>acu_close_ipt()</code>	Closes the IP Telephony API for a given card
<code>acu_identify_card()</code>	Aids identification of a card by illuminating LEDs on the card.
<code>acu_eject_card()</code>	Marks an Aculab card for ejection
<code>acu_get_card_info()</code>	Returns information about a given card

#### Miscellaneous functions

API	Description
<code>acu_set_system_notification_queue()</code>	Set an event queue
<code>acu_get_system_notification()</code>	Retrieves the next event in the queue
<code>acu_get_system_notification_wait_object()</code>	Gets a wait object associated with the application's global system event queue
<code>acu_get_card_notification()</code>	Retrieves an event that is associated with a particular card
<code>acu_get_card_notification_wait_object()</code>	Returns a wait object that can be used to wait for card-specific notification events
<code>acu_allocate_event_queue()</code>	Creates a new event queue
<code>acu_free_event_queue()</code>	Releases an event queue
<code>acu_get_event_from_queue()</code>	Retrieves an event from an event queue
<code>acu_get_event_queue_wait_object()</code>	Retrieves a platform-specific wait object associated with an event queue
<code>acu_get_card_app_context_token()</code>	Retrieves application-defined data that has been associated with a card
<code>acu_set_card_app_context_token()</code>	Associates application-defined data with a card
<code>acu_error_desc()</code>	Provides a descriptive string for an Aculab error code.

## Prosody IP card management functions

API	Description
<code>acu_register_prosody_ip_card()</code>	This function is used to register a remote Prosody X card with the system. This allows the system to configure and use Prosody X cards located in another chassis.
<code>acu_configure_prosody_ip_card()</code>	This function is used to change the configuration of a registered Prosody IP card.
<code>acu_get_prosody_ip_card_config()</code>	This function is used to retrieve the configuration of a registered Prosody IP card.
<code>acu_unregister_prosody_ip_card()</code>	This function is used to remove the registration for a Prosody X card.
<code>acu_get_prosody_ip_device_info()</code>	This function returns diagnostic information about the hardware and software configuration of a Prosody X or PMX device.
<code>acu_prosody_ip_get_registered_cards()</code>	This function populates a struct with a collection of the cards currently registered with the system.

## Enumerating and opening cards

### 3.1 `acu_get_system_snapshot()`

This function populates a struct with a collection of the cards currently present in the system.

#### NOTE

**It is possible that in between obtaining the snapshot and trying to use it, that the system configuration could change. An application should be designed with this in mind.**

### Synopsis

```
ACU_ERR acu_get_system_snapshot(ACU_SNAPSHOT_PARMS* cardsp)
```

```
typedef struct _ACU_SNAPSHOT_PARMS
{
    ACU_ULONG    size;                /* IN */
    ACU_UINT     count;              /* OUT */
    ACU_CHAR     serial_no[MAX_SNAPSHOT_CARDS][ACU_MAX_SERIAL]; /* OUT */
} ACU_SNAPSHOT_PARMS;
```

The function `acu_get_system_snapshot()` takes a pointer *cardsp*, to a structure `ACU_SNAPSHOT_PARMS`. The structure must be initialised before invoking the function.

### Input Parameters

*size*

Must be initialised to the size of the structure.

### Return Values

*count*

The number of cards in the snapshot. The structure contains an array of information about each of the cards in the system. This is the number of elements in the *serial\_no* array that are populated.

***serial\_no***

Contains an array of the serial numbers of the cards in the snapshot.

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

**Example Usage**

```
ACU_SNAPSHOT_PARMS system_snapshot;
ACU_ERR error = 0;
ACU_INT card;

INIT_ACU_STRUCT(&system_snapshot);

error = acu_get_system_snapshot(&system_snapshot);
if (error == 0)
{
    /* iterate through all the reported cards */
    for (card = 0; card < system_snapshot.count; card++)
    {
        printf("Found card %s\n", system_snapshot.serial_no[card]);
    }
}
```

**3.2 acu\_open\_card()**

This function makes a card available for use by the application. This function must be called before it is possible to open any of the resources belonging to the card.

When an application has finished with a card, it must call `acu_close_card()` to release it.

**NOTE**

**An application must make a matching call to `acu_close_card()` for every call to `acu_open_card()`.**

**Synopsis**

```
ACU_ERR acu_open_card(ACU_OPEN_CARD_PARMS* openp);
```

```
typedef struct _ACU_OPEN_CARD_PARMS
{
    ACU_ULONG          size;                          /* IN */
    ACU_CHAR           serial_no[ACU_MAX_RESOURCE_NAME]; /* IN */
    ACU_CARD_ID       card_id;                        /* OUT */
    ACU_EVENT_QUEUE   notification_queue;            /* IN */
    ACU_ACT           app_context_token;             /* IN */
} ACU_OPEN_CARD_PARMS;
```

The function `acu_open_card()` takes a pointer `openp`, to a structure `ACU_OPEN_CARD_PARMS`. The structure must be initialised before invoking the function.

**Input Parameters*****size***

Must be initialised to the size of the structure.

***serial\_no***

Is the serial number, name (as specified using `acu_set_card_name`) or network address of the card to open (one way of obtaining the serial number is from the function `acu_get_system_snapshot`). This is a null terminated string.

***notification\_queue***

Can be set to the id of an event queue. This event queue will be used to report events that occur on the card (e.g. hot swap events). If this field is left set to 0, the card will not be associated with any event queue. Events for this card will still be available through `acu_get_card_notification()`.

***app\_context\_token***

Can be set to an application-defined value to be associated with this card. This value can be retrieved using `acu_get_card_info()` or `acu_get_card_app_context_token()` and will also be returned when notification events occur on this card. The value can be changed at a later time using the `acu_set_card_app_context_token()` function.

**Return Values*****card\_id***

Returns a value that can be used in other API calls to represent the card.

**NOTE**

**The card id is not transferable between processes. No attempt should be made to decompose or otherwise interpret the value. It is up to the application to keep track of the card ids for cards that it has opened.**

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

**Example Usage**

Opening a card for use:

```
ACU_SNAPSHOT_PARMS system_snapshot;
ACU_OPEN_CARD_PARMS open_params;
ACU_ERR error = 0;
ACU_UINT card;

INIT_ACU_STRUCT(&system_snapshot);

error = acu_get_system_snapshot(&system_snapshot);

if (error == 0)
{
    /* iterate through all the reported cards */
    for (card = 0; card < system_snapshot.count; card++)
    {
        /* open the card and store the id for use later on */
        INIT_ACU_STRUCT(&open_params);

        strcpy(open_params.serial_no, system_snapshot.serial_no[card]);

        error = acu_open_card(&open_params);

        if (error == 0)
        {
            /* store the card id */
        }
    }
}
```

See `acu_open_call`, `acu_open_switch`, `acu_open_prosody`, and `acu_get_card_info` for examples of using the card id.

### 3.3 acu\_close\_card()

This function is used by an application to detach from a card. This function should only be called after an application has relinquished any other resources it previously allocated on the card (including calls and Prosody channels).

#### NOTE

**Every call to `acu_open_card()` must have a matching call to `acu_close_card()`.**

#### Synopsis

```
ACU_ERR acu_close_card(ACU_CLOSE_CARD_PARMS* close_card_parms);
```

```
typedef struct _ACU_CLOSE_CARD_PARMS
{
    ACU_ULONG      size;                /* IN */
    ACU_CARD_ID   card_id;            /* IN */
} ACU_CLOSE_CARD_PARMS;
```

The function `acu_close_card()` takes a pointer `close_card_parms`, to a structure `ACU_CLOSE_CARD_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *size*

Must be initialised to the size of the structure.

##### *card id*

This function takes the *card id* (as returned by `acu_open_card()`) of the card to close.

#### Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

#### Example Usage

```
ACU_OPEN_CARD_PARMS open_parms;
ACU_CLOSE_CARD_PARMS close_parms;
ACU_ERR error = 0;

INIT_ACU_STRUCT(&open_parms);

/* In this example we have already obtained the card's serial number
from elsewhere */
strcpy(open_parms.serial_no, serial_no);
error = acu_open_card(&open_parms);

if (error != 0)
{
    printf("Failed opening card %s with error %d\n", serial_no, error);
    exit(-1);
}

/* ... */
/* application uses the card's resources */
/* ... */

INIT_ACU_STRUCT(&close_parms);
close_parms.card_id = open_parms.card_id;
error = acu_close_card(&close_parms);
```

## 3.4 acu\_open\_call()

This function is used to open a card for use with the Aculab Call API.

### Synopsis

```
ACU_ERR acu_open_call(ACU_OPEN_CALL_PARMS* openp);
```

```
typedef struct _ACU_OPEN_CALL_PARMS
{
    ACU_ULONG      size;          /* IN */
    ACU_CARD_ID    card_id;       /* IN */
} ACU_OPEN_CALL_PARMS;
```

The function `acu_open_card()` takes a pointer `openp`, to a structure `ACU_OPEN_CALL_PARMS`. The structure must be initialised before invoking the function.

### Input Parameters

**size**

Must be initialised to the size of the structure.

**card\_id**

Must be set to a valid card id returned by the `acu_open_card()` function.

### Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### Example Usage

```
ACU_OPEN_CARD_PARMS open_card_parms;
ACU_OPEN_CALL_PARMS open_call_parms;
OPEN_PORT_PARMS open_port_parms;
ACU_CLOSE_CARD_PARMS close_card_parms;
ACU_CLOSE_CALL_PARMS close_call_parms;
CLOSE_PORT_PARMS close_port_parms;

OUT_XPARMS openout_parms;
CAUSE_XPARMS cause_parms;

ACU_CHAR* serial_no = "123456";
ACU_CHAR* dest_address = "0818118181";
ACU_CHAR* orig_address = "987654321";
ACU_ERR error = 0;

/* initialise Aculab structures before use */
INIT_ACU_STRUCT(&open_card_parms);
INIT_ACU_STRUCT(&open_call_parms);
INIT_ACU_STRUCT(&open_port_parms);
INIT_ACU_STRUCT(&close_card_parms);
INIT_ACU_STRUCT(&close_call_parms);
INIT_ACU_STRUCT(&close_port_parms);
INIT_ACU_STRUCT(&openout_parms);
INIT_ACU_STRUCT(&cause_parms);

strcpy(open_card_parms.serial_no, serial_no);

error = acu_open_card(&open_card_parms);
if (error != 0)
{
    printf("Failed opening card %s with error %d\n", serial_no, error);
    exit(-1);
}

open_call_parms.card_id = open_card_parms.card_id;
error = acu_open_call(&open_call_parms);

if (error != 0)
```

```
{
    printf("Failed opening Call API for card %s with error %d\n",
        serial_no, error);
    close_card_parms.card_id = open_card_parms.card_id;
    acu_close_card(&close_card_parms);
    exit(-1);
}

open_port_parms.card_id = open_call_parms.card_id;
open_port_parms.port_ix = 0;

error = call_open_port(&open_port_parms);
if (error != 0)
{
    printf("Failed opening port %d on card %s with error %d\n",
        open_port_parms.port_ix, serial_no, error);

    close_call_parms.card_id = open_card_parms.card_id;
    acu_close_call(&close_call_parms);

    close_card_parms.card_id = open_card_parms.card_id;
    acu_close_card(&close_card_parms);

    exit(-1);
}

openout_parms.net = open_port_parms.port_id;
openout_parms.ts = -1;
openout_parms.cnf = CNF_REM_DISC;
strcpy(openout_parms.destination_addr, dest_address);
strcpy(openout_parms.originating_addr, orig_address);

error = call_openout(&openout_parms);

if (error != 0)
{
    printf("Failed opening outgoing call with error %d\n", error);

    close_port_parms.port_id = open_port_parms.port_id;
    call_close_port(&close_port_parms);

    close_call_parms.card_id = open_call_parms.card_id;
    acu_close_call(&close_call_parms);

    close_card_parms.card_id = open_card_parms.card_id;
    acu_close_card(&close_card_parms);

    exit(-1);
}

/* ... use call handle here and wait for it to go idle ... */

cause_parms.handle = openout_parms.handle;
cause_parms.cause = LC_NORMAL;

error = call_release(&cause_parms);

if (error != 0)
{
    printf("Failed releasing call 0x%X, error %d\n",
        openout_parms.handle, error);
}

close_port_parms.port_id = open_port_parms.port_id;
call_close_port(&close_port_parms);

close_call_parms.card_id = open_call_parms.card_id;
acu_close_call(&close_call_parms);

close_card_parms.card_id = open_card_parms.card_id;
acu_close_card(&close_card_parms);
```

### 3.5 acu\_close\_call()

This function is used by an application to close the call driver for a given card. This function should only be called after an application has relinquished any resources it previously allocated using the call API (including call handles and ports).

#### NOTE

**Do not rely on this function to clean up resources that an application has neglected to release.**

#### Synopsis

```
ACU_ERR acu_close_call(ACU_CLOSE_CALL_PARMS* closep);
```

```
typedef struct _ACU_CLOSE_CALL_PARMS
{
    ACU_ULONG      size;          /* IN */
    ACU_CARD_ID   card_id;       /* IN */
} ACU_CLOSE_CALL_PARMS;
```

The function `acu_close_call()` takes a pointer `closep`, to a structure `ACU_CLOSE_CALL_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

**size**

Must be initialised to the size of the structure.

**card\_id**

Must be set to a valid call ID returned from the `acu_open_card()` function.

#### Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

#### Example Usage

See the example under `acu_open_call()` for an example of how to use this API call.

### 3.6 acu\_open\_switch()

This function is used to open the switch driver for the specified card.

#### Synopsis

```
ACU_ERR acu_open_switch(ACU_OPEN_SWITCH_PARMS* openp);
```

```
typedef struct _ACU_OPEN_SWITCH_PARMS
{
    ACU_ULONG      size;          /* IN */
    ACU_CARD_ID   card_id;       /* IN */
} ACU_OPEN_SWITCH_PARMS;
```

The function `acu_open_switch()` takes a pointer `openp`, to a structure `ACU_OPEN_SWITCH_PARMS`. The structure must be initialised before invoking the function.

## Input Parameters

### *size*

Must be initialised to the size of the structure.

### *card\_id*

Must be set to a valid card id returned by the `acu_open_card()` function.

## Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## 3.7 `acu_close_switch()`

This function is used by an application to detach from the Switch API for a given card.

### Synopsis

```
ACU_ERR acu_close_switch(ACU_CLOSE_SWITCH_PARMS* closep);
```

```
typedef struct _ACU_CLOSE_SWITCH_PARMS
{
    ACU_ULONG      size;          /* IN */
    ACU_CARD      card_id;       /* IN */
} ACU_CLOSE_SWITCH_PARMS;
```

The function `acu_close_switch()` takes a pointer `closep`, to a structure `ACU_CLOSE_SWITCH_PARMS`. The structure must be initialised before invoking the function.

### Input Parameters

#### *size*

Must be initialised to the size of the structure.

#### *card\_id*

Must be set to a valid card id returned from the `acu_open_card()` for which `acu_open_switch()` has been called.

### Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## 3.8 `acu_open_mg()`

This function is used to open the specified card for use with the Media Gateway API.

### Synopsis

```
ACU_ERR acu_open_mg(ACU_OPEN_MG_PARMS* openp);
```

```
typedef struct _ACU_OPEN_MG_PARMS
{
    ACU_ULONG      size;          /* IN */
    ACU_CARD_ID    card_id;       /* IN */
} ACU_OPEN_MG_PARMS;
```

The function `acu_open_mg()` takes a pointer `openp`, to a structure `ACU_OPEN_MG_PARMS`. The structure must be initialised before invoking the function.

## Input Parameters

### *size*

Must be initialised to the size of the structure.

### *card\_id*

Must be set to a valid card id returned by the `acu_open_card()` function.

## Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## 3.9 `acu_close_mg()`

This function is used by an application to detach from the Media Gateway API for a given card.

### Synopsis

```
ACU_ERR acu_close_mg(ACU_CLOSE_MG_PARMS* closep);

typedef struct _ACU_CLOSE_MG_PARMS
{
    ACU_ULONG          size;          /* IN */
    ACU_CARD           card_id;       /* IN */
} ACU_CLOSE_MG_PARMS;
```

The function `acu_close_mg()` takes a pointer `closep`, to a structure `ACU_CLOSE_MG_PARMS`. The structure must be initialised before invoking the function.

### Input Parameters

#### *size*

Must be initialised to the size of the structure.

#### *card\_id*

Must be set to a valid card id returned from the `acu_open_card()` for which `acu_open_mg()` has been called.

### Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## 3.10 `acu_open_ipt()`

This function is used to open the specified card for use with the IP Telephony API.

### Synopsis

```
ACU_ERR acu_open_ipt(ACU_OPEN_IPTEL_PARMS* openp);

typedef struct _ACU_OPEN_IPTEL_PARMS
{
    ACU_ULONG          size;          /* IN */
    ACU_CARD_ID       card_id;       /* IN */
} ACU_OPEN_IPTEL_PARMS;
```

The function `acu_open_ipt()` takes a pointer `openp`, to a structure `ACU_OPEN_IPTEL_PARMS`. The structure must be initialised before invoking the function.

### Input Parameters

**size**

Must be initialised to the size of the structure.

**card\_id**

Must be set to a valid card id returned by the `acu_open_card()` function.

### Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## 3.11 `acu_close_ipt()`

This function is used by an application to detach from the IP Telephony API for a given card.

### Synopsis

```
ACU_ERR acu_close_ipt(ACU_CLOSE_IPTTEL_PARMS* closep);
```

```
typedef struct _ACU_CLOSE_IPTTEL_PARMS
{
    ACU_ULONG      size;          /* IN */
    ACU_CARD       card_id;       /* IN */
} ACU_CLOSE_IPTTEL_PARMS;
```

The function `acu_close_ipt()` takes a pointer `closep`, to a structure `ACU_CLOSE_IPTTEL_PARMS`. The structure must be initialised before invoking the function.

### Input Parameters

**size**

Must be initialised to the size of the structure.

**card\_id**

Must be set to a valid card id returned from the `acu_open_card()` for which `acu_open_ipt()` has been called.

### Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.12 acu\_open\_prosody()

This function is used to open the Prosody driver for a given card.

#### NOTE

**Should this function fail despite `acu_get_card_info()` indicating that there are Prosody resources on a card, this is likely to be due to a disabled device driver.**

#### NOTE

**When linking TiNG statically on Windows (i.e. having defined `ACU_DONT_USE_TING_DLL`) this function should not be called and, instead, one of the following functions from the TiNG API should be used dependent on the target platform:**

- Prosody S:** `sm_open_prosody_s()` - passing in the Prosody S server's port (the `ACULAB_PROSODYS_PORT` environment variable), generally 7780;
- Prosody X:** `sm_open_prosody_x()` - passing in the card's IP address which may be obtained from a call to `acu_get_card_info()`.

### Synopsis

```
ACU_ERR acu_open_prosody(ACU_OPEN_PROSODY_PARMS* openp);
```

```
typedef struct _ACU_OPEN_PROSODY_PARMS
{
    ACU_ULONG          size;           /* IN */
    ACU_CARD_ID       card_id;       /* IN */
} ACU_OPEN_PROSODY_PARMS;
```

The function `acu_open_prosody()` takes a pointer `openp`, to a structure `ACU_OPEN_PROSODY_PARMS`. The structure must be initialised before invoking the function.

#### NOTE

**Almost all uses of a Prosody X card require the `datafeed` firmware module to be downloaded**

### Input Parameters

*size*

Must be initialised to the size of the structure.

*card\_id*

Must be set to a valid card id returned by the `acu_open_card()` function.

### Return Values

On successful completion a value of zero is returned, it will then be possible to use the card id passed in with functions in the Prosody API (e.g. `sm_open_module()`, `sm_get_card_info()`); otherwise a negative values is returned indication the type of error.

### 3.13 acu\_close\_prosody()

This function is used by an application to detach from the Prosody driver for a given card. This function should only be called after an application has relinquished any resources it previously allocated using the Prosody API (including modules and channels).

#### Synopsis

```
ACU_ERR acu_close_prosody(ACU_CLOSE_PROSODY_PARMS* closep);

typedef struct _ACU_CLOSE_PROSODY_PARMS
{
    ACU_ULONG      size;          /* IN */
    ACU_CARD_ID    card_id;       /* IN */
} ACU_CLOSE_PROSODY_PARMS;
```

The function `acu_close_prosody()` takes a pointer `closep`, to a structure `ACU_CLOSE_PROSODY_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *size*

Must be initialised to the size of the structure.

##### *card\_id*

Must be set to a valid card id returned from the `acu_open_card()` function which has previously been used in a call to `acu_open_prosody()`.

#### Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.14 acu\_identify\_card()

This function is used to aid identification of a card from outside of the chassis it is in. Typically this involves illuminating one or more of the LEDs on the card.

#### NOTE

**This function is only applicable to cards that have externally visible LEDs.**

#### Synopsis

```
ACU_ERR acu_identify_card(ACU_IDENTIFY_PARMS* identify);

typedef struct _ACU_IDENTIFY_PARMS
{
    ACU_UINT       size;          /* IN */
    ACU_CARD_ID    card_id;       /* IN */
    ACU_UINT       identify_on;   /* IN */
} ACU_IDENTIFY_PARMS;
```

The function `acu_identify_card()` takes a pointer `identify`, to a structure `ACU_IDENTIFY_PARMS`. The structure must be initialised before invoking the function.

## Input Parameters

### *size*

Must be initialised to the size of the structure.

### *card\_id*

Must be set to the card id of the card to identify.

### *identify\_on*

Is a flag indicating whether the card lights are to be turned on or off. The possible values for this flag are:

#define	Description
ACU_IDENTIFY_CARD_ON	Turn the card lights on
ACU_IDENTIFY_CARD_OFF	Turn the card lights off

## Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## Example Usage

```
#define ILLUMINATE_DURATION 10000

ACU_CARD_ID card_id; /* id of a previously opened card */
ACU_IDENTIFY_PARMS identify_parms;
ACU_INT error = 0;

INIT_ACU_STRUCT(&identify_parms);

identify_parms.card_id = card_id;
identify_parms.identify_on = ACU_IDENTIFY_CARD_ON;

error = acu_identify_card(&identify_parms);

if (error == 0)
{
    Sleep(ILLUMINATE_DURATION);

    identify_parms.identify_on = ACU_IDENTIFY_CARD_OFF;

    error = acu_identify_card(&identify_parms);

    if (error != 0)
    {
        printf("Error %d turning off card illumination for card %d\n");
    }
}
```

### 3.15 acu\_eject\_card()

This function marks an Aculab card for ejection. When this API call is used, all applications with the card open (including the one making the `acu_eject_card()` API call) receive a `ACU_CARD_EVT_CARD_REMOVE_PENDING` notification (obtained by calling `acu_get_card_notification()`).

When all applications that have the card open have released their resources and closed the card, the flashing blue light indicates it is safe to eject the card. In the meantime, applications attempting to open the card after this API call has been made will receive the `ERR_CARD_EJECT_PENDING` error.

#### NOTE

**This function is currently only applicable to Aculab's remote Prosody X cards registered with the system.**

**NOTE**

**In addition to using this API call, additional platform and/or hardware specific steps may be required before the card can be removed from the system. Consult the documentation for your platform and for the hardware you want to use.**

**Synopsis**

```
ACU_ERR acu_eject_card(ACU_EJECT_CARD_PARMS* ejectp);

typedef struct _ACU_EJECT_CARD_PARMS
{
    ACU_ULONG      size;          /* IN */
    ACU_CARD_ID   card_id;       /* IN */
} ACU_EJECT_CARD_PARMS;
```

The function `acu_eject_card()` takes a pointer `ejectp`, to a structure `ACU_EJECT_CARD_PARMS`. The structure must be initialised before invoking the function.

**Input Parameters*****size***

Must be initialised to the size of the structure.

***card\_id***

Must be set to the card id of the card to eject.

**Return Values**

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

**Example Usage**

```
ACU_CARD_ID card_id; /* card id previously opened elsewhere */
ACU_ERR error = 0;
ACU_EJECT_CARD_PARMS eject_parms;

INIT_ACU_STRUCT(&eject_parms);
eject_parms.card_id = card_id;

error = acu_eject_card(&eject_parms);

if (error == 0)
{
    printf("Eject pending for card 0x%X\n", card_id);
}
else
{
    printf("Eject of card 0x%X failed with error %d\n", card_id, error);
}
```

### 3.16 acu\_set\_card\_name()

This function is used to set the name of a card. A name set in this way can be retrieved through the `acu_get_card_info()` API call.

This function is primarily intended to aid users in distinguishing between cards through an application interface.

#### NOTE

**The name set using this function is not persistent and will be lost when the system restarts.**

#### Synopsis

```
ACU_ERR acu_set_card_name(ACU_SET_CARD_NAME_PARMS* namep);
```

```
typedef struct tACU_SET_CARD_NAME_PARMS
{
    ACU_ULONG      size;                /* IN */
    ACU_CARD_ID   card_id;             /* IN */
    ACU_CHAR      card_name[ACU_MAX_CARD_NAME]; /* IN */
} ACU_SET_CARD_NAME_PARMS;
```

#### Input parameters

The function `acu_set_card_name()` takes a pointer `namep`, to a structure `ACU_SET_CARD_NAME_PARMS`. The structure must be initialised before invoking the function.

##### *Size*

This is the size of the struct, initialised using `INIT_ACU_STRUCT`.

##### *card\_id*

This is the card id of the card (as returned by `acu_open_card()`)

##### *card\_name*

This is a 0-terminated ASCII string containing the card name.

#### Return values

On successful completion, a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.17 acu\_get\_card\_info()

This function returns information about the given card.

#### Synopsis

```
ACU_ERR acu_get_card_info(ACU_CARD_INFO_PARMS* infop);
```

```
typedef struct _ACU_CARD_INFO_PARMS
{
    ACU_ULONG          size;                /* IN */
    ACU_CARD_ID       card_id;             /* IN */
    ACU_UINT          card_type;           /* OUT */
    ACU_UINT          resources_available; /* OUT */
    ACU_CHAR          serial_no[ACU_MAX_SERIAL]; /* OUT */
    ACU_CHAR          hw_version[ACU_MAX_HWVER]; /* OUT */
    ACU_ACT           app_context_token;    /* OUT */
    ACU_CARD_POSITION hw_position;         /* OUT */
    ACU_CHAR          card_name[ACU_MAX_CARD_NAME]; /* OUT */
    ACU_CHAR          card_desc[ACU_MAX_CARD_DESC]; /* OUT */
    ACU_CHAR          ip_address[ACU_MAX_IP_ADDRESS]; /* OUT */
    ACU_CHAR          card_key[ACU_MAX_CARD_KEY]; /* OUT */
} ACU_CARD_INFO_PARMS;
```

The function `acu_get_card_info()` takes a pointer `infop`, to a structure `ACU_CARD_INFO_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *size*

Must be initialised to the size of the structure.

##### *card\_id*

Should be set to the id of the card to query (obtained from `acu_open_card()`).

#### Return Values

##### *serial\_no*

Contains the serial number of the card as a null terminated string.

##### *card\_type*

Contains one of the following values:

#define	Description
ACU_PROSODY_S_CARD	A Prosody S virtual card
ACU_PROSODY_X_CARD	A Prosody X card

##### *resources\_available*

Contains a bit mask consisting of one or more of the following values:

#define	Meaning
ACU_RESOURCE_CALL	The card has call resources
ACU_RESOURCE_SWITCH	The card has switch resources
ACU_RESOURCE_SPEECH	The card has Prosody resources
ACU_RESOURCE_IP_TELEPHONY	The card has IP telephony resources
ACU_RESOURCE_ODPR	The card has ODPR resources
ACU_RESOURCE_TRM	The card supports the TiNG resource manager

Using this field an application can determine which APIs can be used with a card.

***hw\_version***

Contains a string describing the card version. This is for information purposes only.

***app\_context\_token***

Will contain the application defined context associated with the card.

***hw\_position***

Is reserved for future expansion.

***card\_name***

Contains the user-defined name for the card as configured using the Aculab Configuration Tool or the `acu_set_card_name` function.

***card\_desc***

Contains a description of the card (for example, "Aculab ProsodyX PCI card"). This is for information purposes only.

***ip\_address***

The IP address of the card.

***card\_key***

The access control key for use in accessing the card. This is for internal and diagnostic purposes.

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## Example Usage

```
ACU_CARD_INFO_PARMS card_info;
ACU_CARD_ID card_id; /* id of a previously opened card */
ACU_INT error = 0;

INIT_ACU_STRUCT(&card_info);
card_info.card_id = card_id;

error = acu_get_card_info(&card_info);

if (error == 0)
{
    /* display information about each card */
    printf("Card %s, has the following resources available:\n",
        card_info.serial_no);

    if (card_info.resources_available & ACU_RESOURCE_SWITCH)
    {
        printf("\tSWITCH\n");
    }

    if (card_info.resources_available & ACU_RESOURCE_CALL)
    {
        printf("\tCALL\n");
    }

    if (card_info.resources_available & ACU_RESOURCE_SPEECH)
    {
        printf("\tSPEECH\n\n");
    }
}
```

## Miscellaneous functions

### 3.18 `acu_set_system_notification_queue()`

This function is used to register or un-register a queue that will receive system notification events (such as hotswap events) that are sent to the application.

#### Synopsis

```
ACU_ERR acu_set_system_notification_queue(ACU_QUEUE_PARMS* queue_parms);
```

```
typedef struct tACU_QUEUE_PARMS
{
    ACU_ULONG          size;                /* IN */
    ACU_RESOURCE_ID   resource_id;         /* IN */
    ACU_EVENT_QUEUE   queue_id;           /* IN */
} ACU_QUEUE_PARMS;
```

The function `acu_set_system_notification_queue()` takes a pointer `queue_parms`, to a structure `ACU_QUEUE_PARMS`. The structure must be initialised before invoking the function.

#### Input parameters

##### *size*

This is the size of the input structure.

##### *resource\_id*

This field is not used for this API call.

##### *queue\_id*

To register a queue to receive system notification events, set this field to the id of the queue that is to be used for system notifications. This id must have previously been allocated using `acu_allocate_event_queue()`.

To unregister any event queue that has been registered to receive system notifications, set this field to 0.

#### Return values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.19 acu\_get\_system\_notification()

The API layer queues a number of global events, including hot swap events. This function is used to retrieve the next event in the queue.

Events returned by this function are not associated with any particular card but with the system as a whole.

#### NOTE

**Global system events can be associated with an event queue using the `acu_set_system_notification_queue()` function.**

#### Synopsis

```
ACU_ERR acu_get_system_notification(ACU_SYSTEM_NOTIFICATION_PARMS* notifyp);

typedef struct _ACU_CARD_INSERT_NOTIFICATION
{
    ACU_CHAR          serial_no[ACU_MAX_SERIAL];
} ACU_CARD_INSERT_NOTIFICATION;

typedef struct _ACU_CARD_REMOVE_NOTIFICATION
{
    ACU_CHAR          serial_no[ACU_MAX_SERIAL];
} ACU_CARD_REMOVE_NOTIFICATION;

typedef struct tACU_PROSODY_IP_CARD_CARD_NOTIFICATION
{
    ACU_CHAR          serial_no[MAX_RESOURCE_SERIAL];
    ACU_CHAR          ip_address[ACU_MAX_IP_ADDRESS];
} ACU_PACK_DIRECTIVE ACU_PROSODY_IP_CARD_NOTIFICATION;

typedef struct tACU_PROSODY_IP_CARD_IP_CONFLICT_NOTIFICATION
{
    ACU_CHAR serial_no[MAX_RESOURCE_SERIAL];
    ACU_CHAR mac_address[ACU_MAX_IP_ADDRESS];
} ACU_PACK_DIRECTIVE ACU_PROSODY_IP_CARD_IP_CONFLICT_NOTIFICATION;

typedef struct _ACU_SYSTEM_NOTIFICATION_PARMS
{
    ACU_ULONG          size;                /* IN */
    ACU_INT            event;              /* OUT */
    union
    {
        ACU_CARD_INSERT_NOTIFICATION    insert_data;
        ACU_CARD_REMOVE_NOTIFICATION    remove_data;
        ACU_PROSODY_IP_CARD_NOTIFICATION    prosody_ip_data;
        ACU_PROSODY_IP_CARD_IP_CONFLICT_NOTIFICATION    prosody_ip_conflict_data;
    } data;
} ACU_SYSTEM_NOTIFICATION_PARMS;
```

The function `acu_get_system_notification()` takes a pointer `notifyp`, to a structure `ACU_SYSTEM_NOTIFICATION_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

***size***

Must be initialised as with any other Aculab API call.

## Return Values

### *event*

Returns one of the following values:

#define	Description
ACU_SYS_EVT_NO_EVENT	No event in queue
ACU_SYS_EVT_CARD_REMOVED	A card has been removed from the system. The <i>remove_data</i> member of the data union contains the <code>serial</code> number of the removed card.
ACU_SYS_EVT_CARD_ADDED	A card has been added to the system. The <i>insert_data</i> member of the data union contains the <code>serial</code> number of the inserted card.
ACU_SYS_EVT_RESMGR_RUNNING	The Resource Manager has started up.
ACU_SYS_EVT_RESMGR_STOPPED	The Resource Manager has been shut down
ACU_SYS_EVT_PROSODY_IP_CARD_REGISTERED	A card has been registered with the system. The <i>prosody_ip_data</i> member of the union contains details of the card.
ACU_SYS_EVT_PROSODY_IP_CARD_UNREGISTERED	A card registration has been removed from the system. The <i>prosody_ip_data</i> member of the union contains details of the card.
ACU_SYS_EVT_PROSODY_IP_CARD_IP_CONFLICT	A card has detected an IP conflict with another device. The <i>prosody_ip_conflict_data</i> member of the union contains the MAC address of the offending device.

### *data*

Contains any data associated with the returned *event*.

## NOTE

**Aculab recommends that newly inserted cards are thoroughly tested before being used by a running application.**

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## Example Usage

See the example for `acu_get_system_notification_wait_object()`.

### 3.20 acu\_get\_system\_notification\_wait\_object()

This function is used to get a wait object associated with the application's global system event queue. The wait object returned by this function can be used with operating system specific wait functions such as `WaitForMultipleObjects()` or `poll()`.

#### Synopsis

```
ACU_ERR acu_get_system_notification_wait_object(ACU_GET_SYS_WAIT_OBJECT_PARMS*
                                               objectp);

typedef struct _ACU_GET_SYS_WAIT_OBJECT_PARMS
{
    ACU_ULONG          size;           /* IN */
    ACU_CARD_ID        card_id        /* NOT USED */
    ACU_WAIT_OBJECT    wait_object;   /* OUT */
} ACU_GET_SYS_WAIT_OBJECT_PARMS;
```

The function `acu_get_system_notification_wait_object()` takes a pointer `objectp`, to a structure `ACU_GET_SYS_WAIT_OBJECT_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *size*

Must be initialised as with any other Aculab API call.

##### *card\_id*

This field is reserved for future use and must be set to 0, for example, using `INIT_ACU_STRUCT`.

#### Return Values

##### *wait\_object*

Will be set to a valid operating system specific event associated with the system event queue.

#### NOTE

**The wait object will remain signalled while there are system notification events queued.**

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## Example Usage

```

ACU_GET_SYS_WAIT_OBJECT_PARMS wo_parms;
ACU_SYSTEM_NOTIFICATION_PARMS event_parms;
ACU_ERR error = 0;

INIT_ACU_STRUCT(&wo_parms);

error = acu_get_system_notification_wait_object(&wo_parms);

if (error != 0)
{
    printf("Failed getting wait object with error %d\n", error);
    exit(-1);
}

/* Wait for the event */
error = WaitForSingleObject(wo_parms.wait_object, INFINITE);

if (error == WAIT_OBJECT_0)
{
    INIT_ACU_STRUCT(&event_parms);

    error = acu_get_system_notification(&event_parms);

    if (error == 0)
    {
        /* handle event */
        switch (event_parms.event)
        {
            case ACU_SYS_EVT_NO_EVENT:
                printf("No event\n");
                break;
            case ACU_SYS_EVT_CARD_ADDED:
                printf("Card %s added\n",
                    event_parms.data.insert_data.serial_no);
                break;
            case ACU_SYS_EVT_CARD_REMOVED:
                printf("Card %s removed\n",
                    event_parms.data.remove_data.serial_no);
                break;
        }
    }
}

```

### 3.21 acu\_set\_card\_notification\_queue()

This function is used to associate a card with an event queue. All notification events for this card will be signalled through this queue.

#### Synopsis

```
ACU_ERR acu_set_card_nofication_queue(ACU_QUEUE_PARMS& queue_parms);
```

```

typedef struct tACU_QUEUE_PARMS
{
    ACU_ULONG          size;          /* IN */
    ACU_EVENT_QUEUE   queue_id;     /* IN */
    ACU_RESOURCE_ID   resource_id;  /* IN */
} ACU_QUEUE_PARMS;

```

The function `acu_set_card_notification_queue()` takes a pointer `queue_parms`, to a structure `ACU_QUEUE_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *Size*

This is the size of the structure, initialised with `INIT_ACU_STRUCT()`.

##### *queue\_id*

This is the id of the queue to be used for card events.

##### *resource\_id*

This is the ID for the card, as returned from an earlier call to `acu_open_card()`.

## Return values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## 3.22 acu\_get\_card\_notification()

This function is used to retrieve an event that is associated with a particular card.

### Synopsis

```
ACU_ERR acu_get_card_notification(ACU_CARD_NOTIFICATION_PARMS notifyp);
```

```
typedef struct _ACU_CARD_NOTIFICATION_PARMS
{
    ACU_ULONG      size;           /* IN */
    ACU_CARD_ID   card_id;       /* IN */
    ACU_UINT      event;        /* OUT */
} ACU_CARD_NOTIFICATION_PARMS;
```

The function `acu_get_card_notification()` takes a pointer `notifyp`, to a structure `ACU_CARD_NOTIFICATION_PARMS`. The structure must be initialised before invoking the function.

### Input Parameters

*size*

Must be initialised as with any other Aculab API call.

*card id*

Must be set to a valid card id (returned from an earlier call to `acu_open_card()`).

### Return Values

*event*

If this function completes successfully, the *event* field will be set to one of the following values:

#define	Description
ACU_CARD_EVT_NO_EVENT	There is no event pending
ACU_CARD_EVT_CARD_REMOVE_PENDING	Ejection is pending for this card
ACU_CARD_EVT_CARD_SURPRISE_REMOVE	This card has been removed without having been correctly shut down

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### Example Usage

See the example under `acu_get_card_notification_wait_object()`.

### 3.23 acu\_get\_card\_notification\_wait\_object()

This function returns a wait object that can be used to wait for card-specific notification events. The wait object returned by this function can be used with operating system specific wait functions such as `WaitForMultipleObjects()` or `poll()`.

#### Synopsis

```
ACU_ERR acu_get_card_notification_wait_object(ACU_GET_CARD_WAIT_OBJECT_PARMS  
                                             waitp);
```

```
typedef struct _ACU_GET_CARD_WAIT_OBJECT_PARMS  
{  
    ACU_ULONG          size;          /* IN */  
    ACU_CARD_ID        card_id;       /* IN */  
    ACU_WAIT_OBJECT    wait_object;   /* OUT */  
} ACU_GET_CARD_WAIT_OBJECT_PARMS;
```

The function `acu_get_card_notification_wait_object()` takes a pointer `waitp`, to a structure `ACU_GET_CARD_WAIT_OBJECT_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

*size*

Must be initialised as with any other Aculab API call.

*card\_id*

Must be set to a valid card id (returned from an earlier call to `acu_open_card()`).

#### Return Values

*wait\_object*

Will be set to a valid operating system specific wait object associated with the selected event queue.

#### NOTE

**The wait object will remain signalled while there are notification events queued.**

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## Example Usage

```

ACU_GET_CARD_WAIT_OBJECT_PARMS wo_parms;
ACU_CARD_NOTIFICATION_PARMS event_parms;
ACU_ERR error;

INIT_ACU_STRUCT(&wo_parms);

wo_parms.card_id = 0;
error = acu_get_card_notification_wait_object(&wo_parms);

if (error != 0)
{
    printf("Failed getting wait object with error %d\n", error);
    exit(-1);
}

/* Wait for the event */
error = WaitForSingleObject(wo_parms.wait_object, INFINITE)
{
    if (error == WAIT_OBJECT_0)
    {
        INIT_ACU_STRUCT(&event_parms);

        error = acu_get_card_notification(&event_parms);

        if (error == 0)
        {
            /* handle event */
            switch (event_parms.event)
            {
                case ACU_CARD_EVT_NO_EVENT:
                    printf("No event\n");
                    break;
                case ACU_CARD_EVT_CARD_REMOVE_PENDING:
                    printf("Card 0x%X remove pending\n", event_parms.card_id);

                    /* the application should gracefully close all resources */
                    /* allocated on that card at this point */

                    break;
                case ACU_SYS_EVT_CARD_SURPRISE_REMOVE:
                    printf("Card 0x%X surprise remove\n", event_parms.card_id);

                    /* the application should close all resources */
                    /* allocated on that card at this point */

                    break;
            }
        }
    }
}

```

### 3.24 acu\_allocate\_event\_queue()

This function is used to create a new event queue for receiving events. A queue created in this way can receive events from call *handles*, call notifications, and system notifications. Creating multiple queues allows an application to arbitrarily divide up events to suit the chosen application model.

Resources are associated with an event queue using a variety of functions. Calls can be associated with a queue at creation (with the *queue\_id* fields in the parameters of `call_openout()` or `call_openin()`). Similarly, card notification events can be associated with a queue using the *notification\_queue* parameter of `acu_open_card()`.

An event queue provides a way to wait for multiple types of event from multiple resources in a single function. It does not directly replace `call_event` or any of the new V6 functions such as `acu_get_card_notification()`. Instead, it provides an indication that there is an event to be read using one of those other functions. Each event that gets queued in an event queue has a type (which lets you determine which function to call to get the actual event) and a context (which lets you determine which resource the event pertains to). The context is the user defined application context token assigned to the resource that generated the event. If no application context token has been assigned, the default context is the resource ID for that resource (i.e. the call handle for a call event, the card id for a card notification event).

#### NOTE

**There is no need to create an event queue per call handle as each handle already contains a queue of its own events.**

#### Synopsis

```
ACU_ERR acu_allocate_event_queue(ACU_ALLOC_EVENT_QUEUE_PARMS* alloc_parms);
```

```
typedef struct _ACU_ALLOC_EVENT_QUEUE_PARMS
{
    ACU_ULONG          size;          /* IN */
    ACU_EVENT_QUEUE   queue_id;      /* OUT */
} ACU_ALLOC_EVENT_QUEUE_PARMS;
```

The function `acu_allocate_event_queue()` takes a pointer *alloc\_parms*, to a structure `ACU_ALLOC_EVENT_QUEUE_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

*size*

Must be set to the size of the input structure.

#### Return Values

*queue\_id*

Will contain the id of the newly created queue. This queue must be freed using `acu_free_event_queue()` when the application has finished with the queue.

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.25 acu\_free\_event\_queue()

This function is used to release an event queue previously allocated using `acu_allocate_event_queue()`.

#### NOTE

**Every resource that was associated with the event queue to be released must have been closed or associated with a new event queue before this function is used.**

#### Synopsis

```
ACU_ERR acu_free_event_queue(ACU_FREE_EVENT_QUEUE_PARMS* free_parms);
```

```
typedef struct _ACU_FREE_EVENT_QUEUE_PARMS
{
    ACU_ULONG          size;          /* IN */
    ACU_EVENT_QUEUE    queue_id;     /* IN */
} ACU_FREE_EVENT_QUEUE_PARMS;
```

The function `acu_free_event_queue()` takes a pointer `free_parms`, to a structure `ACU_FREE_EVENT_QUEUE_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

*size*

Must be set to the size of the input structure.

*queue\_id*

The identity of the queue to be freed. This must be a valid ID as returned by a call to `acu_alloc_event_queue()`.

#### Return Values

On successful completion, a value of zero is returned; otherwise, a negative error code is returned indicating what went wrong.

### 3.26 acu\_get\_event\_from\_queue()

This function is used to retrieve an event from an event queue. The events that are returned by this function do not tell you exactly what happened, instead they tell you that something *has* happened and where to go to find out exactly what it was.

#### Synopsis

```
ACU_ERR acu_get_event_from_queue(ACU_EVENT_QUEUE_PARMS* eventp);
```

```
typedef struct _ACU_EVENT_QUEUE_PARMS
{
    ACU_ULONG          size;          /* IN */
    ACU_EVENT_QUEUE   queue_id;      /* IN */
    ACU_LONG          timeout;       /* IN */
    ACU_UINT          event_type;    /* OUT */
    ACU_ACT           context;       /* OUT */
    ACU_RESOURCE_ID   resource_id;   /* OUT */
} ACU_EVENT_QUEUE_PARMS;
```

The function `acu_get_event_from_queue()` takes a pointer `eventp`, to a structure `ACU_EVENT_QUEUE_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *size*

Must be initialised as with any other Aculab API call.

##### *queue\_id*

Must be set to the id of a queue created using `acu_alloc_event_queue()`.

##### *timeout*

The `timeout` field can be set to a maximum length of time in milliseconds to wait for an event to occur. To wait indefinitely for an event, use a timeout value of `ACU_INFINITE_WAIT`.

#### Return Values

##### *event\_type*

On successful completion, the `event_type` field will contain a value from the following table.

These values tell you which function to call to retrieve the actual event.

Event type	Call this function
ACU_RES_SYSTEM_NOTIFICATION	<code>acu_get_system_notification()</code>
ACU_RES_CARD_NOTIFICATION	<code>acu_get_card_notification()</code>
ACU_SWITCH_NOTIFICATION	<code>sw_get_card_notification()</code>
ACU_CALL_EVENT	<code>call_event()</code>
ACU_CALL_PORT_NOTIFICATION	<code>call_get_port_notification()</code>
ACU_CALL_GLOBAL_NOTIFICATION	<code>call_get_global_notification()</code>
ACU_IPT_CARD_NOTIFICATION	<code>ipt_get_card_notification()</code>
ACU_MG_CONTEXT_NOTIFICATION	<code>mg_context_get_notification()</code>
ACU_MG_TERMINATION_NOTIFICATION	<code>mg_termination_get_notification()</code>
ACU_RSM_NOTIFICATION	<code>rsm_get_notification()</code>

Or zero if no event.

**context**

The *context* field contains the application context token previously associated with the resource. If there is no application context token is associated with the resource, then the resource id (e.g. call handle) is returned as the context.

**NOTE**

**Changes to the application context token for a resource are not made retrospectively to events in the queue.**

**resource\_id**

The API-specific resource identifier associated with the event (for example, this will be a call handle for `ACU_CALL_EVENT`).

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

**3.27 acu\_get\_event\_queue\_wait\_object()**

This function is used to retrieve a platform-specific wait object associated with an event queue. This wait object can be used in platform-specific wait functions such as `poll()` Or `WaitForMultipleObjects()`.

**Synopsis**

```
ACU_ERR acu_get_event_queue_wait_object(ACU_QUEUE_WAIT_OBJECT_PARMS*
                                        wo_parms);
```

```
typedef struct _ACU_QUEUE_WAIT_OBJECT_PARMS
{
    ACU_ULONG          size;          /* IN */
    ACU_EVENT_QUEUE   queue_id;      /* IN */
    ACU_WAIT_OBJECT   wait_object;   /* OUT */
} ACU_QUEUE_WAIT_OBJECT_PARMS;
```

The function `acu_get_event_queue_wait_object()` takes a pointer *wo\_parms*, to a structure `ACU_QUEUE_WAIT_OBJECT_PARMS`. The structure must be initialised before invoking the function.

**Input Parameters****size**

Must be initialised as with any other Aculab API call.

**queue\_id**

Must be initialised to a valid queue id returned from an earlier call to `acu_allocate_event_queue()`.

**Return Values****wait\_object**

Will contain a platform-specific wait object suitable for use with platform-specific functions such as `poll()` Or `WaitForMultipleObjects()`.

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.28 acu\_get\_card\_app\_context\_token()

This function is used to retrieve application defined data that has been associated with a card. This data could be a pointer to a structure describing the card. The data associated with the card using this function can be set using `acu_open_card()` or `acu_set_card_app_context_token()`. The context token is also returned by `acu_get_card_info()` and will be returned by `acu_get_event_from_queue()`.

#### Synopsis

```
ACU_ERR acu_get_card_app_context_token(ACU_APP_CONTEXT_TOKEN_PARMS*
                                     contextp);

typedef struct _ACU_APP_CONTEXT_TOKEN_PARMS
{
    ACU_ULONG          size;                /* IN */
    ACU_RESOURCE_ID   resource_id;         /* IN */
    ACU_ACT            app_context_token;   /* OUT */
} ACU_APP_CONTEXT_TOKEN_PARMS;
```

The function `acu_get_card_app_context_token()` takes a pointer `contextp`, to a structure `ACU_APP_CONTEXT_TOKEN_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *size*

Must be initialised as with any other Aculab API call.

##### *resource\_id*

Must be initialised to a valid card id as returned from an earlier call to `acu_open_card()`.

#### Return Values

##### *app\_context\_token*

On successful completion, the `app_context_token` field will be set to the application-defined data set earlier.

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

#### Example

```
typedef struct tCARD_STRUCT
{
    /* card information */
} CARD_STRUCT;

ACU_APP_CONTEXT_TOKEN_PARMS token_params;
ACU_CARD_ID card_id; /* id of previously opened card */
CARD_STRUCT* card_data = NULL;
ACU_ERR result = 0;

INIT_ACU_STRUCT(&token_params);

token_params.resource_id = card_id;

result = acu_get_card_app_context_token(&token_params);

if (result == 0)
{
    card_data = (CARD_STRUCT*) token_params.token;
}
```

### 3.29 acu\_set\_card\_app\_context\_token()

This function is used to associate application-defined data with a card. This data could be a pointer to a structure describing the card. The data associated with the card using this function can be retrieved using `acu_get_card_info()` or `acu_get_card_app_context_token()`. The context token is also used when the card is associated with an event queue.

#### Synopsis

```
ACU_ERR acu_set_card_app_context_token(ACU_APP_CONTEXT_TOKEN_PARMS*
                                     contextp);
```

```
typedef struct _ACU_APP_CONTEXT_TOKEN_PARMS
{
    ACU_ULONG          size;                /* IN */
    ACU_RESOURCE_ID   resource_id;         /* IN */
    ACU_ACT            app_context_token;  /* IN */
} ACU_APP_CONTEXT_TOKEN_PARMS;
```

The function `acu_set_card_app_context_token()` takes a pointer `contextp`, to a structure `ACU_APP_CONTEXT_TOKEN_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *size*

Must be initialised as with any other Aculab API call.

##### *resource\_id*

Must be initialised to a valid card id as returned from an earlier call to

`acu_open_card()`.

##### *app\_context\_token*

Must be initialised to the value to be associated with the card.

#### Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

#### Example Usage

```
typedef struct tCARD_STRUCT
{
    /* card information */
} CARD_STRUCT;

ACU_APP_CONTEXT_TOKEN_PARMS token_params;
ACU_CARD_ID card_id; /* id of previously opened card */
CARD_STRUCT* card_data = malloc(sizeof(CARD_STRUCT));
ACU_ERR result = 0;

...
/* set up card data */
...

INIT_ACU_STRUCT(&token_params);

token_params.resource_id = card_id;
token_params.app_context_token = (ACU_ACT) card_data;

result = acu_set_card_app_context_token(&token_params);
```

### 3.30 acu\_get\_res\_api\_version()

This function returns the version of the Resource API layer currently in use

#### Synopsis

```
ACU_ERR acu_get_res_api_version(ACU_RESOURCE_VERSION_PARMS* version_parms);
```

```
typedef struct tACU_RESOURCE_VERSION_PARMS
{
    ACU_UINT      size;                /* IN */
    ACU_UINT      major;              /* OUT */
    ACU_UINT      minor;             /* OUT */
    ACU_UINT      rev;               /* OUT */
    ACU_CHAR      desc[ACU_MAX_RES_VERSION]; /* OUT */
} ACU_RESOURCE_VERSION_PARMS;
```

The function `acu_get_res_api_version()` takes a pointer `version_parms`, to a structure `ACU_RESOURCE_VERSION_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

**size**

Must be initialised to the size of the structure.

#### Return Values

**major**

The major component of the Resource API version

**minor:**

The minor component of the Resource API version

**rev:**

The revision component of the Resource API version

**desc:**

An ASCII string containing the Resource API version plus any additional version information (such as beta status). e.g. "v6.4.0B21".

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.31 acu\_get\_res\_mgr\_version()

This function returns the version of the Resource Manager currently in use.

#### Synopsis

```
ACU_ERR acu_get_res_mgr_version(ACU_RESOURCE_VERSION_PARMS* version_params);
```

```
typedef struct tACU_RESOURCE_VERSION_PARMS
{
    ACU_UINT      size;                /* IN */
    ACU_UINT      major;              /* OUT */
    ACU_UINT      minor;              /* OUT */
    ACU_UINT      rev;                /* OUT */
    ACU_CHAR      desc[ACU_MAX_RES_VERSION]; /* OUT */
} ACU_RESOURCE_VERSION_PARMS;
```

The function `acu_get_res_mgr_version()` takes a pointer `version_params`, to a structure `ACU_RESOURCE_VERSION_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

*size*

Must be initialised to the size of the structure.

#### Return Values

*major*

The major component of the Resource Manager version

*minor:*

The minor component of the Resource Manager version

*rev:*

The revision component of the Resource Manager version

*desc:*

An ASCII string containing the Resource Manager version plus any additional version information (such as beta status). e.g. "v6.4.0B21".

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.32 acu\_get\_card\_configuration\_state()

When the Resource Manager detects a card for the first time after a system boot it will automatically apply any configuration that was specified either using the ACT or by using some other method to create a configuration file for that card. The configuration is applied using the config tool. This function provides a way to determine whether the config tool successfully applied the configuration or not.

#### NOTE

**This function only returns the result of Resource Manager-triggered config invocations.**

#### Synopsis

```
ACU_ERR acu_get_card_configuration_state(ACU_CARD_CONFIGURATION_STATE_PARMS*
state_parms);
```

```
typedef struct tACU_CARD_CONFIGURATION_STATE_PARMS
{
    ACU_ULONG          size;                /* IN */
    ACU_CARD_ID        card_id;            /* IN */
    ACU_UINT           state;              /* OUT */
} ACU_CARD_CONFIGURATION_STATE_PARMS;
```

The function `acu_get_card_configuration_state()` takes a pointer `state_parms`, to a structure `ACU_CARD_CONFIGURATION_STATE_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *size*

Must be initialised to the size of the structure.

##### *card\_id*

Must be set to a valid card id returned from the `acu_open_card()`.

#### Return Values

##### *state*

Contains the result from config, which will be one of:

Parameter value	Description
ACU_CONFIG_SUCCEEDED	Configuration was successfully applied.
ACU_CONFIG_INVALID_SERIAL	Configuration tool couldn't find card with the specified serial number.
ACU_CONFIG_SWITCH_RESTART_FAILED	Switch driver needs restarting, restart failed.
ACU_CONFIG_SWITCH_CONFIG_FAILED	Configuration tool failed applying switch configuration.
ACU_CONFIG_PORT_CONFIG_FAILED	Configuration tool failed applying port configuration.
ACU_CONFIG_PROSODY_CONFIG_FAILED	Prosody configuration failed to be applied.
ACU_CONFIG_GENERAL_CONFIG_FAILED	Configuration tool failed applying general configuration.
ACU_CONFIG_IP_CONFIG_FAILED	Configuration tool failed applying IP configuration.

Parameter value	Description
ACU_CONFIG_INVALID_COMMAND_LINE	Invalid command line option supplied.
ACU_CONFIG_ERROR	Configuration tool encountered an error.
ACU_CONFIG_RESTART_REQUIRED	Card restart required.
ACU_CONFIG_NO_CFG_FILE	No configuration file for card.
ACU_CONFIG_APPLYING_CONFIGURATION	Configuration currently being applied.
ACU_CONFIG_TIMED_OUT	Configuration tool took too long and was terminated.
ACU_CONFIG_FAILED_TO_RUN	Configuration tool failed to run.
ACU_CONFIG_TERMINATED_UNEXPECTEDLY	Configuration tool terminated unexpectedly.

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.33 `acu_get_card_state_desc()`

This function is used to convert the state returned by `acu_get_card_configuration_state()` into an ASCII string.

#### NOTE

**This function is provided for diagnostic purposes only. Applications should not rely on the content of the string returned.**

#### Synopsis

```
ACU_CHAR* acu_get_card_state_desc(ACU_UINT card_state);
```

#### Input Parameters

*card\_state*

The state returned by `acu_get_card_configuration_state()`

#### Return Values

The function returns a 0-terminated string containing an ASCII representation of the state.

### 3.34 `acu_error_desc()`

This function returns a string describing an Aculab error code and may be used to obtain error information for the following Aculab APIs:

Resource API

Call API

Switch API

Prosody API

#### Synopsis

```
ACU_CHAR * acu_error_desc(ACU_ERR error);
```

## Description

This resource API function allows applications to get error descriptions for Aculab's entire core APIs from a single function. There is no equivalent function in the switch API or in the Prosody API.

For the call, switch, and resource API, output strings are in the format:

```
ERR_xxxx (nnn): dddddd
```

Where;

xxxx is the name of the error

nnn is the error code

dddddd is the description.

For example:

```
ERR_COMMAND (-3) Command not valid in current state
```

For the Prosody API, output strings are in the format :

```
ERR_xxxx (nnn)
```

When you put in an error code that `acu_error_desc()` does not know about, you will get `ERR_UNKNOWN` - Unknown error value.

## Input Parameters

*error*

The error code returned from an Aculab function

## Return values

This function returns a pointer to a 0-terminated ASCII string describing the error.

See Appendix A: for a list of possible error messages.

### 3.35 `acu_get_aculab_directory()`

This function is used to return the current value of `ACULAB_ROOT`.

## Synopsis

```
ACU_ERR acu_get_aculab_directory(ACU_UINT size, char* buffer);
```

## Input parameters

*Size*

This is the size of the buffer provided.

*Buffer*

This is a caller-allocated buffer of size bytes.

## Returns

On successful completion buffer will contain a 0-terminated ASCII string.

## Prosody IP card management functions

### 3.36 acu\_register\_prosody\_ip\_card()

This function is used to register a remote Prosody X card with the system. This allows the system to configure and use Prosody X cards located in another chassis.

#### Synopsis

```
ACU_ERR acu_register_prosody_ip_card(ACU_REGISTER_PROSODY_IP_CARD_PARMS*
                                     registerp);

typedef struct tACU_REGISTER_PROSODY_IP_CARD_PARMS
{
    ACU_ULONG        size;                /* IN */
    ACU_CHAR         serial_no[MAX_RESOURCE_SERIAL]; /* IN */
    ACU_INT          persistent;          /* IN */
    ACU_INT          configure;          /* IN */
    ACU_CHAR         ip4_address[ACU_MAX_IP_ADDRESS]; /* IN */
    ACU_CHAR         ip4_netmask[ACU_MAX_IP_ADDRESS]; /* IN */
    ACU_CHAR         ip4_gateway[ACU_MAX_IP_ADDRESS]; /* IN */
    ACU_INT          use_dhcp;           /* IN */
    ACU_CHAR         key[ACU_MAX_CARD_KEY]; /* IN */
    ACU_INT          watchdog_timeout;   /* IN */
} ACU_REGISTER_PROSODY_IP_CARD_PARMS;
```

The function `acu_register_prosody_ip_card()` takes a pointer `registerp`, to a structure `ACU_REGISTER_PROSODY_IP_CARD_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *size*

Must be initialised as with any other Aculab API call.

##### *serial\_no*

The serial number of the card to be registered. If the card is being configured with a fixed network address then this may be blank, otherwise it must be initialised.

##### *persistent*

If this parameter set to a non-zero value then the card registration will persist over system restarts. Otherwise the card registration will not be recorded.

##### *configure*

If this parameter is set to a non-zero value then the system will configure the card if it is not booted. Otherwise the system will rely on another system performing initial configuration of the card. If configuration is enabled then either `use_dhcp` must be enabled or an IPv4 address and netmask must be supplied.

If this parameter is set to zero then all of the fields below it in the structure will be ignored.

##### *ip4\_address*

The host name or IPv4 address of the card. This can not be specified if `use_dhcp` is specified.

##### *ip4\_netmask*

The IPv4 network mask to use with the card. This can not be specified if `use_dhcp` is specified.

##### *ip4\_gateway*

The default router for the card, specified as either a host name or IPv4 address. This can not be specified if `use_dhcp` is specified.

***use\_dhcp***

If this parameter is set to a non-zero value then the card will acquire its network configuration dynamically. If this is specified then none of the IPv4 configuration options may be used.

***key***

The access control key which will be used to restrict access to the card, specified as a null terminated string containing any alpha-numeric character and most punctuation characters except ':' or '>'.

***watchdog timeout***

This parameter specifies the timeout, which will be used with the card watchdog if this system configures the card. If set to zero, then the card watchdog will be disabled. Otherwise it specifies the watchdog timeout in seconds. If no host system is controlling the cards for the watchdog time then the card will reboot. Timeouts of less than 60 seconds may not be specified.

**Return Values**

On successful completion, a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

**3.37 acu\_configure\_prosody\_ip\_card()**

This function is used to change the configuration of a registered Prosody IP card.

**Synopsis**

```
ACU_ERR acu_configure_prosody_ip_card(ACU_PROSODY_IP_CARD_REGISTRATION_PARMS*
                                     registerp);
```

```
typedef struct tACU_PROSODY_IP_CARD_REGISTRATION_PARMS
{
    ACU_ULONG        size;                               /* IN */
    ACU_CHAR         card[ACU_MAX_RESOURCE_NAME];       /* IN */
    ACU_INT          use_remote;                         /* IN */
    ACU_INT          persistent;                        /* IN */
    ACU_INT          configure;                        /* IN */
    ACU_CHAR         ip4_address[ACU_MAX_IP_ADDRESS];  /* IN */
    ACU_CHAR         ip4_netmask[ACU_MAX_IP_ADDRESS];  /* IN */
    ACU_CHAR         ip4_gateway[ACU_MAX_IP_ADDRESS];  /* IN */
    ACU_INT          use_dhcp;                         /* IN */
    ACU_CHAR         key[ACU_MAX_CARD_KEY];            /* IN */
    ACU_INT          watchdog_timeout;                 /* IN */
    ACU_UINT         auto_generated_key;               /* UNUSED */
} ACU_PROSODY_IP_CARD_REGISTRATION_PARMS;
```

The function `acu_configure_prosody_ip_card()` takes a pointer `registerp`, to a structure `ACU_PROSODY_IP_CARD_REGISTRATION_PARMS`. The structure must be initialised before invoking the function.

**Input Parameters*****size***

Must be initialised as with any other Aculab API call.

***card***

The serial number or IPv4 address of the card to be configured.

***use\_remote***

If this parameter is set to a non-zero value then the system will use this card registration with cards not physically located in the system. If it is set to zero then this registration will be ignored if the card is not physically present in the system.

***persistent***

If this parameter set to a non-zero value then the card registration will persist over system restarts. Otherwise the card registration will not be recorded.

***configure***

If this parameter is set to a non-zero value then the system will configure the card if it is not booted. Otherwise the system will rely on another system performing initial configuration of the card. If configuration is enabled then either `use_dhcp` must be enabled or an IPv4 address and netmask must be supplied.

***ip4\_address***

The host name or IPv4 address of the card. This can not be specified if `use_dhcp` is specified.

***ip4\_netmask***

The IPv4 network mask to use with the card. This can not be specified if `use_dhcp` is specified.

***ip4\_gateway***

The default router for the card, specified as either a host name or IPv4 address. This can not be specified if `use_dhcp` is specified.

***use\_dhcp***

If this parameter is set to a non-zero value then the card will acquire its network configuration dynamically. If this is specified then none of the IPv4 configuration options may be used.

***key***

The access control key which will be used to restrict access to the card, specified as a null terminated string containing any alpha-numeric character and most punctuation characters except ':' or '>'.

***watchdog\_timeout***

This parameter specifies the timeout, which will be used with the card watchdog if this system configures the card. If set to zero, then the card watchdog will be disabled. Otherwise it specifies the watchdog timeout in seconds. If no host system is controlling the cards for the watchdog time then the card will reboot. Timeouts of less than 60 seconds may not be specified.

***auto\_generated\_key***

This parameter is unused in this function.

**Return Values**

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.38 acu\_get\_prosody\_ip\_card\_config()

This function is used to retrieve the configuration of a registered Prosody IP card.

#### Synopsis

```
ACU_ERR
acu_get_prosody_ip_card_config(ACU_PROSODY_IP_CARD_REGISTRATION_PARMS*
                               configp);

typedef struct tACU_PROSODY_IP_CARD_REGISTRATION_PARMS
{
    ACU_ULONG      size; /* IN */
    ACU_CHAR      card[ACU_MAX_RESOURCE_NAME]; /* IN */
    ACU_INT       use_remote; /* OUT */
    ACU_INT       persistent; /* OUT */
    ACU_INT       configure; /* OUT */
    ACU_CHAR      ip4_address[ACU_MAX_IP_ADDRESS]; /* OUT */
    ACU_CHAR      ip4_netmask[ACU_MAX_IP_ADDRESS]; /* OUT */
    ACU_CHAR      ip4_gateway[ACU_MAX_IP_ADDRESS]; /* OUT */
    ACU_INT       use_dhcp; /* OUT */
    ACU_CHAR      key[ACU_MAX_CARD_KEY]; /* OUT */
    ACU_INT       watchdog_timeout; /* OUT */
    ACU_UINT      auto_generated_key; /* OUT */
} ACU_PACK_DIRECTIVE ACU_PROSODY_IP_CARD_REGISTRATION_PARMS;
```

The function `acu_get_prosody_ip_card_config()` takes a pointer `configp`, to a structure `ACU_PROSODY_IP_CARD_REGISTRATION_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *size*

Must be initialised as with any other Aculab API call.

##### *card*

The serial number or IPv4 address of the card for which configuration should be retrieved.

#### Return Values

##### *use\_remote*

If this parameter is set to a non-zero value then the system will use this card registration with cards not physically located in the system. If it is set to zero then this registration will be ignored if the card is not physically present in the system.

##### *persistent*

If this parameter set to a non-zero value then the card registration will persist over system restarts. Otherwise the card registration will not be recorded.

##### *configure*

If this parameter is set to a non-zero value then the system will configure the card if it is not booted. Otherwise the system will rely on another system performing initial configuration of the card.

##### *ip4\_address*

The host name or IPv4 address of the card.

##### *ip4\_netmask*

The IPv4 network mask to use with the card.

***ip4\_gateway***

The default router for the card, specified as either a host name or IPv4 address.

***use\_dhcp***

If this parameter is set to a non-zero value then the card will acquire its network configuration dynamically.

***key***

The access control key, which will be used to restrict access to the card, specified as a null terminated string.

***watchdog\_timeout***

This parameter specifies the timeout which will be used with the card watchdog if this system configures the card. If set to zero, then the card watchdog will be disabled. Otherwise it specifies the watchdog timeout in seconds.

***auto\_generated\_key***

Older versions of the Aculab resource manager did not need a `card_key` to be specified to restrict access to the card. Newer resource managers need a `card_key` specified for every card. When upgrading the resource manager to one in which the `card_keys` are mandatory, the resource manager will generate a suitable `card_key` for each Prosody X card in the system. This field indicates that the key has been automatically generated.

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.39 `acu_unregister_prosody_ip_card()`

This function is used to remove the registration for a Prosody X card.

#### Synopsis

```
ACU_ERR acu_unregister_prosody_ip_card(ACU_UNREGISTER_PROSODY_IP_CARD_PARMS*
                                     cardp)
```

```
typedef struct _ACU_SNAPSHOT_PARMS
{
    ACU_ULONG    size;                /* IN */
    ACU_CHAR    serial_no[ACU_MAX_RESOURCE_NAME]; /* IN */
} ACU_UNREGISTER_PROSODY_IP_CARD_PARMS;
```

The function `acu_unregister_prosody_ip_card()` takes a pointer `cardp`, to a structure `ACU_UNREGISTER_PROSODY_IP_CARD_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

***size***

Must be initialised to the size of the structure.

***serial\_no***

Contains the serial number or IP address of the card to be unregistered.

#### Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.40 acu\_prosody\_ip\_get\_device\_info()

This function returns diagnostic information about the hardware and software configuration of a Prosody X or PMX device.

#### Synopsis

```
ACU_ERR acu_prosody_ip_get_device_info(ACU_PROSODY_IP_DEVICE_INFO_PARMS_
                                     PARMS* infp)

typedef struct tACU_PROSODY_IP_DEVICE_INFO_PARMS
{
    ACU_ULONG          size;                /* IN */
    ACU_CARD_ID        card_id;            /* IN */
    ACU_INT            device;             /* IN */
    ACU_CHAR           model[ACU_MAX_HWVER]; /* OUT */
    ACU_CHAR           serial_no[MAX_RESOURCE_SERIAL]; /* OUT */
    ACU_CHAR           hw_version[ACU_MAX_HWVER]; /* OUT */
    ACU_CHAR           description[ACU_MAX_CARD_DESC]; /* OUT */
    ACU_CHAR           bootloader_version[ACU_MAX_FWVER]; /* OUT */
    ACU_CHAR           firmware_version[ACU_MAX_FWVER]; /* OUT */
    ACU_ULONG          temp0;              /* OUT */
    ACU_ULONG          temp1;              /* OUT */
    ACU_ULONG          voltage_5_0;        /* OUT */
    ACU_ULONG          voltage_3_3;        /* OUT */
    ACU_SHORT          control_port;       /* OUT */
} ACU_PACK_DIRECTIVE ACU_PROSODY_IP_DEVICE_INFO_PARMS;
```

The function `acu_get_prosody_ip_device_info()` takes a pointer `infp`, to a structure `ACU_PROSODY_IP_DEVICE_INFO_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *size*

Must be initialised to the size of the structure.

##### *card\_id*

Must be set to a valid card id returned from the `acu_open_card()` for which `acu_open_mg()` has been called.

##### *device*

This must be set to the index of the device for which information is to be retrieved.

For example, for Prosody X PCI cards this is:

- 0 for the Prosody X PCI card itself.
- 1 for the optional PMX (if fitted).

#### Return Values

##### *model*

This contains the Aculab model number of the device.

##### *serial\_no*

This contains the serial number of the device.

##### *hw\_version*

This contains the hardware revision of the device.

##### *description*

This contains a string describing the device (for example "PMX" or "Prosody X").

##### *bootloader\_version*

This contains a string identifying the version of the boot loader used by the device.

***firmware\_version***

This contains a string identifying the version of the firmware running on the device.

***temp0***

This contains the temperature reported by the first temperature monitoring system on the device in units of milli-degrees Celsius (1000<sup>th</sup> of a degree Celsius).

***temp1***

This contains the temperature reported by the first temperature monitoring system on the device in milli-degrees Celsius (1000<sup>th</sup> of a degree Celsius).

***voltage\_5\_0***

This contains the voltage reported for the 5.0V power rail by the hardware monitoring system on the device in milli-Volts.

***voltage\_3\_3***

This contains the voltage reported for the 3.3V power rail by the hardware monitoring system on the device in milli-Volts.

***control\_port***

This indicates the network port to be used for control of this device. It is intended for internal use only.

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.41 `acu_prosody_ip_get_registered_cards()`

This function populates a struct with a collection of the cards currently registered with the system.

#### NOTE

**It is possible that in between obtaining the snapshot and trying to use it, that the system configuration could change. An application should be designed with this in mind.**

### Synopsis

```
ACU_ERR acu_get_registered_prosody_ip_cards(ACU_SNAPSHOT_PARMS* cardsp)
```

```
typedef struct _ACU_SNAPSHOT_PARMS
{
    ACU_ULONG    size;                /* IN */
    ACU_UINT     count;              /* OUT */
    ACU_CHAR     serial_no[MAX_SNAPSHOT_CARDS][ACU_MAX_SERIAL]; /* OUT */
} ACU_SNAPSHOT_PARMS;
```

The function `acu_prosody_ip_get_registered_cards()` takes a pointer `cardsp`, to a structure `ACU_SNAPSHOT_PARMS`. The structure must be initialised before invoking the function.

### Input Parameters

***size***

Must be initialised to the size of the structure.

## Return Values

### *count*

The number of cards in the snapshot. The structure contains an array of information about each of the cards in the system. This is the number of elements in the *serial\_no* array that are populated.

### *serial\_no*

Contains an array of the serial numbers of the cards in the snapshot.

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## 3.42 `acu_prosody_ip_card_restart()`

This function is used to restart a Prosody X class device. The device will reboot and be reconfigured.

### NOTE

**It is typically not necessary to use this function. After some configuration changes (e.g. changing the IP configuration of the card) it is necessary to restart the card for the card to pick up the new configuration.**

## Synopsis

```
ACU_ERR acu_prosody_ip_card_restart(ACU_PROSODY_IP_CARD_RESTART_PARMS*
restart_parms);
```

```
typedef struct tACU_PROSODY_IP_CARD_RESTART_PARMS
{
    ACU_ULONG        size;                /* IN */
    ACU_CHAR         card[ACU_MAX_RESOURCE_NAME]; /* IN */
    ACU_INT          method;             /* IN */
} ACU_PROSODY_IP_CARD_RESTART_PARMS;
```

The function `acu_prosody_ip_card_restart()` takes a pointer *restart\_parms*, to a structure `ACU_PROSODY_IP_CARD_RESTART_PARMS`. The structure must be initialised before invoking the function.

## Input Parameters

### *size*

Must be initialised to the size of the structure.

### *card*

The serial number or IPv4 address of the card for which configuration should be retrieved.

### *method*

The method to be use to restart the card. This may be one of:

`ACU_RESTART_METHOD_REMOTE` - Restarts the card straight away regardless of what it is doing.

`ACU_RESTART_METHOD_GRACEFUL` - When this function is called, the card will wait for all users of the card to stop using it before restarting it. The function returns straight away.

## Return Values

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.43 acu\_prosody\_ip\_card\_status()

This function is used to return the status of a Prosody X class device.

#### NOTE

The information returned by this function is for diagnostic and information purposes only. In particular, an application should not rely on specific diagnostic messages being returned at specific times.

#### NOTE

This information is updated regularly whilst the Resource Manager attempts to establish communications with a card.

### Synopsis

```
ACU_ERR acu_prosody_ip_card_status(ACU_PROSODY_IP_CARD_STATUS_PARMS*
status_parms);
```

```
typedef struct tACU_PROSODY_IP_CARD_STATUS_PARMS
{
    ACU_ULONG      size;                /* IN */
    ACU_CHAR       card[ACU_MAX_RESOURCE_NAME]; /* IN */
    ACU_CHAR       status[ACU_MAX_CARD_STATUS]; /* OUT */
    ACU_CHAR       diagnostic[ACU_MAX_CARD_DIAGNOSTIC]; /* OUT */
} ACU_PROSODY_IP_CARD_STATUS_PARMS;
```

The function `acu_prosody_ip_card_status()` takes a pointer `status_parms`, to a structure `ACU_PROSODY_IP_CARD_STATUS_PARMS`. The structure must be initialised before invoking the function.

### Input Parameters

*size*

Must be initialised to the size of the structure.

*card*

The serial number or IPv4 address of the card for which configuration should be retrieved.

### Return Values

*status*

This is a short 0-terminated string describing the status of the card

*diagnostic*

This is a longer 0-terminated string, which provides additional diagnostic information relevant to the cards state. This field may include operating system error messages.

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

### 3.44 `acu_prosody_ip_get_local_card_info()`

This function is used to return information about NIC adaptors of local Prosody X cards.

#### Synopsis

```
ACU_ERR  
acu_prosody_ip_get_local_card_info(ACU_PROSODY_IP_LOCAL_CARD_INFO_PARMS*  
local_card_info_parms);
```

```
typedef struct tACU_PROSODY_IP_LOCAL_CARD_INFO_PARMS  
{  
    ACU_ULONG    size;  
    ACU_CHAR     card[ACU_MAX_RESOURCE_NAME];  
    ACU_CHAR     local_ethernet_device[ACU_MAX_RESOURCE_NAME];  
    ACU_CHAR     local_ethernet_os_specific_info[ACU_MAX_RESOURCE_NAME];  
};
```

The function `acu_prosody_ip_get_local_card_info()` takes a pointer `local_card_info_parms`, to a structure `ACU_PROSODY_IP_LOCAL_CARD_INFO_PARMS`. The structure must be initialised before invoking the function.

#### Input Parameters

##### *size*

Must be initialised to the size of the structure.

##### *card*

The serial number of the card for which information should be retrieved.

#### Return Values

##### *local\_ethernet\_device*

This is a 0-terminated string, which contains the name of the Ethernet device as known to the OS.

##### *local\_ethernet\_os\_specific\_info*

This is a 0-terminated string, which contains information specific to each OS as follows:

Windows: Human readable device description

Linux: acumam device for the card, if one exists. Empty string otherwise.

Solaris: Empty string.

On successful completion a value of zero is returned; otherwise a negative error code is returned indicating what went wrong.

## Appendix A: Error codes

Error message	Code	Description
ERR_NO_ERROR	0	There was no error
ERR_HANDLE	-2	Illegal handle or out of handles
ERR_COMMAND	-3	Command not valid in current state
ERR_NET	-4	Illegal network number
ERR_PARM	-5	Illegal parameter
ERR_INVALID_PARM	-5	Illegal parameter DUPLICATE
ERR_RESPONSE	-6	Application failed to respond within response time
ERR_NOCALLIP	-7	No call in progress
ERR_TSBAR	-8	Timeslot barred
ERR_TSBUSY	-9	Timeslot busy
ERR_CFAIL	-10	Command failed to execute
ERR_SERVICE	-11	Invalid service code
ERR_BUFF_FAIL	-12	Out of buffer resources
ERR_DNLD_ZAP	-13	Failed wait for board bootstrap code to respond-Check firmware is appropriate for hardware.
ERR_DNLD_NOCMD	-14	Firmware not loaded - command denied
ERR_DNLD_NODNLD	-15	Firmware installed - download denied
ERR_DNLD_GEN	-16	General failure during download
ERR_DNLD_NOSIG	-17	Downloaded firmware failed to start
ERR_DNLD_NOEXEC	-18	Downloaded firmware is not executing
ERR_DNLD_NOCARD	-19	Downloaded firmware is not executing
ERR_DNLD_SYSTAT	-20	Downloaded firmware detected an error
ERR_DNLD_BADTLS	-21	Driver does not support the firmware
ERR_DNLD_POST	-22	Board failed power on self test
ERR_DNLD_SW	-23	Switch setup error
ERR_DNLD_MEM	-24	Could not allocate memory for download
ERR_DNLD_FILE	-25	Could not find the file to download
ERR_DNLD_TYPE	-26	The file is not downloadable
ERR_LIB_INCOMPAT	-27	Incompatible library used
ERR_DRV_INCOMPAT	-28	Incompatible driver used

Error message	Code	Description
ERR_DRV_CALLINIT	-29	Another process attempted to call 'call_init' while others are accessing the driver
ERR_TS_BLOCKED	-30	Timeslot blocked
ERR_NO_SYS_RES	-31	Out of OS resources
ERR_PORT_BLOCKED	-32	Port blocked
ERR_INVALID_ADDR	-40	Tried to access invalid address
ERR_INVALID_PORT	-41	Tried to access invalid port
ERR_MANAGEMENT_RPC	-42	Failed to startup Management RPC session
ERR_SESSION_RPC	-43	Failed to startup Session RPC session
ERR_NO_SERVICE	-44	The service did not respond
ERR_NO_BOARD	-45	The board did not respond
ERR_BOARD_UNLOADED	-46	The board has not been downloaded
ERR_BOARD_VERSION	-47	Board software incompatible with host software
ERR_DNLD_CRCERRORS	-48	CRC error detected in board firmware during download
ERR_MAX_APP_LIMIT	-50	Tried to start more than the supported number of applications
ERR_INVALID_FW_PARM	-51	Unrecognised firmware switch
ERR_UNSUPPORTED	-52	Unsupported operation attempted
ERR_NOT_IMPLEMENTED	-53	Function isn't implemented yet
ERR_NO_PLUGINS	-54	Call library found no plugins for communicating with drivers
ERR_DUPLICATE_PLUGINS_FOUND	-55	Call library found multiple plugins with the same id
ERR_NO_PORTS	-56	The call library was unable to find any ports on this card
ERR_DNLD_ALREADY_IN_USE	-57	Port is in use by another host
ERR_DNLD_NOT_RESPONDING	-58	The port is not responding
ERR_NO_FREE_CHANNELS	-59	There are no free channels on this port
ERR_INVALID_PLUGIN	-60	An invalid file was found in the call control plugins directory
ERR_SW_INVALID_COMMAND	-200	Command code is not supported
ERR_SW_DEVICE_ERROR	-202	An error was returned from a device driver called by this driver.
ERR_SW_NO_RESOURCES	-204	An internal driver resource has been exhausted.

Error message	Code	Description
ERR_SW_INVALID_SWITCH	-209	Out of range switch driver index
ERR_SW_INVALID_STREAM	-210	Stream number in parameter list is out of range.
ERR_SW_INVALID_TIMESLOT	-211	Time slot in parameter list is out of range.
ERR_SW_INVALID_CLOCK_PARM	-213	Invalid clock configuration parameter
ERR_SW_INVALID_MODE	-216	Incorrect SET_OUTPUT or QUERY_OUTPUT mode.
ERR_SW_INVALID_MINOR_SWITCH	-217	Minor (internal) switch error.
ERR_SW_INVALID_PARAMETER	-218	General invalid parameter error.
ERR_SW_NO_PATH	-220	Connection cannot be made due to switch limitation
ERR_SW_NO_SCBUS_CLOCK	-224	No card driving SCbus clock.
ERR_SW_OTHER_SCBUS_CLOCK	-225	Another non-MVSWDEV controlled card is driving scbus clock
ERR_SW_PATH_BLOCKED	-226	eg. Because of MVIP output MUX
ERR_SW_OS_INTERRUPTED	-227	eg. Unix event wait interrupted through signal
ERR_SW_OS_INCONSISTENT_STATE	-228	Switch device and driver inconsistent
ERR_SW_PORT_NOT_LOADED	-229	Firmware is not running on PMX port
ERR_SW_PORT_RATE_MISMATCH	-230	NETREF rate does not match line rate
ERR_SW_PMX_FPGA	-231	Request not supported by PMX FPGA
ERR_SW_COMPONENT_MISMATCH	-232	pxscs and t8110.ko components do not match on Prosody X
ERR_SW_NO_SUCH_DSP	-233	No DSP is fitted in requested position
ERR_SW_NO_SUCH_DSP_PORT	-234	Requested DSP serial port does not exist
ERR_FAIL	-501	An error occurred for which there is no better error code
ERR_NO_MEMORY	-502	Operation failed because insufficient memory was allocated
ERR_CARD_NOT_FOUND	-503	Specified card could not be found
ERR_INVALID_RESOURCE	-505	An invalid resource was specified
ERR_ALREADY_OPEN	-506	Application attempted to open a resource that's already open
ERR_SERVER_NOT_RESPONDING	-507	The server is not responding
ERR_CARD_EJECT_PENDING	-508	The operation has failed because the card is waiting to be ejected
ERR_TIMEOUT	-509	The operation timed out

Error message	Code	Description
ERR_STILL_IN_USE	-510	The resource could not be closed as it is still in use
ERR_INVALID_CARD	-511	An invalid card id was specified
ERR_INVALID_CONFIG	-512	A configuration problem occurred
ERR_BUFF_SIZE	-513	The buffer provided is not big enough
ERR_RESOURCE_RELEASED	-514	The resource has been released
ERR_ALREADY_EXISTS	-515	This resource already exists
ERR_LIBRARY_NOT_LOADED	-516	A required library is not loaded or doesn't contain the required function
ERR_ENVIRONMENT_NOT_SET	-517	A required environment variable doesn't exist
ERR_FILE_ACCESS	-518	File I/O failed
ERR_BOARD_COMM_FAILURE	-519	Communication with board failed.
ERR_FILE_FORMAT	-520	Unable to parse file
ERR_BOOTLOADER_FAILED	-521	The firmware bootloader failed
ERR_INTERRUPTED	-522	The operating system has interrupted a system function
ERR_FUNCTION_NOT_FOUND	-523	The specified function is not available
ERR_SERVICE_DEPENDENCY_FAILED	-524	A dependency of the specified service has failed
ERR_SERVICE_UNKNOWN	-525	An unknown service was specified
ERR_SERVICE_RUNNING	-526	The specified service is running
ERR_SERVICE_NOT_RUNNING	-527	The specified service is not running
ERR_SERVICE_EXISTS	-528	The specified service already exists
ERR_SERVICE_START_FAILED	-529	The specified service failed to start
ERR_SERVICE_STOP_FAILED	-530	The specified service failed to stop
ERR_FILE_NOT_FOUND	-531	The specified file could not be found
ERR_BAD_KEY	-532	The specified card key is invalid
ERR_WRONG_SUBNET	-533	The specified IP address is on the wrong subnet