aculab

ACULAB.COM

Aculab IP telephony API guide

Revision 6.7.3

MAN1782

PROPRIETARY INFORMATION

Document Revision

Rev	Date	By	Detail
6.2.0		DJL	First Draft Issue
6.2.2	07.09.04	DJL	Beta release
6.2.2	15.09.04	DJL	Full release
6.2.3	13.10.04	DJL	Correction of `struct` information
6.3.0	24.12.04	DJL	Various updates including support for new hardware
6.3.1	26.01.05	DJL	Small changes, for example, new note in `ipt_card_info`
6.3.2	02.02.05	DJL	SIP transport type values updated
6.3.4	18.04.05	DJL	Addition of missing control definition and update of revision to reflect current software release.
6.3.5	1.06.05	MAB	Minor corrections to reflect the current status of the API.
6.4.0	02.11.05	DJL	Updates for V6.4.0 release
6.4.1	18.01.06	DJL	Correction of typos and some small changes
6.4.2	15.05.06	DJL	Change of references to include Prosody X
6.4.3	05.10.06	DJL	Updates to H.323 support
6.4.4	17.11.06	DJL	Update to `ipt_set_dtmf_handling` API
6.4.5	15.10.09	AR	Removed sections 2.4 (ipt_card_configure) and 2.5 (ipt_card_download) Fixed typos and grammar throughout.
6.4.6	19.10.10	DF	Removed references to EOL products
6.4.7	27.10.10	EBJ	Updated to corporate fonts
6.5.0	31.07.13	NC	Clarify ipt_delete_alias() for SIP
6.5.1	06.10.15	NMC	AsyncRegister config option for ipt_add_alias()
6.7.0	10.03.17	NMC	Rebranding for 6.7 release
6.7.1	21.12.20	ACP	Updated for IPTel_Development 6.7.31

6.7.2	22.03.22	ACP	Clarify ipt_query_alias() for SIP
6.7.3	13.03.24	ACP	Format only

CONTENTS

Introduction

This document details the API functions required for Prosody X cards. It includes addressing, configuration and registration parameter information for both SIP and H.323 protocols.

You only need to use the API calls in this guide when you want to change media or protocol settings, add/remove SIP sockets or register with an H.323 gatekeeper or SIP server etc.

When you do use API calls in this guide, you will need to include the following header:

```
#include "iptel_lib.h"
```

On Windows the following library should be used:

```
iptel_lib.lib
```

On Linux and Solaris this library should be used:

```
libacu_ipt.so
```

NOTE

For further details of IP telephony call control functions, refer to the V6 call control API guide.

1 Card management

This section describes the API functions used to manage the ProsodyX cards.

1.1 acu_open_ipt()

Used to open an ProsodyX card when using the Aculab IP Telephony API. To use IP Telephony call control protocols with the Aculab Call API, `acu_open_call` should be used as normal.

Synopsis

```
ACU_ERR acu_open_ipt(ACU_OPEN_IPTEL_PARMS* openp);

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

Input parameters

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

card_id

This must be a `card_id` returned by `acu_open_card()`. Cards that may be opened for IP Telephony will have the resource type `ACU_RESOURCE_IP_TELEPHONY`.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

1.2 acu_close_ipt()

Used by an application to close the IP Telephony API for a given ProsodyX card.

NOTE

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

Synopsis

```
ACU_ERR acu_close_ipt(ACU_CLOSE_IPTEL_PARMS* closep);

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

Input parameters

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

card_id

This must be a `card_id` returned by `acu_open_card()` that has previously been opened for IP Telephony with `acu_open_ipt()`.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

1.3 ipt_card_info()

Provides information about the status of the ProsodyX card.

NOTE

When the card is not fully in service, some of the information may not be available. The available information will depend on the release of the software, the type of card in use and the reasons why the card is not in service.

Synopsis

```
ACU_ERR ipt_card_info(IPT_CARD_INFO_XPARMS *infop);

   typedef struct tIPT_CODEC_INFO
   {
     ACU_INT               codec_type;                          /* OUT */
     ACU_INT               channels_supported;                  /* OUT */
     ACU_INT               channels_free;                       /* OUT */
   } IPT_CODEC_INFO;

   typedef struct tIPT_ETHERNET_INFO
   {
     ACU_INT               active;                              /* OUT */
     ACU_INT               connected;                           /* OUT */
     ACU_INT               speed;                               /* OUT */
     ACU_INT               duplex;                              /* OUT */
   } IPT_ETHERNET_INFO;

   typedef struct tIPT_CARD_INFO_XPARMS
   {
     ACU_ULONG             size;
     ACU_CARD_ID           card_id;                             /* IN */
     ACU_INT               active;                              /* OUT */
     ACU_CHAR              address[MAXHOSTADDRESS];             /* OUT */
     ACU_INT               firmware_running;                    /* OUT */
     ACU_INT               clock;                               /* OUT */
     ACU_INT               codec_count;                         /* OUT */
     IPT_CODEC_INFO        codecs[MAXCODECS];
     ACU_INT               ethernet_ports;                      /* OUT */
     IPT_ETHERNET_INFO     ethernet_status[MAXETHERNET];
     ACU_CHAR              dsp_module_model[ACU_MAX_HWVER];     /* OUT */
     ACU_CHAR              dsp_module_serial_no[MAX_RESOURCE_SERIAL];/* OUT
   */
     ACU_CHAR              firmware_version[IPT_MAX_DESCR];     /* OUT */
     IPT_VALIDITY          switch_clocking;                     /* OUT */
   } IPT_CARD_INFO_XPARMS;
```

Input parameters

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

card_id

Must be set to a valid card id returned by the `acu_open_card()` function. The card must have been opened using `acu_open_ipt()`.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

active
Set to 1 when the card is capable of handling calls, otherwise it is set to zero.

address
This will be populated by the IPv4 address of the ProsodyX card.

firmware_running
Set to zero when firmware is not running on the card, and set to 1 when the card has firmware running.

clock
Contains a value (heartbeat) that is regularly incremented while firmware is running on the card, which may be useful for diagnostic purposes.

codec_count
This will be populated by the number of different codec types that the card being queried supports.

codecs
This is an array of type `CODEC_INFO`, that will be populated according to the codec types available on this particular card.

codec_type
Specifies the codec.

channels_supported
Specifies the number of channels of that codec, supported by the card.

channels_free
Specifies the number of channels currently free for use.

ethernet_ports
This contains the number of physical network ports on the card.

ethernet_status
This array provides information about the status of the physical Ethernet ports on the card.

Active
Set to zero when the port is not being used to provide traffic, otherwise, it is set to 1.

Connected
Set to 1 when the port has detected a physical network connection, otherwise it is set to zero.

Speed

Reports the speed of the network connection in units of Mbits.

Duplex

Set to zero for a half duplex connection and 1 for a full duplex connection.

dsp_module_model
The model number of the DSP module (if any) attached to the card.

dsp_module_serial_no
The serial number of the DSP module (if any) attached to the card.

firmware_version
The version of the firmware running on the card.

switch_clocking

Indicates when the board has detected a valid or invalid clocking setup. The options are:

`IPT_VALID`
`IPT_INVALID`
`IPT_INDETERMINATE` - indicating that there is insufficient information to say either way.

1.4 Card events

Changes in the status of ProsodyX cards are notified to the user through the IP Telephony API. This section details functions for accessing these events.

1.5 ipt_get_card_notification()

Used to retrieve notification events for IP card status changes.

Synopsis

```
ACU_ERR ipt_get_card_notification(IPT_CARD_NOTIFICATION_PARMS* parms);

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

Input Parameters

`ipt_get_card_notification()` takes a pointer, `parms`, to a structure, `IPT_CARD_NOTIFICATION_XPARMS`. The structure must be initialised before invoking the function.

card_id

Must be set to a valid card id returned by the `acu_open_card()` function. The card must have been opened using `acu_open_ipt()`.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

event

The `event` field is set to one of the following values:

#define	Description
IPT_CARD_NO_EVENT	There are no events in the queue
IPT_CARD_EV_L1_CHANGE	Layer 1 has changed on the specified port – use `ipt_card_info()` to determine what the change is
IPT_CARD_EV_IN_SERVICE	The card is operational.
IPT_CARD_EV_OUT_OF_SERVICE	The card is not operational. Use `ipt_card_info()` to determine why.
IPT_CARD_EV_FIRMWARE_STOPPED	The card firmware has stopped running (for example, during a firmware restart).
IPT_CARD_EV_SWITCH_CLOCKING	The switch clocking status of the card has changed. Use ipt_card_info() to determine the new state.

NOTE

These notifications are an indication that something has changed. Upon receipt of one of these events, the application will need to make further API calls to determine what has changed.

NOTE

These notifications are queued. It may be possible that when an application retrieves an event, the state change it is describing has been superseded by another state change. Applications should be designed to cope with this. (i.e. don't assume that because a Layer 1 state change notification has been received, Layer1 has gone down).

To avoid polling this function you can either:

- use `ipt_card_get_notification_wait_object()` to obtain a wait object that is signaled when an event is queued for the card; or
- create an event queue (using `acu_allocate_event_queue()`) and then use `ipt_card_notification_queue()` to associate a particular port with that queue then wait for events to occur on the queue.

1.6 ipt_card_notification_get_wait_object()

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

Synopsis

```
ACU_ERR ipt_card_notification_get_wait_object(IPT_WAIT_OBJECT_PARMS
                                                              *woparms);

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

Input Parameters

`ipt_card_notification_get_wait_object()` takes a pointer, `woparms`, to a structure, `IPT_WAIT_OBJECT_PARMS`. The structure must be initialised before invoking the function.

card_id

Must be set to a valid card id returned by the `acu_open_card()` function. The card must have been opened using `acu_open_ipt()`.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

wait_object

`wait_object` will be set to a valid operating system specific event associated with the specified card.

NOTE

The wait object associated with a card will remain signalled while there are notification events queued for that card.

1.7 ipt_card_set_notification_queue()

This function is used to associate a port with a queue. All port notification events for this port will be notified via this event queue.

Synopsis

```
ACU_ERR ipt_card_set_notification_queue(ACU_QUEUE_PARMS* queue_parms);

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

NOTE

This function can be called at any time – any notification events pending for the card will be transferred from the old queue to the new queue. This struct can be found in the acu_type.h file.

Input Parameters

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

resource_id

The `resource_id` field must be set to a valid card id returned by the `acu_open_card()` function.

queue_id

The `queue_id` field must be set to a valid queue.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

1.8 ipt_card_get_app_context_token()

This function is used to retrieve application-defined data that is associated with a card. The data can be set using `ipt_card_set_app_context_token()`.

Synopsis

```
ACU_ERR ipt_card_get_app_context_token(ACU_APP_CONTEXT_TOKEN_PARMS*
                                                                token_parms);

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

NOTE

This struct can be found in the acu_type.h file.

Input Parameters

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

resource_id

The `resource_id` field 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 value will be returned indicating the type of error.

app_context_token

On successful completion the *app_context_token* field will be set to the associated data.

1.9 ipt_card_set_app_context_token()

This function is used to associate application-defined data with a card. This data is returned as the *context* field by `acu_get_event_from_queue()`.

The token assigned using this function can also be retrieved using `call_get_port_app_context_token()`.

Synopsis

```
ACU_ERR ipt_card_set_app_context_token(ACU_APP_CONTEXT_TOKEN_PARMS*
                                                                token_parms);

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

NOTE

This struct can be found in the acu_type.h file.

Input Parameters

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

resource_id

The `resource_id` field must be set to a valid card id returned by the `acu_open_card()` function.

app_context_token

The *app_context_token* field should be set to the data you want to associate with the card.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

1.10 ipt_card_get_if_stats()

This API call is used to obtain information about the Ethernet interface of the card. It is not supported for Prosody S.

Synopsis

```
ACU_ERR ipt_card_get_if_stats(IPT_CARD_IF_STATS_XPARMS *statsp);

typedef struct tIPT_CARD_IF_STATS_XPARMS
{
  ACU_ULONG       size;
  ACU_CARD_ID     card_id;                  /* IN */
  ACU_INT         mtu;                      /* OUT */
  ACU_INT         speed;                    /* OUT */
  ACU_CHAR        hw_addr[IPT_MAX_MAC];     /* OUT */
  ACU_INT         oper_status;              /* OUT */
  ACU_UINT        in_octets;                /* OUT */
  ACU_UINT        in_ucast_pkts;            /* OUT */
  ACU_UINT        in_nucast_pkts;           /* OUT */
  ACU_UINT        in_discards;              /* OUT */
  ACU_UINT        in_errors;                /* OUT */
  ACU_UINT        in_unknown_protos;        /* OUT */
  ACU_UINT        out_octets;               /* OUT */
  ACU_UINT        out_ucast_pkts;           /* OUT */
  ACU_UINT        out_nucast_pkts;          /* OUT */
  ACU_UINT        out_discards;             /* OUT */
  ACU_UINT        out_errors;               /* OUT */
} IPT_CARD_IF_STATS_XPARMS;
```

Input Parameters

`ipt_card_get_if_stats()` takes a pointer, `statsp`, to a structure, `IPT_CARD_IF_STATS_XPARMS`. The structure must be initialised before invoking the function.

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 value will be returned indicating the type of error.

Mtu
The maximum packet size supported by the Ethernet interface of the card.

Speed
The data speed of the network connection in units of Mbits.

hw_addr
The MAC address of the card.

oper_status
The operational status of the card, for example:

```
1  Up
2  Down
```

Please refer to IETF RFC 2863 for further details.

in_octets
The number of bytes that the Ethernet interface has received.

in_ucast_pkts
The number of unicast packets that the Ethernet interface has received.

in_nucast_pkts
The number of non-unicast packets that the Ethernet interface has received.

in_discards
The number of incoming packets discarded by the Ethernet interface.

in_errors
The number of packets with Ethernet errors detected by the Ethernet interface.

in_unknown_protos
The number of packets with unknown Ethernet protocols received by the Ethernet interface.

out_octets
The number of bytes transmitted by the Ethernet interface.

out_ucast_pkts
The number of unicast packets transmitted by the Ethernet interface.

out_nucast_pkts
The number of non-unicast packets transmitted by the Ethernet interface.

out_discards
The number of packets discarded at the Ethernet layer by the card.

out_errors
The number of Ethernet transmission errors detected by the card.

2 Default configurations

This section details the API functions that enable a user to configure defaults for ProsodyX cards and the signalling protocols.

2.1 Card default configuration

The routines below allow applications to configure defaults used by the IP cards in the system.

2.2 ipt_card_set_media_defaults()

Used to set the media defaults to be used by a particular ProsodyX card on all subsequent calls. These defaults persist while the card is closed and are shared between all applications on the system.

Synopsis

```
ACU_ERR ipt_card_set_media_defaults(IPT_MEDIA_DEFAULTS_XPARMS* mdp);

   typedef struct tMEDIA_DEFAULTS
   {
     ACU_ULONG             size;
     ACU_CARD_ID           card_id;              /* IN */
     ACU_INT               tdm_encoding;         /* IN */
     ACU_INT               encode_gain;          /* IN */
     ACU_INT               decode_gain;          /* IN */
     ACU_INT               reserved;             /* IN */
     ACU_INT               echo_cancellation;    /* IN */
     ACU_INT               echo_suppression;     /* IN */
     ACU_INT               echo_span;            /* IN */
     ACU_UINT              rtp_tos;              /* IN */
     ACU_UINT              rtcp_tos;             /* IN */
     ACU_UINT              def_jitter;           /* IN */
     ACU_UINT              max_jitter;           /* IN */
     ACU_UINT              max_jitter_buffer;    /* IN */
     ACU_UINT              dtmf_detector;        /* IN */
   } IPT_MEDIA_DEFAULTS_XPARMS;
```

Input parameters

`ipt_card_set_media_defaults()` takes a pointer, `mdp`, to a structure, `IPT_MEDIA_DEFAULTS_XPARMS`. The structure must be initialised before invoking the function.

card_id

Must be set to a valid card id returned by the `acu_open_card()` function. The card must have been opened using `acu_open_ipt()`.

tdm_encoding

The `tdm_encoding` parameter allows μ-law or a-law encoding to be selected for the telephony interface on a per card basis, and has no effect on the selected IP Telephony codec. When no value is specified, the system will default to the encoding configured for the firmware, which is currently set to μ-law. The permitted values are:

```
TDM_ULAW  1
TDM_ALAW  2
```

encode_gain/decode_gain

The `encode_gain` parameter allows adjustment of the input signal from the telephony interface to the IP Telephony encoder, while the `decode_gain` parameter allows adjustment of the output signal from the IP Telephony decoder to the telephony interface. Permitted values for these two parameters are:

0	to use the existing default gain level
0x0001 – 0xFFFF	specify a gain level manually

NOTE

encode_gain and decode_gain are not supported on Prosody X cards, and will be ignored.

echo_cancellation

The possible values are:

`EC_OFF` – disables echo canceller (invalidates `echo_span` option)

`EC_ON` – enables G.165 echo canceller for ProsodyX cards and G.168 echo canceller for Prosody X cards

`EC_ON_NLP` – enables G.168 echo canceller with non-linear processing for Prosody X cards

The default value is `EC_ON`

echo_suppression

The possible values are:

`ES_OFF` - echo suppression option is disabled

`ES_12DB` - echo suppression option is enabled

The default value is `ES_OFF`

NOTE

The echo canceller and suppressor are independent subsystems of the echo software and as such can be controlled independently.

echo_span

This is the length, in milliseconds, of the echo canceller tail. It may be 4, 6, 8, 10, 12, 14, 16 or 32ms tail length.

NOTE

A 32ms tail length cannot be used with G.723.1

The default value is 16.

NOTE

echo_suppression and echo_span are not supported on Prosody X cards, and will be ignored.

rtp_tos (see note *)
The field `rtp_tos` specifies the value of the 8 bit type of service field that will be used in the IP headers of RTP packets sent by the board on a per call basis for `call_openout` and `xcall_accept` functions. To set a value of zero, a value of 0x100 should be used.

rtcp_tos (see note *)
The field `rtcp_tos` specifies the value of the 8 bit type of service field that will be used in the IP headers of RTCP packets sent by the board on a per call basis for `call_openout` and `xcall_accept` functions. To set a value of zero, a value of 0x100 should be used.

def_jitter, max_jitter and max_jitter_buffer
The integer fields `def_jitter`, `max_jitter` and `max_jitter_buffer` contain respectively the default jitter, maximal jitter and maximal transient jitter that will be used by the board, expressed in milliseconds. The amount of the jitter buffering used will vary adaptively between 10ms and `max_jitter` with `def_jitter` being the amount at the start of a call. The value specified by `max_jitter_buffer` limits the maximum depth of the jitter buffer at any one moment and should be greater than `def_jitter`.

NOTE

max_jitter_buffer is not supported on Prosody X cards, and will be ignored

dtmf_detector
The ProsodyX card can detect DTMF in the audio stream switched to it and treat it differently to normal audio by blocking DTMF in the outgoing audio stream and sending RFC 2833 frames instead.

When `dtmf_detector` is set to `IPT_ENABLED` then this processing will be performed.

when `dtmf_detector` is set to `IPT_DISABLED` then DTMF will not be detected, and will be treated as normal audio.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

NOTE

*** See the call API documentation for further details on `call_openout` and `xcall_accept`.**

2.3 ipt_card_get_media_defaults()

Used to get the media defaults that are to be used, by a particular card, on all subsequent calls. These defaults are shared by all programs running on the system.

Synopsis

```
ACU_ERR ipt_card_get_media_defaults(IPT_MEDIA_DEFAULTS_XPARMS* mdp);

   typedef struct tMEDIA_DEFAULTS
   {
     ACU_ULONG             size;
     ACU_CARD_ID           card_id;              /* IN */
     ACU_INT               tdm_encoding;         /* OUT */
     ACU_INT               encode_gain;          /* OUT */
     ACU_INT               decode_gain;          /* OUT */
     ACU_INT               reserved;             /* OUT */
     ACU_INT               echo_cancellation;    /* OUT */
     ACU_INT               echo_suppression;     /* OUT */
     ACU_INT               echo_span;            /* OUT */
     ACU_UINT              rtp_tos;              /* OUT */
     ACU_UINT              rtcp_tos;             /* OUT */
     ACU_UINT              def_jitter;           /* OUT */
     ACU_UINT              max_jitter;           /* OUT */
     ACU_UINT              max_jitter_buffer;    /* OUT */
     ACU_UINT              dtmf_detector;        /* OUT */
   } IPT_MEDIA_DEFAULTS_XPARMS;
```

Input parameters

`ipt_card_get_media_defaults()` takes a pointer, `mdp`, to a structure, `IPT_MEDIA_DEFAULTS_XPARMS`. The structure must be initialised before invoking the function.

card_id

Must be set to a valid card id returned by the `acu_open_card()` function. The card must have been opened using `acu_open_ipt()`.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

The structure will be initialised as for `ipt_set_media_defaults()`. See section 2.2 for parameter definitions.

2.4 Protocol specific configurations

The routines detailed here provide access to the defaults used by IP telephony protocols in the Aculab Call API.

2.5 ipt_set_protocol_defaults()

Used to set global protocol defaults that are used by the IP telephony protocols in the Aculab Call API.

Synopsis

```
ACU_ERR ipt_set_protocol_defaults(PROTOCOL_DEFAULTS_XPARMS
                                                    *protocol_defaultsp);

   typedef struct
   {
     ACU_ULONG        size;
     ACU_UINT         protocol;                        /* IN */
     ACU_CODEC        default_codecs[MAXCODECS];       /* IN */
     union
     {
       struct
       {
         ACU_INT      h245_tunneling;                  /* IN */
         ACU_INT      faststart;                       /* IN */
         ACU_INT      early_h245;                      /* IN */
       } sig_h323;
       struct
       {
         ACU_INT      zero_connection_address_hold;    /* IN */
         ACU_INT      disable_early_media;             /* IN */
       } sig_sip;
     } unique_xparms;
   } PROTOCOL_DEFAULTS_XPARMS;
```

Input parameters

`ipt_set_protocol_defaults()` takes a pointer, `protocol_defaultsp`, to a structure, `PROTOCOL_DEFAULTS_XPARMS`. The structure must be initialised before invoking the function.

protocol

Used to identify the protocol service to which you wish to send the request. Valid values are:

`S_H323` send to the H.323 service

`S_SIP` send to the SIP service

default_codecs

This must be populated with an array of codec types. The IP telephony service will cache this array and use it by default in calls to `call_openout` and `xcall_accept` when the user omits to set the codecs in these calls. When codecs are specified that are not supported by a given board then they will be ignored for calls on that board.

H.323 specific parameters

h245_tunneling

Allows H.245 tunnelling to be enabled by default for H.323 calls. This is the process of sending H.245 PDUs through the Q.931 channel (encapsulating the H.245 messages within H.225/Q.931 messages). The same TCP/IP socket that is already in use for the call signalling channel is also used by the H.245 control channel. When set to `IPT_ENABLED`, tunnelling is enabled. When set to `IPT_DISABLED`, tunnelling is disabled.

faststart

Allows Fast Start, also known as Fast Connect, to be enabled by default for H.323 calls. This procedure reduces the time required to set up a call to one round-trip delay

following the H.225 TCP connection and allows audio data to be transmitted prior to the call being connected. When set to `IPT_ENABLED`, Fast Start is enabled. When set to `IPT_DISABLED`, Fast Start is disabled.

early_h245
Allows early H.245 to be enabled by default for H.323 calls. This involves opening the H.245 channel before the call has been accepted, allowing the call to be connected more quickly, and the transmission of audio data. When set to `IPT_ENABLED`, early H.245 is enabled. When set to `IPT_DISABLED`, early H.245 is disabled.

SIP specific parameters

zero_connection_address_hold
When set to `IPT_ENABLED` the service assumes that the remote party implements call hold to the earlier Internet specification, that is c=0.0.0.0 in the SDP body.

When set to `IPT_DISABLED` the service assumes the latest specification (`a=sendonly/recvonly`).

When set to `0`, the service will use the default value. This may be set in `call_openout` or `xcall_accept`.

The default value is `IPT_DISABLED`.

disable_early_media
For an outgoing call, when this is set to `IPT_ENABLED` the calling party refuses to participate in an early media session, even when one is offered by the called party.

When set to `IPT_DISABLED`, the calling party will participate in such sessions when offered by called party.

When set to `0`, the service will use the default value.

The default value is `IPT_DISABLED`.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

2.6 ipt_get_protocol_defaults()

This routine allows clients to get defaults used by the IP Telephony protocols in the Aculab Call API.

Synopsis

```
ACU_ERR ipt_get_protocol_defaults(PROTOCOL_DEFAULTS_XPARMS
                                                    *protocol_defaultsp);

  typedef struct
  {
    ACU_ULONG         size;
    ACU_UINT          protocol;                     /* IN */
    ACU_CODEC         default_codecs[MAXCODECS];    /* OUT */
    union
    {
      struct
      {
        ACU_INT       h245_tunneling;               /* OUT */
        ACU_INT       faststart;                    /* OUT */
        ACU_INT       early_h245;                   /* OUT */
      } sig_h323;
      struct
      {
        ACU_INT       zero_connection_address_hold; /* OUT */
        ACU_INT       disable_early_media;          /* OUT */
      } sig_sip;
    } unique_xparms;
  } PROTOCOL_DEFAULTS_XPARMS;
```

Input parameters

`ipt_get_protocol_defaults()` takes a pointer, `protocol_defaultsp`, to a structure, `PROTOCOL_DEFAULTS_XPARMS`. The structure must be initialised before invoking the function.

protocol

Used to identify the protocol service to which you wish to send the request. Valid values are:

`S_H323`	send to the H.323 service
`S_SIP`	send to the SIP service

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

The structure will be initialised as documented for `ipt_set_protocol_defaults()`. See section 2.5 for parameter definitions. The current values for each of the settings used by the service will be returned in these fields.

3 SIP socket selection

By default the SIP service opens listening sockets on all the local hosts NIC cards on port=5060 and transport=UDP. Therefore, when a host possesses 2 NIC's, one configured as 1.2.3.4 and the other 5.6.7.8, then the SIP service will listen on 2 sockets as follows:

```
1.2.3.4:5060:UDP
5.6.7.8:5060:UDP
```

The SIP user agent stack, utilised by our SIP service, can be configured to listen for SIP signalling on a multitude of sockets.

In V6 we are providing the user access to this facility and allowing them to fine tune the sockets on which SIP traffic is listened for by using the following API functions.

3.1 ipt_add_listen_address()

Used to add a specific listen parameter to the current set that is being used by the stack to listen for SIP traffic.

Synopsis

```
ACU_ERR ipt_add_listen_address(LISTEN_XPARMS *listen_detailsp);

   typedef struct
   {
     ACU_ULONG             size;
     HOST_DETAILS          listen_details;          /* IN */
   } LISTEN_XPARMS;

   typedef struct host_details
   {
     ACU_CHAR              address[MAXHOSTADDRESS];  /* IN */
     ACU_UINT              port;                     /* IN */
     ACU_INT               transport_type;           /* IN */
   } HOST_DETAILS;
```

Input parameters

`ipt_add_listen_address()` takes a pointer, `listen_detailsp`, to a structure, `LISTEN_XPARMS`. The structure must be initialised before invoking the function.

listen_details
Details of the listen parameter being added to the set being listened on by the SIP UA.

address
IP address or FQDN (Fully Qualified Domain Name) specifying the NIC in the host.

port
IP port on which to listen for traffic. When zero is used we default to 5060.

transport_type
Protocol type to listen on, `ACUTCP` or `ACUUDP`. When set to zero `ACUUDP` is used.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

3.2 ipt_remove_listen_address()

Used to remove a specific listen parameter from the current set being used by the stack to listen for SIP traffic.

Synopsis

```
ACU_ERR ipt_remove_listen_address(LISTEN_XPARMS *listen_detailsp);

   typedef struct
   {
     ACU_ULONG             size;
     HOST_DETAILS          listen_details;
   } LISTEN_XPARMS;

   typedef struct host_details
   {
     ACU_CHAR              address[MAXHOSTADDRESS];     /* IN */
     ACU_UINT              port;                        /* IN */
     ACU_INT               transport_type;              /* IN */
   } HOST_DETAILS;
```

Input parameters

`ipt_remove_listen_address()` takes a pointer, `listen_detailsp`, to a structure, `LISTEN_XPARMS`. The structure must be initialised before invoking the function.

listen_details
Details of the listen parameter being removed from the set being listened on by the SIP UA.

address
IP address or FQDN (Fully Qualified Domain Name) specifying the NIC in the host.

port
IP port. When zero is used we default to 5060.

transport_type
Protocol type, `ACUTCP` or `ACUUDP`. When set to zero `ACUUDP` is used.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

3.3 ipt_get_listen_list()

This function can be used to query the list of listen parameters currently being used by the SIP stack. When successful, a structure that is populated with the set of listen parameters being used by the SIP UA will be returned.

Synopsis

```
ACU_ERR ipt_get_listen_list(LISTEN_LIST_XPARMS *listen_listp);

  typedef struct
  {
    ACU_ULONG             size;
    ACU_UINT              num_of_listen_params;                /* OUT */
    HOST_DETAILS          listen_details[MAXLISTENPARAMS];
  } LISTEN_LIST_XPARMS;

  typedef struct host_details
  {
    ACU_CHAR              address[MAXHOSTADDRESS];             /* OUT */
    ACU_UINT              port;                                /* OUT */
    ACU_INT               transport_type;                      /* OUT */
  } HOST_DETAILS;
```

Input parameters

`ipt_get_listen_list()` takes a pointer, `listen_listp`, to a structure, `LISTEN_LIST_XPARMS`.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

num_of_listen_params
The number of listen parameter structures populated by this call.

listen_details
An array of listen parameter structures being used by the SIP stack. Each structure will contain the following information.

address
IP address or FQDN (Fully Qualified Domain Name) specifiying a NIC.

port
IP port.

transport_type
Protocol type.

4 H.323-specific functionality

The following functions are specific to the H.323 protocol and cannot be used by SIP.

4.1 ipt_translate_h225rcr()

This function can be used to translate an H.225 Release Complete Reason to a Q.931 clearing cause.

Synopsis

```
ACU_INT ipt_translate_h225rcr(ACU_INT h225_rcr);
```

Input parameters

h225_rcr

The H.225 Release Complete Reason that you wish to translate.

Return values

On successful completion, a positive value representing the mapped Q.931 clearing cause is returned; otherwise, a negative value will be returned indicating the type of error.

4.2 ipt_set_dtmf_handling()

NOTE

Although this function call is valid, the preferred API call for controlling DTMF UII is call_set_dtmf_handling

This function can be used to control DTMF User Input Indication event notification and relay.

Synopsis

```
ACU_INT ipt_set_dtmf_handling(DTMF_HANDLING_XPARMS *dtmf_handlingp);

   typedef struct
   {
     ACU_ULONG             size;
     ACU_CALL_HANDLE       handle;                    /* IN */
     ACU_UINT              enable_event_notification; /* IN */
     ACU_UINT              relay_disabled;            /* IN */
   } DTMF_HANDLING_XPARMS;
```

Input parameters

`ipt_set_dtmf_handling()` takes a pointer, `dtmf_handlingp`, to a structure, `DTMF_HANDLING_XPARMS`. The structure must be initialised before invoking the function.

handle

The handle field is used to identify the call to which the DTMF handling options are to apply.

enable_event_notification

When set to 1, any User Input Indications that are received from the network will be identified at the API level through event notification. Valid values are:

`0` – disable event notification

`1` – enable event notification

relay_disabled **(Not applicable to TiNG media configurations)**

By default, any User Input Indications that are received from the network will be relayed to the TDM side. This is disabled by setting `relay_disabled` to `1`.

0 – relay enabled

1 – relay disabled

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

When the network indicates an `EV_DETAILS`, `call_details()` will allow an application to see the DTMF information.

4.3 ipt_set_h323_listen_addresses()

This function can be used to change the addresses to listen on for H.225 and RAS messages. By default we listen on the first local interface, port 1720 for H.225 and port 1719 for RAS

Synopsis

```
ACU_ERR ipt_set_h323_listen_addresses(H323_LISTEN_ADDRESSES_XPARMS* listenp);

   typedef struct
   {
     ACU_ULONG         size;
     ACU_CHAR          h225_address[MAXHOSTADDRESS];     /* IN */
     ACU_UINT          h225_port;                        /* IN */
     ACU_CHAR          ras_address[MAXHOSTADDRESS];      /* IN */
     ACU_UINT          ras_port;                         /* IN */
   } H323_LISTEN_ADDRESSES_XPARMS;
```

Input Parameters

`ipt_set_h323_listen_addresses()` takes a pointer, `listenp`, to a structure, `H323_LISTEN_ADDRESSES_XPARMS`. The structure must be initialised before invoking the function.

h225_address
The IP address or host name that will resolve to the IP address on which to listen.

h225_port
The port to listen on for H.225 messages.

ras_address
The IP address or host name that will resolve to the IP address on which to listen.

ras_port
The port to listen on for RAS messages.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

4.4 ipt_get_h323_listen_addresses()

This function can be used to query the addresses to listen on for H.225 and RAS messages.

Synopsis

```
ACU_ERR ipt_get_h323_listen_addresses(H323_LISTEN_ADDRESSES_XPARMS* listenp);
```

```
typedef struct
{
  ACU_ULONG        size;
  ACU_CHAR         h225_address[MAXHOSTADDRESS];    /* OUT */
  ACU_UINT         h225_port;                       /* OUT */
  ACU_CHAR         ras_address[MAXHOSTADDRESS];     /* OUT */
  ACU_UINT         ras_port;                        /* OUT */
} H323_LISTEN_ADDRESSES_XPARMS;
```

Input Parameters

`ipt_get_h323_listen_addresses()` takes a pointer, `listenp`, to a structure, `H323_LISTEN_ADDRESSES_XPARMS`.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

h225_address
The IP address currently being used to listen for H.225 messages.

h225_port
The port currently being used to listen for H.225 messages.

ras_address
The IP address currently being used to listen for RAS messages.

ras_port
The port currently being used to listen for RAS messages.

4.5 ipt_h323_stop_listening()

This function can be used to stop listening for H.225 or RAS messages.

Synopsis

```
ACU_ERR ipt_h323_stop_listening(H323_STOP_LISTEN_XPARMS* listenp);

   typedef struct
   {
     ACU_ULONG         size;
     ACU_INT           protocol;      /* IN */
   } H323_STOP_LISTEN_XPARMS;
```

Input Parameters

`ipt_h323_stop_listening()` takes a pointer, `listenp`, to a structure, `H323_STOP_LISTEN_XPARMS`. The structure must be initialised before invoking the function.

protocol

This should be set to indicate which messages should no longer be listened for:

`IPT_H225_PROTOCOL`	H.225 messages.
`IPT_RAS_PROTOCOL`	RAS messages.
`IPT_H225_RAS_PROTOCOL`	Both H.225 and RAS messages.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

4.6 ipt_h323_send_non_standard() - Call independent signalling

Some supplementary services require the use of a connectionless network service to transmit some messages. `ipt_h323_send_non_standard()` allows these messages to be transmitted.

This message is not associated with a call and no handle is associated with it.

Synopsis

```
ACU_ERR ipt_h323_send_non_standard(H323_NON_STANDARD_XPARMS* nsp);

typedef struct h323_non_standard_xparms
{
  ACU_ULONG                              size;
  ACU_INT                                message_type;       /* IN */
  ADDRESSED_NON_STANDARD_DATA_XPARMS     data;               /* IN */
} H323_NON_STANDARD_XPARMS;
```

See the call control API guide for further details on *ADDRESSED_NON_STANDARD_DATA_XPARMS* definitions

Input parameters

The `ipt_h323_send_non_standard()` function takes a pointer, `nsp`, to a structure, `H323_NON_STANDARD_XPARMS`. The structure must be initialised before invoking the function.

message_type

The `message_type` field should be used to indicate the type of feature.

To send a connectionless `FACILITY` message the `message_type` field should be supplied with the value `IPT_NSM_CONNECTIONLESS_FACILITY`.

To send a H.225 RAS non-standard message the `message_type` field should be supplied with the value `IPT_NSM_RAS`.

To send a H.225 RAS XRS message the `message_type` field should be supplied with the value `IPT_NSM_XRS`.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

NOTE

A successful invocation of ipt_h323_send_non_standard() is no guarantee that the network side will receive the message. When an error occurs, the message may be discarded.

4.7 ipt_h323_get_non_standard() - Call independent signalling

Some supplementary services require the use of a connectionless network service to transmit some messages. `ipt_h323_get_non_standard()` allows these messages to be read. In order to enable reception of these messages, `ipt_h323_enable_non_standard()` must be called by the application.

Synopsis

```
ACU_ERR ipt_h323_get_non_standard(H323_NON_STANDARD_XPARMS* nsp);

typedef struct h323_non_standard_xparms
   {
     ACU_ULONG                          size;
  ACU_INT                               message_type;      /* OUT */
  ADDRESSED_NON_STANDARD_DATA_XPARMS    data;              /* OUT */
   } H323_NON_STANDARD_XPARMS;
```

See the call control API guide for further details on the `ADDRESSED_NON_STANDARD_DATA_XPARMS` definitions.

Input parameters

The `ipt_h323_get_non_standard()` function takes a pointer, `nsp`, to a structure, `H323_NON_STANDARD_XPARMS`. The structure must be initialised before invoking the function.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

message_type

The `message_type` field should be used to indicate the type of feature.

When a connectionless `FACILITY` message has been read the `message_type` field should be supplied with the value `IPT_NSM_CONNECTIONLESS_FACILITY`.

When a H.225 RAS non-standard message has been read the `message_type` field should be supplied with the value `IPT_NSM_RAS`.

When a H.225 RAS XRS message has been read the `message_type` field should be supplied with the value `IPT_NSM_XRS`.

4.8 ipt_h323_enable_non_standard() - Call independent signalling

Some supplementary services require the use of a connectionless network service to transmit some messages.

Synopsis

```
ACU_ERR ipt_h323_enable_non_standard(H323_NON_STANDARD_ENABLE_XPARMS*
enablep);

typedef struct h323_non_standard_xparms
   {
     ACU_ULONG              size;
  ACU_INT                   message_type;      /* IN */
  ACU_INT                   enable;            /* IN */
} H323_NON_STANDARD_ENABLE_XPARMS;
```

Input parameters

The `ipt_h323_enable_non_standard()` function takes a pointer, *enablep,* to a structure, `H323_NON_STANDARD_ENABLE_XPARMS`. The structure must be initialised before invoking the function.

message_type

The `message_type` field should be used to indicate the type of feature.

To control connectionless FACILITY messages the `message_type` field should be set to `IPT_NSM_CONNECTIONLESS_FACILITY`.

To control H.225 RAS non-standard messages the `message_type` field should be set to `IPT_NSM_RAS`. While these messages are enabled, the application is responsible for generating XRS messages for any non-standard messages that it does not understand.

To control H.225 RAS XRS messages the `message_type` field should be set to `IPT_NSM_XRS`.

enable

The `enable` field should be set to zero to disable notification of the specified message type or 1 to enable notification of the specified message type.

Return values

On successful completion, a value of zero is returned; otherwise a negative value will be returned indicating the type of error.

5 IP registration API

This section details the API functions required for use with SIP proxy and H.323 gatekeeper registration work.

5.1 Proxy/Gatekeeper configuration

The routines below allow clients to configure the proxies/gatekeepers they want to use in their system.

For further guidance on H.323 registration, please see Appendix A:

5.2 ipt_set_sip_proxy()

This function can be used to set the local outbound proxy details in the SIP service. Once this is done all outgoing requests are routed via this proxy.

Synopsis

```
ACU_ERR ipt_set_sip_proxy(SIP_PROXY* proxy_detailsp);

   typedef struct sip_proxy
   {
     ACU_ULONG             size;
     HOST_DETAILS          proxy;
     ACU_INT               disable_insertion_into_routeset;      /* IN */
   } SIP_PROXY;

   typedef struct host_details
   {
     ACU_CHAR              address[MAXHOSTADDRESS];              /* IN */
     ACU_UINT              port;                                 /* IN */
     ACU_INT               transport_type;                       /* IN */
   } HOST_DETAILS;
```

Input parameters

`ipt_set_sip_proxy()` takes a pointer, `proxy_detailsp`, to a structure, `SIP_PROXY`. The structure must be initialised before invoking the function.

proxy
Address details of the proxy to be set.

address
IP address or FQDN (Fully Qualified Domain Name) of the proxy.

port
Port number. When set to zero, port number 5060 is used.

transport_type
One of the following:

ACU_SIP_TRANSPORT_UDP, ACU_SIP_TRANSPORT_TCP, ACU_SIP_TRANSPORT_TLS, ACU_SIP_TRANSPORT_ANY

ACU_SIP_TRANSPORT_ANY means use the preferred transport as specified by the target service.

disable_insertion_into_route_set
Reserved for future use.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

5.3 ipt_query_sip_proxy()

This function can be used to query the SIP service for the details of the local outbound proxy that is currently set.

Synopsis

```
ACU_ERR ipt_query_sip_proxy(SIP_PROXY* proxy_queryp);
```

```
typedef struct sip_proxy
{
  ACU_ULONG             size;
  HOST_DETAILS          proxy;
  ACU_INT               disable_insertion_into_routeset;
} SIP_PROXY;
```

```
typedef struct host_details
{
  ACU_CHAR              address[MAXHOSTADDRESS];        /* OUT */
  ACU_UINT              port;                           /* OUT */
  ACU_INT               transport_type;                 /* OUT */
} HOST_DETAILS;
```

Input Parameters

`ipt_query_sip_proxy()` takes a pointer, `proxy_queryp`, to a structure, `SIP_PROXY`.

disable_insertion_into_route_set
Reserved for future use.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

proxy
Address details of the local outbound proxy that is currently set.

address
IP addressor FQDN (Fully Qualified Domain Name) of the proxy.

port
Port number.

transport_type
Will be `ACU_SIP_TRANSPORT_UDP`, `ACU_SIP_TRANSPORT_TCP`, `ACU_SIP_TRANSPORT_TLS` or `ACU_SIP_TRANSPORT_ANY`.

5.4 ipt_clear_sip_proxy()

This function can be used to stop using the currently set local outbound proxy.

Synopsis

```
ACU_ERR ipt_clear_sip_proxy();
```

Input Parameters

None.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

5.5 ipt_set_h323_gatekeeper()

This function can be used to register the system with a gatekeeper. Once this is done, all outgoing requests are routed via this gatekeeper. Please note that this applies to the whole system, so in a multiple application environment when one application calls this function any additional applications will also be registered.

Synopsis

```
ACU_ERR ipt_set_h323_gatekeeper(REGISTER_XPARMS *regdetailsp);

   typedef struct register_xparms
   {
     ACU_ULONG             size;
     ACU_INT               ttl;                                    /* IN */
     ACU_CHAR              registration_address[MAXREGADDR];       /* IN */
     ACU_CHAR              gatekeeper_id[MAXID];                   /* IN */
     ACU_INT               registration_mode;                      /* IN */
   } REGISTER_XPARMS;
```

Input Parameters

`ipt_set_h323_gatekeeper()` takes a pointer, `regdetailsp`, to a structure, `REGISTER_XPARMS`. The structure must be initialised before invoking the function.

ttl

This is the time (in seconds) that a registration remains valid. The H.323 service will automatically renew the registrations at this interval (or at the interval specified by the gatekeeper if it chose to override this value).

registration_address

This contains the IP address OR the hostname of the gatekeeper to be used for RAS management.

gatekeeper_id

This contains a valid ID for the gatekeeper to which you wish to register. This is optional but may be required by some gatekeepers to accept an endpoint into its zone.

registration_mode

Within the H.323 standard, registered endpoints can be H.323 Terminals, Gateways or MCUs. The Aculab ProsodyX card only supports being a terminal or a gateway. The `registration_mode` contains the mode type to be used for the system. Valid values are:

`IPT_REG_MODE_GATEWAY`	Gateway
`IPT_REG_MODE_TERMINAL`	Terminal

When set to zero, `IPT_REG_MODE_GATEWAY` will be used.

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

5.6 ipt_query_h323_gatekeeper()

This function can be used to query the H.323 system and determine if we are registered with a gatekeeper or not. When we are, the address of the gatekeeper will also be returned.

Synopsis

```
ACU_ERR ipt_query_h323_gatekeeper(QUERY_REGISTRATION_XPARMS *queryp);

   typedef struct query_registration_xparms
   {
     ACU_ULONG        size;
     ACU_INT          registration_status;                    /* OUT */
     ACU_CHAR         registration_address[MAXREGADDR];       /* OUT */
   } QUERY_REGISTRATION_XPARMS;
```

Input Parameters

`ipt_query_h323_gatekeeper()` takes a pointer, `queryp`, to a structure, `QUERY_REGISTRATION_XPARMS`.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

registration_status
A boolean value that indicates if the system is registered or not.

0 – indicates that the system is not registered with a gatekeeper

1 – indicates that the system is registered with a gatekeeper

registration_address
This contains the IP address OR hostname of the gatekeeper to which the system is registered.

5.7 ipt_clear_h323_gatekeeper()

This function can be used to unregister the system from the gatekeeper. Please note that this applies to the whole system, so in a multiple application environment when one application calls this function any additional applications will also be unregistered.

Synopsis

```
ACU_ERR ipt_clear_h323_gatekeeper();
```

Input Parameters

None.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

5.8 Registration functionality

The functions detailed in this section are used to manage the database of well-known addresses (aliases) to real addresses (contacts) on a proxy or gatekeeper. Note that the response from the proxy/gateway to these functions is asynchronous. A global Call API event will be generated to notify the application to notify it that something has happened.

5.9 ipt_add_alias()

`ipt_add_alias()` can be used for both H.323 and SIP protocols.

For SIP this function can be used to add a mapping between a well-known address (an "address of record" or alias) and a real contact address on a registrar's database.

For H.323 this function can be used to register an alias address with the gatekeeper. Alias addresses provide an alternative method of addressing endpoints. For example, a given endpoint may have an E.164 address, e-mail address, and a web page address. Please note that only one alias address can be registered per function call.

Synopsis

```
ACU_ERR ipt_add_alias(ADD_ALIAS_XPARMS* add_aliasp);
```

```
typedef struct add_alias_xparms
{
  ACU_ULONG             size;
  ACU_INT               protocol;                   /* IN */
  ACU_UINT              registration_handle;        /* OUT */
  ACU_CHAR              alias[MAXALIAS];            /* IN */
  union
     {
    struct
    {
      SIP_ADD_ALIAS     sip_add_alias;
    }sig_sip;
    struct
    {
      ACU_INT           prefix;                     /* IN */
    }sig_h323;
  } unique_xparms;
} ADD_ALIAS_XPARMS;

typedef struct sip_add_alias
{
  HOST_DETAILS          registrar;
  ACU_CHAR              admin_sip_url[MAXALIAS];    /* IN */
  ACU_CHAR              contact[MAXALIAS];          /* IN */
} SIP_ADD_ALIAS;

typedef struct host_details
{
  ACU_CHAR              address[MAXHOSTADDRESS];    /* IN */
  ACU_UINT              port;                       /* IN */
  ACU_CHAR              transport_type;             /* IN */
} HOST_DETAILS;
```

Input Parameters

`ipt_add_alias()` takes a pointer, `add_aliasp`, to a structure, `ADD_ALIAS_XPARMS`. The structure must be initialised before invoking the function.

protocol
Used to identify the protocol to which we wish to send the request. Valid values are:

`S_H323` Register a H.323 alias.

`S_SIP` Register a SIP alias.

alias
This is the alias address that we wish to register. It should be in the form of a URL. URLs are written as follows:

`<scheme>:<scheme-specific-part>`

A URL contains the name of the scheme being used (`<scheme>`) followed by a colon and then a string (the <scheme-specific-part>) whose interpretation depends on the scheme.

When the alias supplied does not conform exactly to the format detailed above, for example the `<scheme>`: section missing, the service will try to determine what has been entered.

With SIP when using the common Internet scheme syntax, when the host part of the address is omitted then the local outbound proxy or multicast address will be used as a default.

unique_xparms
The input parameter `unique_xparms` is a union that provides extensions required by specific signalling systems.

Unique parameters for SIP

registrar
Address details of the registrar to which to send the alias request. When not set (i.e. zeroed) the service will first try to use the "local outbound proxy". If unsuccessful it will try to use the SIP registrar multicast address - `sip.mcast.net`.

address
IP address or FQDN (Fully Qualified Domain Name) of the proxy.

port
Port number. When set to zero, port number 5060 is used.

transport_type
`ACUTCP` or `ACUUDP`. When set to zero, `ACUUDP` is used.

admin_sip_url
This is the SIP URL of the agent responsible for the registration. When not set the alias address will be used instead.

contact
A contact address (sip, tel: etc.) to which the alias is mapped (default scheme is `sip`), by the registrar/proxy. e.g.

`sip:matt@10.202.165.150`. or `matt@10.202.165.150`.

If the AsyncRegister = 1 is written in the sipserv.cfg all ipt_add_alias() function calls will return immediately with error code = 0. This enables the asynchronous mode. If it is not possible to resolve the address of the registrar EV_ADD_ALIAS_FAILED is raised.

If the AsyncRegister = 0 or is not included in the sipserv.cfg the ipt_add_alias()

function would wait until the address was resolved before returning the error code, which can block the calling thread for a significant period of time.

Unique parameters for H.323

prefix

Used to identify that the alias supplied is a prefix. When prefix is true, then the alias must be in the form of a tel: URL.

- `0` indicates that the alias is not a prefix
- `1` indicates that the alias is a prefix

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

registration_handle

This is a unique registration identification value that is assigned when an alias is registered. It will be associated throughout the lifetime of the registration and will be returned with any event notification relating to that alias. This value must be used for all subsequent operations relating to the registration.

5.10 ipt_remove_alias()

This function can be used to remove an alias from the registration system, be it SIP or H.323. For SIP it will remove the mapping that was added using `ipt_add_alias()`. For H.323 it will un-register the alias that was previously registered using `ipt_add_alias()`. The registration handle for the alias will not be released as a result of making this call. We need the handle for `ipt_query_alias()` to determine the remote ends response to our local request to remove the alias. The `ipt_delete_alias()` function must be used to release the handle.

Synopsis

```
ACU_ERR ipt_remove_alias(REMOVE_ALIAS_XPARMS* remove_aliasp);
```

```
typedef struct remove_alias_xparms
{
  ACU_ULONG             size;
  ACU_UINT              protocol;                   /* IN */
  ACU_UINT              registration_handle;        /* IN */
} REMOVE_ALIAS_XPARMS;
```

Input Parameters

`ipt_remove_alias()` takes a pointer, `remove_aliasp`, to a structure, `REMOVE_ALIAS_XPARMS`. The structure must be initialised before invoking the function.

protocol

Used to identify the protocol. Valid values are:

`S_H323` Remove a H.323 alias.

`S_SIP` Remove a SIP registration.

registration_handle

Must contain a valid registration handle as generated by `ipt_add_alias()`. It is used to identify which registration you wish to remove.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

5.11 ipt_delete_alias()

This function can be used to remove an alias from the registration system for H.323. For SIP it will delete the handle created using `ipt_add_alias()`. For SIP to remove the mapping call `ipt_remove_alias()` first. For H.323 it will un-register the alias that was previously registered using `ipt_add_alias()`. The registration handle will be deleted and may no longer be used by the application.

Synopsis

```
ACU_ERR ipt_delete_alias(REMOVE_ALIAS_XPARMS* delete_aliasp);
```

```
typedef struct remove_alias_xparms
{
  ACU_ULONG             size;
  ACU_UINT              protocol;                   /* IN */
  ACU_UINT              registration_handle;        /* IN */
} REMOVE_ALIAS_XPARMS;
```

Input Parameters

`ipt_delete_alias()` takes a pointer, `delete_aliasp`, to a structure, `REMOVE_ALIAS_XPARMS`. The structure must be initialised before invoking the function.

protocol

Used to identify the protocol. Valid values are:

`S_H323`	H.323
`S_SIP`	SIP

registration_handle

Must contain a valid registration handle as generated by `ipt_add_alias()`. It is used to identify which registration you wish to delete.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

5.12 ipt_query_alias()

This function can be used to query the registration system about a particular alias. It is normally called as a result of an event notification relating to a particular alias, but can also be called as a general query.

For H.323 this function queries the system to determine the state of a previously registered alias. As a result of an un-register request from the local or remote end, error information will also be provided in the form of an error code.

For SIP, this function should be called after an attempt to register an alias has failed only, as indicated by notification of the `EV_ADD_ALIAS_FAILED` event.

Synopsis

```
ACU_ERR ipt_query_alias(QUERY_ALIAS_XPARMS* query_aliasp);

   typedef struct query_alias_xparms
   {
     ACU_ULONG             size;
     ACU_UINT              protocol;                  /* IN */
     ACU_UINT              registration_handle;       /* IN */
     ACU_CHAR              alias[MAXALIAS];           /* OUT */
     ACU_INT               prefix;                    /* OUT */
     ACU_UINT              state;                     /* OUT */
     ACU_UINT              error;                     /* OUT */
   } QUERY_ALIAS_XPARMS;
```

Input Parameters

`ipt_query_alias()` takes a pointer, `query_aliasp`, to a structure, `QUERY_ALIAS_XPARMS`. The structure must be initialised before invoking the function.

protocol
The protocol to query. Currently only H.323 is supported so this must be set to S_H323.

registration_handle
Must contain a valid registration handle as generated by `ipt_add_alias()`. It is used to identify which registration you wish to query.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error. For SIP, `ERR_COMMAND` will be returned if the state of the registration is not `EV_ADD_ALIAS_FAILED`.

alias
H323 only.

Will contain the address of the alias we have just queried. It will be in the form of a URL.

Prefix
H.323 only.

Will identify if the alias we have just queried is a prefix.

0 – indicates that the alias is not a prefix

1 – indicates that the alias is a prefix

state
H.323 only.

Indicates whether the alias is registered/mapped with the registration system.

0 – indicates that the alias is not registered/mapped

1 – indicates that the alias is registered/mapped

error
For H.323: When an alias has been unregistered/unmapped, whether it be locally or remotely, an associated error code will be returned to provide a reason why.

For SIP: When a registration attempt has failed, an associated error code will be returned to provide a reason why.

5.13 Registration event notification

The registration functions merely send a request to an element on the network. This request may succeed, fail or never arrive. In order to notify the client application what has happened, an event is raised. Please note that these events are global and in a multiple application system all applications will be notified.

The following global events may be raised through the Call API:

`EV_ADD_ALIAS_SUCCEEDED`
Raised when a success response has been received from the registration system for an `ipt_add_alias()` request.

`EV_ADD_ALIAS_FAILED`
Raised when a failure response has been received from the registration system for an `ipt_add_alias()` request.

`EV_ALIAS_REMOVED`
Raised when for whatever reason an alias has been unregistered/unmapped from the registration system, be it locally or remotely.

On receipt of an event, you can inspect the context field of the global event structure to determine to which alias this event relates. The context field contains the registration handle that was assigned during `ipt_add_alias()`. A call can then be made to `ipt_query_alias()` to determine the state of the alias. Note: when using SIP, this should only be done when the reported event is `EV_ADD_ALIAS_FAILED`.

5.14 ipt_snapshot_registrations()

This function can be used to query the registration system to determine what aliases are currently registered.

NOTE

This function is not supported for SIP.

Synopsis

```
ACU_ERR ipt_snapshot_registrations(SNAPSHOT_REGISTRATIONS_XPARMS *snapshotp);

   typedef struct snapshot_registrations_xparms
   {
     ACU_ULONG             size;
     ACU_UINT              protocol;                    /* IN */
     ACU_UINT              count;                       /* OUT */
     ACU_UINT              handles[MAXREGISTRATIONS];   /* OUT */
} SNAPSHOT_REGISTRATIONS_XPARMS;
```

Input Parameters

`ipt_snapshot_registrations()` takes a pointer, `snapshotp`, to a structure, `SNAPSHOT_REGISTRATIONS_XPARMS`. The structure must be initialised before invoking the function.

protocol
The protocol to query. Currently only H.323 is supported so this must be set to S_H323.

Return Values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

count
The number of aliases registered.

handles
An array containing the handles of the aliases registered.

Appendix A: H.323 registration

This section includes guidance on H.323 registration with Aculab's H.323 products.

A.1 Adding aliases

For best results, it is advised to add aliases before calling `set_h323_gatekeeper`. This means that after the first RCF is received all of your aliases will be registered, and removes the requirement to send additional RCFs for each `add_alias` request.

Aliases are registered with the service and persist across all applications using the system. For on board H.323 this means that the port registrations are valid for all applications using that port.

A.2 Alias format

Aliases are defined in URI format: `<scheme>:<alias name>`

Valid schemes are:

Scheme	Meaning
"h323"	H.323 URI
"mailto"	Email address
"http"	URL
"h323id"	H.323 ID
"tel"	E.164 number

Alternatively, an IP address or a hostname may be provided. If no scheme is given, the system will try to "guess". If it is entirely numeric it will be assumed to be an E.164 number, if it is a dotted quad it will be assumed to be an IP address, if we can resolve it, it will be assumed to be a hostname. If all of these fail, an `ERR_PARM` will be returned.

A.3 Removing aliases and clearing the gatekeeper

When an application controlling registration exits, it should first remove its aliases, then clear the gatekeeper (if it set it) and delete any aliases it registered. Alternatively, an application should on start up deal with the currently active registration state, and/or aliases already present.

Removing an alias does not delete it from the system. This allows the application to check its status if any problems develop during un-registration. Aliases, which are no longer used, should be deleted. This is somewhat analogous to `call_disconnect` and `call_release`, there is a separation between un-registering and deleting associated resources.

Clearing the gatekeeper does not delete aliases, although it will remove all registered aliases. After clearing a gatekeeper, it is good practise to delete any aliases, which are no longer necessary. If they are not deleted, they will be re-registered the next time `set_h323_gatekeeper` is called. This is to facilitate manual gatekeeper failovers.

A.4 General points to note

It is possible to get a list of all current aliases by using `snapshot_registrations`. If you do not know for sure if another application has been storing aliases, or if you are unsure if a previous run of your application cleaned up correctly, this can be used to discover which aliases currently exist in the system.

We generally try to cope with applications trying to re-add aliases that are already present, and return the same handle.

If you are rapidly clearing and then setting the H.323 gatekeeper then it is required to wait for the un-registration to complete before calling `set_h323_gatekeeper`.

Contact us

Phone

+44 (0)1908 273800 (UK)
+1(781) 352 3550 (USA)

Email

Info@aculab.com

Sales@aculab.com

Support@aculab.com

Socials

Certificate number IS 722024
ISO 27001:2013

Certificate number FS722030
ISO 9001:2015