WENZEL ELECTRONIC::SYSTEMS

Scope:

Version: see release note	Responsible: Wenzel	Doc.-Title: PAS Control
Doc.-ID.: see release note		

PAS Control

Interface Specification

Change Index

Release	Date	Name	Description
See release note	See release note	TZ, SG	First Edition

Table of Contents

Abbreviations / Term	Description
PAS	A Public Address System is an electronic system comprising microphones, amplifiers, loudspeakers, and related equipment.
PA	In this context the sound reinforcement edge device in a PAS.
OCS	Operator Call Station, that is usually installed in OCCs or BOCCs for manual (live) and automated announcements.
OCC	Operation and Control Center.
BOCC	Backup Operation and Control Center.
VoIP Server	Central server network element in a Wenzel Elektronik PAS, combining call switching, registrar, proxy, database, monitoring and device fault and performance management.

1 Introduction

This document specifies an interface for operators and third parties to remote control and monitor public address systems (PAS) of Wenzel Elektronik GmbH and issue manual and automated announcements to these systems.

This document is intended to continue to stimulate discussion and serve as a basis for suggestions for improvement that can be incorporated into future levels.

1.1 Abstract

This document describes an interface to public announcement systems of Wenzel Elektronik GmbH, which defines the communication between the technical devices of the loudspeaker systems at remote locations and the devices in the control centre or other central equipment. It includes centralized voice switching, voice transport and monitoring of operating parameters.

1.2 Scope

The document describes an IP-based interface, the first layer of the Internet protocol family independent of the transmission medium, in the application area of sound reinforcement at remote locations.

This document does not cover local call stations, central operator call stations (OCS), data hubs for automated announcements and terminals and servers for system performance and fault management.

2 System Structure and Terms

2.1 PAS

The public address system consists of a sound reinforcement system with IP-based endpoints, control logics, amplifiers and loudspeakers organized in loudspeaker circuits, the remote call stations and the central equipment.

Within this document, PA refers to the total set of these units as a communication subscriber that serves as a communication sink for a set of speaker circuits and maintains and reports status information for the communication, as well as subcomponents. Each sink is referred to as a target in the document.

If there are several PAs and several loudspeaker circuits at a remote location, each loudspeaker circuit is assigned to a PA.

2.2 Operator Call Stations – OCS

The OSC typically consists of a PC on which the application call station is operated, screen and input devices of the announcer. This equipment is out of the scope of this document.

2.3 Call Station

A call station is a remote announcer unit with intercom in the area of remote location. It is also referred to as a SIP-connected pushbutton station.

This equipment is out of the scope of this document.

2.4 VoIP Server

The VoIP Server coordinates the operation of the PAs, receives announcements from the operator call stations or announcement data hubs.

It may be deployed redundantly.

The main tasks in the operational context is:

- Registrar for SIP
- Monitoring endpoint for SIP
- Monitoring SNMP Client
- Monitoring SNMP server for notification reception (trap)
- Repository for audio centralized stored files.

The VoIP server is implemented on virtual machines and might be delivered pre-installed on server hardware in a redundant or non redundant configuration.

2.5 Network

The PAS network designates the IPv4 level of technical communication of all PAS components. The PAS network might be secured externally by firewalls.

Each component of the PAS network is assigned an IPv4 address for this purpose, can be reached via this address, and can send IPv4 packets from this address. It must also be able to handle routing and store the default route and gateway.

The PAS network might be divided into two areas, the remote locations and the central part, which may be separated by IPv4 subnets.

All network elements are participants of the PAS network on IPv4 level. The participants of the PAS network only use standardized proven IPv4 protocols. Address translation (NAT) between the IPv4 networks of the PAS network is not supported.

2.6 Intentionally empty

2.7 PAS Control Centre

The PAS control centre may consists of one or more data hubs, management consoles and operator call stations.

It also may host the VoIP server (see 2.4), that provides individual services, such as SNMP, monitoring, SIP registrar.

If in the following specification the control centre is referred to in a functional context, such as "the control centre polls PAs", this describes the communication in the PAS network, it leaves open by which device types within the PAS control centre the described communication is realized.

2.8 Intentionally empty

2.9 Automated Announcement Control

The announcement controller is not a Wenzel deliverable and has to be provided by the customer. It is everything located in the PAS control centre excluding the VoIP server. It shall have the following minium functionality:

- an optional certificate manager for updating the security certificates of this controller, in case the PA system shall have the maximum security measures applied,
- a download service that provides playout-ready audio files for download by the PAs.

2.10 Range of action of the IP interface

The set of rules described in this document operates at IP level in the PAS network between the PAS control centre and the remote locations. Since PAs are primarily used as devices in the area of remote locations, the document primarily describes communication to and from the PA.

2.11 Application Levels

2.11.1 SIP

All participants of the PA network that talk to each other directly (transport level) or indirectly (switching level) via SIP are referred to as SIP endpoints, also referred to as just endpoints for short.

Each SIP endpoint shall register with the registrar in the VoIP server and have all calls switched through the registrar.

The SIP protocol itself is not classified as security relevant.

2.11.2 SNMP

All participants of the PAS network can be queried by further customer-specific participants of this network via SNMP. The PAs report only intended destinations to these participants via Notify (TRAP). SNMPv3 TSM or SNMPv2 is used.

2.12 Approved IP Protocols

Only the following protocols may be used:

Protocol/port	Identifier	Usage
UDP/161	SNMPv2	Technical and operational Monitoring
UDP/162	SNMPv2	Notification for SNMP
TCP/10161	SNMPv3 TLS	Technical and operational Monitoring
TCP/10162	SNMPv3 TLS	Notification for SNMP
UDP/5060	SIP/2.0	Operator
TCP/5061	SIP/2.0 TLS	Operator
UDP/5000-6000	RTP	Audio in connection with SIP
UDP/6001-6500	SRTP	Audio in connection with SIP TLS
ICMP	PING	Network monitoring
UDP/53	DNS	ELA domain name service
UDP/514	SYSLOG	Logging
UDP/6514	SYSLOG TLS	Logging
UDP/123	NTP	Time synchronization
TCP/443	HTTPS TLS	Language audio files, certificates, configuration upload
TCP/80	HTTP	Certificate Challenge, temporary
TCP/22	SSH	Administration, if necessary language audio files

The RFC or protocol specification for implementing the named protocols are included with the references.

2.13 PA ID and Hostname

Each PA has a globally unique ID according to RFC 4122. This is formed as UUID 5 via NAMESPACE_DNS using the host name. The host name is also relevant for certificates and endpoints of communication services.

2.14 Access Control

All devices in the responsibility of Wenzel Elektronik optionally provide access control at IP packet level (firewall).

Only defined destinations and services might be allowed (`ACCEPT`) or dropped (`DROP`) for communication (send and receive).

It is recommended to keep records (`IPSETS`) for the definition of groups of targets on the one hand and local ports on the other hand.

Flat-rate supported protocols for all are:

- SSH
- ICMP
- ARP

As part of the deployment scenario, access to encrypted and unencrypted services, as well as the ability to communicate with selected targets, are enabled or disabled at the service level.

2.15 Encryption

2.15.1 Transportation

All communication between the PAS network elements and its communication partners can optionally be encrypted. The following approved hash algorithms and key lengths shall be provided as a minimum:

Hash:	SHA-2 group and SHA3
Key:	DH with 2048 bits or longer
Higher Encryption:	AES128 or longer

TLS 1.3 or DTLS based on TLS 1.3 and SSH are permitted for service encryption.

Older communication partners of the PAS shall be downward compatible up to TLS 1.2.

All devices also optionally support enforcement of TLS 1.2 without Perfect Forward Security in case of fault analyses. For this purpose, the security level of all affected devices is lowered at the beginning of the analysis phase and increased again after its end.

2.15.2 Storage of Keys and Certificates

If encryption is optionally used, than generated private keys, keys shared in advance and certificates received that are used for communication in the network are regarded as data requiring protection. They are stored in encrypted form.

2.15.3 Key generation

All network elements in the PAS shall be able to generate private keys and certificate requests

(X.509) if encryption shall be used.

2.15.4 Certificate interface

If encryption as described is used, the customer shall provide a certificate service (CA) according to ACMEv2 (RFC8555). All services that can use X.509 certificates must use certificates issued by that CA, except for this:

- SSH
- RTP
- DNS
- NTP
- ICMP
- SYSLOG (if transmitting for debugging or nonsensitive data)

Certificates are issued for a group of services. For this purpose, wildcard capable common names can be used to address several services in one certificate. Services are qualified as host names in the PA network.

Extensions such as SIP EKU are not supported.

PA and PA managers use HTTP Challenging to obtain certificates. PA clients are implemented for certificate reference differentiated by interface type:

- Automatic announcement control
- Network Access Control via EAP

Multiple services may share a common certificate.

2.16 Key exchange in Secure RTP

SRTP is only used in connection with SIP. The keys are exchanged within the session description of the SIP communication, as defined in RFC 4568.

No separate key exchange is provided (compare DTLS implementation for SRTP).

The key exchange may only take place via SIP 2.0 TLS.

As soon as SIP can be operated with TLS (both endpoints support this), SRTP might optionally be used. Mixed operation, like encryption of SIP but unencrypted RTP is not provided.

2.17 SNMP Compatibility

In the following, only SNMPv3 with Transport Security Model is referred to. However, it is assumed that there is dual operation of SNMPv3 and SNMPv2, with users of v2 and v3 using the same views. Supported access types are

- `NoAuthNoPriv`: for older SNMPv2 devices, unencrypted where username becomes community

- `AuthPriv` for SNMPv3 devices

Access via `NoAuthNoPriv` must be restricted via access control to an IP address list. This can be done either by SNMP implementation or by firewall using the version in the message header, if not used on a separate port.

Accesses via SNMPv3 are project specific, here the PA is be able to store other users, passwords and privacy keys, as well as hash algorithm and encryption method per communication partner (`trap` or `get`).

Supported password hashing algorithms are: SHA-2.

Supported transport ciphers are: AES128.

TLS 1.3 and DTLS 1.3 are supported as encryption methods.

2.18 SNMP Trap vs Notify

In the comparison of PDU formats between SNMPv1 and SNMPv2, two different PDU formats are specified for the former, one for TRAPs and one for all other operations (`GET`, `SET`, ...). In SNMPv2 the PDU format is identical for TRAPs and all other operations (without `getbulk`).

To standardize the PDU format of SNMPv1 traps, the concept of notifications was introduced in SNMPv2 and continued in SNMPv3.

Since PAS communication also takes place on SNMPv2, this document also makes use of the term notification.

The differences are:

- the macro used to set the traps
- trap PDU contains the agent address, where the notification PDU contains error status and error index.
- trap PDU contains information about generic and specific traps, while notification uses an OID.

This specification is omitted with the use of SNMPv3.

3 SIP

SIP is used to make any type of realtime announcements, regardless of the source being a microphone or a recorded or synthesized message. It is also possible to trigger certain other configured actions on the PA via SIP calls using predefined SIP URIs.

3.1 Addressing

Subscribers are identified by call numbers following E.164. Subscribers can be both sources (e.g. call stations or data hubs) and sinks (destinations in the PA).

OCS and local stations are assigned a fixed source number. Each SIP endpoint operated on it can receive exactly one call number.

PAs are assigned a set of destination numbers per loudspeaker circuit or group of loudspeaker circuits for dedicated control of the output systems. This is done via certain configuration entries on the specific PA.

The PA supports SIP trunking. Speaker circuits and groups of speaker circuits can coexist.

3.2 Switching

All call stations and destinations are equally registered to the registrar, and only call stations may call destinations and subscribe to status information for destinations.

The registrar switches to destinations on the basis of its stored (in the VoIP server) call distribution plan. Destinations are operated as a trunk.

The registrar is optionally operated in a standby-redundant manner, so that registration always takes place on the active of the two instances.

3.3 SIP Call Setup

The call setup is always done via the registrar by URI. The URI consists of a destination number and a set of control characters defined as follows.

Hash ('#') and asterisk ('*') are used to activate special functions in the PA per target. The hash is used as a separator of function commands and the asterisk as a separator of parameters of a list within the function.

```
SIP URI = "sip:" destination number *('#' function)
```

where

```
function = command *('*' parameter)
```

In the following, the grammar is explained with examples.

If a pre-chime is to be played during the live announcement, the following URI shall be send:

```
sip:55-03925-0248#C
```

A fictitious example of using a list of different parametrized functions is:

```
sip:55-03925-0248#XX*77*99#YY*22
```

this would activate function `XX` with parameters `77` and `99` and function `YY` with parameter `22`

3.4 Supported SIP functions

The currently only supported SIP functions is the playback of recorded audio files, stored in the VoIP server, to the PA, Example:

```
sip:55-03925-0248#22
```

Playback of audio file #22 to the number 55-03925-0248.

3.5 Post Dialling

Post dialling by DTMF tones or call number extensions that deviate from the "SIP call setup" defined here are not supported.

3.6 Locally Stored Announcements

Announcements stored locally in the PA at the remote location are triggered by a SIP call to the pre-configured URI.

3.7 Central Recorded Announcements

Recorded announcements from the OCS or a data hub or other central equipment are treated like live announcements from PA's point of view.

3.8 Group Call

A group call / announcement allows dialling a set of destinations on different PAs or a set of loudspeaker circuits or loudspeaker groups on one PA, or a combination of both.

In such a case, the numbers are separated by an ampersand '`&`'.

Example:

```
sip:55-03925-0101&55-03926-0201&55-03927-0101
```

3.9 Call Number Scheme

The phone number scheme can be freely selected.

3.10 Voice Transport

3.10.1 Audio codecs used and priorities

The following audio codecs are intended for use. Codecs with a higher priority are to be used preferentially or to be provided as higher priority in the endpoints.

Highest prioritization in this case is the value 1.

Priority	Codec	Bandwidth per channel
1	OPUS	32 kbit/s
2	L16 PCM	256 kbit/s
3	G.722.1	64 kbit/s
4	G.711a	64 kbit/s
5	G.711u	64 kbit/s

3.10.2 3.10.2 Quality of Service

All SIP endpoints set the following DSCP values for outgoing packets:

Application	DSCP
Call Signaling (SIP)	`CS5`
Audio (RTP)	`EF`

4 Audio file service

Audio files can transferred to the PA via an HTTPS REST API call to the WeView WEB configuration interface, if they can’t be transferred locally during the configuration process.

5 Monitoring

5.1 RTP audio stream

RTP packet loss is continuously monitored from the beginning of the announcement. As soon as the threshold of 5% packet loss is exceeded, the connection is terminated by the detecting SIP endpoint. A negative SNMP trap notification is issued in case.

5.2 SIP Call Channels

If the maximum number of connections per PA is exceeded, the PA will not accept any further calls.

5.3 SIP Keep-Alive

Each SIP endpoint performs a SIP `keep-alive`.

5.4 ICMP test

The VoIP server will ping each remote located PA in a defined interval. Each PA will ping the PA VoIP server.

If a timeout occurs three times in succession, the pinged target is considered to be down or offline.

Failures are noted in the device’ system log.

5.5 SNMP Communication

The control centre network equipment can regularly read out the status of the PAs via SNMP query (polling). For this purpose, the PAs support the so called ELISAIII-MIB.

Each status query is based on a typed list of elements (table). Each element (table entry) contains a readout index, an identifier unique to the PA, and a set of properties; but at least one status.

A PA control panel may act as an SNMP receiver for notifications from the PA. Other devices may also be provided as SNMP receivers.

The notification recipients are stored on the PA. The PA sends a notification to all designated recipients immediately after a status is changed.

6 SNMP

6.1 Query Concept

The PAs support the ELISAIII-MIB and the MIB-II.

6.2 ELISAIII-MIB

The ELISAIII-MIB is designed as a flat structured set of tables. All traps are based on updating table entries. This allows the querier to carry the changes of entries of the tables as a query (`Poll`) or notification (`Notify`) within a dual query system.

All tables have a running index (suffixed with `-Index`), which is used for the unique assignment of entries. It is not assigned any further content to this entry.

All tables have a minimum structure that includes the running index, the functional unique identifier of the entry, and a value (`Value`) or status (`Status`).

The unique identifier may only have characters in the range `[a-zA-Z0-9_:]`.

6.3 ELISAIII-MIB objects

6.3.1 E3MasterParams

Global system parameter.

Identifier	Function
`masterDeviceName`	Name of elisa master
`systemName`	Name of elisa system
`firmwareVersion`	Version string describing Elisa Firmware
`configurationDate`	Date of last configuration modification

6.3.2 Tables for functional groups

All different function of the PA are kept per type in a separate table with the columns `Index`, `Name` and contexed based enties. The main function is realized by changing the entries, because then a trap is sent. A high-frequency query of these tables is not provided.

6.3.2.1 E3DeviceTable

A list of device entries.

Assignment per circuit	Meaning
deviceUuid	UUID of the device
deviceType	Device type
deviceTemperature	Chassis temperature in degree
devicePowerSupplyDC	Device DC power-supply: ok/fault
devicePowerSupplyAC	Device AC power-supply: ok/fault
deviceLocalInterconnectionStatus	Internal connection status to other ela devices: down/up/upRedundant

6.3.2.2 E3AmplifierStatusTable

A list of amplifier entries.

Assignment per circuit	Meaning
amplifierStatus	Amplifier error status: ok/fault

6.3.2.3 E3OutStatusTable

Describes the failure state of each output line.

Assignment per circuit	Meaning
outShortCircuit	A short circuit is actual occurred: ok/fault
outImpedanceFault	An impedance fault is actual occurred. The impedance is out of the specified impedance window: ok/fault
outEarthFault	An earth fault is actual occurred: ok/fault
outlineBreak	A line break is actual occurred: ok/fault

6.3.2.4 E3SinkStatusTable

A list of sink entries.

Assignment per circuit	Meaning
sinkStoplightStatus	Sink error state: green(work correct, inclusive redundance if configured), yellow(redundance loss, but it still work), red(drop out)
sinkLastAck	Last acknowlege value
sinkNumber	sink VoIP number
sinkUsage	Sink usage state: idle (currently not in use) normal (busy by normal announcement) prealarm (busy by prealarm) alarm (busy by alarm)
sinkCallerNumber	sink VoIP caller number if available

6.3.2.5 E3ExtFailureStatusTable

A list of external failure entries.

Assignment per circuit	Meaning
`extFailureStatus`	Depending Error state: ok/fault

6.3.2.6 E3AudioInStatusTable

A list of audio input entries.

Assignment per circuit	Meaning
`audioInPilotToneStatus`	Last pilot-tone failure status: ok/fault

6.3.2.7 E3AmbientStatusTable

A list of ambient sound sensor entries.

Assignment per circuit	Meaning
`ambientStatus`	Microphone error: ok/fault

6.3.2.8 E3GpInTable

A list of general purpose input (GPI) entries. The General Purpose Input interfaces of the PAs are mapped to signal level (`High` / `Low`) via SNMP.

Assignment per circuit	Meaning
`gpInValue`	Logical value of the GPI: low/high

6.3.2.9 E3GpOutTable

A list of general purpose output (GPO) entries. The General Purpose Output interfaces of the PAs are mapped to signal level (`High` / `Low`) via SNMP.

Assignment per circuit	Meaning
`gpOutValue`	Logical value of the GPI: low/high

6.3.2.10 E3VoIPReceiptTable

A list of all VoIP acknowledgents for all destinations and sources.

Assignment per circuit	Meaning
`receiptCounter`	Every new receipt value increments the counter
`receiptSrcNumber`	VoIP announcement source number
`receiptDstNumber`	VoIP announcement destination number

6.4 Text Conventions

All status information is textually defined.

Status type	Assignment	Function
`sinkStoplightStatus`	`green(0)`	Function guaranteed, no errors present
	`yellow(1)`	Function guaranteed, error present
	`red(2)`	Function not guaranteed
`sinkUsage`	`idle(1)`	not in use
	`normal(2)`	busy by announcement
	`prealarm(3)`	busy by pre-alarm
	`alarm(4)`	busy by alarm
`errorStatus`	`ok(1)`	Normal operation
	`on fault(2)`	Fault
`GPIOStatus`	`low(1)`	Low signal applied
	`high(2)`	Present high signal
`deviceLocalInterconnectionStatus`	`up(1)`	Connection up (only single)
	`down(2)`	Connection down
	`upRedundant(3)`	Connection up redundant

7 Tutorial

***Note**: This tutorial is for guidance only. The source code shown does not claim to be complete or correct. As we only use standard protocols for status queries, control and audio transmission, the software packages suggested here are only to be understood as recommendations. Other open source or proprietary implementations available on the market can be used. However, we use the suggested implementations ourselves and therefore expect the highest compatibility.*

7.1 SIP

7.1.1 Introduction

The Session Initiation Protocol (SIP) is a signalling protocol used for initiating, maintaining, and terminating communication sessions that include voice, video and messaging applications. SIP is used in Internet telephony, in private IP telephone systems, as well as mobile phone calling over LTE (VoLTE).

The protocol defines the specific format of messages exchanged and the sequence of communications for cooperation of the participants. SIP is a text-based protocol, incorporating many elements of the Hypertext Transfer Protocol (HTTP) and the Simple Mail Transfer Protocol (SMTP). A call established with SIP may consist of multiple media streams, but no separate streams are required for applications, such as text messaging, that exchange data as payload in the SIP message.

SIP **works in conjunction with several other protocols** that specify and carry the session media.

1. Most commonly, media type and parameter negotiation and media setup are performed with the Session Description Protocol (SDP), which is carried as payload in SIP messages.
2. SIP is designed to be independent of the underlying transport layer protocol and can be used with the User Datagram Protocol (UDP), the Transmission Control Protocol (TCP),
3. and the Stream Control Transmission Protocol (SCTP).
4. For secure transmissions of SIP messages over insecure network links, the protocol may be encrypted with Transport Layer Security (TLS).
5. Most important: For the transmission of media streams (voice, video) the SDP payload carried in SIP messages typically employs the Real-time Transport Protocol (**RTP**) or the Secure Real-time Transport Protocol (SRTP).

You do not have to worry about all the details behind such protocols, as modern libraries and frameworks make most of the job for you.

7.1.2 Framework and Resources

The SIP tutorial is based on the OpenSource framework PJSIP: `https://github.com/pjsip/pjproject`.

Through this link a lot more information and the complete documentation is available.

PJSIP is available under under both General Public License (GPL) version 2 or later and a proprietary license that can be arranged with the developer of PJSIP. In practical sense, this means:

- If you can release your software source code, that uses PJSIP, use it under freely GPL. But please double check `https://docs.pjsip.org/en/latest/overview/license.html` for OSS license compatibility with GPL.
- Alternatively, if you are unable to release your application as Open Source software, you may arrange alternative licensing with the developer of PJSIP. Just send your inquiry to `licensing@pjsip.org` to discuss this option.

PJSIP may include third party software in its source code distribution. Please make sure that you comply with the licensing term of each software.

7.1.3 Prepare the Toolchain

We can't discuss in this tutorial all the functionality, that the used library provides. Please consider the documentation: `https://docs.pjsip.org/en/latest/`. There are also tremendous resources in the WEB available, where use cases and issues are discussed.

It is recommended that you first study the introductory chapters in the documentation.

The first step is to compile the library and the sample applications (part of the PJSIP source) for your specific operating system and used compiler as described in the documentation.

7.1.4 Make a call to ELISA III-IP

To play audio data to ELISA III-IP the SIP + RTP protocols are required, both provided by PJSIP.

The sample program shows the basic for usage and is based on the high level PJSUA C-interface. There is also a C++-interface and other bindings available. In addition, it is possible to use the low level interface. However, you might not need this in order to use the full functionality of the ELISA III-IP API described above.

The steps in the source code below are:

1. Initialize the library with logging feature and 2 important callbacks.
2. Create a wave file player for audio data with specific file name.
3. Start the library.
4. Add local account for call (mind, that we use SIP in a so called direct address mode, there is no registrar necessary).
5. Make the call itself,
6. When call was successful, all the stuff is running in background.
7. Once the wave file is played an automatic hangup for the connection is done.

Watch out for this line in the code:

```
static constexpr const char *CALL_ADDRESS = "sip:elisa@192.168.70.15";
```

The IP-address of the device is expected to be `192.168.70.15` and the URI phone number `elisa` needed to be configured in the device as shown in the configuration manual of ELISA III-IP. It is expected, that a wave audio file `test.wav` exists in the path.

```
#include <pjsua-lib/pjsua.h>

#define THIS_FILE "playdemo"

// make it simple

static constexpr const char *CALL_ADDRESS = "sip:elisa@192.168.70.15";
static constexpr const char *waveFileName = "test.wav";

bool fUseAudioDevice = false;
pjsua_conf_port_id call_conf_slot = PJSUA_INVALID_ID;
pjsua_player_id wav_id = PJSUA_INVALID_ID;
unsigned play_options = PJMEDIA_FILE_NO_LOOP;
int wav_port = PJSUA_INVALID_ID;
pjmedia_port *port = nullptr;
pj_timer_entry auto_hangup_timer;

// Auto hangup timer callback
static void hangupTimeoutCallback(pj_timer_heap_t *timer_heap,
                                  struct pj_timer_entry *entry)
{
    puts("## Hangup timer expired, HANGUP call");

    PJ_UNUSED_ARG(timer_heap);
    PJ_UNUSED_ARG(entry);

    auto_hangup_timer.id = 0;
    pjsua_call_hangup_all();
}

/* Playfile done notification, set timer to hangup calls */
static void onPlayFileDone(pjmedia_port *port, void *usr_data)
{
    pj_time_val delay;

    PJ_UNUSED_ARG(port);
    PJ_UNUSED_ARG(usr_data);

    /* Just rewind WAV when it is played outside of call */
    if (pjsua_call_get_count() == 0)
    {
        pjsua_player_set_pos(wav_id, 0);
    }

    /* Timer is already active */
    if (auto_hangup_timer.id == 1)
```

```
        return;

    auto_hangup_timer.id = 1;
    delay.sec = 0;
    delay.msec = 200; /* Give 200 ms before hangup */
    pjsip_endpt_schedule_timer(pjsua_get_pjsip_endpt(), &auto_hangup_timer, &delay);
}

/* Callback called by the library when call's state has changed */
static void onCallState(pjsua_call_id call_id, pjsip_event *e)
{
    pjsua_call_info ci;

    PJ_UNUSED_ARG(e);

    pjsua_call_get_info(call_id, &ci);
    PJ_LOG(3, (THIS_FILE, "Call %d state=%.*s", call_id,
               (int)ci.state_text.slen,
               ci.state_text.ptr));

    switch (ci.state)
    {
    case PJSIP_INV_STATE_CONFIRMED:
        puts("## Call Connected, start transfer ##");
        break;

    case PJSIP_INV_STATE_DISCONNECTED:
        puts("## Call disconnected ##");
        break;

    default:
        break;
    }
}

/* Callback called by the library when call's media state has changed */
static void onCallMediaState(pjsua_call_id call_id)
{
    pjsua_call_info ci;

    pjsua_call_get_info(call_id, &ci);

    if (ci.media_status == PJSUA_CALL_MEDIA_ACTIVE)
    {
        puts("## CALL MEDIA ACTIVE");

        if (fUseAudioDevice)
        {
            pjsua_conf_connect(ci.conf_slot, 0);
            pjsua_conf_connect(0, ci.conf_slot);
        }
        else
        {
            // When media is active, connect call to our wave player data pump.

            call_conf_slot = ci.media[0].stream.aud.conf_slot;

            /* Make sure conf slot is valid (e.g: media dir is not "inactive") */
            if (call_conf_slot == PJSUA_INVALID_ID)
            {
                return;
            }

            pjsua_conf_connect(wav_port, call_conf_slot);
        }
    }
}

/* Display error and exit application */
static void errorExit(const char *title, pj_status_t status)
{
    pjsua_perror(THIS_FILE, title, status);
    pjsua_destroy();
    exit(1);
}

const pj_str_t *pjcstr(pj_str_t *str, const char *s)
{
    str->ptr = (char *)s;
    str->slen = s ? (pj_ssize_t)strlen(s) : 0;
    return str;
}

int main(int argc, char *argv[])
{
    pjsua_acc_id acc_id = 0;
    pj_status_t status = 0;

    //
    // Create pjsua layer first!
    //

    status = pjsua_create();
    if (status != PJ_SUCCESS)
        errorExit("Error in pjsua_create()", status);

    //
    // Init pjsua with basic callbacks
    //
    {
        pjsua_config cfg;
        pjsua_logging_config log_cfg;

        pjsua_config_default(&cfg);

        cfg.cb.on_call_media_state = &onCallMediaState;
        cfg.cb.on_call_state = &onCallState;

        pjsua_logging_config_default(&log_cfg);
        log_cfg.console_level = 4;

        status = pjsua_init(&cfg, &log_cfg, NULL);
        if (status != PJ_SUCCESS)
            errorExit("Error in pjsua_init()", status);
    }
    pj_str_t pjWaveFile;
    status = pjsua_player_create(pjcstr(&pjWaveFile, waveFileName), play_options,
&wav_id);
    if (status != PJ_SUCCESS)
        errorExit("Error opening wave file", status);

    wav_port = pjsua_player_get_conf_port(wav_id);

    //
    // autmatic hangup when wave file is played
    //

    pjsua_player_get_port(wav_id, &port);
    status = pjmedia_wav_player_set_eof_cb2(port, NULL, &onPlayFileDone);

    pj_timer_entry_init(&auto_hangup_timer, 0, NULL, &hangupTimeoutCallback);

    int transportId = 0;

    //
    // Add UDP transport.
    //
    {
        pjsua_transport_config cfg;

        pjsua_transport_config_default(&cfg);
        cfg.port = 5060;
        status = pjsua_transport_create(PJSIP_TRANSPORT_UDP, &cfg, &transportId);
        if (status != PJ_SUCCESS)
            errorExit("Error creating transport", status);
    }

    //
    // Initialization is done, now start pjsua
    //

    status = pjsua_start();
    if (status != PJ_SUCCESS)
        errorExit("Error starting pjsua", status);

    //
    // Create local account, used for call
    //
    status = pjsua_acc_add_local(transportId, PJ_TRUE, &acc_id);

    //
    // Make the SIP call itself
    //

    pj_str_t uri;
    status = pjsua_call_make_call(acc_id, pjcstr(&uri, CALL_ADDRESS), 0, NULL, NULL,
NULL);
    if (status != PJ_SUCCESS)
        errorExit("Error making call", status);

    //
    // Now all other stuff is running in background
    //
    puts("Call pending, all stuff in background");

    //
    // Wait until user press "q" to quit.
    //

    for (;;)
    {
        char option[10];

        puts("Press 'h' to hangup all calls, 'q' to quit");
        if (fgets(option, sizeof(option), stdin) == NULL)
        {
            puts("EOF while reading stdin, will quit now..");
            break;
        }

        if (option[0] == 'q')
            break;

        if (option[0] == 'h')
            pjsua_call_hangup_all();
    }

    if (wav_id != PJSUA_INVALID_ID)
    {
        pjsua_player_destroy(wav_id);
    }

    // Destroy pjsua
    pjsua_destroy();

    return 0;
}
```

7.2 SNMP

7.2.1 Introduction

Simple Network Management Protocol (SNMP) is an Internet Standard protocol for collecting and organizing information about managed devices on IP networks and for modifying that information to change device behaviour. Devices that typically support SNMP include cable modems, routers, switches, servers, workstations, printers, much more and the ELISA III-IP.

SNMP is widely used in network management for network monitoring. SNMP exposes management data in the form of variables on the managed systems organized in a management information base (MIB), which describes the system status and configuration. These variables can then be remotely queried (and, in some circumstances, manipulated) by managing applications.

Three significant versions of SNMP have been developed and deployed. SNMPv1 is the original version of the protocol. More recent versions, SNMPv2 and SNMPv3, feature improvements in performance, flexibility and security. ELISA III-IP supports SNMPv2 and later SNMPv3.

SNMP is a component of the Internet Protocol Suite as defined by the Internet Engineering Task Force (IETF). It consists of a set of standards for network management, including an application layer protocol, a database schema, and a set of data objects.

Note*: The SNMP API of ELISA III-IP is not intended to modify or configure or control the device only reading is supported.*

Each ELISA III-IP provides an SNMP agent that can be polled and sends traps to configurable targets.

The agent speaks SNMPv2 (later SNMPv3) and supports the MIB2 as well as the Wenzel-specific ELISAIII-MIB.

When status changes occur, the agent sends traps according to the MIB.

A SNMP manager is an application for reading/polling SNMP MIBs and receiving traps.

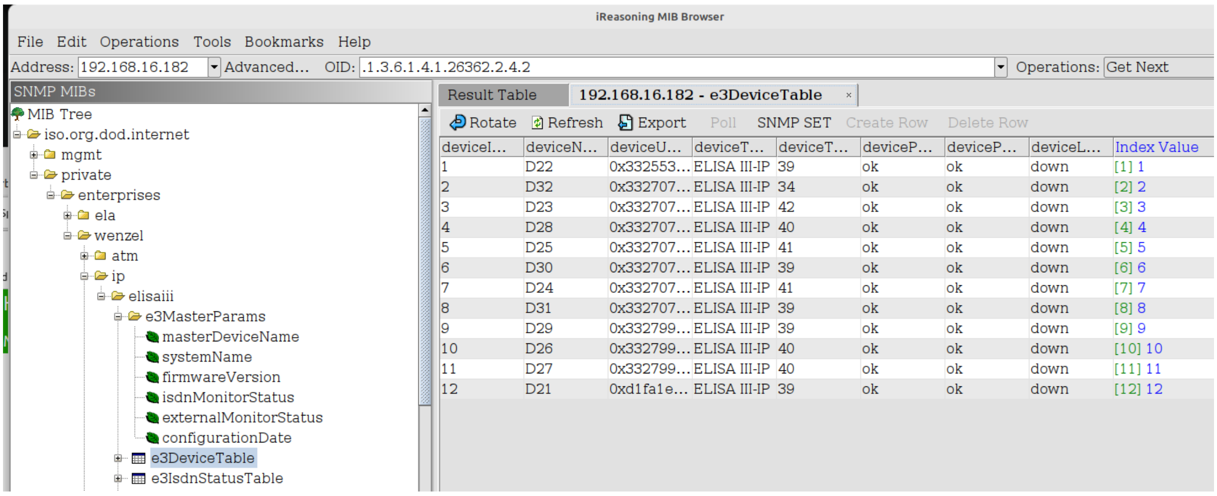
7.2.2 Framework and Resources

7.2.2.1 MIB Browser

A quick and easy way to familiarize yourself with the MIB2 and ELISAIII-MIB is to use the "MIB Browser Free Personal Edition" of the ireasoning MIB browser `https://www.ireasoning.com/download.shtml`.

This allows the MIB to be read out and there is also a trap receiver.

The ELISAIII-MIB and all MIBs included in it must be loaded once, SNMPv2 must be selected without password and then SNMP can be accessed via the ELISA III-IP LAN interface.



Address: 192.168.16.182

OID: .1.3.6.1.4.1.26362.2.4.2

Operations: Get Next

192.168.16.182 - e3DeviceTable

deviceI...	deviceN...	deviceU...	deviceT...	deviceT...	deviceP...	deviceP...	deviceL...	Index Value
1	D22	0x332553...	ELISA III-IP	39	ok	ok	down	[1] 1
2	D32	0x332707...	ELISA III-IP	34	ok	ok	down	[2] 2
3	D23	0x332707...	ELISA III-IP	42	ok	ok	down	[3] 3
4	D28	0x332707...	ELISA III-IP	40	ok	ok	down	[4] 4
5	D25	0x332707...	ELISA III-IP	41	ok	ok	down	[5] 5
6	D30	0x332707...	ELISA III-IP	39	ok	ok	down	[6] 6
7	D24	0x332707...	ELISA III-IP	41	ok	ok	down	[7] 7
8	D31	0x332707...	ELISA III-IP	39	ok	ok	down	[8] 8
9	D29	0x332799...	ELISA III-IP	39	ok	ok	down	[9] 9
10	D26	0x332799...	ELISA III-IP	40	ok	ok	down	[10] 10
11	D27	0x332799...	ELISA III-IP	40	ok	ok	down	[11] 11
12	D21	0xd1fa1e...	ELISA III-IP	39	ok	ok	down	[12] 12

Figure 1: MIB browser example window currently browsing the ELISAIII-MIB

7.2.2.2 More access methods

If you want monitor ELISAIII-MIB data or evaluate it or process it further you can use this applications:

- an (command line) API (see 7.2.2.3)
- embed open source it to your application (see 7.2.2.4)
- a network monitoring software (see 7.2.2.5)

7.2.2.3 Net-SNMP

In the example below, Net-SNMP (`http://www.net-snmp.org`) is used to access the ELISAIII-MIB via the operating system command line. This tool must either be installed via OS package manager (Linux / MAC OS) or downloaded from the site above and compiled according to the documentation (e.g. Windows).

Under Debian Linux, the steps to compile after downloading would be:

```
cd <directory of the unpacket downloaded package>
./configure
make
sudo make install
```

Examples:

Command for reading an ELISAIII-MIB element with `snmpget`:

```
snmpget -v2c -c public 192.168.16.182 ELISAIII-MIB::masterDeviceName.0
```

result:

```
ELISAIII-MIB::masterDeviceName.0 = STRING: "Pinneberg"
```

Command for reading an ELISAIII-MIB table with `snmptable`:

```
snmptable -c public -v 2c 192.168.16.182 ELISAIII-MIB::e3VoIPReceiptTable
```

result:

```
SNMP table: ELISAIII-MIB::e3VoIPReceiptTable

 receiptIndex receiptValue receiptCounter         receiptSrcNumber         receiptDstNumber
            1           ok              0 "sip:8002@10.26.47.226" "sip:4101@10.26.47.226"
            2           ok              0 "sip:8002@10.26.47.226" "sip:4102@10.26.47.226"
            3           ok              0 "sip:8002@10.26.47.226" "sip:4103@10.26.47.226"
            4           ok              0 "sip:8002@10.26.47.226" "sip:4104@10.26.47.226"
            5           ok              0 "sip:8002@10.26.47.226" "sip:4105@10.26.47.226"
```

```
        6         ok         0       "sip:8002@10.26.47.226" "sip:4106@10.26.47.226"
        7         ok         0       "sip:8002@10.26.47.226" "sip:4107@10.26.47.226"
        8         ok         0       "sip:8002@10.26.47.226" "sip:4108@10.26.47.226"
        9         ok         0       "sip:8002@10.26.47.226" "sip:4109@10.26.47.226"

      115         ok         0 "sip:pycall-test@10.26.47.226" "sip:4101@10.26.47.226"
      116         ok         0 "sip:pycall-test@10.26.47.226" "sip:4102@10.26.47.226"
      117         ok         0 "sip:pycall-test@10.26.47.226" "sip:4103@10.26.47.226"

      153      fault        20     "sip:1411@192.168.16.226" "sip:4101@10.26.47.226"
      154      fault        20     "sip:1411@192.168.16.226" "sip:4102@10.26.47.226"
      155      fault        19     "sip:1411@192.168.16.226" "sip:4103@10.26.47.226"
      156      fault        20     "sip:1411@192.168.16.226" "sip:4104@10.26.47.226"
      157      fault        18     "sip:1411@192.168.16.226" "sip:4105@10.26.47.226"
      158         ok         0     "sip:1411@192.168.16.226" "sip:4106@10.26.47.226"
```

7.2.2.4 Embedding

To embed a SNMP manager into your application, the package Net-SNMP is also a good choice, the de-facto standard and open source.

The sources are written in C programming language and there are perl and python modules available.

Consult the simple source code of `snmpget`, `snmptable` and `snmptrap` for how to write embedded code and reuse the sources in this package.

7.2.2.5 Network monitoring software

There are ready-made software packages for monitoring network devices, e.g. Icinga `https://icinga.com/` or prometheus `https://prometheus.io/`

These can also handle SNMP and you can prepare SNMP data representatively without in-depth knowledge.

8 References

SIP

- [RFC 3261] SIP: Session Initiation Protocol
- [RFC 4235] SIP: Dialog Event Package
- [RFC 4028] SIP: Session Timers
- [RFC 4568] SDP: Security Descriptions for Media Streams

SNMP

- [RFC 3416] Simple Network Management Protocol
- [RFC 1213] Management Information Base for Network Management of TCP/IP-based internets: MIB-II
- [ELISAIII-MIB] Management Information Base for PAs

SNMP3

- [RFC 3410] Introduction and Applicability Statements for Internet Standard Management Framework
- [RFC 3411] An Architecture for Describing SNMP Management Frameworks
- [RFC 3412] Message Processing and Dispatching
- [RFC 3413] SNMP Applications
- [RFC 3414] User-based Security Model
- [RFC 3415] View-based Access Control Model
- [RFC 3416] Version 2 of SNMP Protocol Operations
- [RFC 3417] Transport Mappings
- [RFC 3584] Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework
- [RFC 3826] The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model
- [RFC 5343] Simple Network Management Protocol (SNMP) Context EngineID Discovery

DNS

- [RFC 1034] DOMAIN NAMES - CONCEPTS AND FACILITIES
- [RFC 1035] DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION

UUID

- [RFC 4122] A Universally Unique IDentifier (UUID) URN Namespace

NTP

- [RFC 5905] Network Time Protocol

HTTPS

- [RFC 2818] Hypertext Transfer Protocol Secure

PING

- [RFC 792] Internet Control Message Protocol

SYSLOG

- [RFC 5426] SYSLOG

SSH

- [RFC 4250] Secure Shell

CA

- [RFC 8555] Automatic Certificate Management Environment (ACME)

PEM

- [RFC 7468] Textual Encodings of PKIX, PKCS, and CMS Structures