aculab

ACULAB.COM

ProsodyS

User Guide

MAN1170

PROPRIETARY INFORMATION

Document Revision

Rev	Date	By	Detail
1.0	28.05.04	BC	First issue
1.1	19.06.04	BC	Updates following initial feedback
1.2	03.10.04	BC	Updates to reflect V6 software release
1.3	03.02.05	BC	Addition of conferencing and media point switching
1.4	02.03.05	BC	Additional conferencing and media point switching information
1.4.1	23.03.05	BC	Additional media point appendix added
1.4.2	01.06.05	BC	Minor updates.
2.0	02.03.06	BC	Updates to reflect re-architecting to RTP API
2.0.1	11.07.06	BC	Copy of version 2 flowchart and new codec support added
2.2.0	01.12.06	BC	Updates for new version, T.38, SRTP and new codec support
3.0.0B1	25.06.08	BC	Updated for Prosody S version 3.0
3.0.0B5	16.09.08	BC	ACT and AIT updates
3.0.0	10.11.08	BC	Minor updates
3.0.1	07.04.09	BC	Minor updates – document links fixed
3.0.2	07.07.09	BC	Minor updates
3.1.0B1	06.05.09	BC	Additions for video
3.1.0	23.10.09	BC	Additional codecs, updated licensing
3.2.0	01.04.10	BC	SRTP support removed
3.3.0	02.06.10	BC	SRTP plugins, modems added
3.4.0	10.04.14	PGD	Align 6.5, remove video references
3.5.1	22.01.15	PGD	Updated references, added SILK codec, IPv6
3.5.2	19.02.15	PGD	G729 plugin added

3.5.4	29.09.15	PGD	Align version with package
3.5.9	13.12.16	PGD	Add 6.7 supported platform info, align version with package, correct log location, extra note regarding plugins
3.5.10	06.03.17	PGD	Correct stand alone PS install instruction, extra step note in config section, mention syslog logging
3.5.10.1	07.03.17	PGD	Correct 2nd stand alone PS install instruction and date
3.5.10.2	11.04.17	PGD	Add note on how to add ProsodyS from command line
3.5.10.3	19.06.17	PGD	Remove obsolete caveats in speech replay/record section
3.5.16	2.8.17	PGD	Align with release 3.5.16
3.5.20	17.10.17	PGD	Align with release 3.5.20, Linux optional mem configuration
3.5.27	6.9.18	PGD	Align with release 3.5.27, extra config options
3.5.28	7.12.18	PGD	Align with release 3.5.28, Opus codec
3.5.33	29.11.19	PGD	Align with release 3.5.33, update O/S list, fix web links
3.6.2	26.7.22	PGD	Align with release 3.6.2
3.6.5	12.9.22	PGD	Align with release 3.6.5

CONTENTS

1 About this document

1.1 Purpose

This document provides guidance for users of **Aculab Prosody S** version 3.6.2 and later**.** It contains a brief quick-start guide for Prosody S with information on installation, configuration, maintenance, and application development.

1.2 Scope

This document provides information for developers of telephony applications, systems integrators and other users of IP telephony and speech processing.

1.3 How to use this document

First-time users of Prosody S who are already familiar with the Aculab call and speech APIs should go directly to section 3, Getting started, for a basic installation, configuration and operation description.

Users who require a more detailed understanding of how to develop applications for Prosody S should first read section 4, Using Prosody S.

1.4 References

The following Aculab publications are referenced by this document:

[1] Aculab Installation Tool (AIT)
[2] Aculab Configuration Tool (ACT)
[3] Aculab Accessing Prosody S virtual cards reference manual
[4] Aculab Resource Management API Guide
[5] Aculab Call Control API Guide
[6] Aculab Extended SIP API Guide
[7] Aculab SIP Programmer's Guide
[8] Aculab Prosody™ TiNG Documentation
[9] Aculab Prosody™ Generic API reference manual
[10] Aculab Prosody™ Speech processing API reference manual
[11] Aculab Prosody™ WAV file API reference manual
[12] Aculab Prosody™ High level play/record API reference manual
[13] Aculab Prosody™ RTP API reference manual
[14] Aculab Prosody™ High Level Conferencing API reference manual
[15] Aculab Prosody™ FMP API reference manual (T.38)
[16] Aculab Prosody™ FAX API Guide
[17] Aculab Telephony Software Installation Guide

Copies of Aculab documentation are available from the Aculab web site at:

https://www.aculab.com/help/documentation.

The following 3rd party publications are referenced by this document:

[18] How to configure a RHEL 7 or RHEL 8 system to be able to run programs requiring Real-Time Scheduling - https://access.redhat.com/articles/3696121

2 Introduction

Prosody S is a host-based solution used to develop telephony applications such as IVR, voicemail, fax servers, call centre and conferencing platforms using IP connectivity. It does not require any proprietary hardware and provides IP telephony connections that can be terminated on speech processing channels.

An application using Prosody S typically makes and receives IP calls via the Aculab call API. The Aculab SIP server interacts with the Prosody S server to establish IP media sessions through which the media streams are sent and received. It uses a standard PC network interface card (NIC) to send and receive the IP packets. Both IPv4 and IPv6 packets are supported. Incoming IP packets are processed to produce a reliable audio stream. Speech-processing channels are created on the host via the Aculab Prosody API to provide the ability to play/record data from/to memory or file, handle DTMF or perform conferencing functions. The speech-processing channel and IP session are software-switched together using the datafeed switching facilities in the Prosody API.

As far as the Prosody APIs are concerned, the Prosody S server appears as a single 'virtual' card providing speech processing resources. These APIs are used in the same way that they are used when controlling Aculab speech processing hardware resources such as an Aculab Prosody X card in a Prosody X chassis. Prosody S provides a large subset of features within the Prosody API.

The extended SIP API and TiNG RTP API allow applications to have direct control of IP calls and RTP endpoints. They also provide advanced control of SIP-specific features.

Remote Prosody S servers

Prosody S can be installed on a machine remote from the controlling application. As is the case for Prosody X cards in Prosody X chassis, multiple remote Prosody S servers can be controlled from a single application.

Coexistence with Aculab hardware

Prosody S can be installed on the same machine that controls one or more Prosody X chassis. An application can therefore provide TDM call connectivity from a Prosody X chassis while servicing IP calls via Prosody S.

Note, however, that there is no inherent data connectivity between a TDM trunks and Prosody S.

Only IP calls can be attached to Prosody S host-based media processing resources.

Plugins

ProsodyS is shipped as a core server plus a set of standard plugins adding extra functionality. The standard distribution of ProsodyS does not include support for secure RTP. However a plugin supporting this is available from Aculab as a separate AIT installable plugin. Contact Aculab support for more information. Additional plugins developed for specific customer needs are occasionally provided to those customers.

Platforms

ProsodyS can only be run on 64 bit platforms. Windows applications interacting with ProsodyS may be 32 bit or 64 bit. Linux applications must be 64 bit.

3 Getting Started

This section provides the basic information necessary to install and use Prosody S

3.1 Minimum system requirements

Network interface

Standard Ethernet NIC

CPU

Single processor

Multi processor systems will significantly improve performance

CPU – x86-64 (AMD64) architecture.

Use of opus codec requires SSE4.1 capable CPU

Memory

512 MB

Operating system

Microsoft Windows Server 2016 or later (64 bit O/S only)

Linux 3.x.x kernel or later (64 bit O/S only, glibc 2.17 or later)

Licensing

Making IP calls requires at least a single-channel Prosody S licence key.

Use of newly introduced features may require updated licence keys.

3.2 System configuration

In order to maximise the performance of a Prosody S installation, the hardware and operating system configuration should be carefully considered. Note that some aspects of system configuration may need to be addressed before Prosody S installation. See the section on Performance.

3.3 Installation

3.3.1 Windows Server 2016/2019 installation

The installation procedure for Prosody S under Windows Server 2016, or Windows Server 2019 includes the following stages:

1. Install the Windows distribution using the Aculab installation tool (AIT)
2. Optionally install the Prosody S server on any remote machines
3. Either run the Aculab configuration tool (ACT) to:
   - Add local (and remote) Prosody S server(s)
   - Install Prosody S licence keys
   - Select the required IP services (SIP)

   Or do the same from a command line

NOTE

You must have administrative privileges to install the software. As required, see your network administrator to set these up

CAUTION

Check that your system's date and time are correct before installation; if the date and time are incorrect the licence software may not validate the keys. Changing the date of the system after the licence keys have been loaded may also invalidate the keys

3.3.2 Linux installation

The installation procedure for Prosody S under Linux includes the following stages:

1. Install the Linux distribution using the Aculab installation tool (AIT)
2. Optionally install the Prosody S server on any remote machines
3. Check system configuration
4. Configure Aculab components and start the Aculab resource manager
5. Run the Aculab configuration tool (ACT) to:
   - Add local (and remote) Prosody S server(s)
   - Install Prosody S licence keys
   - Select the required IP services (SIP)

3.3.3 Alternative installation tools

The graphical versions of the Aculab installation applications are described above. For users who prefer text based tools or require applications that do not rely on a windowing system, alternative tools are available.

- For installing software a text only version of the AIT is provided
- For adding and configuring cards prosody_ip_card_mgr may be used
- To install licences the LicenceAdmin tool is available.
- To select required IP services, a suitable "voip_rm.cfg" file needs to be present in installation "cfg" subdirectory. See "Creating configuration files" in Telephony software installation guide [17] .

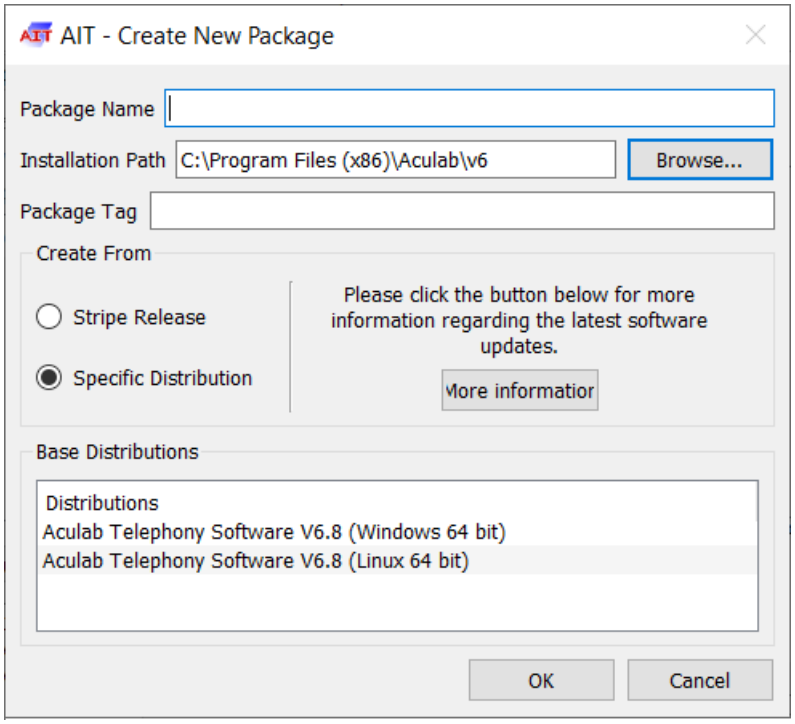
Some tools may not be distributed on all operating systems

3.4 Installing the Prosody S distribution using AIT

The Prosody S package is downloaded and installed using the Aculab Installation tool (AIT). A generic description of the AIT, including how to create self-extracting .zip files, is covered in the ***Aculab installation tool*** [1] document.

1. Run the Aculab installation tool (AIT) application.

Select *Connection Details* from the *Connection* menu. You will be presented with a dialogue window.



The dialogue window options available are:

Distributions – a list of supported operating systems and/or applications.

Package Name – a user defined unique identifier for the new package.

Installation Path – the installation directory for the distribution. The default can be edited if required or *Browse* can be selected to choose another location.

2. Make the required selections then select *OK* to complete the request. The AIT will read the current component information for the selected distribution and display the details in the main AIT window.

3. All components are included by default, this is the full install of all available Aculab software. It is not necessary to download all the Aculab software components in order to use Prosody S, although it is safe to do so. Users that wish to download and install only the components that are required for Prosody S should right-click on *Included Components* and select *Exclude* from the menu.

   This will move all components to the *Excluded Components* list. Select *Prosody S Software* from the *Excluded Components* list and use the right-click menu to *Include* just this component.

NOTE

Installing just the Prosody S component and its dependencies will give you a minimal installation that allows the use of Prosody S and SIP. Users wishing to use other features must download any additional components that may be required.

NOTE

SRTP, V.34 fax modem, and G.729 support are supplied as separate packages that should be installed in addition to the Prosody S package. The Prosody S package on its own does not support SRTP, V.34 fax modem, or G.729.

4. To install the Prosody S software, right click on the *Prosody S Software* entry (now in the *Included Components* list) then select *Install*, this will download and then install only the Prosody S component and its dependencies from the Aculab distribution server to your system.
5. When you are prompted with notification of component dependencies, select *Yes*.
6. Users are presented with the Aculab licence agreement and are required to decline or accept the agreement. If the agreement is declined the Aculab software will not be installed.
7. The output window will give feedback regarding the progress of the installation. Any errors that occur during installation will be reported here.
8. Once installation has completed successfully, close the AIT.

- Windows

  The installed Prosody S server will now be running as a service and you are now ready to configure the Prosody S server using the Aculab configuration tool (see 3.8 Running the Aculab configuration tool).

- Linux

  The Aculab components need to be configured for this system and the Aculab resource manager needs to be started prior to configuring the Prosody S server.

3.5 Optionally install the Prosody S server on remote machines

Prosody S can be installed and run on a machine with a minimum of other Aculab software installed and controlled from a remote application.

To install Prosody S in isolation, on remote machine just install ProsodyS and Licencing elements from AIT, create aculab.config file and set up security key on Windows, by running (with bin\amd64 directory in PATH):

```
ProsodySServ.exe -stop
ProsodySServ.exe -securitykey xxxx
ProsodySServ.exe -start
```

or on Linux, run by running (with bin64 directory in PATH)::

```
ProsodySServ -stop
ProsodySServ -securitykey xxxx
ProsodySServ -start
```

where xxxx is a unique alphanumeric string. This string must match that entered in the ACT Prosody S details page when adding the server on the controlling machine.

Next install licences for remote machine ProsodyS instance by using Act or LicenceAdmin tool on controlling machine.

3.6 Check system configuration (Linux)

It is advisable to check the configuration of some Linux system parameters before running Prosody S.

1. IP/hostname lookup - Prosody S attempts to verify that an IP address passed to `sm_vmprx_create()` is valid for the local machine. This requires that the IP address can be mapped to a valid hostname. Generally this lookup is controlled by the Named Service Switch (NSS) scheme (see man nsswitch.conf). In systems where the local machine is not looked up via DNS, it is common to add a line mapping IP address to hostname to the /etc/hosts file.
2. File descriptor limit – the number of file descriptors used by Prosody S is related to the number of vmp pairs opened. Therefore in order to support this it may be necessary to raise the file descriptor limit (NOFILE). Typically this is configured by the limits.conf file (see man limits.conf).
3. Cgroups – ProsodyS needs ability to create threads with real time scheduling, if Linux system is configured with "`cpu,cpuacct cgroup`" activated (for example through use of systemd CPUAccounting or CPUQuota unit settings), such thread type creation may be refused (logged as "failed to set thread priority: Operation not permitted"), in these cases it would be necessary to run ProsodyS under a realtime systemd slice, or disable the cpu cgroup by adding kernel boot parameter "`cgroup_disable=cpu`". A redhat article [18] gives useful information on doing this.
4. ProsodyS requires the Linux capability NET_RAW in order to create raw mode sockets.

3.7 Configuring Aculab components (Linux)

The configuration scripts are located in the *driver* directory, which is beneath the Aculab installation directory. To configure a system to use a Prosody S server, follow the steps below:

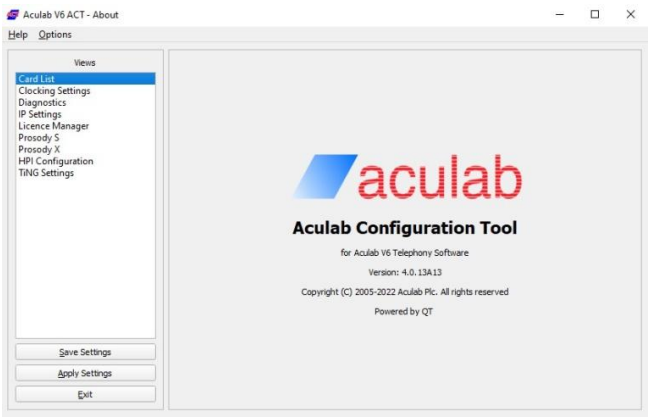
1. In the Aculab installation directory, source `setV6.sh`. This will set up the Aculab environment.
2. In the driver directory, with super-user privileges, run: `dacpinst build`.
3. A number of questions are posed, make sure that you answer "yes" when asked if you would like Prosody S support.
4. When `dacpinst` has completed another script is generated: `aculab_dacp`. This script can be used to start the Aculab components that you have configured.
5. Run: `aculab_dacp start` to start the Aculab components.

At this point both the Aculab resource manager and the Prosody S service should be running.

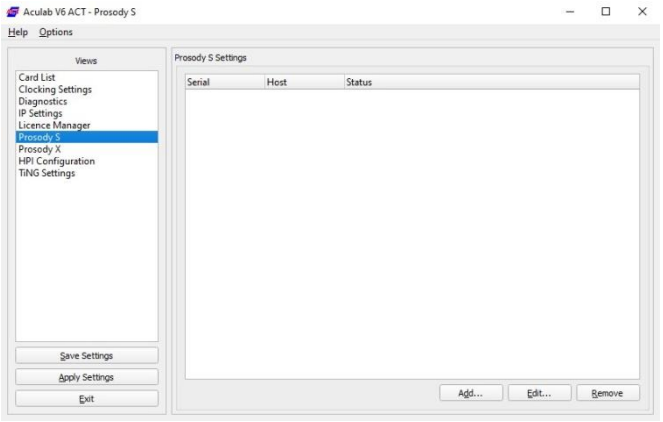
3.8 Setting up ProsodyS with Aculab configuration tool (ACT).

3.8.1 Adding a Prosody S server

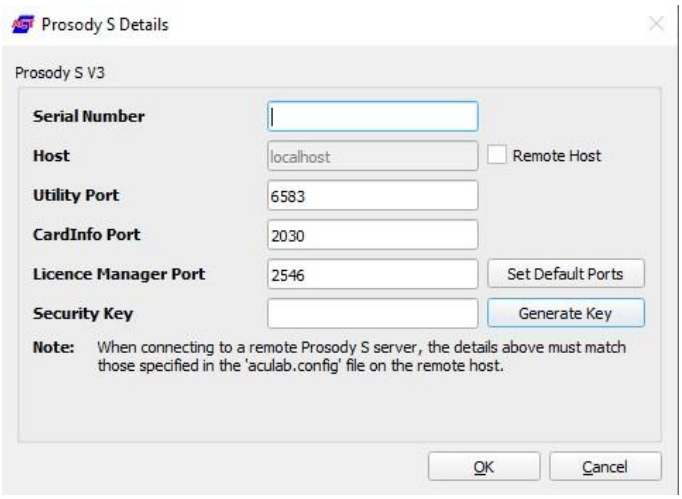
1. Start the Aculab V6 ACT [2] application. You will be presented with the following dialogue.



2. Select the *Prosody S* view to display the list of configured Prosody S servers.



3. Select the *Add* button to configure and add a Prosody S server.



Both a Prosody S server running locally and a Prosody S server running on a remote machine can be added to the local Prosody S configuration.

If Prosody S has been correctly installed via the AIT locally, then it will be running already as a service.

- Enter a unique serial number for the instance of Prosody S that is being added.
- If a local Prosody S server is being added the IP address field remains greyed out and the address will subsequently be reported as 127.0.0.1 via the resource manager APIs.

  If a remote Prosody S server is being added, click the *Remote Host* check box and enter either the hostname or IP address of the remote machine on which it is running.

  Use the default port values for the *Utility Port*, *CardInfo Port and Licence Manager Port* unless there is a conflict with another application on the machine where the selected server is running.

NOTE

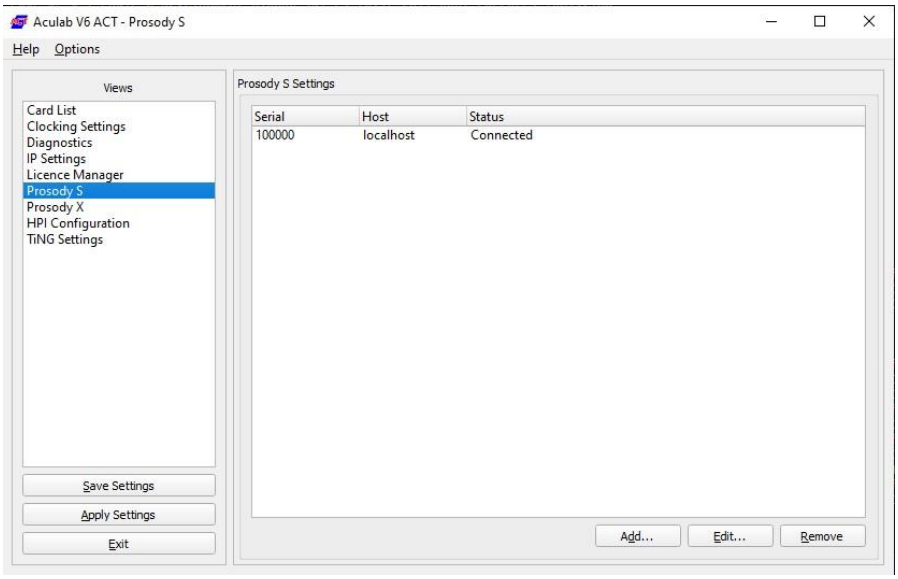
These values do not actively set the port settings in the server, but must match those configured locally for that server in its configuration file.

Enter a (security) *Key* value or press the *Generate Key* button to create a new one. This property is used to ensure the security of communication between application and server. It can be any alpha-numeric string.

NOTE

For remote servers, this Key must match the –securitykey argument entered when the server was started/installed and that is persisted in the configuration file. For a server local to the ACT this key is configured and the server restarted automatically.

4. The *Prosody S* view shows the currently configured Prosody S servers.



The status of each Prosody S server is indicated in the *Status* column.

Once the *IP Address* and *Utility P*ort settings for a Prosody S server are correct and applied, the status of the running Prosody S server will be displayed as *connected*. If there is no network connectivity, the address is incorrect, the server is not running or the *utility port* does not match that configured on the specified server, the status is displayed as *Connection attempt failed*.

3.8.2 Requesting and installing Prosody S licence keys

Please refer to the Aculab Configuration Tool (ACT) [2] document for details on how to request and install Prosody S licence keys.

3.9 Setting up ProsodyS with command line tools

The equivalent procedure to the ACT set up described in the previous section can be effected through the command line using a “prosody_ip_card_mgr” add command similar to the following (the name of the server in this example will be “MY_PS_SERVER” and its security key will be `"my_security_key"` and the TCP port for ProsodyS licence management will be 2546:

```
sudo -E prosody_ip_card_mgr --add --serial MY_PS_SERVER --key
my_security_key --ip 127.0.0.1 --licence 2546 --S
```

Licence key information (machine id) may be requested using the LicenceAdmin command line tool, and once the ProsodyS licence obtained, the licence key may be installed using the same tool. When LicenceAdmin is invoked, before issuing other commands, use command option “1” to set the port number to “2546” as was specified when adding ProsodyS server, for example:

```
IP address (default Local system):

Port number (default 54821): 2546
```

3.10 Where to find help on developing applications

Starting development with Prosody S is straightforward for those developers who are familiar with Aculab’s APIs. These APIs are used to control the IP media and speech processing resources of Prosody S in the same way that they are used to control similar Aculab card based resources.

Resource management API [4] – discovering the Aculab Prosody S server

Call control API [5] – generic call control

Extended SIP API [6] – extended SIP call control

Prosody APIs [8] – speech, wav file, DTMF, conferencing audio processing resources.

For those developers new to Aculab’s generic APIs, the wealth of documentation and example code that is available eases the developer into building simple applications that perform both call control and speech processing.

Prosody S supports most of the functionality that is available through the Prosody APIs.

RTP session handling is controlled via the ***Prosody RTP API*** [13]

Playback and record sessions are managed via the ‘high-level’ or ‘low-level’

interfaces in the ***Prosody Speech Processing API*** [10] ***WAV File API*** [11]**.** DTMF generation and detection, and universal tone generation and detection are supported using the methods in the ***Prosody Speech Processing API*** [10] and ***Prosody Generic API*** [9].

Conferencing is controlled via the ***Prosody High-Level Conferencing API*** [14] or, for low-level control, the ***Prosody Speech Processing API*** [10]***.***

Prosody S supports the event-driven model for the Prosody APIs.

Prosody S supports the use of datafeeds to enable switching of data between IP calls and speech processing resources.

For further details on available API functions, please refer to the appropriate manuals mentioned above.

4 Using ProsodyS

4.1 Overview

Aculab Prosody S provides audio media processing facilities for use on standard PC hardware. It does not require any specialised hardware and uses any standard network interface card (NIC) to provide IP connectivity.

4.1.1 Call control and IP endpoints

Prosody S supports control of VoIP calls and VoIP media resources using the Session initiation protocol (SIP) and the Aculab RTP API. It also supports simple call and media control using the generic Call API.

Aculab call control is accessed via two alternative means:

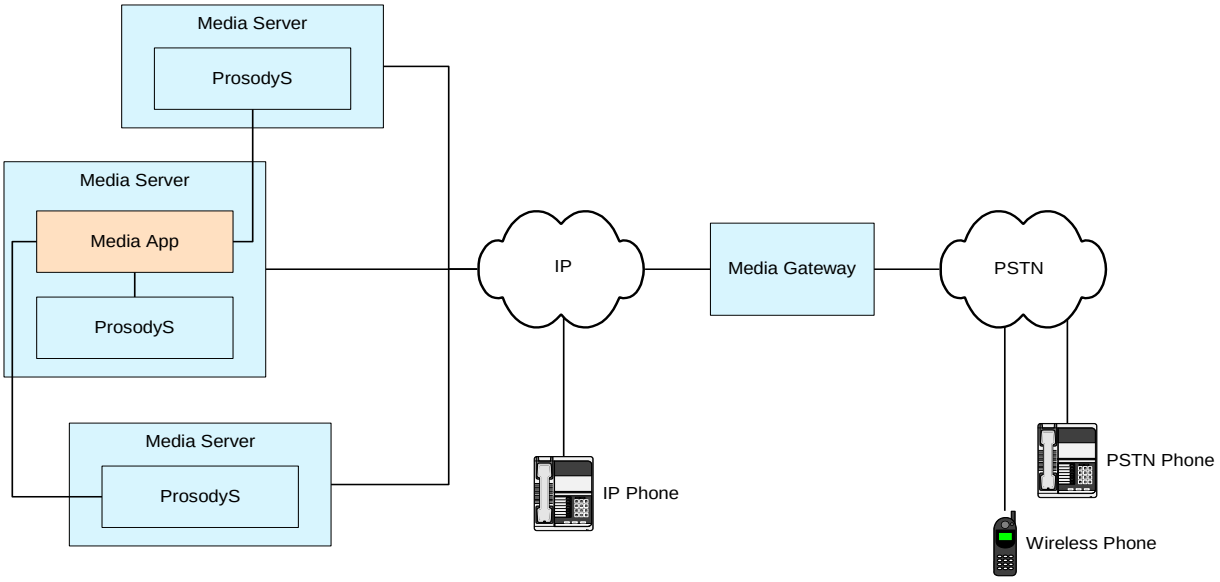
- Aculab generic call control API

or

- Aculab extended SIP API.

Proprietary call control is also supported.

The RTP media sessions send and receive audio data on IP connections in a variety of codec types.

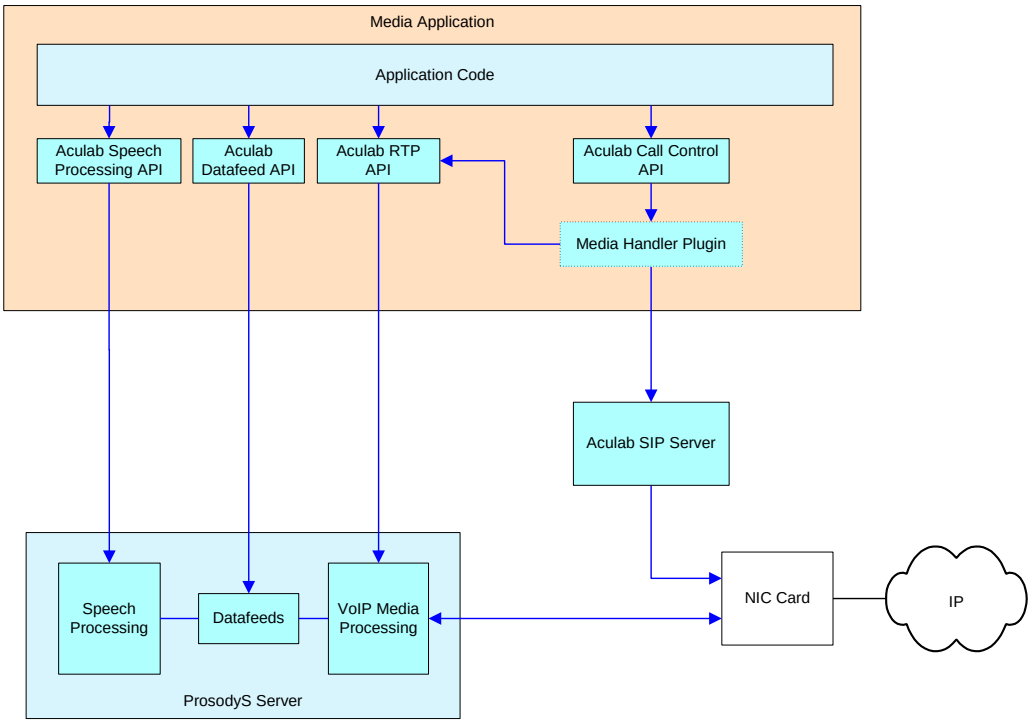


Prosody S basic scenario

A Prosody S media application provides IP endpoints in the IP network. It controls the local media processing resources for tasks such as IVR, voicemail, conferencing or similar. The IP network may be connected to the PSTN via an IP-PSTN media gateway based on hardware such as Aculab's Groomer III.

Media applications can control a Prosody S server on its local machine and also control Prosody S servers on remote machines over a network connection.

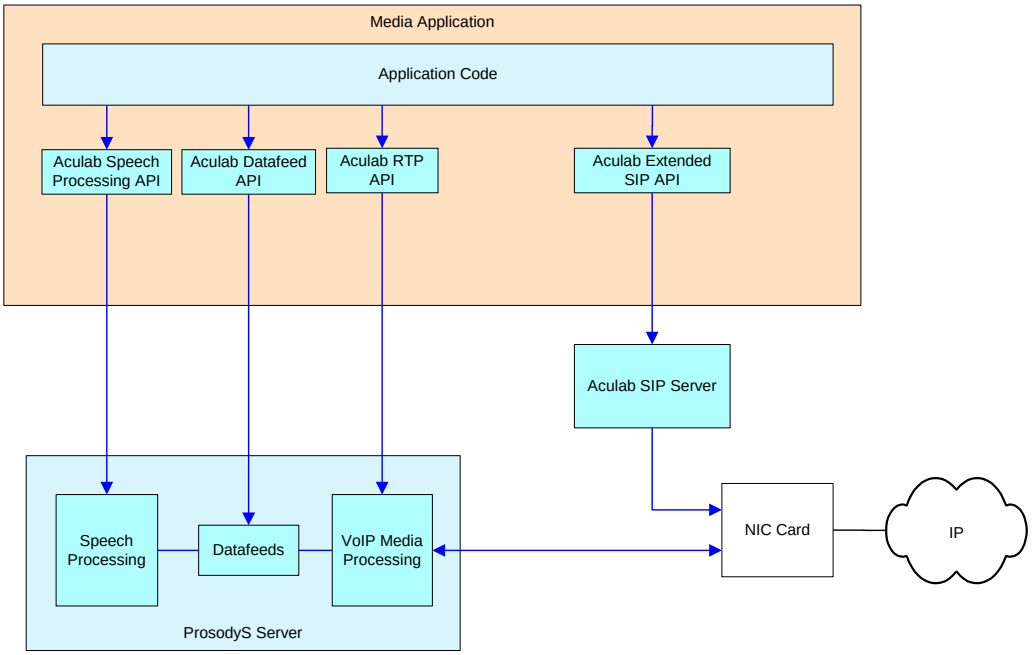
Prosody S itself is represented by a single ‘virtual card’ with zero call ports and a single media processing module.



An Audio media application using the generic call control API

This diagram shows how a Prosody S audio media application can use the existing Aculab generic call control API to control the Aculab IP signalling servers and Prosody S server. ‘Under the hood’ the media handler plugin (MHP) module controls both the IP signalling and the VoIP media processors (VMP) for each call. The application uses the RTP API to create VMPs, which the MHP manages in response to SIP events.

The audio media application uses the call control API just as it would for an Aculab Prosody X card in a Prosody X chassis.



An Audio media application using the extended SIP API

This diagram shows how a Prosody S media application can use the Aculab extended SIP API to control the Aculab SIP server directly. This exposes a high level of SIP detail to the application and provides support for more specialised SIP functionality.

The application is responsible for handling the special SIP events and using the returned SIP details to then control the VMPs (both for Tx and Rx) to establish the media session.

The media application uses the extended SIP API just as it would for an Aculab Prosody X card in a Prosody X chassis.

The Prosody S server provides much of the speech processing functionality that an Aculab Prosody X card provides via the Prosody speech processing API.

Speech processing channels are connected to VMPs using objects called datafeeds. These are sources of data, for example: VMP receivers and replay channels.

4.2 Running ProsodyS on local and remote machines

4.2.1 Installing local servers

Prosody S can be installed and run on the local machine (where the controlling telephony application resides) by installing the AIT Prosody S package. This installs the server as a local running service.

4.2.2 Installing remote servers

Prosody S can be installed and run on a machine with no other Aculab software installed and controlled from a remote application. See section 3.5 for instructions.

4.2.3 Using remote servers

`acu_get_card_info()` now returns the IP address that has been configured in the ACT for a PS server. For remote servers this will be the address specified. For a local server this will be the loopback address 127.0.0.1.

4.3 Call control

4.3.1 Overview

IP calls are created and controlled via either the generic Aculab call API or the extended SIP API.

A telephony application with simple signalling requirements can use the generic call API to communicate with the SIP IP signalling server to control how IP calls are established and closed down. It does not have to handle the details negotiated by the signalling and control the media session directly. This is managed under-the-hood and the VMPs that handle the RTP tx and rx are configured automatically.

4.3.2 IP ports

Prosody S does not expose an IP port for call control purposes. It is solely an RTP media server.

The Aculab SIP server implements its own system-wide IP port that can be used in the generic call API.

4.3.3 Call control using the generic call API

In order to use the generic call API with Prosody S there are two specific steps that must be taken. Firstly, the application must obtain the system-wide IP (SIP) port. Secondly the application must create a pair of VMPs and pass these to the call API.

4.3.3.1 Obtaining the IP port

An application obtains the IP-specific signalling port using the function:

`call_open_iptel_port()`

specifying which protocol (SIP) is required.

This port can then be used when opening an IP call using the generic call control methods `call_openin()` and `call_openout()`.

4.3.3.2 Creating the VMP objects

The application must create a pair of VMPs (one tx, one rx) and pass these to `call_openout()` or `xcall_accept()`.

These are then managed internally to control the RTP streams in accordance with the IP signalling negotiation. Address and port details, codec details etc. are required to set up the media sessions.

NOTE

The lifetime of vmprx and vmptx objects is licensed. A vmp will fail to startup if a suitable licence is not available.

NOTE

vmp creation is asynchronous and therefore failure to startup, either due to licence unavailability or for other reasons, is indicated by a corresponding error returned by `sm_vmprx_status()` or `sm_vmptx_status()`. A non-error return from `sm_vmprx_create()` or `sm_vmptx_create()` DOES NOT imply that a licence has been obtained.

Creating a vmprx

Use `sm_vmprx_create()` to create a vmprx.

The address parameter passed to this function specifies the IP address of a local network interface card (NIC) that will be used to receive RTP packets for that vmprx.

It must be set either to 0 or to the IP address of an enabled local NIC.

If the address parameter is set to 0, then a local IP address will be selected automatically.

NOTE

The automatic selection of connection address assumes that IP signalling and RTP stream is to go over the same network for the call.

Creating a vmptx

Use `sm_vmptx_create()` to create a vmptx.

4.3.4 Call control using the extended SIP API

The extended SIP API provides a more flexible means of controlling SIP signalling. It also supports 3rd party call control and re-invites.

In order to use the extended call API with Prosody S the application must obtain the system-wide SIP port. It must use the sip-specific `sip_` functions where they are provided and handle the extra sip-specific events when they occur and control the media sessions themselves using the Prosody RTP API.

4.3.4.1 Obtaining the SIP port

An application that requires extended SIP control obtains a SIP-specific signalling port using the function:
`sip_open_port()`

This port can then be used when opening a SIP call using the extended SIP API methods `sip_openin()` and `sip_openout()`.

4.3.4.2 Extended SIP functions

From then on, where available, the sip-specific methods must be used in place of the generic methods. Additionally the extended SIP events such as `EV_MEDIA_PROPOSE` need to be handled (see V6 Extended SIP API Guide [6] and SIP programmer's guide [7]).

4.3.4.3 Managing the media sessions

When using the extended SIP API it is the application's responsibility to respond to the sip-specific events as appropriate. The application must create and configure `vmprx` and `vmptx` objects as required and manage their lifetime (see Prosody RTP API manual [13]).

4.3.5 Call control using proprietary signalling

Proprietary IP signalling can be used by an application that uses Prosody S media sessions.

Media sessions can be controlled in the same way as for use with the extended SIP API described above.

4.4 Media processing

4.4.1 Overview

Prosody speech-processing resources in Prosody S are created and controlled via the standard Aculab Prosody API. A telephony application using Prosody S to perform speech processing communicates directly with the Prosody S server to control these resources.

The Prosody S server provides a subset of the functionality available in the Prosody API supporting the features listed in the Technical specification section of this document.

4.4.2 Prosody module

The API model has concept of cards populated with media processing modules For a Prosody S server, the module count for any Prosody S "card" is always 1. Hence module index 0 should be used in a subsequent call to `sm_open_module()`.

NOTE

The algorithms available on a Prosody S 'module' are not configurable and Prosody S does not support the API methods associated with firmware configuration. Additional customer specific algorithms over and above the base set can be added through installation of Prosody S plugins.

4.4.3 VoIP media processor resources (VMPs)

4.4.3.1 Overview

VMPTxs and VMPRxs are VoIP endpoints which handle the transmission and reception of RTP streams.

4.4.3.2 VMPTx

A VMPTx is created using the `sm_vmptx_create()` function.

This function initiates the startup of a VMPTx.

Startup can fail if a valid licence is not available in which case an `ERR_SM_NO_LICENCE` error is returned by `sm_vmptx_status()`.

The VMPTx needs to be configured to send RTP and RTCP data to a specific IP address and port number using the `sm_vmptx_config()` function.

A VMPTx does not have inherent IPv6 or IPv4 type until it is configured with destination address. For IPv6 this would be done with `sm_vmptx_config_ipv6()`.

NOTE

The equivalent field to the IPv4 TOS_RTP for IPv6 is bits 20..27 of `destination_rtp.sin6_flowinfo` which will get placed in the RTP IP packets Traffic Class 8 bit field. Due to Operating System limitations, this setting of traffic class functionality is not currently available under Windows.

NOTE

Note if an IPv6 source address is specified when configuring VMPTx, both source and destination addresses must be of same scope, i.e. either both global or both link local.

4.4.3.3 VMPRx

A VMPRx is created using the `sm_vmprx_create()` function. When creating a VMPRx, it is necessary to specify whether it is to accept IPv6 RTP or IPv4 RTP using the `type` field

This function initiates the startup of a VMPRx.

Startup can fail if a valid licence is not available in which case an `ERR_SM_NO_LICENCE` error is returned by `sm_vmprx_status()`.

The VMPRx is allocated two local ports numbers (for RTP and RTCP), the values of which are returned by `sm_vmprx_status()` for the `kSMVMPrxStatusGotPorts` or `kSMVMPrxStatusGotPortsIPv6` status. The range of ports numbers defaults to the maximum allowable range 1024-65531. This range can be reduced by specifying max and min port numbers in the configuration file.

In the parameters for this function, the IP address of a local network interface card (NIC) can be specified. This limits reception of RTP packets to the specified address, and consequently packets arriving with a different address will be ignored. If no address is specified in this function, RTP packets addressed to all local interfaces will be processed.

4.4.4 Allocating speech processing resources

A Prosody S channel is allocated in the normal way using

`sm_channel_alloc_placed()`, using the module id returned from `sm_open_module()`.

4.4.5 Using speech processing resources

Conferencing, replay, record, DTMF detection and DTMF generation are all supported in Prosody S. Both the low-level and high-level replay/record APIs are supported, as is the WAV file API.

4.4.5.1 Replaying data

Using the low-level playback API

The `sm_replay_start()`, `sm_replay_put_data()` and associated functions can be used to control data replay as described in the speech processing API guide [10].

Using the high-level playback API

The `sm_replay_file_start()`, `sm_replay_file_progress()` and associated functions can be used to control data replay as described in the high-level play/record API application note. The functions `sm_replay_wav_start()` (and `sm_replay_wav_close()`) are also supported. There is a parallel set of “bfile” variants of these functions that can be used to overcome `stdio` limit of number of simultaneous file streams being read/written.

4.4.5.2 Recording data

Using the low-level record API

The `sm_record_start()`, `sm_get_recorded_data()` and associated functions can be used to control data recording as described in the Prosody generic API documentation.

Using the high-level record API

The `sm_record_file_start()`, `sm_record_file_progress()` and associated functions can be used to control data recording as described in the high-level play/record API application note. The functions `sm_record_wav_start()` (and `sm_record_wav_close()`) are also supported. As for replay case there is a parallel set of “bfile” variants of these functions.

4.4.5.3 Detecting tones and DTMF

DTMF signals on IP calls are usually sent via a special (RFC2833) packet, not as audio data. The VMP Rx can detect and report these packets, and/or convert them to an audio representation.

Detecting DTMF using the VMP Rx

When a DTMF RTP packet is received a suitably configured VMP’s event will be signalled and `sm_vmprx_status()` will return `kSMVMPrxStatusDetectTone` with the tone ID and volume received.

Detecting tones and speech energy using the signal detector

The `sm_listen_for()` function is used for detecting in-band DTMF and other tones. This should be used in conjunction with the Prosody event mechanism and the function `sm_get_recognised()`, described in the Prosody generic API [9] and speech processing API [10] documentation respectively.

RFC2833 DTMF will not be detected by the tone detector unless the VMP Rx has been configured to convert incoming DTMF packets into their audio representation. See the function `sm_vmprx_config_tones()` in the Prosody RTP API manual [13] for controlling the conversion.

4.4.5.4 Generating tones and DTMF

DTMF signals on IP calls are usually sent via a special packet, not as audio data. The VMP Tx can directly generate these packets, or it can monitor the audio data passed to it and convert detected audio DTMF into the correct packet.

Generating DTMF packets using the VMP Tx

Call `sm_vmptx_generate_tones()` from the Prosody RTP API manual [13] to directly generate DTMF packets.

Generating tones and DTMF packets using the signal generator

The following API functions are supported in Prosody S, as documented in Prosody speech processing API [10]:

`sm_play_digits()`, `sm_play_digits_status()`, `sm_play_tone()`, `sm_play_tone_abort()`, `sm_play_tone_status()`, `sm_add_output_freq()`, and `sm_add_output_tone()`

These functions will directly generate audio data which will be transmitted as-is unless the VMP Tx has been configured to detect DTMF in audio data via `sm_vmptx_config_tones()` (see the Prosody RTP API manual [13]). Only tones with a corresponding RTP tone ID can be converted.

4.4.5.5 Using conferencing

Using the low-level conferencing primitives

Prosody S supports all features of the low-level conferencing API. See the Prosody speech processing API [10] for details on how to create and manage conferences. In particular, the Prosody guide ‘How to perform conferencing’.

Using the high-level conferencing API

The high-level conferencing API can be used to implement simple conferences where all parties are equal. See the Prosody high level conferencing API [14] and the Prosody guide ‘How to perform conferencing’.

4.4.5.6 Recording from a conference output

A recorder is set up in the normal way using the low-level recording API.

In the call to `sm_record_start()` specify the `alt_data_source_type` to be `kSMRecordAltSourceOutput` and `alt_data_source` to be the channel of the conference output.

4.4.6 Fax processing

ProsodyS supports T.30 fax over G.711 RTP and T.38 fax. See the Fax Library documentation [16] for more details.

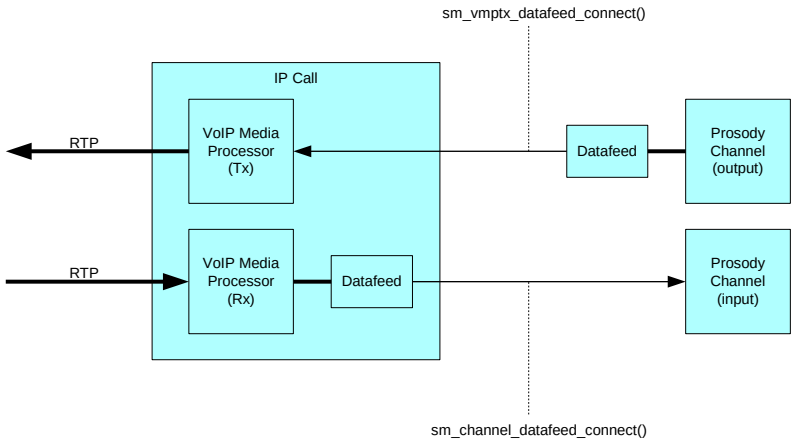
4.5 Switching

4.5.1 Connecting speech processing to IP calls

IP calls typically will involve a VMP Tx and VMP Rx.

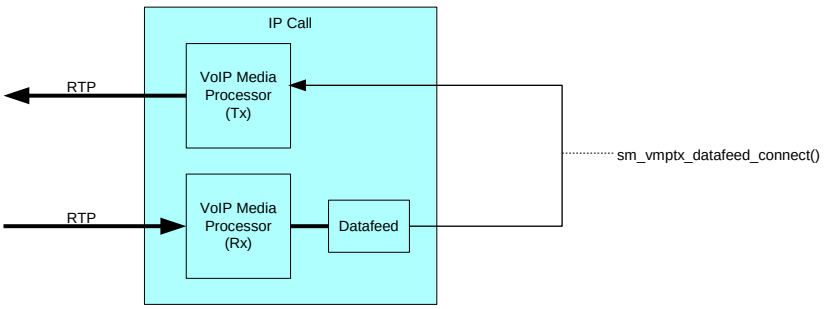
A VMP Rx owns a datafeed object, the Id of which can be retrieved using `sm_vmprx_get_datafeed()`. Input speech processing channels can source their data from this datafeed using `sm_channel_datafeed_connect()`.

An output speech processing channel, similarly, owns a datafeed object, the Id of which can be retrieved using `sm_channel_get_datafeed()`. A VMP Tx can source its data from this datafeed using `sm_vmptx_datafeed_connect()`.



4.5.2 Connecting IP calls to IP calls

IP calls can also be connected together using the datafeed objects as follows:



Alternatively, use may be made of sm_vmprx_config_forwarding API call.

4.6 Configuration file

Prosody S stores a number of global parameters in an xml configuration file called aculab.config which is read on server startup.

This file will be first written by Prosody S when a local Prosody S server is added via the ACT or when the Prosody S server is run from the command line with the security key argument.

In Windows this file is located in `$(ACULAB_ROOT)\cfg\`.

If $(ACULAB_ROOT) is not set (e.g. a minimum remote installation of Prosody S) this configuration file will be located in the same directory as the Prosody S executable.

In Linux this file is always located in `/etc/aculab/`.

A simple example of the configuration file is shown below. The attributes that control Prosody S are located within the XML element <card type="prosodys_v3">...</card>

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
<aculab>
        <v6>
             <cards>
                  <card type="prosodys_v3">
                       <security key="1234"/>
                       <resources>
                            <ports>
                                 <port value="16385" name="assp"/>
                                 <port value="2031" name="asspmon"/>
                                 <port value="2030" name="cardinfo"/>
                                 <port value="6583" name="utility"/>
                                 <port value="2546" name="licence-manager"/>
                            </ports>
                            <rtpports>
                                 <rxrange minvalue="1024" maxvalue="65531"/>
                                 <defaulttx value="65535"/>
                            </rtpports>
                       </resources>
                       <logging>
                            <log type="file" minlevel="0"
                                 maxlevel="2" filename="ProsodySServ.log"/>
                       </logging>
                  </card>
             </cards>
        </v6>
   </aculab>
</config>
```

4.6.1 <security> settings

This element defines the security parameters associated with the Prosody S server.

It has a single attribute:

- key (no default) – that is used to secure the communication between authorised applications and the Prosody S server. It must match the value set in the ACT Prosody S details dialog – security key field.

4.6.2 <resources> settings

4.6.2.1 <ports>

This element contains a number of port values that are used by the application layer and Prosody S to communicate. A number of them must match those settings specified in the ACT.

- assp (default 16385) - this determines the port used for ASSP message passing between applications and Prosody S.
- cardinfo (default 2030) – this is used by the resource manager and must match the value set in the ACT Prosody S details dialog – cardinfo port field.
- asspmon (default 2031) – this is used by the TiNG library and should be 1 greater than the cardinfo setting.

- utility (default 6583) – this is used by the ACT to determine the current status of the selected Prosody S server and must match the value set in the ACT Prosody S details dialog – utility port field.
- licence-manager (default 2546) – this provides access to the licence manager facilities in Prosody S and must match the value set in the ACT Prosody S Details dialog – licence manager port field.

4.6.2.2 <rtpports>

This element contains a number of settings relating to the rtp ports used to transmit and receive rtp packets.

- rxrange – this defines the range of local ports that can be allocated by the Prosody S server in order to receive rtp packets. It has the following attributes:
  - minvalue (default 1024)
  - maxvalue (default 65531)

  The range of allowable values for these settings is 1024-65535. However it is sensible for this range not to overlap the defaulttx port.

  Registered port numbers:

  The well-known port range 0-1023 is unavailable for use.

  The range of ports that should be formally registered by applications is 1024-49151.

  The remaining range (49152-65535) is for dynamic use by applications.

NOTE

If the extent of this range is decreased too far it can increase the likelihood of `sm_vmprx_create()` failures as there is a delay between closing a vmprx and the allocated port becoming available for reallocation.

- defaulttx (default 65535) – this determines the value that is to be used as the UDP src port in transmitted RTP packets if none is specified for a vmptx via `sm_vmptx_config()`.
- namespace – Linux only - this defines net namespace used for RTP ports, it has following attribute:
  - net (default is not to use a namespace)

  For example if there was namespace added for rtp:

  ```
  ip netns add rtp
  ```

  Then this element should be

  ```
  <namespace> net="rtp" />
  ```

4.6.3 <logging> settings

This element contains the settings for any loggers that are required.

Typically a file logger may be configured to record only errors. For various reasons, further diagnostic logging may be required. In which case either the logging level for the file logger can be increased or additional loggers can be added to the configuration.

File loggers write their output to the specified filenames in specific locations:

In Windows, this is typically located in `$(ACULAB_ROOT)\log\`.

If $(ACULAB_ROOT) is not set (for example, for a minimum remote installation of Prosody S) the log file will be located in the same folder as the Prosody S executable.

In Linux, this file is always located in "`/var/log/aculab/`".

4.6.3.1 Logging level

In the following descriptions of available loggers the `minlevel` and `maxlevel` attributes determine the range of logs that are to be logged for that logger and can be set to values from the following range:

- 1 LogLevelAlways - critical log entries such as version details, startup
- 2 LogLevelError - errors
- 3 LogLevelWarning - unexpected results that are not serious
- 4 LogLevelInfo - normal operational information
- 5 LogLevelVerbose - intensive operational information

4.6.3.2 File logger

The following line enables a logger that logs all information of the specified range of levels to a single file of name `filename`.

```
<log type="file" minlevel="0" maxlevel="2" filename="ProsodySServ.log"/>
```

4.6.3.3 Rotating file logger

The following line enables a logger that logs all information of the specified range of levels to a series of files.

```
<log type="rotatingfile" minlevel="0" maxlevel="2" filename="ProsodySServ"
fileext="log" maxlines="1000" maxfiles="5" />
```

Each file has name `filenameDD.fileext` where DD indicate a two digit identifier which increments from 00.

The `maxlines` attribute determines the maximum number of lines in each file. Log entries that exceed this number of lines will cause the next file in the series to be opened.

The `maxfiles` attribute determines the maximum number of files written. Log entries that exceed this number of files will then be written to file 00, overwriting the original log data and so on.

4.6.3.4 Console logger

The following line enables a logger that logs all information of the specified range of levels to the screen console. Typically this will only be useful if running the server in 'debug' mode using the –d command option.

```
<log type="console" minlevel="0" maxlevel="2"/>
```

4.6.3.5 Syslog logger

For Linux only, the following line redirects log output through invocations of Linux "syslog"

```
<log type="syslog" minlevel="0" maxlevel="2"/>
```

4.6.4 <buffering> settings

This element contains the settings that can override the default buffer size for ProsodyS replay (`fromhost` attribute) and record (`tohost` attribute) activities. Normally the default settings are appropriate but if responsiveness of application to read/write wakeup events is very slow and it suffers from reported replay underruns or record overruns, larger buffers can be assigned. Zero sets a default value of 8192 bytes of buffering (about 1s if replay/record format is 8KHz A-law samples), larger values can be specified for more leeway in application wakeup responsiveness.

```
<host fromhost="0" tohost="0" />
```

4.6.5 <affinity> settings

This element allows finer control over ProsodyS exploitation of system CPUs which may be necessary for a system where the telephony application cohabits with ProsodyS and handles a large number of simultaneous calls, or a system with a large number of CPU cores where it is advantageous to dedicate some to packet processing or other uses. By default ProsodyS will use all CPU cores on a system for processing TiNG tasks, thus will have N task processing threads on a N CPU core machine. However this can be restricted to a smaller number using the affinity `count` attribute. To specify use of N CPU cores, a value of N should be specified. To specify use of all except M CPU cores, a value of –M should be specified.

These task processing threads are normally scheduled freely among available CPU cores, however if the affinity `pin` attribute is set to "1", the task processing threads will be pinned to CPU cores starting from a selected base CPU core (normally that specified by affinity `base` attribute). On some (generally Linux) systems with multiple CPUs, the cores are numbered alternating between CPUs, in order to place task processing threads on same physical CPU in these systems, a `step` attribute may be specified which will be added to `base` for each task processing thread, for example.

```
<host base="1" count="4" step="2" pincores="1"/>
```

For some system configurations, where further control of other elements of ProsodyS scheduling may be necessary (for example packet handling), additional attributes may be applicable – contact Aculab support for more details).

On Windows systems in order to obtain greater accuracy for packet arrival times used when measuring round trip times by RTCP functions and packet to epoch times reported in augmented summary reports it is necessary to dedicate a separate thread for packet reception, this is done by specifying an rxqueue attribute, typically this is specified together with the `count` attribute specifying the number of cores for processing TiNG tasks, for example on a 24 core machine, 12 threads could be dedicated to ProsodyS TiNG tasks and one to ProsodyS packet reception.

```
<host count="12" rxqcount="1"/>
```

It is not currently possible to configure Linux ProsodyS for improved packet arrival time accuracy.

4.6.6 <mem> settings

For ProsodyS on Linux for some system use configurations it may be appropriate to preallocate some heap memory and lock the pages for this memory and PS code into

memory. This can be achieved by configuring a <mem> element with attributes similar to the following where `sbrkmb` specifies number of additional megabytes to reserve before locking PS current code/data pages into memory.

```
<host sbrkmb="32" mlockall="1"/>
```

4.6.7 <clock> settings

If ProsodyS RTCP functions are used to measure round trip times and augmented summary reports used to measure render delays, then on Windows systems in order to obtain more finely grained timing measurements ProsodyS can be configured to obtain timestamps using `QueryPerformanceCounter`, on some systems this may be at the cost of using more CPU time. If this extra accuracy is required, the following attribute may be specified.

```
<clock source="QPCFORUS"/>
```

If this attribute is used, for best results the host system should have a CPU with the TSC-INVARIANT property as reported by Microsoft/Sysinternals coreinfo utility.

4.6.8 <assp> settings

ProsodyS uses some UDP based protocols (ASSP/cardinfod) to interact with an application linked with the TiNG library. In some configurations it may be necessary to ensure the source IP address of ProsodyS transmitted ASSP/cardinfod UDPs are the same as the destination IP address of the ASSP UDPs ProsodyS receives. The following attribute can be set to enable this mode (on some systems this may be at the cost of using more CPU time).

```
<assp settxsrcaddr="1"/>
```

4.6.9 <options> settings

4.6.9.1 use_epoll

Linux only - by default ProsodyS uses multiple Linux epoll system calls on multiple task processing threads to detect incoming RTP packets on multiple ports at end of epoch when idle. By setting this value to zero this behaviour is replaced to have a single task processing thread do this work at the beginning of epoch which might be more appropriate for some workload/platform combinations. Setting this value to a value to greater than one increases the number of epolls done to look for multiple packets arriving on same port during current epoch.

4.6.9.2 cpumonfmt

Normally the values read from TiNG data channel configured for `kSMDCProtocolCpu` will be sum of base workload time for task processing and transitory API message processing time. Sometimes it is useful to separate out the two values, setting this option, setting this option changes the format of values read from data channel to give four 32 bit values of CPUMON info about each epoch of which 2nd word is base-load+message-processing (as normal) and 3rd word is message-processing time on its own.

4.6.9.3 tdm_stride

For Aculab use only (embedded PS integration within chassis products)

4.6.9.4 enable_f16c

Enable use of AVX type instructions for faster companding of G.711 samples

4.6.9.5 sched_wakeup_latency_us

Linux only - Power management Quality of Service control – writes specified value `to /dev/cpu_dma_latency` and holds file descriptor open to make it stick. By default, a value of 20 written, if -1 is specified, effect is just to report in Prosody S log the latency value that is currently set.

5 Performance

5.1 Linux file systems

The Linux "ext4" file system is commonly used on modern Linux installations for partitions / mounts points where data integrity is paramount. However, when high data throughput is the main consideration, it may be advantageous to use an alternative non-journaling file system such as “xfs”. This is particularly of note for high load applications that record data to file.

Using the "xfs" file system on the partition where files are to be recorded can in some systems substantially increase the number of supported recordings to be performed simultaneously by the same application.

5.2 Windows power options

Windows servers allow configuration of the system power plan. For ProsodyS, the system is best exploited by setting the plan to “High Performance”.

5.3 Antivirus

Some anti-virus tools implement packet scanning on incoming network packets. This can introduce overhead to handling network activity that can be difficult to diagnose as it is at such a low level. In windows this can show up as kernel activity related to deferred procedure calls associated with network interrupts from incoming packets.

5.4 Affinity

On systems where ProsodyS cohabits with telephony application, it may be desirable to partition use of system CPUs between ProsodyS and other processes. See section 4.6.5 for details of configuration of ProsodyS affinity.

5.5 Linux network stack

For high density applications, alternative configuration of Linux network handling may be appropriate, such as assigning additional threads to network stack to process incoming packets (threaded NAPI), or setting a bigger value for network device `txqueuelen`. An application note is available from Aculab support that discusses these alternate configurations that are system and NIC dependant.

6 Technical specification

IP call control

SIP using extended API
SIP using generic call control API
Proprietary call control

API availability

Call control API via the Aculab call API
Speech processing via the Aculab Prosody API
Virtual switching via the Aculab Prosody API (datafeeds)

Network interface

Standard Ethernet connection

Audio IP media streaming:

RTP - frame sizes, multiples of 10ms
Codecs:
- G.711 A-law
- G.711 μ-law
- G.729AB (plugin supplied)
- G.726 (16, 24, 32 and 40 Kbit/s)
- G.723.1 (5.3 and 6.3 Kbit/s)
- G.728
- G.722
- G.722.1
- Opus (excepting rx FEC)
- iLBC
- EVRC
- AMR-NB
- AMR-WB (G.722.2)
- GSM-FR
- Tetra
- Speex
- SILK

Voice activity detection (VAD)
Discontinuous transmission (DTx)
Packet loss concealment
RTCP
RTP TOS settings
RFC 2833 tones
In-band tones

T.38

Over UDP as an Internet aware Fax (IAF) device
Via the Aculab datacomms API
Data rate management method 2 (transferred TCF)
Versions 0, 1 and 2 of the ASN.1 syntax
ECM and non-ECM image transfer
UDPTL redundancy

T.30

Modems: V.17, V.21, V.27, V.29, V.34 (plugin supplied)

Audio processing features

- Replay: Volume control
  Automatic gain control
- Record: Volume control
  Automatic gain control
  Termination on silence
  Tone elimination
- Tone generation:
  - DTMF generation
  - Universal tone generation
- Tone detection:
  - DTMF detection
  - ANSAM detection
  - Universal tone detection
  - Call progress tone detection
- File formats:
  - G.711 A-law PCM
  - G.711 µ-law PCM
  - OKI ADPCM
  - IMA ADPCM
  - 16 bit data
  - 8 bit data
  - 8 bit (signed) data
- Multiparty conferences
  - Noise gating
  - AGC
  - DTMF clamping
- Signal categorisation:
  - Identification of call as live speaker or answer machine
- Sample rate conversion:
  - Audio data can be re-sampled using the signal path processing API

Licence management

Per VMP licensing via a software licence key.

Optional Plugins

Secure RTP

7 Support

7.1 FAQ

7.1.1 Why are there more than one Prosody S packages?

The ProsodyS_Standalone_Server package contains only the Prosody S server, without any dependencies. This package can be installed on its own for use as a remote standalone server.

The ProsodyS package contains documentation and dependencies on a variety of other packages that are required in order to connect to a local or remote Prosody S server and build and run Prosody S applications.

The ProsodyS_SRTP_Plugin package contains a plugin to the Prosody S server that provides SRTP support.

The ProsodyS_G729_Plugin package contains a plugin to the Prosody S server that provides G729AB support.

The ProsodyS_V34HD_Plugin package contains a plugin to the Prosody S server that provides a V.34 half duplex modem for fax support.

7.1.2 Where is the log file?

By default Prosody S writes error level log entries to a log file in the following locations:

In Windows this is typically `$(ACULAB_ROOT)\log\`.

If $(ACULAB_ROOT) is not set (for example, for a minimum remote installation of Prosody S) this is in the same folder as the Prosody S executable.

In Linux this is always `/var/aculab/log`.

7.1.3 How do I obtain a port Id to use with Prosody S?

Either: use the `call_open_iptel_port()` function - when using the generic call API.

Or: use the `sip_open_port()` function - when using the extended SIP API.

7.1.4 Why is the switch API not available?

Prosody S does not use the standard switching API as this is designed for use with TDM systems.

All switching is performed using datafeed objects and associated functions in the Prosody API.

7.1.5 When is a licence used?

A single licence enables the creation of two media processing (VMP) objects. Typically this will be a `vmprx` and a `vmptx` that go to make up a single bidirectional call. VMPs subsequently release their licences when they are destroyed. Alternatively a licence enables creation of FMP objects for T.38 fax calls.

7.1.6 How do I know how many licences I have?

The Aculab configuration tool (ACT) provides information on the number of licences

currently available on any particular Prosody S server.

7.1.7 How do I increase the number of IP calls I can make?

Additional licences can be purchased directly from Aculab, contact your Aculab Account Manager for further assistance.

7.1.8 What happens in Prosody S if the CPU activity reaches near 100%?

As happens on dedicated telephony boards, if the CPU activity reaches near 100% there may be some degradation of audio quality due to missing packets or packets not being transmitted at the required rate. This is due to the CPU not being able to fully service each call sufficiently regularly to supply the call with the expected amount of data.

The Prosody S server should continue to operate correctly during periods of high CPU activity but the audio quality may be affected.

7.1.9 How are multi-homed machines handled?

A machine with multiple network interface cards (NICs) requires some care in detailing which addresses to use for both IP signalling and RTP transmission and reception.

The source address for an RTP packet transmitted by a `vmptx` is that of the interface card on which it is sent. This is determined from the current main routing table configuration of the machine..

The address on which each `vmprx` is to listen for RTP packets is specified in the `sm_vmprx_create()` function. Only packets received on the specified address will be processed. If address 0.0.0.0 is specified (the default) in `sm_vmprx_create()` then packets received on any local addresses will be processed.

For server control protocol used by TiNG library, see also note on settings in 4.6.8

For Linux systems, Prosody S can be configured to use a network namespace for RTP packet transmission and reception.

7.1.10 How do I set the security key?

The security key needs to be known both by the Prosody S server and the application communicating with it.

The security key can be configured for the application via the ACT – Prosody S settings page, either when adding a server or via the edit button.

For a Prosody S server local to the application, setting this key in the ACT will configure the key in the server and restart it.

For Prosody S servers remote to the application, the key can be configured on the server via the Prosody S command line. The same key must then be entered in the ACT for use by the application.

7.1.11 Why does acu_get_card_info() return server address 127.0.0.1?

acu_get_card_info() returns the IP address of the specified server as stored in the Resource Manager. This is typically configured via the ACT when adding a Prosody S card.

When installed locally, Prosody S will default to reporting its IP address as 127.0.0.1, the loopback address.

7.2 Troubleshooting

7.2.1 Prosody S server fails to start

Check the log file. Startup errors will typically be reported here.

Reasons for startup failure include:

- The server may have been unable to open one of more of its ports. Some ports may be unavailable if they are in use by other applications. Alternative port numbers may be configured in the config file (see the section on Configuration file).
- Only one instance of Prosody S can run on a single host.
- Prosody S requires a CPU that supports at least the SSE2 instruction set.
- File permissions on Linux.
- CPU CGROUP settings on Linux (for example through CPU accounting enabled by a systemd service) prevent Prosody S from setting required setting thread priority – see section 3.6

7.2.2 Prosody S starts but API calls return ERR_SM_DISCONNECTED

Check the security keys used by the server and the application match. Ensure the security key is set correctly in the aculab.config file and that it is in the correct location for the server to read.

On Linux ensure the server is running with super-user privileges. The server may not operate correctly with standard user rights.

7.2.3 acu_get_system_snapshot() fails to return a Prosody S server

A Prosody S server is not running or has not been added to the resource manager or is not in the connected state.

The act of installing the ProsodyS_Standalone_Server package via the Aculab Install Tool (AIT) installs the Prosody S server as a running service. Reinstall this package to ensure the server is running.

The Prosody S server can then be added to the resource manager's known list of Prosody S servers via the Aculab configuration tool (ACT).

The resource manager will try to connect to the running Prosody S server via the utility port and display 'connected' status when successful. The utility port setting must match that set in the server's configuration file.

7.2.4 acu_open_prosody() fails with error -1

If a Prosody S server has been added to the resource manager, is running and connected, then this error may be returned if the cardinfo port setting in the ACT does not match that configured on the Prosody S server itself via its configuration file.

7.2.5 sm_vmptx_status()/sm_vmprx_status fail after calling sm_vmptx_create()/sm_vmprx_create()

One possible cause of this is mismatched security key configuration.

The security key used by the Prosody S server must match the security key used by the application. If the Prosody S server security key is configured from the command

line (e.g. on a remote server) it must match that specified for the server in the ACT.

A second cause may be the lack of a valid licence.

Use of either of the functions `sm_vmptx_create()` or `sm_vmprx_create()` will result in licence acquisition. If the number of licences installed are already in use then `sm_vmptx_status()` or `sm_vmprx_status()` will subsequently report an error `ERR_SM_NO_LICENCE`.

Another possible cause is when the vmp is requested to use a feature that is not supported by Prosody S, such as the melpe codec, or if an attempt is made to use new codec with licence that predates introduction of this feature. In this case `sm_vmptx_status()` or `sm_vmprx_status()` will report `ERR_SM_NO_SUCH_FIRMWARE`.

Only on Linux, this may also be caused by passing in a local address to `sm_vmprx_create()` that cannot be found and that is not entered in the `/etc/hosts` file. In this case `sm_vmprx_status()` will return `ERR_SM_BAD_PARAMETER`.

7.2.6 There is nothing audible on the call

No RTP packets will be transmitted on an IP call unless the input of the call's vmptx is switched to a valid datafeed, using the `sm_vmptx_datafeed_connect()` function. Also check system firewall is permiting transmission/reception of RTP. In addition check RTP payload type is set correctly.

7.2.7 The audio quality is severely degraded

If the recorded or outgoing audio quality is very noisy and barely recognisable as speech, a common cause is incorrect configuration of the file companding settings.

The file format of the replayed file must be specified correctly in the relevant speech processing functions, `sm_replay_start()` for example.

7.2.8 There are discontinuities in the replayed data

If VAD is disabled or there are no periods of low audio level in the replayed data then the discontinuities may be due to either a highly loaded network or an overloaded CPU.

Ensure the CPU is not being loaded by excessive logging from either the Prosody S server or one of the VoIP servers.

Except for the purposes of diagnosis the logging for all these servers should be minimised to logging only errors at most.

Ensure that other applications or services are not putting undue load on the CPU.

7.2.9 Why does recording not terminate?

If `max_silence` or `max_octets` are used to trigger the termination of a recording, the Prosody channel must be switched to any valid datafeed using `sm_channel_datafeed_connect()`.

Prosody S does not record anything if no data is supplied to its input and hence both `max_silence` and `max_octets` will not be applied.

`max_elapsed_time` does not have this limitation and should stop the recording after the specified time whether data is switched to the channel or not.

7.2.10 Getting ERR_SM_NO_RESOURCES on Linux at low channel counts

If Prosody API calls return `ERR_SM_NO_RESOURCES` at low channel counts, it is possible that there are insufficient files descriptors available to the current process. It is often necessary to raise the per process limit for file descriptors (NOFILE) when running applications that use the Prosody API.

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