Wazo Documentation

Release 18.03

The Wazo Authors

Mar 05, 2018

Contents

1 Table of Contents **3**

1.1 Introduction . . . 3
1.2 Installation . . . 3
1.3 Getting Started . . . 9
1.4 Upgrading . . . 15
1.5 AdminUI . . . 66
1.6 Wazo Client . . . 70
1.7 System . . . 90
1.8 Ecosystem . . . 169
1.9 Administration . . . 185
1.10 Contact Center . . . 347
1.11 High Availability (HA) . . . 385
1.12 Scalability and Distributed Systems . . . 396
1.13 API and SDK . . . 409
1.14 Contributors . . . 491
1.15 Quality assurance . . . 591
1.16 Troubleshooting . . . 593
1.17 Community Documentation . . . 601
1.18 Documentation changelog . . . 602
1.19 Attribution Notice . . . 602

2 Changelog **603**

3 Indices and tables **605**

Wazo is an application suite based on several free existing components including Asterisk, and our own developments to provide communication services (IPBX, Unified Messaging, . . .) to businesses.

Wazo is free software. Most of its distinctive components, and Wazo as a whole, are distributed under the *GPLv3 license*.

You may also check the Wazo blog for more information.

Wazo documentation is also available as a downloadable HTML, EPUB or PDF file. See the downloads page for a list of available files or use the menu on the lower right.

See *Attribution Notice*

CHAPTER 1

Table of Contents

1.1 Introduction

Wazo is a PABX application based on several free existant components including Asterisk and our own developments. Wazo provides a solution for enterprises who wish to replace or add telephone services (PABX).

Wazo is free software. Most of its distinctive components, and Wazo as a whole, are distributed under the GPLv3 license.

1.1.1 Wazo History

Wazo is a fork of XiVO, which was created in 2005 in France by Sylvain Boily and the company Proformatique. In 2010, Proformatique merged with Avencall, and Avencall acquired the copyright and trademark of XiVO.

Sylvain then moved to Quebec City and founded Proformatique, Inc. where the XiVO core development team worked from 2011 until November 2016.

In November 2016, Proformatique Inc. was shut down and the development team forked XiVO to create Wazo. Its first release, Wazo 16.16, was released in December 2016.

1.2 Installation

1.2.1 Installing the System

Please refer to the section *Troubleshooting* if ever you have errors during the installation.

There are two official ways to install Wazo:

- using the official ISO image
- using a minimal Debian installation and the Wazo installation script

Wazo can be installed on both virtual (QEMU/KVM, VirtualBox, . . .) and physical machines. That said, since Asterisk is sensitive to timing issues, you might get better results by installing Wazo on real hardware.

Requirements

For a small install of about 20 users, less than 20 calls per day:

- RAM: 2 GiB with 1 GiB swap
- Storage: 2.5 GiB of storage is a very tight minimum, 8 GiB is comfortable

Installing from the ISO image

- Download the ISO image. (latest version) (all versions)
- Boot from the ISO image, select `Install` and follow the instructions. You must select a locale with charset UTF-8.
- At the end of the installation, if you are installing version 16.13 or before, you need to configure your system to *use the Wazo infrastructure*, otherwise some errors might occur.
- Continue by running the *configuration wizard.*

Installing from a minimal Debian installation

Wazo can be installed directly over a **32-bit** or a **64-bit** Debian stretch. When doing so, you are strongly advised to start with a clean and minimal installation of Debian stretch.

The latest installation image for Debian stretch can be found at https://www.debian.org/releases/stretch/debian-installer.

Requirements

The installed Debian must:

- use the architecture `i386` or `amd64`
- have a default locale with charset UTF-8

In case you want to migrate a Wazo from `i386` to `amd64`, see *Migrate Wazo from i386 (32 bits) to amd64 (64 bits)*.

Installation

Once you have your Debian stretch properly installed, download the Wazo installation script and make it executable:

```
wget http://mirror.wazo.community/fai/xivo-migration/wazo_install.sh
chmod +x wazo_install.sh
```

And run it:

```
./wazo_install.sh
```

At the end of the installation, you can continue by running the *configuration wizard.*

Alternatives versions

The installation script can also be used to install a specific version of Wazo. For example, if you want to install Wazo 16.16:

```
./wazo_install.sh -a 16.16
```

You may also install development versions of Wazo with this script. These versions may be unstable and should not be used on a production server. Please refer to the usage of the script:

```
./wazo_install.sh -h
```

Other installation methods

It's also possible to install Wazo by PXE. More details available on our blog:

- http://blog.wazo.community/around-xivo-describe-industrial-installation-process.html
- http://blog.wazo.community/around-xivo-pxe-server-setup.html
- http://blog.wazo.community/around-xivo-pxe-and-preseeding.html

1.2.2 Running the Wizard

After the system installation, you must go through the wizard before being able to use your Wazo. Open your browser and enter your server's IP address in the navigation bar. (For example: http://192.168.1.10)

Language

You first have to select the language you want to use for the wizard.

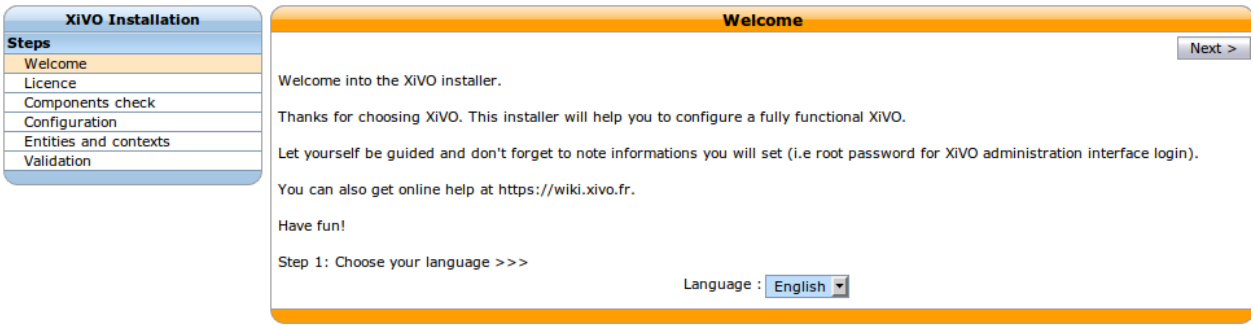


Fig. 1.1: Select the language

License

You then have to accept the *GPLv3 License* under which Wazo is distributed.

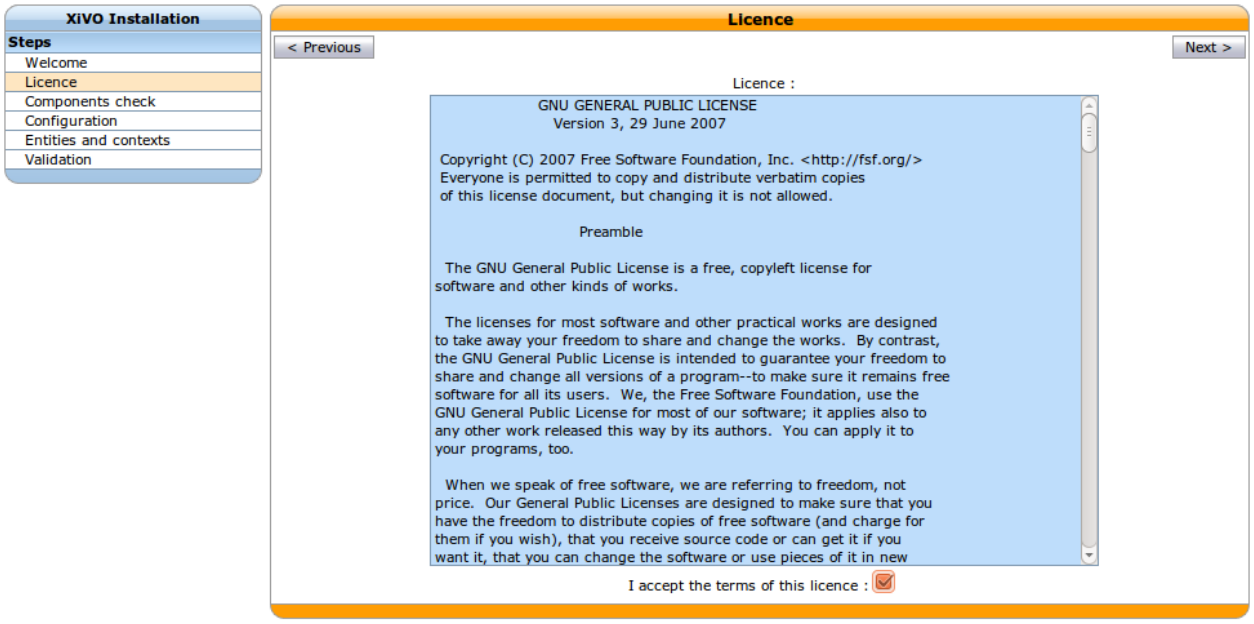


Fig. 1.2: Accept the license

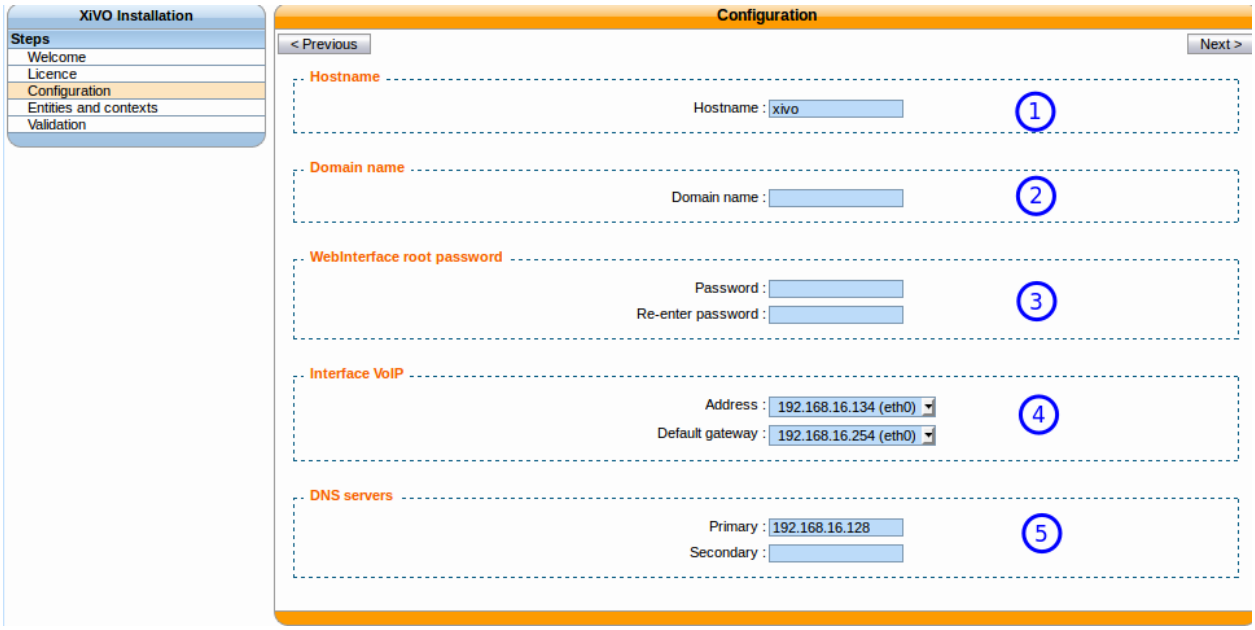


Fig. 1.3: Basic configuration

Configuration

1. Enter the hostname (Allowed characters are : `A-Z a-z 0-9 -`)
2. Enter the domain name (Allowed characters are : `A-Z a-z 0-9 - .`)
3. Enter the password for the `root` user of the web interface,
4. Configure the IP address and gateway used by the VoIP interface
5. Finally, modify the DNS server information if needed.

Entities and Contexts

Contexts are used for managing various phone numbers that are used by your system.

- The Internal calls context manages extension numbers that can be reached internally
- The Incalls context manages calls coming from outside of your system
- The Outcalls context manages calls going from your system to the outside

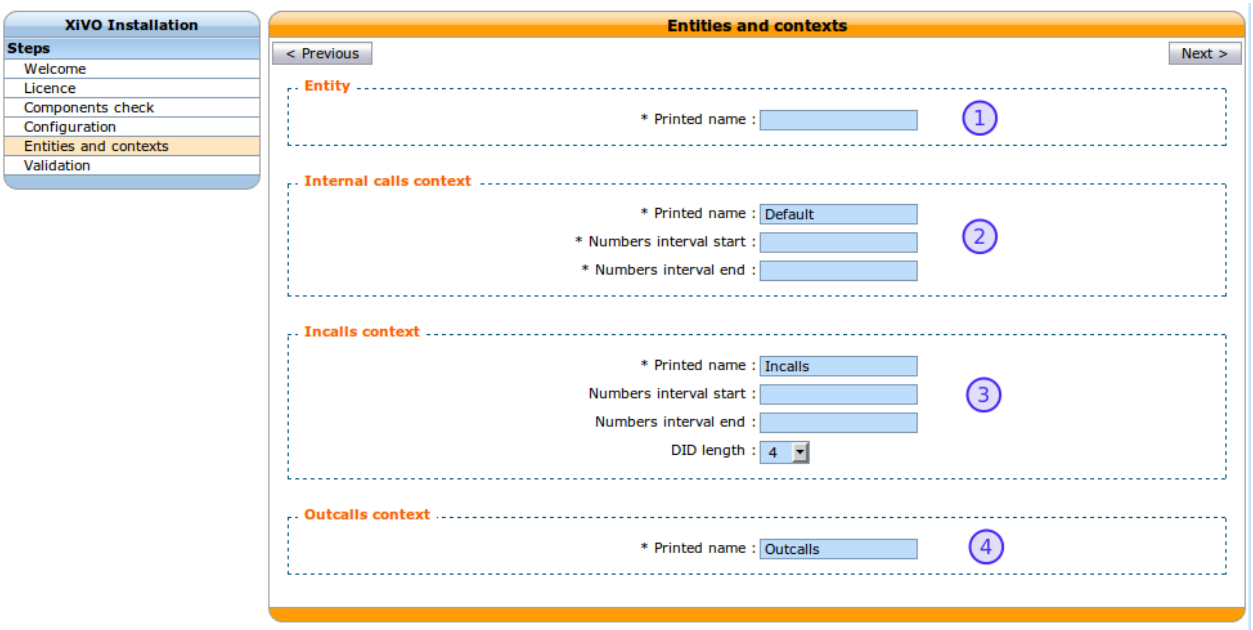


Fig. 1.4: Entities and Contexts

1. Enter the entity name (e.g. your organization name) (Allowed characters are : `A-Z a-z 0-9 - .`)
2. Enter the number interval for you internal context. The interval will define the users's phone numbers for your system (you can change it afterwards)
3. Enter the DID range and DID length for your system.
4. You may change the name of your outgoing calls context.

Validation

Finally, you can validate your configuration by clicking on the `Validate` button. Note that if you want to change one of the settings you can go backwards in the wizard by clicking on the `Previous` button.

Warning: This is the last time the `root` password will be displayed. Take care to note it.

Congratulations, you now have a fully functional Wazo server.

To start configuring Wazo, see *Getting Started*.

1.2.3 Post Installation

Here are a few configuration options that are commonly changed once the installation is completed. Please note that these changes are optional.

Display called name on internal calls

When you call internally another phone of the system you would like your phone to display the name of the called person (instead of the dialed number only). To achieve this you must change the following SIP options:

- *Services* → *IPBX* → *General settings* → *SIP Protocol* → *Default*:
  - Trust the Remote-Party-ID: yes,
  - Send the Remote-Party-ID: select `PAI`

Incoming caller number display

The caller ID number on incoming calls depends on what is sent by your operator. You can modify it via the file `/etc/xivo/asterisk/xivo_in_callerid.conf`.

Note: The reverse directory lookup use the caller ID number after it has been modified by `xivo_in_callerid.conf`

Examples:

- If you use a prefix to dial outgoing numbers (like a 0) you should add a 0 to all the `add =` sections,
- You may want to display incoming numbers in E.164 format. For example, you can change the `[national1]` section to:

```
callerid = ^0[1-9]\d{8}$
strip = 1
add = +33
```

To enable the changes you have to restart xivo-agid:

```
service xivo-agid restart
```

Time and date

- Configure your locale and default time zone device template => *Configuration* → *Provisioning* → *Template Device* by editing the default template
- Configure the timezone in => *Services* → *IPBX* → *General settings* → *Advanced* → *Timezone*

- If needed, reconfigure your timezone for the system:

```
dpkg-reconfigure tzdata
```

Codecs

You should also select default codecs. It obviously depends on the telco links, the country, the phones, the usage, etc. Here is a typical example for Europe (the main goal in this example is to select *only* G.711 A-Law instead of both G.711 A-Law and G.711 µ-Law by default):

- SIP : *Services* → *IPBX* → *General settings* → *SIP Protocol* → *Signaling*:
  - Customize codec : enabled
  - Codec list:

```
G.711 A-Law
G.722
G.729A
H.264
```

- IAX2 : *Services* → *IPBX* → *General settings* → *IAX Protocol* → *Default*:
  - Customize : enabled
  - Codec list:

```
G.711 A-Law
G.722
G.729A
H.264
```

1.3 Getting Started

This section will show you how to create a user with a SIP line. This simple use case covers what a lot of people need to start using a phone. You can use these steps for configuring a phone (e.g a softphone, an Analog-to-Digital switch or a SIP phone).

This tutorial doesn't cover how to automatically provision a *supported device*. For this, consult the *provisionning section.*

We first need to log into the Wazo web interface. The web interface is where you can administer the whole system.

When logged in, you will see a page with all the status information about your system. This page helps you monitor the health of your system and gives you information about your network. Please note the IP address of your server, you will need this information later on when you will configure your device (e.g. phone)

To configure a device for a user, start by navigating to the IPBX menu. Hover over the *Services* tab, a dropdown menu will appear. Click on *IPBX*.

Select the *Users* setting in the left menu.

From here, press on the "plus" sign. A pop up will appear where you can click on *Add*.

We now have the form that will allow us to create a new user. The three most important fields are 'First name', 'Last name' and 'Language'. Fill in the fields and click on *Save* at the bottom. For our example, we will create a used called 'Alice Wonderland'.



Fig. 1.5: Logging into the Wazo

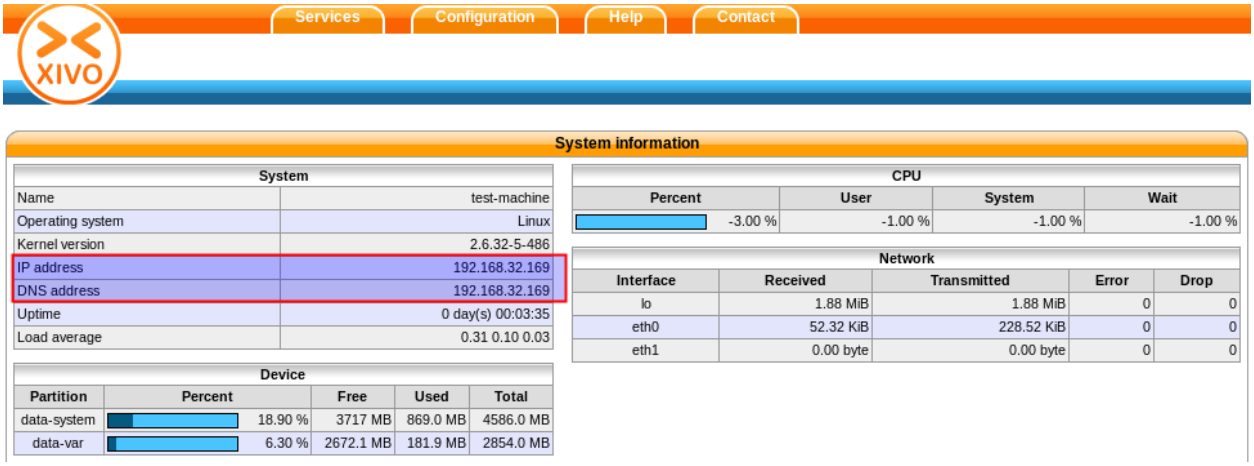


Fig. 1.6: System informations

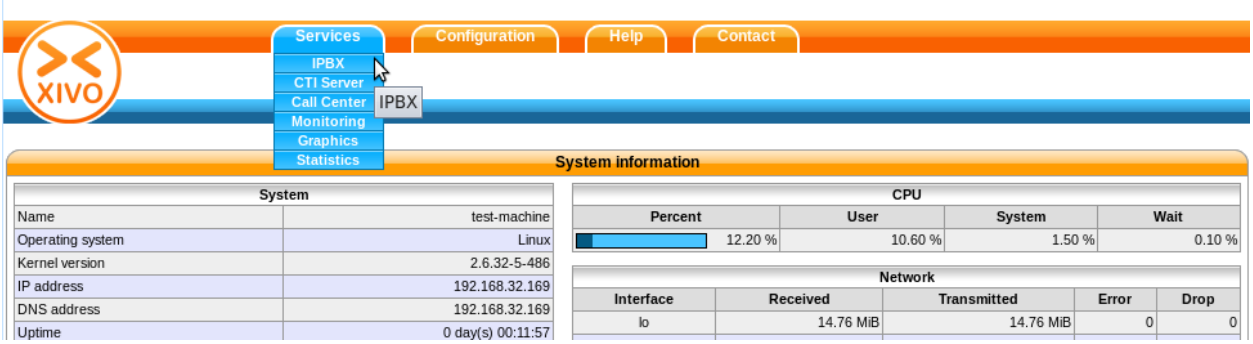


Fig. 1.7: Menu IPBX

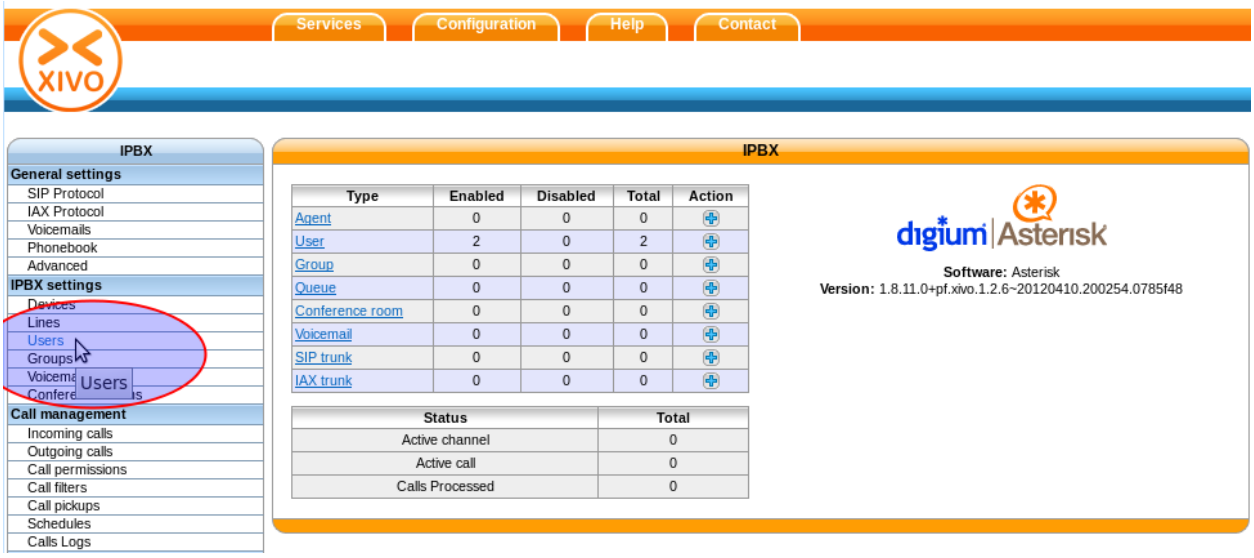


Fig. 1.8: Users settings



Fig. 1.9: Adding a new line

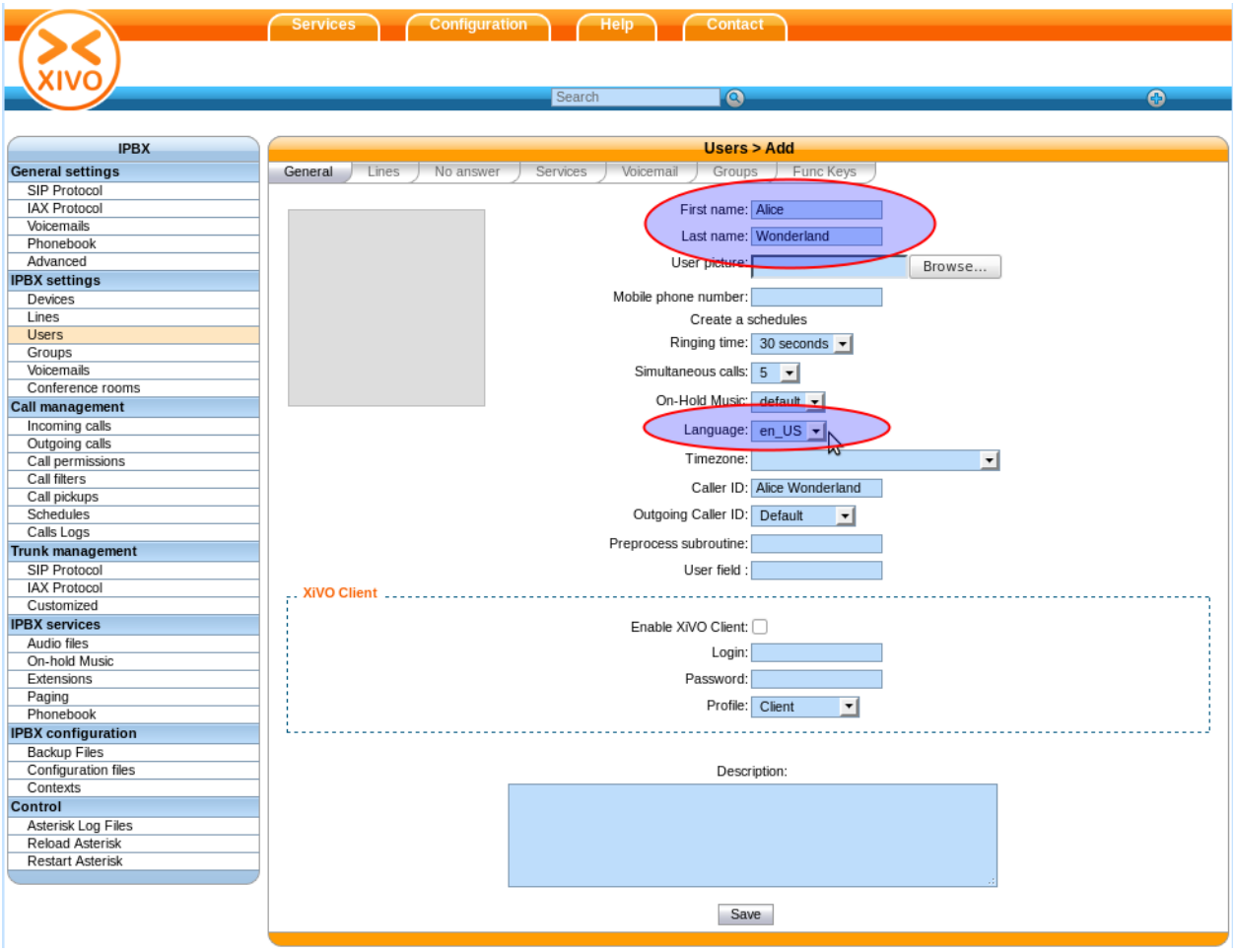


Fig. 1.10: User information

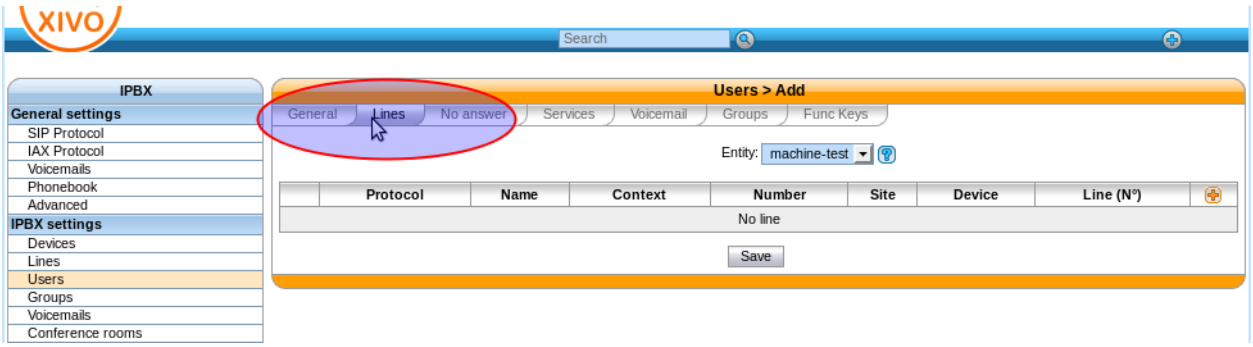


Fig. 1.11: Lines menu

Afterwards, click on the “Lines” tab.

Enter a number for your phone. If you click inside the field, you will see the range of numbers you can use. For our example, we will use ‘1000’.

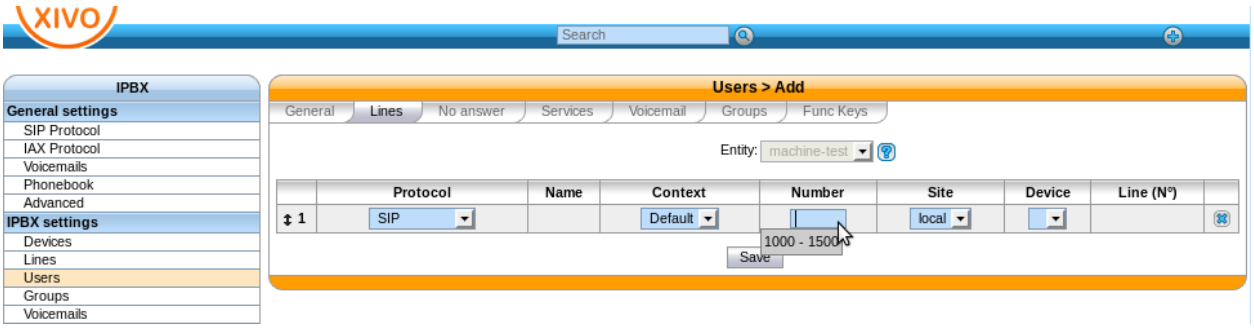


Fig. 1.12: Line information

By default, the selected protocol is SIP, which is what we want for now. Click on Save to create the line.

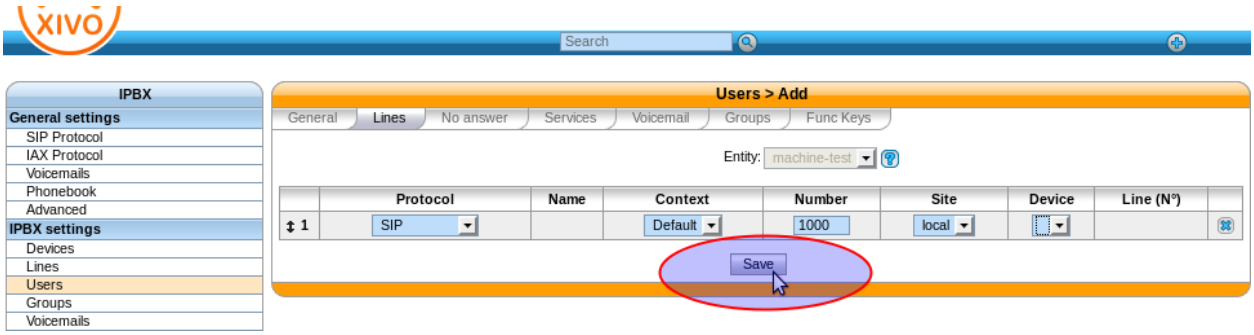


Fig. 1.13: Save

We now have a user named ‘Alice Wonderland’ with the phone number ‘1000’.

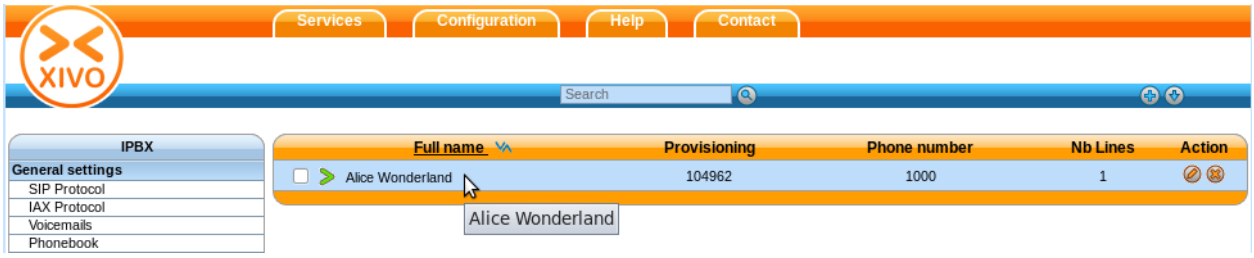


Fig. 1.14: User added information

Now we need to go get the SIP username and password to configure our phone. Go back to the IPBX menu on the left, and click on ‘Lines’.

You will see a line associated with the user we just created. Click on the pencil icon to edit the line.

We can now see the username and password for the SIP line. you can configure your phone using the IP for your server, the username and the password.

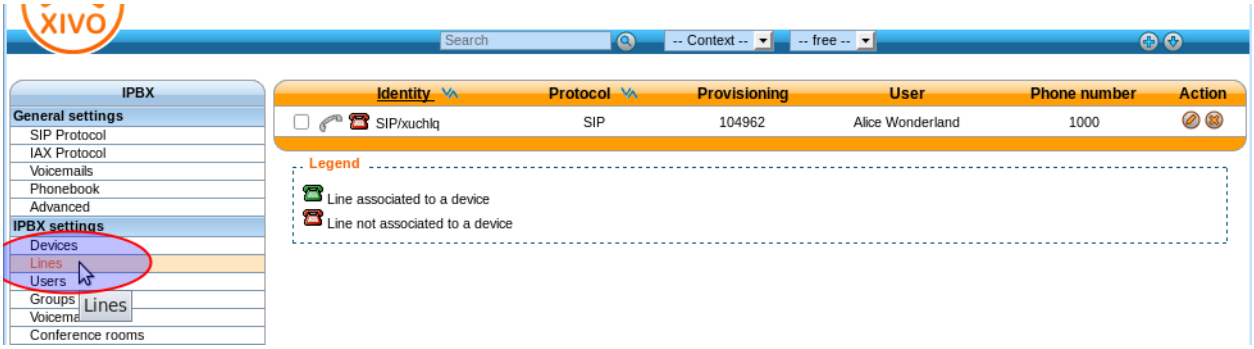


Fig. 1.15: Lines information



Fig. 1.16: Edit line

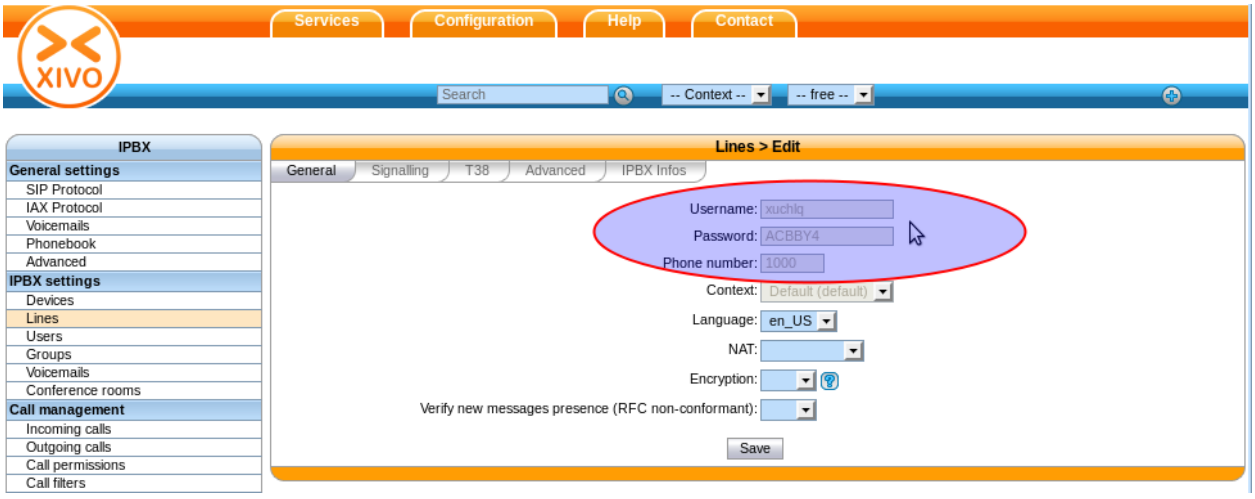


Fig. 1.17: General line information

1.4 Upgrading

Upgrading a Wazo is done by executing commands through a terminal on the server. You can connect to the server either through SSH or with a physical console.

To upgrade your Wazo to the latest version, you **must** use the `wazo-upgrade` script. You can start an upgrade with the command:

```
wazo-upgrade
```

Note:

- You can't use wazo-upgrade if you have not run the wizard yet
- Upgrading from a *deprecated version* is not supported.
- When upgrading Wazo, you **must** also upgrade **all** associated Wazo Clients. There is currently no retro-compatibility on older Wazo Client versions. The only exception is Wazo 16.16, which is compatible with Wazo Client 16.13.

This script will update Wazo and restart all services.

There are 2 options you can pass to wazo-upgrade:

- `-d` to only download packages without installing them. **This will still upgrade the package containing wazo-upgrade**.
- `-f` to force upgrade, without asking for user confirmation

`wazo-upgrade` uses the following environment variables:

- `XIVO_CONFD_PORT` to set the port used to query the *HTTP API of xivo-confd* (default is 9486)

1.4.1 Upgrade procedure

- Read all existing *Upgrade notes* starting from your version to the latest version.
- For custom setups, follow the required procedures described below (e.g. HA cluster).
- To download the packages beforehand, run `wazo-upgrade -d`. This is not mandatory, but it does not require stopping any service, so it may be useful to reduce the downtime of the server while upgrading.
- When ready, run `wazo-upgrade` which will start the upgrade process. **Telephony services will be stopped during the process**
- When finished, check that all services are running (the list is displayed at the end of the upgrade).
- Check that services are correctly working like SIP registration, ISDN link status, internal/incoming/outgoing calls, Wazo Client connections etc.

1.4.2 Version-specific upgrade procedures

Upgrading from XiVO 16.13 and before

When upgrading from XiVO 16.13 or before, you must use the special *XiVO to Wazo upgrade procedure* instead of simply running `xivo-upgrade`.

Upgrading from XiVO 14.01, 14.02, 14.03, 14.04 installed from the ISO

In those versions, xivo-upgrade keeps XiVO on the same version. You must do the following, before the normal upgrade:

```
echo "deb http://mirror.wazo.community/debian/ xivo-five main" > /etc/apt/sources.
↪list.d/xivo-upgrade.list \
&& apt-get update \
&& apt-get install xivo-fai \
&& rm /etc/apt/sources.list.d/xivo-upgrade.list \
&& apt-get update
```

Upgrading from XiVO 13.24 and before

When upgrading from XiVO 13.24 or earlier, you must do the following, before the normal upgrade:

1. Ensure that the file `/etc/apt/sources.list` is *not* configured on `archive.debian.org`. Instead, it must be configured with a non-archive mirror, but still on the `squeeze` distribution, even if it is not present on this mirror. For example:

   ```
   deb http://ftp.us.debian.org/debian squeeze main
   ```

2. Add `archive.debian.org` in another file:

   ```
   cat > /etc/apt/sources.list.d/squeeze-archive.list <<EOF
   deb http://archive.debian.org/debian/ squeeze main
   EOF
   ```

And after the upgrade:

```
rm /etc/apt/sources.list.d/squeeze-archive.list
```

Upgrading from XiVO 13.03 and before

When upgrading from XiVO 13.03 or earlier, you must do the following, before the normal upgrade:

```
wget http://mirror.wazo.community/xivo_current.key -O - | apt-key add -
```

1.4.3 Upgrading a cluster

Here are the steps for upgrading a cluster, i.e. two Wazo with *High Availability (HA)*:

1. On the master : deactivate the database replication by commenting the cron in `/etc/cron.d/xivo-ha-master`
2. On the slave, deactivate the xivo-check-master-status script cronjob by commenting the line in `/etc/cron.d/xivo-ha-slave`
3. On the slave, start the upgrade:

   ```
   xivo-slave:~$ wazo-upgrade
   ```

4. When the slave has finished, start the upgrade on the master:

```
xivo-master:~$ wazo-upgrade
```

5. When done, launch the database replication manually:

```
xivo-master:~$ xivo-master-slave-db-replication <slave ip>
```

6. Reactivate the cronjobs (see steps 1 and 2)

1.4.4 Upgrading to a specific version of Wazo

Upgrade to a specific version of Wazo

What is the point?

Sometimes, you may need to upgrade your Wazo to a specific version, in case you don't want to upgrade to the latest (which is not recommended, but sometimes necessary).

Prerequisites

Warning: These procedures are *complementary* to the upgrade procedure listed in *Version-specific upgrade procedures*. You must follow the version-specific procedure *before* running the following procedures.

Before starting the upgrade, you must have a xivo or Wazo version greater than 14.18.

Upgrade an older xivo installation

Those procedures are valid if your xivo installation is newer than 14.18 and older than 16.08.

Upgrade to Wazo < 18.01

Example upgrade to Wazo 17.02:

```
# --no-check-certificate is needed only if you are affected by http://projects.wazo.
↪community/issues/6024
wget --no-check-certificate https://raw.githubusercontent.com/wazo-pbx/xivo-upgrade/
↪master/bin/xivo-to-wazo-upgrade
chmod +x xivo-to-wazo-upgrade
XIVO_TO_WAZO_DEB_LINE="deb http://mirror.wazo.community/archive wazo-17.02 main" ./
↪xivo-to-wazo-upgrade
xivo-dist phoenix
```

Upgrade to Wazo >= 18.01

Example upgrade to Wazo 18.02:

```
# --no-check-certificate is needed only if you are affected by http://projects.wazo.
↪community/issues/6024
wget --no-check-certificate https://raw.githubusercontent.com/wazo-pbx/xivo-upgrade/
↪master/bin/xivo-to-wazo-upgrade
chmod +x xivo-to-wazo-upgrade
./xivo-to-wazo-upgrade
```

This will upgrade your xivo to Wazo 17.17. From there:

1. Read the *upgrade notes*
2. upgrade to Wazo 18.02:

```
wazo-dist-upgrade -t wazo-18.02
wazo-dist phoenix-stretch
```

My xivo is stuck in a specific version

Procedures for upgrading to specific versions may freeze the version of your xivo. Run the following commands to get the latest updates:

```
# --no-check-certificate is needed only if you are affected by http://projects.wazo.
↪community/issues/6024
wget --no-check-certificate https://raw.githubusercontent.com/wazo-pbx/xivo-upgrade/
↪master/bin/xivo-to-wazo-upgrade
chmod +x xivo-to-wazo-upgrade
./xivo-to-wazo-upgrade
```

Upgrade from Wazo < 18.01

Those procedures are valid if your Wazo installation is newer than 16.08 and older than 18.01.

Upgrade to Wazo < 18.01

Example to upgrade to Wazo 17.02:

```
xivo-dist wazo-17.02
apt-get update
apt-get install xivo-upgrade/wazo-17.02
wazo-upgrade
xivo-dist phoenix
```

Upgrade to Wazo >= 18.01

Example to upgrade to Wazo 18.02:

```
wazo-upgrade
```

This will upgrade your xivo to Wazo 17.17. From there:

1. Read the *upgrade notes*

2. upgrade to Wazo 18.02:

```
wazo-dist-upgrade -t wazo-18.02
wazo-dist phoenix-stretch
```

My Wazo is stuck in a specific version

Procedures for upgrading to specific versions may freeze the version of your xivo. Run the following commands to get the latest updates:

```
xivo-dist phoenix
wazo-upgrade
```

Upgrade from Wazo >= 18.01

Those procedures are valid if your Wazo installation is newer than 18.01.

Upgrade to specific Wazo

Example to upgrade to Wazo 18.02:

```
wazo-dist wazo-18.02
apt-get update
apt-get install xivo-upgrade/wazo-18.02
wazo-upgrade
wazo-dist phoenix-stretch
```

My Wazo is stuck in a specific version

Procedures for upgrading to specific versions may freeze the version of your Wazo. Run the following commands to get the latest updates:

```
wazo-dist phoenix-stretch
wazo-upgrade
```

1.4.5 Upgrading from i386 (32 bits) to amd64 (64 bits)

Migrate Wazo from `i386` (32 bits) to `amd64` (64 bits)

There is no fully automated method to migrate Wazo from `i386` to `amd64`.

The procedure is:

1. *Upgrade* your `i386` machine to XiVO/Wazo >= 15.13
2. *Install* a Wazo `amd64` **using the same version as the upgraded Wazo i386**
3. Make a backup of your Wazo `i386` by following the *backup procedure*
4. Copy the backup tarballs to the Wazo `amd64`
5. Restore the backup by following the *restore procedure*

Before starting the services after restoring the backup on the Wazo `amd64`, you should ensure that there won't be a conflict between the two machines, e.g. two DHCP servers on the same broadcast domain, or both Wazo fighting over the same SIP trunk register. You can disable the Wazo `i386` by running:

```
wazo-service stop
```

But be aware the Wazo `i386` will be enabled again after you reboot it.

1.4.6 Unsupported versions

Deprecated Wazo versions

General policy

On January 1st of every year, Wazo/XiVO versions that are more than 4 years old will be considered as deprecated.

Planned deprecation calendar:

Date	Deprecated versions
2017-01-01	older than 13.01
2018-01-01	older than 14.01
2019-01-01	older than 15.01
2020-01-01	older than 16.01
2021-01-01	older than 17.01

What does it mean to be in a deprecated version?

- A deprecated Wazo version does not have a supported upgrade path directly to the latest Wazo version. This means that running a straight `wazo-upgrade` is not guaranteed to succeed.
- Asking questions about a deprecated version (e.g. on the forum) will probably get the following answer: "get a newer version first, then come back and ask your question".
- Binaries (ISO images, CTI clients) for deprecated versions are not available for download.

Why are versions being deprecated?

- Hosting the binaries of older versions is costly and mostly useless: most people install the latest version of Wazo, and the very few cases where an old binary is needed is not worth the cost.
- Maintaining the upgrade machinery for older versions is time-consuming for developers: the more versions are supported by the upgrade, the more cases there are to handle; more cases make the code harder to read, understand and modify, bugs become more probable and the latest upgrades are more difficult to write.
- There are very few Wazo installed with older versions, as far as we can tell: all software should be upgraded frequently and Wazo is no exception. We consider 4 years to be a reasonable time range to upgrade at least once an IPBX. We do not want to hinder development for the very few who did not take the time to upgrade.

I have a deprecated version. What are my options?

There are two main options:

- upgrade to a Wazo version that is more recent, but not the latest: you can use the procedures listed in *Upgrade to a specific version of Wazo*.
- install a new server with the latest Wazo version, and reproduce your configuration by using the export/import features of Wazo and copying files

1.4.7 Troubleshooting

Postgresql

When upgrading Wazo, if you encounter problems related to the system locale, see *PostgreSQL localization errors*.

wazo-upgrade

If wazo-upgrade fails or aborts in mid-process, the system might end up in a faulty condition. If in doubt, run the following command to check the current state of xivo's firewall rules:

```
iptables -nvL
```

If, among others, it displays something like the following line (notice the DROP and 5060):

```
0     0 DROP       udp  --  *      *       0.0.0.0/0            0.0.0.0/0           ␣
↪udp dpt:5060
```

Then your Wazo will not be able to register any SIP phones. In this case, you must delete the DROP rules with the following command:

```
iptables -D INPUT -p udp --dport 5060 -j DROP
```

Repeat this command until no more unwanted rules are left.

1.4.8 Upgrade notes

Upgrade notes

18.03

- If you have a *custom certificate configured*, you will need to add a new symlink for wazo-upgrade:

```
mkdir -p /etc/wazo-upgrade/conf.d
ln -s "/etc/xivo/custom/custom-certificate.yml" "/etc/wazo-upgrade/conf.d/010-
↪custom-certificate.yml"
```

- Default passwords for phones' web interfaces have been changed. You can change the password in *Configuration → Provisioning → Template device*.
- The default NAT option in General SIP settings has been automatically changed from `auto_force_rport` to `auto_force_rport,auto_comedia`. This makes NAT configuration easier but has no impact on environments without NAT.
  - In the rare cases where you want to keep `nat=auto_force_rport` you must explicitly change this value in the administation interface *Services → IPBX → General Settings → SIP Protocol* in tab *Default*. See Asterisk sip.conf sample for more informations.

- The NAT configuration of every SIP line and SIP trunk has been automatically changed from `nat=auto_force_rport` to nothing, so that they inherit this setting from the General SIP settings.

Consult the 18.03 Roadmap for more information.

18.02

- For wazo-auth backend developers: The API to implement a wazo-auth backend has changed. Old implementations have to be updated. If the BaseAuthenticationBackend class was used as a base class for the backend the *get_metadata* method from the base class will use *get_ids* to generate the result of *get_metadata*.
  - The *get_ids* method has been remove.
  - The *get_metadata* method has been added.

Consult the 18.02 Roadmap for more information.

18.01

- **Debian has been upgraded from version 8 (jessie) to 9 (stretch).** Please consult the following detailed upgrade notes for more information:

Debian 9 (stretch) Upgrade Notes

The upgrade to Wazo 18.01 or later will take longer than usual, because the whole Debian system will be upgraded.

The database management system (postgresql) will also be upgraded from version 9.4 to version 9.6 at the same time. This will upgrade the database used by Wazo. This operation should take at most a few minutes.

After the upgrade, the system will need to be rebooted.

Before the upgrade

- Make sure your have sufficient space for the upgrade. You might run into trouble if you have less than 2 GiB available in the file system that holds the `/var` and `/` directories.
- If you have customized the Debian system of your XiVO in some nontrivial way, you might want to review the official Debian release notes before the upgrade. Most importantly, you should:
  - Make sure you don't have any unofficial sources in your `/etc/apt/sources.list` or `/etc/apt/sources.list.d` directory. If you were using the `jessie-backports` source, you must remove it.
  - Remove packages that were automatically installed and are not needed anymore, by running `apt-get autoremove --purge`.
  - Purge removed packages. You can see the list of packages in this state by running `dpkg -l | awk '/^rc/ { print $2 }'` and purge all of them with `apt-get purge $(dpkg -l | awk '/^rc/ { print $2 }')`
  - Remove `.dpkg-old`, `.dpkg-dist` and `.dpkg-new` files from previous upgrade. You can see a list of these files by running `find /etc -name '*.dpkg-old' -o -name '*.dpkg-dist' -o -name '*.dpkg-new'`.

Upgrade

The upgrade must be done with three commands:

- `xivo-dist phoenix`: Ensures your system is not restricted to a specific version
- `wazo-upgrade`: Installs the `wazo-dist-upgrade` script and makes sure the system is up-to-date.
- `wazo-dist-upgrade`: Upgrade to the latest version of Wazo with Debian 9 (stretch). This upgrade will take longer than usual.

You may need to reboot your machine before running `wazo-dist-upgrade`. `wazo-dist-upgrade` will tell you if a reboot is needed.

To minimize the downtime, you can pre-download the packages required for the upgrade with:

```
wazo-upgrade -d
wazo-dist-upgrade -d
```

After the upgrade

- Check that customization to your configuration files is still effective.

  During the upgrade, new version of configuration files are going to be installed, and these might override your local customization. For example, the vim package provides a new `/etc/vim/vimrc` file. If you have customized this file, after the upgrade you'll have both a `/etc/vim/vimrc` and `/etc/vim/vimrc.dpkg-old` file, the former containing the new version of the file shipped by the vim package while the later is your customized version. You should merge back your customization into the new file, then delete the `.dpkg-old` file.

  You can see a list of affected files by running `find /etc -name '*.dpkg-old'`. If some files show up that you didn't modify by yourself, you can ignore them.

- Purge removed packages. You can see the list of packages in this state by running `dpkg -l | awk '/^rc/ { print $2 }'` and purge all of them with `apt-get purge $(dpkg -l | awk '/^rc/ { print $2 }')`
- Reboot your system. It is necessary for the new Linux kernel to be effective.

Changes

Here's a non-exhaustive list of changes that comes with Wazo on Debian 9:

- **Network interface names (only for new installs, not upgrades)**: Debian Stretch uses the new standard naming scheme for network interfaces instead of `eth0`, `eth1`, etc. The new enumeration method relies on more sources of information, to produce a more repeatable outcome. It uses the firmware/BIOS provided index numbers and then tries PCI card slot numbers, producing names like `ens0` or `enp1s1`.

External Links

- Official Debian 9 release notes

- If you *did not* setup a custom X.509 certificate for HTTPS (e.g. from Let's Encrypt), the certificate will be regenerated to include SubjectAltName fields. The two main reasons are Chrome compatibility and avoiding a lot of log warnings. This implies that you will have to add a new exception in your browser to access the Wazo web interface or services like Unicom.
- If you *did* setup a custom X.509 certificate for HTTPS (e.g. from Let's Encrypt), you will have to add a link to the wazo-auth-cli configuration using the following command.

```
ln -s "/etc/xivo/custom/custom-certificate.yml" "/etc/wazo-auth-cli/conf.d/010-
↪custom-certificate.yml"
```

- The Python API for xivo-confd plugins has been updated to reflect Python API of other daemons. If you have created a custom xivo-confd plugin, you must update it:

Listing 1.1: plugin.old.py

```
class Plugin(object):

   def load(self, core):
       api = core.api
       config = core.config
```

Listing 1.2: plugin.py

```
class Plugin(object):

   def load(self, dependencies):
       api = dependencies['api']
       config = dependencies['config']
```

- The web interface no longer validates the queue skill rules fields added in *Services* → *Call Center* → *Configuration* → *Skill rules*. If a rule is wrong, it will appear in the Asterisk console.

Consult the 18.01 Roadmap for more information.

17.17

- The default NAT option has changed from `no` to `auto_force_rport`. This makes NAT configuration easier but has no impact on environments without NAT.
  - In the rare cases where you want to keep `nat=no` you must explicitly change this value in the administation interface *Services* → *IPBX* → *General Settings* → *SIP Protocol* in tab *Default*. See Asterisk sip.conf sample for more informations.
- The `sources` section of the `xivo-dird` service configuration has been changed to be a key-value setting.
  - If you have configured directories manually in `/etc/xivo-dird` you should update your manual configuration:

Listing 1.3: old.yml

```
services:
  lookup:
    default:
      sources:
        - source_one
        - source_two
      timeout: 2
```

Listing 1.4: new.yml

```
services:
  lookup:
    default:
      sources:
        source_one: true
        source_two: true
      timeout: 2
```

- The `enabled_plugins` section of the `xivo-confd` service configuration has been changed. If you have configured enabled plugins manually you should update your manual configuration
  - This section is now a key-value setting.
  - All plugins have been renamed without the suffix `_plugins`.

Listing 1.5: old.yml

```
enabled_plugins:
  - user_plugin
  - conference_plugin
```

Listing 1.6: new.yml

```
enabled_plugins:
  user: true
  conference: true
```

- There is a new `channelvars` option in `/etc/asterisk/manager.d/99-general.conf`. If you have manually configured `channelvars` already, you will have to manually merge the Wazo version with your version for them to work together.

Consult the 17.17 Roadmap for more information.

17.16

- You must update the Wazo Client to 17.16. See the *Wazo Client compatibility table*.
- The *enabled_plugins* section of the `wazo-auth` service has been renamed *enabled_backend_plugins* and is now a dictionary.
  - If you have hand made configuration to modify the list of enabled backends it should be modified see `/etc/wazo-auth/config.yml`
- The *ldap_user* backend in `wazo-auth` is now disabled in the base configuration file.
  - If you are using the `ldap_user` authentication backend a file with the following content should be added to `/etc/wazo-auth/conf.d`

    ```
    enabled_backend_plugins:
      ldap_user: true
    ```

- The *enabled_plugins* section of the `xivo-dird` service is now a dictionary.
  - If you have hand made configuration to modify the list of enabled plugins, it should be modified see `/etc/xivo-dird/config.yml`

- wazo-admin-ui has been upgraded to python3. All plugins by *Wazo Team* has been migrated, but if you have installed a non-official/custom plugin that add something to the new interface, it probably broken. To fix this, you must convert your plugin to python3 or wait an available upgrade from the maintainer.
- If you have setup a custom X.509 certificate for HTTPS (e.g. from Let's Encrypt), you have to update your config in `/etc/xivo/custom/custom-certificate.yml`, according to the *updated documentation*, namely for the config regarding `websocketd`.

Consult the 17.16 Roadmap for more information.

17.15

- `xivo-call-logd` has been renamed `wazo-call-logd`
  - The custom configuration has been moved to `/etc/wazo-call-logd/conf.d/`.
  - The log file has been renamed to `wazo-call-logd.log`.
  - The NGINX proxy has been recreated in `/etc/nginx/locations/https-enabled/wazo-call-logd`
- `Asterisk` has been upgraded to version 15.0.0
  - If you have installed asterisk modules manually, you will have to install the asterisk 15 version, otherwise Asterisk will crash when starting.

Consult the 17.15 Roadmap for more information.

17.14

- `xivo-auth` has been renamed `wazo-auth`
  - If you have developed a `xivo-auth` authentication backend the name of the entry point has changed to `wazo_auth.backends`. You should make this modification in your plugin's `setup.py` file in the `entry_point` section.
  - If your custom development use service discovery to find `xivo-auth`, you will have to search for the `wazo-auth` service instead of `xivo-auth`.
- We released a new version of the CTI client, rebranded as *Wazo Client 17.14.1*. It is compatible with all previous versions of Wazo (i.e. not before 16.16). See also the *compatibility table*.

Consult the 17.14 Roadmap for more information.

17.13

Consult the 17.13 Roadmap for more information.

17.12

- Wazo has a new database named `mongooseim`. The *backup-restore procedure* has been updated to include this new database.

Consult the 17.12 Roadmap for more information.

17.11

- wazo-plugind REST API version `0.1` has been deprecated and will be removed in Wazo `18.02`. See changelog for version *REST API changelog*

Consult the 17.11 Roadmap for more information.

17.10

Consult the 17.10 Roadmap for more information.

17.09

- Codecs can now be customized in the */etc/asterisk/codecs.d/* directory. If you had custom configuration in */etc/asterisk/codecs.conf* you will have to create a new file in *codecs.d* to use your customized configuration. A file named *codecs.conf.dpkg-old* will be left in */etc/asterisk* if this operation is required.
- Provd plugins from the addons repository have been merged into the main plugin repository. If you were using the addons repository you can safely switch back to the stable repository. See *Alternative plugins repository* for more details.
- The command `xivo-call-logs` has been deprecated in favor of `wazo-call-logs`.
- The command `xivo-service` has been deprecated in favor of `wazo-service`.
- If you have a *custom certificate configured*, you will need to add a new symlink for the new daemon wazo-webhookd:

```
ln -s "/etc/xivo/custom/custom-certificate.yml" "/etc/wazo-webhookd/conf.d/010-
↪custom-certificate.yml"
```

Consult the 17.09 Roadmap for more information.

17.08

- The call logs has been improved by adding `date_end` and `date_answer` informations. If you want to add these new informations to the old call logs, you need to regenerate them. For example, to regenerate the last month of call logs:

```
xivo-call-logs delete -d 30
xivo-call-logs generate -d 30
```

  This is only useful if you plan to use the call logs REST API to read calls that have been placed before the upgrade.

- If you have setup a custom X.509 certificate for HTTPS (e.g. from Let's Encrypt), you have to update your config in `/etc/xivo/custom/custom-certificate.yml`, according to the *updated documentation*, namely for the config regarding `plugind`.

Consult the 17.08 Roadmap for more information.

17.07

Consult the 17.07 Roadmap for more information.

17.06

- Upgrade from version older than 13.01 are not supported anymore.

Consult the 17.06 Roadmap for more information.

17.05

- *python-flask-cors* has been updated from 1.10.3 to 3.0.2. Configuration files with custom *allow_headers* will have to be updated to the new syntax. The following command can be used to see if you have a configuration file which needs to be updated.

```
for f in $(find /etc/*/conf.d -name '*.yml'); do grep -H allow_headers $f; done
```

  The old config in `/etc/xivo-*/conf.d` looked like:

```
rest_api:
  cors:
    allow_headers: Content-Type, X-Auth-Token
```

  The new config in `/etc/xivo-*/conf.d` looks like:

```
rest_api:
  cors:
    allow_headers: ["Content-Type", "X-Auth-Token"]
```

  See also the reference ticket #6617.

Consult the 17.05 Roadmap for more information.

17.04

Consult the 17.04 Roadmap for more information.

17.03

Consult the 17.03 Roadmap for more information.

17.02

- A few more services are now available by default on port TCP/443 (the complete list is documented in the *Nginx* section). This does not pose any additional security risk by default, but if you have extra strict requirements about security, they can be manually disabled.

Consult the 17.02 Roadmap for more information.

17.01

Consult the 17.01 Roadmap for more information.

16.16

Wazo 16.16 is the *first public release* of the project under the Wazo name. It is also the first release of Wazo under the "phoenix" codename.

- A *special procedure* is required to upgrade from XiVO to Wazo.
- Asterisk has been upgraded from version 13.11.2 to 14.2.1, which is a major Asterisk upgrade.
- If you are using *custom sheets* that are stored locally, they *must* now be readable by the system user `xivo-ctid`. Make sure that this user has read access to the UI file of your custom sheets.
- Switchboard statistics have been removed. The existing statistics data remain in the database for later migration but no more statistics will be collected.
- The `conference` destination type in incalls REST API has been renamed to `meetme`.
- The phonebook has been migrated from the web interface to xivo-dird. The phonebook contacts from the web interface have been moved to new dird-phonebooks. For users with many entities on the same Wazo, this will create one phonebook for each entity. The configuration has been updated to keep the previous behavior. No manual actions are required for installations with only one entity or if one phonebook by entity is the desired configuration. If only one phonebook is desired for all entities, some of the duplicate phonebooks can be deleted from the web interface and their matching configuration can also be removed.
  - The list of phonebooks can be modified in *Services* → *IPBX* → *IPBX services* → *Phonebook*
  - The list of phonebooks sources can be modified in:
    - *Configuration* → *Management* → *Directories*
    - *Services* → *CTI Server* → *Directories* → *Definitions*
  - The selected phonebooks for reverse lookups can be modified in *Services* → *CTI Server* → *Directories* → *Reverse directories*
  - Direct directories can be modified in *Services* → *CTI Server* → *Directories* → *Direct directories*

Please consult the following detailed upgrade notes for more information:

XiVO to Wazo Upgrade Notes

The Wazo project is a continuation of the original XiVO project. Programs, filenames, packages, plugins, etc, still use the "xivo" name as to not break backward compatibility. In this regard, upgrading from XiVO 16.13 to Wazo 16.16 is not different from upgrading XiVO 16.10 to XiVO 16.13, for example.

More information about the Wazo project is available on the Wazo blog.

Using the Wazo Infrastructure on your XiVO

Since `*.xivo.io` has been shut down, you may use the infrastructure at `*.wazo.community` instead of `*.xivo.io`. This step is only needed if you **don't intend to upgrade to Wazo** right away, i.e. you want to continue using your XiVO installation in its current version for some time. The features needing `*.xivo.io` are:

- installing new provisioning plugins
- keeping your DHCP configuration up-to-date (for new phones OUI, for example)
- upgrading to XiVO <= 16.13, i.e. not Wazo

In this case, you'll need to run the following commands:

```
# --no-check-certificate is needed only if you are affected by http://projects.wazo.
↪community/issues/6024
wget --no-check-certificate https://raw.githubusercontent.com/wazo-pbx/xivo-upgrade/
↪master/bin/use-wazo-infrastructure
chmod +x use-wazo-infrastructure
./use-wazo-infrastructure
```

The `use-wazo-infrastructure` script adds lines to the `/etc/hosts` file such that hostnames that used to refer to the infrastructure of the XiVO project (e.g. mirror.xivo.io) now points to the infrastructure of the Wazo project (e.g. mirror.wazo.community).

The script can be run multiple times. If you want to revert the modification done by the script, just execute it with the `--revert` option.

This script is compatible with any future upgrade, you don't have to revert it manually.

Upgrading to Wazo

To upgrade your XiVO to Wazo, run the following commands:

```
# --no-check-certificate is needed only if you are affected by http://projects.wazo.
↪community/issues/6024
wget --no-check-certificate https://raw.githubusercontent.com/wazo-pbx/xivo-upgrade/
↪master/bin/xivo-to-wazo-upgrade
chmod +x xivo-to-wazo-upgrade
./xivo-to-wazo-upgrade
```

After the Upgrade

You should make sure that you don't have any reference left to the xivo.io domain on your Wazo. In particular, you should check the `/etc` directory with the command:

```
grep -rF xivo.io /etc
```

There is no release of the Wazo Client 16.16, but Wazo 16.16 is compatible with the Wazo Client 16.13.

Asterisk 13 to 14 Upgrade Notes

You might be impacted by the upgrade to Asterisk 14 if you have:

- custom Asterisk configuration (other than custom dialplan)
- custom application using AMI or ARI
- custom Asterisk modules (e.g. codec_g729a.so)

If you find yourself in one of these cases, you should make sure that your customizations still work with Asterisk 14.

In particular, if you are using custom Asterisk modules, you'll need to either obtain the Asterisk 14 version of these modules or recompile them against Asterisk 14. Not doing so usually leads to major instability issues in Asterisk.

If you are upgrading from Asterisk 11, you should also check the *Asterisk 11 to 13 upgrade notes*.

Changes Between Asterisk 13 and 14

Some of the more common changes to look for:

- AMI: The Command action now sends the output from the CLI command as a series of Output headers for each line instead of as a block of text with the `--END COMMAND--` delimiter to match the output from other actions.

You can see the complete list of changes from the Asterisk website:

- https://wiki.asterisk.org/wiki/display/AST/Upgrading+to+Asterisk+14
- https://github.com/asterisk/asterisk/blob/14/CHANGES

Consult the 16.16 Roadmap for more information.

16.13

XiVO 16.13 is the *last public release* of the project under the name XiVO.

- Previously, a user's DND (Do Not Distrub) was effective only if this user had DND enabled *and* the DND extension (*25 by default) was also enabled. Said differently, disabling the DND extension meant that no user could effectively be in DND. Starting from XiVO 16.13, a user's DND is effective regardless of the state of the DND extension. The following features are impacted in the same way: call recording, incoming call filtering, forward on non-answer, forward on busy and unconditional forward.
- If you have manually added nginx configuration files to the `/etc/nginx/locations/http` directory, you'll need to move these files to `/etc/nginx/locations/http-available` and then create symlinks to them in the `/etc/nginx/locations/http-enabled` directory. This also applies to the https directory. See *Nginx*.
- A regression has been introduced in the switchboard statistics. See issue 6443.

Consult the 16.13 Roadmap for more information.

16.12

Consult the 16.12 Roadmap for more information.

16.11

- Fax reception: the "log" backend type has been removed. You should remove references to it in your `/etc/xivo/asterisk/xivo_fax.conf` if you were using it. Now, every time a fax is processed, a log line is added to `/var/log/xivo-agid.log`.

Consult the 16.11 Roadmap for more information.

16.10

- The config file `/etc/xivo/xivo-confgend.conf` has been replaced with `/etc/xivo-confgend/config.yml` and `/etc/xivo-confgend/conf.d`. Custom modifications to this file are not migrated automatically, so manual intervention is required to migrate custom values to the `conf.d` directory. The file `/etc/xivo/xivo-confgend/asterisk/contexts.conf` has been moved to `/etc/xivo-confgend/templates/contexts.conf`, but custom modification are left untouched. See also *Configuration Files* for more details about configuration files in XiVO.

Consult the 16.10 Roadmap for more information.

16.09

- The Wazo Client now uses xivo-ctid-ng to do transfers. Those new transfers cannot be cancelled with the `*0` DTMF sequence and there is no interface in the Wazo Client to cancel a transfer for profiles other than the switchboard (bug #6321). This will be addressed in a later version.
- Transfers started from the Wazo Client do not respect the `Dial timeout on transfer` option anymore (bug #6322). This feature will be reintroduced in a later version.

Consult the 16.09 Roadmap for more information.

16.08

- *CTI Protocol* is now in version *2.2*
- Some *security features have been added to the XiVO provisioning server*. To benefit from these new features, you'll need to *update your xivo-provd plugins to meet the system requirements*.

  If you have many phones that are connected to your XiVO through a NAT equipment, you should review the default configuration to make sure that the IP address of your NAT equipment don't get banned unintentionally by your XiVO.
- Newly created groups and queues now ignore call forward requests from members by default. Previously, call forward requests from members were always followed. This only applies to call forward configured directly on the member's phone: call forward configured via *21 have always been ignored in these cases.

  Note that during the upgrade, the previous behaviour is kept for already existing queues and groups.

  This behaviour is now configurable per queue/group, via the "Ignore call forward requests from members" option under the "Application" tab. We recommend enabling this option.

Consult the 16.08 Roadmap for more information.

16.07

- If you were affected by the bug #6213, i.e. if your agent login time statistics were incorrect since your upgrade to XiVO 15.20 or later, and you want to fix your statistics for that period of time, you'll need to manually apply a fix.

Consult the 16.07 Roadmap for more information.

16.06

Consult the 16.06 Roadmap for more information.

16.05

- The `view`, `add`, `edit`, `delete` and `deleteall` actions of the "lines" web service provided by the web interface have been removed. As a reminder, note that the web services provided by the web interface are deprecated.

Consult the 16.05 Roadmap for more information.

16.04

- *CTI Protocol* is now in version *2.1*
- The field *Rightcall Code* from *Services -> IPBX -> IPBX Settings -> Users* under *Services* tab will overwrite all password call permissions for the user.
- Faxes stored on FTP servers are now converted to PDF by default. See *Using the FTP backend* if you want to keep the old behavior of storing faxes as TIFF files.

Consult the 16.04 Roadmap for more information.

16.03

- The new section *Services* → *Statistics* → *Switchboard* in the web interface will only be visible by a non-root administrator after adding the corresponding permissions in the administrator configuration.
- Update the switchboard configuration page for the statistics in *Configuration for multiple switchboards*.
- The API for associating a line to a device has been replaced. Consult the *xivo-confd changelog* for further details
- The configuration parameters of *xivo_ldap_user* plugin of *xivo-auth* has been changed. See *xivo_ldap plugin*.
- The user's email is now a unique constraint. Every duplicate email will be deleted during the migration. (This does not apply to the voicemail's email)

Consult the 16.03 Roadmap for more information.

16.02

- The experimental *xivo_ldap_voicemail* plugin of *xivo-auth* has been removed. Use the new *xivo_ldap plugin*.
- Bus messages in the *xivo* exchange are now sent with the content-type *application/json*. Some libraries already do the message conversion based the content-type. Kombu users will receive a python dictionnary instead of a string containing json when a message is received.
- *xivo-ctid encryption* is automatically switched on for every XiVO server and Wazo Client >= 16.02. If you really don't want encryption, you must disable it manually on the server after the upgrade. In that case, Wazo Clients will ask whether to accept the connection the first time.

Consult the 16.02 Roadmap for more information.

16.01

- The page *Configuration* → *Management* → *Web Services Access* → *Acces rights* has been removed. Consequently, every Web Services Access has now all access rights on the web services provided by the web interface. These web services are deprecated and will be removed soon.
- During the upgrade, if no CA certificates were trusted at the system level, all the CA certificates from the ca-certificates package will be added. This is done to resolve an issue with installations from the ISO and PXE. In the (rare) case you manually configured the ca-certificates package to trust no CA certificates at all, you'll need to manually reconfigure it via `dpkg-reconfigure ca-certificates` after the upgrade.
- *xivo-ctid* uses *xivo-auth* to authenticate users. See *Authentication*.
- the *service_discovery* section of the *xivo-ctid* configuration has changed. If you have set up *Contact and Presence Sharing*, you should update your xivo-ctid configuration.

- the *CTI Protocol* is now versioned and a message will be displayed if the server and a client have incompatible protocol versions.

Consult the 16.01 Roadmap for more information.

Archives

Archived Upgrade Notes

2015

15.20

Consult the 15.20 Roadmap

- Debian has been upgraded from version 7 (wheezy) to 8 (jessie).
- CSV webservices in the web interface have been removed. Please use the *xivo-confd REST API* instead.
- The CSV import format has been changed. Consult *CSV Migration* for further details.
- xivo-ctid now uses STARTTLS for the client connections.
  - For users already using the CTIS protocol the client can be configured to use the default port (5003)

Please consult the following detailed upgrade notes for more information:

Debian 8 (jessie) Upgrade Notes

The upgrade to XiVO 15.20 or later will take longer than usual, because the whole Debian system will be upgraded.

The database management system (postgresql) will also be upgraded from version 9.1 to version 9.4 at the same time. This will upgrade the database used by XiVO. This operation should take at most a few minutes.

After the upgrade, the system will need to be rebooted.

Before the upgrade

- If you are upgrading from XiVO 13.24 or earlier, you'll need to first upgrade to Debian 7 (wheezy) before being able to upgrade to Debian 8 (jessie). To do so, you'll have to:
  - Run `xivo-upgrade` a first time, which will upgrade your XiVO to version 15.19 (Debian 7)
  - Reboot your system
  - Run `xivo-upgrade` a second time, which will upgrade your XiVO to the latest version (Debian 8)
  - Reboot your system

  Consult the *Debian 7 (wheezy) Upgrade Notes* for more information on the first upgrade.
- Make sure your have sufficient space for the upgrade. You might run into trouble if you have less than 2 GiB available in the file system that holds the /var and / directories.
- If you have customized the Debian system of your XiVO in some nontrivial way, you might want to review the official Debian release notes before the upgrade. Most importantly, you should:

  - Make sure you don't have any unofficial sources in your /etc/apt/sources.list or /etc/apt/sources.list.d directory. If you were using the wheezy-backports source, you must remove it.
  - Remove packages that were automatically installed and are not needed anymore, by running `apt-get autoremove --purge`.
  - Purge removed packages. You can see the list of packages in this state by running `dpkg -l | awk '/^rc/ { print $2 }'` and purge all of them with `apt-get purge $(dpkg -l | awk '/^rc/ { print $2 }')`
  - Remove `.dpkg-old`, `.dpkg-dist` and `.dpkg-new` files from previous upgrade. You can see a list of these files by running `find /etc -name '*.dpkg-old' -o -name '*.dpkg-dist' -o -name '*.dpkg-new'`.

After the upgrade

- Check that customization to your configuration files is still effective.

  During the upgrade, new version of configuration files are going to be installed, and these might override your local customization. For example, the vim package provides a new `/etc/vim/vimrc` file. If you have customized this file, after the upgrade you'll have both a `/etc/vim/vimrc` and `/etc/vim/vimrc.dpkg-old` file, the former containing the new version of the file shipped by the vim package while the later is your customized version. You should merge back your customization into the new file, then delete the `.dpkg-old` file.

  You can see a list of affected files by running `find /etc -name '*.dpkg-old'`. If some files shows up that you didn't modify by yourself, you can ignore them.

- Purge removed packages. You can see the list of packages in this state by running `dpkg -l | awk '/^rc/ { print $2 }'` and purge all of them with `apt-get purge $(dpkg -l | awk '/^rc/ { print $2 }')`
- If you had customizations in one of these files:
  - `/etc/default/asterisk`
  - `/etc/default/consul`
  - `/etc/default/xivo-ctid`

  Then you'll need to review your customizations to make sure they still work with systemd. This is necessary since these 3 files aren't read under systemd.

  For `/etc/default/asterisk`, only the CONFD_* options are automatically migrated to `/etc/systemd/system/asterisk.service.d/auto-sysv-migration.conf`.

  For `/etc/default/consul`, only the WAIT_FOR_LEADER and CONFIG_DIR options are automatically migrated to `/etc/systemd/system/consul.service.d/auto-sysv-migration.conf`.

  For `/etc/default/xivo-ctid`, only the XIVO_CTID_AMI_PROXY option is automatically migrated to `/etc/systemd/system/xivo-ctid.service.d/auto-sysv-migration.conf`.

- Reboot your system. It is necessary for the upgrade to the Linux kernel and init system (systemd) to be effective.

Changes

Here's a non-exhaustive list of changes that comes with XiVO on Debian 8:

- In Debian 7, the `halt` command powered off the machine. In Debian 8, the command halts the system, but does not power off the machine. To halt the machine and turn it off, use the `poweroff` or `shutdown` command.

- With the init system switch from SysV to systemd, you should now use the `systemctl` command to manage services (i.e. start/stop/status) instead of the `service` command or `/etc/init.d/<service>`, although these two methods should still work fine.

  If you are new to systemd, you can find some basic usage on the systemd page of the Debian Wiki.

- The bootlogd package is not installed by default anymore, since it is not needed with systemd. If you want to see the boot messages, use the `journalctl -b` command instead.
- The virtual terminals (tty1 to tty6) now shows up earlier during the boot, before all services have been started.
- The way the *ami-proxy is configured* for xivo-ctid has changed. If your XiVO was using the ami-proxy, the configuration will be automatically upgraded.
- Customization to asterisk and consul startup is now done by customizing the systemd unit file (by creating a drop-in file for example) instead of editing the `/etc/default/asterisk` and `/etc/default/consul` files. These files are not used anymore.

List of Known Bugs And Limitations

- If your system is using a swap partition or file and is using more memory than it can fit in the RAM, then system power-off or reboot might hangs indefinitely. This is due to a limitation in the current systemd version.

  If you find yourself in this case, you should try allocating more RAM to your system. Otherwise, you can try stopping the xivo services using `wazo-service stop` before rebooting to lessen the likelihood of this problem.

  See http://projects.wazo.community/issues/6016

External Links

- Official Debian 8 release notes

CSV Migration

This page describes how to migrate CSV files from the legacy format to the new format. Consult the API documentation on user imports for further details.

CSV Data

- Only data encoded as UTF-8 will be accepted
- The pipe delimiter (`|`) has been replaced by a comma (`,`)
- Double-quotes (`"`) must be escaped by writing them twice (e.g `Robert ""Bob"" Jenkins`)

Field names

Fields have been renamed in the new CSV format. Use the following table to rename your fields. Fields marked as **N/A** are no longer supported.

Old name	New name
entityid	entity_id
firstname	firstname
lastname	lastname
language	language
outcallerid	outgoing_caller_id
mobilephonenumber	mobile_phone_number
agentnumber	**N/A**
bosssecretary	**N/A**
callerid	**N/A**
enablehint	supervision_enabled
enablexfer	call_transfer_enabled
enableclient	cti_profile_enabled
profileclient	cti_profile_name
username	username
password	password
phonenumber	exten
context	context
protocol	line_protocol
linename	sip_username
linesecret	sip_secret
incallexten	incall_exten
incallcontext	incall_context
incallringseconds	incall_ring_seconds
voicemailname	voicemail_name
voicemailnumber	voicemail_number
voicemailcontext	voicemail_context
voicemailpassword	voicemail_password
voicemailemail	voicemail_email
voicemailattach	voicemail_attach_audio
voicemaildelete	voicemail_delete_messages
voicemailaskpassword	voicemail_ask_password

15.19

Consult the 15.19 Roadmap

- The sound file `/usr/share/asterisk/sounds/fr_FR/une.wav` has been moved to `/usr/share/asterisk/sounds/fr_FR/digits/1F.wav`.
- If you would like to use the new "transfer to voicemail" feature from the People Xlet, you'll need to update your directory definition and your directory display, i.e.:
  - edit your "internal" directory definition (Services / CTI server / Directories / Definitions) and add a field "voicemail" with value "voicemail_number"
  - edit your display (Services / CTI server / Directories / Display filters) and add a row with title "Voicemail", field type "voicemail" and field name "voicemail"
  - restart xivo-dird
- It is now possible to send an email to a user with a configured email address in the *people* xlet. See *Views* to add the appropriate field to your configured displays.

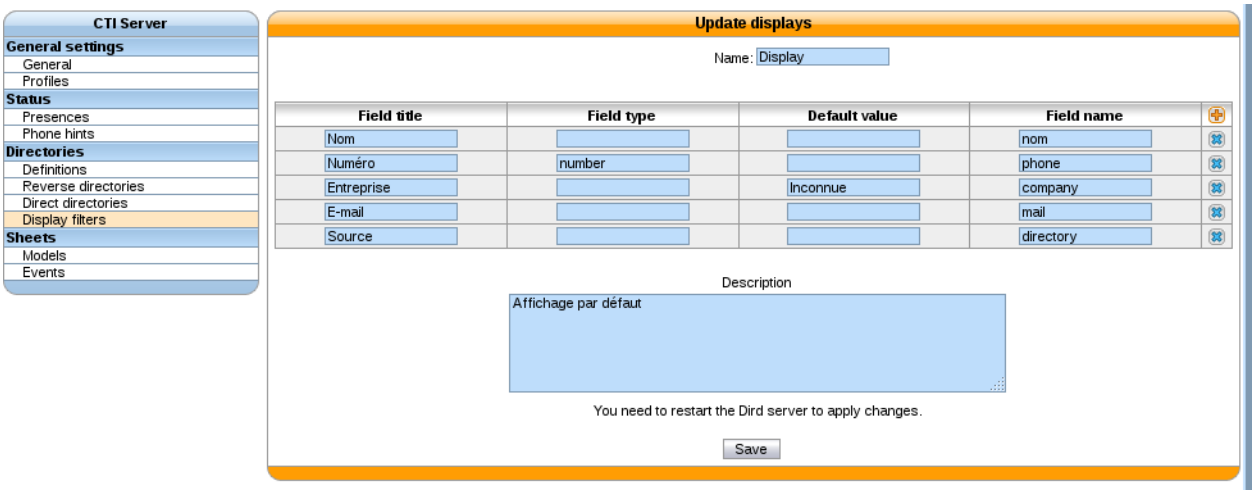
- The *Contacts* xlet (aka. *Search*) has been removed in favor of the *People Xlet*. You may need to do some manual configuration in the directories for the People Xlet to be fully functional. See *the detailed upgrade notes* for more details.
- If you need context separation in the People Xlet, you will have to **manually configure** xivo-dird to keep it working, see *Context separation*. This procedure is only temporary, later versions will handle the context separation automatically.
- xivo-agentd now uses mandatory token authentication for its REST API. If you have custom development using this service, update your program accordingly.
- Some actions that used to be available in the *contact* xlets are not implemented in the *people* xlet yet.
  - Cancel transfer is only available using the *switchboard* xlet
  - Hanging up a call is only possible using the *switchboard* xlet
  - Call interception is not available anymore
  - Conference room invitation is not available anymore

Please consult the following detailed upgrade notes for more information:

People Xlet features Upgrade Notes

When upgrading your XiVO to 15.19, there are some features in the directories that could not be upgraded automatically, because it risked breaking some manual configurations.

After you upgrade your XiVO, your CTI displays in *Services* → *CTI Server* → *Directories* → *Displays* may look like this:



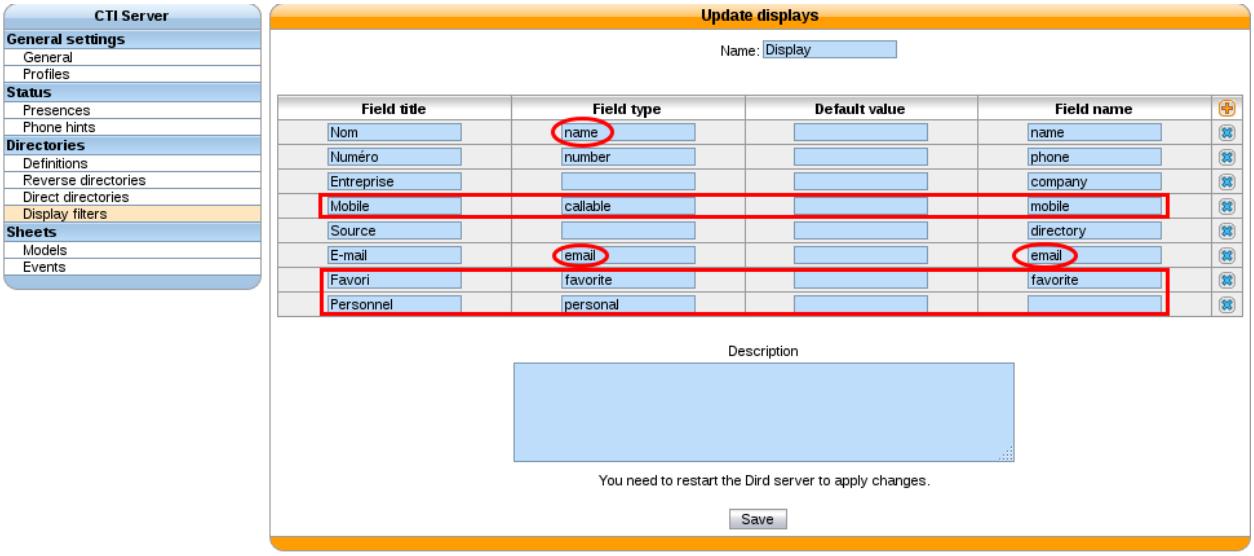
You should update your displays to make them look like:

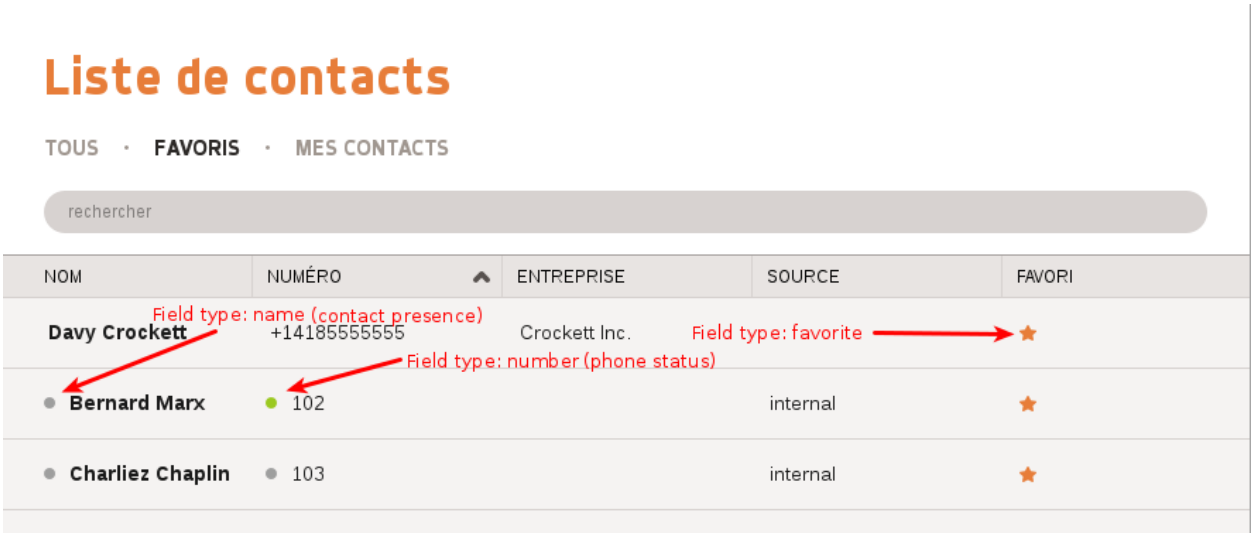
This will give you a Xlet People looking like this:

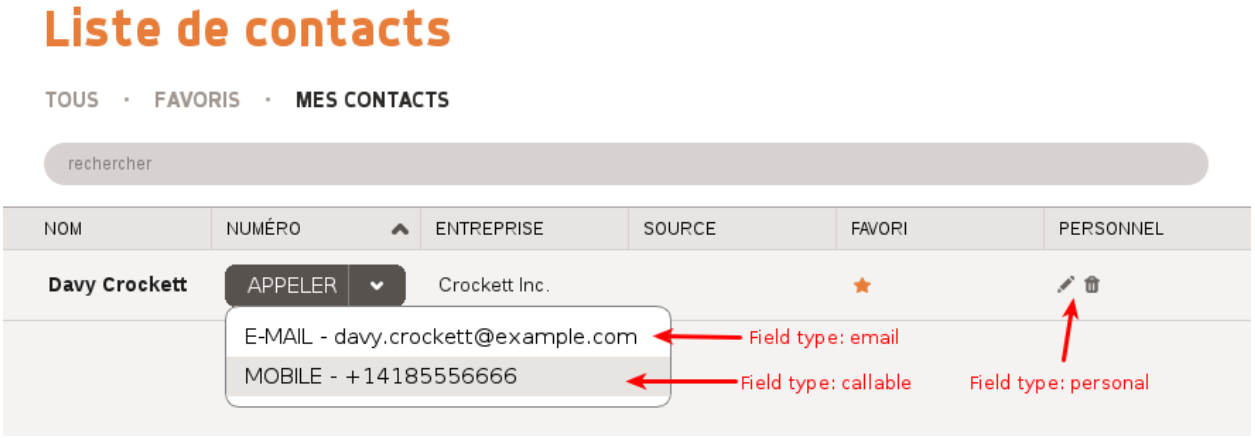
You can find more details about the field types in *Integration of xivo-dird with the rest of Wazo*.

Context separation

Without context separation, you only need one contact source for all the users of your XiVO.


CTI Server
General settings
General
Profiles
Status
Presences
Phone hints
Directories
Definitions
Reverse directories
Direct directories
Display filters
Sheets
Models
Events
Update displays
Name: Display
Field title
Field type
Default value
Field name
Nom
name
name
Numéro
number
phone
Entreprise
company
Mobile
callable
mobile
Source
directory
E-mail
email
email
Favori
favorite
favorite
Personnel
personal
Description
You need to restart the Dird server to apply changes.
Save


Liste de contacts
TOUS · FAVORIS · MES CONTACTS
rechercher
NOM
NUMÉRO
ENTREPRISE
SOURCE
FAVORI
Field type: name (contact presence)
Davy Crockett
+14185555555
Crockett Inc.
Field type: favorite
Field type: number (phone status)
Bernard Marx
102
internal
Charliez Chaplin
103
internal


Liste de contacts
TOUS · FAVORIS · MES CONTACTS
rechercher
NOM
NUMÉRO
ENTREPRISE
SOURCE
FAVORI
PERSONNEL
Davy Crockett
APPELER
Crockett Inc.
E-MAIL - davy.crockett@example.com
Field type: email
MOBILE - +14185556666
Field type: callable
Field type: personal

However, if you need context separation, each context is considered as a separate independant source of contacts, each with a different context filter. For this, you need:

- one contact source per context (a file in `/etc/xivo-dird/sources.d`), so that we have a source containing only the contacts from one context
- one profile per context (equivalent to *Services* → *CTI Server* → *Directories* → *Direct directories*) so that users in one context only see people from the same context.

Each source should look like this one, e.g. the context is named `INSIDE`:

```
confd_config:
  host: localhost
  https: false
  port: 9487
  timeout: 4
  verify_certificate: false
  version: '1.1'
first_matched_columns: [exten]
format_columns:
  directory: "R\xE9pertoire XiVO Interne"
  location: '{description}'
  mobile: '{mobile_phone_number}'
  name: '{firstname} {lastname}'
  number: '{exten}'
  sda: '{userfield}'
  voicemail: '{voicemail_number}'
searched_columns: [firstname, lastname, userfield, description]
type: xivo
unique_column: id
name: internal_INSIDE  # <--- each source has a different name, one per context
extra_search_params:
  context: INSIDE      # <--- each source filters users according to one context
```

The parameters in this file have the same effect than *Configuration* → *Directories* and *Services* → *CTI Server* → *Directories* → *Direct directories* put together.

You may generate these config files from `xivo-confgen dird/sources.yml`. Be sure to have `name` and `extra_search_params` correct for each source file.

Now that we have our contact sources, we need our search profiles.

Create a new file to override the profiles generated by *xivo-confgen*. You only need one file, which will define all your profiles at once.

```
xivo-confgen dird/services.yml >> /etc/xivo-dird/conf.d/001-context-separation.yml
```

In this file, there is a list of services (favorites, lookup, ...) where each profile has a set of sources. You need to match one profile to the right internal source for each service. For example, to have context separation between contexts INSIDE and INDOORS:

```
services:
  favorites:
    __default_phone:
      sources: [xivodir, internal, ldaptest, personal]
    __switchboard_directory:
      sources: [xivodir, ldaptest, personal]
    INSIDE:
      sources: [xivodir, internal_INSIDE, ldaptest, personal]   # <--- profile INSIDE␣
↪uses the source internal_INSIDE
```

```
    INDOORS:
      sources: [xivodir, internal_INDOORS, ldaptest, personal]  # <--- profile␣
↪INDOORS uses the source internal_INDOORS
  lookup:
    __default_phone:
      sources: [xivodir, internal, ldaptest, personal]
    __switchboard_directory:
      sources: [xivodir, ldaptest, personal]
    INSIDE:
      sources: [xivodir, internal_INSIDE, ldaptest, personal]   # <--- same HERE
    INDOORS:
      sources: [xivodir, internal_INDOORS, ldaptest, personal]  # <--- and HERE
```

15.18

Consult the 15.18 Roadmap

- The provd_pycli command (deprecated in 15.06) has been removed in favor of xivo-provd-cli. If you have custom scripts referencing provd_pycli, you'll need to update them.
- The xivo-agentctl command (deprecated in 15.06) has been removed in favor of xivo-agentd-cli. If you have custom scripts referencing xivo-agentctl, you'll need to update them.
- xivo-agentd now uses HTTPS. If you have custom development using this service, update your configuration accordingly. The xivo-agentd-client library, used to interact with xivo-agentd, has also been updated to use HTTPS by default.
- xivo-confd ports 50050 and 50051 have been removed. Please use 9486 and 9487 instead

Configuration File Upgrade Notes

The file format of configuration files for daemons exposing an HTTP/S API has changed. The following services have been affected :

- xivo-agentd
- xivo-amid
- xivo-auth
- xivo-confd
- xivo-ctid
- xivo-dird
- xivo-dird-phoned

Ports and listening addresses are now organised in the following fashion:

```
rest_api:
  https:
    enabled: true
    port: 9486
    listen: 0.0.0.0
    certificate: /usr/share/xivo-certs/server.crt
    private_key: /usr/share/xivo-certs/server.key
    ciphers: "ALL:!aNULL:!eNULL:!LOW:!EXP:!RC4:!3DES:!SEED:+HIGH:+MEDIUM"
  http:
    enabled: true
```

```
port: 9487
listen: 127.0.0.1
```

If you have any custom configuration files for these daemons, please modify them accordingly. Consult *Network* for further details on which network services are available for each daemon.

15.17

Consult the 15.17 Roadmap

- Online call recording is now done via automixmon instead of automon. This has no impact unless you have custom dialplan that is passing directly the "w" or "W" option to the Dial or Queue application. In these cases, you should modify your dialplan to pass the "x" or "X" option instead.
- The remote directory service available from *supported phones* is now provided by the new unified directory service, i.e. xivo-dird. Additional upgrade steps are required to get the full benefit of the new directory service; see the *detailed upgrade notes*.
- The field `enableautomon` has been renamed to `enableonlinerec` in the users web services provided by the web-interface (these web services are deprecated).
- The agent status dashboard now shows that an agent is calling or receiving a non ACD call while in wrapup or paused.
- SIP endpoints created through the REST API will not appear in the web interface until they have been associated with a line
- Due to limitations in the database, only a limited number of optional parameters can be configured on a SIP endpoint. Consult the *xivo-confd changelog* for further details

Please consult the following detailed upgrade notes for more information:

Phone Remote Directory Upgrade Notes

If you are not using the remote directory from your phones, you can safely skip this page.

Starting from XiVO 15.17, the remote directory used by the phones is now provided by the new directory service, composed principally of *xivo-dird* and *xivo-dird-phoned*. It was previously provided by the XiVO web interface.

This brings a few changes for the administrators, the biggest one being that lookup from both the Wazo Client and phones are now configured at the same place, namely the (incorrectly named) *Services* → *CTI Server* → *Directories* section, with some advanced configuration only available in the configuration files. This means that lookup from the phones can now also display results from CSV or web services directories. For details on how to configure directories, refer to the *Directories* page.

For users, the biggest change is that they can now consult their personal contacts (that they added from their Wazo Client) when doing a search from their phone.

Changes

Web Interface - LDAP Filters

The following options have been removed from the web interface, in the *Services* → *IPBX* → *LDAP filters* page:

- the `Phone number type` field

- the `Attributes` tab

The phone number type is now configurable on a per source basis (and for all type of source, not just LDAP), in *Services → CTI Server → Directories*. For example, if you have LDAP records with the attribute `telephoneNumber` that you want to be displayed on your phone with the suffix "(Office)", just make sure that your directory definition is configured with a field named `phone_office` with the value `{telephoneNumber}`.

By default, the following fields are available:

- `phone`: doesn't add a suffix
- `phone_office`: add a "(Office)" suffix
- `phone_mobile`: add a "(Mobile)" suffix
- `phone_home`: add a "(Home)" suffix
- `phone_other`: add a "(Other)" suffix

Note: These fields will automatically be added in your LDAP directory definitions during the upgrade, so you may only need to *review your directory configuration*.

This list of fields and the suffix associated to it is currently only configurable in the *xivo-dird configuration files*, in the *views/displays_phone* section.

This is causing 2 functional changes:

- Previously, the suffix displayed was translated in function of the phone's language. This is not possible anymore, and you'll have to edit the configuration files if you want the suffix to be in a different language than english.
- For "custom" phone number type, you'll have to add a new entry in the configuration files and add the correspond field in the directory definition.

In XiVO 15.16, the `Attributes` tab would allow a "fallback" mechanism, where if an LDAP attribute for a record was missing/empty, another attribute would be used. In XiVO 15.17, this mechanism is available (for all type of sources) by mapping the first attribute to a field name `phone`, the second to a field name `phone1`, etc. The fallback mechanism is available on the fields `phone`, `phone_office`, `phone_mobile`, `phone_home`, `phone_other` and `display_name`.

Web Interface - Phonebook

The following options have been removed from the web interface, in the *Services → IPBX → Phonebook* page:

- the `LDAP filters` tab

LDAP sources used for lookup from the phone are now selected in the same place as for the XiVO client, i.e. in *Services → CTI Services → Direct directories*. A consequence of that is that it's not possible anymore to have sources only used for lookup from phone and other sources only used for lookup from the Wazo Client.

Note: The LDAP filters that were used for phone lookup will be automatically added to all the profiles during the upgrade.

Additional Upgrade Steps

After upgrading your XiVO to 15.17 or later, you should do the following steps.

Upgrade Your Provisioning Plugins

This step is optional, although strongly recommended.

For the users to be able to search their personal contacts from their phone, the phone configuration needs to be updated. This means:

1. Installing new xivo-provd plugins or upgrading existing plugins
2. Restarting all affected phones

See the *provisioning* section for more information on installing or upgrading plugins.

Here's the list of plugins which have received modifications to be compatible with the new directory service:

Name	Version
xivo-aastra-3.3.1-SP4	1.5
xivo-aastra-4.1.0	1.5
xivo-cisco-sccp-9.0.3	0.8
xivo-cisco-sccp-cipc-2.1.2	0.8
xivo-cisco-sccp-legacy	0.8
xivo-cisco-sccp-wireless-1.4.5	0.8
xivo-cisco-spa-7.5.5	0.12
xivo-cisco-spa-legacy	0.12
xivo-polycom-4.0.4	1.4
xivo-polycom-5.3.0	1.5
xivo-snom-8.7.5.17	1.5
xivo-technicolor-ST2022-4.78-1	0.4
xivo-technicolor-ST2030-2.74	0.3
xivo-technicolor-TB30-1.74.0	0.3
xivo-yealink-v70	1.24
xivo-yealink-v72	1.24
xivo-yealink-v73	1.24
xivo-yealink-v80	1.24

Plugins with greater version number or greater firmware-version number are also compatible.

If the xivo-provd plugins are not updated or the phone are not rebooted, the user will by default only be able to search in the "internal" and "xivodir" directory definitions. If you want to add or remove sources for these phones, you'll need to edit xivo-dird configuration files. More precisely, you'll need to edit the sources associated to the profile named `default_phone`.

Update Your Firewall Rules

If there's a firewall (or a NAT equipement) between your XiVO and your phones, you must know that the port used for the directory lookup from the phone has changed from port TCP/80 to port TCP/9498. The new port is going to be used only by phones which are using a compatible plugins (see list above) and have been rebooted; otherwise, the port TCP/80 will still be used.

Review Your Directory Configuration

During the upgrade, new LDAP directory definitions might be created and fields to existing one might be added.

For example, if you had an LDAP filter which was used for directory lookup from your phones, then a corresponding LDAP directory definition will be created if nonexistent, and otherwise be updated to make sure the `display_name` and `phone_office` (or another field, depending on the phone number type of your LDAP filter) fields are defined. The directory definition will also be added to all the direct directories entries, i.e. added to all items in the *Services → CTI Server → Direct directories* page.

If you were using LDAP filters with custom phone number types, the custom part will be lost, and to get back the same behaviour, you'll need to modify xivo-dird configuration files and update the field's name in your directory definition.

Also, if you have other directory defintions that you now want to use from your phones (e.g. CSV directories), make sure that their configuration is working, i.e. that they have a `display_name` and `phone` fields. During the upgrade, these fields are automatically added to the directory defintion "xivodir", "internal" and for LDAP source, like described above.

15.16

Consult the 15.16 Roadmap

- The directory column type "mobile" was removed in favor of the new "callable" type. If you have hand-written configuration files for xivo-dird, in section "views", subsection "displays", all keys "type" with value "mobile" must be changed to value "callable".
- The `xivo-auth` backend interface has changed, `get_acls` is now `get_consul_acls`. All unofficial back ends must be adapted and updated. No action is required for "normal" installations.
- Voicemails can now be deleted even if they are associated to a user.

15.15

Consult the 15.15 Roadmap

Voicemail Upgrade Notes

- Voicemail webservices in the web interface have been removed. Please use the *xivo-confd REST API* instead.
- Voicemail IMAP configuration has been migrated to the new `Advanced` tab.
- Voicemail option `Disable password checking` has been converted to `Ask password`. The value has also been inverted. (e.g. If `Disable password checking` was false, `Ask password` is true.) `Ask password` is activated by default.
- After an upgrade, if ever you have errors when searching for voicemails, please try clearing cookies in your web browser.
- A voicemail must be dissociated from any user prior to being deleted. Voicemail are dissociated by editing the user and clicking on the `Delete voicemail` button in the `Voicemail` tab. This constraint will disappear in future versions.
- Deleting a user will dissociate any voicemail that was attached, but will not delete it nor any messages.
- Creating a line is no longer necessary when attaching a voicemail to a user.
- The following fields have been modified when importing a CSV file:

Old name	New name	Required ?	New default value
voicemailmailbox	voicemailnumber	yes	
voicemailskippass	voicemailaskpassword	no	1
	voicemailcontext	yes	

Directories

- Concatenated fields in directories are now done in the directory definitions instead of the displays
- The field column in directory displays are now field names from the directory definition. No more *{db-*}* are required
- In the directory definitions fields can be modified using a python format string with the fields comming from the source.
- Most of the configuration for xivo-dird is now generated from xivo-confgen using the values in the web interface.
- The *remote directory* xlet has been removed in favor of the new *people* xlet.

See *Directories* and *Integration of xivo-dird with the rest of Wazo* for more details

15.14

- Consult the 15.14 Roadmap
- Default password for `xivo-polycom-4.0.4` plugin version >= 1.3 is now **9486** (i.e. the word "xivo" on a telephone keypad).
- Default password for `xivo-polycom-5.3.0` plugin version >= 1.4 is now **9486**.
- Caller id management for users in confd has changed. Consult the *xivo-confd changelog*.
- The Local Directory Xlet is replaced with the People Xlet. Contacts are automatically migrated to the server. Note that the CSV format for importing contacts has changed (see *People Xlet* for more information).

15.13

- Consult the 15.13 Roadmap
- Asterisk has been upgraded from version 11.17.1 to 13.4.0, which is a major Asterisk upgrade.
- An ARI user has been added to `/etc/asterisk/ari.conf`. If you have configured Asterisk HTTP server to bind on a publicly reachable address (in `/etc/asterisk/http.conf`), then you should update your configuration to prevent unauthorized access on your Asterisk.
- The xivo-dird configuration option *source_to_display_columns* has been removed in favor of the new option *format_columns*. All source configuration using the *source_to_display_columns* must be updated. A migration script will automatically modify source configuration in the */etc/xivo-dird/sources.d* directory.

Please consult the following detailed upgrade notes for more information:

Asterisk 11 to 13 Upgrade Notes

You might be impacted by the upgrade to Asterisk 13 if you have:

- custom dialplan
- custom Asterisk configuration
- custom application using AGI, AMI or any other Asterisk interface
- custom application exploiting CEL or queue_log
- custom Asterisk modules (e.g. codec_g729a.so)
- customized Asterisk in some other way

- DAHDI trunks using SS7 signaling

If you find yourself in one of these cases, you should make sure that your customizations still work with Asterisk 13.

If you are upgrading from Asterisk 1.8, you should also check the *Asterisk 1.8 to 11 upgrade notes*.

Changes Between Asterisk 11 and 13

Some of the more common changes to look for:

- SS7 support is not available in the Asterisk package of XiVO between version 15.13 and 16.08 inclusively.
- All channel and global variable names are evaluated in a case-sensitive manner. In previous versions of Asterisk, variables created and evaluated in the dialplan were evaluated case-insensitively, but built-in variables and variable evaluation done internally within Asterisk was done case-sensitively.
- The SetMusicOnHold dialplan application was deprecated and has been removed. Users of the application should use the CHANNEL function's `musicclass` setting instead.
- The WaitMusicOnHold dialplan application was deprecated and has been removed. Users of the application should use MusicOnHold with a `duration` parameter instead.
- The SIPPEER dialplan function no longer supports using a colon as a delimiter for parameters. The parameters for the function should be delimited using a comma.
- The SIPCHANINFO dialplan function was deprecated and has been removed. Users of the function should use the CHANNEL function instead.
- For SIP, the codec preference order in an SDP during an offer is slightly different than previous releases. Prior to Asterisk 13, the preference order of codecs used to be:

  1. Our preferred codec
  2. Our configured codecs
  3. Any non-audio joint codecs

  Now, in Asterisk 13, the preference order of codecs is:

  1. Our preferred codec
  2. Any joint codecs offered by the inbound offer
  3. All other codecs that are not the preferred codec and not a joint codec offered by the inbound offer
- Queue strategy `rrmemory` (Round robin memory) now has a predictable order. Members will be called in the order that they are added to the queue. For agents, this means they will be called in the order they are logged.
- When performing queue pause/unpause on an interface without specifying an individual queue, the PAUSEALL/UNPAUSEALL event will only be logged if at least one member of any queue exists for that interface. This has an impact on the agent performance statistics; an agent must be a member of at least 1 queue for its pause time to show up in the statistics.

You can see the complete list of changes from the Asterisk website:

- https://wiki.asterisk.org/wiki/display/AST/Upgrading+to+Asterisk+12
- https://wiki.asterisk.org/wiki/display/AST/Upgrading+to+Asterisk+13
- http://git.asterisk.org/gitweb/?p=asterisk/asterisk.git;a=blob;f=CHANGES;h=d0363f7c3b03cec5f71b3806535c4f9d2b2baa02;hb=refs/heads/13

The AGI protocol did not change between Asterisk 11 and Asterisk 13; if you have custom AGI applications, you only need to make sure that the dialplan applications and functions you are using from the AGI are still valid.

List of Known Bugs And Limitations

List of known bugs and limitations for Asterisk 13 in XiVO:

- When direct media is active and DTMF are sent using SIP INFO, DTMF are not working properly. It is also impossible to do an attended transfer from the Wazo Client in these conditions.

  See http://projects.wazo.community/issues/5692.

15.12

- Consult the 15.12 Roadmap
- The certificate used for HTTPS in the web interface will be regenerated if the default certificate was used. Your browser will complain about the new certificate, and it is safe to accept it (see #3656). See also *Certificates for HTTPS*.
- If you have an *HA configuration*, then you should run `xivo-sync -i` on the master node to setup file synchronization between the master and the slave. File synchronization will then be done automatically every hour via rsync and ssh.
- xivo-auth and xivo-dird now use HTTPS, if you have custom development using these services, update your configuration accordingly.

15.11

- Consult the 15.11 Roadmap
- The call records older than 365 days will be periodically removed. The first automatic purge will occur in the night after the upgrade. See *xivo-purge-db* for more details.

15.10

- Consult the 15.10 Roadmap

15.09

- Consult the 15.09 Roadmap

15.08

- Consult the 15.08 Roadmap
- The Dialer Xlet has been integrated in Identity Xlet.

15.07

- Consult the 15.07 Roadmap

15.06

- Consult the 15.06 Roadmap
- The provd client has been moved into a new python package, xivo_provd_client. If you have custom scripts using this client, you'll need to update them. See http://projects.wazo.community/issues/5469 for more information.
- The provd_pycli command name has been deprecated in favor of xivo-provd-cli. These 2 commands do the same thing, the only difference being the name of the command. The provd_pycli command name will be removed in 15.18, so if you have custom scripts referencing provd_pycli, you'll need to update them.
- The xivo-agentctl command name has been deprecated in favor of xivo-agentd-cli. These 2 commands do the same thing, the only difference being the name of the command. The xivo-agentctl command name will be removed in 15.18, so if you have custom scripts referencing xivo-agentctl, you'll need to update them.

15.05

- Consult the 15.05 Roadmap
- The Xlet identity has been modified to follow the new Wazo Client design which implies the removal of some details.

15.04

- Consult the 15.04 Roadmap

15.03

- Consult the 15.03 Roadmap

15.02

- Consult the 15.02 Roadmap

15.01

- Consult the 15.01 Roadmap
- The *confd REST API* is now more restrictive on HTTP headers. Particularly, the headers Accept and Content-Type must be set to (typically) `application/json`.
- The following configuration files have been created:
  - `/etc/xivo-agid/config.yml`
  - `/etc/xivo-call-logd/config.yml`
  - `/etc/xivo-amid/config.yml`
  - `/etc/xivo-agentd/config.yml`

2014

14.24

- Consult the 14.24 Roadmap

The following security vulnerability has been fixed:

- XIVO-2014-01: Queues and groups permit callers to make unwanted calls

14.23

- Consult the 14.23 Roadmap
- The "waiting calls / logged agents ratio" *queue diversion scenario* has been renamed to "number of waiting calls per logged agents".
- A new *community* section was added to the official documentation for all user-contributed documentation.

14.22

- Consult the 14.22 Roadmap
- The sheet event *Dial* on queues is now only sent to the ringing agent. The sheet is also sent a little later during the call, when the ringing agent is known.

14.21

- Consult the 14.21 Roadmap
- The *confd REST API* is now accessible via HTTPS on port 9486 and via HTTP on port 9487 (localhost only). These ports are replacing the 50051 and 50050 ports respectively. It will still be possible to access the confd REST API via the 50051 and 50050 ports for the next year, but you are advised to update your confd REST API clients as soon as possible.
- The old (unsupported) ami-proxy is now replaced by an ami-proxy built in xivo-ctid. You must uninstall the old ami-proxy before activating the built-in version. See *troubleshooting xivo-ctid* to learn how to activate.

14.20

- Consult the 14.20 Roadmap
- Default parameters for all Cisco SPA ATA plugins have changed to be better suited for european faxes.
- Following the POODLE attack (CVE-2014-3566), SSL 3.0 has been disabled for the web interface and the xivo-confd REST API.

If you have Aastra phones and are using the remote directory on them, consult the following detailed upgrade notes:

Aastra Remote Directory Upgrade Notes

Starting from XiVO 14.20, it is not possible anymore to use SSL 3.0 when connecting to XiVO using HTTPS.

This has the unfortunate consequence of breaking the remote directory on Aastra phones configured by the `xivo-aastra` provisioning plugins in version 1.2 and earlier.

Upgrade procedure

To be able to use the remote directory on your Aastra phones on XiVO 14.20 or later, you'll need to take one of the following actions:

Upgrade to the Latest Plugin

This is the recommended solution. This can be done either before or after the upgrade. You'll have to:

1. Upgrade your `xivo-aastra` plugin to version 1.3 or later
2. Restart/synchronize all your phones

The correction is only available for plugin `xivo-aastra-3.3.1-SP2` and later. If you are using an older plugin (`xivo-aastra-3.2.2-SP3` for example), then you'll need to install a newer plugin and *update all your phones to use the new plugin*.

If you were already using custom templates, make sure to update them so that the phones access the remote directory via HTTP instead of HTTPS. This can be done using the following command:

```
find /var/lib/xivo-provd/plugins/xivo-aastra* -name '*.tpl' -exec sed -i '/X_xivo_
↪phonebook_ip/s/\bhttps:/http:/' {} \;
```

Update the Templates

If you can't or don't want to update to a newer plugin, you can instead update the templates used by the plugin. This can be done either before or after the upgrade. You'll have to:

1. Update the templates so that the directory is accessed via HTTP
2. Restart/synchronize all your phones

In this specific case, it is safe to directly modify the templates used by the plugin instead of *creating custom templates*. To update the templates, you can use the following command:

```
find /var/lib/xivo-provd/plugins/xivo-aastra* -name '*.tpl' -exec sed -i '/X_xivo_
↪phonebook_ip/s/\bhttps:/http:/' {} \;
```

Re-enable SSL 3.0

If you can't restart/synchronize your phones, the last solution is to re-enable SSL 3.0 on your XiVO. This should only be used as a temporary solution to give you more time to plan a firmware upgrade for your phones. This can be done only after the upgrade. You'll have to:

1. Update nginx configuration
2. Reload nginx

This can be done using the following commands:

```
sed -i 's/ssl_protocols .*/ssl_protocols SSLv3 TLSv1 TLSv1.1 TLSv1.2;/' /etc/nginx/
↪sites-available/xivo
service nginx reload
```

14.19

- Consult the 14.19 Roadmap

14.18

- Consult the 14.18 Roadmap
- xivo-fai packages were replaced with xivo-dist : a new tool to handle repositories sources. Upon upgrade, xivo-dist is installed and run and all xivo-fai packages are purged. *Consult xivo-dist use cases*

14.17

- Consult the 14.17 Roadmap
- DAHDI configuration file `/etc/dahdi/modules` is no more created by default and must now be maintained manually. No action is needed upon upgrade but be aware that the upstream sample file is now available in `/usr/share/dahdi/modules.sample`. See *dahdi modules documentation* for detailed info.
- The new *CCSS feature* will not be enabled upon upgrade, you must explicitly enable it in the *IPBX* → *IPBX Services* → *Extensions* menu.

14.16

- Consult the 14.16 Roadmap
- See the *changelog* for xivo-confd's REST API
- DAHDI is upgraded to 2.10.0. If the upgrade process asks about `/etc/dahdi/modules`, we recommend that you keep the old version of the file.
- Asterisk now inserts CEL and queue log entries via the ODBC asterisk modules instead of the pgsql modules.

14.15

- Consult the 14.15 Roadmap
- Duplicate function keys will be deleted upon upgrade. If multiple function keys pointing to the same destination are detected for a given user, only the one with the lowest position will be kept. To see the list of deleted function keys, check the xivo-upgrade log file such as:

```
grep MIGRATE_FK /var/log/xivo-upgrade.log
```

DAHDI 2.9.2 Upgrade Notes

These notes only apply to:

- Digium TE133/TE131 cards that are in firmware version 780017 or earlier
- Digium TE435/TE235 cards that are in firmware version e0017 or ealier

Warning: The system will need to be power cycled after the upgrade. Your cards will not be usable until then.

After the upgrade

First, you need to install the latest firmware for your TE133/TE131 or TE435/TE235 cards:

```
xivo-fetchfw install digium-te133
xivo-fetchfw install digium-te435
```

Then stop all the services and reload the DAHDI modules. Reloading the DAHDI module might take up to 30 seconds:

```
wazo-service stop
service dahdi stop
service dahdi start
```

Following this manipulation, you should see something similar at the end of the `/var/log/messages` file:

```
dahdi: Telephony Interface Unloaded
dahdi: Version: 2.9.2
dahdi: Telephony Interface Registered on major 196
wcte13xp 0000:03:0c.0: Firmware version 780017 is running, but we require version␣
↪780019.
wcte13xp 0000:03:0c.0: firmware: agent loaded dahdi-fw-te133.bin into memory
wcte13xp 0000:03:0c.0: Found dahdi-fw-te133.bin (version: 780019) Preparing for flash
wcte13xp 0000:03:0c.0: Uploading dahdi-fw-te133.bin. This can take up to 30 seconds.
wcte13xp 0000:03:0c.0: Delaying reset. Firmware load requires a power cycle
wcte13xp 0000:03:0c.0: Running firmware version: 780017
wcte13xp 0000:03:0c.0: Loaded firmware version: 780019 (Will load after next power␣
↪cycle)
wcte13xp 0000:03:0c.0: FALC version: 5
wcte13xp 0000:03:0c.0: Setting up global serial parameters for T1
wcte13xp 0000:03:0c.0: VPM450: firmware dahdi-fw-oct6114-032.bin not available from␣
↪userspace
```

For the firmware update to complete, you **must halt** the machine (a reboot won't be enough) before restarting it.

14.14

- Consult the 14.14 Roadmap
- See the *changelog* for REST API
- Upon an important freeze of Asterisk, Asterisk will be restarted. See the associated ticket for more information.

14.13

- Consult the 14.13 Roadmap
- See the *changelog* for REST API
- Skills-based routing: for an agent which doesn't have the skill X, the rule X < 10 was previously evaluated to true, since not having the skill X was equivalent to having it with a value of 0. This behaviour has changed, and the same expression is now evaluated to false. If you are using skills-based routing, you'll need to check that your rules are still doing what you expect. See *skill evaluation* for more information.

14.12

- Consult the 14.12 Roadmap
- All provisioning plugins were modified. Although not mandatory, it is strongly advised to update all used plugins.
- The function key 'Activate voicemail' was removed as it was a duplicate of existing function key 'Enable voicemail'. All users having the 'Activate voicemail' function key will have to be reconfigured with a 'Enable voicemail' function key in order to keep the equivalent feature.
- Log files have changed for the following daemons (previously in `/var/log/daemon.log`):
  - xivo-provd: `/var/log/xivo-provd.log`
  - xivo-agid: `/var/log/xivo-agid.log`
  - xivo-sysconfd: `/var/log/xivo-sysconfd.log`

14.11

- Consult the 14.11 Roadmap
- The API URL `/lines/<id>/extension` is now deprecated. Use `/lines/<id>/extensions` instead.

14.10

- Consult the 14.10 Roadmap
- Custom MOH have been fixed, but can not be used for playing uploaded files anymore. See *Music on Hold*.

14.09

- Consult the 14.09 Roadmap
- REST API 1.0 is no more. All code, tests and documentation was removed from XiVO. All code developped for REST API 1.0 must now be adapted to use REST API 1.1.

14.08

- Consult the 14.08 Roadmap
- The `xivo` database has been merged into the `asterisk` database. The database schema has also been altered in a way that it might make the upgrade longer than usual.

Please consult the following detailed updated notes for more information:

Databases Merge Upgrade Notes

The `xivo` database has been merged into the `asterisk` database in XiVO 14.08. This has an impact on:

- The *restore* procedure. There's only one database to restore now. Also, the procedure to restore the data while keeping the system configuration has been updated.
- The data that is replicated between the master and the slave in a *high availability* cluster.

Previously, all the configuration that was under the "Configuration" menu of the web interface was not replicated between the master and slave. This is now replicated, except for:

- HA settings
- All the network configuration (i.e. everything under the *Configuration* → *Network* section)
- All the support configuration (i.e. everything under the *Configuration* → *Support* section)

The call center statistics have also been excluded from the replication.

The way the replication is done has also been updated, which makes it faster.

Optional Upgrade Procedure

When upgrading to XiVO 14.08, the database schema will be altered.

This will result in a longer upgrade time if you have a lots of rows in the queue_log table.

You can see the number of rows in your queue_log table with:

```
sudo -u postgres psql -c "SELECT count(*) FROM queue_log" asterisk
```

On ordinary hardware, you can expect that it will take ~10 minutes for every 2.5 million of rows. So if you have 5 million of rows in your queue_log table, you can expect that the upgrade will take an extra 20 minutes.

It is possible to reduce the amount of additional time the upgrade will take by either removing rows from the table or altering the table before the upgrade.

Both these commands can be run while the XiVO services are up.

For example, if you want to remove all the rows before march 2014, you can use:

```
sudo -u postgres psql -c "DELETE FROM queue_log WHERE \"time\" < '2014-03-01'"␣
↪asterisk
```

If you want to alter the table before the upgrade, you can use:

```
sudo -u postgres psql -c "ALTER TABLE queue_log ADD COLUMN id SERIAL PRIMARY KEY;␣
↪GRANT ALL ON SEQUENCE queue_log_id_seq TO asterisk" asterisk
```

Note: It is recommended to execute this command when there's no activity on the system.

More Technical Information

The way the database is initially provisioned and the way it is altered during an upgrade has also been changed.

In XiVO 14.07 and earlier, the database was provisioned by executing the `/usr/share/xivo-manage-db/datastorage/asterisk.sql` SQL script. Starting with XiVO 14.08, the `xivo-init-db` is responsible for provisioning the database. This script should not be used by an administrator in normal circumstance.

Starting with XiVO 14.08, database migration are done with the help of alembic instead of the asterisk-XXX.sql and xivo-XXX.sql scripts. The alembic migration scripts can be found inside the `/usr/share/xivo-manage-db` directory.

Otherwise, the `xivo-check-db` and `xivo-update-db` commands have been updated to work with both the old and the new systems and are still the official way to check the database state and update the database respectively.

14.07

- Consult the 14.07 Roadmap
- Configuration for phones used for the switchboard has changed.

Please consult the following detailed updated notes for more information:

Switchboard Phone Configuration Upgrade Notes

The xivo-aastra-switchboard and xivo-snom-switchboard plugins have been removed and their functionalities are now provided by the generic xivo-aastra and xivo-snom plugins respectively.

The upgrade is not done automatically, so please follow the *Upgrade Procedure* section below.

Although you are strongly advised to upgrade your switchboard phone configuration, backwards compatibility with the old system will be maintained.

Note that if you need to install a switchboard for a previous version of XiVO, the old xivo-aastra-switchboard and xivo-snom-switchboard plugins can be found in *the archive repository*.

Upgrade Procedure

This procedure should be executed after the upgrade to 14.07 or later: the options used in this procedure are not available in versions before 14.07.

The following upgrade procedure suppose that you are using an Aastra phone as your switchboard phone. The same upgrade procedure apply for Snom phones, with the only difference being the different plugin name.

1. Update the list of installable plugins.
2. Install the latest xivo-aastra plugin, or upgrade it to the latest version if it is already installed.
3. Install the needed language files and firmware files.
4. For each phone used for the switchboard, *change the plugin and activate the switchboard option*:
    - Select the generic xivo-aastra plugin.
    - Check the “switchboard” checkbox.
    - Synchronize the phone.
5. Once this is completed, you can uninstall the xivo-aastra-switchboard plugin.

An unofficial script that automates this procedure is also available on github:

```
cd /tmp
wget --no-check-certificate https://raw.githubusercontent.com/wazo-pbx/xivo-tools/
↪master/scripts/migrate_switchboard_1407.py
python migrate_switchboard_1407.py
```

14.06

- Consult the 14.06 Roadmap
- The Wazo Client now uses Qt 5 instead of Qt 4. There is nothing to be aware of unless you are *building your own version* of it.

14.05

- Consult the 14.05 Roadmap
- The *CTI Protocol* has been updated.
- The specification of the 'answered-rate' queue statistic has changed to exclude calls on a closed queue
- The switchboard can now choose which incoming call to answer
- The package versions do not necessarily contain the current XiVO version, it may contain older versions. Only the package `xivo` is guaranteed to have the current XiVO version.

Please consult the following detailed updated notes for more information:

DAHDI 2.9.0 Upgrade Notes

These notes only apply to Digium TE133 or TE134 cards that are in firmware version 770017 or earlier.

> **Warning:** The system will need to be power cycled after the upgrade. Your cards will not be usable until then.

After the upgrade

First, you need to install the latest firmware for your TE133 or TE134 cards:

```
xivo-fetchfw install digium-te133
xivo-fetchfw install digium-te134
```

Then stop all the services and reload the DAHDI modules. Reloading the DAHDI module might take up to 30 seconds:

```
wazo-service stop
service dahdi stop
service dahdi start
```

Following this manipulation, you should see something similar at the end of the `/var/log/messages` file:

```
dahdi: Telephony Interface Unloaded
dahdi: Version: 2.9.0
dahdi: Telephony Interface Registered on major 196
wcte13xp 0000:03:0c.0: Firmware version 6f0017 is running, but we require version␣
↪780017.
wcte13xp 0000:03:0c.0: firmware: agent loaded dahdi-fw-te134.bin into memory
wcte13xp 0000:03:0c.0: Found dahdi-fw-te134.bin (version: 780017) Preparing for flash
wcte13xp 0000:03:0c.0: Uploading dahdi-fw-te134.bin. This can take up to 30 seconds.
wcte13xp 0000:03:0c.0: Delaying reset. Firmware load requires a power cycle
wcte13xp 0000:03:0c.0: Running firmware version: 6f0017
wcte13xp 0000:03:0c.0: Loaded firmware version: 780017 (Will load after next power␣
↪cycle)
wcte13xp 0000:03:0c.0: FALC version: 5
wcte13xp 0000:03:0c.0: Setting up global serial parameters for T1
wcte13xp 0000:03:0c.0: VPM450: firmware dahdi-fw-oct6114-032.bin not available from␣
↪userspace
wcte13xp 0000:03:0c.0: Found a Wildcard TE132/TE134 (SN: 1TE134F - DF05132600690 - B1␣
↪- 20130702)
```

For the firmware update to complete, you **must halt** the machine (a reboot won't be enough) before restarting it.

SCCP Upgrade Notes

Important modification have been made to the internal structure of the SCCP channel driver, xivo-libsccp.

The modifications mostly affect administrators; users are not affected.

Major changes are:

- Improved support for live modifications; no more manual intervention in the asterisk CLI is needed.
- Improved handling of concurrency; crash and deadlock due to concurrency problems should not occur anymore.

CLI

The following commands have been removed because they were not needed:

- sccp resync
- sccp set directmedia
- sccp show lines
- sccp update config

The behavior of the following commands have been changed:

- `module reload chan_sccp` reloads the module configuration, without interrupting the telephony service. A device will only be resetted/restarted if needed, and only once the device is idle. Some changes don't even require the device to be resetted.
- `sccp show config` output format has been changed a little.
- `sccp show devices` only show the connected devices instead of all the devices. This might change in the future. To get a list of all the devices, use `sccp show config`.

Configuration File

The format of the `sccp.conf` configuration file has been changed. This will only impact you if you are using xivo-libsccp without using XiVO.

The format has been changed because the module is now using the ACO module from asterisk, which expect configuration file to have a specific format.

See sccp.conf.sample for a configuration file example.

Other

Each SCCP session/connection now use 3 file descriptors instead of 1 previously. On XiVO, the file descriptor limit for the asterisk process is 8192, which means that the increase in used file descriptors should not be a problem, even on a large installation.

14.04

- Consult the 14.04 Roadmap
- Live reload of the configuration can be enabled and disabled using the REST API
- The generation of call logs for unanswered calls from the Wazo Client have been improved.

14.03

- Consult the 14.03 Roadmap
- A migration script adds an index on the linkedid field in the cel table. Tests have shown that this operation can last up to 11.5 minutes on a XiVO Corporate with 18 millions CELs. xivo-upgrade will thus be slightly longer.
- Two new daemons are now operationnal, xivo-amid and xivo-call-logd:
  - xivo-amid constantly reads the AMI and sends AMI events to the RabbitMQ bus
  - xivo-call-logd generates call-logs in real time based on AMI LINKEDID_END events read on the bus
- An increase in load average is expected with the addition of these two new daemons.
- The cron job calling xivo-call-logs now runs once a day at 4:25 instead of every 5 minutes.

14.02

- Consult the 14.02 Roadmap
- PHP Web services has been removed from documentation
- REST API 1.0 Web services has been removed from documentation
- REST API 1.1 User-Line-Extension service is replaced by User-Line and Line-Extension services

14.01

- Consult the 14.01 Roadmap
- The following paths have been renamed:
  - `/etc/pf-xivo` to `/etc/xivo`
  - `/var/lib/pf-xivo` to `/var/lib/xivo`
  - `/usr/share/pf-xivo` to `/usr/share/xivo`

You must update any dialplan or configuration file using these paths

2013

13.25

- Consult the 13.25 Roadmap
- Debian has been upgraded from version 6 (squeeze) to 7 (wheezy).

Please consult the following detailed upgrade notes for more information:

Debian 7 (wheezy) Upgrade Notes

Before the upgrade

- The upgrade will take longer than usual, because the whole Debian system will be upgraded
- The system must be restarted after the upgrade, because the Linux kernel will also be upgraded

LDAPS

In case XiVO is using a LDAP server through SSL/TLS (LDAPS), the documentation instructed you to append the certificate to `/etc/ssl/certs/ca-certificates.crt`. However, this is the wrong way to add a new certificate, because it will be erased by the upgrade.

To keep your certificate installed through the upgrade, you must follow the instructions given in the *LDAP documentation*.

After the upgrade

GRUB (Cloned Virtual Machines only)

GRUB installations on cloned virtual machines may lead to unbootable systems, if not fixed properly before restarting the system. If xivo-upgrade detects your system is in a broken state, it will display a few commands to repair the GRUB installation.

13.24

- Consult the 13.24 Roadmap
- Default Quality of Service (QoS) settings have been changed for SCCP. The IP packets containing audio media are now marked with the EF DSCP.

13.23

- Consult the 13.23 Roadmap
- The *New call* softkey has been removed from SCCP phones in *connected* state. To start a new call, the user will have to press *Hold* then *New call*. This is the same behavior as a *Call Manager*.
- Some softkeys have been moved on SCCP phones. We tried to keep the keys in the same position at any given time. As an example, the *transfer* key will not become *End call* while transfering a call. Note that this is a work in progress and some models still need some tweaking.

13.22

- Consult the 13.22 Roadmap
- PostgreSQL will be upgraded from 9.0 to 9.1. The upgrade of XiVO will take longer than usual, depending on the size of the database. Usually, the database grows with the number of calls processed by XiVO. The upgrade will be stopped if not enough space is available on the XiVO server.

13.21

- Consult the 13.21 Roadmap
- It is no more possible to delete a device associated to a line using REST API.

13.20

- Consult the 13.20 Roadmap
- xivo-libsccp now supports direct media on wifi phone 7920 and 7921
- xivo-confd now implements a voicemail list

13.19

- Since XiVO 13.18 was not released, the 13.19 release contains all developments of both 13.18 and 13.19, therefore please consult both Roadmaps :
- Consult the 13.19 Roadmap
- Consult the 13.18 Roadmap
- Call logs are now generated automatically, incrementally and regularly. Call logs generated before 13.19 will be erased one last time.
- The database was highly modified for everything related to devices : table devicefeatures does not exist anymore and now relies on information from xivo-provd.

13.17

- Consult the 13.17 Roadmap
- There is a major change to call logs. They are no longer available as a web report but only as a csv export. See the *call logs documentation*. Furthermore, call logs are now fetched from xivo-confd REST API.
- Paging group numbers are now exclusively numeric. All non-numeric paging group numbers are converted to their numeric-only equivalent while upgrading to XiVO 13.17 (*58 becomes 58, for example).

13.16

- Consult the 13.16 Roadmap
- A migration script modifies the user and line related-tables and the way users, lines and extensions are associated. As a consequence of this script, it is not possible any more to associate a user and a line without extensions. Existing associations between users and one or more lines having no extensions will be removed. Users and lines will still exist unassociated.
- The *call logs* page is able to display partial results of big queries, instead of displaying a blank page.
- Two new CEL messages are now enabled : LINKEDID_END and BRIDGE_UPDATE. Those events will only exist in CEL for calls passed after upgrading to XiVO 13.16.
- The new REST API now makes possible to associate multiple user to a given line and/or extension. There are currently some limitations on how those users and lines can be manipulated using the web interface.

13.15

- There was no production release of XiVO 13.15. All 13.15 developments are included in the official 13.16 release.

13.14

- Consult the 13.14 Roadmap
- The latest Polycom plugin enables the phone lock feature with a default user password of '123'. All Polycom phones used with XiVO also have a default admin password. In order for the phone lock feature to be secure, one should change every phone's admin AND user passwords.
- WebServices for SIP trunks/lines: field `nat`: value `yes` changed to `force_rport,comedia`
- The database has beed updated in order to remove deprecated tables (generalfeatures, extenumbers, extenhash, cost_center).

13.13

- Consult the 13.13 Roadmap

13.12

- Consult the 13.12 Roadmap
- CTI protocol: Modified values of agent `availability`. Read *CTI Protocol changelog*

- Clean-up was made related to the minimization of the Wazo Client. Some visual differences have been observed on Mac OS X that do not affect the Wazo Client in a functional way.

13.11

- Consult the 13.11 Roadmap
- Asterisk has been upgraded from version 11.3.0 to 11.4.0

API changes:

- Dialplan variable XIVO_INTERFACE_0 is now XIVO_INTERFACE
- Dialplan variable XIVO_INTERFACE_NB and XIVO_INTERFACE_COUNT have been removed
- The following fields have been removed from the lines and users web services
  - line_num
  - roles_group
  - rules_order
  - rules_time
  - rules_type

13.10

- Consult the 13.10 Roadmap

API changes:

- CTI protocol: for messages of class `getlist` and function `updateconfig`, the `config` object/dictionary does not have a `rules_order` key anymore.

13.09

- Consult the 13.09 Roadmap
- The *Restart CTI server* link has been moved from *Services* → *CTI Server* → *Control* to *Services* → *IPBX* → *Control*.
- The Agent Status Dashboard has been optimized.
- The Directory xlet can now be used to place call.

13.08

- Consult the 13.08 Roadmap
- asterisk has been upgraded from version 1.8.21.0 to 11.3.0, which is a major asterisk upgrade.
- The switchboard's queue now requires the *xivo_subr_switchboard* preprocess subroutine.
- A fix to bug #4296 introduced functional changes due to the order in which sub-contexts are included. Please refer to ticket for details.

Please consult the following detailed upgrade notes for more information:

Asterisk 1.8 to 11 Upgrade Notes

Table of modules that were available in the asterisk 1.8 package but that are not available anymore in the asterisk 11 package:

Name	Description	Loaded in AST1.8	Asterisk Status	Replaced By
app_dahdibarge	Barge in on DAHDI channel application	Yes	Deprecated	app_chanspy
app_readfile	Stores output of file into a variable	Yes	Deprecated	func_env (FILE())
app_saycountpl	Say polish counting words	Yes	Deprecated	say.conf
app_setcallerid	Set CallerID Presentation Application	Yes	Deprecated	func_callerid
cdr_sqlite	SQLite CDR Backend	No	Removed	cdr_sqlite3_custom
chan_gtalk	Gtalk Channel Driver	No	Deprecated	chan_motif
chan_jingle	Jingle Channel Driver	No	Deprecated	chan_motif
chan_vpb	Voicetronix API driver	No	Supported	
format_sln16	Raw Signed Linear 16KHz Audio support	Yes	Removed	format_sln
res_ais	SAForum AIS	No	Removed	res_corosync
res_jabber	AJI - Asterisk Jabber Interface	No	Deprecated	res_xmpp

List of modules that were loaded in asterisk 1.8 but that are not loaded anymore in asterisk 11 (see modules.conf):

- res_calendar.so
- res_calendar_caldav.so
- res_calendar_ews.so
- res_calendar_exchange.so
- res_calendar_icalendar.so
- res_config_sqlite.so
- res_stun_monitor.so

List of debian packages that are not available anymore for asterisk 11:

- asterisk-config
- asterisk-mysql
- asterisk-web-vmail

Note: These packages were not installed by default for asterisk 1.8.

If you are using some custom dialplan or AGIs, it is your responsibility to make sure it still works with asterisk 11. See the *External Links* for more information.

External Links

- http://svnview.digium.com/svn/asterisk/branches/11/UPGRADE-10.txt
- http://svnview.digium.com/svn/asterisk/branches/11/UPGRADE.txt

- https://wiki.asterisk.org/wiki/display/AST/New+in+10
- https://wiki.asterisk.org/wiki/display/AST/New+in+11

The switchboard's queue preprocess subroutine

The switchboard's queue now uses a preprocess subroutine named *xivo_subr_switchboard*. This preprocess subroutine will be associated with all queues named *__switchboard* that have no preprocess subroutine defined before the upgrade.

If your switchboard queue is named anything other than *__switchboard* you should add the preprocess subroutine manually.

If your switchboard queue already has a preprocess subroutine, you should add a Gosub(xivo_subr_switchboard) to you preprocess subroutine.

Warning: This change is only applied to the switchboard distribution queue, not the queue for calls on hold.

13.07

- Consult the 13.07 Roadmap
- Agent Status Dashboard has more features and less limitations. See related *agent status dashboard documentation*
- XiVO call centers have no more notion of 'disabled agents'. All previously disabled agents in web interface will become active agents after upgrading.
- asterisk has been upgraded from version 1.8.20.1 to 1.8.21.0. Please note that in XiVO 13.08, asterisk will be upgraded to version 11.
- DAHDI has been upgraded from version 2.6.1 to 2.6.2.
- libpri has been upgraded from version 1.4.13 to 1.4.14.
- PostgreSQL upgraded from version 9.0.4 to 9.0.13

13.06

- Consult the 13.06 Roadmap
- The new Agent Status Dashboard has a few known limitations. See related *dashboard xlet known issues section*
- Status Since counter in xlet list of agents has changed behavior to better reflect states of agents in queues as seen by asterisk. See Ticket #4254 for more details.

13.05

- Consult the 13.05 Roadmap
- The bug #4228 concerning BS filter only applies to 13.04 servers installed from scratch. Please upgrade to 13.05.
- The order of softkeys on SCCP phones has changed, e.g. the *Bis* button is now at the left.

13.04

- Consult the 13.04 Roadmap
- Upgrade procedure for HA Cluster has changed. Refer to *Specific Procedure : Upgrading a Cluster*.
- Configuration of switchboards has changed. Since the directory xlet can now display any column from the lookup source, a display filter has to be configured and assigned to the __switchboard_directory context. Refer to *Directory xlet documenttion*.
- There is no more context field directly associated with a call filter. Boss and secretary users associated with a call filter must necessarily be in the same context.

1.5 AdminUI

AdminUI is the management interface for Wazo. This is your entry point to the configuration of your Wazo server. It is accessible at https://\T1\textless{}wazo\T1\textgreater{}/admin

If you did not *configure your own HTTPS certificates* you will have to accept the self signed certificate that was generated on your Wazo.

1.5.1 Login

The login page allows the administrator to login into AdminUI. The username and password used here are those that were choosen during the configuration wizard.

1.5.2 Plugins

Wazo plugins can be installed from AdminUI in the plugin menu. Features can be installed here to customize your Wazo installation.

Official plugins are developed by the Wazo development team, are tested and should not break during an upgrade. Non-official plugins are developed by the community and are available in the market for your convenience. The default is to show only official plugins.

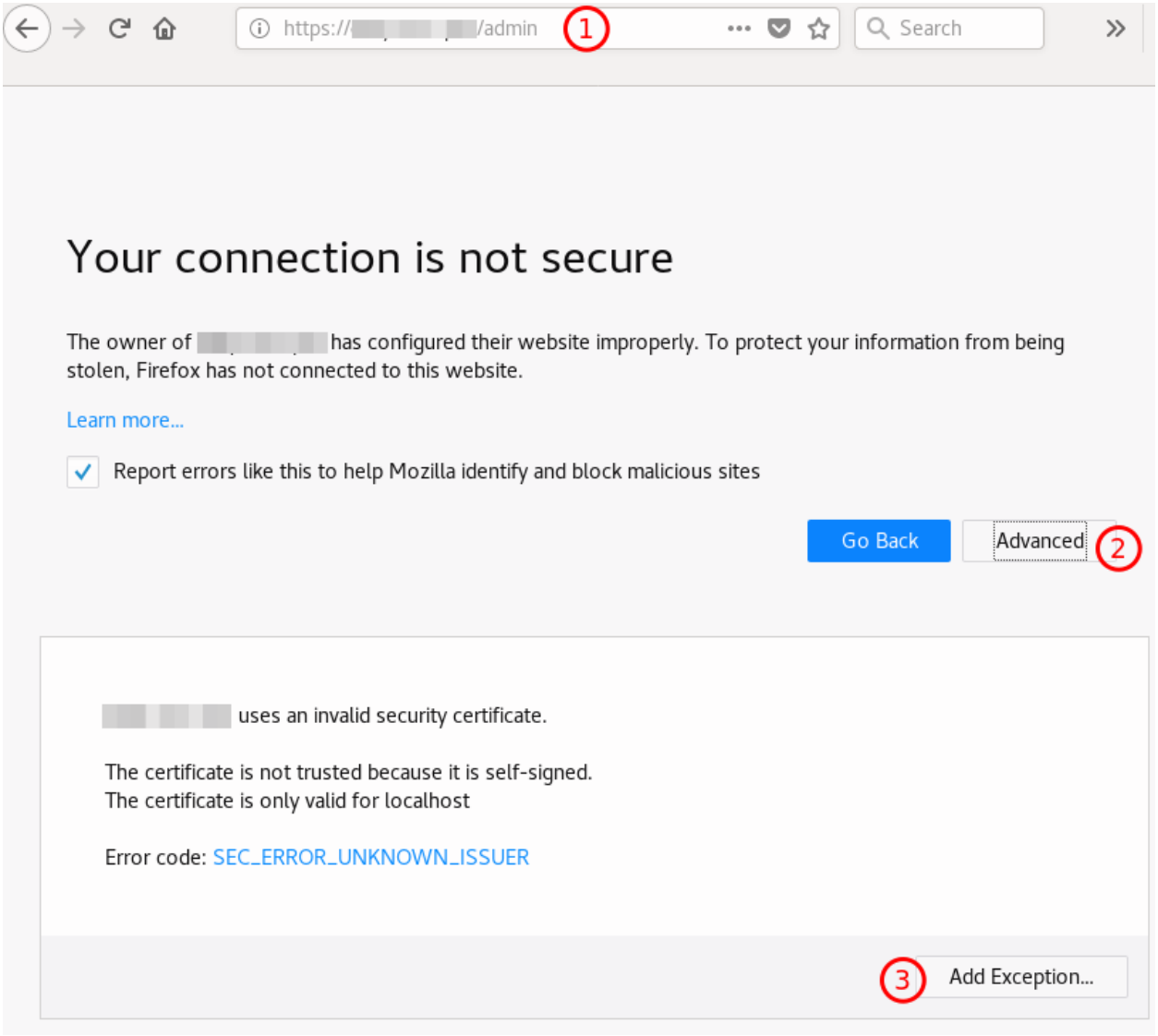
Installing plugins

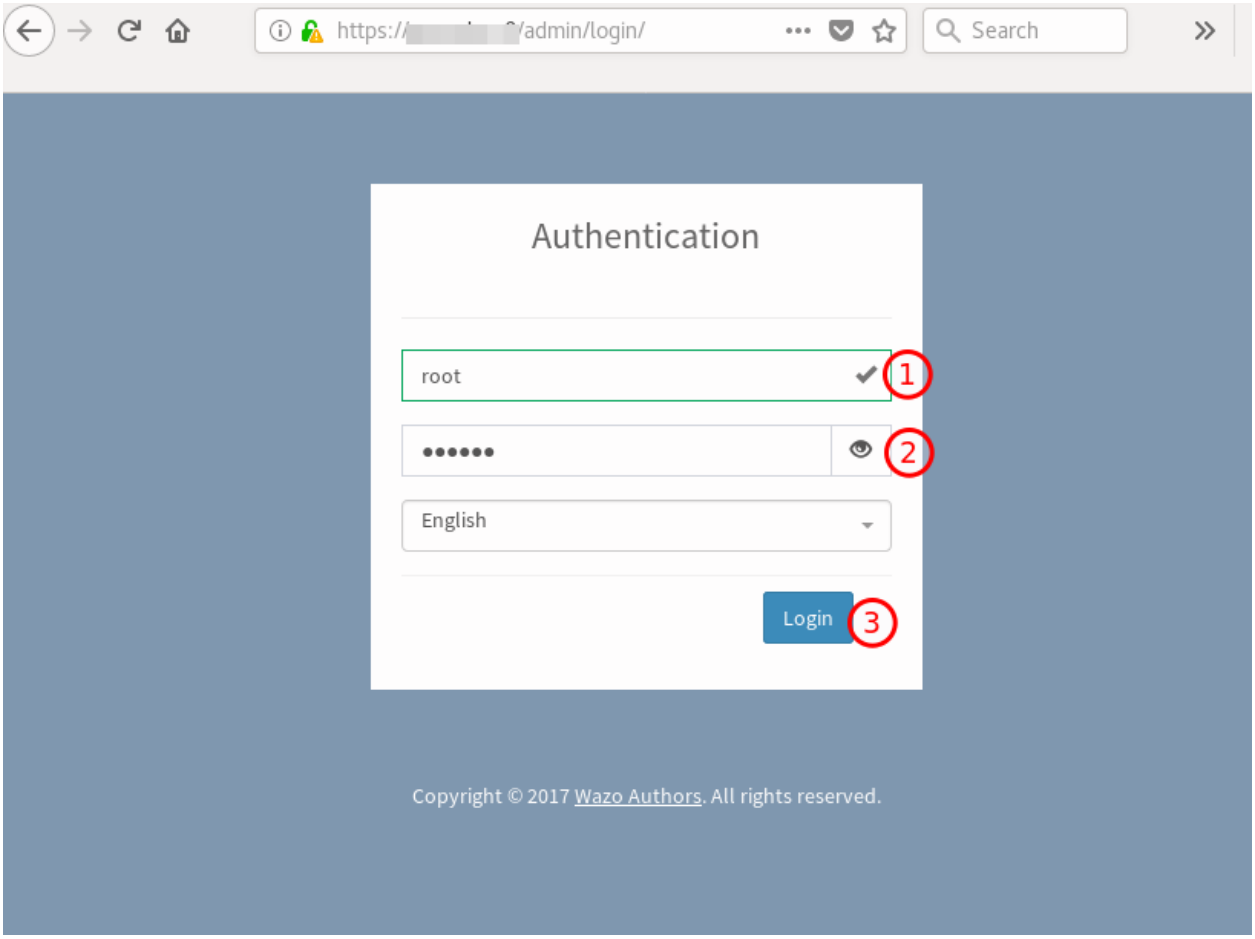
Installing a plugin will also install all of its dependencies. This process is done in the background: you can continue to use the web interface while the plugins are being installed.

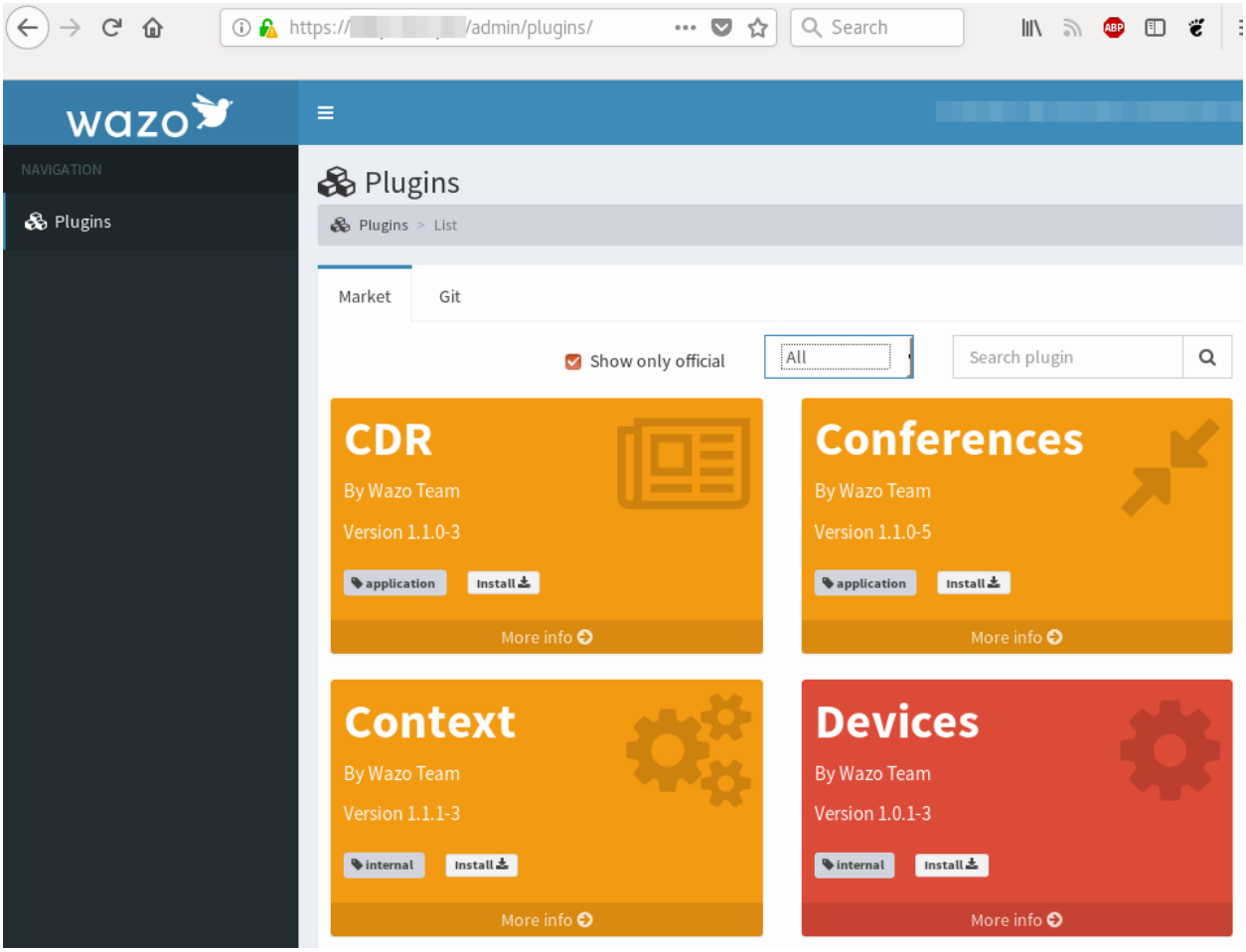
Upgrading plugins

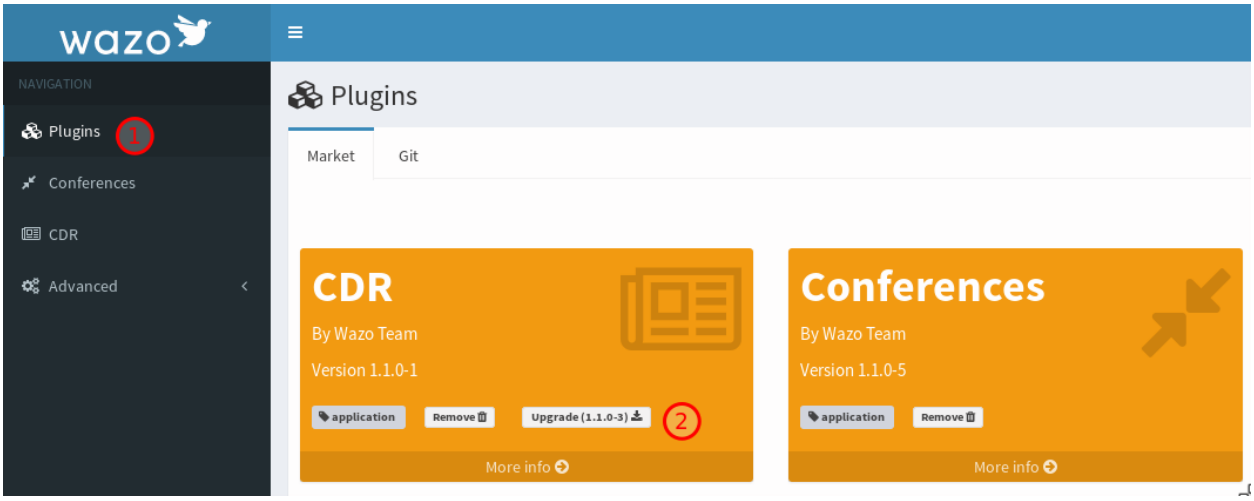
All official plugins will be automatically upgraded when you manually *upgrade Wazo*.

However, you may still upgrade individual plugins: when a plugin is installed on your system and a new version of that plugin is available, an upgrade button will be available.


https://[redacted]/admin
1
Search
Your connection is not secure
The owner of [redacted] has configured their website improperly. To protect your information from being stolen, Firefox has not connected to this website.
Learn more...
Report errors like this to help Mozilla identify and block malicious sites
Go Back
Advanced
2
[redacted] uses an invalid security certificate.
The certificate is not trusted because it is self-signed.
The certificate is only valid for localhost
Error code: SEC_ERROR_UNKNOWN_ISSUER
3
Add Exception...


https://[redacted]/admin/login/
Search
Authentication
root
1
2
English
Login
3
Copyright © 2017 Wazo Authors. All rights reserved.


https://[redacted]/admin/plugins/
Search
wazo
NAVIGATION
Plugins
Plugins
Plugins > List
Market
Git
Show only official
All
Search plugin
CDR
By Wazo Team
Version 1.1.0-3
application
Install
More info
Conferences
By Wazo Team
Version 1.1.0-5
application
Install
More info
Context
By Wazo Team
Version 1.1.1-3
internal
Install
More info
Devices
By Wazo Team
Version 1.0.1-3
internal
Install
More info


wazo
NAVIGATION
Plugins
1
Conferences
CDR
Advanced
Plugins
Market
Git
CDR
By Wazo Team
Version 1.1.0-1
application
Remove
Upgrade (1.1.0-3)
2
More info
Conferences
By Wazo Team
Version 1.1.0-5
application
Remove
More info

Dependencies

Dependencies are not upgraded when upgrading a single plugin.

For example, the plugin "user" needs the plugin "extension":

Before upgrade	Upgrades available	After upgrading user
user(1.0.1-1)	user(1.2.0-0)	user(1.2.0-0)
extension(1.5.42-0)	extension(2.0.0-4)	extension(1.5.42-0)

The "extension" plugin was not upgraded during the upgrade of the "user" plugin. You will need to upgrade the "extension" plugins as well.

Removing plugins

Removing a plugin will remove the plugin and any other plugin depending on it.

For example, removing "extension" will remove "user" since "user" depends on extension.

Writing a plugin

See *plugin development*.

1.6 Wazo Client

1.6.1 What is the Wazo Client

The Wazo Client is an application that you install on your computer and is connected to the Wazo server. This application offers the following features:

- search contacts and show their presence, phone status
- make calls through your phone (the Wazo Client is **NOT** a softphone, it is complementary to the phone)
- access your voicemail through your phone
- enable call forwards, call filtering
- show the history of your calls
- list conference rooms and members
- send faxes

It also offers some call center features:

- show screen popups or open URLs when you receive/answer a call
- list agents, queues, calls in queues
- login/logoff, pause/unpause other agents (for supervisors)
- listen/whisper to agents through you phone (for supervisors)

A lot of those features are modular and may be enabled for each user by choosing which *Xlets* they can see.

1.6.2 Getting the Wazo Client

Binaries of the Wazo Client are available on our mirror. (latest version) (all versions)

Versions

Here is the compatibility table between Wazo server and Wazo Client:

Server	Compatible with client
Until 16.12	Identical version as the server
16.13 to 17.15	16.13, 17.13, 17.14, 17.15
17.16 and later	17.16 and later

Download

Choose the version you want and in the right directory, get :

- the `.exe` file for Windows
- the `.deb` file for Ubuntu or Debian (i386 or amd64, depending on your computer)
- the `.dmg` file for Mac OS

For Windows, double-click on the file and follow the instructions. You can also install it silently:

```
wazoclient-17.XX-x86.exe /S
```

For Ubuntu/Debian, double-click on the file or execute the following command:

```
$ gdebi wazoclient-*.deb
```

For Mac OS, double-click on the file and drag-and-drop the inner file on the Application entry of the Finder.

The Wazo Client should then be available in the applications menu of each platform.

If you want to build your own Wazo Client, see *Building the Wazo Client*.

1.6.3 Connection to the server

To connect to the server using the Wazo Client you need a user name, a password and the server's address. Optionally, it is possible to login an agent while connecting to the server.

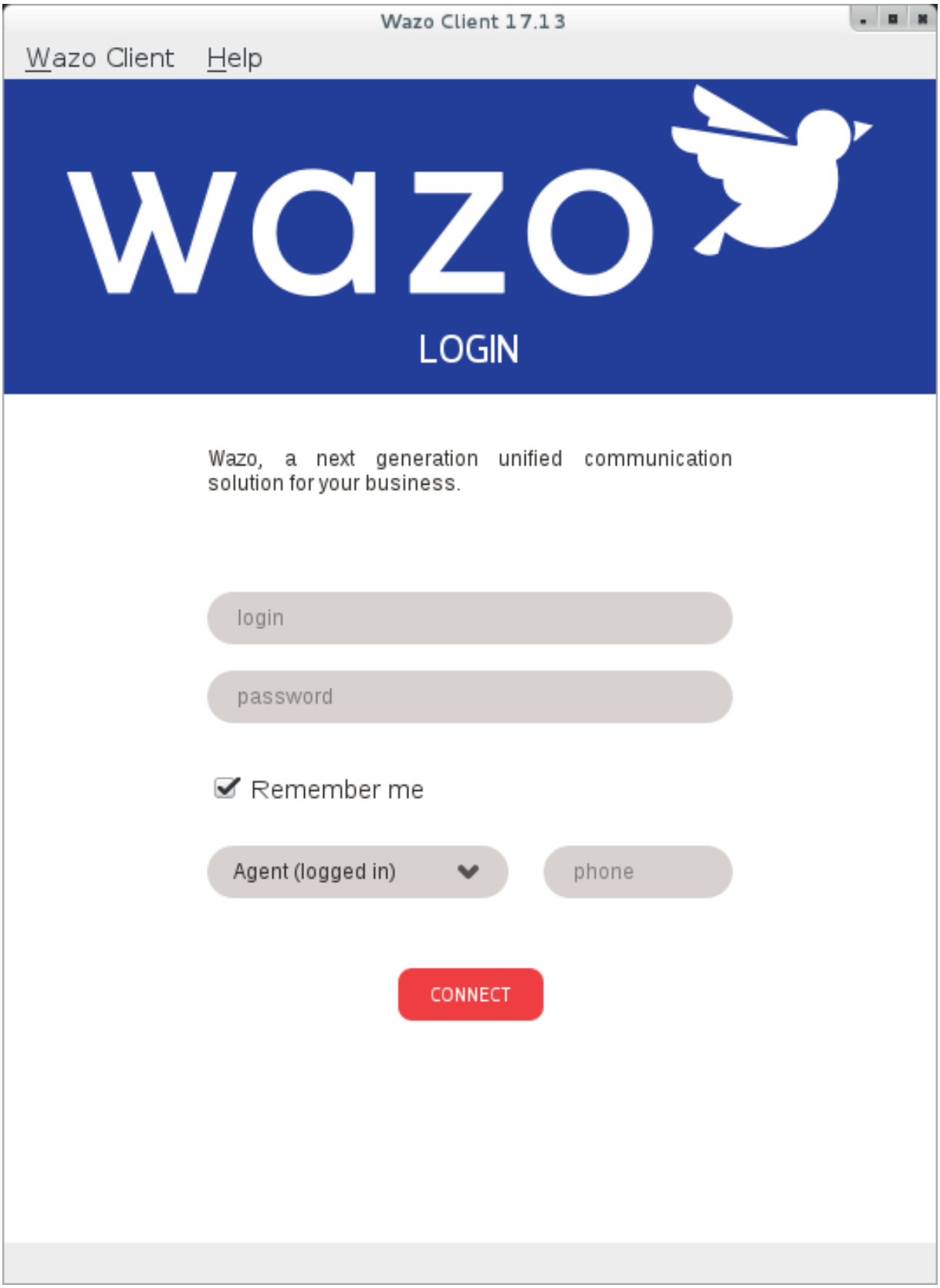
1.6.4 Xlets

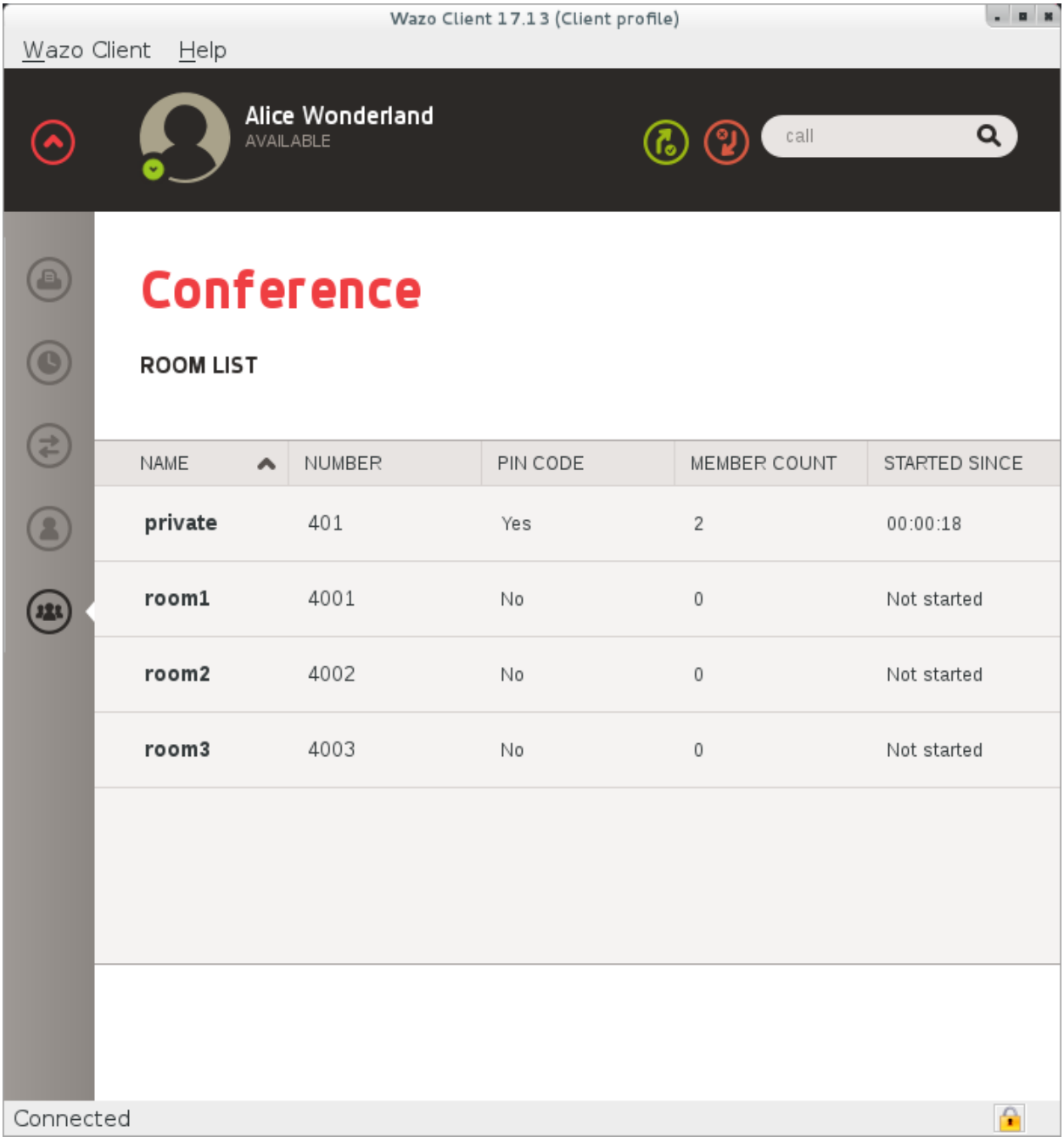
Xlets are features of the Wazo Client. It is the contraction of XiVO applets. To select which xlets are displayed in your client, see *CTI profiles*.

Conference Xlet

Overview

The conference xlet allow the user to join conferences and view conference room statuses.

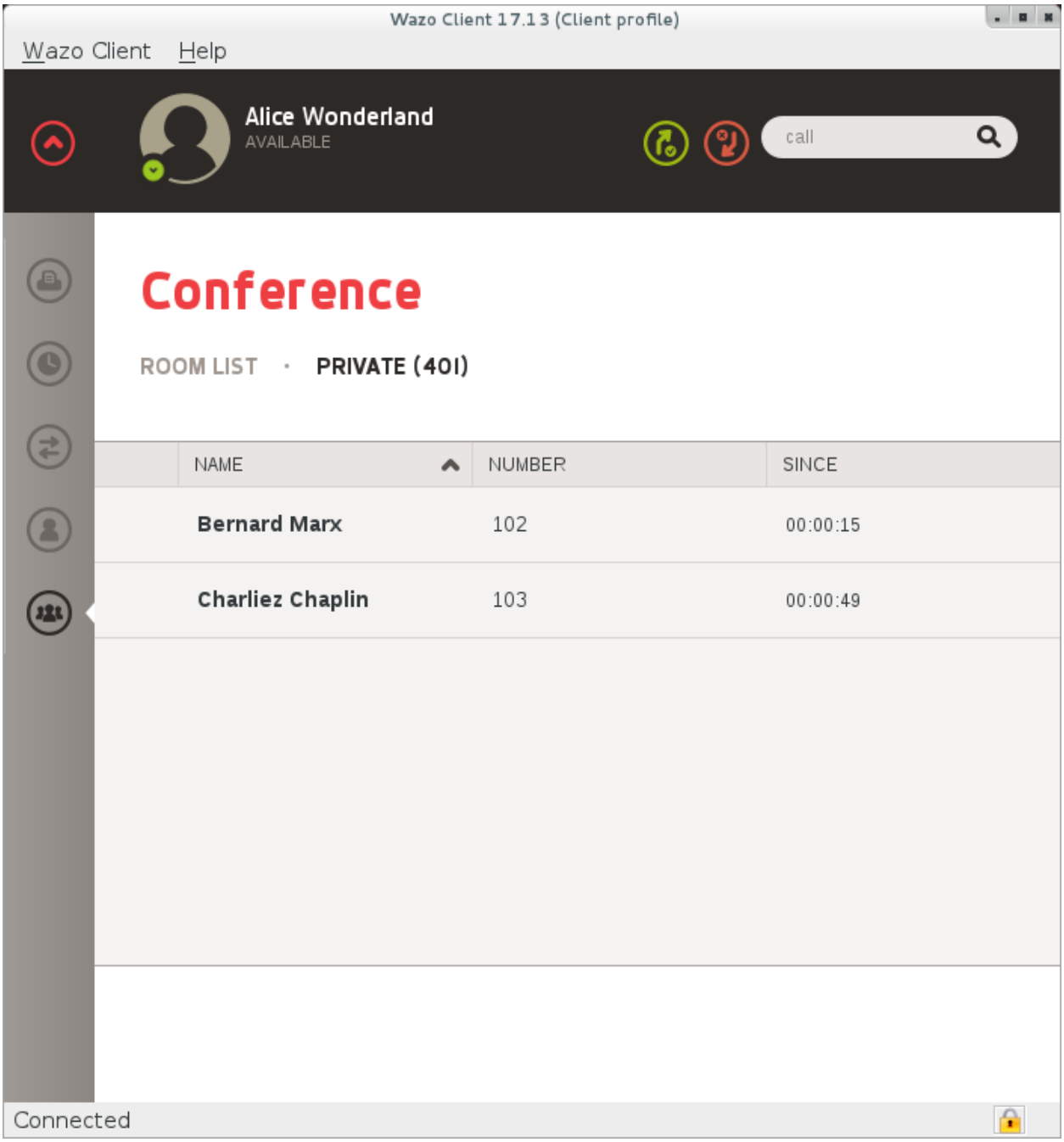

Wazo Client 17.13
Wazo Client
Help
wazo
LOGIN
Wazo, a next generation unified communication solution for your business.
login
password
Remember me
Agent (logged in)
phone
CONNECT


Wazo Client 17.13 (Client profile)
Wazo Client
Help
Alice Wonderland
AVAILABLE
call
Conference
ROOM LIST
NAME
NUMBER
PIN CODE
MEMBER COUNT
STARTED SINCE
private
401
Yes
2
00:00:18
room1
4001
No
0
Not started
room2
4002
No
0
Not started
room3
4003
No
0
Not started
Connected

Usage

The *Conference room list* tab show all available conference rooms configured on the Wazo. The user can click on a conference number to join the conference. When a user joins a conference, his phone will ring and the conference will be joined when the user answers the phone.

When clicking on a conference room a new tab is opened for the selected conference room. The new tab contains information about the members of the conference. The name and number of the member will be displayed when available. Users can also mute and unmute themselves using the microphone icon on the left.

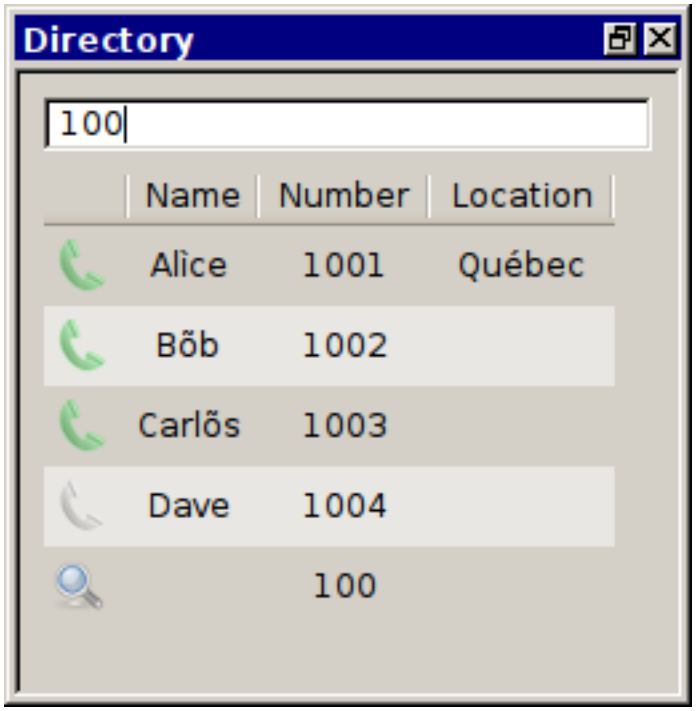


Directory Xlet

Overview

Warning: This xlet should only be used with a Switchboard profile. It is not meant to be used alone.

The goal of the directory xlet is to allow the user to search through Wazo users, directory entries and arbitrary numbers to be able to call and transfer calls to these destinations.



Usage

The list of entries in the xlet is searched using the top field. Entries are filtered by column content. The entry list will initially appear as empty.

If the current search term is a valid number, it will be displayed in the result list with no name to allow transfer to numbers that are not currently in the phonebook or configured on the Wazo.

Legend

- Users available 
- Users ringing 
- Users talking 

- Users 
- Mobile phone 
- External contacts 
- Current search (not a contact) 

Phonebook

Phonebook searches are triggered after the user has entered 3 characters. Results from remote directories will appear after 1 second.

If a directory entry as the same number as a mobile or a phone configured on the Wazo, it's extra columns will be added to the corresponding entry instead of creating a new line in the search result.

For example:

If *User 1* has number *1000* and is also in a configured LDAP with a location in "Québec", if the display filter contains the *Location* column, the entry for *User 1* will show "Québec" in the *Location* column after the search results are received.

Configuration

Context

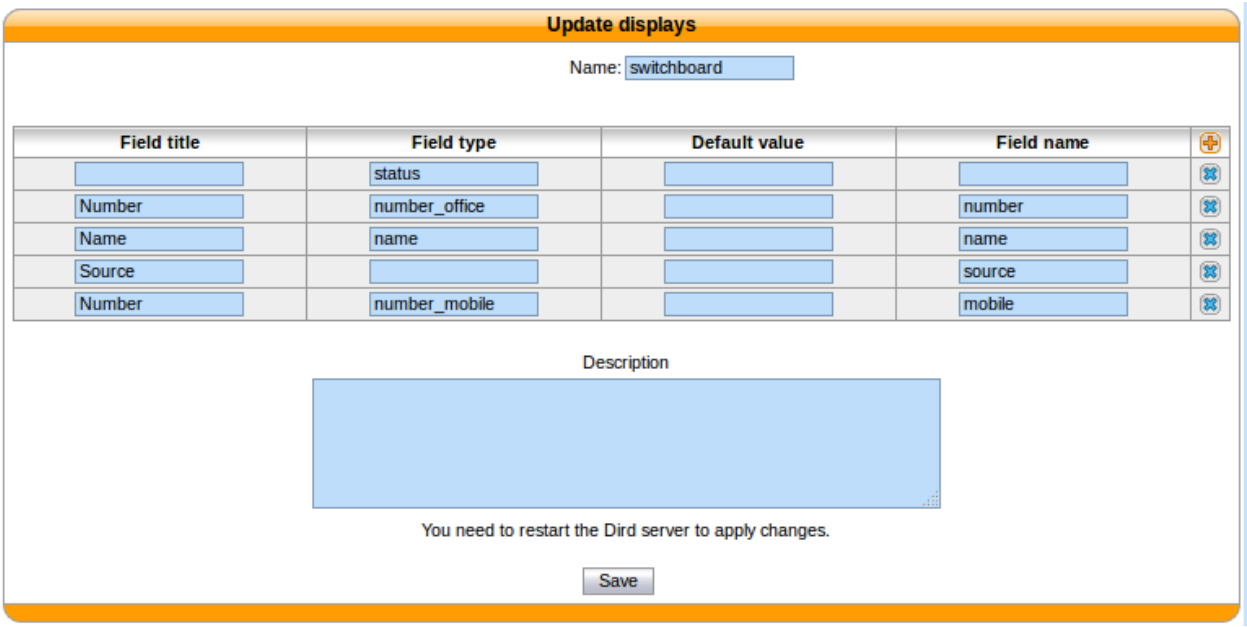
The directory xlet needs a special context named *__switchboard_directory*. In *Services* → *IPBX* → *IPBX configuration* → *Contexts* add a new context with the following parameters :

- Name : `__switchboard_directory`
- Type of context : **Other**
- Display name : Switchboard

Name	Displayed name	Context type	Entity	Action
default	default	Internal	pcm-dev (pcm-dev)	
from-extern	Incalls	Incall	pcm-dev (pcm-dev)	
invalid	invalid	Incall	pcm-dev (pcm-dev)	
pcm-dev	pcm-dev	Internal	pcm-dev (pcm-dev)	
statscenter	statscenter	Internal	pcm-dev (pcm-dev)	
__switchboard_directory	Switchboard	Other	pcm-dev (pcm-dev)	
to-extern	Outcalls	Outcall	pcm-dev (pcm-dev)	

Display filter

A new display filter must be created for the directory xlet.



The following fields must be configured with the correct value for the *Field type* column in order for entries to be displayed in the xlet:

1. *status* is the column that will be used to display the status icon, the title can be empty
2. *name* is displayed in the *Name* column of the xlet
3. *number_office* is displayed in the *Number* column with a phone icon in the xlet
4. *number_mobile* is displayed in the *Number* column with a mobile icon in the xlet
5. *number_...* any other field starting with *number_* will be displayed in the *Number* column of the xlet with a generic directory icon
6. Any other field will be displayed in their own column of the directory xlet

The values in the *Field name* column must contain values that were created in the *Directory definition*.

The title used for the *Number* column is the title of the first field whose type starts with *number_*.

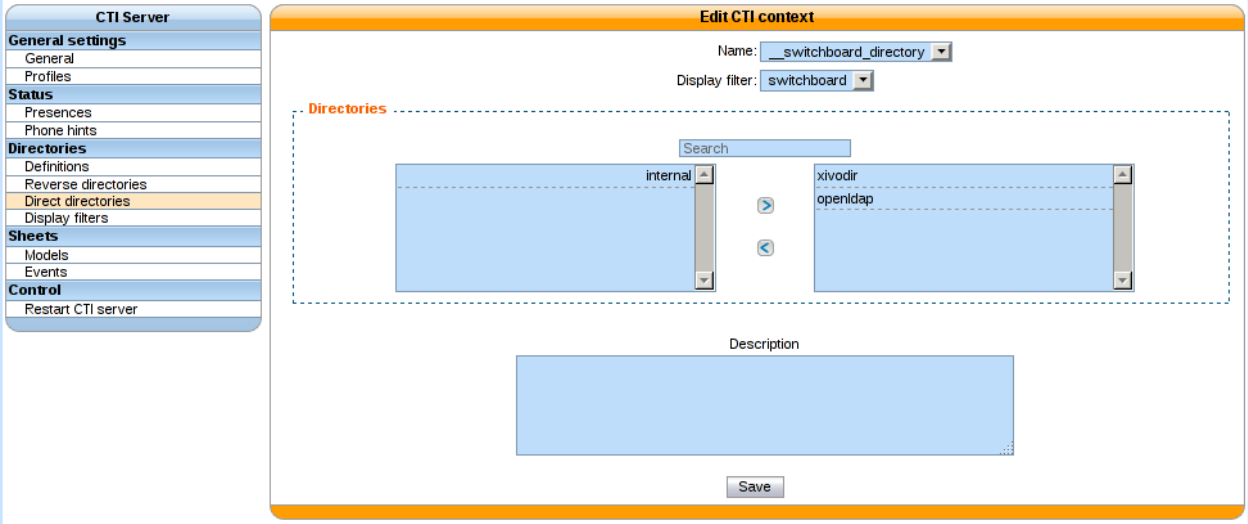
Note: The field title of the first number column will be used for the header title in the xlet.

Warning: Make sure that the fields entered in the display format are also available in the directory definition, otherwise the filter will not return any results

Context and filter association

The new *Display filter* has to be assigned to the *__switchboard_directory* context

You can then choose which directories will be searched by the Xlet.



Warning: You must **not select internal** directory, as it is already handled.

Fax Xlet

Overview

The Fax xlet allows the user to send faxes from his Wazo Client.

Usage

The *Choose a file to send* field is used to select which file you want to send. Only PDF documents are supported.

The *Choose destination number* field is the fax destination, directory search can be used to find the fax number in available directories.

History Xlet

Overview

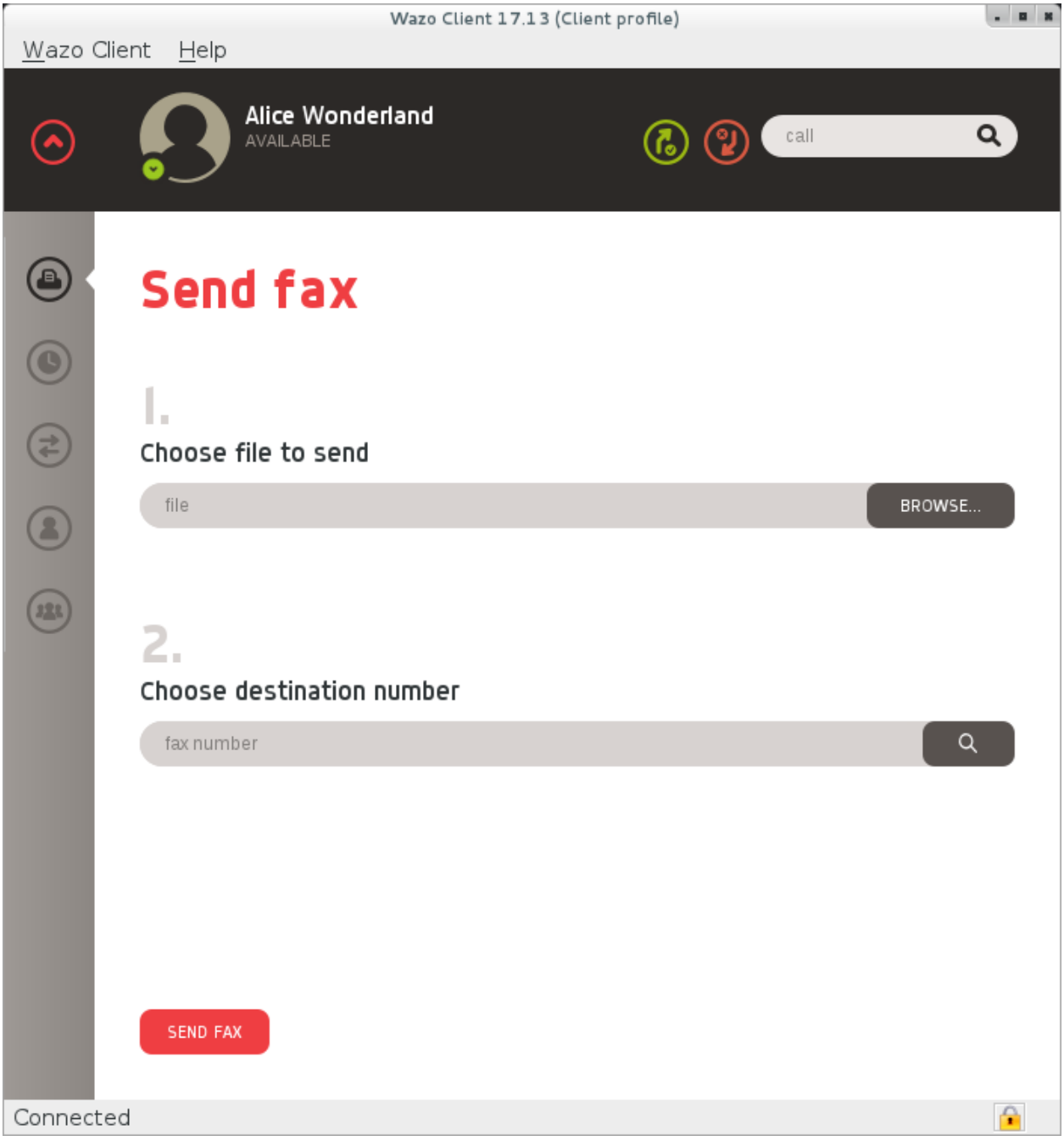
The history xlet allow the user to view his last calls. The user can filter by sent, received and missed calls.

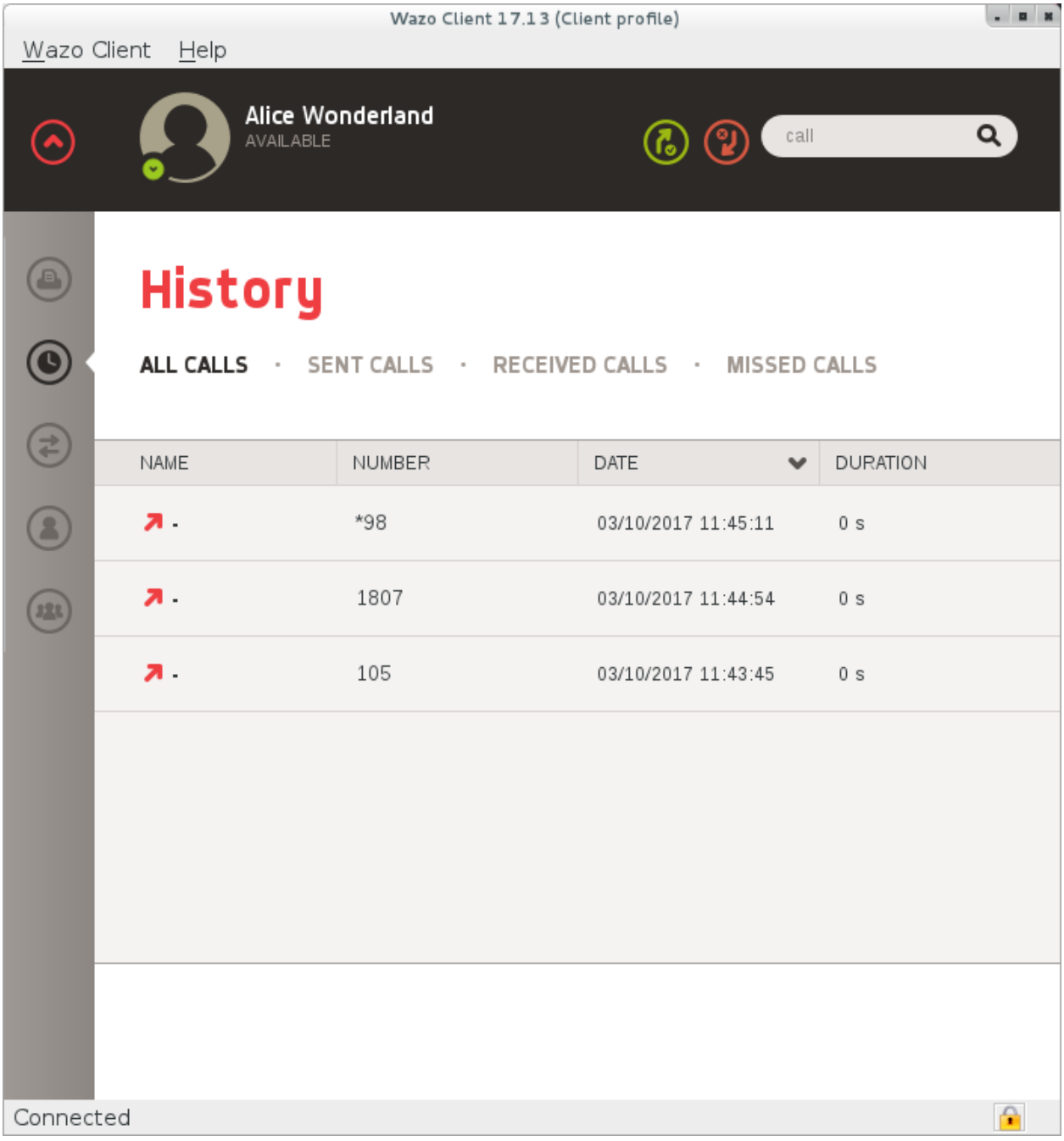
Usage

The user can click on the number to initiate a new call with a given correspondent.

Warning:

- The column content is only refreshed when moving from one view to the other.
- The *Sent calls* tab displays only the phone number of the called party (the name column will be void).

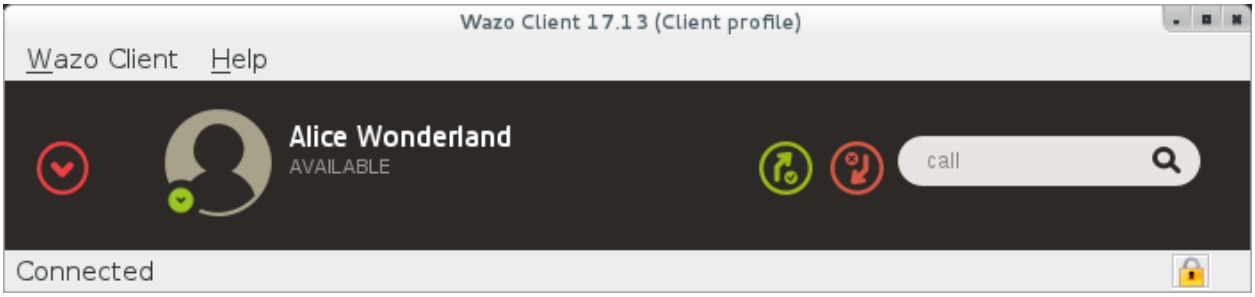

Wazo Client 17.13 (Client profile)
Wazo Client
Help
Alice Wonderland
AVAILABLE
call
Send fax
1.
Choose file to send
file
BROWSE...
2.
Choose destination number
fax number
SEND FAX
Connected


Wazo Client 17.13 (Client profile)
Wazo Client
Help
Alice Wonderland
AVAILABLE
call
History
ALL CALLS
SENT CALLS
RECEIVED CALLS
MISSED CALLS
NAME
NUMBER
DATE
DURATION
-
*98
03/10/2017 11:45:11
0 s
-
1807
03/10/2017 11:44:54
0 s
-
105
03/10/2017 11:43:45
0 s
Connected

Identity Xlet

Overview

The Identity Xlet allows you to make calls from your computer, via your phone. This means that you can enter the number that you want to dial on your computer, then your phone rings and when you answer it, the called phone will ring.



Usage

You can enter the number you want to dial in the text box and then click the button or press enter to dial it.

If you dial an invalid extension (a number is an extension), your phone will ring and you will be told that the extension is not valid.

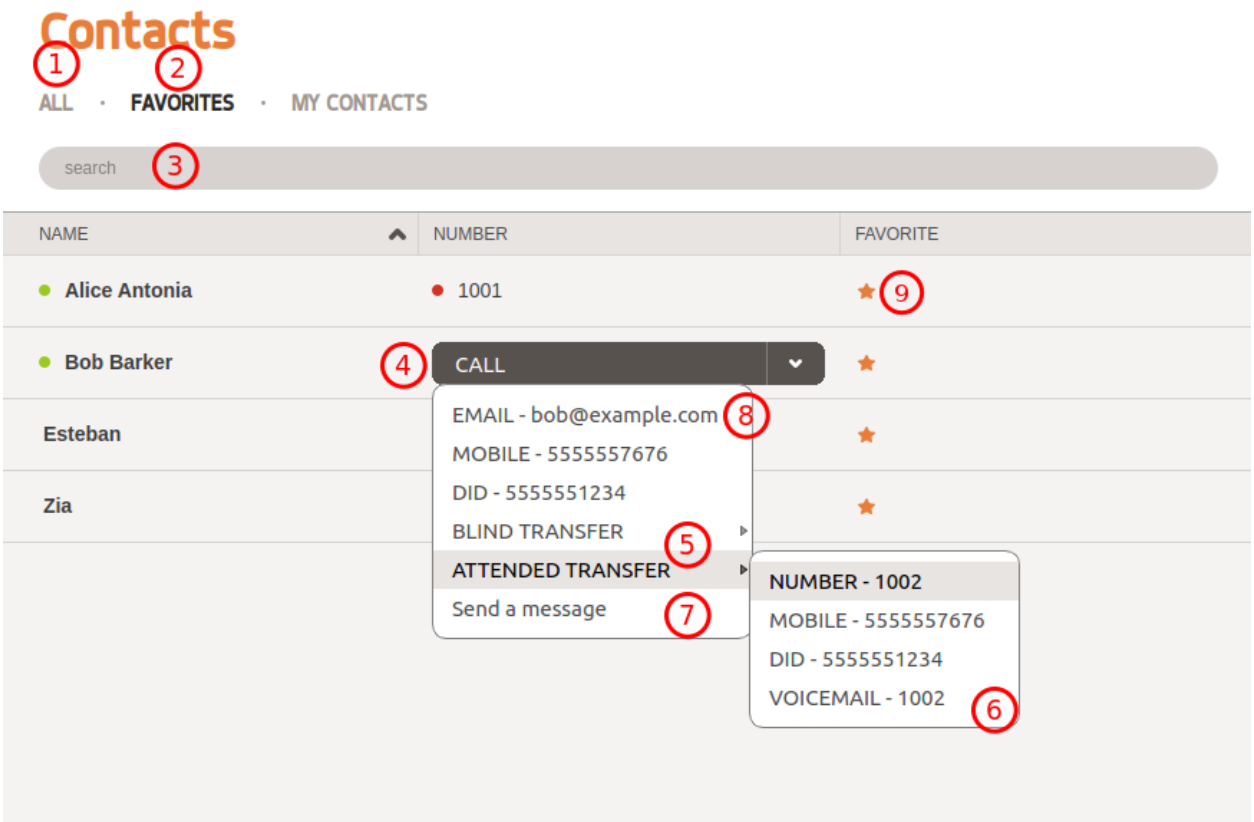
People Xlet

Overview

The People XLet lists the people of your company and personal contacts, giving you access to their phone, status and other information configured by the administrator.

1. Display results of the search
2. Display favorite contacts
3. Search contacts
4. Call a contact
5. Transfer a call to a contact
6. Transfer a call to a contact's voicemail
7. Chat with a contact
8. Send an email to a contact
9. Bookmark/unmark the contact as a favorite

1. View all personal contacts
2. Edit or remove a personal contact
3. Create a personal contact
4. Import personal contacts from a CSV file
5. Export personal contacts to a CSV file



6. Delete all personal contacts

1. Wazo Client status (see *Presence Option*)
2. Phone status (see *Services* → *CTI Server* → *Status* → *Phone hints* page)
3. Agent status (logged in or logged out)

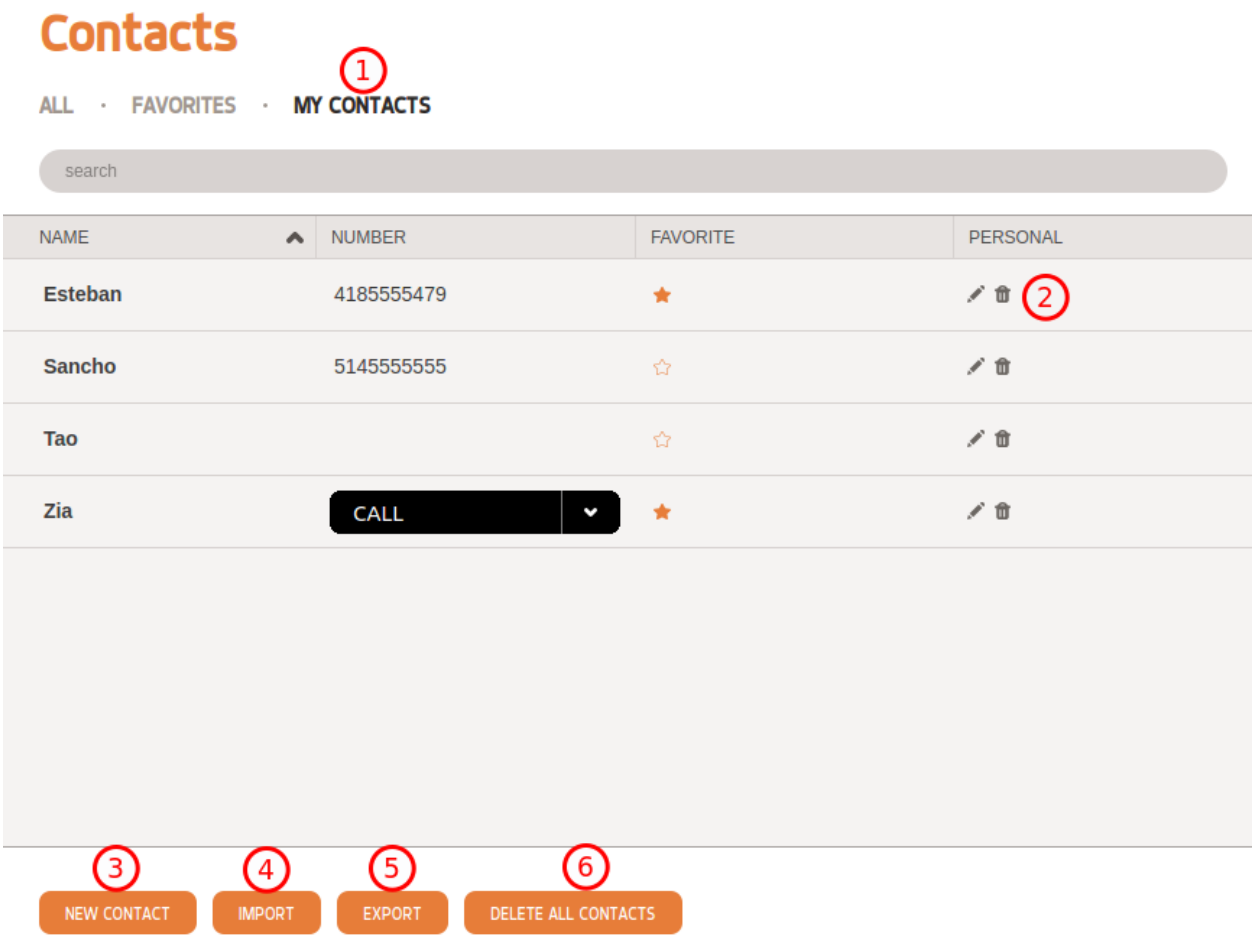
Note: Most information (e.g. columns displayed, allowed actions, searched directories, etc.) is configurable through the *web interface*.

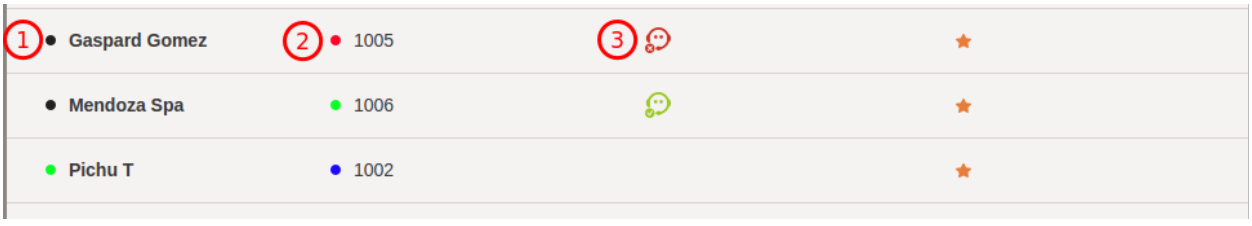
Importing contacts via CSV file

Imported files should have the following structure:

```
firstname,lastname,number,email,company,fax,mobile
John,Doe,5555551111,my@email,wazo,5555552222,5555553333
```

- The field order is not important.
- The file must be encoded in `UTF-8`.
- Invalid lines of the CSV file will be skipped and an error will be displayed in the import report.


Contacts
1
ALL · FAVORITES · MY CONTACTS
search
NAME
NUMBER
FAVORITE
PERSONAL
Esteban
4185555479
2
Sancho
5145555555
Tao
Zia
CALL
3
NEW CONTACT
4
IMPORT
5
EXPORT
6
DELETE ALL CONTACTS


1
Gaspard Gomez
2
1005
3
Mendoza Spa
1006
Pichu T
1002

Exporting contacts via CSV file

The file has the same structure as the import file, with a supplementary field: `id`, which is the internal contact ID from Wazo.

- The first line (the list of field names) is ordered in alphabetical order.
- The file will be encoded in `UTF-8`.

Copying the number or email address

It is possible to copy a contact's number or email address to the system's clipboard. To do so, right click on a contact's action menu and select the value you wish to copy.

Note: When using a mac without a right mouse button use *ctrl-Left click* to show the copy menu.

Service Xlet

Overview

The service xlet allows the user to enable and disable telephony services such as call forwarding, call filter and do not disturb.

Configuration

The available service list is configured from the web interface in *Services* → *CTI Server* → *General settings* → *Profiles*.

The right side of the *Services* section contains services that are available to a given profile.

1.6.5 Configuration

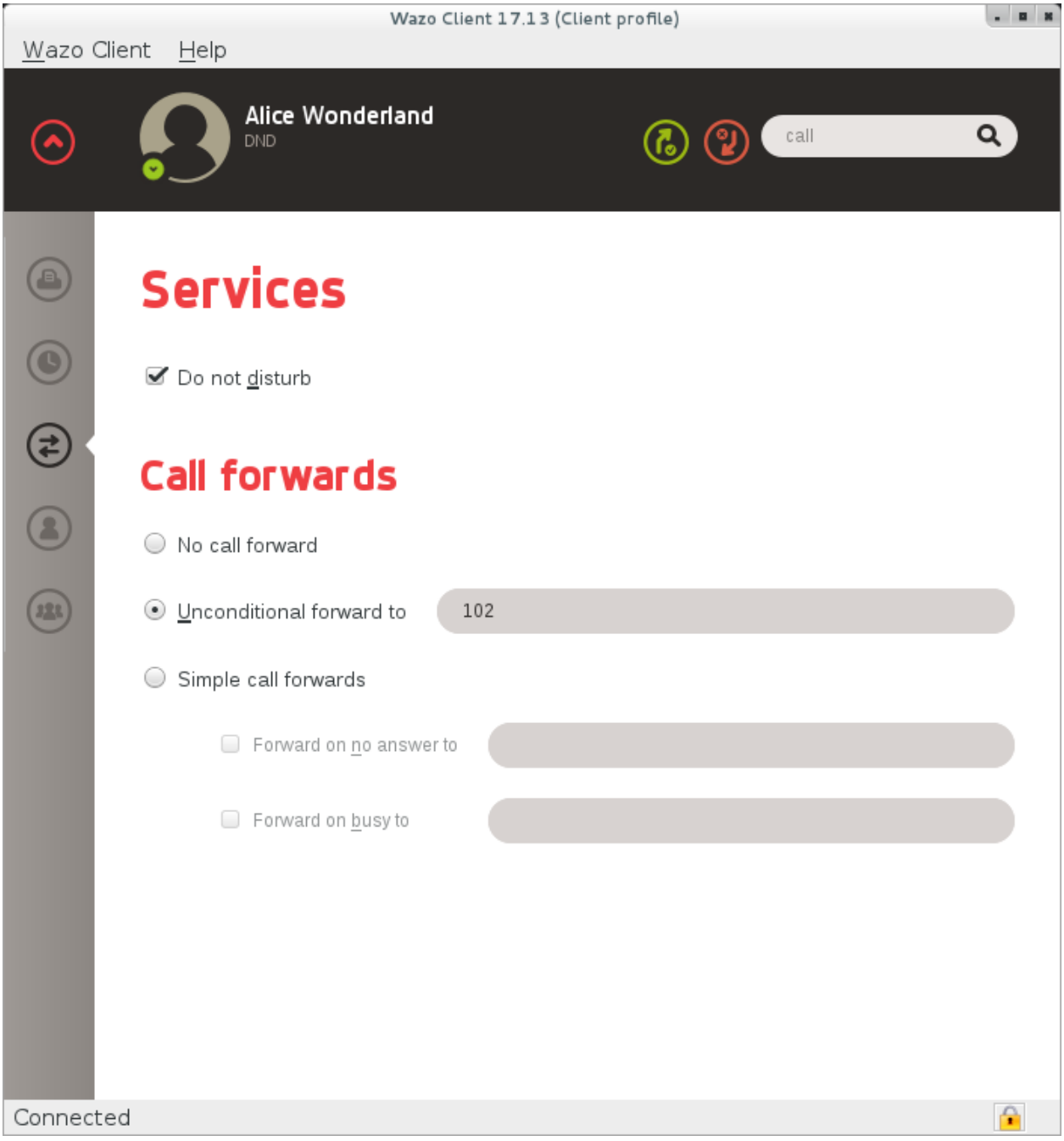
The Wazo Client configuration options can be accessed under *Wazo Client* → *Configure*.

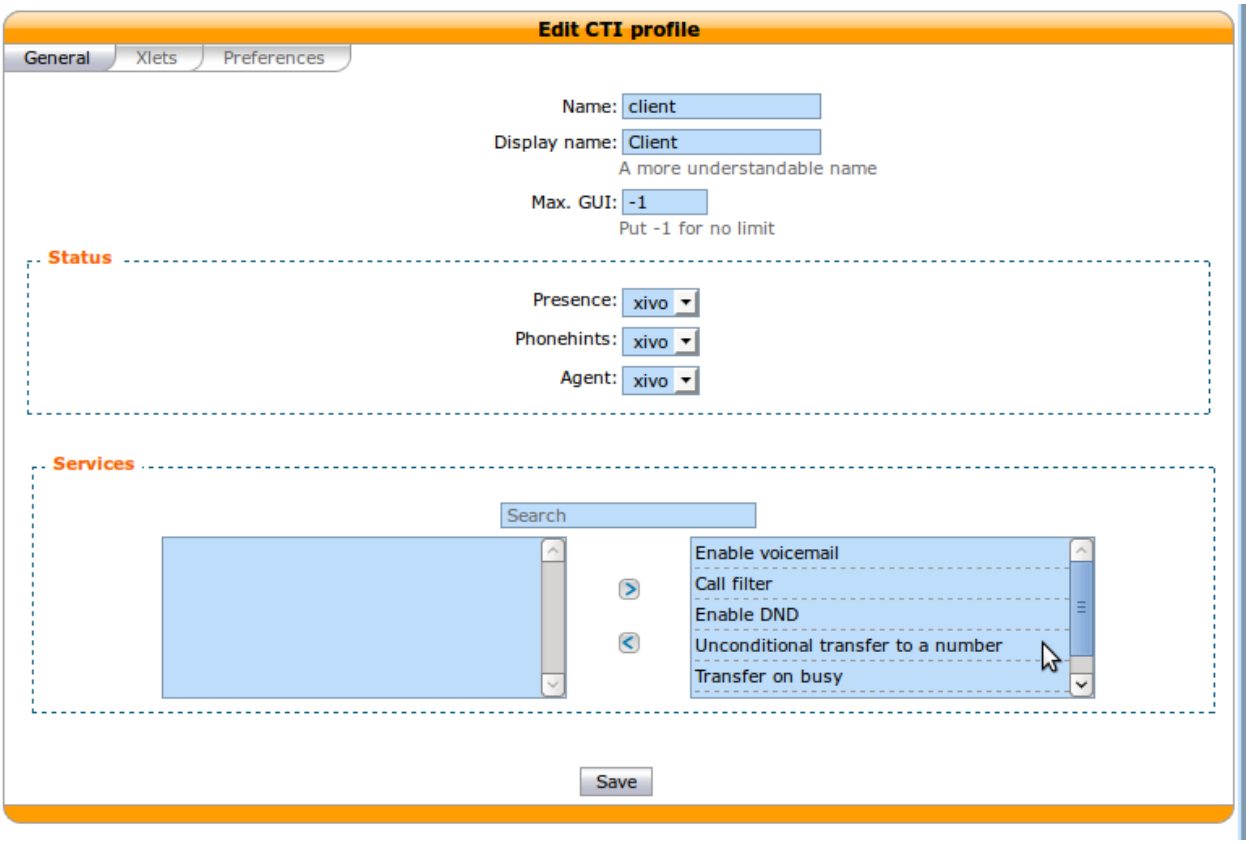
Connection Configuration

This page allows the user to set his network information to connect to the xivo-ctid server.

- *Server* is the IP address of the server.
- *Backup server* is the IP address of the backup server.
- *Port* is the port on which xivo-ctid is listening for connections. (default: 5003)
- *STARTTLS* is used to specify that a secure connect should be used

Note: To use STARTTLS, the server needs to be configured to *accept encrypted connection*.


Wazo Client 17.13 (Client profile)
Wazo Client
Help
Alice Wonderland
DND
call
Services
Do not disturb
Call forwards
No call forward
Unconditional forward to
102
Simple call forwards
Forward on no answer to
Forward on busy to
Connected



1.6.6 Handling callto: and tel: URLs

The Wazo Client can handle telephone number links that appear in web pages. The client will automatically dial the number when you click on a link.

Note: You must already be logged in for automatic dialing to work, otherwise the client will simply start up and wait for you to log in.

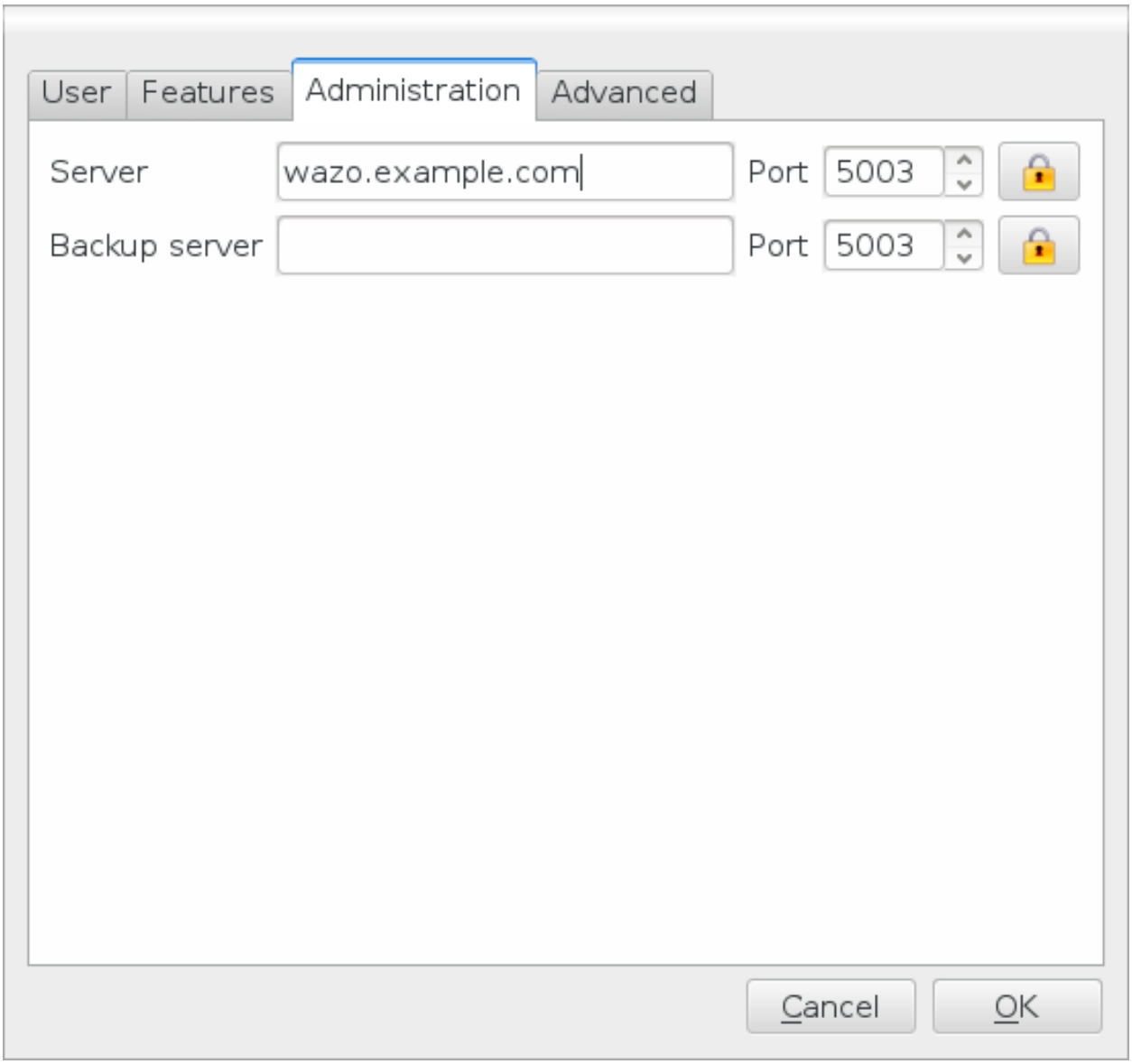
Warning: The option in the Wazo Client *Advanced → Allow multiple instances of Wazo Client* must be disabled, else you will launch one new Wazo Client with every click.

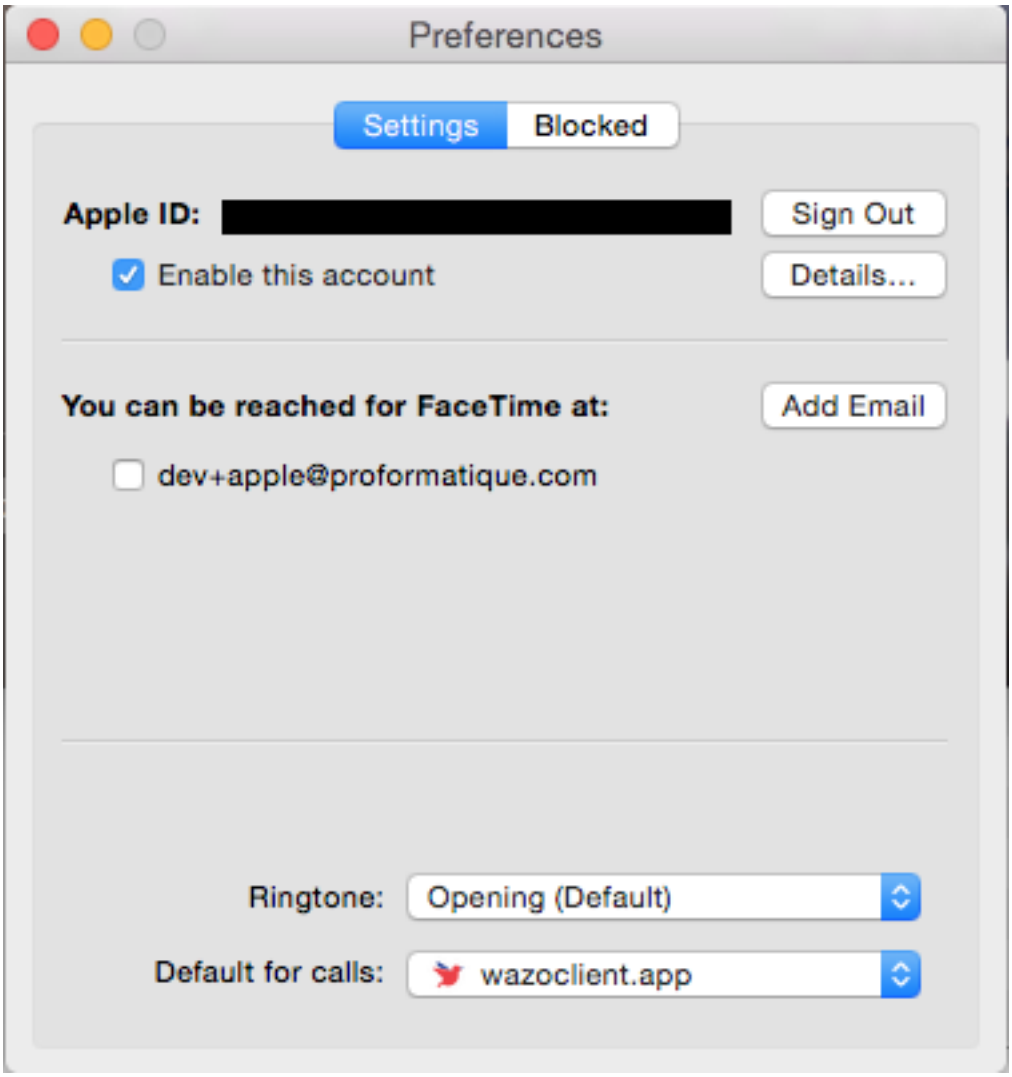
Mac OS

`callto:` links will work out-of-the-box in Safari and other web browsers after installing the client.

`tel:` links will open FaceTime after installing the client. To make the Wazo Client the default application to open `tel:` URLs in Safari.

1. Open the FaceTime application
2. Connect using your apple account
3. Open the FaceTime preferences
4. Change the *Default for calls* entry to *wazoclient.app*


User
Features
Administration
Advanced
Server
wazo.example.com
Port
5003
Backup server
Port
5003
Cancel
OK

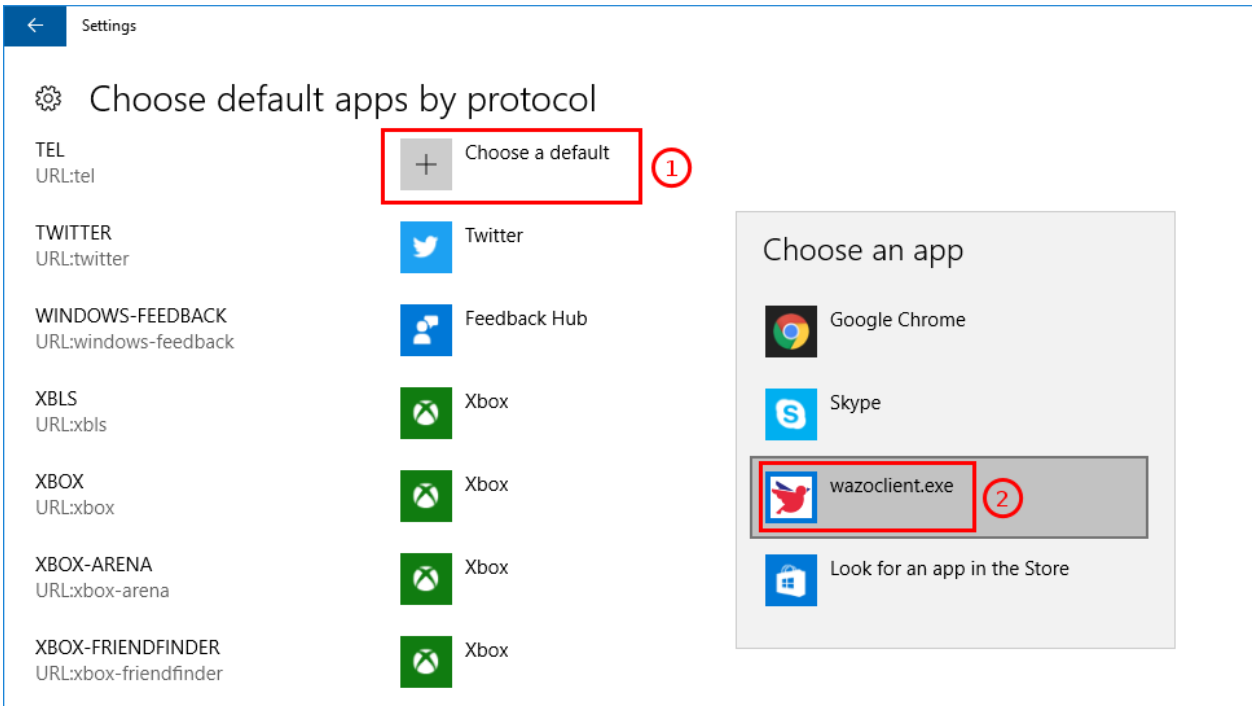

Preferences
Settings
Blocked
Apple ID:
Sign Out
Enable this account
Details...
You can be reached for FaceTime at:
Add Email
dev+apple@proformatique.com
Ringtone:
Opening (Default)
Default for calls:
wazoclient.app

Note: The `tel:` URL works out-of-the-box in versions of mac osx before 10.10.

Windows

Wazo Client is associated with `callto:` and `tel:` upon installation. Installing other applications afterward could end up overriding these associations.

- Windows Vista: *Start menu* → *Control Panel* → *Default Programs*, subsection Protocols
- Windows 7: *Start menu* → *Control Panel* → *Default Programs*, subsection Protocols
- Windows 10: *Start menu* → *Settings* → *System* → *Default apps* → *Choose default apps by protocol*



Ubuntu

Currently, `callto:` or `tel:` links are only supported in Firefox. There is no configuration needed.

GNU/Linux Debian

Currently, `callto:` or `tel:` links are only supported in Firefox. If the Wazo Client is not listed in the proposition when you open the link, browse your files to find `/usr/bin/wazoclient`.

Manual association in Firefox

If, for some reason, Firefox does not recognize `callto:` or `tel:` URIs you can manually associate them to the Wazo Client using the following steps:

1. Type `about:config` in the URL bar

2. Click the *I'll be careful, I promise !* button to close the warning
3. Right-click anywhere in the list and select *New* -> *Boolean*
4. Enter `network.protocol-handler.external.callto` as preference name
5. Select `false` as value
6. Repeat steps 3 to 6, but replace `callto` by `tel` at step 4

The next time that you click on a telephone link, Firefox will ask you to choose an application. You will then be able to choose the Wazo Client for handling telephone numbers.

1.7 System

1.7.1 DHCP Server

Wazo includes a DHCP server used for assisting in the provisioning of phones and other devices. (See *Basic Configuration* for the basic setup). This section contains additional notes on how to configure more advanced options that may be helpful when integrating the server with different VOIP subnets.

Activating DHCP on another interface

DHCP Server can be activated through the Wazo Web Interface *Configuration* → *Network* → *DHCP* :

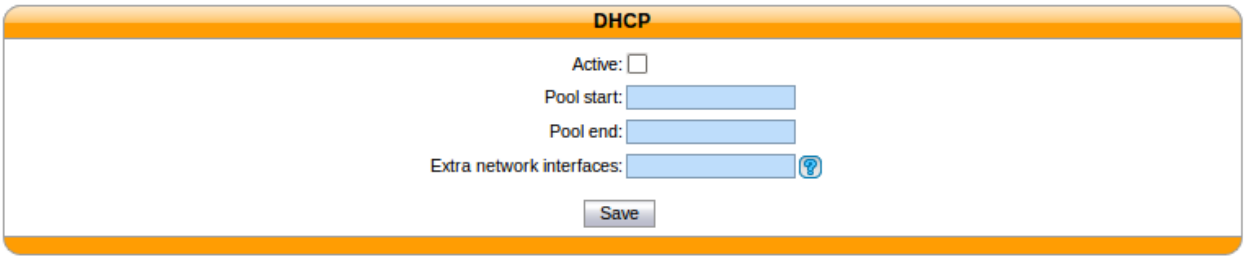


Fig. 1.18: *Configuration* → *Network* → *DHCP*

By default, it will only answer to DHCP requests coming from the VoIP subnet (defined in the *Configuration* → *Network* → *Interfaces* section). If you need to activate DHCP on an other interface, you have to fill in the *Extra network interfaces* field with the interface name , for example : `eth0`

After saving your modifications, click on *Apply system configuration* so that the new settings can take effect.

Changing default DHCP gateway

By default, the Wazo DHCP server uses the Wazo's IP address as the routing address. To change this you must create a custom-template:

1. Create a custom template for the `dhcpd_subnet.conf.head` file:

```
mkdir -p /etc/xivo/custom-templates/dhcp/etc/dhcp/
cd /etc/xivo/custom-templates/dhcp/etc/dhcp/
cp /usr/share/xivo-config/templates/dhcp/etc/dhcp/dhcpd_subnet.conf.head .
```

2. Edit the custom template:

```
vim dhcpd_subnet.conf.head
```

3. In the file, replace the string `#XIVO_NET4_IP#` by the routing address of your VoIP network, for example:

```
option routers 192.168.2.254;
```

4. Re-generate the dhcp configuration:

```
xivo-update-config
```

DHCP server should have been restarted and should now use the new routing address.

Configuring DHCP server to serve unknown hosts

By default, the Wazo DHCP server serves only known hosts. That is:

- either hosts which MAC address prefix (the OUI) is known
- or hosts which Vendor Identifier is known

Known OUIs and Vendor Class Identifiers are declared in `/etc/dhcp/dhcpd_update/*` files.

If you want your Wazo DHCP server to serve also unknown hosts (like PCs) follow these instructions:

1. Create a custom template for the `dhcpd_subnet.conf.tail` file:

```
mkdir -p /etc/xivo/custom-templates/dhcp/etc/dhcp/
cd /etc/xivo/custom-templates/dhcp/etc/dhcp/
cp /usr/share/xivo-config/templates/dhcp/etc/dhcp/dhcpd_subnet.conf.tail .
```

2. Edit the custom template:

```
vim dhcpd_subnet.conf.tail
```

3. And add the following line at the head of the file:

```
allow unknown-clients;
```

4. Re-generate the dhcp configuration:

```
xivo-update-config
```

DHCP server should have been restarted and should now serve all network equipments.

DHCP-Relay

If your telephony devices aren't located on the same site and the same broadcast domain as the Wazo DHCP server, you will have to add the option *DHCP Relay* to the site's router. This parameter will allow the DHCP requests from distant devices to be transmitted to the IP address you specify as DHCP Relay.

Warning: Please make sure that the IP address used as DHCP Relay is the same as one of Wazo's interfaces, and that this interface is configured to listen to DHCP requests (as decribed in previous part). Also verify that routing is configured between the distant router and the choosen interface, otherwise DHCP requests will never reach the Wazo server.

Configuring DHCP server for other subnets

This section describes how to configure Wazo to serve other subnets that the VOIP subnet. As you can't use the Web Interface to declare other subnets (for example to address DATA subnet, or a VOIP subnet that isn't on the same site that Wazo server), you'll have to do the following configuration on the Command Line Interface.

Creating "extra subnet" configuration files

First thing to do is to create a directory and to copy into it the configuration files:

```
mkdir /etc/dhcp/dhcpd_sites/
cp /etc/dhcp/dhcpd_subnet.conf /etc/dhcp/dhcpd_sites/dhcpd_siteXXX.conf
cp /etc/dhcp/dhcpd_subnet.conf /etc/dhcp/dhcpd_sites/dhcpd_lanDATA.conf
```

Note: In this case we'll create 2 files for 2 differents subnets. You can change the name of the files, and create as many files as you want in the folder `/etc/dhcp/dhcpd_sites/`. Just adapt this procedure by changing the name of the file in the different links.

After creating one or several files in `/etc/dhcp/dhcpd_sites/`, you have to edit the file `/etc/dhcp/dhcpd_extra.conf` and add the according include statement like:

```
include "/etc/dhcp/dhcpd_sites/dhcpd_siteXXX.conf";
include "/etc/dhcp/dhcpd_sites/dhcpd_lanDATA.conf";
```

Adjusting Options of the DHCP server

Once you have created the subnet in the DHCP server, you must edit each configuration file (the files in `/etc/dhcp/dhcpd_sites/`) and modify the different parameters. In section **subnet**, write the IP subnet and change the following options (underlined fields in the example):

```
subnet 172.30.8.0 netmask 255.255.255.0 {
```

- subnet-mask:

  ```
  option subnet-mask 255.255.255.0;
  ```

- broadcast-address:

  ```
  option broadcast-address 172.30.8.255;
  ```

- routers (specify the IP address of the router that will be the default gateway of the site):

  ```
  option routers 172.30.8.1;
  ```

In section **pool**, modify the options:

```
pool {
```

- log (add the name of the site or of the subnet):

  ```
  log(concat("[", binary-to-ascii(16, 8, ":", hardware), "] POOL VoIP Site XXX"));
  ```

- range (it will define the range of IP address the DHCP server can use to address the devices of that subnet):

```
range 172.30.8.10 172.30.8.200;
```

Warning: Wazo only answers to DHCP requests from *supported devices*. In case of you need to address other equipment, use the option *allow unknown-clients;* in the `/etc/dhcp/dhcpd_sites/` files

At this point, you can apply the changes of the DHCP server with the command:

```
service isc-dhcp-server restart
```

After that, Wazo will start to serve the DHCP requests of the devices located on other sites or other subnets than the VOIP subnet. You will see in `/var/log/daemon.log` all the DHCP requests received and how they are handled by Wazo.

1.7.2 Mail

This section describes how to configure the mail server shipped with Wazo (Postfix) and the way Wazo handles mails.

In *Configuration* → *Network* → *Mail*, the following options can be configured:

- *Domain Name messaging* : the server's displayed domain. Will appear in "Received" mail headers.
- *Source address of the server* : domain part of headers "Return-Path" and "From".
- *Relay SMTP* and *FallBack relay SMTP* : relay mail servers.
- *Rewriting shipping addresses* : Canonical address Rewriting. See Postfix canonical documentation for more info.

Warning: Postfix, the mail server shipped with Wazo, should be stopped on an installed Wazo with no valid and reachable DNS servers configured. If Postfix is not stopped, messages will bounce in queues and could end up affecting core pbx features.

If you need to disable Postfix here is how you should do it:

```
systemctl stop postfix
systemctl disable postfix
```

If you ever need to enable Postfix again:

```
systemctl enable postfix
systemctl start postfix
```

Alternatively, you can empty Postfix's queues by issuing the following commands on the Wazo server:

```
postsuper -d ALL
```

Configure Wazo for authenticated SMTP

Let's say we want to send mails from Wazo through the following SMTP server:

- SMTP host: `smtp.example.com`

- SMTP port: `587`
- SMTP user: `smtp_user@example.com`
- SMTP password: `smtp_password`

Install the required dependencies:

```
apt-get update && apt-get install libsasl2-modules sasl2-bin
```

In *Configuration → Mail*:

- set *Relay SMTP* to `smtp.example.com:587`
- set *Source address of the server* to `example.com`
- set *Rewriting shipping addresses* to something like `asterisk smtp_user\nroot smtp_user\nxivo-agid smtp_user` (you must leave the `\n` verbatim)

Then apply the changed in *Configuration → Apply system configuration*.

Create a custom template for Postfix configuration:

```
mkdir -p /etc/xivo/custom-templates/mail/etc/postfix
cp /usr/share/xivo-config/templates/mail/etc/postfix/main.cf /etc/xivo/custom-
↪templates/mail/etc/postfix/main.cf
```

Then, add at the end of the file `/etc/xivo/custom-templates/mail/etc/postfix/main.cf`:

```
smtp_sasl_auth_enable = yes
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
smtp_sasl_security_options = noanonymous
```

Create the file `/etc/postfix/sasl_passwd`:

```
smtp.example.com:587 smtp_user@example.com:smtp_password
```

The SMTP hostname must be the exact same value than *Configuration → Mail → Relay SMTP*.

The file containing the credentials must have specific permissions:

```
chmod 400 /etc/postfix/sasl_passwd
```

Then tell Postfix about this new config file:

```
postmap /etc/postfix/sasl_passwd
```

Then regenerate the Postfix configuration (this does the same thing than *Configuration → Apply system configuration*):

```
xivo-update-config
```

1.7.3 Network

This section describes how to configure additional network devices that may be used to better accommodate more complex network infrastructures. Network interfaces are managed in the Wazo web interface via the page *Configuration → Network → Interfaces*.

Wazo offers 2 types of interfaces: *VoIP* and *Data*. The *VoIP* interface is used by the DHCP server, provisioning server, and phone devices connected to your Wazo. These services will use the information provided on the *VoIP* interface for their configuration. For example, the DHCP server will only listen on the VoIP interface by default.

To change these settings, you must either create a new interface or edit an existing one and change its type. When adding a new *VoIP* interface, the type of the old one will automatically be changed to *Data*.

Configuring a physical interface

In this example, we'll add and configure the **eth1** network interface on our Wazo.

First, we see there's already an unconfigured network interface named **eth1** on our system:

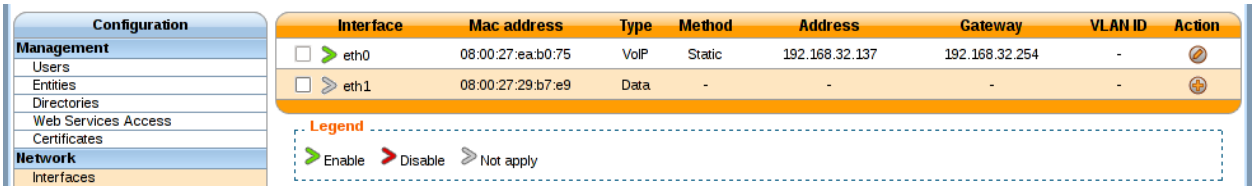


Fig. 1.19: *Configuration → Network → Interfaces*

To add and configure it, we click on the small plus button next to it, and we get to this page:

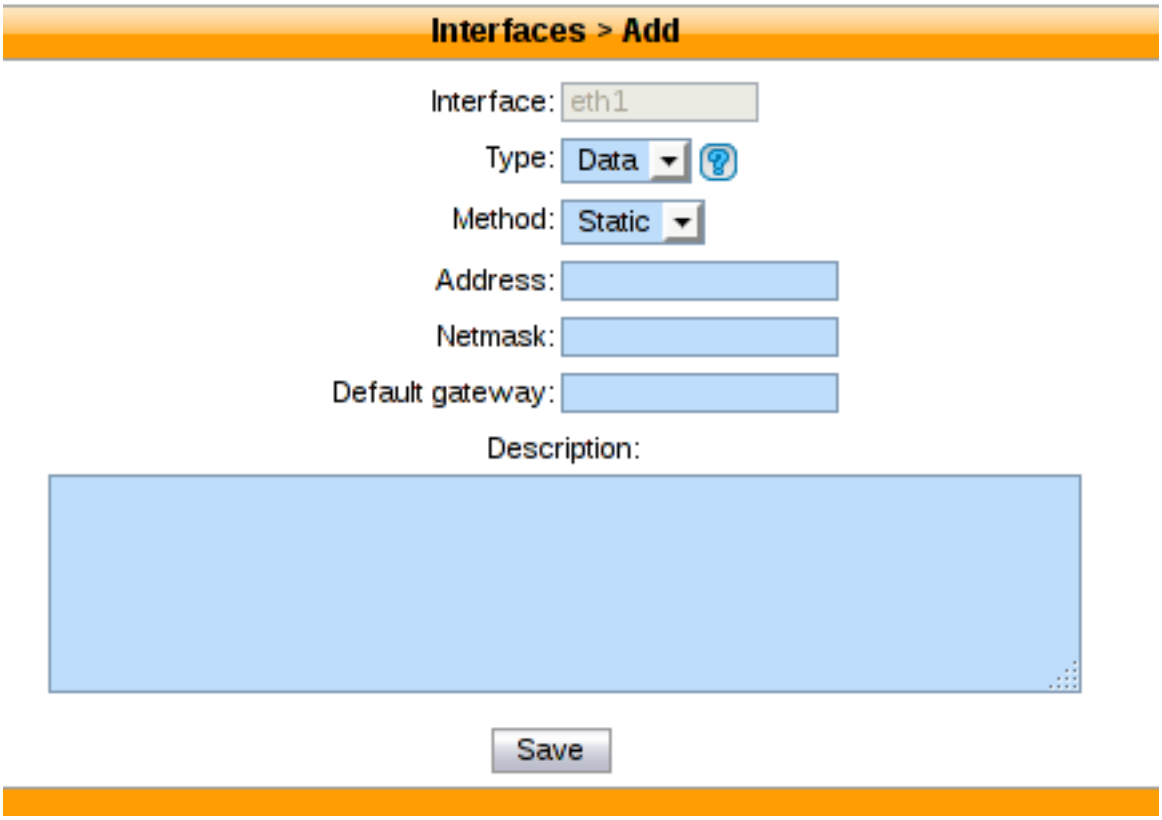


Fig. 1.20: *Configuration → Network → Interfaces → eth1 → Add*

In our case, since we want to configure this interface with static information (i.e. not via DHCP), we fill the following fields:

Note that since our **eth0** network interface already has a default gateway, we do not enter information in the `Default gateway` field for our **eth1** interface.

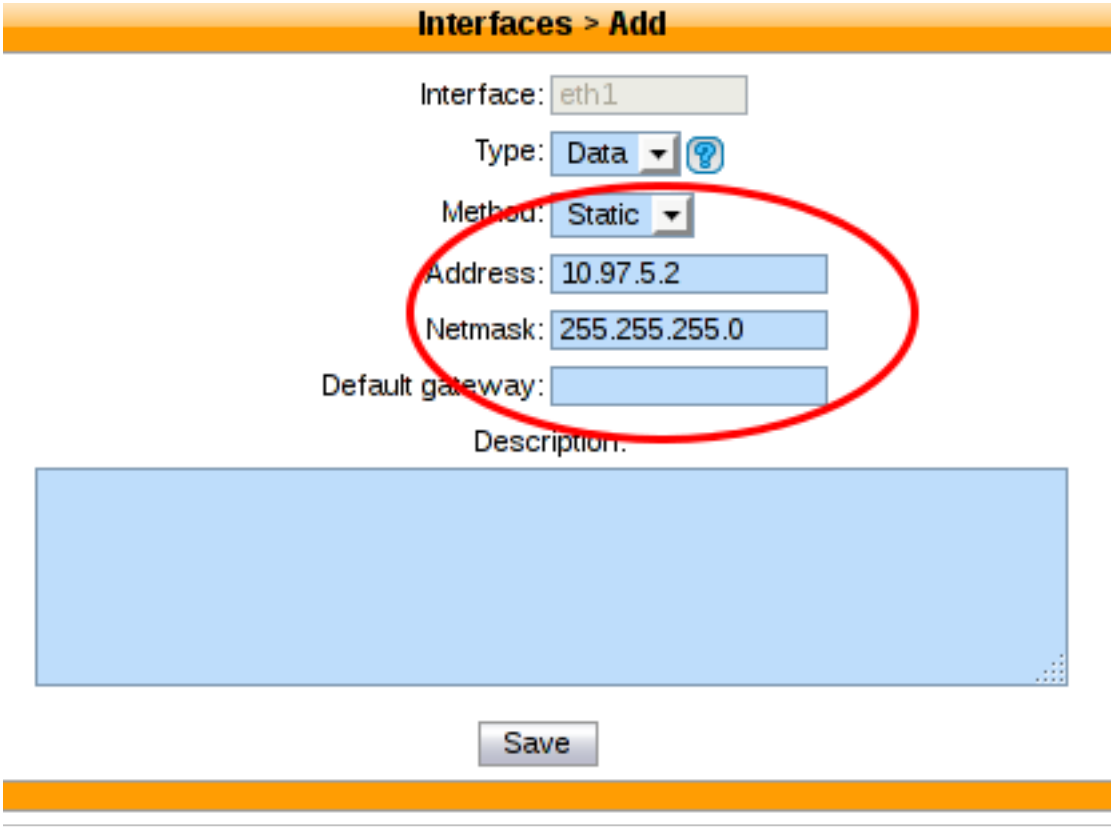


Fig. 1.21: *Configuration → Network → Interfaces → eth1 → Add*

Once the changes have been saved, the action **Apply network configuration** will appear in bold. This action must be clicked in order for the changes to take effect.

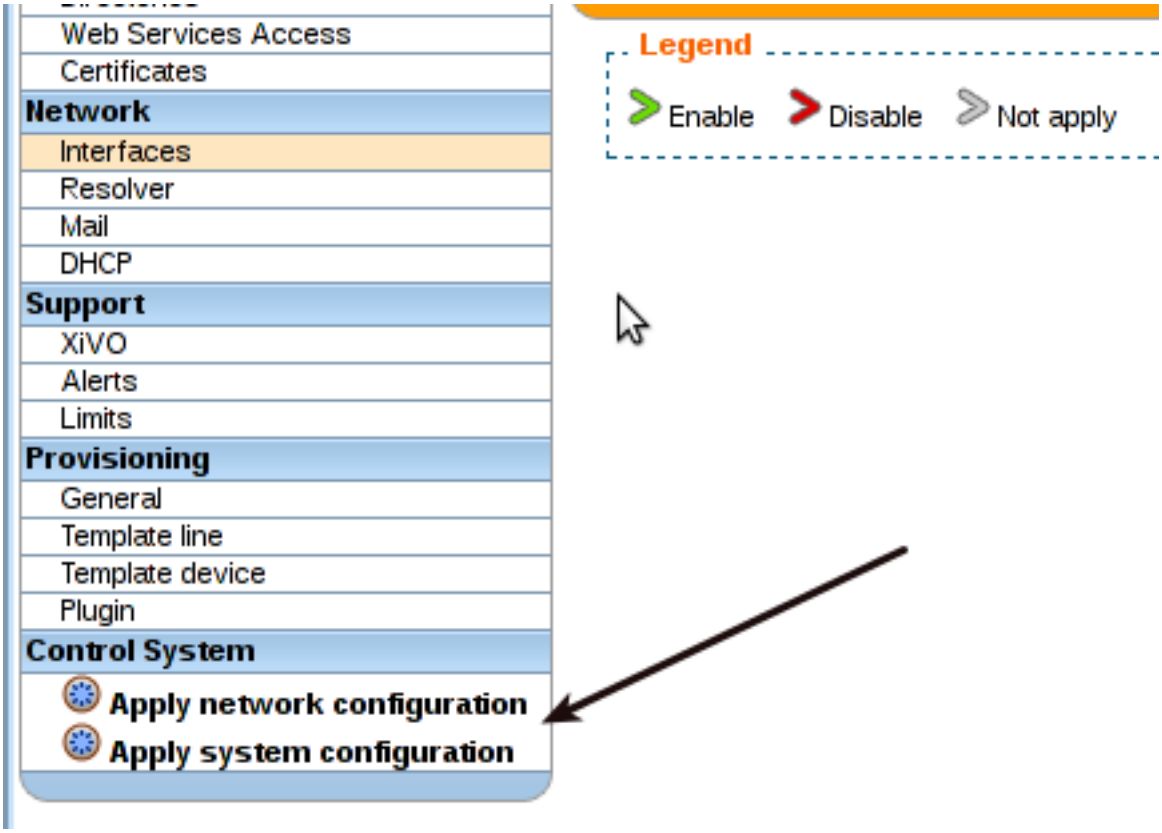


Fig. 1.22: Apply after modify interface

Adding a VLAN interface

In this example, the Wazo already has 2 network interfaces configured:

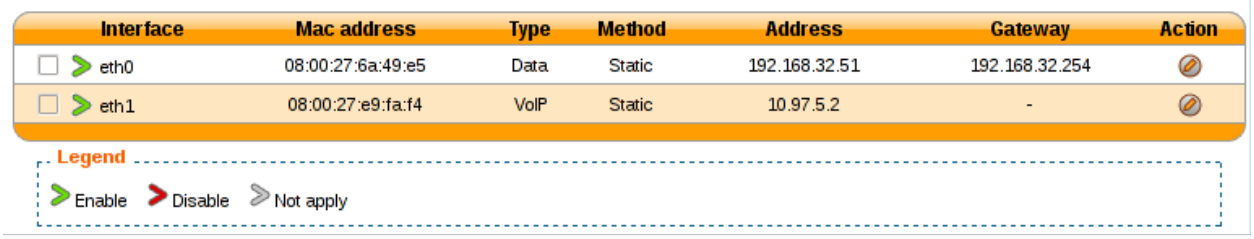


Fig. 1.23: *Configuration → Network → Interfaces*

Listing the network interfaces

To add and configure a new VLAN interface, we click on the small plus button in the top right corner,

and we get to this page:

In our case, since we want to configure this interface with static information:

Click on **Save** list the network interfaces:

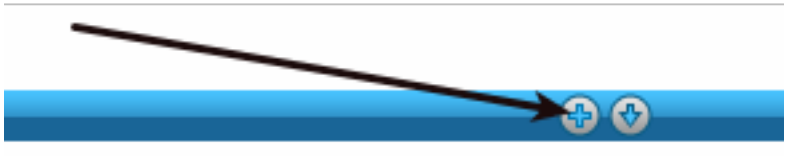


Fig. 1.24: *Configuration → Network → Interfaces → Add button*

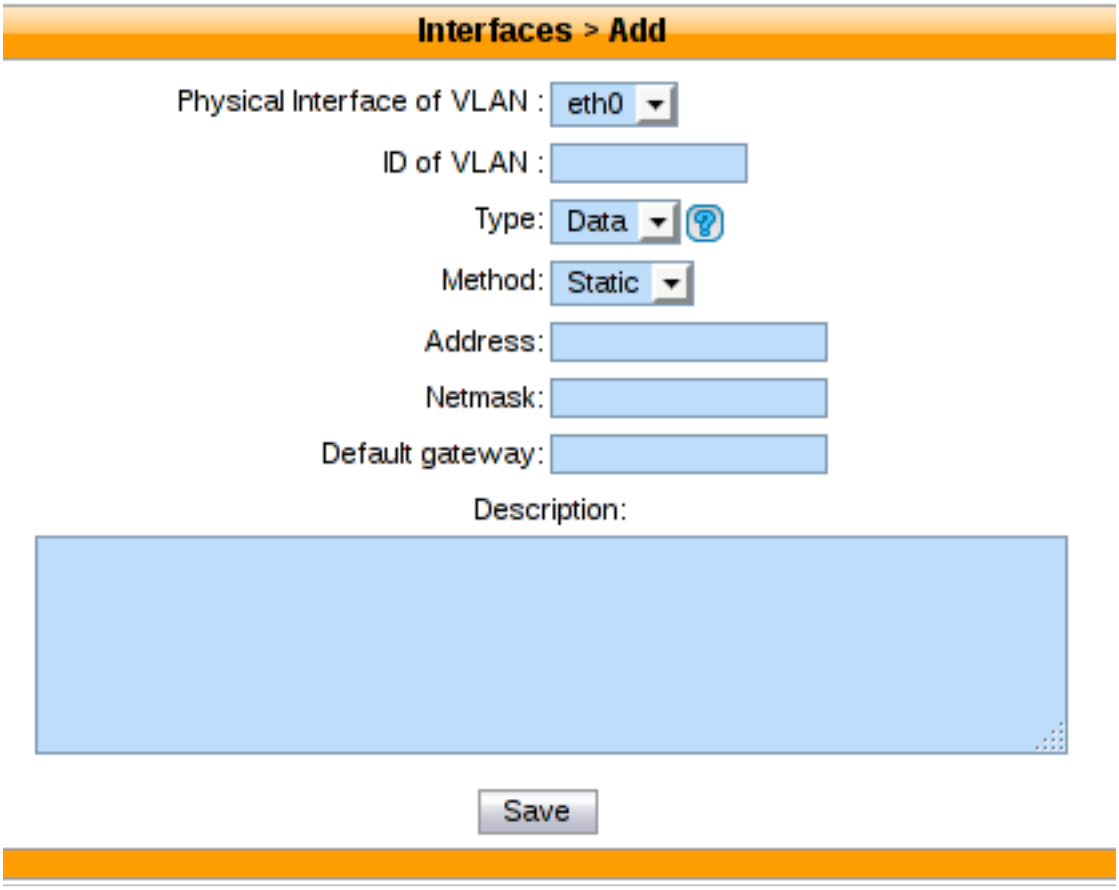


Fig. 1.25: *Configuration → Network → Interfaces → Add*

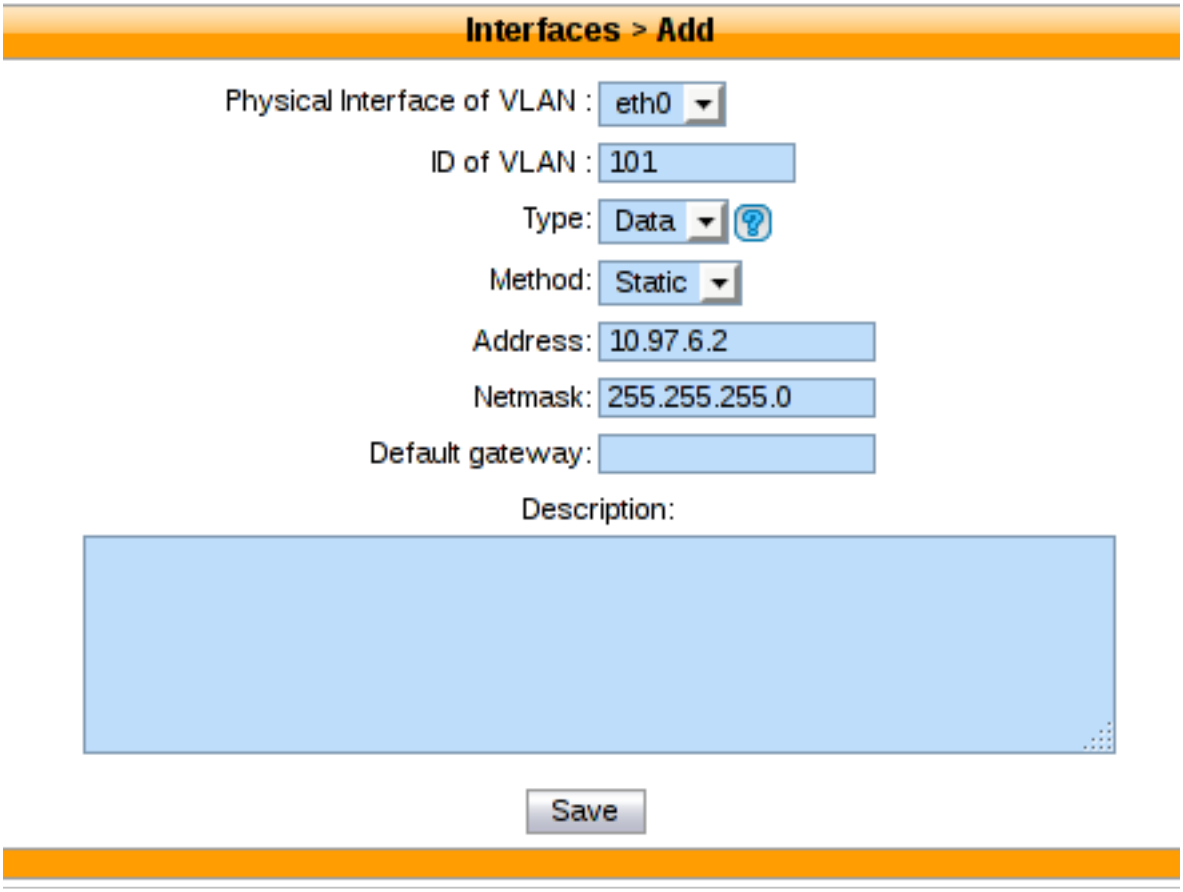


Fig. 1.26: *Configuration → Network → Interfaces → Add*

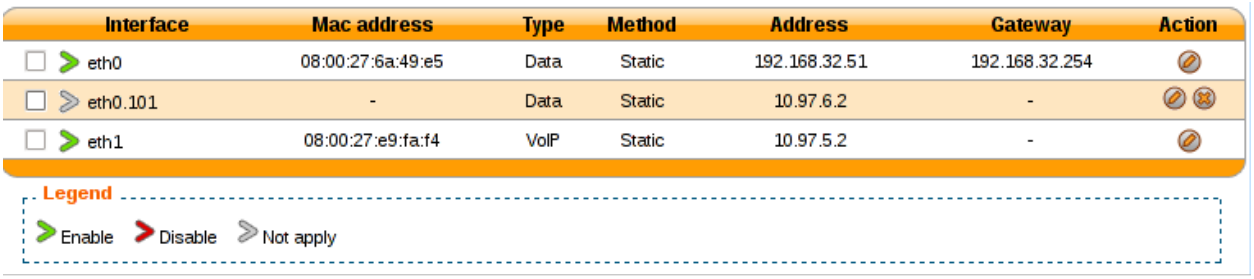


Fig. 1.27: *Configuration → Network → Interfaces*

- The new virtual interface has been successfully created.

Note: Do not forget after you finish the configuration of the network to apply it with the button: **Apply network configuration**

After applying the network configuration:

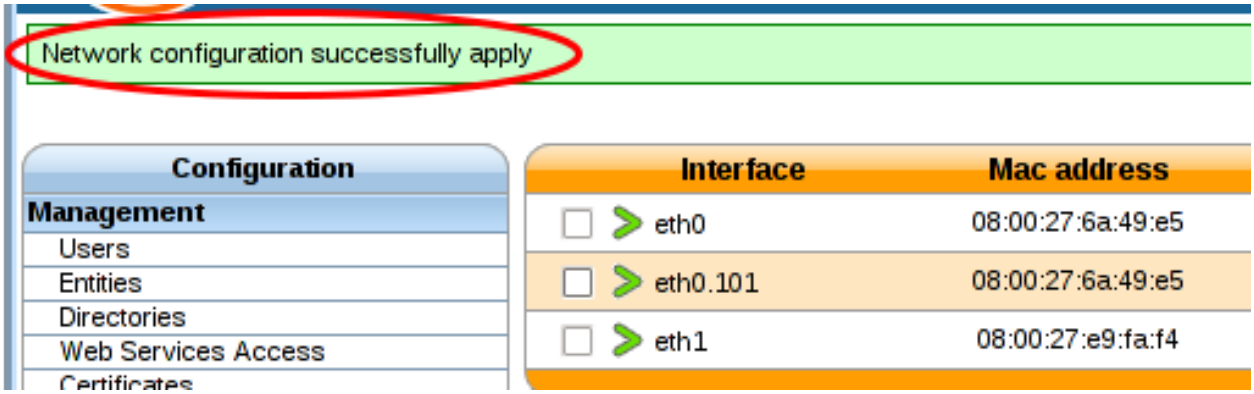


Fig. 1.28: Network configuration successfully apply

Add static network routes

Static routes cannot be added via the web interface. However, you may add static routes to your Wazo by following following the steps described below. This procedure will ensure that your static routes are applied at startup (i.e. each time the network interface goes up).

1. Create the file `/etc/network/if-up.d/xivo-routes`:

```
touch /etc/network/if-up.d/xivo-routes
chmod 755 /etc/network/if-up.d/xivo-routes
```

2. Insert the following content:

```
#!/bin/sh

if [ "${IFACE}" = "<network interface>" ]; then
    ip route add <destination> via <gateway>
    ip route add <destination> via <gateway>
fi
```

3. Fields <network interface>, <destination> and <gateway> should be replaced by your specific configuration. For example, if you want to add a route for 192.168.50.128/25 via 192.168.17.254 which should be added when eth0 goes up:

```
#!/bin/sh

if [ "${IFACE}" = "eth0.2" ]; then
    ip route add 192.168.50.128/25 via 192.168.17.254
fi
```

Note: The above check is to ensure that the route will be applied only if the correct interface goes up. This check

should contain the actual name of the interface (i.e. *eth0* or *eth0.2* or *eth1* or . . .). Otherwise the route won't be set up in every cases.

Change interface MTU

Warning: Manually changing the MTU is risky. Please only proceed if you are aware of what you are doing.

These steps describe how to change the MTU:

```
#. Create the file :file:`/etc/network/if-up.d/xivo-mtu`::
```

touch /etc/network/if-up.d/xivo-mtu chmod 755 /etc/network/if-up.d/xivo-mtu

1. Insert the following content:

```
#!/bin/sh

# Set MTU per iface
if [ "${IFACE}" = "<data interface>" ]; then
    ip link set ${IFACE} mtu <data mtu>
elif [ "${IFACE}" = "<voip interface>" ]; then
    ip link set ${IFACE} mtu <voip mtu>
fi
```

2. Change the <*data interface*> to the name of your interface (e.g. eth0), and the <*data mtu*> to the new MTU (e.g. 1492),
3. Change the <*voip interface*> to the name of your interface (e.g. eth1), and the <*voip mtu*> to the new MTU (e.g. 1488)

Note: In the above example you can set a different MTU per interface. If you don't need a per-interface MTU you can simply write:

```
#!/bin/sh

ip link set ${IFACE} mtu <my mtu>
```

1.7.4 Backup

Periodic backup

A backup of the database and the data are launched every day with a logrotate task. It is run at 06:25 a.m. and backups are kept for 7 days.

Logrotate task:

```
/etc/logrotate.d/xivo-backup
```

Logrotate cron:

```
/etc/cron.daily/logrotate
```

Retrieve the backup

You can retrieve the backup from the web-interface in *Services* → *IPBX* → *IPBX Configuration* → *Backup Files* page.

Otherwise, with shell access, you can retrieve them in `/var/backups/xivo`. In this directory you will find `db.tgz` and `data.tgz` files for the database and data backups.

Backup scripts:

```
/usr/sbin/xivo-backup
```

Backup location:

```
/var/backups/xivo
```

What is actually backed-up?

Data

Here is the list of folders and files that are backed-up:

- `/etc/asterisk/`
- `/etc/consul/`
- `/etc/crontab`
- `/etc/dahdi/`
- `/etc/dhcp/`
- `/etc/hostname`
- `/etc/hosts`
- `/etc/ldap/`
- `/etc/mongooseim/`
- `/etc/network/if-up.d/xivo-routes`
- `/etc/network/interfaces`
- `/etc/ntp.conf`
- `/etc/profile.d/xivo_uuid.sh`
- `/etc/resolv.conf`
- `/etc/ssl/`
- `/etc/systemd/`
- `/etc/wanpipe/`
- `/etc/wazo-admin-ui/`
- `/etc/wazo-auth/`
- `/etc/wazo-call-logd/`
- `/etc/wazo-plugind/`
- `/etc/wazo-webhookd/`
- `/etc/xivo-agentd/`

- /etc/xivo-agid/
- /etc/xivo-amid/
- /etc/xivo-confd/
- /etc/xivo-confgend-client/
- /etc/xivo-ctid/
- /etc/xivo-ctid-ng/
- /etc/xivo-dird/
- /etc/xivo-dird-phoned/
- /etc/xivo-dxtora/
- /etc/xivo-purge-db/
- /etc/xivo-websocketd/
- /etc/xivo/
- /usr/local/bin/
- /usr/local/sbin/
- /usr/share/xivo/XIVO-VERSION
- /var/lib/asterisk/
- /var/lib/consul/
- /var/lib/xivo-provd/
- /var/lib/xivo/
- /var/log/asterisk/
- /var/spool/asterisk/
- /var/spool/cron/crontabs/

The following files/folders are excluded from this backup:

- folders:
  - /var/lib/consul/checks
  - /var/lib/consul/raft
  - /var/lib/consul/serf
  - /var/lib/consul/services
  - /var/lib/xivo-provd/plugins/*/var/cache/*
  - /var/spool/asterisk/monitor/
  - /var/spool/asterisk/meetme/
- files
  - /var/lib/xivo-provd/plugins/xivo-polycom*/var/tftpboot/*.ld
- log files, coredump files
- audio recordings
- and, files greater than 10 MiB or folders containing more than 100 files if they belong to one of these folders:

- /var/lib/xivo/sounds/
- /var/lib/asterisk/sounds/custom/
- /var/lib/asterisk/moh/
- /var/spool/asterisk/voicemail/
- /var/spool/asterisk/monitor/

Database

The following databases from PostgreSQL are backed up:

- asterisk: all the configuration done via the web interface (exceptions: High Availability, Provisioning, Certificates)
- mongooseim: chat history

Creating backup files manually

> **Warning:** A backup file may take a lot of space on the disk. You should check the free space on the partition before creating one.

Database

You can manually create a *database* backup file named db-manual.tgz in /var/tmp by issuing the following commands:

```
xivo-backup db /var/tmp/db-manual
```

Files

You can manually create a *data* backup file named data-manual.tgz in /var/tmp by issuing the following commands:

```
xivo-backup data /var/tmp/data-manual
```

1.7.5 Restore

Introduction

A backup of both the configuration files and the database used by a Wazo installation is done automatically every day. These backups are created in the /var/backups/xivo directory and are kept for 7 days.

Limitations

- You must restore a backup on the **same version** of Wazo that was backed up (though the architecture – i386 or amd64 – may differ)

- You must restore a backup on a machine with the **same hostname and IP address**
- Be aware that this procedure applies **only to XiVO/Wazo >= 14.08** (see *14.08*).

Before Restoring the System

Warning: Before restoring a Wazo on a fresh install you have to setup Wazo using the wizard (see *Running the Wizard* section).

Stop monit and all the Wazo services:

```
wazo-service stop
```

Restoring System Files

System files are stored in the data.tgz file located in the `/var/backups/xivo` directory.

This file contains for example, voicemail files, musics, voice guides, phone sets firmwares, provisioning server configuration database.

To restore the file

```
tar xvfp /var/backups/xivo/data.tgz -C /
```

Once the database and files have been restored, you can *finalize the restore*

Restoring the Database

Warning:

- This will destroy all the current data in your database.
- You have to check the free space on your system partition before extracting the backups.
- If restoring Wazo >= 18.01 on a different machine, you should not restore the system configuration, because of network interface names that would change. See *Alternative: Restoring and Keeping System Configuration*.

Database backups are created as `db.tgz` files in the `/var/backups/xivo` directory. These tarballs contains a dump of the database used in Wazo.

In this example, we'll restore the database from a backup file named `db.tgz` placed in the home directory of root.

First, extract the content of the `db.tgz` file into the `/var/tmp` directory and go inside the newly created directory:

```
tar xvf db.tgz -C /var/tmp
cd /var/tmp/pg-backup
```

Drop the asterisk database and restore it with the one from the backup:

```
sudo -u postgres dropdb asterisk
sudo -u postgres pg_restore -C -d postgres asterisk-*.dump
```

Drop the mongooseim database and restore it with the one from the backup:

```
sudo -u postgres dropdb mongooseim
sudo -u postgres pg_restore -C -d postgres mongooseim-*.dump
```

Once the database and files have been restored, you can *finalize the restore*

Troubleshooting

When restoring the database, if you encounter problems related to the system locale, see *PostgreSQL localization errors*.

Alternative: Restoring and Keeping System Configuration

System configuration like network interfaces is stored in the database. It is possible to keep this configuration and only restore Wazo data.

Rename the asterisk database to asterisk_previous:

```
sudo -u postgres psql -c 'ALTER DATABASE asterisk RENAME TO asterisk_previous'
```

Restore the asterisk database from the backup:

```
sudo -u postgres pg_restore -C -d postgres asterisk-*.dump
```

Restore the system configuration tables from the asterisk_previous database:

```
sudo -u postgres pg_dump -c -t dhcp -t netiface -t resolvconf asterisk_previous |␣
↪sudo -u postgres psql asterisk
```

Drop the asterisk_previous database:

```
sudo -u postgres dropdb asterisk_previous
```

Warning: Restoring the data.tgz file also restores system files such as host hostname, network interfaces, etc. You will need to reapply the network configuration if you restore the data.tgz file.

Drop the mongooseim database and restore it with the one from the backup:

```
sudo -u postgres dropdb mongooseim
sudo -u postgres pg_restore -C -d postgres mongooseim-*.dump
```

Once the database and files have been restored, you can *finalize the restore*

After Restoring The System

1. Resynchronize the xivo-auth keys:

   ```
   xivo-update-keys
   ```

2. Restore the server UUID:

```
XIVO_UUID=$(sudo -u postgres psql -d asterisk -tA -c 'select uuid from infos')
echo "export XIVO_UUID=$XIVO_UUID" > /etc/profile.d/xivo_uuid.sh
```

Then edit `/etc/systemd/system.conf` to update `XIVO_UUID` in `DefaultEnvironment`

3. You may reboot the system, or execute the following steps.
4. Update systemd runtime configuration:

```
source /etc/profile.d/xivo_uuid.sh
systemctl set-environment XIVO_UUID=$XIVO_UUID
systemctl daemon-reload
```

5. Restart the services you stopped in the first step:

```
wazo-service start
```

1.7.6 Certificates for HTTPS

X.509 certificates are used to authorize and secure communications with the server. They are mainly used for HTTPS, but can also be used for SIPS, CTIS, WSS, etc.

There are two categories of certificates in Wazo:

- the default certificate, used for HTTPS in the web interface, REST APIs and WebSockets
- the certificates created and managed via the web interface

This article is about the former. For the latter, see *Telephony certificates*.

Default certificate

Wazo uses HTTPS where possible. The certificates are generated at install time (or during the *upgrade to 15.12+*). The main certificate is placed in `/usr/share/xivo-certs/server.crt`.

However, this certificate is self-signed, and HTTP clients (browser or REST API client) will complain about this default certificate because it is not signed by a trusted Certification Authority (CA).

The default certificate is untrusted

To make the HTTP client accept this certificate, you have two choices:

- configure your HTTP client to trust the self-signed Wazo certificate by adding a new trusted CA. The CA certificate (or bundle) is the file `/usr/share/xivo-certs/server.crt`.
- replace the self-signed certificate with your own trusted certificate.

Use your own certificate

For this, follow these steps:

1. Replace the following files with your own private key/certificate pair:
   - Private key: `/usr/share/xivo-certs/server.key`
   - Certificate chain: `/usr/share/xivo-certs/server.crt`

The certificate chain file `server.crt` must contain all necessary certificates to verify that it is trusted. In particular, if you got your certificate from a provider, `server.crt` must contain the intermediary certificates, allowing the client to follow the trust chain from the CA down to your certificate. There are three possible situations:

- The certificate provider gave you only the certificate file. In this case, you don't have much choice and the certificate file will serve as `server.crt`
- The certificate provider gave you a complete chain file (also called bundle) and you must use this complete chain file as `server.crt`.
- The certificate provider gave you (among others) two different files: a certificate file and an "intermediate" file containing all intermediate certificates. You must get those two files into one with the following command:

```
cat certificate.crt intermediate.pem > full-certificate-chain.pem
```

  Then `full-certificate-chain.pem` must be used for `server.crt`.

Both `server.crt` and `server.key` **must** be readable by the group `www-data`. You can check with the following command:

```
sudo -u www-data cat /usr/share/xivo-certs/server.{key,crt}
```

2. Change the hostname of Wazo for each Wazo component: the different processes of Wazo heavily use HTTPS for internal communication, and for these connection to establish successfully, all hostnames used must match the Common Name (CN) of your certificate. Basically, you must replace all occurrences of `localhost` (the default hostname) with your CN in the *configuration of the Wazo services*. For example:

```
mkdir /etc/xivo/custom
cat > /etc/xivo/custom/custom-certificate.yml << EOF
consul:
  host: wazo.example.com
agentd:
  host: wazo.example.com
ajam:
  host: wazo.example.com
amid:
  host: wazo.example.com
auth:
  host: wazo.example.com
confd:
  host: wazo.example.com
call_logd:
  host: wazo.example.com
ctid_ng:
  host: wazo.example.com
dird:
  host: wazo.example.com
plugind:
  host: wazo.example.com
websocketd:
  host: wazo.example.com
EOF
for config_dir in /etc/{xivo,wazo}-*/conf.d/ ; do
    ln -s "/etc/xivo/custom/custom-certificate.yml" "$config_dir/010-custom-
↪certificate.yml"
done
```

Also, you must replace `localhost` in the definition of your directories in the web interface under *Configuration → Directories*.

3. If your certificate is not self-signed, and you obtained it from a third-party CA that is trusted by your system, you must enable the system-based certificate verification. By default, certificate verification is set to consider `/usr/share/xivo-certs/server.crt` as the only CA certificate.

   The options are the following:

   - Consul: `verify: True`
   - Other Wazo services: `verify_certificate: True`

   The procedure is the same as 2. with more configuration for each service. For example:

   ```
   cat > /etc/xivo/custom/custom-certificate.yml << EOF
   consul:
     host: wazo.example.com
     verify: True
   agentd:
     host: wazo.example.com
     verify_certificate: True
   ajam:
     host: wazo.example.com
     verify_certificate: True
   ...
   ```

   Setting `verify_certificate` to `False` will disable the certificate verification, but the connection will still be encrypted. This is pretty safe as long as Wazo services stay on the same machine, however, this is dangerous when Wazo services are separated by an untrusted network, such as the Internet.

4. You need an entry in `/etc/hosts` resolving your CN to `127.0.0.1`. For this, *do not* edit the file manually, because your modifications will be rewritten when you "Apply system configuration" from the web interface. Instead, create a custom template for `/etc/hosts`, and this template will be used when generating `/etc/hosts`:

   ```
   mkdir -p /etc/xivo/custom-templates/system/etc
   sed 's/127\.0\.1\.1/127.0.0.1/' /usr/share/xivo-config/templates/system/etc/hosts␣
   ↪> /etc/xivo/custom-templates/system/etc/hosts
   xivo-update-config
   ```

   You can check the configuration with the following command, it should give you `127.0.0.1`:

   ```
   getent ahosts wazo.example.com
   ```

5. Restart all Wazo services:

   ```
   wazo-service restart all
   ```

Troubleshooting

Here are a few commands that can help find what is wrong:

```
# Tell me curl, what is the problem with my certificate?
curl https://localhost:443

# Tell me openssl, what is the problem with my certificate?
openssl s_client -connect localhost:443 >/dev/null </dev/null
```

```
# Check that nginx has the right certificate loaded
grep -R ssl /etc/nginx/sites-enabled/

# See the certificate returned by nginx
openssl s_client -connect localhost:443 </dev/null

# See the certificate chain returned by nginx
openssl s_client -connect localhost:443 </dev/null 2>/dev/null | sed -ne '/
↪Certificate chain/,/---/p'
```

Note that you can replace 443 with the ports of the Wazo daemons, e.g. 9497 for wazo-auth. See the full list in *Network*.

1.7.7 Configuration Files

This section describes some of the Wazo configuration files.

Configuration priority

Usually, the configuration is read from two locations: a configuration file `config.yml` and a configuration directory `conf.d`.

Files in the `conf.d` extra configuration directory:

- are used in alphabetical order and the first one has priority
- are ignored when their name starts with a dot
- are ignored when their name does not end with `.yml`

For example:

`.01-critical.yml`:

```
log_level: critical
```

`02-error.yml.dpkg-old`:

```
log_level: error
```

`10-debug.yml`:

```
log_level: debug
```

`20-nodebug.yml`:

```
log_level: info
```

The value that will be used for `log_level` will be `debug` since:

- `10-debug.yml` comes before `20-nodebug.yml` in the alphabetical order.
- `.01-critical.yml` starts with a dot so is ignored
- `02-error.yml.dpkg-old` does not end with `.yml` so is ignored

File configuration structure

Configuration files for every service running on a Wazo server will respect these rules:

- Default configuration directory in `/etc/xivo-{service}/conf.d` (e.g. `/etc/xivo-agentd/conf.d/`)
- Default configuration file in `/etc/xivo-{service}/config.yml` (e.g. `/etc/xivo-agentd/config.yml`)

The files `/etc/xivo-{service}/config.yml` should not be modified because **they will be overridden during upgrades**. However, they may be used as examples for creating additional configuration files as long as they respect the *Configuration priority*. Any exceptions to these rules are documented below.

wazo-auth

- Default configuration directory: `/etc/wazo-auth/conf.d`
- Default configuration file: `/etc/wazo-auth/config.yml`

xivo-agentd

- Default configuration directory: `/etc/xivo-agentd/conf.d`
- Default configuration file: `/etc/xivo-agentd/config.yml`

xivo-amid

- Default configuration directory: `/etc/xivo-amid/conf.d`
- Default configuration file: `/etc/xivo-amid/config.yml`

xivo-confgend

- Default configuration directory: `/etc/xivo-confgend/conf.d`
- Default configuration file: `/etc/xivo-confgend/config.yml`
- Default templates directory: `/etc/xivo-confgend/templates`

xivo-ctid

- Default configuration directory: `/etc/xivo-ctid/conf.d`
- Default configuration file: `/etc/xivo-ctid/config.yml`

xivo-dao

- Default configuration directory: `/etc/xivo-dao/conf.d`
- Default configuration file: `/etc/xivo-dao/config.yml`

This configuration is read by many Wazo programs in order to connect to the Postgres database of Wazo.

xivo-dird-phoned

- Default configuration directory: `/etc/xivo-dird-phoned/conf.d`
- Default configuration file: `/etc/xivo-dird-phoned/config.yml`

xivo-websocketd

- Default configuration directory: `/etc/xivo-websocketd/conf.d`
- Default configuration file: `/etc/xivo-websocketd/config.yml`

xivo_ring.conf

- Path: `/etc/xivo/asterisk/xivo_ring.conf`
- Purpose: This file can be used to change the ringtone played by the phone depending on the origin of the call.

Warning: Note that this feature has not been tested for all phones and all call flows. This page describes how you can customize this file but does not intend to list all validated call flows or phones.

This file `xivo_ring.conf` consists of :

- profiles of configuration (some examples for different brands are already included: `[aastra]`, `[snom]` etc.)
- one section named `[number]` where you apply the profile to an extension or a context etc.

Here is the process you should follow if you want to use/customize this feature :

1. Create a new profile, e.g.:

```
[myprofile-aastra]
```

2. Change the `phonetype` accordingly, in our example:

```
[myprofile-aastra]
phonetype = aastra
```

3. Chose the ringtone for the different type of calls (note that the ringtone names are brand-specific):

```
[myprofile-aastra]
phonetype = aastra
intern = <Bellcore-dr1>
group = <Bellcore-dr2>
```

4. Apply your profile, in the section `[number]`

- to a given list of extensions (e.g. 1001 and 1002):

```
1001@default = myprofile-aastra
1002@default = myprofile-aastra
```

- or to a whole context (e.g. default):

```
@default = myprofile-aastra
```

5. Restart `xivo-agid` service:

```
service xivo-agid restart
```

ipbx.ini

- Path: `/etc/xivo/web-interface/ipbx.ini`
- Purpose: This file specifies various configuration options and paths related to Asterisk and used by the web interface.

Here is a partial glimpse of what can be configured in file `ipbx.ini` :

1. Enable/Disable modification of SIP line username and password:

   ```
   [user]
   readonly-idpwd = "true"
   ```

   When editing a SIP line, the username and password fields cannot be modified via the web interface. Set this option to false to enable the modification of both fields. This option is set to "true" by default.

Warning: This feature is not fully tested. It should be used only when absolutely necessary and with great care.

1.7.8 Consul

The default consul installation in Wazo uses the configuration file in `/etc/consul/xivo/*.json`. All files in this directory are installed with the package and *should not* be modified by the administrator. To use a different configuration, the adminstrator can add it's own configuration file at another location and set the new configuration directory by creating a systemd unit drop-in file in the `/etc/systemd/system/consul.service.d` directory.

The default installation generates a master token that can be retrieved in `/var/lib/consul/master_token`. This master token will not be used if a new configuration is supplied.

Variables

The following environment variables can be overridden in a systemd unit drop-in file:

- `CONFIG_DIR`: the consul configuration directory
- `WAIT_FOR_LEADER`: should the "start" action wait for a leader ?

Example, in `/etc/systemd/system/consul.service.d/custom.conf`:

```
[Service]
Environment=CONFIG_DIR=/etc/consul/agent
Environment=WAIT_FOR_LEADER=no
```

Agent mode

It is possible to run consul on another host and have the local consul node run as an agent only.

To get this kind of setup up and running, you will need to follow the following steps.

Downloading Consul

For a 32 bits system

```
wget --no-check-certificate https://releases.hashicorp.com/consul/0.5.2/consul_0.5.2_
↪linux_386.zip
```

For a 64 bits system

```
wget --no-check-certificate https://releases.hashicorp.com/consul/0.5.2/consul_0.5.2_
↪linux_amd64.zip
```

Installing Consul on a new host

```
unzip consul_0.5.2_linux_386.zip
```

Or

```
unzip consul_0.5.2_linux_amd64.zip
```

```
mv consul /usr/bin/consul
mkdir -p /etc/consul/xivo
mkdir -p /var/lib/consul
adduser --quiet --system --group --no-create-home \
        --home /var/lib/consul consul
```

Copying the consul configuration from the Wazo to a new host

On the new consul host, modify `/etc/consul/xivo/config.json` to include to following lines.

```
"bind_addr": "0.0.0.0",
"client_addr": "0.0.0.0",
"advertise_addr": "<consul-host>"
```

```
# on the consul host
scp root@<wazo-host>:/lib/systemd/system/consul.service /lib/systemd/system
systemctl daemon-reload
scp -r root@<wazo-host>:/etc/consul /etc
scp -r root@<wazo-host>:/usr/share/xivo-certs /usr/share
consul agent -data-dir /var/lib/consul -config-dir /etc/consul/xivo/
```

Note: To start consul with the systemd unit file, you may need to change owner and group (consul:consul) for all files inside `/etc/consul`, `/usr/share/xivo-certs` and `/var/lib/consul`

Adding the agent configuration

Create the file `/etc/consul/agent/config.json` with the following content

```
{
    "acl_datacenter": "<node_name>",
    "datacenter": "xivo",
    "server": false,
    "bind_addr": "0.0.0.0",
    "advertise_addr": "<wazo_address>",
    "client_addr": "127.0.0.1",
    "bootstrap": false,
    "rejoin_after_leave": true,
    "data_dir": "/var/lib/consul",
    "enable_syslog": true,
    "disable_update_check": true,
    "log_level": "INFO",
    "ports": {
        "dns": -1,
        "http": -1,
        "https": 8500
    },
    "retry_join": [
        "<remote_host>"
    ],
    "cert_file": "/usr/share/xivo-certs/server.crt",
    "key_file": "/usr/share/xivo-certs/server.key"
}
```

- `node_name`: Arbitrary name to give this node, `wazo-paris` for example.
- `remote_host`: IP address of your new consul. Be sure the host is accessible from your Wazo and check the firewall. See the documentation *here*.
- `wazo_address`: IP address of your Wazo.

This file should be owned by consul user.

```
chown -R consul:consul /etc/consul/agent
```

Enabling the agent configuration

Add or modify `/etc/systemd/system/consul.service.d/custom.conf` to include the following lines:

```
[Service]
Environment=CONFIG_DIR=/etc/consul/agent
```

Restart your consul server.

```
service consul restart
```

Updating the consul section of xivo-ctid

Add a file in `/etc/xivo-ctid/conf.d/remote_consul.yml` with the following content

```
rest_api:
  http:
    listen: 0.0.0.0
```

```
service_discovery:
  advertise_address: <xivo-ctid-host>
  check_url: http://<xivo-ctid-host>:9495/0.1/infos
```

- `xivo-ctid-host`: Hostname to reach xivo-ctid

1.7.9 Log Files

Every Wazo service has its own log file, placed in `/var/log`.

asterisk

The Asterisk log files are managed by logrotate.

It's configuration files `/etc/logrotate.d/asterisk` and `/etc/asterisk/logger.conf`

The message log level is enabled by default in `logger.conf` and contains notices, warnings and errors. The full log entry is commented in `logger.conf` and should only be enabled when verbose debugging is required. Using this option in production would produce VERY large log files.

- Files location: `/var/log/asterisk/*`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-auth

- File location: `/var/log/wazo-auth.log`
- Rotate configuration: `/etc/logrotate.d/wazo-auth`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-upgrade

- File location: `/var/log/xivo-upgrade.log`
- Rotate configuration: `/etc/logrotate.d/xivo-upgrade`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-agentd

- File location: `/var/log/xivo-agentd.log`
- Rotate configuration: `/etc/logrotate.d/xivo-agentd`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-agid

- File location: `/var/log/xivo-agid.log`
- Rotate configuration: `/etc/logrotate.d/xivo-agid`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-amid

- File location: `/var/log/xivo-amid.log`
- Rotate configuration: `/etc/logrotate.d/xivo-amid`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-call-logd

- File location: `/var/log/wazo-call-logd.log`
- Rotate configuration: `/etc/logrotate.d/wazo-call-logd`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-confd

- File location: `/var/log/xivo-confd.log`
- Rotate configuration: `/etc/logrotate.d/xivo-confd`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-confgend

The xivo-confgend daemon output is sent to the file specified with the `--logfile` parameter when launched with twistd.

The file location can be changed by customizing the xivo-confgend.service unit file.

- File location: `/var/log/xivo-confgend.log`
- Rotate configuration: `/etc/logrotate.d/xivo-confgend`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-ctid

- File location: `/var/log/xivo-ctid.log`
- Rotate configuration: `/etc/logrotate.d/xivo-ctid`
- Number of archived log files: 15
- Rotation frequence: Daily

xivo-ctid-ng

- File location: `/var/log/xivo-ctid-ng.log`
- Rotate configuration: `/etc/logrotate.d/xivo-ctid-ng`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-dird

- File location: `/var/log/xivo-dird.log`
- Rotate configuration: `/etc/logrotate.d/xivo-dird`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-dird-phoned

- File location: `/var/log/xivo-dird-phoned.log`
- Rotate configuration: `/etc/logrotate.d/xivo-dird-phoned`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-dxtora

- File location: `/var/log/xivo-dxtora.log`
- Rotate configuration: `/etc/logrotate.d/xivo-dxtora`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-provd

- File location: `/var/log/xivo-provd.log`
- Rotate configuration: `/etc/logrotate.d/xivo-provd`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-purge-db

- File location: `/var/log/xivo-purge-db.log`
- Rotate configuration: `/etc/logrotate.d/xivo-purge-db`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-stat

- File location: `/var/log/xivo-stat.log`
- Rotate configuration: `/etc/logrotate.d/xivo-stat`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-sysconfd

- File location: `/var/log/xivo-sysconfd.log`
- Rotate configuration: `/etc/logrotate.d/xivo-sysconfd`
- Number of archived files: 15
- Rotation frequence: Daily

xivo-web-interface

- File location: `/var/log/xivo-web-interface/*.log`
- Rotate configuration: `/etc/logrotate.d/xivo-web-interface`
- Number of archived files: 21
- Rotation frequence: Daily

xivo-websocketd

- File location: `/var/log/xivo-websocketd.log`
- Rotate configuration: `/etc/logrotate.d/xivo-websocketd`
- Number of archived files: 15
- Rotation frequence: Daily

1.7.10 Nginx

Wazo use nginx as a web server and reverse proxy.

In its default configuration, the nginx server listens on port TCP/80 and TCP/443 and allows these services to be used:

- The agent management server (xivo-agentd)
- The authentication server (wazo-auth)

- The configuration server (xivo-confd)
- The telephony service interface (xivo-ctid-ng)
- The directory service (xivo-dird)
- The AMI HTTP interface (xivo-amid)
- web interface (xivo-web-interface)
- API documentation (xivo-swagger-doc)
- The websocket interface (xivo-websocketd)
- Asterisk WebSocket (xivo-config)

An administrator can easily modify the configuration to allow or disallow some services.

To do so, an administrator only has to create a symbolic link inside the `/etc/nginx/locations/http-enabled` directory to the corresponding file in the `/etc/nginx/locations/http-available` directory, and then reload nginx with `systemctl reload nginx`. A similar operation must be done for HTTPS.

For example, to enable all the available services:

```
ln -sf /etc/nginx/locations/http-available/* /etc/nginx/locations/http-enabled
ln -sf /etc/nginx/locations/https-available/* /etc/nginx/locations/https-enabled
systemctl reload nginx
```

To disable all the services other than the web interface:

```
rm /etc/nginx/locations/http-enabled/* /etc/nginx/locations/https-enabled/*
ln -s /etc/nginx/locations/http-available/xivo-web-interface /etc/nginx/locations/
↪http-enabled
ln -s /etc/nginx/locations/https-available/xivo-web-interface /etc/nginx/locations/
↪https-enabled
systemctl reload nginx
```

1.7.11 NTP

Wazo has a NTP server, that must be synchronized to a reference server. This can be a public one or customized for specific target networking architecture. Wazo's NTP server is used by default as NTP server for the devices time reference.

Usage

Show NTP service status:

```
service ntp status
```

Stop NTP service:

```
service ntp stop
```

Start NTP service:

```
service ntp start
```

Restart NTP service:

```
service ntp restart
```

Show NTP synchronization status:

```
ntpq -p
```

Configuring NTP service

1. Edit `/etc/ntp.conf`
2. Give your NTP reference servers:

```
server 192.168.0.1                          # LAN existing NTP Server
server 0.debian.pool.ntp.org iburst dynamic  # default in ntp.conf
server 1.debian.pool.ntp.org iburst dynamic  # default in ntp.conf
```

3. If no reference server to synchronize to, add this to synchronize locally:

```
server 127.127.1.0              # local clock (LCL)
fudge 127.127.1.0 stratum 10    # LCL is not very reliable
```

4. Restart NTP service
5. Check NTP synchronization status.

Warning: If #5 shows that NTP doesn't use NTP configuration in `/etc/ntp.conf`, maybe have you done a `dhclient` for one of your network interface and the dhcp server that gave the IP address also gave a NTP server address. Thus you might check if the file `/var/lib/ntp/ntp.conf.dhcp` exists, if yes, this is used for NTP configuration prior to `/etc/ntp.conf`. Remove it and restart NTP, check NTP synchronization status, then it should work.

1.7.12 Proxy Configuration

If you use Wazo behind an HTTP proxy, you must do a couple of manipulations for it to work correctly.

apt

Create the `/etc/apt/apt.conf.d/90proxy` file with the following content:

```
Acquire::http::Proxy "http://domain\username:password@proxyip:proxyport";
```

provd

Proxy information is set via the *Configuration* → *Provisioning* → *General* page.

dhcp-update

This step is needed if you use the DHCP server of the Wazo. Otherwise the DHCP configuration won't be correct.

Proxy information is set via the `/etc/xivo/dhcpd-update.conf` file.

Edit the file and look for the `[proxy]` section.

xivo-fetchfw

This step is not needed if you don't use xivo-fetchfw.

Proxy information is set via the `/etc/xivo/xivo-fetchfw.conf` file.

Edit the file and look for the `[proxy]` section.

1.7.13 Service Discovery

Overview

Wazo uses consul for service discovery. When a daemon is started, it registers itself on the configured consul node.

Consul template may be used to generate the configuration files for each daemons that requires the availability of another service. Consul template can also be used to reload the appropriate service.

1.7.14 Service Authentication

Wazo services expose more and more resources through REST API, but they also ensure that the access is restricted to the authorized programs. For this, we use an *authentication daemon* who delivers authorizations via tokens.

Call flow

Here is the call flow to access a REST resource of a Wazo service:

1. Create a username/password (also called service_id/service_key) with the right *ACLs*, via *Web Services Access*.
2. *Create a token* with these credentials and the backend *xivo-service*.
3. *Use this token* to access the REST resource defined by the *ACL*.

Service Service who needs to access a REST resource.

xivo-{daemon} Server that exposes a REST resource. This resource must have an attached ACL.

wazo-auth Server that authenticates the *Service* and validates the required ACL with the token.

Wazo services directly use this system to communicate with each other, as you can see in their Web Services Access.

1.7.15 wazo-auth

wazo-auth is a scalable, extendable and configurable authentication service. It uses an HTTP interface to emit tokens to users who can then use those tokens to identify and authenticate themselves with other services compatible with wazo-auth.

The HTTP API reference is at http://api.wazo.community.

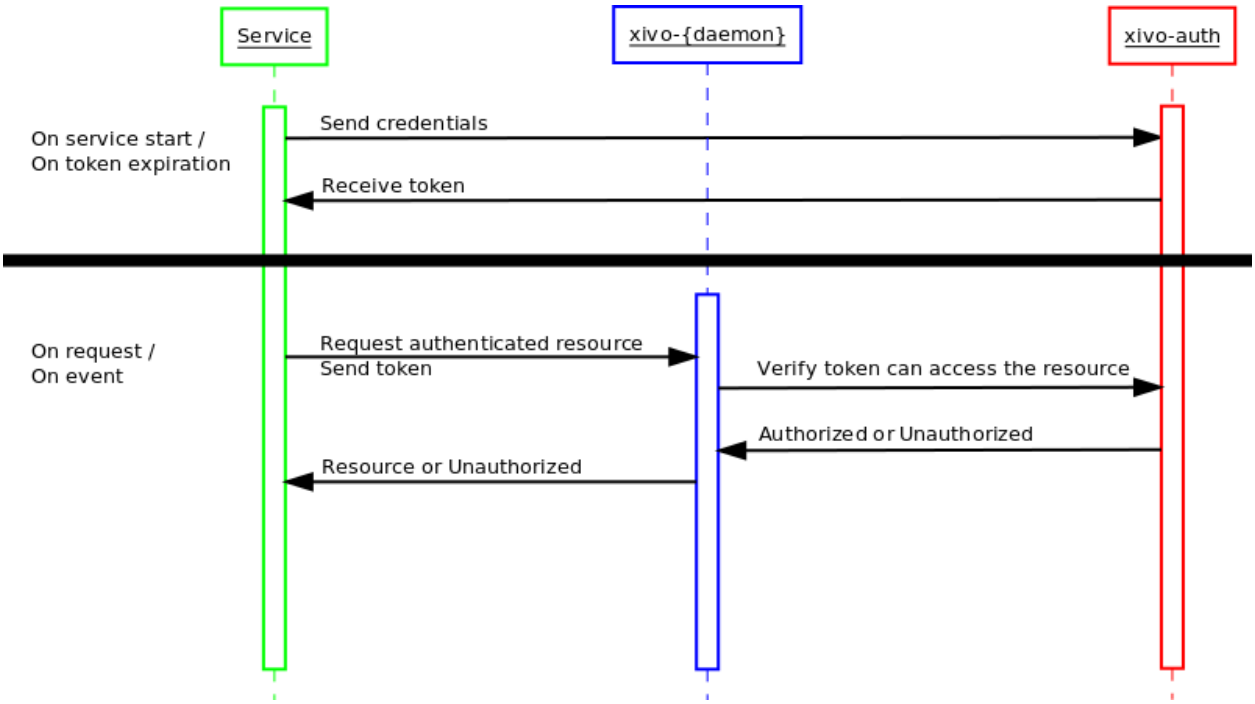


Fig. 1.29: Call flow of service authentication

wazo-auth Developer's Guide

Architecture

wazo-auth contains 3 major components, an HTTP interface, authentication backends and a storage module. All operations are made through the HTTP interface, tokens are stored in postgres as well as the persistence for some of the data attached to tokens. Backends are used to test if a supplied username/password combination is valid and provide the xivo-user-uuid.

wazo-auth is made of the following modules and packages.

backend_plugins

the plugin package contains the wazo-auth backends that are packaged with wazo-auth.

http_plugins

The http module is the implementation of the HTTP interface.

- Validate parameters
- Calls the backend the check the user authentication
- Forward instructions to the *token_manager*
- Handle exceptions and return the appropriate status_code

controller

The controller is the plumbin of wazo-auth, it has no business logic.

- Start the HTTP application
- Load all enabled plugins
- Instanciate the token_manager

token

The token modules contains the business logic of wazo-auth.

- Creates and delete tokens
- Creates ACLs for Wazo
- Schedule token expiration

Plugins

wazo-auth is meant to be easy to extend. This section describes how to add features to wazo-auth.

Backends

wazo-auth allows its administrator to configure one or many sources of authentication. Implementing a new kind of authentication is quite simple.

1. Create a python module implementing the backend interface.
2. Install the python module with an entry point *wazo_auth.backends*

An example backend implementation is available here.

External Auth

wazo-auth allows the user to enable arbitrary external authentication, store sensible information which can be retrieved later given an appropriate ACL.

An external authentication plugin is made of the following parts.

1. A setup.py adding the plugin the the *wazo_auth.http* entry point
2. A flask_restful class implementing the route for this plugin
3. A marshmallow model that can filter the stored data to be safe for unpriviledged view
4. A plugin_info dictionary with information that should be displayed in UI concerning this plugin

The restful class should do the following:

- POST: This is where the plugin should setup any information with the external service and usually return a validation code and a validateion URL to the user.
- GET: After activating the external authentication, following the POST. The GET can be used to retrieve credentials granting access to certain resource of the external service.
- DELETE: Should remove the stored data from wazo-auth

- PUT: (optional) Could be implemented to modify the scope of the generated credentials if the external service allow that kind of modification.

OAuth2 helpers

If the external service uses OAuth2 it is possible to use some helper functions in the external_auth service.

Those helpers can be used to get notified when the user has accepted wazo-auth on the external service.

The following helpers are available:

```
external_auth_service.register_oauth2_callback(auth_type, user_uuid, state, callback,␣
↪*args, **kwargs)
```

- auth_type: The name of the authentication backend
- user_uuid: The user UUID of the user creating the external auth
- state: The state returned from the authorization URL query
- callback: the callable that should be triggered when the authorization is complete
- args and kwargs: arguments that will be added to the callback arguments

When the callback function gets called, its last args will be the message sent to the redirect URL by the external service.

Note: The callback is not executed in the main thread. You should take care of thread synchronization when sharing data structures between threads.

The callback is usually used to create a first token on the external service.

```
external_auth_service.build_oauth2_redirect_url(auth_type)
```

This helper returns a URL that can be used by the OAuth2Session to trigger a redirection and receives a callback when the authorization is complete.

Example

Files:

```
setup.py
src/plugin.py
```

Listing 1.7: setup.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from setuptools import find_packages
from setuptools import setup

setup(
    name='auth_bar',
    version='0.1',

```

```
    packages=find_packages(),
    entry_points={
        'wazo_auth.external_auth': [
            'bar = src.plugin:BarPlugin',
        ],
    }
)
```

Listing 1.8: src/plugin.py

```
# -*- coding: utf-8 -*-

from marshmallow import Schema, fields, pre_load
from flask import request
from wazo_auth import http


class BarService(http.AuthResource):

    auth_type = 'bar'  # Should be the same as the entry point
    authorization_base_url = 'https://accounts.bar.com/oauth/v2/auth'
    token_url = 'https://accounts.bar.com/oauth/v2/token'
    client_id = 'client_id'
    client_secret = 'client_secret'

    def __init__(self, external_auth_service):
        self.external_auth_service = external_auth_service
        self.redirect_uri = self.external_auth_service.build_oauth2_redirect_url(self.
↪auth_type)

    @http.required_acl('auth.users.{user_uuid}.external.bar.delete')
    def delete(self, user_uuid):
        # Remove all stored data for the BAR service for this user
        self.external_auth_service.delete(user_uuid, self.auth_type)
        return '', 204

    @http.required_acl('auth.users.{user_uuid}.external.bar.read')
    def get(self, user_uuid):
        # The GET retrieves all stored data from the service and return the secret␣
↪that is
        # required to use the Bar service

        # The GET will also need to generate a new token if the current one has␣
↪expired.
        return self.external_auth_service.get(user_uuid, self.auth_type), 200

    @http.required_acl('auth.users.{user_uuid}.external.bar.create')
    def post(self, user_uuid):
        session = OAuth2Session(self.client_id, scope=self.scope, redirect_uri=self.
↪redirect_uri)
        # Should use the body of the POST and create a token with the Bar service
        data = request.get_json(force=True)
        authorization_url, state = session.authorization_url(
            self.authorization_base_url,
            access_type='offline',
        )
        self.external_auth_service.register_oauth2_callback(
            state,
```

```
            self.create_first_token,
            session,
            user_uuid,
        )

        return {'authorization_url': authorization_url}, 201

    def create_first_token(self, session, user_uuid, msg):
        # This callback is triggered when the user authorize wazo-auth using the authorization_url
        token_data = session.fetch_token(
            self.token_url,
            client_secret=self.client_secret['us'],
            code=msg['code'],
        )

        data = {
            'access_token': token_data['access_token'],
            'refresh_token': token_data.get('refresh_token'),
            'token_expiration': get_timestamp_expiration(token_data['expires_in'])
        }

        self.external_auth_service.update(user_uuid, self.auth_type, data)


# When GET /users/:uuid/external is called this model will be used to filter the private data
class BarSafeData(Schema):

    # Only the scope field will be returned
    scope = fields.List(fields.String)

    @pre_load
    def ensure_dict(self, data):
        return data or {}


class BarPlugin(object):

    plugin_info = {'required_acl': ['view-all-contacts', 'list-email-addresses']}

    def load(self, dependencies):
        api = dependencies['api']
        external_auth_service = dependencies['external_auth_service']
        args = (external_auth_service,)

        # If the plugin does not register a safe mode an empty dictionary will be used when doing
        # a GET /users/:uuid/external
        external_auth_service.register_safe_auth_model('bar', BarSafeData)

        api.add_resource(BarService, '/users/<uuid:user_uuid>/external/bar', resource_class_args=args)
```

Stock Plugins Documentation

Backends Plugins

wazo_user

Backend name: `wazo_user`

Purpose: Authenticate a user created by wazo-auth. These users do not map to telephony users at the moment.

Supported policy variables

- username: The username of the user
- groups: A list of groups associated to this user
  - group.uuid: The group UUID
  - group.name: The group name
  - group.users: A list of users associated to this group each user having the following fields
    - username
    - uuid
- tenants: A list of tenants associated to this user
  - tenant.uuid: The tenant UUID
  - tenant.name: The tenant name

xivo_admin

Backend name: `xivo_admin`

Purpose: Authenticate a Wazo administrator. The login/password is configured in *Configuration* → *Management* → *Users*.

Supported policy variables

- entity: The entity of the administrator

Note: The *entity* variable can be *None* which usually mean that this administrator has access to all entities.

---

xivo_service

Backend name: `xivo_service`

Purpose: Authenticate a Wazo *Web Services Access*. The login/password is configured in *Configuration* → *Management* → *Web Service Access*.

xivo_user

Backend name: `xivo_user`

Purpose: Authenticate a Wazo user. The login/password is configured in *IPBX* → *Services* → *PBX Settings* → *Users* in the CTI client section.

Supported policy variables

- id: The ID of the user authenticating
- uuid: The UUID of the user authenticating
- voicemails: a list of voicemail ID associated to this user
- lines: a list of line ID associated to this user
- extensions: a list of extension ID associated to this user
- endpoint_sip: a list of SIP endpoint ID associated to this user
- endpoing_sccp: a list of SCCP endpoint ID associated to this user
- endpoint_custom: a list of custum endpoint ID associated to this user
- agent: a dictionnary containing the agent's property, may be none and should be tested with an if before accessing its fields
- agent.id: an agent id if the user is an agent
- agent.number: an agent number if the user is an agent

LDAP

Backend name: `ldap_user`

Purpose: Authenticate with an LDAP user.

For example, with the given configuration:

```
enabled_backend_plugins:
    ldap_user: true
ldap:
    uri: ldap://example.org
    bind_dn: cn=wazo,dc=example,dc=org
    bind_password: bindpass
    user_base_dn: ou=people,dc=example,dc=org
    user_login_attribute: uid
    user_email_attribute: mail
```

When an authentication request is received for username `alice` and password `userpass`, the backend will:

1. Connect to the LDAP server at example.org
2. Do an LDAP "bind" operation with bind DN `cn=wazo,dc=example,dc=org` and password `bindpass`
3. Do an LDAP "search" operation to find an LDAP user matching `alice`, using:
    - the base DN `ou=people,dc=example,dc=org`
    - the filter `(uid=alice)`

    - a SUBTREE scope
4. If the search returns exactly 1 LDAP user, do an LDAP "bind" operation with the user's DN and the password `userpass`
5. If the LDAP "bind" operation is successful, search in Wazo a user with an email matching the `mail` attribute of the LDAP user
6. If a Wazo user is found, success

To use an anonymous bind instead, the following configuration would be used:

```
ldap:
    uri: ldap://example.org
    bind_anonymous: True
    user_base_dn: ou=people,dc=example,dc=org
    user_login_attribute: uid
    user_email_attribute: mail
```

The backend can also works in a "no search" mode, for example with the following configuration:

```
ldap:
    uri: ldap://example.org
    user_base_dn: ou=people,dc=example,dc=org
    user_login_attribute: uid
    user_email_attribute: mail
```

When the server receives the same authentication request as above, it will directly do an LDAP "bind" operation with the DN `uid=alice,ou=people,dc=example,dc=org` and password `userpass`, and continue at step 5.

Note: User's email and voicemail's email are two separate things. This plugin only use the user's email.

Configuration

`uri` the URI of the LDAP server. Can only contain the scheme, host and port of an LDAP URL.

`user_base_dn` the base dn of the user

`user_login_attribute` the attribute to login a user

`user_email_attribute` (optional) the attribute to match with the Wazo user's email (default: mail)

`bind_dn` (optional) the bind DN for searching for the user DN.

`bind_password` (optional) the bind password for searching for the user DN.

`bind_anonymous` (optional) use anonymous bind for searching for the user DN (default: false)

Supported policy variables

- id: The ID of the user authenticating
- uuid: The UUID of the user authenticating
- voicemails: a list of voicemail ID associated to this user
- lines: a list of line ID associated to this user
- extensions: a list of extension ID associated to this user

- endpoint_sip: a list of SIP endpoint ID associated to this user
- endpoing_sccp: a list of SCCP endpoint ID associated to this user
- endpoint_custom: a list of custum endpoint ID associated to this user
- agent: a dictionnary containing the agent's property, may be none and should be tested with an if before accessing its fields
- agent.id: an agent id if the user is an agent
- agent.number: an agent number if the user is an agent

Usage

wazo-auth is used through HTTP requests, using HTTPS. Its default port is 9497. As a user, the most common operation is to get a new token. This is done with the POST method.

Alice retrieves a token using her username/password:

```
$ # Alice creates a new token, using the xivo_user backend, expiring in 10 minutes
$ curl -k -X POST -H 'Content-Type: application/json' -u 'alice:s3cre7' "https://
↪localhost:9497/0.1/token" -d '{"backend": "xivo_user", "expiration": 600}';echo
{"data": {"issued_at": "2015-06-05T10:16:58.557553", "utc_issued_at": "2015-06-
↪05T15:16:58.557553", "token": "1823c1ee-6c6a-0cdc-d869-964a7f08a744", "auth_id":
↪"63f3dc3c-865d-419e-bec2-e18c4b118224", "xivo_user_uuid": "63f3dc3c-865d-419e-bec2-
↪e18c4b118224", "expires_at": "2015-06-05T11:16:58.557595", "utc_expires_at": "2015-
↪06-05T16:16:58.557595"}}
```

In this example Alice used here Wazo CTI client login `alice` and password `s3cre7`. The authentication source is determined by the *backend* in the POST data.

Alice could also have specified an expiration time on her POST request. The expiration value is the number of seconds before the token expires.

After retrieving her token, Alice can query other services that use wazo-auth and send her token to those service. Those services can then use this token on Alice's behalf to access her personal storage.

If Alice wants to revoke her token before its expiration:

```
$ curl -k -X DELETE -H 'Content-Type: application/json' "https://localhost:9497/0.1/
↪token/1823c1ee-6c6a-0cdc-d869-964a7f08a744"
```

See http://api.wazo.community for more details about the HTTP API.

See *Service Authentication* for details about the authentication process.

Usage for services using wazo-auth

A service that requires authentication and identification can use wazo-auth to externalise the burden of authentication. The new service can then accept a token as part of its operations to authenticate the user using the service.

Once a service receives a token from one of its user, it will need to check the validity of that token. There are 2 forms of verification, one that only checks if the token is valid and the other returns information about this token's session if it is valid.

Checking if a token is valid:

```
$ curl -k -i -X HEAD -H 'Content-Type: application/json' "https://localhost:9497/0.1/
↪token/1823c1ee-6c6a-0cdc-d869-964a7f08a744"
HTTP/1.1 204 NO CONTENT
Content-Type: text/html; charset=utf-8
Content-Length: 0
Date: Fri, 05 Jun 2015 14:49:50 GMT
Server: pcm-dev-0

$ # get more information about this token
$ curl -k -X GET -H 'Content-Type: application/json' "https://localhost:9497/0.1/
↪token/1823c1ee-6c6a-0cdc-d869-964a7f08a744";echo
{"data": {"issued_at": "2015-06-05T10:16:58.557553", "utc_issued_at": "2015-06-
↪05T15:16:58.557553", "token": "1823c1ee-6c6a-0cdc-d869-964a7f08a744", "auth_id":
↪"63f3dc3c-865d-419e-bec2-e18c4b118224", "xivo_user_uuid": "63f3dc3c-865d-419e-bec2-
↪e18c4b118224", "expires_at": "2015-06-05T11:16:58.557595", "utc_expires_at": "2015-
↪06-05T16:16:58.557595"}}
```

Launching wazo-auth

```
usage: wazo-auth [-h] [-c CONFIG_FILE] [-u USER] [-d] [-f] [-l LOG_LEVEL]

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG_FILE, --config-file CONFIG_FILE
                        The path to the config file
  -u USER, --user USER  User to run the daemon
  -d, --debug           Log debug messages
  -f, --foreground      Foreground, don't daemonize
  -l LOG_LEVEL, --log-level LOG_LEVEL
                        Logs messages with LOG_LEVEL details. Must be one of:
                        critical, error, warning, info, debug. Default: None
```

Configuration

Policies

Policies can be assigned to backends in order to generate the appropriate permissions for a token created with this backend.

To change to policy associated to a backend, add a new configuration file in `/etc/wazo-auth/conf.d` with the following content:

```
backend_policies:
  <backend_name>: <policy_name>
```

- backend_name: The name of the backend to associate to a new policy
- policy_name: The name of the policy to assign to the backend

Note: Each backend may support different variables. A policy tailored for a user oriented backend will probably not be usable if assigned to an administrator backend.

Policies

A policy is a list of ACL templates that is used to generate the ACL of a token. Policies can be created, deleted or modified using the REST API.

ACL templates

ACL templates use jinja2 templates. Each backend is responsible of supplying a list of variables to the template engine for rendering.

A backend supplying the following variables:

```
{"uuid": "fd64193f-7260-4299-9bc2-87c0106e5302",
 "lines": [1, 42],
 "agent": {"id": 50, "number": "1001"}}
```

With the following ACL templates:

```
confd.users.{{ uuid }}.read
{% for line in lines %}confd.lines.{{ line }}.#\n{% endfor %}
dird.me.#
{% if agent %}agentd.agents.by-id.{{ agent.id }}.read{% endif %}
```

Would create tokens with the following ACL:

```
confd.users.fd64193f-7260-4299-9bc2-87c0106e5302.read
confd.lines.1.#
confd.lines.42.#
dird.me.#
agentd.agentd.by-id.50.read
```

HTTP API Reference

The complete HTTP API documentation is at http://api.wazo.community.

See also the *wazo-auth changelog*.

Development

See *wazo-auth Developer's Guide*.

1.7.16 wazo-service

Wazo has many running services. To restart the whole stack, the `wazo-service` command can be used to make sure the service is restarted in the right order.

Usage

Show all services status:

```
wazo-service status
```

Stop XiVO services:

```
wazo-service stop
```

Start XiVO services:

```
wazo-service start
```

Restart XiVO services:

```
wazo-service restart
```

The commands above will only act upon Wazo services. Appending an argument `all` will also act upon `nginx` and `postgresql`. Example:

```
wazo-service restart all
```

UDP port 5060 will be closed while services are restarting.

1.7.17 wazo-webhookd

wazo-webhookd is the microservice responsible for webhooks: it manages the list of webhooks and triggers them when an event occurs.

How to add a new webhookd type (a.k.a service)

Here is an example of a webhook type that does nothing. Actually, it is very busy and sleeps for N seconds `:)` You may of course change this behaviour for something more suited to your needs.

Files:

```
setup.py
example_service/plugin.py
```

`setup.py`:

```
from setuptools import setup
from setuptools import find_packages

setup(
    name='wazo-webhookd-service-example',
    version='1.0',
    packages=find_packages(),
    entry_points={
    'wazo_webhookd.services': [
            # * "example" is the name of the service.
            #   It will be used when creating a subscription.
            # * "example_service" is the name of the directory above,
            #   the one that contains plugin.py
            # * "plugin" is the name of the above file "plugin.py"
            # * "Service" is the name of the class shown below
            'example = example_service.plugin:Service',
        ]
    }
)
```

example_service/plugin.py:

```
import time

class Service:

    def load(self, dependencies):
        celery_app = dependencies['celery']

        @celery_app.task
        def example_callback(subscription, event):
            '''
            * "subscription" is the subscription dict, same as the one returned by the REST API.
              The service-specific options are available in the "config" key, e.g. for http: the
              url is in subscription['config']['url'].
            * "event" contains the Wazo event that triggered the webhook.
              "event" is of the form:
              {
                  "name": "user_created",
                  "origin_uuid": "the UUID of the Wazo server that sent the event",
                  "data": {
                    "id": 12,  # the ID of the user that was created
                  }
              }
            '''
            tired = subscription['config']['sleep_time']
            time.sleep(tired)

        self._callback = example_callback

    def callback(self):
        return self._callback
```

To install this Python plugin, run:

```
python setup.py install
```

Once installed, you may create subscriptions with the type `example`:

```
POST /subscriptions
{
  "name": "Example webhook",
  "service": "example",
  "config": {
    "time_sleep": 10
  },
  "events": ["user_created"],
}
```

How to trigger code on a bus event

example_service/plugin.py:

```
class Service:

```

```
    def load(self, dependencies):
        ...
        bus_consumer = dependencies['bus_consumer']
        bus_consumer.subscribe_to_event_names(uuid=uuid.uuid4(),
                                              event_names=['user_created'],
                                              user_uuid=None,
                                              wazo_uuid=None,
                                              callback=self.on_user_created)

    def on_user_created(self, body, event):
        logger.debug('User %s has been created!', body['uuid'])
```

How to programmatically create a subscription

example_service/plugin.py:

```
from wazo_webhookd.plugins.subscription.service import SubscriptionService

class Service:

    def load(self, dependencies):
        ...
        subscription_service = SubscriptionService(dependencies['config'])
        ...
        subscription = subscription_service.create({
            'name': 'my-subscription',
            'service': 'http',
            'events': ['call_created'],
            'config': {
                'method': 'get',
                'url': 'https://me.example.com',
            },
        })
```

1.7.18 xivo-confd

xivo-confd is a HTTP server that provides a RESTful API service for configuring and managing basic resources on a Wazo server.

The HTTP API reference is available at http://api.wazo.community.

Developer's Guide (xivo-confd)

xivo-confd resources are organised through a plugin mechanism. There are 2 main plugin categories:

Resource plugins A plugin that manages a resource (e.g. users, extensions, voicemails, etc). A resource plugin exposes the 4 basic CRUD operations (Create, Read, Update, Delete) in order to operate on a resource in a RESTful manner.

Association plugins A plugin for associating or dissociating 2 resources (e.g a user and a line). An association plugin exposes an HTTP action for associating (either `POST` or `PUT`) and another for dissociating (`DELETE`)

The following diagram outlines the most important parts of a plugin:

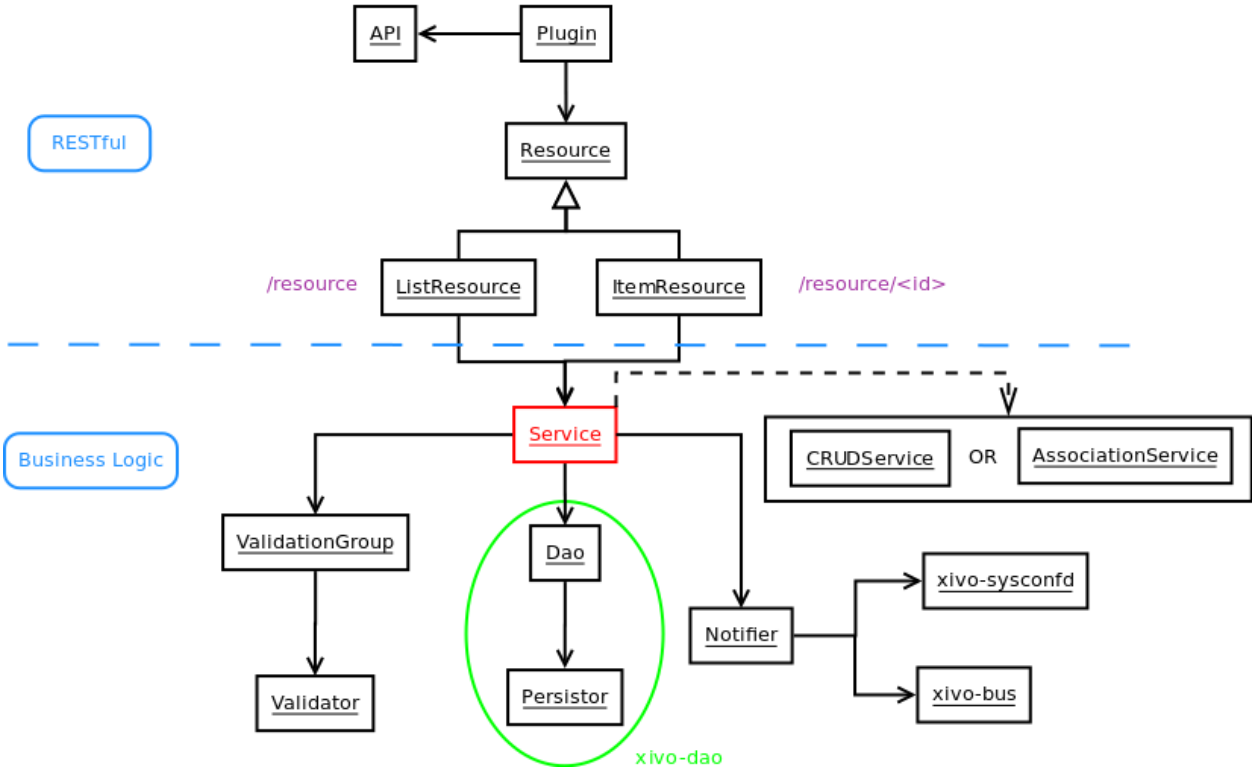


Fig. 1.30: Plugin architecture of xivo-confd

Resource Class that receives and handles HTTP requests. Resources use flask-restful for handling requests.

There are 2 kinds of resources: *ListResource* for root URLs and *ItemResource* for URLs that have an ID. *ListResource* will handle creating a resource (`POST`) and searching through a list of available resources (`GET`). *ItemResource* handles fetching a single item (`GET`), updating (`PUT`) and deleting (`DELETE`).

Service Class that handles business logic for a resource, such as what to do in order to get, create, update, or delete a resource. *Service* classes do not manipulate data directly. Instead, they coordinate what to do via other objects.

There are 2 kinds of services: *CRUDService* for basic CRUD operations in *Resource plugins*, and *AssociationService* for association/dissociation operations in *Association plugins*.

Dao Data Access Object. Knows how to get data and how to manipulate it, such as SQL queries, files, etc.

Notifier Sends events after an operation has completed. An event will be sent in a messaging queue for each CRUD operation. Certain resources also need to send events to other daemons in order to reload some configuration data. (i.e. asterisk needs to reload the dialplan when an extension is updated)

Validator Makes sure that a resource's data does not contain any errors before doing something with it. A *Validator* can be used for validating input data or business rules.

1.7.19 xivo-confgend

xivo-confgend is a configuration file generator. It is mainly used to generate the Asterisk configuration files.

XiVO confgend developer's guide

xivo-confgend uses drivers to implement the logic required to generate configuration files. It uses stevedore to do the driver instantiation and discovery.

Plugins in xivo-confgend use setuptools' entry points. That means that installing a new plugin to xivo-confgend requires an entry point in the plugin's *setup.py*.

Drivers

Driver plugin are classes that are used to generate the content of a configuration file.

The implementation of a plugin should have the following properties.

1. It's `__init__` method should take one argument
2. It should have a `generate` method which will return the content of the file
3. A setup.py adding an entry point

The `__init__` method argument is the content of the configuration of xivo-confgend. This allows the driver implementor to add values to the configuration in `/etc/xivo-confgend/conf.d/*.yml` and these values will be available in the driver.

The generate method has no argument, the configuration provided to the `__init__` should be sufficient for most cases. `generate` is called within a `scoped_session` of xivo-dao, allowing the usage of xivo-dao without prior setup in the driver.

The namespaces used for entry points in xivo-confgend have the following form:

    xivo_confgend.<resource>.<filename>

as an example, a generator for sip.conf would have the following namespace:

    xivo_confgend.asterisk.sip.conf

Example

Here is a typical `setup.py`:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright 2016 The Wazo Authors  (see the AUTHORS file)
# SPDX-License-Identifier: GPL-3.0+

from setuptools import setup
from setuptools import find_packages


setup(
    name='Wazo confgend driversample',
    version='0.0.1',

    description='An example driver',

    packages=find_packages(),

    entry_points={
        'xivo_confgend.asterisk.sip.conf': [
```

```
            'my_driver = src.driver:MyDriver',
        ],
    }
)
```

With the following package structure:

```
.
├── setup.py
└── src
    └── driver.py
```

`driver.py`:

```
# -*- coding: utf-8 -*-
# Copyright 2016 The Wazo Authors  (see the AUTHORS file)
# SPDX-License-Identifier: GPL-3.0+


class MyDriver(object):

    def __init__(self, config):
        self._config = config

    def generate(self):
        return 'Hello World!'
```

To enable this plugin, you need to:

1. Install the plugin with:

   ```
   python setup.py install
   ```

2. Create a config file in `/etc/xivo-confgend/conf.d`:

   ```
   plugins:
     asterisk.sip.conf: my_driver
   ```

3. Restart xivo-confgend:

   ```
   systemctl restart xivo-confgend
   ```

1.7.20 xivo-dird

xivo-dird is the directory server for Wazo. It offers a simple REST interface to query all directories that are configured. xivo-dird is extendable with plugins.

xivo-dird configuration

There are three sources of configuration for xivo-dird:

- the *command line options*
- the main configuration file
- the sources configuration directory

The command-line options have priority over the main configuration file options.

Main Configuration File

Default location: `/etc/xivo-dird/config.yml`. Format: YAML

The default location may be overwritten by the command line options.

Here's an example of the main configuration file:

```
debug: False
foreground: False
log_filename: /var/log/xivo-dird.log
log_level: info
pid_filename: /var/run/xivo-dird/xivo-dird.pid
source_config_dir: /etc/xivo-dird/sources.d
user: www-data

rest_api:
    wsgi_socket: /var/run/xivo-dird/xivo-dird.sock

enabled_plugins:
   backends:
       csv: true
       ldap: true
       phonebook: true
   services:
       lookup: true
   views:
       cisco_view: true
       default_json: true

views:
    displays:
        switchboard_display:
            -
                title: Firstname
                default: Unknown
                field: firstname
                type: name
            -
                title: Lastname
                default: Unknown
                field: lastname
                type: name
        default_display:
            -
                title: Firstname
                field: fn
                type: name
            -
                title: Location
                default: Canada
                field: country
            -
                title: Number
                field: number
                type: number
```

```
    displays_phone:
        default:
            name:
                - display_name
            number:
                -
                    field:
                        - phone
                -
                    field:
                        - phone_mobile
                    name_format: "{name} (Mobile)"
     profile_to_display:
         default: default_display
         switchboard: switchboard_display
     profile_to_display_phone:
         default: default

services:
    lookup:
        default:
            sources:
                my_csv: true
                ldap_quebec: true
            timeout: 0.5
        switchboard:
            sources:
                my_csv: true
                xivo_phonebook: true
                ldap_quebec: true
            timeout: 1

sources:
    my_source:
        name: my_source
        type: ldap
        ldap_option1: value
        ldap_option2: value
        ...
```

Root section

debug Enable log debug messages. Overrides `log_level`. Default: `False`.

foreground Foreground, don't daemonize. Default: `False`.

log_filename File to write logs to. Default: `/var/log/xivo-dird.log`.

log_level Logs messages with LOG_LEVEL details. Must be one of: `critical`, `error`, `warning`, `info`, `debug`. Default: `info`.

pid_filename File used as lock to avoid multiple xivo-dird instances. Default: `/var/run/xivo-dird/xivo-dird.pid`.

source_config_dir The directory from which sources configuration are read. See *Sources Configuration*. Default: `/etc/xivo-dird/sources.d`.

user The owner of the process. Default: `www-data`.

enabled_plugins section

This sections controls which plugins are to be loaded at xivo-dird startup. All plugin types must have at least one plugin enabled, or xivo-dird will not start. For back-end plugins, sources using a back-end plugin that is not enabled will be ignored.

views section

displays A dictionary describing the content of each display. The key is the display's name, and the value are the display's content.

The display content is a list of fields. Each field is a dictionary with the following keys:

- title: The label of the field
- default: The default value of the field
- type: An arbitrary identifier of the field. May be used by consumers to identify the field without matching the label. For meaningful values inside Wazo, see *Integration of xivo-dird with the rest of Wazo*.
- field: the key of the data from the source that will be used for this field.

The display may be used by a plugin view to configure which fields are to be presented to the consumer.

displays_phone A dictionary describing the content of phone-related displays. Like `displays`, the key is the display's name and the value is the display's content. These displays are used by phone-related view plugins, like the `cisco_view` plugin.

The display content contains 2 keys, `name` and `number`.

The value of the `name` key is a list of source result fields. For a given source result, the first field that will return a non-empty value will be used as the display name on the phone. For example, if `name` is configured with `["display_name", "name"]` and you have a source result with fields `{"display_name": "", "name": "Bob"}`, then "Bob" will be displayed on the phone.

The value of the `number` key is a list of number item. Each item is composed of a dictionary containing at least a `field` key, and optionally a `name_format` key. For example, if you have the following number configuration:

```
name:
    - display_name
number:
    -
        field:
            - phone
    -
        field:
            - phone_mobile
        name_format: "{name} (Mobile)"
```

and you have a source result `{"display_name": "Bob", "phone": "101", "phone_mobile": "102"}`, then 2 results will be displayed on your phone:

1. "Bob", with number "101"
2. "Bob (Mobile)", with number "102"

The `name_format` value is a python format string. There's two substitution variables available, `{name}` and `{number}`.

profile_to_display A dictionary associating a profile to a display. It allows xivo-dird to use the right display when a consumer makes a query with a profile. The key is the profile name and the value is the display name.

profile_to_display_phone: A dictionary associating a profile to a phone display. This is similar to `profile_to_display`, but only used by phone-related view plugins.

services section

This section is a dictionary whose keys are the service plugin name and values are the configuration of that service. Hence the content of the value is dependent of the service plugin. See the documentation of the service plugin (*Stock Plugins Documentation*).

sources section

This section is a dictionary whose keys are the source name and values are the configuration for that source. See the *Sources Configuration* section for more details about source configuration.

Sources Configuration

There are two ways to configure sources:

- in the sources section of the main configuration
- in files of a directory, one file for each source:
  - Default directory location `/etc/xivo-dird/sources.d`
  - Files format: YAML
  - File names are ignored
  - Each file listed in this directory will be read and used to create a data source for xivo-dird.

Here is an example of a CSV source configuration in its own file:

```
type: csv
name: my_contacts_in_a_csv_file
file: /usr/local/share/my_contacts.csv
unique_column: id
searched_columns:
    - fn
    - ln
format_columns:
    name: "{fn} {ln}"
    number: "{num}"
```

This is strictly equivalent in the main configuration file:

```
sources:
    my_contacts_in_a_csv_file:
        type: csv
        name: my_contacts_in_a_csv_file
        file: /usr/local/share/my_contacts.csv
        unique_column: id
        searched_columns:
            - fn
            - ln
```

```
        source_to_display_columns:
            ln: lastname
            fn: firstname
            num: number
```

type the type of the source. It must be the same than the name of one of the enabled back-end plugins.

name is the name of this given configuration. The name is used to associate the source to profiles. The value is arbitrary, but it must be unique across all sources.

> **Warning:** Changing the name of the source will make all favorites in that source disappear. There is currently no tool to help you migrate favorites between source names, so choose your source names carefully.

The other options are dependent on the source type (the back-end used). See the documentation of the back-end plugin (*Stock Plugins Documentation*). However, the following keys should be present in all source configurations:

first_matched_columns (optional) the columns used for the reverse lookup. Any column having the search term will be a reverse lookup result.

format_columns (optional) a mapping between result fields and a format string. The new key will be added to the result, if this name already exists in the result, it will be replaced with the new value. The syntax is a python format string. See https://docs.python.org/2/library/string.html#formatspec for a complete reference.

searched_columns (optional) the columns used for the lookup. Any column containing the search term substring will be a lookup result.

unique_column (optional) This column is what makes an entry unique in this source. The `unique_column` is used to build the `uid` that is passed to the list method to fetch a list of results by unique ids. This is necessary for listing and identifying favorites.

xivo-dird developer's guide

The xivo-dird architecture uses plugins as extension points for most of its job. It uses stevedore to do the plugin instantiation and discovery and ABC classes to define the required interface.

Plugins in xivo-dird use setuptools' entry points. That means that installing a new plugin to xivo-dird requires an entry point in the plugin's setup.py. Each entry point's *namespace* is documented in the appropriate documentation section. These entry points allow xivo-dird to be able to discover and load extensions packaged with xivo-dird or installed separately.

Each kind of plugin does a specific job. There are three kinds of plugins in dird.

1. *Back-End*
2. *Service*
3. *View*

All plugins are instantiated by the core. The core then keeps a catalogue of loaded extensions that can be supplied to other extensions.

The following setup.py shows an example of a python library that add a plugin of each kind to xivo-dird:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from setuptools import setup
from setuptools import find_packages
```

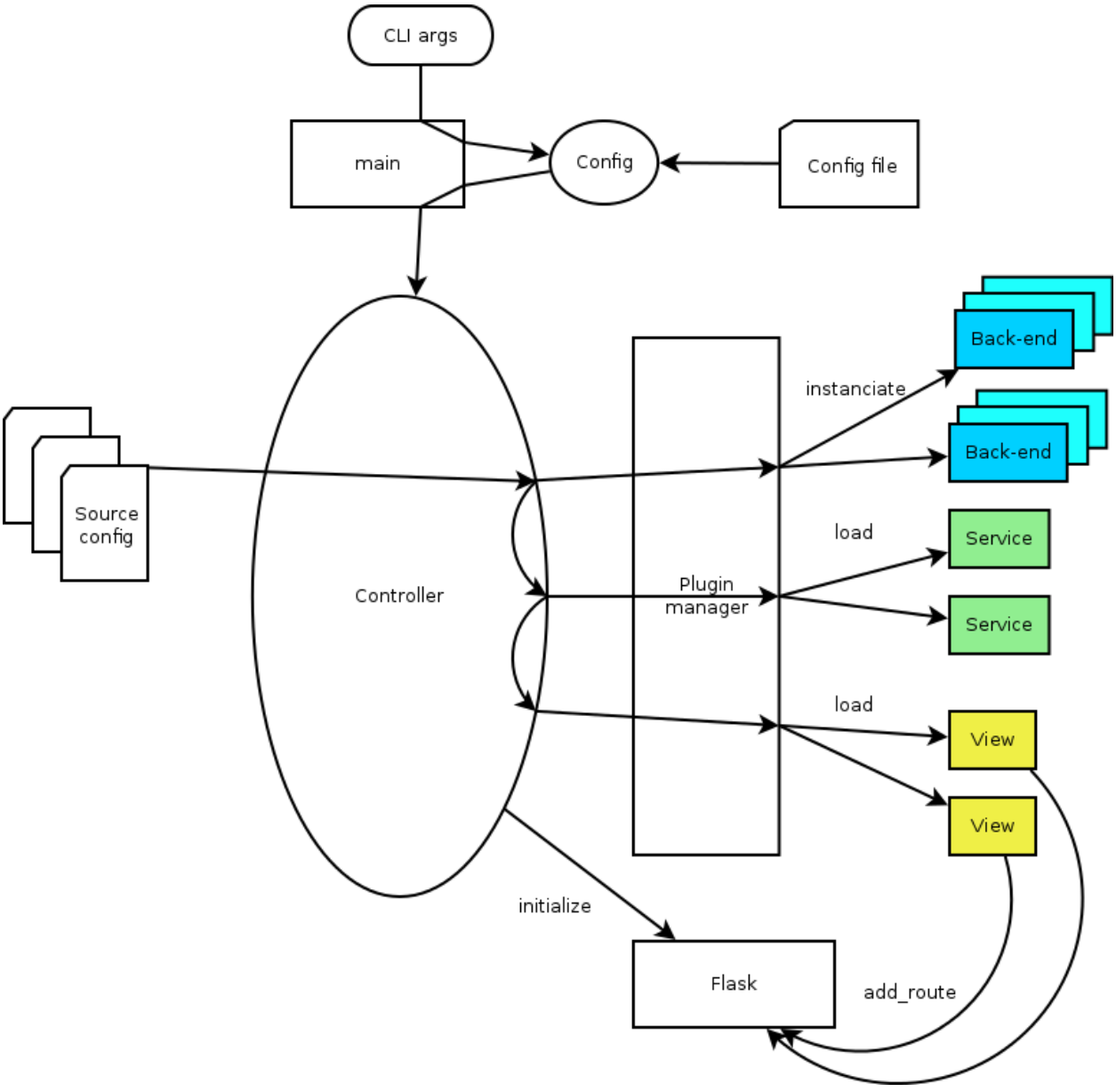


Fig. 1.31: xivo-dird startup flow

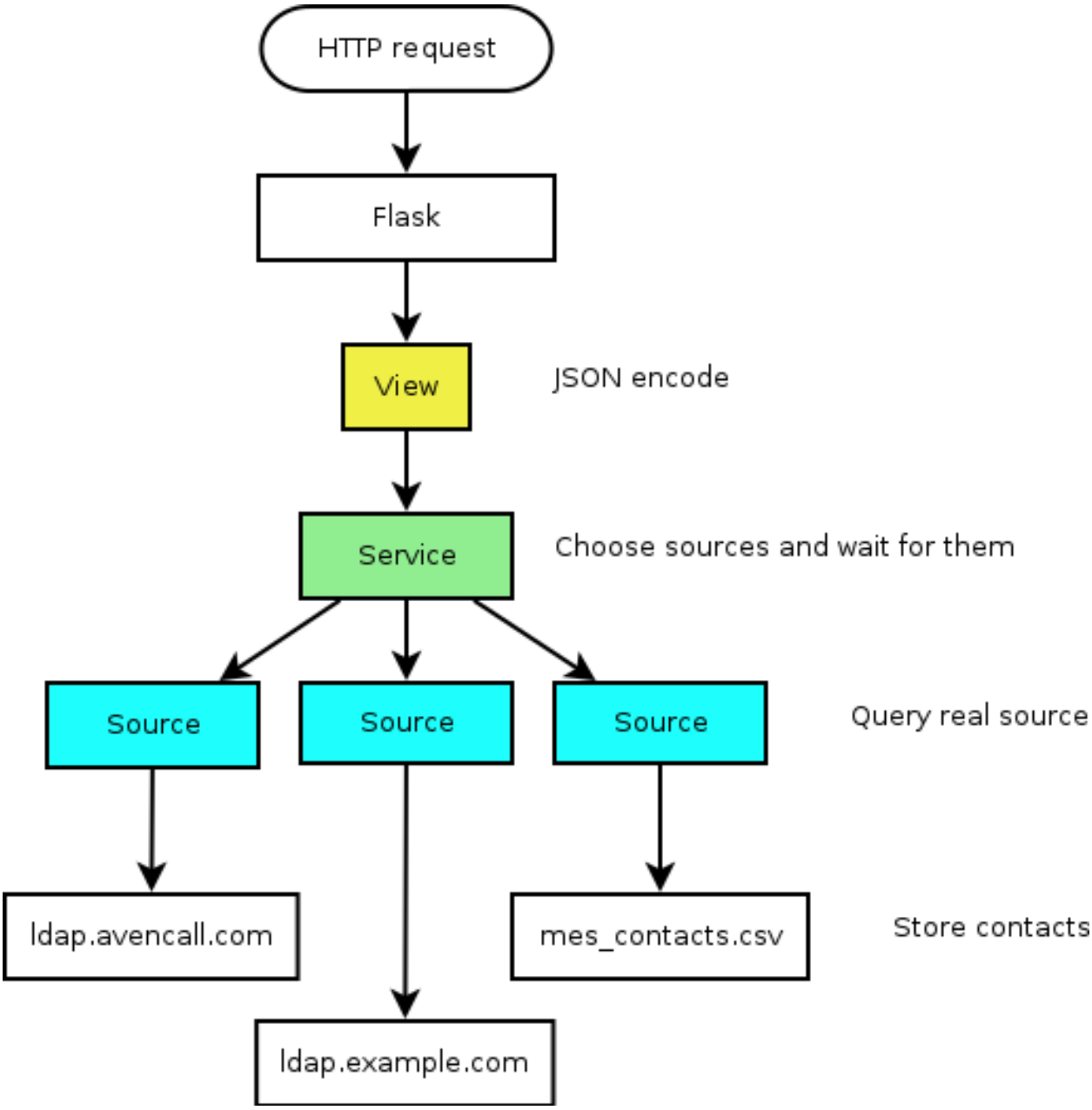


Fig. 1.32: xivo-dird HTTP query

```
6
7
8  setup(
9      name='Wazo dird plugin sample',
10     version='0.0.1',
11
12     description='An example program',
13
14     packages=find_packages(),
15
16     entry_points={
17         'xivo_dird.services': [
18             'my_service = dummy:DummyServicePlugin',
19         ],
20         'xivo_dird.backends': [
21             'my_backend = dummy:DummyBackend',
22         ],
23         'xivo_dird.views': [
24             'my_view = dummy:DummyView',
25         ],
26     }
27 )
```

Back-End

Back-ends are used to query directories. Each back-end implements a way to query a given directory. Each instance of a given back-end is called a source. Sources are used by the services to get results from each configured directory.

Given one LDAP back-end, I can configure a source from the LDAP at alpha.example.com and another source from the other LDAP at beta.example.com. Both of these sources use the LDAP back-end.

Implementation details

- Namespace: `xivo_dird.backends`
- Abstract source plugin: BaseSourcePlugin
- Methods:
  - `name`: the name of the source, typically retrieved from the configuration injected to `load()`
  - `load(args)`: set up resources used by the plugin, depending on the config. `args` is a dictionary containing:
    - key `config`: the source configuration for this instance of the back-end
    - key `main_config`: the whole configuration of xivo-dird
  - `unload()`: free resources used by the plugin.
  - `search(term, args)`: The search method returns a list of dictionary.
    - Empty values should be `None`, instead of empty string.
    - `args` is a dictionary containing:
      - key `token_infos`: data associated to the authentication token (see *wazo-auth*)
  - `first_match(term, args)`: The first_match method returns a dictionary.

    * Empty values should be `None`, instead of empty string.
    * `args` is a dictionary containing:
        · key `token_infos`: data associated to the authentication token (see *wazo-auth*)

- `list(uids, args)`: The list method returns a list of dictionary from a list of uids. Each uid is a string identifying a contact within the source.
    * `args` is a dictionary containing:
        · key `token_infos`: data associated to the authentication token (see *wazo-auth*)

See *Sources Configuration*. The implementation of the back-end should take these values into account and return results accordingly.

Example

The following example add a backend that will return random names and number.

`dummy.py`:

```
# -*- coding: utf-8 -*-

import logging

logger = logging.getLogger(__name__)

class DummyBackendPlugin(object):

    def name(self):
        return 'my_local_dummy'

    def load(self, args):
        logger.info('dummy backend loaded')

    def unload(self):
        logger.info('dummy backend unloaded')

    def search(self, term, args):
        nb_results = random.randint(1, 20)
        return _random_list(nb_results)

    def list(self, unique_ids):
        return _random_list(len(unique_ids))

    def _random_list(self, nb_results):
        columns = ['Firstname', 'Lastname', 'Number']
        return [_random_entry(columns) for _ in xrange(nb_results)]

    def _random_entry(self, columns):
        random_stuff = [_random_string() for _ in xrange(len(columns))]
        return dict(zip(columns, random_stuff))

    def _random_string(self):
        return ''.join(random.choice(string.lowercase) for _ in xrange(5))
```

Service

Service plugins add new functionality to the dird server. These functionalities are available to views. When loaded, a service plugin receives its configuration and a dictionary of available sources.

Some service examples that come to mind include:

- A lookup service to search through all configured sources.
- A reverse lookup service to search through all configured sources and return a specific field of the first matching result.

Implementation details

- Namespace: `xivo_dird.services`
- Abstract service plugin: BaseServicePlugin
- Methods:
  - `load(args)`: set up resources used by the plugin, depending on the config. `args` is a dictionary containing:
    - key `config`: the whole configuration file in dict form
    - key `sources`: a dictionary of source names to sources

    `load` must return the service object, which is any kind of python object.
  - `unload()`: free resources used by the plugin.

Example

The following example adds a service that will return an empty list when used.

`dummy.py`:

```
# -*- coding: utf-8 -*-

import logging

from xivo_dird import BaseServicePlugin

logger = logging.getLogger(__name__)

class DummyServicePlugin(BaseServicePlugin):
    """
    This plugin is responsible fow instantiating and returning the
    DummyService. It manages its life time and should take care of
    its cleanup if necessary
    """

    def load(self, args):
        """
        Ignores all provided arguments and instantiate a DummyService that
        is returned to the core
        """
        logger.info('dummy loaded')
        self._service = DummyService()
```

```
        return self._service

    def unload(self):
        logger.info('dummy unloaded')


class DummyService(object):
    """
    A very dumb service that will return an empty list every time it is used
    """

    def list(self):
        """
        This function must be called explicitly from the view, `list` is not a
        special method name for xivo-dird
        """
        return []
```

View

View plugins add new routes to the HTTP application in xivo-dird, in particular the REST API of xivo-dird: they define the URLs to which xivo-dird will respond and the formatting of data received and sent through those URLs.

For example, we can define a REST API formatted in JSON with one view and the same API formatted in XML with another view. Supporting the directory function of a phone is generally a matter of adding a new view for the format that the phone consumes.

Implementation details

- Namespace: `xivo_dird.views`
- Abstract view plugin: BaseViewPlugin
- Methods:
  - `load(args)`: set up resources used by the plugin, depending on the config. Typically, register routes on Flask. Those routes would typically call a service. `args` is a dictionary containing:
    - key `config`: the section of the configuration file for all views in dict form
    - key `services`: a dictionary of services, indexed by name, which may be called from a route
    - key `http_app`: the Flask application instance
    - key `rest_api`: a Flask-RestFul Api instance
  - `unload()`: free resources used by the plugin.

Example

The following example adds a simple view: `GET /0.1/directories/ping` answers `{"message": "pong"}`.

`dummy.py`:

```python
# -*- coding: utf-8 -*-

import logging

from flask_restful import Resource

logger = logging.getLogger(__name__)


class PingViewPlugin(object):

    name = 'ping'

    def __init__(self):
        logger.debug('dummy view created')

    def load(self, args):
        logger.debug('dummy view args: %s', args)

        args['rest_api'].add_resource(PingView, '/0.1/directories/ping')

    def unload(self):
        logger.debug('dummy view unloaded')


class PingView(Resource):
    """
    Simple API using Flask-Restful: GET /0.1/directories/ping answers "pong"
    """

    def get(self):
        return {'message': 'pong'}
```

Stock Plugins Documentation

View Plugins

default_json

View name: default_json

Purpose: present directory entries in JSON format. The format is detailed in http://api.wazo.community.

headers

View name: headers

Purpose: List headers that will be available in results from `default_json` view.

personal_view

View name: personal_view

Purpose: Expose REST API to manage personal contacts (create, delete, list).

phonebook_view

View name: phonebook_view

Purpose: Expose REST API to manage xivo-dird's internal phonebooks.

aastra_view

View name: aastra_view

Purpose: Expose REST API to search in configured directories for Aastra phone.

cisco_view

View name: cisco_view

Purpose: Expose REST API to search in configured directories for Cisco phone (see CiscoIPPhone_XML_Objects).

polycom_view

View name: polycom_view

Purpose: Expose REST API to search in configured directories for Polycom phone.

snom_view

View name: snom_view

Purpose: Expose REST API to search in configured directories for Snom phone.

thomson_view

View name: thomson_view

Purpose: Expose REST API to search in configured directories for Thomson phone.

yealink_view

View name: yealink_view

Purpose: Expose REST API to search in configured directories for Yealink phone.

Service Plugins

lookup

Service name: lookup

Purpose: Search through multiple data sources, looking for entries matching a word.

Configuration

Example (excerpt from the main configuration file):

```
services:
    lookup:
        default:
            sources:
                my_csv: true
            timeout: 0.5
```

The configuration is a dictionary whose keys are profile names and values are configuration specific to that profile.

For each profile, the configuration keys are:

sources The list of source names that are to be used for the lookup

timeout The maximum waiting time for an answer from any source. Results from sources that take longer to answer are ignored. Default: no timeout.

favorites

Service name: favorites

Purpose: Mark/unmark contacts as favorites and get the list of all favorites.

personal

Service name: personal

Purpose: Add, delete, list personal contacts of users.

phonebook

Service name: phonebook

Purpose: Add, delete, list phonebooks and phonebook contacts.

Configuration

Example (excerpt from the main configuration file):

```
services:
    favorites:
        default:
            sources:
                my_csv: true
            timeout: 0.5
```

The configuration is a dictionary whose keys are profile names and values are configuration specific to that profile.

For each profile, the configuration keys are:

sources The list of source names that are to be used for the lookup

timeout The maximum waiting time for an answer from any source. Results from sources that take longer to answer are ignored. Default: no timeout.

reverse

Service name: reverse

Purpose: Search through multiple data sources, looking for the first entry matching an extension.

Configuration

Example:

```
services:
    reverse:
        default:
            sources:
                my_csv: true
            timeout: 1
```

The configuration is a dictionary whose keys are profile names and values are configuration specific to that profile.

For each profile, the configuration keys are:

sources The list of source names that are to be used for the reverse lookup

timeout The maximum waiting time for an answer from any source. Results from sources that take longer to answer are ignored. Default: 1.

Service Discovery

Service name: service_discovery

Purpose: Creates sources when services are registered using service discovery.

To configure new sources, the service needs the following things:

1. A template the for the source configuration file.
2. A set of configuration that will be applied to the template.
3. A set of service and profile that will use the new source.

Template

The template is used to generate the content of the configuration file for the new service. Its content should be the same as the content of a source for the desired backend.

The location of the templates are configured in the service configuration

Example:

```
type: xivo
name: xivo-{{ uuid }}
searched_columns:
- firstname
- lastname
```

```
first_matched_columns:
- exten
confd_config:
  host: {{ hostname }}
  port: {{ port }}
  version: "1.1"
  username: {{ service_id }}
  password: {{ service_key }}
  https: true
  verify_certificate: false
format_columns:
  name: "{firstname} {lastname}"
  phone: "{exten}"
  number: "{exten}"
  reverse: "{firstname} {lastname}"
  voicemail: "{voicemail_number}"
```

Example:

```
services:
  service_discovery:
    template_path: /etc/xivo-dird/templates.d
    services:
      xivo-confd:
        template: confd.yml
```

In this example, the file */etc/xivo-dird/templates.d/confd.yml* would be used to create a new source configuration when a new *xivo-confd* service is registered.

The following keys are available to use in the templates:

- uuid: The Wazo uuid that was in the service registry notification
- hostname: The advertised host from the remote service
- port: The advertised port from the remote service

All other fields are configured in the *hosts* section of the service_discovery service.

Host configuration

The host section allow the administrator to configure some information that are not available in the service discovery to be available in the templates. This will typically be the *service_id* and *service_key* that are configured with the proper ACL on the remote Wazo.

Example:

```
services:
  service_discovery:
    hosts:
      ff791b0e-3d28-4b4d-bb90-2724c0a248cb:
        uuid: ff791b0e-3d28-4b4d-bb90-2724c0a248cb
        service_id: some-service-name
        service_key: secre7
        datacenter: dc1
        token: 3f031816-84a6-3960-fcd1-9cca67eacde2
```

- uuid: the XIVO_UUID of the remote Wazo

- service_id: the web service login on the remote Wazo
- service_key: the secret key of the web service
- datacenter(optional): the name of the consul datacenter on which the other Wazo is running
- token(optional): the token to access service discovery on the remote consul

Profile and service association

The service and profile association for discovered services is defined in the service_discovery service configuration.

Example:

```
services:
  service_discovery:
    services:
      xivo-confd:
        lookup:
          default: true
          foobar: true
        reverse:
          foobar: true
        favorites:
          default: true
          foobar: true
```

In this example, a new xivo-confd service would generate a configuration based on the template and that new source would be added to the lookup and favorites

Back-end Configuration

This sections completes the *Sources Configuration* section.

csv

Back-end name: csv

Purpose: read directory entries from a CSV file.

Limitations:

- the CSV delimiter is not configurable (currently: `,` (comma)).

Configuration

Example (a file inside `source_config_dir`):

```
type: csv
name: my_csv
file: /var/tmp/test.csv
unique_column: id
searched_columns:
    - fn
    - ln
```

```
first_matched_columns:
    - num
format_columns:
    lastname: "{ln}"
    firstname: "{fn}"
    number: "{num}"
```

With the CSV file:

```
id,fn,ln,num
1,Alice,Abrams,55553783147
2,Bob,Benito,5551354958
3,Charles,Curie,5553132479
```

file the absolute path to the CSV file

CSV web service

Back-end name: csv_ws

Purpose: search using a web service that returns CSV formatted results.

Given the following configuration, *xivo-dird* would call "https://example.com:8000/ws-phonebook?firstname=alice&lastname=alice" for a lookup for the term "alice".

Configuration

Example (a file inside `source_config_dir`):

```
type: csv_ws
name: a_csv_web_service
lookup_url: "https://example.com:8000/ws-phonebook"
list_url: "https://example.com:8000/ws-phonebook"
verify_certificate: False
searched_columns:
  - firstname
  - lastname
first_matched_columns:
    - exten
delimiter: ","
timeout: 16
unique_column: id
format_columns:
    number: "{exten}"
```

lookup_url the URL used for directory searches.

list_url (optional) the URL used to list all available entries. This URL is used to retrieve favorites.

verify_certificate (optional) whether the SSL cert will be verified. A CA_BUNDLE path can also be provided. Defaults to True.

delimiter (optional) the field delimiter in the CSV result. Default: ','

timeout (optional) the number of seconds before the lookup on the web service is aborted. Default: 10.

dird_phonebook

back-end name: dird_phonebook

Purpose: search the xivo-dird's internal phonebooks

Configuration:

```
type: dird_phonebook
name: phonebook
db_uri: 'postgresql://asterisk:proformatique@localhost/asterisk'
tenant: default
phonebook_id: 42
phonebook_name: main
first_matched_columns:
  - number
searched_columns:
  - firstname
  - lastname
format_columns:
    name: "{firstname} {lastname}"
```

db_uri the URI of the DB used by xivo-dird to store the phonebook.

tenant the tenant of the phonebook to query.

phonebook_name the *name* of the phonebook used by this source.

phonebook_id (deprecated, use phonebook_name) the *id* of the phonebook used by this source.

ldap

Back-end name: ldap

Purpose: search directory entries from an LDAP server.

Configuration

Example (a file inside `source_config_dir`):

```
type: ldap
name: my_ldap
ldap_uri: ldap://example.org
ldap_base_dn: ou=people,dc=example,dc=org
ldap_username: cn=admin,dc=example,dc=org
ldap_password: foobar
ldap_custom_filter: (l=québec)
unique_column: entryUUID
searched_columns:
    - cn
first_matched_columns:
    - telephoneNumber
format_columns:
    firstname: "{givenName}"
```

```
    lastname: "{sn}"
    number: "{telephoneNumber}"
```

ldap_uri the URI of the LDAP server. Can only contains the scheme, host and port part of an LDAP URL.

ldap_base_dn the DN of the entry at which to start the search

ldap_username (optional) the user's DN to use when performing a "simple" bind.

Default to an empty string.

When both ldap_username and ldap_password are empty, an anonymous bind is performed.

ldap_password (optional) the password to use when performing a "simple" bind.

Default to an empty string.

ldap_custom_filter (optional) the custom filter is used to add more criteria to the filter generated by the back end.

Example:

- ldap_custom_filter: (l=québec)
- searched_columns: [cn,st]

will result in the following filter being used for searches. `(&(l=québec)(|(cn=*%Q*)(st=*%Q*)))`

If only the custom filter is to be used, leave the `searched_columns` field empty.

This must be a valid LDAP filter, where the string `%Q` will be replaced by the (escaped) search term when performing a search.

Example: `(&(o=ACME)(cn=*%Q*))`

ldap_network_timeout (optional) the maximum time, in second, that an LDAP network operation can take. If it takes more time than that, no result is returned.

Defaults to 0.3.

ldap_timeout (optional) the maximum time, in second, that an LDAP operation can take.

Defaults to 1.0.

unique_column (optional) the column that contains a unique identifier of the entry. This is necessary for listing and identifying favorites.

For OpenLDAP, you should set this option to "entryUUID".

For Active Directory, you should set this option to "objectGUID" and also set the "unique_column_format" option to "binary_uuid".

unique_column_format (optional) the unique column's type returned by the queried LDAP server. Valid values are "string" or "binary_uuid".

Defaults to "string".

personal

Back-end name: personal

Purpose: search directory entries among users' personal contacts

You should only have one source of type `personal`, because only one will be used to list personal contacts. The `personal` backend needs a working Consul installation. This backend works with the personal service, which allows users to add personal contacts.

The complete list of fields is in *Personal contacts*.

Configuration

Example (a file inside `source_config_dir`):

```
type: personal
name: personal
first_matched_columns:
    - number
format_columns:
    firstname: "{firstname}"
    lastname: "{lastname}"
    number: "{number}"
```

`unique_column` is not configurable, its value is always `id`.

xivo

Back-end name: xivo

Purpose: add users from a Wazo (may be remote) as directory entries

Configuration

Example (a file inside `source_config_dir`):

```
type: xivo
name: my_xivo
confd_config:
    https: True
    host: xivo.example.com
    port: 9486
    version: 1.1
    username: admin
    password: password
    timeout: 3
unique_column: id
first_matched_columns:
    - exten
searched_columns:
    - firstname
    - lastname
format_columns:
    number: "{exten}"
    mobile: "{mobile_phone_number}"
```

confd_config:host the hostname of the Wazo (more precisely, of the xivo-confd service)

confd_config:port the port of the xivo-confd service (usually 9486)

confd_config:version the version of the xivo-confd API (should be 1.1)

Integration of xivo-dird with the rest of Wazo

Configuration values

Views

In the directory displays (also in the *main configuration file* of xivo-dird, in the `views` section), the following keys are interpreted and displayed in xlet people of the Wazo Client:

`title` The `title` will be shown as a header for the column

`type`

- `agent`: the field value will be ignored and replaced by an icon showing the status of the agent assigned to the contact (e.g. green icon for logged agent, red icon for unlogged agent, . . .)
- `callable`: a dropdown action on the `number` field will be added to call the field value.
- `email`: a dropdown action on the `number` field will be added to send an email to the field value.
- `favorite`: the boolean field value will be replaced by an icon showing if the status is favorite (yellow star filled) or not (yellow star empty).
- `name`: a decoration will be added to the field value (typically a color dot) showing the presence status of the contact (e.g. Disconnected, Available, Away, . . .)
- `number`: only one number type can be defined per profile. The field value will be:
  - added a decoration (typically a color dot) showing the status of the phone of the contact (e.g. Offline, Ringing, Talking, . . .)
  - replaced with a button to call the contact with your phone when using the mouse
- `personal`: the boolean field value will be used to show a deletion action for the contact
- `voicemail`: the voicemail number of the contact

See *People Xlet features Upgrade Notes* for an example with screenshots.

Personal contacts

Here are the list of available attributes of a personal contact:

- `id`
- `company`
- `email`
- `fax`
- `firstname`
- `lastname`
- `mobile`
- `number`

To be able to edit and delete personal contacts, you need a column of type *personal* in your display.

Adding the *personal* column to your display

In the web interface under *Services* → *CTI Server* → *Directories* → *Display filters*.

1. Edit the filter on which you which to enable favorites.
2. Add a column with the type *personal* and display format *personal*.

Favorites

Enabling favorites in the Wazo Client.

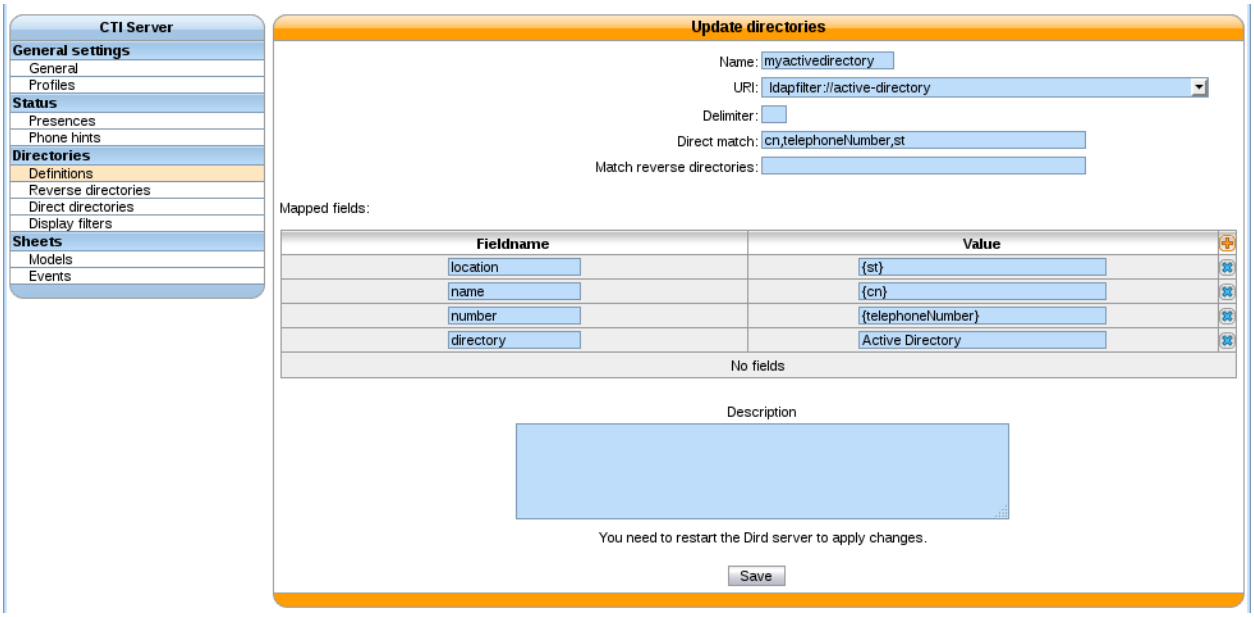
- Add a *unique_column* to your sources.
- Add a *favorite* column to your display

Adding a *unique_column* to your sources

The web interface does not allow the administrator to specify the *unique_column* and *unique_column_format*. To add these configuration options, add a file to */etc/xivo-dird/sources.d* containing *the same name* than the directory definition and all missing fields.

Example:

Given an *ldap* directory source using Active Directory named `myactivedirectory`:



Add a file `/etc/xivo-dird/sources.d/myactivedirectory.yml` with the following content to enable favorites on this source.

```
name: myactivedirectory  # the same name than the directory definition
unique_column: objectGUID
unique_column_format: binary_uuid
```

Adding the *favorite* column to your display

In the web interface under *Services* → *CTI Server* → *Directories* → *Display filters*.

1. Edit the filter on which you which to enable favorites.
2. Add a column with the type *favorite* and display format *favorite*.

Customizing sources

Some configuration options are not available in the web interface. To add configuration to a source that is configured in the web interface, create a file in */etc/xivo-dird/sources.d/* with the key *name* matching your web interface configuration and add all missing fields.

Example:

adding a timeout configuration to a CSV web service source

```
name: my_csv_web_service
timeout: 16
```

Launching xivo-dird

```
usage: xivo-dird [-h] [-c CONFIG_FILE] [-d] [-f] [-l LOG_LEVEL] [-u USER]

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG_FILE, --config-file CONFIG_FILE
                        The path where is the config file. Default: /etc/xivo-dird/
↪config.yml
  -d, --debug           Log debug messages. Overrides log_level. Default:
                        False
  -f, --foreground      Foreground, don't daemonize. Default: False
  -l LOG_LEVEL, --log-level LOG_LEVEL
                        Logs messages with LOG_LEVEL details. Must be one of:
                        critical, error, warning, info, debug. Default: info
  -u USER, --user USER  The owner of the process.
```

Terminology

Back-end

A back-end is a connector to query a specific type of directory, e.g. one back-end to query LDAP servers, another back-end to query CSV files, etc.

Source

A source is an instance of a back-end. One backend may be used multiples times to query multiple directories of the same type. For example, I could have the customer-csv and the employee-csv sources, each using the CSV back-end, but reading a different file.

Plugins

A plugin is an extension point in xivo-dird. It is a way to add or modify the functionality of xivo-dird. There are currently three types of plugins:

- Back-ends to query different types of directories (LDAP, CSV, etc.)
- Services to provide different directory actions (lookup, reverse lookup, etc.)
- Views to expose directory results in different formats (JSON, XML, etc.)

API

See http://api.wazo.community, section Wazo Dird.

1.7.21 xivo-dird-phoned

xivo-dird-phoned is an interface to use directory service with phone. It offers a simple REST interface to authenticate a phone and search result from *xivo-dird*.

Usage

xivo-dird-phoned is used through HTTP requests, using HTTP and HTTPS. Its default port is 9498 and 9499. As a user, the common operation is to search through directory from a phone. The phone need to send 2 informations:

- *xivo_user_uuid*: The Wazo user uuid that the phone is associated. It's used to search through personal contacts (see *personal*).
- *profile*: The profile that the user is associated. It's used to format results as configured.

Note: Since most phones dont't support HTTPS, a small protection is to configure authorized_subnets in *Configuration Files* or in *Services* → *General settings* → *Phonebook* → *Hosts*

Launching xivo-dird-phoned

On command line, type `xivo-dird-phoned -h` to see how to use it.

1.7.22 xivo-purge-db

Keeping records of personal communications for long periods may be subject to local legislation, to avoid personal data retention. Also, keeping too many records may become resource intensive for the server. To ease the removal of such records, `xivo-purge-db` is a process that removes old log entries from the database. This allows keeping records for a maximum period and deleting older ones.

By default, xivo-purge-db removes all logs older than a year (365 days). xivo-purge-db is run nightly.

Note: Please check the laws applicable to your country and modify `days_to_keep` (see below) in the configuration file accordingly.

Tables Purged

The following features are impacted by xivo-purge-db:

- *Call Logs*
- *Call center statistics*

More technically, the tables purged by `xivo-purge-db` are:

- `call_log`
- `cel`
- `queue_log`
- `stat_agent_periodic`
- `stat_call_on_queue`
- `stat_queue_periodic`
- `stat_switchboard_queue`

Configuration File

We recommend to override the setting `days_to_keep` from `/etc/xivo-purge-db/config.yml` in a new file in `/etc/xivo-purge-db/conf.d/`.

> **Warning:** Setting `days_to_keep` to 0 will NOT disable `xivo-purge-db`, and will remove ALL logs from your system.

See *Configuration priority* and `/etc/xivo-purge-db/config.yml` for more details.

Manual Purge

It is possible to purge logs manually. To do so, log on to the target Wazo server and run:

```
xivo-purge-db
```

You can specify the number of days of logs to keep. For example, to purge entries older than 365 days:

```
xivo-purge-db -d 365
```

Usage of `xivo-purge-db`:

```
usage: xivo-purge-db [-h] [-d DAYS_TO_KEEP]

optional arguments:
  -h, --help            show this help message and exit
  -d DAYS_TO_KEEP, --days_to_keep DAYS_TO_KEEP
                        Number of days data will be kept in tables
```

Maintenance

After an execution of `xivo-purge-db`, postgresql's Autovacuum Daemon should perform a VACUUM ANALYZE automatically (after 1 minute). This command marks memory as reusable but does not actually free disk space, which is fine if your disk is not getting full. In the case when `xivo-purge-db` hasn't run for a long time (e.g. upgrading to 15.11 or when *days_to_keep* is decreased), some administrator may want to perform a VACUUM FULL to recover disk space.

Warning: VACUUM FULL will require a service interruption. This may take several hours depending on the size of purged database.

You need to:

```
$ wazo-service stop
$ sudo -u postgres psql asterisk -c "VACUUM (FULL)"
$ wazo-service start
```

Archive Plugins

In the case you want to keep archives of the logs removed by xivo-purge-db, you may install plugins to xivo-purge-db that will be run before the purge.

Wazo does not provide any archive plugin. You will need to develop plugins for your own need. If you want to share your plugins, please open a pull request.

Archive Plugins (for Developers)

Each plugin is a Python callable (function or class constructor), that takes a dictionary of configuration as argument. The keys of this dictionary are the keys taken from the configuration file. This allows you to add plugin-specific configuration in `/etc/xivo-purge-db/conf.d/`.

There is an example plugin in the xivo-purge-db git repo.

Example

Archive name: sample

Purpose: demonstrate how to create your own archive plugin.

Activate Plugin

Each plugin needs to be explicitly enabled in the configuration of `xivo-purge-db`. Here is an example of file added in `/etc/xivo-purge-db/conf.d/`:

```
enabled_plugins:
    archives:
        - sample
```

sample.py

The following example will be save a file in `/tmp/xivo_purge_db.sample` with the following content:

```
Save tables before purge. 365 days to keep!
```

```
sample_file = '/tmp/xivo_purge_db.sample'

def sample_plugin(config):
    with open(sample_file, 'w') as output:
        output.write('Save tables before purge. {0} days to keep!'.format(config[
↪'days_to_keep']))
```

Install sample plugin

The following `setup.py` shows an example of a python library that adds a plugin to xivo-purge-db:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from setuptools import setup
from setuptools import find_packages


setup(
    name='xivo-purge-db-sample-plugin',
    version='0.0.1',

    description='An example program',
    packages=find_packages(),
    entry_points={
        'xivo_purge_db.archives': [
            'sample = xivo_purge_db_sample.sample:sample_plugin',
        ],
    }
)
```

1.7.23 xivo-sysconfd

xivo-sysconfd is the system configuration server for Wazo. It does quite a few different things; here's a non exhaustive list:

- configuring network (interfaces, hostname, DNS)
- configuring high availability
- staring/stopping/restarting services
- reloading asterisk configuration
- sending some events to components (xivo-agentd, xivo-agid and xivo-ctid)

Configuration File

Default location: `/etc/xivo/sysconfd.conf`. Format: INI.

The default location may be overwritten by the command line options.

Here's an example of the configuration file:

```
[general]
xivo_config_path = /etc/xivo
templates_path = /usr/share/xivo-sysconfd/templates
custom_templates_path = /etc/xivo/sysconfd/custom-templates
backup_path = /var/backups/xivo-sysconfd

[resolvconf]
hostname_file = /etc/hostname
hostname_update_cmd = /etc/init.d/hostname.sh start
hosts_file = /etc/hosts
resolvconf_file = /etc/resolv.conf

[network]
interfaces_file = /etc/network/interfaces

[wizard]
templates_path = /usr/share/xivo-config/templates
custom_templates_path = /etc/xivo/custom-templates

[commonconf]
commonconf_file = /etc/xivo/common.conf
commonconf_cmd = /usr/sbin/xivo-update-config
commonconf_monit = /usr/sbin/xivo-monitoring-update

[openssl]
certsdir = /var/lib/xivo/certificates

[monit]
monit_checks_dir = /usr/share/xivo-monitoring/checks
monit_conf_dir = /etc/monit/conf.d

[request_handlers]
synchronous = true

[bus]
username = guest
password = guest
host = localhost
port = 5672
exchange_name = xivo
exchange_type = topic
exchange_durable = true
```

request_handlers section

synchronous If this option is true, when xivo-sysconfd receives a request to reload the dialplan for example, it will wait for the dialplan reload to complete before replying to the request.

When this option is false, xivo-sysconfd reply to the request immediately.

Setting this option to false will speed up some operation (for example, editing a user from the web interface or from xivo-confd), but this means that there will be a small delay (up to a few seconds in the worst case) between the time you create your user and the time you can dial successfully its extension.

1.8 Ecosystem

1.8.1 Devices

The supported devices are expected to work across upgrades and phone features should work on the latest version.

Supported Devices

`xivo-provd` plugins for these devices can be installed from the *"Supported devices" repository*.

Aastra

Aastra has been acquired by Mitel in 2014. In Wazo, the 6700 series and 6800 series phones are still referenced as Aastra phones, for historical and compatibility reasons.

6700i series

	6731i	6735i	6737i	6739i	6755i
Provisioning	Y	Y	Y	Y	Y
H-A	Y	Y	Y	Y	Y
Directory XIVO	Y	Y	Y	Y	Y
Funckeys	8	26	30	55	26
	Supported programmable keys				
User with supervision function	Y	Y	Y	Y	Y
Group	Y	Y	Y	Y	Y
Queue	Y	Y	Y	Y	Y
Conference Room with supervision function	Y	Y	Y	Y	Y
General Functions					
Online call recording	N	N	N	N	N
Phone status	Y	Y	Y	Y	Y
Sound recording	Y	Y	Y	Y	Y
Call recording	Y	Y	Y	Y	Y
Incoming call filtering	Y	Y	Y	Y	Y
Do not disturb	Y	Y	Y	Y	Y
Group interception	Y	Y	Y	Y	Y
Listen to online calls	Y	Y	Y	Y	Y
Directory access	Y	Y	Y	Y	Y
Filtering Boss - Secretary	Y	Y	Y	Y	Y
Transfers Functions					
Blind transfer	HK	Y	Y	HK	Y
Indirect transfer	HK	Y	Y	HK	Y
Forwards Functions					
Disable all forwarding	Y	Y	Y	Y	Y
Enable/Disable forwarding on no answer	Y	Y	Y	Y	Y
Enable/Disable forwarding on busy	Y	Y	Y	Y	Y
Enable/Disable forwarding unconditional	Y	Y	Y	Y	Y
Voicemail Functions					
Enable voicemail with supervision function	Y	Y	Y	Y	Y

Continu

Table 1.2 – continued from previous page

	6731i	6735i	6737i	6739i	6755i
Reach the voicemail	Y	Y	Y	HK	Y
Delete messages from voicemail	Y	Y	Y	Y	Y
Agent Functions					
Connect/Disconnect a static agent	Y	Y	Y	Y	Y
Connect a static agent	Y	Y	Y	Y	Y
Disconnect a static agent	Y	Y	Y	Y	Y
Parking Functions					
Parking	Y	Y	Y	Y	Y
Parking position	Y	Y	Y	Y	Y
Paging Functions					
Paging	Y	Y	Y	Y	Y

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
6730i	No	8	Yes
6753i	Yes	6	Yes
6757i	Yes	30	Yes
9143i	Yes	7	Yes
9480i	No	6	Yes
9480CT	No	6	Yes

Supported expansion modules:

- Aastra® M670i (for Aastra® 35i/37i/39i/53i/55i/57i)
- Aastra® M675i (for Aastra® 35i/37i/39i/55i/57i)

6800i series

	6863i	6865i	6867i
Provisioning	Y	Y	Y
H-A	Y	Y	Y
Directory XIVO	Y	Y	Y
Funckeys	0	8	38
	Supported programmable keys		
User with supervision function	N	Y	Y
Group	N	Y	Y
Queue	N	Y	Y
Conference Room with supervision function	N	Y	Y
General Functions			
Online call recording	N	Y	Y
Phone status	N	Y	Y
Sound recording	N	Y	Y
Call recording	N	Y	Y

Continued on nex

[1] `Tested` means the device has been tested by the Wazo development team and that the developers have access to this device.

[2] `Fkeys` is the number of programmable function keys that you can configure from the Wazo web interface. It is not necessarily the same as the number of physical function keys the device has (for example, a 6757i has 12 physical keys but you can configure 30 function keys because of the page system).

[3] `Wazo HA` means the device is confirmed to work with *Wazo HA*.

Table 1.3 – continued from previous page

	6863i	6865i	6867i
Incoming call filtering	N	Y	Y
Do not disturb	N	Y	Y
Group interception	N	Y	Y
Listen to online calls	N	Y	Y
Directory access	N	Y	Y
Filtering Boss - Secretary	N	Y	Y
Transfers Functions			
Blind transfer	HK	HK	HK
Indirect transfer	HK	HK	HK
Forwards Functions			
Disable all forwarding	N	Y	Y
Enable/Disable forwarding on no answer	N	Y	Y
Enable/Disable forwarding on busy	N	Y	Y
Enable/Disable forwarding unconditional	N	Y	Y
Voicemail Functions			
Enable voicemail with supervision function	N	Y	Y
Reach the voicemail	N	Y	Y
Delete messages from voicemail	N	Y	Y
Agent Functions			
Connect/Disconnect a static agent	N	Y	Y
Connect a static agent	N	Y	Y
Disconnect a static agent	N	Y	Y
Parking Functions			
Parking	N	Y	Y
Parking position	N	Y	Y
Paging Functions			
Paging	N	Y	Y

Supported expansion modules:

- Aastra® M680 (for Aastra® 6865i/6867i/6869i)
- Aastra® M685 (for Aastra® 6865i/6867i/6869i)

DECT Infrastructure

	RFP35	RFP36
Provisioning	N	N
H-A	N	N
Directory XIVO	N	N
Funckeys	0	0

Alcatel-Lucent

IP Touch series:

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
4008 Extended Edition	Yes	4	No
4018 Extended Edition	Yes	4	No

Note that you *must not* download the firmware for these phones unless you agree to the fact it comes from a non-official source.

For the plugin to work fully, you need these additional packages:

```
apt-get install p7zip python-pexpect telnet
```

Avaya

1200 series IP Deskphones (previously known as Nortel IP Phones):

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
1220 IP	Yes	0	No
1230 IP	No	0	No

Cisco

Cisco Small Business SPA300 series:

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
SPA301	No	1	No
SPA303	No	3	No

Note: Function keys are shared with line keys for all SPA phones

Cisco Small Business SPA500 series:

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
SPA501G	Yes	8	No
SPA502G	No	1	No
SPA504G	Yes	4	No
SPA508G	Yes	8	No
SPA509G	No	12	No
SPA512G	No	1	No
SPA514G	No	4	No
SPA525G	Yes	5	No
SPA525G2	No	5	No

The SPA500 expansion module is supported.

Cisco Small Business IP Phones (previously known as Linksys IP Phones)

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
SPA901	No	1	No
SPA921	No	1	No
SPA922	No	1	No
SPA941	No	4	No
SPA942	Yes	4	No
SPA962	Yes	6	No

Note: You must install the firmware of each SPA9xx phones you are using since they reboot in loop when they can't find their firmware.

The SPA932 expansion module is supported.

ATAs:

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
PAP2	No	0	No
SPA2102	No	0	No
SPA8800	No	0	No
SPA112	No	0	No

For best results, activate *DHCP Integration* on your Wazo.

Note: These devices can be used to connect Faxes. For better success with faxes some parameters must be changed. You can read the *Using analog gateways* section.

Note: If you want to manually resynchronize the configuration from the ATA device you should use the following url:

```
http://ATA_IP/admin/resync?http://WAZO_IP:8667/CONF_FILE
```

where :

- *ATA_IP* is the IP address of the ATA,
- *WAZO_IP* is the IP address of your Wazo,
- *CONF_FILE* is one of `spa2102.cfg`, `spa8000.cfg`

ATAs

	SPA122	SPA3102	SPA8000
Provisioning	Y	Y	Y
H-A	N	N	N
Directory XIVO	N	N	N
Funckeys	0	0	0

For best results, activate *DHCP Integration* on your Wazo.

These devices can be used to connect faxes. For better success with faxes some parameters must be changed. You can read the *Using analog gateways* section.

Note: If you want to manually resynchronize the configuration from the ATA device you should use the following url:

```
http://ATA_IP/admin/resync?http://WAZO_IP:8667/CONF_FILE
```

where :

- *ATA_IP* is the IP address of the ATA,
- *WAZO_IP* is the IP address of your Wazo,
- *CONF_FILE* is one of `spa3102.cfg`, `spa8000.cfg`

Cisco 7900 Series

	7905G	7906G	7911G	7912G	7920
Provisioning	Y	Y	Y	Y	Y
H-A	Y	Y	Y	Y	NT
Directory XIVO	FK	FK	FK	FK	N
Funckeys	0	0	0	0	0
					Sup
User with supervision function	N	N	N	N	N
Group	N	N	N	N	N
Queue	N	N	N	N	N
Conference Room with supervision function	N	N	N	N	N
					Genera
Online call recording	N	N	N	N	N
Phone status	N	N	N	N	N
Sound recording	N	N	N	N	N
Call recording	N	N	N	N	N
Incoming call filtering	N	N	N	N	N
Do not disturb	SK	SK	SK	SK	N
Group interception	N	N	N	N	N
Listen to online calls	N	N	N	N	N
Directory access	Y	Y	Y	Y	N
Filtering Boss - Secretary	N	N	N	N	N
					Transfe
Blind transfer	N	N	N	N	N
Indirect transfer	SK	SK	SK	SK	SK
					Forwar
Disable all forwarding	N	N	N	N	N
Enable/Disable forwarding on no answer	N	N	N	N	N
Enable/Disable forwarding on busy	N	N	N	N	N
Enable/Disable forwarding unconditional	N	N	N	N	N
					Voicem
Enable voicemail with supervision function	N	N	N	N	N
Reach the voicemail	SK	SK	SK	SK	N
Delete messages from voicemail	N	N	N	N	N
					Agent F
Connect/Disconnect a static agent	N	N	N	N	N
Connect a static agent	N	N	N	N	N
Disconnect a static agent	N	N	N	N	N
					Parking
Parking	N	N	N	N	N
Parking position	N	N	N	N	N
					Paging
Paging	N	N	N	N	N

Warning: These phones can only be used in SCCP mode. They are limited to the *features supported in Wazo's SCCP implementation*.

To install firmware for xivo-cisco-sccp plugins, you need to manually download the firmware files from the Cisco website and save them in the `/var/lib/xivo-provd/plugins/$plugin-name/var/cache` directory.

File permissions should be modified to make the files readable to *xivo-provd*:

- *chmod 640 <filename>*
- *chown xivo-provd:xivo-provd <filename>*

This directory is created by Wazo when you install the plugin (i.e. xivo-cisco-sccp-legacy). If you create the directory manually, the installation will fail.

Warning: Access to Cisco firmware updates requires a Cisco account with sufficient privileges. The account requires paying for the service and remains under the responsibility of the client or partner. The Wazo authors is not responsible for these firmwares and does not offer any updates.

For example, if you have installed the `xivo-cisco-sccp-legacy` plugin and you want to install the `7940-7960-fw`, `networklocale` and `userlocale_fr_FR` package, you must:

- Go to http://www.cisco.com
- Click on "Log In" in the top right corner of the page, and then log in
- Click on the "Support" menu
- Click on the "Downloads" tab, then on "Voice & Unified Communications"
- Select "IP Telephony", then "Unified Communications Endpoints", then the model of your phone (in this example, the 7940G)
- Click on "Skinny Client Control Protocol (SCCP) software"
- Choose the same version as the one shown in the plugin
- Download the file with an extension ending in ".zip", which is usually the last file in the list
- In the Wazo web interface, you'll then be able to click on the "install" button for the firmware

The procedure is similar for the network locale and the user locale package, but:

- Instead of clicking on "Skinny Client Control Protocol (SCCP) software", click on "Unified Communications Manager Endpoints Locale Installer"
- Click on "Linux"
- Choose the same version of the one shown in the plugin
- For the network locale, download the file named "po-locale-combined-network.cop.sgn"
- For the user locale, download the file named "po-locale-$locale-name.cop.sgn, for example "po-locale-fr_FR.cop.sgn" for the "fr_FR" locale
- Both files must be placed in `/var/lib/xivo-provd/plugins/$plugin-name/var/cache` directory. Then install them in the Wazo Web Interface.

Note: Currently user and network locale 11.5.1 should be used for plugins xivo-sccp-legacy and xivo-cisco-sccp-9.4

Digium

	D40	D50	D70
Provisioning	Y	NYT	Y
H-A	Y	NYT	Y
Directory XIVO	N	NYT	N
Funckeys	2	14	106
Supported programmable keys			
User with supervision function	N	NYT	N
Group	Y	NYT	Y
Queue	Y	NYT	Y
Conference Room with supervision function	Y	NYT	Y
General Functions			
Online call recording	N	NYT	N
Phone status	Y	NYT	Y
Sound recording	Y	NYT	Y
Call recording	Y	NYT	Y
Incoming call filtering	Y	NYT	Y
Do not disturb	HK	NYT	HK
Group interception	Y	NYT	Y
Listen to online calls	N	NYT	N
Directory access	N	NYT	N
Filtering Boss - Secretary	Y	NYT	Y
Transfers Functions			
Blind transfer	HK	NYT	HK
Indirect transfer	HK	NYT	HK
Forwards Functions			
Disable all forwarding	Y	NYT	Y
Enable/Disable forwarding on no answer	Y	NYT	Y
Enable/Disable forwarding on busy	Y	NYT	Y
Enable/Disable forwarding unconditional	Y	NYT	Y
Voicemail Functions			
Enable voicemail with supervision function	Y	NYT	Y
Reach the voicemail	HK	NYT	HK
Delete messages from voicemail	Y	NYT	Y
Agent Functions			
Connect/Disconnect a static agent	Y	NYT	Y
Connect a static agent	Y	NYT	Y
Disconnect a static agent	Y	NYT	Y
Parking Functions			
Parking	N	NYT	N
Parking position	N	NYT	N
Paging Functions			
Paging	Y	NYT	Y

Note: Some function keys are shared with line keys

Particularities:

- For best results, activate *DHCP Integration* on your Wazo.

- Impossible to do directed pickup using a BLF function key.
- Only supports DTMF in RFC2833 mode.
- Does not work reliably with Cisco ESW520 PoE switch. When connected to such a switch, the D40 tends to reboot randomly, and the D70 does not boot at all.
- It's important to not edit the phone configuration via the phones' web interface when using these phones with Wazo.
- Paging doesn't work.

Fanvil

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
C62P	Yes	5	Yes

Gigaset

Also known as Siemens.

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
C470 IP	No	0	No
C475 IP	No	0	No
C590 IP	No	0	No
C595 IP	No	0	No
C610 IP	No	0	No
C610A IP	No	0	No
S675 IP	No	0	No
S685 IP	No	0	No
N300 IP	No	0	No
N300A IP	No	0	No
N510 IP PRO	No	0	No

Jitsi

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
Jitsi	Yes	—	No

Mitel

The Mitel 6700 Series and 6800 Series SIP Phones are supported in Wazo. See the *Aastra* section.

Patton

The following analog VoIP gateways are supported:

	SN4112	SN4114	SN4116	SN4118	SN4316	SN4324	SN4332
Provisioning	Y	Y	Y	Y	Y	Y	Y
H-A	Y	Y	Y	Y	Y	Y	Y

Wazo only supports configuring the FXS ports of these gateways. It does not support configuring the FXO ports. If you have a gateway on which you would like to configure the FXO ports, you'll need to write the FXO ports configuration manually by creating a *custom template* for your gateway.

It's only possible to enter a provisioning code on the first FXS port of a gateway. For example, if you have a gateway with 8 FXS ports, the first port can be configured by dialing a provisioning code from it, but ports 2 to 7 can only be configured via the Wazo web interface. Also, if you dial the *"reset to autoprov" extension* from any port, the configuration of all the ports will be reset, not just the port on which the extension was dialed. These limitations might go away in the future.

These gateways are configured with a few regional parameters (France by default). These parameters are easy to change by writing a *custom template*.

Telnet access and web access are enabled by default. You should change the default password by setting an administrator password via a Wazo "template device".

By downloading and installing the Patton firmwares, you agree to the Patton Electronics Company conditions.

To provision a gateway that was previously configured manually, use the following commands on your gateway (configure mode), replacing WAZO_IP by the IP address of your Wazo server:

```
profile provisioning PF_PROVISIONING_CONFIG
  destination configuration
  location 1 http://WAZO_IP:8667/$(system.mac).cfg
  activation reload graceful
  exit
provisioning execute PF_PROVISIONING_CONFIG
```

Panasonic

Panasonic KX-HTXXX series:

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
KX-HT113	No	—	No
KX-HT123	No	—	No
KX-HT133	No	—	No
KX-HT136	No	—	No

Note: This phone is for testing for the moment

Polycom

	|SoundPoint IP		
	SPIP331	SPIP335	SPIP450
Provisioning	NT	Y	Y

	|SoundPoint IP		
H-A	N	Y	N
Directory XIVO	N	N	N
Funckeys	N	0	2
User with supervision function	NYT	N	NYT
Group	NYT	N	NYT
Queue	NYT	N	NYT
Conference Room with supervision function	NYT	N	NYT
Online call recording	NYT	N	NYT
Phone status	NYT	N	NYT
Sound recording	NYT	N	NYT
Call recording	NYT	N	NYT
Incoming call filtering	NYT	N	NYT
Do not disturb	NYT	SK	NYT
Group interception	NYT	N	NYT
Listen to online calls	NYT	N	NYT
Directory access	NYT	N	NYT
Filtering Boss - Secretary	NYT	N	NYT
Blind transfer	NYT	SK	NYT
Indirect transfer	NYT	SK	NYT
Disable all forwarding	NYT	N	NYT
Enable/Disable forwarding on no answer	NYT	SK	NYT
Enable/Disable forwarding on busy	NYT	SK	NYT
Enable/Disable forwarding unconditional	NYT	SK	NYT
Enable voicemail with supervision function	NYT	N	NYT
Reach the voicemail	NYT	SK	NYT
Delete messages from voicemail	NYT	N	NYT
Connect/Disconnect a static agent	NYT	N	NYT
Connect a static agent	NYT	N	NYT
Disconnect a static agent	NYT	N	NYT
Parking	NYT	N	NYT
Parking position	NYT	N	NYT
Paging	NYT	N	NYT

Particularities:

- The latest Polycom firmwares can take a lot of time to download and install due to their size (~650 MiB). For this reason, these files are explicitly excluded from the Wazo backups.
- For directed call pickup to work via the BLF function keys, you need to make sure that the option *Set caller-id in dialog-info+xml notify* is enabled on your Wazo. This option is located on the *Services → IPBX → General settings → SIP Protocol* page, in the *Signaling* tab.

  Also, directed call pickup via a BLF function key will not work if the extension number of the supervised user is different from its caller ID number.

- Default password is **9486** (i.e. the word "xivo" on a telephone keypad).
- On the VVX101 and VVX201, to have the two line keys mapped to the same SIP line, create a *custom template* with the following content:

```
{% extends 'base.tpl' -%}

{% block remote_phonebook -%}
{% endblock -%}

{% block model_specific_parameters -%}
reg.1.lineKeys="2"
{% endblock -%}
```

  This is especially useful on the VVX101 since it supports a maximum of 1 SIP line and does not support function keys.

Note: (Wazo HA cluster) BLF function key saved on the master node are not available.

Supported expansion modules:

- Polycom® VVX Color Expansion (for Polycom® VVX 300/310/400/410/500/600)
- Polycom® VVX Paper Expansion (for Polycom® VVX 300/310/400/410/500/600)
- Polycom® SoundPoint IP Backlit (for Polycom® SoundPoint 650)

Warning: Polycom® VVX® Camera are not supported.

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
SPIP320	No	0	No
SPIP321	No	0	No
SPIP330	No	0	No
SPIP430	No	0	No
SPIP501	Yes	0	No
SPIP600	No	0	No
SPIP601	No	0	No
SPIP670	No	47	No

SoundStation IP:

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
SPIP4000	No	0	No

Others:

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
VVX1500	No	0	No

Snom

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
300	No	6	Yes
320	Yes	12	Yes
360	No	—	Yes
820	Yes	4	Yes
MP	No	—	Yes
PA1	No	0	Yes

Warning: If you are using Snom phones with HA, you should not assign multiple lines to the same device.

There's a known issue with the provisioning of Snom phones in Wazo:

- After a factory reset of a phone, if no language and timezone are set for the "default config device" in *Wazo* → *Configuration* → *Provisioning* → *Template device*, you will be forced to select a default language and timezone on the phone UI.

	370	710	715	720	D725	D745	760	D76
Provisioning	Y	Y	Y	Y	Y	Y	Y	Y
H-A	Y	Y	Y	Y	Y	Y	Y	Y
Directory XIVO	HK	SK	SK	HK	HK	HK	HK	HK
Funckeys	12	5	5	18	18	32	16	16
	Supported programmable keys							
User with supervision function	Y	Y	Y	Y	Y	Y	Y	Y
Group	Y	Y	Y	Y	Y	Y	Y	Y
Queue	Y	Y	Y	Y	Y	Y	Y	Y
Conference Room with supervision function	Y	Y	Y	Y	Y	Y	Y	Y
	General Functions							
Online call recording	N	N	N	N	N	N	N	N
Phone status	Y	Y	Y	Y	Y	Y	Y	Y
Sound recording	Y	Y	Y	Y	Y	Y	Y	Y
Call recording	Y	Y	Y	Y	Y	Y	Y	Y
Incoming call filtering	Y	Y	Y	Y	Y	Y	Y	Y
Do not disturb	HK	SK	SK	HK	HK	HK	HK	HK
Group interception	Y	Y	Y	Y	Y	Y	Y	Y
Listen to online calls	Y	Y	Y	Y	Y	Y	Y	Y
Directory access	Y	Y	Y	Y	Y	Y	Y	Y
Filtering Boss - Secretary	Y	Y	Y	Y	Y	Y	Y	Y
	Transfers Functions							
Blind transfer	Y	SK	SK	HK	HK	HK	HK	HK
Indirect transfer	Y	SK	SK	HK	HK	HK	HK	HK
	Forwards Functions							
Disable all forwarding	Y	Y	Y	Y	Y	Y	Y	Y
Enable/Disable forwarding on no answer	Y	Y	Y	Y	Y	Y	Y	Y
Enable/Disable forwarding on busy	Y	Y	Y	Y	Y	Y	Y	Y
Enable/Disable forwarding unconditional	Y	Y	Y	Y	Y	Y	Y	Y
	Voicemail Functions							
Enable voicemail with supervision function	Y	Y	Y	Y	Y	Y	Y	Y

Co

Table 1.7 – continued from previous page

	370	710	715	720	D725	D745	760	D76
Reach the voicemail	HK	HK	HK	HK	HK	HK	HK	HK
Delete messages from voicemail	Y	Y	Y	Y	Y	Y	Y	Y
Agent Functions								
Connect/Disconnect a static agent	Y	Y	Y	Y	Y	Y	Y	Y
Connect a static agent	Y	Y	Y	Y	Y	Y	Y	Y
Disconnect a static agent	Y	Y	Y	Y	Y	Y	Y	Y
Parking Functions								
Parking	Y	N	N	N	N	N	N	N
Parking position	Y	N	N	N	N	N	N	N
Paging Functions								
Paging	Y	Y	Y	Y	Y	Y	Y	Y

Supported expansion modules:

- Snom® Vision (for Snom® 7xx series and Snom® 8xx series)
- Snom® D7 (for Snom® 7xx series)

Note: For some models, function keys are shared with line keys

There's the following known limitations/issues with the provisioning of Snom phones in Wazo:

- If you are using Snom phones with *HA*, you should not assign multiple lines to the same device.
- The Snom D745 has limited space for function key labels: long labels might be split in a suboptimal way.
- When using a D7 expansion module, the function key label will not be shown on the first reboot or resynchronization. You'll need to reboot or resynchronize the phone a second time for the label to be shown properly.
- After a factory reset of a phone, if no language and timezone are set for the "default config device" in *Wazo* → *Configuration* → *Provisioning* → *Template device*, you will be forced to select a default language and timezone on the phone UI.

Technicolor

Previously known as Thomson:

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
ST2022	No	—	—
ST2030	Yes	10	Yes

Note: Function keys are shared with line keys

Yealink

	T19P	T19P E2	T20P	T21P	T21P E2	T22P	T26P	T28P	T32G
Provisioning	Y	Y	Y	Y	Y	Y	Y	Y	NT

Table 1.8 – continued from previous page

	T19P	T19P E2	T20P	T21P	T21P E2	T22P	T26P	T28P	T32G
H-A	Y	Y	Y	Y	Y	Y	Y	Y	N
Directory XIVO	N	Y	N	N	Y	N	N	N	Y
Funckeys	0	0	2	2	2	3	13	16	3
	Supported programmable keys								
User with supervision function	N	N	Y	Y	Y	Y	Y	Y	NYT
Group	N	N	Y	Y	Y	Y	Y	Y	NYT
Queue	N	N	Y	Y	Y	Y	Y	Y	NYT
Conference Room with supervision function	N	N	Y	Y	Y	Y	Y	Y	NYT
	General Functions								
Online call recording	N	N	N	N	N	N	N	N	NYT
Phone status	N	N	Y	Y	Y	Y	Y	Y	NYT
Sound recording	N	N	Y	Y	Y	Y	Y	Y	NYT
Call recording	N	N	Y	Y	Y	Y	Y	Y	NYT
Incoming call filtering	N	N	Y	Y	Y	Y	Y	Y	NYT
Do not disturb	N	N	Y	SK	SK	SK	SK	SK	NYT
Group interception	N	N	Y	Y	Y	Y	Y	Y	NYT
Listen to online calls	N	N	Y	Y	Y	Y	Y	Y	NYT
Directory access	N	N	Y	Y	Y	Y	Y	Y	NYT
Filtering Boss - Secretary	N	N	Y	Y	Y	Y	Y	Y	NYT
	Transfers Functions								
Blind transfer	SK	SK	HK	HK	HK	HK	HK	HK	NYT
Indirect transfer	SK	SK	HK	HK	HK	HK	HK	HK	NYT
	Forwards Functions								
Disable all forwarding	N	N	Y	Y	Y	Y	Y	Y	NYT
Enable/Disable forwarding on no answer	N	N	Y	Y	Y	Y	Y	Y	NYT
Enable/Disable forwarding on busy	N	N	Y	Y	Y	Y	Y	Y	NYT
Enable/Disable forwarding unconditional	N	N	Y	Y	Y	Y	Y	Y	NYT
	Voicemail Functions								
Enable voicemail with supervision function	N	N	Y	Y	Y	Y	Y	Y	NYT
Reach the voicemail	N	N	HK	HK	HK	HK	HK	HK	NYT
Delete messages from voicemail	N	N	Y	Y	Y	Y	Y	Y	NYT
	Agent Functions								
Connect/Disconnect a static agent	N	N	Y	Y	Y	Y	Y	Y	NYT
Connect a static agent	N	N	Y	Y	Y	Y	Y	Y	NYT
Disconnect a static agent	N	N	Y	Y	Y	Y	Y	Y	NYT
	Parking Functions								
Parking	N	N	Y	Y	Y	Y	Y	Y	NYT
Parking position	N	N	Y	Y	Y	Y	Y	Y	NYT
	Paging Functions								
Paging	N	N	Y	Y	Y	Y	Y	Y	NYT

Regarding the W52P (DECT), there is firmware for both the base station and the handset. The base and the handset are probably going to work if they are not using the same firmware version, although this does not seem to be officially recommended. By default, a base station will try to upgrade the firmware of an handset over the air (OTA) if the following conditions are met:

- Handset with firmware 26.40.0.15 or later
- Base station with firmware 25.40.0.15 or later
- Handset with hardware 26.0.0.6 or later

Otherwise, you'll have to manually upgrade the handset firmware via USB.

In all cases, you should consult the Yealink documentation on Upgrading W52x Handset Firmware.

Note: Some function keys are shared with line keys

Supported expansion modules:

- Yealink® EXP38 (for Yealink® T26P/T28P)
- Yealink® EXP39 (for Yealink® T26P/T28P)
- Yealink® EXP40 (for Yealink® T46G/T48G)

Model	Tested[1]	Fkeys[2]	Wazo HA[3]	Plugin
CP860	No	0	—	xivo-yealink-v72
T23P	No	3	—	xivo-yealink-v80
T23G	Yes	3	Yes	xivo-yealink-v80
T27P	Yes	21	Yes	xivo-yealink-v80
T29G	No	27	—	xivo-yealink-v80
T49G	Yes	29	Yes	xivo-yealink-v80

Note: Some function keys are shared with line keys

Zenitel

Model	Tested[1]	Fkeys[2]	Wazo HA[3]
IP station	Yes	1	No

The supported devices page lists, for each vendor, a table that shows the various features supported by Wazo. Here's an example:

	Model X	Model Y	Model Z
Provisioning	Y	Y	Y
H-A	Y	Y	Y
Directory XIVO	N	Y	Y
Funckeys	0	2	8
	Supported programmable keys		
User with supervision function	Y	Y	Y

The rows have the following meaning:

Provisioning Is the device supported by the *auto-provisioning* system?

H-A Is the device supported by the *high availability* system?

Directory XiVO Is the device supported by the *remote directory*? In other word, is it possible to consult the XiVO's remote directory from the device?

Funckeys How many function keys can be configured on the device from the Wazo web interface?

The number of function keys that can be configured on a device is not necessarily the same as the number of physical function keys the device has. For example, an Aastra 6757i has 12 physical keys but you can configure 30 function keys because of the page system.

Inside a table, the following legend is used:

- Y = Yes / Supported
- N = No / Not supported
- NT = Not tested
- NYT = Not yet tested

Each table also contains a section about the supported function keys. In that section, the following legend can also be used:

- FK = Funckey
- SK = SoftKey
- HK = HardKey
- MN = Menu

Function keys work using the extensions in *Services* → *Extensions*. It is important to enable the function keys you want to use. Also, the enable transfer option in the user configuration services tab must be enabled to use transfer function keys.

1.9 Administration

1.9.1 Advanced Configuration

This section describes the advanced system configuration.

Wazo General Settings

Wazo offers the possibility to configure the general settings via the *Configuration* → *Management* → *General* page.

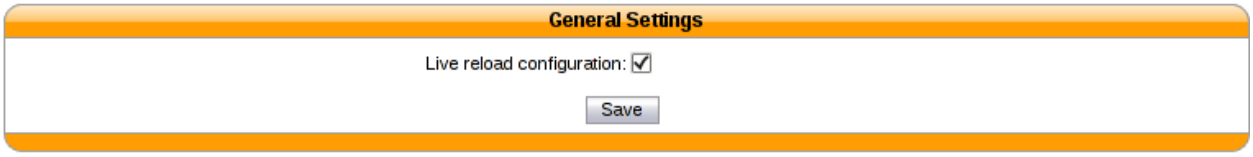


Fig. 1.33: Configure Wazo General Settings

Live reload configuration permit to reload its configuration on command received from WEBI (this option is enabled by default).

Telephony certificates

Wazo offers the possibility to create and manage X.509 certificates via the the *Configuration* → *Management* → *Certificates* page.

These certificates can be used for:

- enabling SIP TLS
- enabling encryption between the CTI server and the Wazo Clients

For the certificate used for HTTPS, see *Certificates for HTTPS*.

Creating certificates

You can add a certificate by clicking on the add button at the top right of the page. You'll then be shown this page:

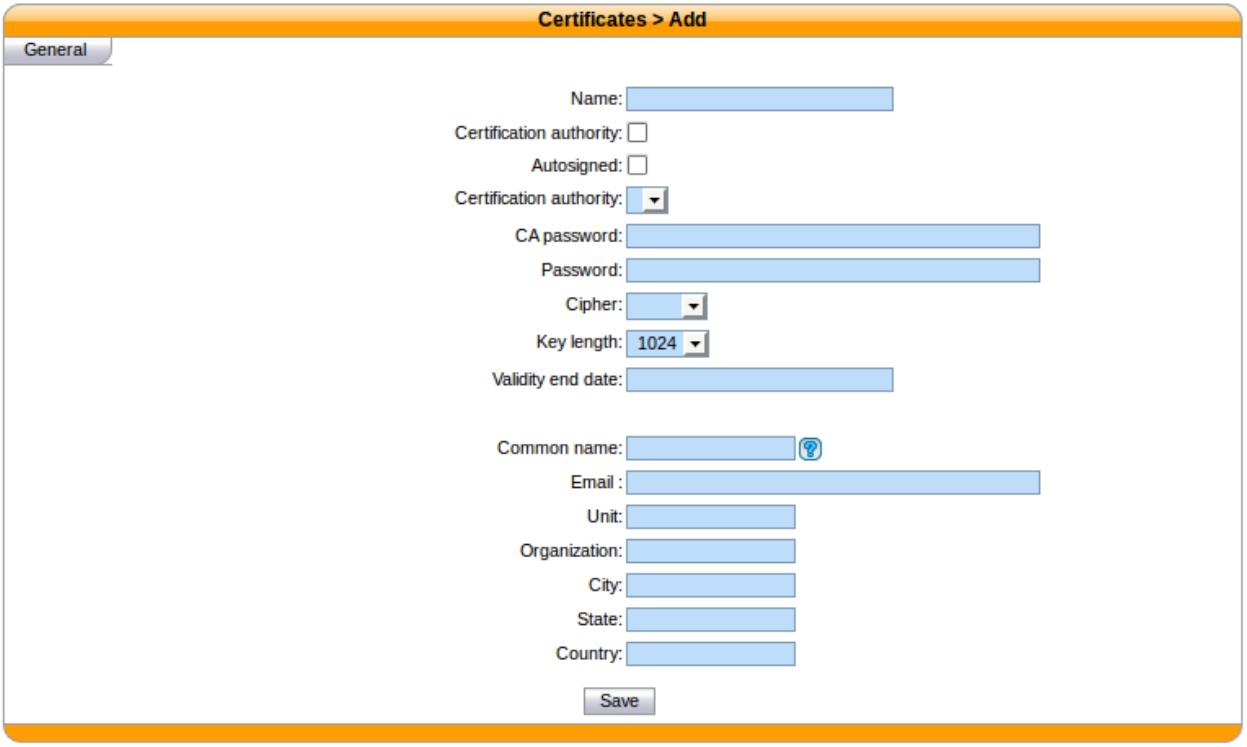


Fig. 1.34: Adding a certificate

You should look at the *examples* if you don't know which attributes to set when creating your certificates.

Removing certificates

When removing a certificate, you should remove all the files related to that certificates.

> **Warning:** If you remove a certificate that is used somewhere in Wazo, then you need to manually reconfigure that portion of Wazo.
>
> For example, if you remove the certificate files used for SIP TLS, then you need to manually disable SIP TLS or asterisk will look for certificate file but it won't be able to find them.

Examples

In the following examples, if a field is not specified than you should leave it at its default value.

Creating certificates for SIP TLS

You need to create both a CA certificate and a server certificate.

CA certificate:

- *Name* : phones-CA
- *Certification authority (checkbox)* : checked
- *Autosigned* : checked
- *Valid end date* : at least one month in the future
- *Common name* : the FQDN (Fully Qualified Domain Name) of your Wazo
- *Organization* : your organization's name, or blank
- *Email* : your email or organization's email

Server certificate:

- *Name* : phones
- *Certification authority (select)* : phones-CA
- *Valid end date* : at least one month in the future
- *Common name* : the FQDN of your Wazo
- *Organization* : your organization's name, or blank
- *Email* : your email or organization's email

Creating certificate for CTI server

- *Name* : xivo-ctid
- *Autosigned* : checked
- *Valid end date* : at least one month in the future
- *Common name* : the FQDN of your Wazo
- *Organization* : your organization's name, or blank
- *Email* : your email or organization's email

Warning: You must *not* set a password for the certificate. If the certificate is password protected, the CTI server will not be able to use it.

1.9.2 Boss Secretary Filter

The boss secretary filter allow to set a secretary or a boss role to a user. Filters can then be created to filter calls directed to a boss using different strategies.

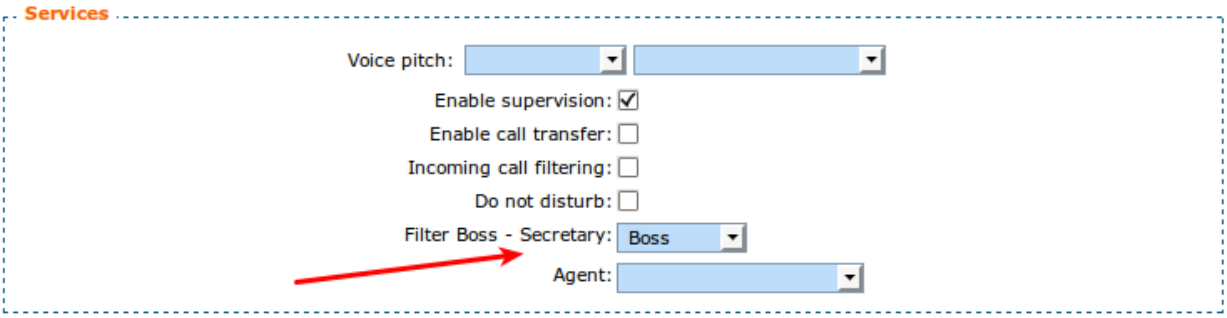
Quick Summary

In order to be able to use the boss secretary filter you have to :

- Select a boss role for one the users
- Select a secretary role for one ot the users
- Create a filter to set a strategy for this boss secretary filter
- Add a function key for the user boss and the user secretary

Defining a Role

The secretary or boss role can be set in the user's configuration page under the service tab. To use this feature, at least one boss and one secretary must be defined.



Creating a Filter

The filter is used to associate a boss to one or many secretaries and to set a ring strategy. The call filter is added in the *Services* → *IPBX* → *Call management* → *Call filters* page.

Different ringing strategies can be applied :

- Boss rings first then all secretaries one by one
- Boss rings first then secretaries are all ringing simultaneously
- Secretaries ring one by one
- Secretaries are all ringing simultaneously
- Boss and secretaries are ringing simultaneously
- Change the caller id if the secretary wants to know which boss was initialy called

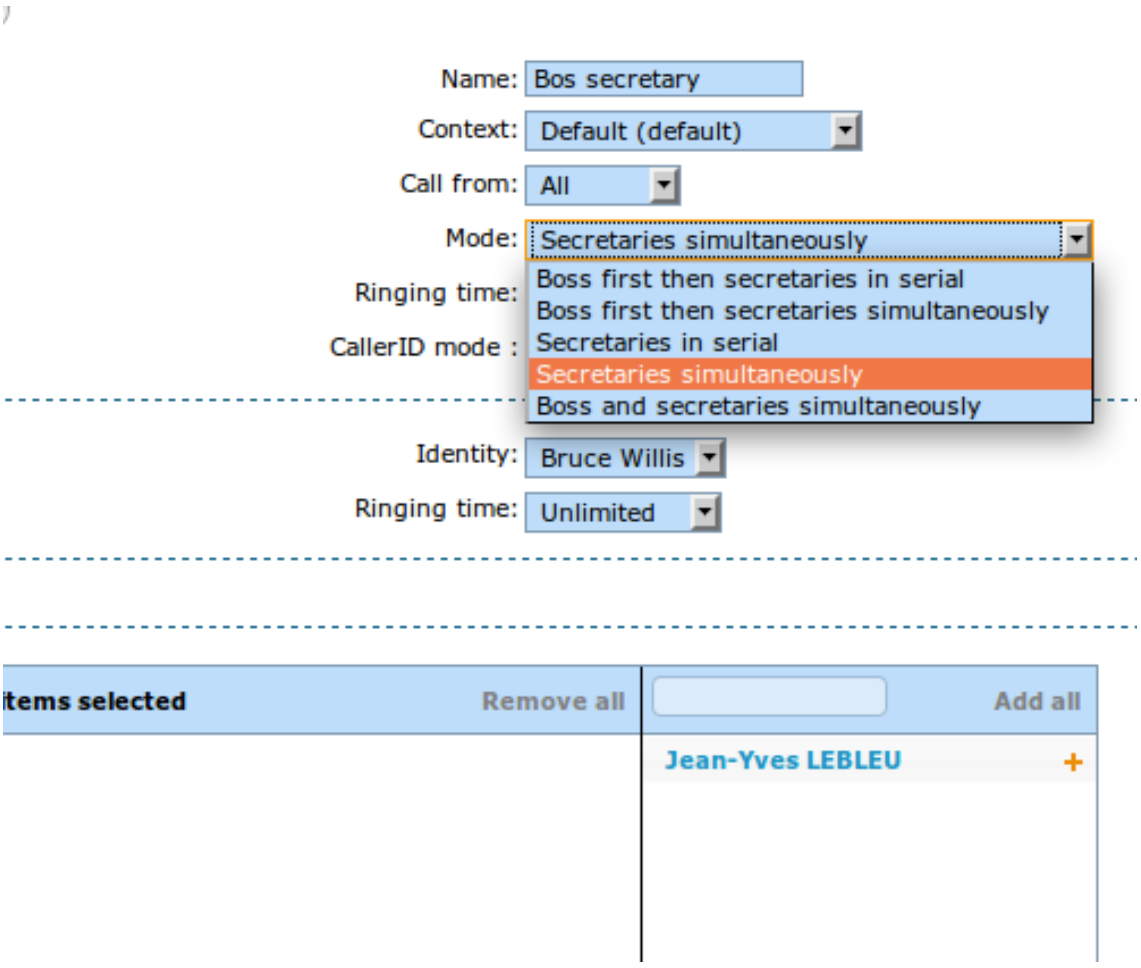
When one of serial strategies is used, the first secretary called is the last in the list. The order can be modified by drag and drop in the list.

Usage

The call filter function can be activated and deactivated by the boss or the secretary using the *37 extension. The extension is defined in *IPBX services* > *Extensions*.

The call filter has to be activated for each secretary if more than one is defined for a given boss.

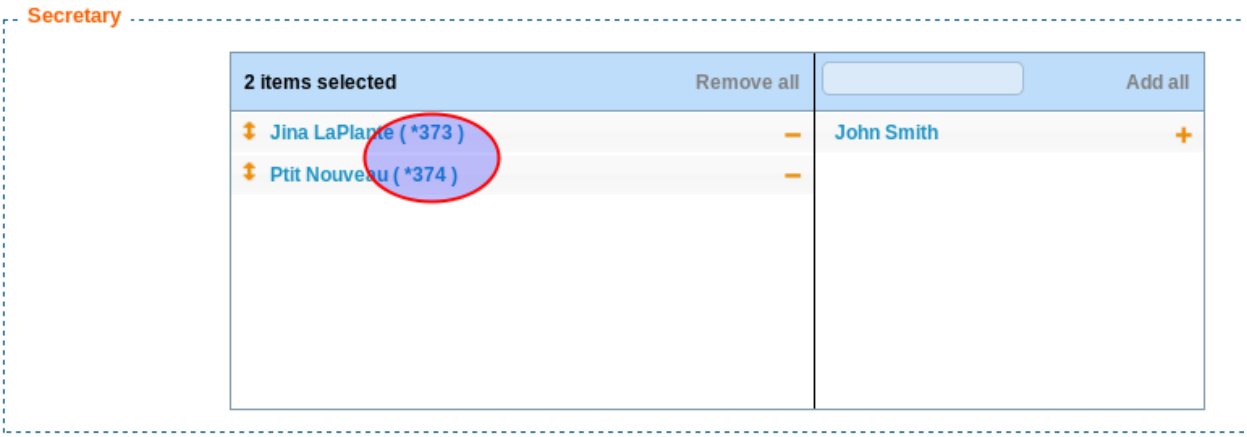
The extension to use is `*37<callfilter member id>`.


Name: Bos secretary
Context: Default (default)
Call from: All
Mode: Secretaries simultaneously
Boss first then secretaries in serial
Boss first then secretaries simultaneously
Secretaries in serial
Secretaries simultaneously
Boss and secretaries simultaneously
Ringing time:
CallerID mode :
Identity: Bruce Willis
Ringing time: Unlimited
items selected
Remove all
Add all
Jean-Yves LEBLEU

In this example, you would set 2 `Func Keys` `*373` and `*374` on the Boss.

On the secretary `Jina LaPlante` you would set `*373`.

On the secretary `Ptit Nouveau` you would set `*374`.



Function Keys

A more convenient way to active the boss secretary filter is to assign a function key on the boss' phone or the secretary's phone. In the user's configuration under `Func Keys`. A function key can be added for each secretaries of a boss.

If supervision is activated, the key will light up when filter is activated for this secretary. If a secretary also has a function key on the same boss/secretary combination the function key's BLF will be in sync between each phones.

> **Warning:** With SCCP phones, you must configure a custom `Func Keys`.

1.9.3 Call Completion

The call completion feature (or CCSS, for Call Completion Supplementary Services) in Wazo allows for a caller to be automatically called back when a called party has become available.

1. To illustrate, let's say Alice attempts to call Bob.
2. Bob is currently on a phone call with Carol, though, so Bob rejects the call from Alice
3. Alice then dials *40 to request call completion.
4. Once Bob has finished his phone call, Alice will be automatically called back by the system.
5. When she answers, Bob will be called on her behalf.

This feature has been introduced in XiVO in version 14.17.

Description

Call completion can be used in two scenarios:

- when the called party is busy (Call Completion on Busy Subscriber)
- when the called party doesn't answer (Call Completion on No Response)

We have already discussed the busy scenario in the introduction section.

Let's now illustrate the no answer scenario:

1. Alice attempts to call Bob.
2. Bob doesn't answer the phone. Alternatively, Alice hangs up before Bob has the time to answer the call.
3. Alice then dial *40 to request call completion.
4. When Bob's phone becomes busy and then is no longer busy, Alice is automatically called back.
5. When she answers, Bob will be called on her behalf.

The important thing to note here is step #4. Bob's phone needs to become busy and then no longer busy for Alice to be called back. This means that if Bob was away when Alice called him, but when he came back he did not received nor placed any call, then Alice will not be called back.

In fact, in all scenarios, after call completion has been requested by the caller, the called phone needs to transition from busy to no longer busy for the caller to be called back. This means that in the following scenario:

1. Alice attempts to call Bob.
2. Bob is currently on a phone call, so he doesn't answer the call from Alice.
3. Bob finish his call a few seconds later.
4. Alice then dials *40 to request call completion (Bob is not busy anymore).

Then, for Alice to be called back, Bob needs to become busy and then not busy.

If Alice is busy when Bob becomes not busy, then the call completion callback will only happen after both Alice and Bob are not busy.

When call completion is active, it can be cancelled by dialing the *40 extension.

Some timers governs the use of call completion. These are:

- offer timer: the time the caller has to request call completion. Defaults to 30 (seconds).
- busy available timer: when call completion on busy subscriber is requested, if this timer expires before the called party becomes available, then the call completion attempt will be cancelled. Defaults to 900 (seconds).
- no response available timer: similar to the "busy available timer", but when call completion on no response is requested. Defaults to 900 (seconds).
- recall timer: when the caller who requested call completion is called back, how long the original caller's phone rings before giving up. Defaults to 30 (seconds).

It's currently impossible to modify the value of these timers in Wazo.

Special Scenarios

There are four special scenarios:

- the call completion will not activate
- the call completion will activate and call back for the original called party
- the call completion will activate and call back for the rerouted called party
- the call completion will activate and call back for the original called party but fail to join him

Call completion will not activate

It is not possible to activate call completion in the following two scenarios.

First scenario: Alice tries to call Bob, but Bob has currently reached its "simultaneous calls" limit. When activating call completion, Alice hears that the call completion can not be activated.

Note: The "simultaneous calls" option is configured per user via the Wazo web interface.

Second scenario: Alice tries to call Bob, but the call is redirected to Charlie.

This occurs when Bob redirects/rejects the call with any of the following:

- Unconditional call forwarding towards Charlie
- Closed schedule towards Charlie
- Call permission forbidding Alice to call Bob
- Preprocess subroutine forwarding the call towards Charlie

Call completion will activate and call back for the original called party

Scenario: Alice tries to call Bob, but the call is redirected to Charlie. When activating call completion, Alice hears that the call completion is activated and eventually Alice is called back to speak with Bob.

This occurs when Bob redirects/rejects the call with any of the following:

- No-answer call forwarding towards Charlie
- Busy call forwarding towards Charlie

Call completion will activate and call back for the rerouted called party

Scenario: Alice tries to call Bob, but the call is redirected to Charlie. When activating call completion, Alice hears that the call completion is activated and eventually Alice is called back to speak with Charlie.

This occurs when Bob redirects the call with any of the following:

- Boss-Secretary filter to the secretary Charlie

Call completion will activate and call back for the original called party but fail to join him

Scenario: Alice tries to call Bob, but the call is redirected to Charlie. When activating call completion, Alice hears that the call completion is activated and eventually Alice is called back to speak with Bob. But when Alice answers, Bob is not called. If Alice activates call completion again, she will hear that the call completion was cancelled.

This occurs when Bob redirects/rejects the call with any of the following:

- Do Not Disturb mode
- a new call forwarding rule that was applied after Alice activated call completion:
  - Unconditional call forwarding towards Charlie
  - Closed schedule towards Charlie
  - Call permission forbidding Alice to call Bob

  - Preprocess subroutine forwarding the call towards Charlie

Limitations

- Call completion can only be used with SIP lines. It can't be used with SCCP lines.
- It can't be used with outgoing calls and incoming calls, except if these calls are passing through a customized trunk of type Local.
- It can't be used with groups or queues.
- The call completion feature can't be enabled only for a few users; either all users have access to it, or none.

Configuration

The call completion extension is enabled via the *Services* → *IPBX* → *IPBX services* → *Extensions* page, in the *General* tab.

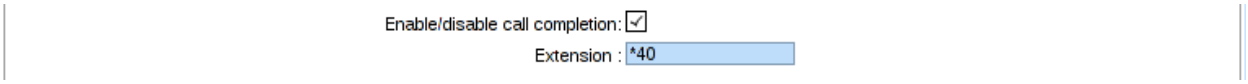


Fig. 1.35: Call Completion Extension

If your Wazo has been installed in version 14.16 or earlier, then this extension is by default disabled. Otherwise, this extension is by default enabled.

1.9.4 Call Permissions

You can manage call permissions via the *Services* → *IPBX* → *Call management* → *Call permissions* page.

Call permissions can be used for:

- denying a user from calling a specific extension
- denying a user of a group from calling a specific extension
- denying a specific extension on a specific outgoing call from being called
- denying an incoming call coming from a specific extension from calling you

More than one extension can match a given call permission, either by specifying more than one extension for that permission or by using extension patterns.

You can also create permissions that allow a specific extension to be called instead of being denied. This make it possible to create a general "deny all" permission and then an "allow for some" one.

Finally, instead of unconditionally denying calling a specific extension, call permissions can instead challenge the user for a password to be able to call that extension.

As you can see, you can do a lot of things with Wazo's call permissions. They can be used to create fairly complex rules. That said, it is probably *not* a good idea to so because it's pretty sure you'll get it somehow wrong.

Examples

Note that when creating or editing a call permission, you must at least:

- fill the *Name* field

- have one extension / extension pattern in the *Extensions* field

Denying a user from calling a specific extension

- Add the extension in the extensions list
- In the *Users* tab, select the user

Note: User's *Rightcall Code* (*Services* -> *IPBX* -> *IPBX Settings* -> *Users* under *Services* tab) overwrite all password call permissions for the user.

Warning: The extension can be anything but it will only work if it's the extension of a user or an extension that pass through an outgoing call. It does *not* work, for example, if the extension is the number of a conference room.

Denying a user of a group from calling a specific extension

First, you must create a group and add the user to this group. Note that groups aren't required to have a number.

Then,

- Add the extension in the extensions list
- In the *Groups* tab, select the group

Denying users from calling a specific extension on a specific outgoing call

- Add the extension in the extensions list
- In the *Outgoing calls* tab, select the outgoing call

Note that selecting both a user and an outgoing call for the same call permission doesn't mean the call permission applies only to that user. In fact, it means that the user can't call that extension and that the extension can't be called on the specific outgoing call. This in redundant and you will get the same result by not selecting the user.

Denying an incoming call coming from a specific extension from calling you

Call permissions on incoming calls are semantically different from the other scenarios since the extension that you add to the permission will match the extension of the caller (i.e. the caller number) and *not* the extension that the caller dialed (i.e. the callee number).

- Add the extension in the extensions list.
- In the *Incoming calls* tab, select the incoming call

1.9.5 Call Recording

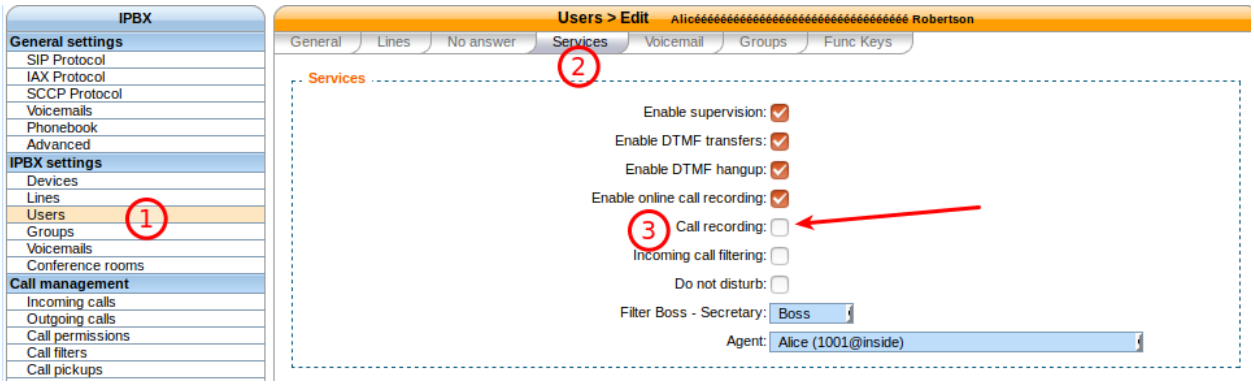
Call recording allow the user of the administrator to record a user's conversation. Recorded files are stored on the Wazo server and are accessible using the web interface.

Enabling

There are many ways to enable call recording. It can be done by the administrator or the user himself.

Administrator

The administrator can enable call recording from the user form in the web interface.



User

The user can enable and disable call recording using the **26* extension on its phone. The user can also enable call recording during a call using the **3* extension during the conversation.

Call Recording Management

Extensions

The extensions for call recording and online call recording are available in the web interface in the extension form.

Disable user call control management

To disable call recording for user, disable the *Call recording* extension in the web interface.

To disable online call recording, uncheck the *Enable online call recording* option in the user form.

Files

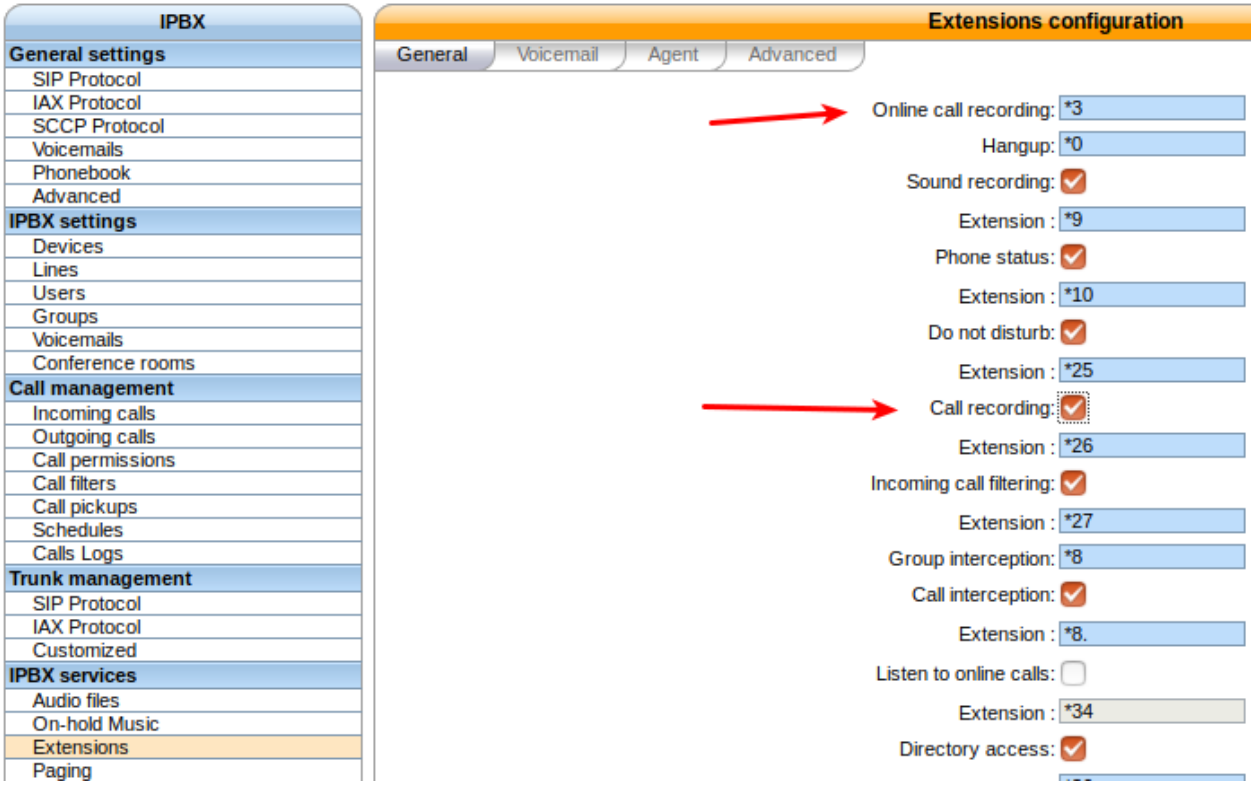
Recorded files are available in the web interface in the *Audio files*, *monitor* menu.

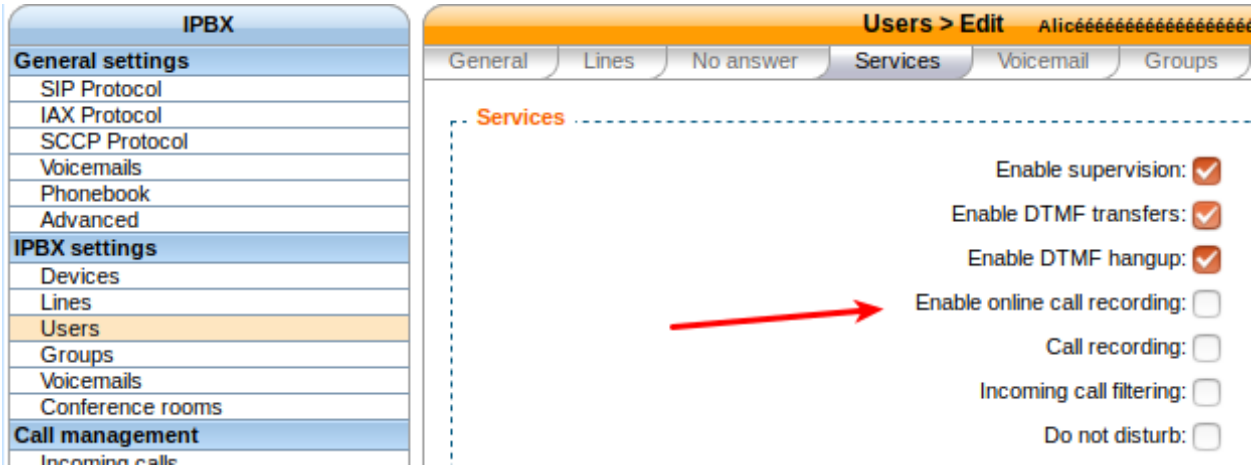
Recordings are located in */var/spool/asterisk/monitor*

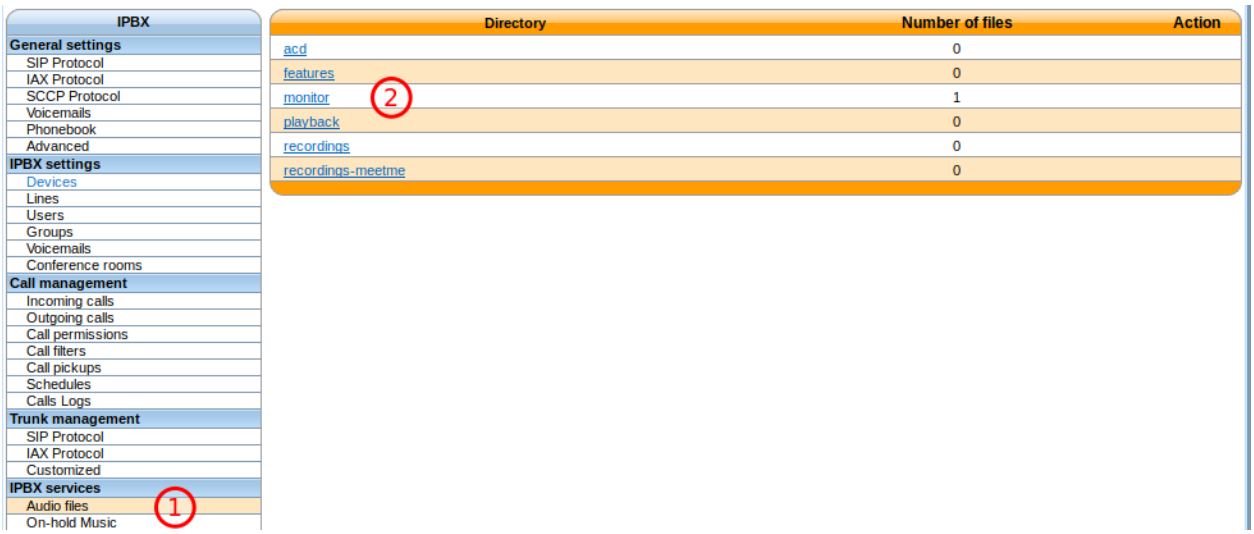
File names

The file names for call recording can be customized using Jinja2 templates.

The following variables can be used in the file name:


IPBX
General settings
SIP Protocol
IAX Protocol
SCCP Protocol
Voicemails
Phonebook
Advanced
IPBX settings
Devices
Lines
Users
Groups
Voicemails
Conference rooms
Call management
Incoming calls
Outgoing calls
Call permissions
Call filters
Call pickups
Schedules
Calls Logs
Trunk management
SIP Protocol
IAX Protocol
Customized
IPBX services
Audio files
On-hold Music
Extensions
Paging
Extensions configuration
General
Voicemail
Agent
Advanced
Online call recording: *3
Hangup: *0
Sound recording:
Extension : *9
Phone status:
Extension : *10
Do not disturb:
Extension : *25
Call recording:
Extension : *26
Incoming call filtering:
Extension : *27
Group interception: *8
Call interception:
Extension : *8.
Listen to online calls:
Extension : *34
Directory access:


IPBX
General settings
SIP Protocol
IAX Protocol
SCCP Protocol
Voicemails
Phonebook
Advanced
IPBX settings
Devices
Lines
Users
Groups
Voicemails
Conference rooms
Call management
Users > Edit
General
Lines
No answer
Services
Voicemail
Groups
Services
Enable supervision:
Enable DTMF transfers:
Enable DTMF hangup:
Enable online call recording:
Call recording:
Incoming call filtering:
Do not disturb:



- srcnum: The caller ID number of the caller
- dstnum: The called extension
- timestamp: A unix timestamp
- local_time: The formated date in the server's timezone
- utc_time: The formated date in UTC
- base_context: The context in which this call entered the Wazo dialplan
- tenant_name: The originating context's entity

Note: You **must** restart xivo-agid to take any config change into effect:

```
systemctl restart xivo-agid
```

Example 1:

Creating recording in a sub-directory for each entity

A file with the following content in */etc/xivo-agid/conf.d/call_recording.yml*:

```
call_recording:
  filename_template: "{{ tenant_name }}/{{ utc_time }}-{{ srcnum }}-{{ dstnum }}"
```

This configuration would write the files in */var/spool/asterisk/monitor/<tenant_name/*. The name of the files would be *<utc_time>-<srcnum>-<dstnum>.wav*

Example 2:

Creating recording in another directory

A file with the following content in */etc/xivo-agid/conf.d/call_recording.yml*:

```
call_recording:
  filename_template: "/home/pcm/call/user-{{ srcnum }}-{{ dstnum }}-{{ timestamp }}"
```

This configuration would write the files in the */home/pcm/call* directory. The name of the files would be *user-<srcnum>-<dstnum>-<timestamp>.wav*. Which is the default with another location.

Note: recording that are not directly in */var/spool/asterisk/monitor* will not be shown in the web interface.

Note: Asterisk needs write permission to be able to write the recordings in the configured directory.

The filename for online call recording cannot be configured from the configuration file but can be modified using a pre-process subroutine.

The file format is always *auto-timestamp-<TOUCH_MIXMONITOR>.wav*. TOUCH_MIXMONITOR is a channel variable that can be set before the call starts.

File extensions

For online call recording, the file format can be modified using the *TOUCH_MIXMONITOR_FORMAT* channel variable.

For call recording the default value is *wav* and can be modified with a configuration file.

Example:

Add a file names */etc/xivo-agid/conf.d/recording.yml* with the following content:

```
call-recording:
  filename_extension: wav
```

1.9.6 Call Logs

Call logs allow users to see the history of the calls placed and received by Wazo.

Note: The oldest call logs are periodically removed. See *xivo-purge-db* for more details.

Search Dashboard

Call logs can be accessed using the menu *Services* → *IPBX* → *Call management* → *Call Logs* page.

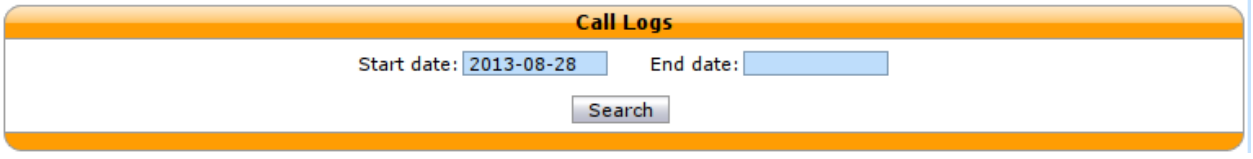


Fig. 1.36: Calls Records Dashboard

Specifying no start date returns all available call logs. Specifying a start date and no end date returns all call logs from start date until now.

Call logs are presented in a CSV format. Here's an example:

```
Call Date,Caller,Called,Period,user Field
2015-01-02T00:00:00,Alice (1001),1002,2,userfield
```

The CSV format has the following specifications:

- field names are listed on the first line
- fields are separated by commas: `,`
- if there is a comma in a field value, the value is surrounded by double quotes: `"`
- the UTF-8 character encoding is used

REST API

Call logs are also available from *wazo-call-logd REST API*.

Categorize call logs with custom tags

Sometimes, it's useful to separate call logs according to a specific value (department, city, etc.). It's possible with the `User field` of a user and the `tags` of a call log. Each `User field` will be copied into the `tags` for a call log and each `User field` must be separated by a comma.

Example

Your company has employees in the *accounting* and *sales* departments. To list call logs from the *sales* department, you must set the `User field` of each user to `sales`. Now when a user tagged with `sales` places or receives a call, this call will be also tagged `sales`. You can now filter call logs by tags `sales`.

Manual generation

Call logs can also be generated manually. To do so, log on to the target Wazo server and run:

```
wazo-call-logs
```

To avoid running for too long in one time, the call logs generation is limited to the N last unprocessed CEL entries (default 20,000). This means that successive calls to `wazo-call-logs` will process N more CELs, making about N/10 more calls available in call logs, going further back in history, while processing new calls as well.

You can specify the number of CEL entries to consider. For example, to generate calls using the 100,000 last unprocessed CEL entries:

```
wazo-call-logs -c 100000
```

Regeneration of call logs

Since call logs are based on CEL, they can be deleted and generated without problems. To regenerate the last month of call logs:

```
wazo-call-logs delete -d 30
wazo-call-logs generate -d 30  // the default behavior of wazo-call-logs command is␣
↪`generate`
```

Technicals

Call logs are pre-generated from CEL entries. The generation is done automatically by wazo-call-logd. wazo-call-logs is also run nightly to generate call logs from CEL that were missed by wazo-call-logd.

1.9.7 CallerID

The CallerID is what users see on their phones when they emit or receive a call, e.g. `Rick Sanchez 963-555-9296`.

The CallerID is composed of two parts: the CallerID name and the CallerID number.

In Wazo, the format is: `"Rick Sanchez" <9635559296>`.

CallerID for internal calls

Users calling each other will see the CallerID configured in the *Caller ID* field of each user.

CallerID for outgoing calls (through a trunk)

There are multiple settings coming into play:

- The calling user's *Outgoing Caller ID*
- The outgoing call's *Callerid* (one for each *Exten*)
- The trunk's operator rules

The current logic for outgoing calls is:

- If the call is not emitted by a user: use the outgoing call's CallerID
- If the call is emitted by a user:
  - If the *Outgoing Caller ID* is Default, use the outgoing call's CallerID
  - If the *Outgoing Caller ID* is Anonymous, remove the CallerID
  - If the *Outgoing Caller ID* is set, use it

Once the call is sent into the trunk, the operator may still override the CallerID before routing the call to the destination. Each operator has its own rules about CallerID: some will always rewrite the CallerID that is attached to the trunk, others will leave the CallerID untouched, some operators will only rewrite the CallerID if you use an unauthorized CallerID, etc.

CallerID for incoming calls (from a trunk)

There are multiple settings coming into play, in order of priority:

1. SIP trusting remote-party CallerID
2. The trunk's *CallerID*
3. CallerID number normalization
4. The Incoming Call's *CallerID mode*
5. Reverse lookup

SIP CallerID

To accept the CallerID sent via all SIP trunks, enable the option *Services* → *IPBX* → *General settings* → *SIP Protocol* → *Default*:

- Trust the Remote-Party-ID: yes

This option may also be enabled on specific SIP trunks, instead of globally.

Trunk CallerID

The trunk's *CallerID* option overwrites the incoming CallerID. Usually, this options is left blank to leave the incoming CallerID untouched.

CallerID number normalization

See *Incoming caller number display* for details.

Incoming Call CallerID

The Incoming Call's *CallerID mode* can prepend, append or overwrite the incoming CallerID.

Reverse Lookup

Reverse lookup is the operation of finding the CallerID name from the CallerID number. Wazo can lookup this information in multiple sources, see *Directories* for more details.

This operation is only triggered when the incoming CallerID has no CallerID name or when the CallerID name equals the CallerID number.

1.9.8 CLI Tools

Wazo comes with a collection of console (CLI) tools to help administer the server.

xivo-dist

xivo-dist is the wazo repository sources manager. It is used to switch between distributions (production, development, release candidate, archived version). Example use cases :

- switch to production repository : `xivo-dist phoenix`
- switch to development repository : `xivo-dist wazo-dev-stretch`
- switch to release candidate repository : `xivo-dist wazo-rc`
- switch to an archived version's repository (here 14.18) : `xivo-dist xivo-14.18`

wazo-reset

wazo-reset is a tool to reset some of the state of a Wazo installation to a pre-wizard state. It does not try to reset everything and will *not* give the same result as a fresh new Wazo installation. For example, all customizations that you have made that are not stored in the database (e.g. installing Debian packages, adding custom configuration files, etc), will not be reset. This tool should be used with extra care.

After using it, you need to pass the wizard once again.

1.9.9 Conference Room

Adding a conference room

In this example, we'll add a conference room with number 1010.

First, you need to define a conference room number range for the ''default" context via the ''Services / IPBX / IPBX configuration / Contexts" page.

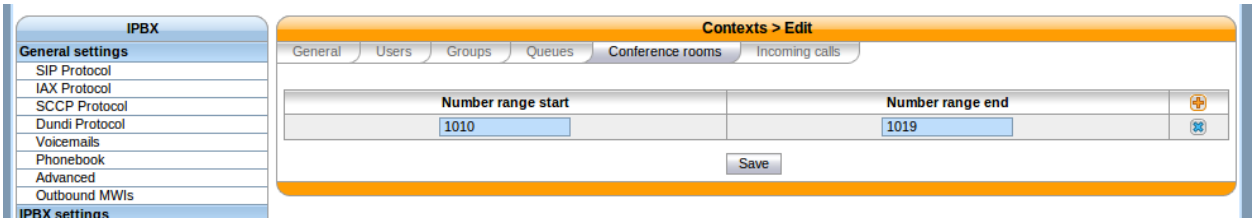


Fig. 1.37: Adding a conference room number range to the default context

You can then create a conference room via the ''Services / IPBX / IPBX settings / Conference rooms" page.

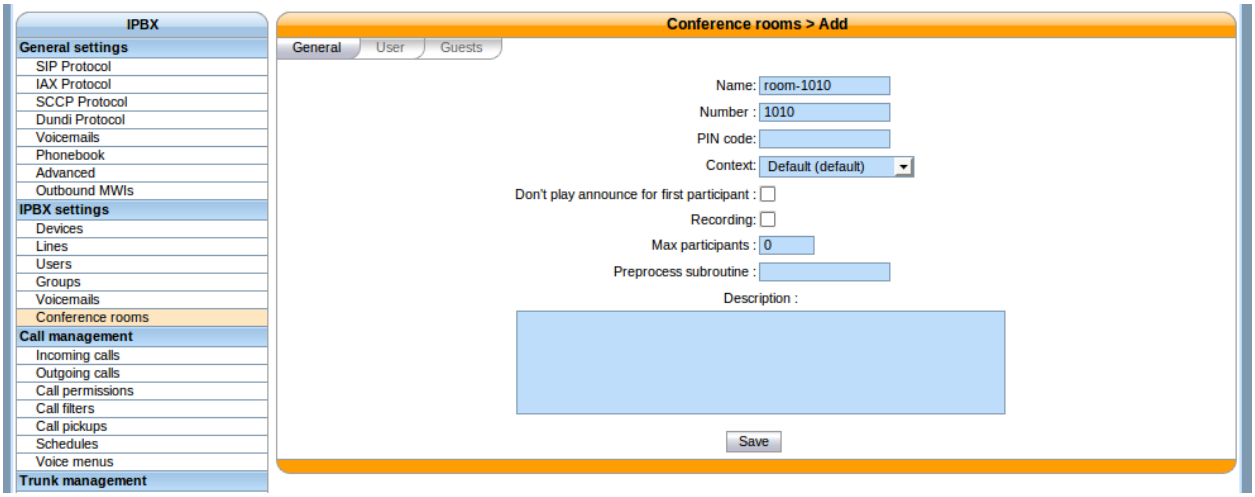


Fig. 1.38: Creating conference room 1010

In this example, we have only filled the ''Name" and ''Number" fields, the others have been left to their default value.

As you can see, there's quite a few options when adding / editing a conference room. Here's a description of the most common one:

General / PIN code Protects your conference room with a PIN number. People trying to join the room will be asked for the PIN code.

General / Don't play announce for first participant Don't play the "you are currently the only person in this conference" for the first participant.

General / Max participants Limits the number of participants in the conference room. A value of 0 means unlimited.

1.9.10 CTI Server

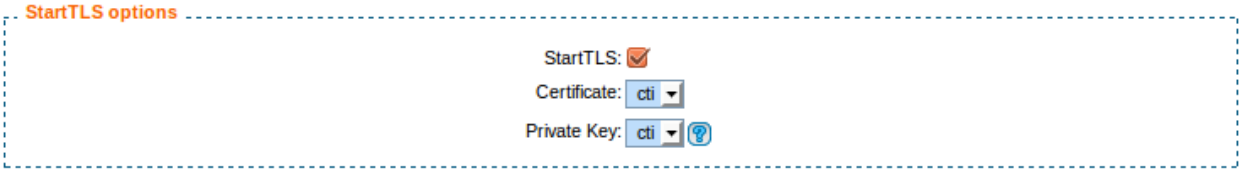
The CTI server configuration options can be found in the web-interface under the services tab.

General Options

The general options allow the administrator to manage network connections between the CTI server and the clients.

The section named `STARTTLS options` allows the administrator to enable encrypted communications between the clients and xivo-ctid and specify the certificate and private keys to use.

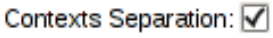
If no certificate and private key is configured, xivo-ctid will use the ones located in `/usr/share/xivo-certs`.



Parting options are used to isolate Wazo users from each other. These options should be used when using the same Wazo for different enterprises.

Context separation is based on the user's line context. A user with no line is not the member of any context and will not be able to do anything with the CTI client.

Note: xivo-dird must be restarted to take into account this parameter.



Authentication

xivo-ctid uses wazo-auth to authenticate users. The default authentication backend is *xivo_user*. To change the authentication backend, add a configuration file in */etc/xivo-ctid/conf.d* with the following content:

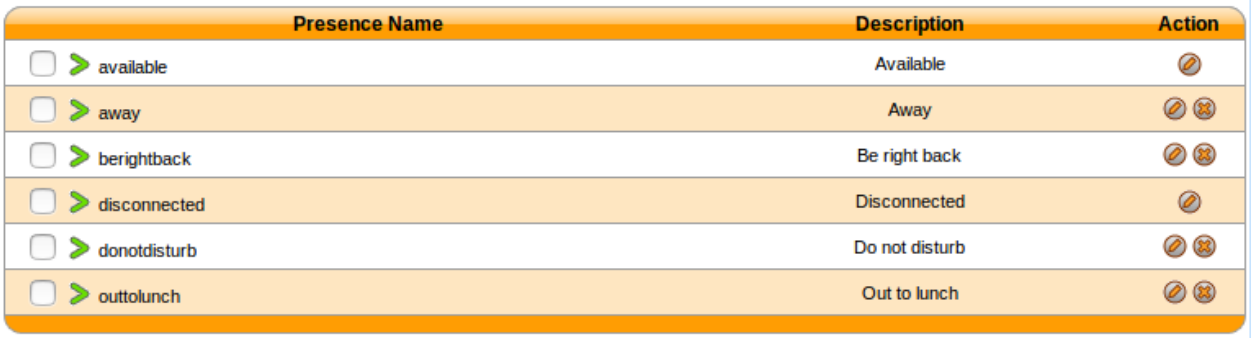
```
auth:
    backend: backend_name
```

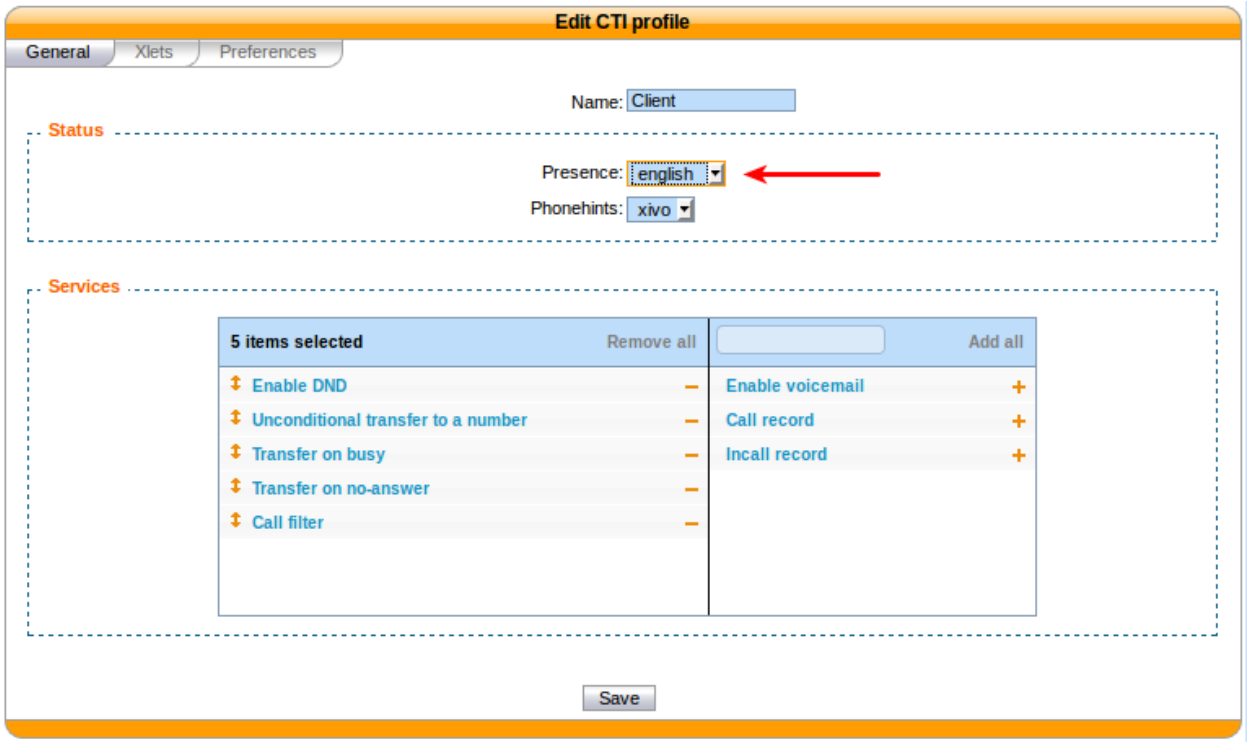
where *backend* name is the name of an enabled *wazo-auth* *Backends Plugins*.

Presence Option

In the *Status* menu, under *Presences*, you can edit presences group. The default presence group is *francais*. When editing a group, you will see a list of presences and their descriptions.

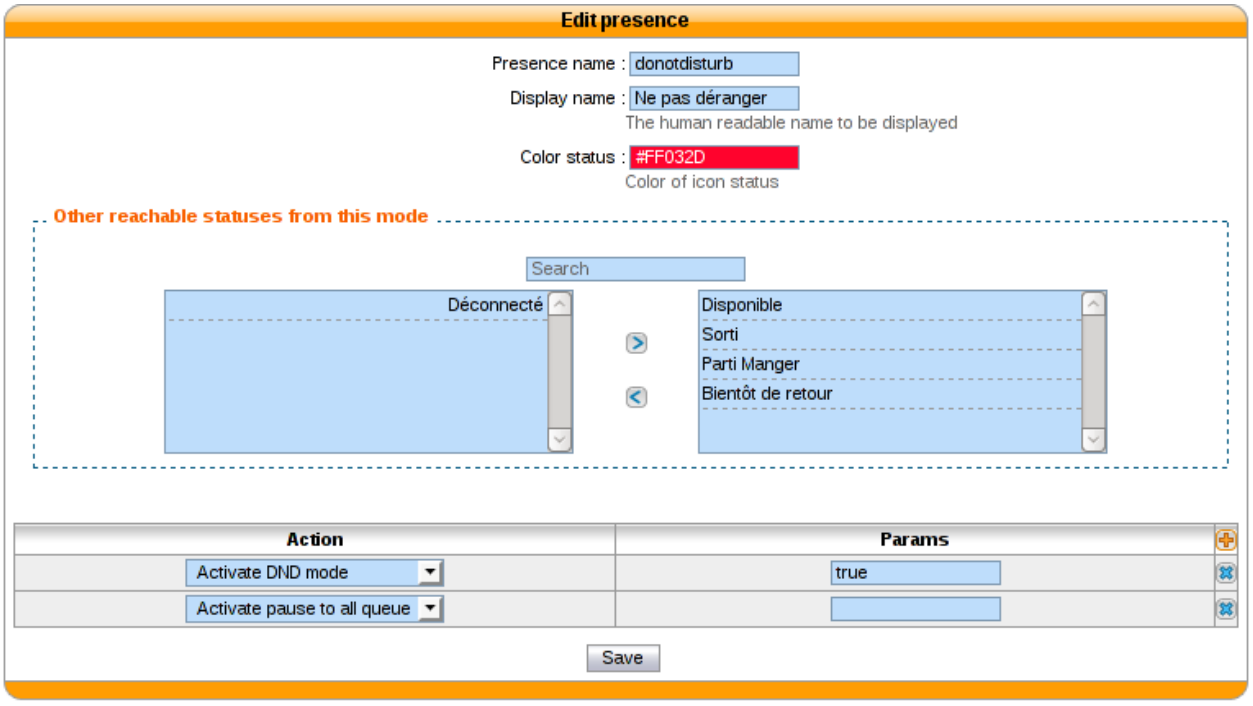
To use another presence group, you can edit the CTI profile you are using and select the appropriate presence group for that profile.


Presence Name
Description
Action
available
Available
away
Away
berightback
Be right back
disconnected
Disconnected
donotdisturb
Do not disturb
outtolunch
Out to lunch


Edit CTI profile
General
Xlets
Preferences
Name: Client
Status
Presence: english
Phonehints: xivo
Services
5 items selected
Remove all
Add all
Enable DND
Unconditional transfer to a number
Transfer on busy
Transfer on no-answer
Call filter
Enable voicemail
Call record
Incall record
Save

Available configuration

- *Presence name* is the name of the presence
- *Display name* is the human readable representation of this presence
- *Color status* is the color associated to this presence
- *Other reachable statuses* is the list of presence that can be switched from this presence state
- *Actions* are post selection actions that are triggered by selecting this presence



Actions

action	param
Enable DND	*{'true','false'}*
Pause agent in all queues	
Unpause agent in all queues	
Agent logoff	

Enable encryption

To enable encryption of CTI communications between server and clients, you have to enable STARTTLS in *CTI Server → General settings → General*

Custom certificates can be added in *Configuration → Certificates* and used in *CTI Server → General settings → General*

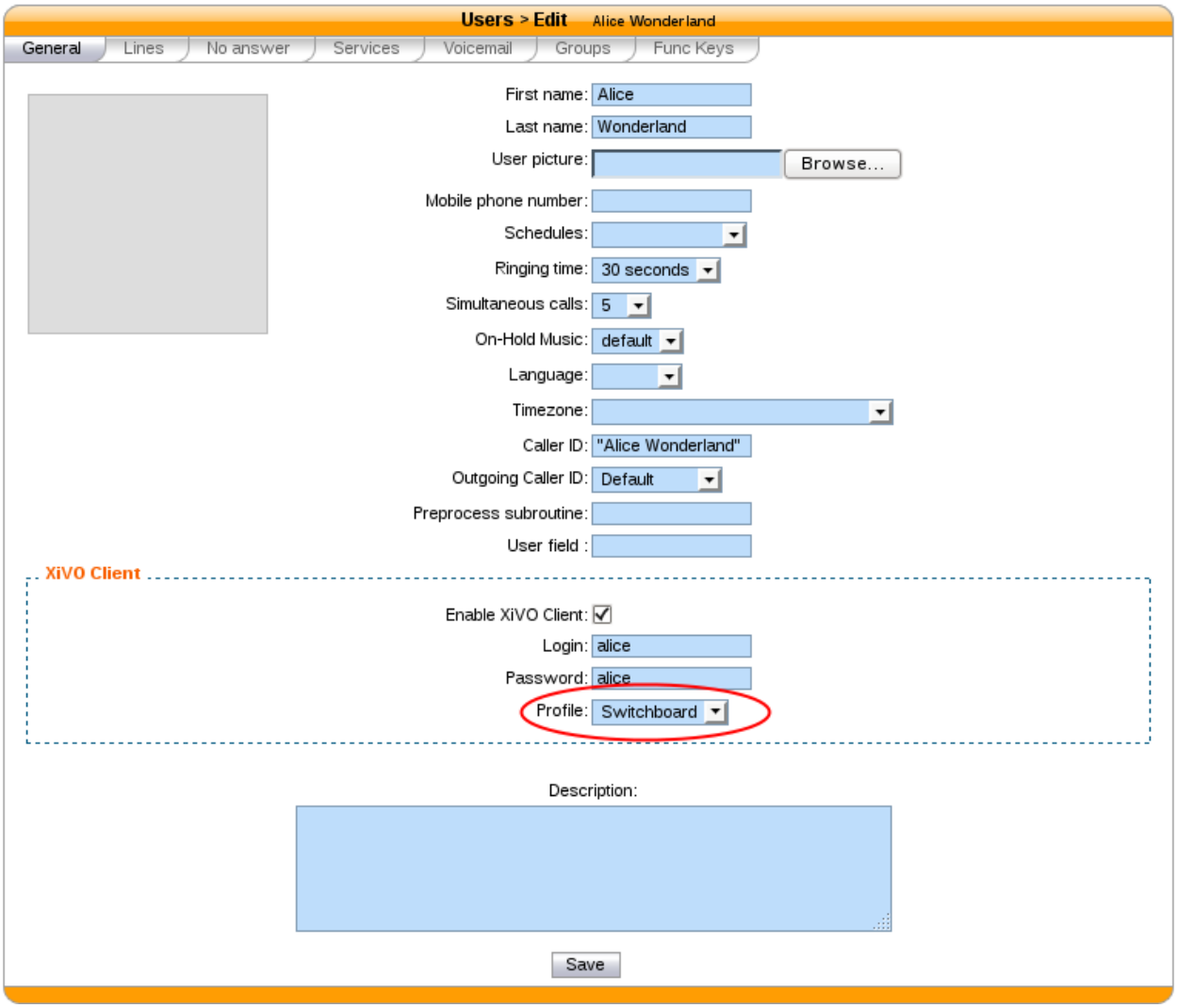
In your Wazo Client, in the menu *Wazo Client → Configure → Connection*, click on the lock icon.

Note: A client which chooses to use encryption will not be able to connect to a server that does not have STARTTLS enabled.

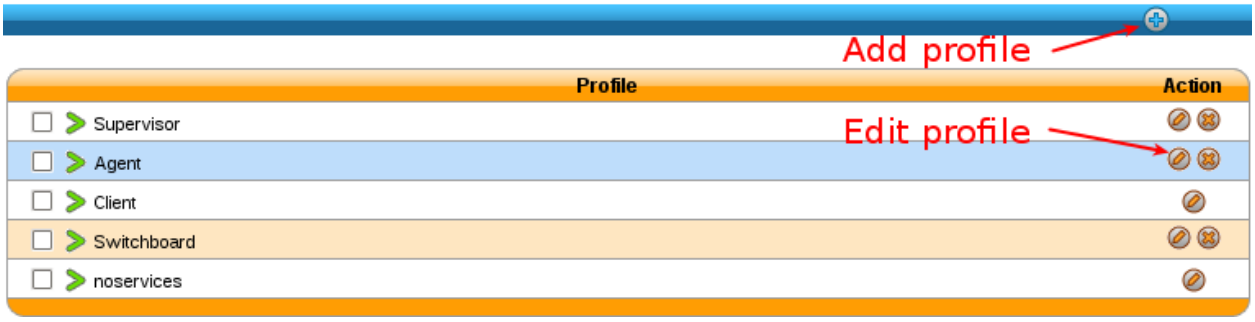
Warning: For now, there is no mechanism for strong authentication of the server. The connection is encrypted, but the identity of the server is not verified.

CTI profiles

The CTI profiles define which features are made available to a user. You can configure which profile will be used by a user in the menu *IPBX → PBX Settings → Users*:



You can also customize the default profiles or add new profiles in the menu *CTI Server → Profiles*:



Xlets

To choose which features are available to users using a profile, you have to select which *Xlets* will be available.

The Xlets are detailed in *Xlets*.

The *Position* attribute determines how the Xlets will be laid out:

- *dock* will display a Xlet in its own frame. This frame can have some options:
  - *Floating* means that the frame can be detached from the main window of the CTI Client.
  - *Closable* means that the Xlet can be hidden
  - *Movable* means that the Xlet can be moved (either inside the main window or outside)
  - *Scroll* means that the Xlet will display a scroll bar if the Xlet is too large.
- *grid* will display a Xlet inside the main window, and it will not be movable. Multiple *grid* Xlets will be laid out vertically (the second below the first).
- *tab* will display a Xlet inside a tab of the Xlet *Tabber*. Thus the Xlet *Tabber* is required and can't be in a *tab* position.

The *Number* attribute gives the order of the Xlets, beginning with 0. The order applies only to Xlets having the same *Position* attribute.

1.9.11 Customer Informations

Display customer informations

Sheet Configuration

Sheets can be defined under *Services* → *CTI Server* → *Models* in the web interface. Once a sheet is defined, it has to be assigned to an event in the *Services* → *CTI Server* → *Events* menu.

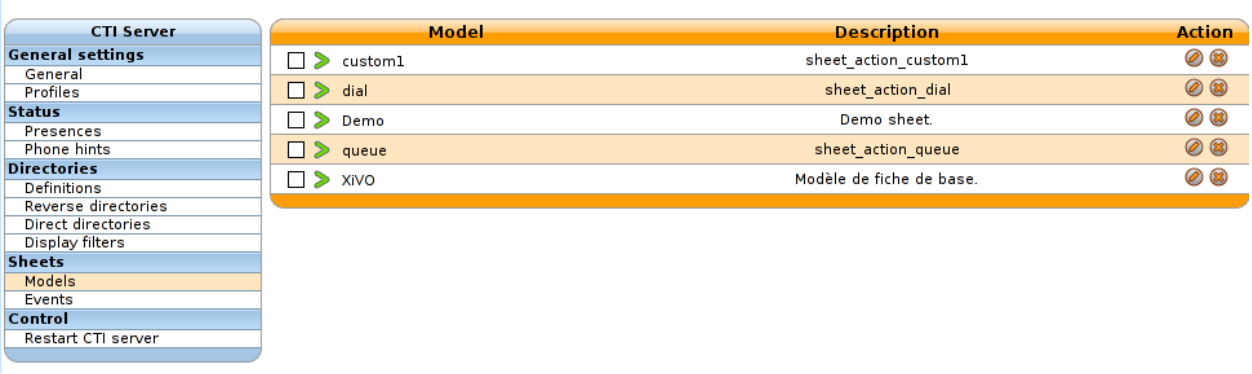
Model The model contains the content of the displayed sheet.

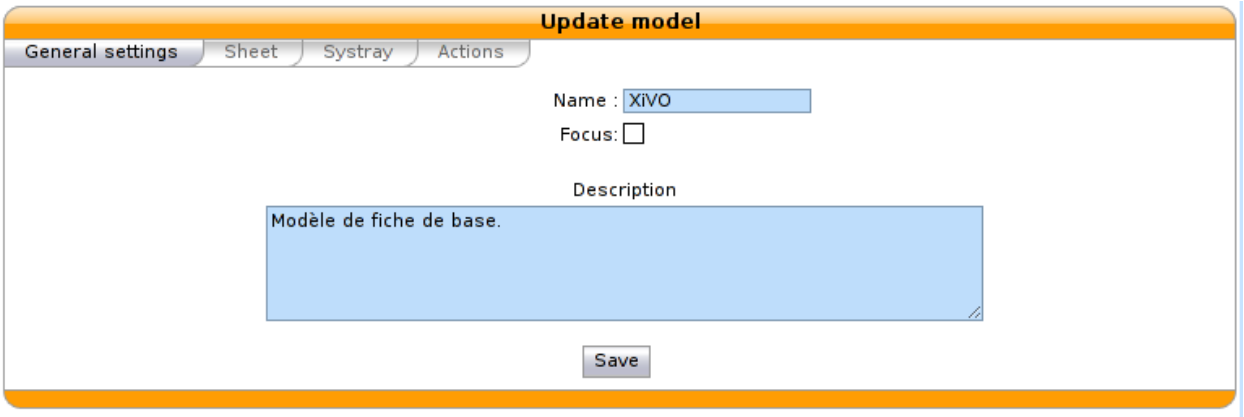
Event Events are actions that trigger the defined sheet. A sheet can be assigned to many events. In that case, the sheet will be raised for each event.

General settings

You must give a name to your sheet to be able to select it later.

The `Focus` checkbox makes the Wazo Client pop up when the sheet is displayed, if the Wazo Client was hidden.





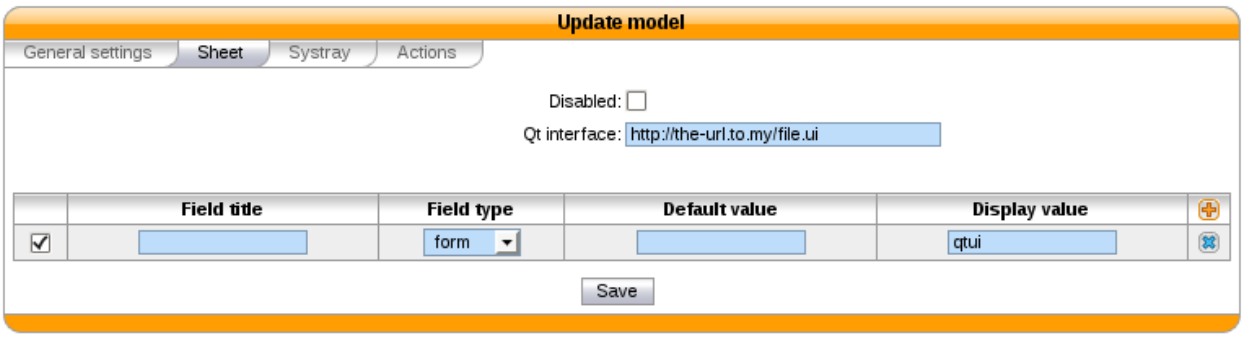
Sheets

There are two different ways to configure the contents of the sheet:

- creating a custom sheet from the Qt designer. This gives you a total control on the layout of the information and allows you to save and process data entered during or after a call.
- listing the different fields and their content. The information will be automatically laid out in a linear fashion and will be read-only.

Custom sheet

Configuring the sheet



The `Qt interface` field is the path to the UI file created by the Qt Designer. The path can either be a local file on your Wazo starting with `file://`, or a HTTP URL. When using a local file, make sure that the system user `xivo-ctid` has read access to the file.

You must add a field with type `form` and display value `qtui` for the form to be displayed.

Create a custom sheet with Qt Designer

The Qt Designer is part of the Qt development kit and is also available in the Qt Creator. They are available on the Qt project website.

Here is an example of a small form created with Qt Designer.



The Qt Designer screenshot.

Warning: In Qt Designer, one must set 'vertical layout' on the top widget (right click on the top widget > Lay out > Vertical layout).

You can download the file generated by this example from Qt Designer: `example-form.ui`

Text fields (QLineEdit, QLabel, QPlainTextEdit) can contain variables that will be substituted. See the *variable list* for more information.

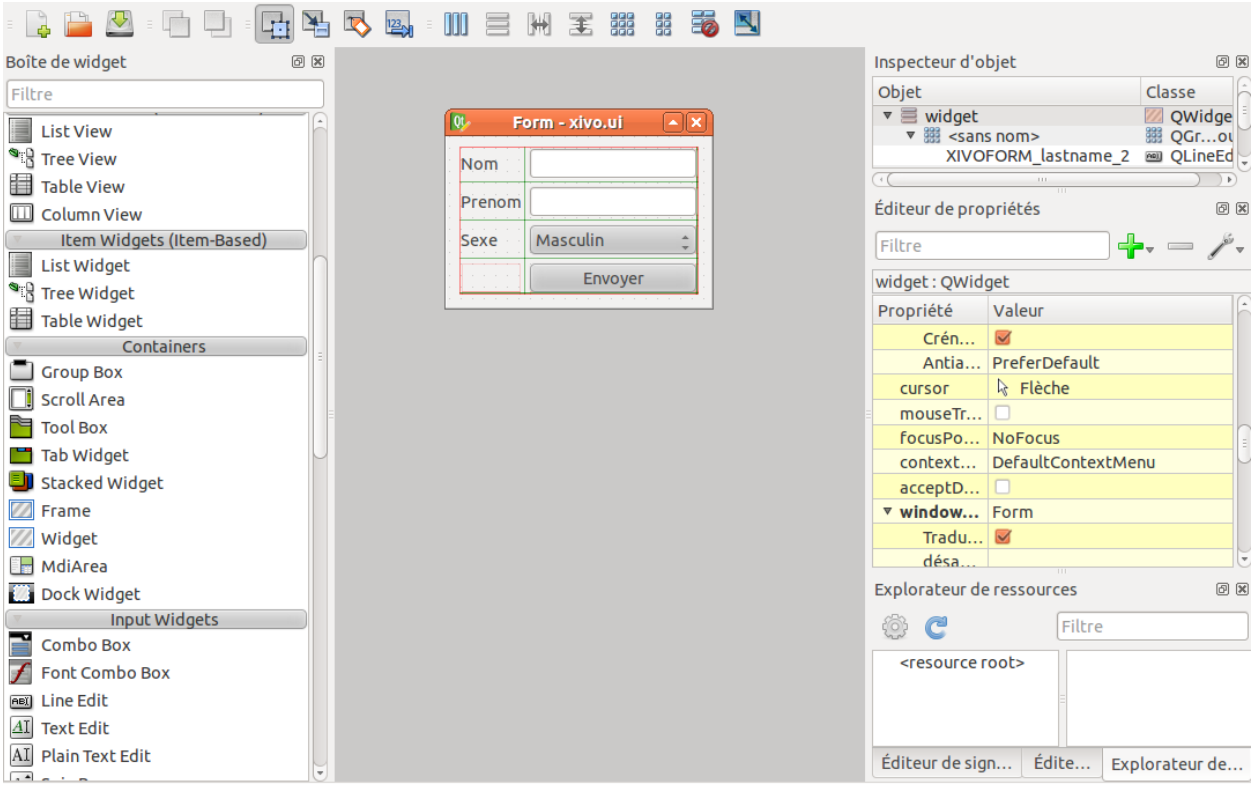
List of fields

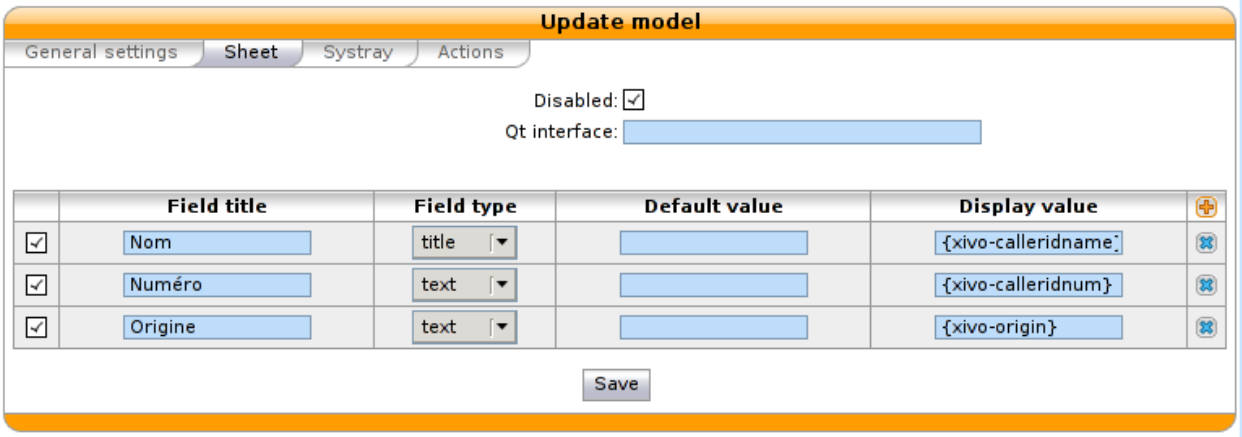
Default Wazo sheet example :

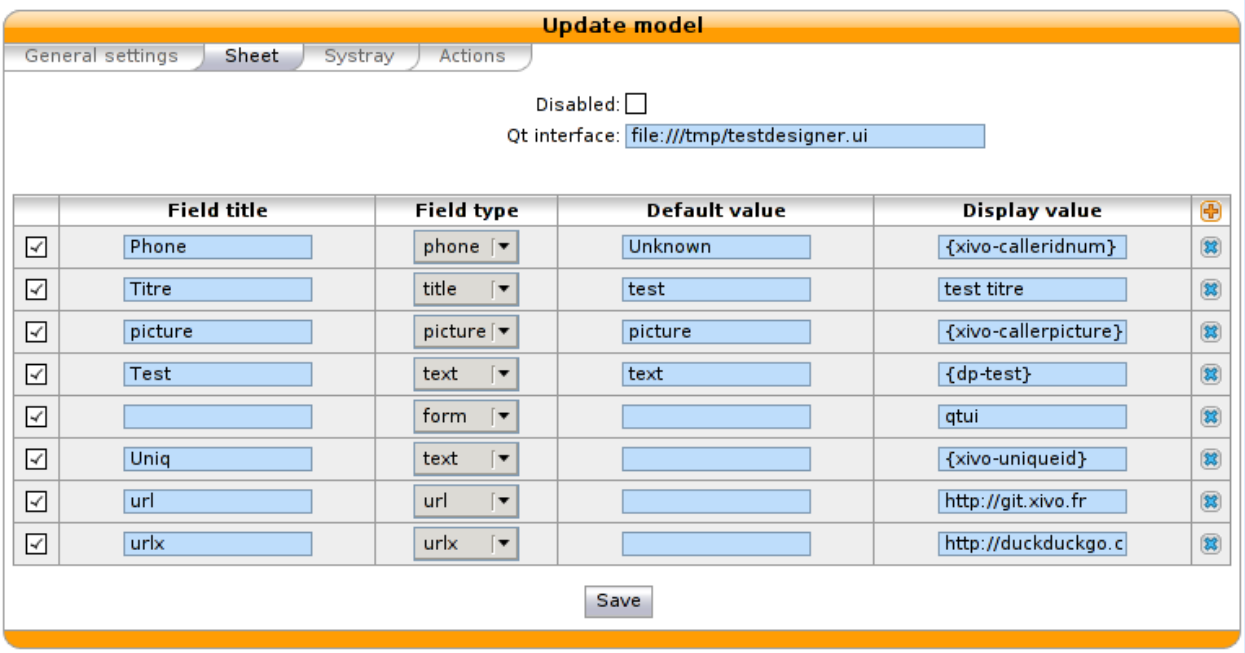
Example showing all kinds of fields:

Each field is represented by the following parameters :

- Field title : name of your line used as label on the sheet.
- Field type : define the type of field displayed on the sheet. Supported field types :
  - title : to create a title on your sheet
  - text : show a text


Boîte de widget
Filtre
List View
Tree View
Table View
Column View
Item Widgets (Item-Based)
List Widget
Tree Widget
Table Widget
Containers
Group Box
Scroll Area
Tool Box
Tab Widget
Stacked Widget
Frame
Widget
MdiArea
Dock Widget
Input Widgets
Combo Box
Font Combo Box
Line Edit
Text Edit
Plain Text Edit
Form - xivo.ui
Nom
Prenom
Sexe
Masculin
Envoyer
Inspecteur d'objet
Objet
Classe
widget
QWidge
<sans nom>
XIVOFORM_lastname_2
QLineEd
Éditeur de propriétés
Filtre
widget : QWidget
Propriété
Valeur
Crén...
Antia...
PreferDefault
cursor
Flèche
mouseTr...
focusPo...
NoFocus
context...
DefaultContextMenu
acceptD...
window...
Form
Tradu...
Explorateur de ressources
Filtre
<resource root>
Éditeur de sign...
Édite...
Explorateur de...


Update model
General settings
Sheet
Systray
Actions
Disabled:
Qt interface:
Field title
Field type
Default value
Display value
Nom
title
{xivo-calleridname}
Numéro
text
{xivo-calleridnum}
Origine
text
{xivo-origin}
Save



  - url : a simple url link, open your default browser.
  - urlx : an url button
  - phone : create a tel: link, you can click to call on your sheet.
  - form : show the form from an ui predefined. It's an xml ui. You need to define qtui in display format.
- Default value : if given, this value will be used when all substitutions in the display value field fail.
- Display value : you can define text, variables or both. See the *variable list* for more information.

Variables

Three kinds of variables are available :

- *xivo-* prefix is reserved and set inside the CTI server:
  - *xivo-where* for sheet events, event triggering the sheet
  - *xivo-origin* place from where the lookup is requested (did, internal, forcelookup)
  - *xivo-direction* incoming or internal
  - *xivo-did* DID number
  - *xivo-calleridnum*
  - *xivo-calleridname*
  - *xivo-calleridrdnis* contains information whether there was a transfer
  - *xivo-calleridton* Type Of Network (national, international)
  - *xivo-calledidnum*
  - *xivo-calledidname*
  - *xivo-ipbxid* (*xivo-astid* in 1.1)

  - *xivo-directory* : for directory requests, it is the directory database the item has been found
  - *xivo-queuename* queue called
  - *xivo-agentnumber* agent number called
  - *xivo-date* formatted date string
  - *xivo-time* formatted time string, when the sheet was triggered
  - *xivo-channel* asterisk channel value (for advanced users)
  - *xivo-uniqueid* asterisk uniqueid value (for advanced users)
- *db-* prefixed variables are defined when the reverse lookup returns a result.

For example if you want to access to the reverse lookup full name, you need to define a field `fullname` in the directory definition, mapping to the full name field in your directory. The `{db-fullname}` will be replaced by the caller full name. Every field of the directory is accessible this way.

- *dp-* prefixed ones are the variables set through the dialplan (through UserEvent application)

For example if you want to access from the dialplan to a variable dp-test you need to add in your dialplan this line (in a subroutine):

```
UserEvent(dialplan2cti,UNIQUEID: ${UNIQUEID},CHANNEL: ${CHANNEL},VARIABLE:␣
↪test,VALUE: "Salut")
```

The `{dp-test}` displays Salut.

Sending informations during/after a call

After showing a sheet, the Wazo Client can also send back information to Wazo for post-processing or archiving.

Here are the requirements:

- The sheet must contain a button named `save` to submit information
- Supported widgets:
  - QCalendarWidget
  - QCheckBox
  - QComboBox
  - QDateEdit
  - QDateTime
  - QDateTimeEdit
  - QDoubleSpinBox
  - QLabel
  - QLineEdit
  - QList
  - QPlainTextEdit
  - QRadioButton
  - QSpinBox
  - QTimeEdit

- Fields must have their name starting with `XIVOFORM_`

If you want to send information that is not visible, you can make the widget invisible on the sheet:

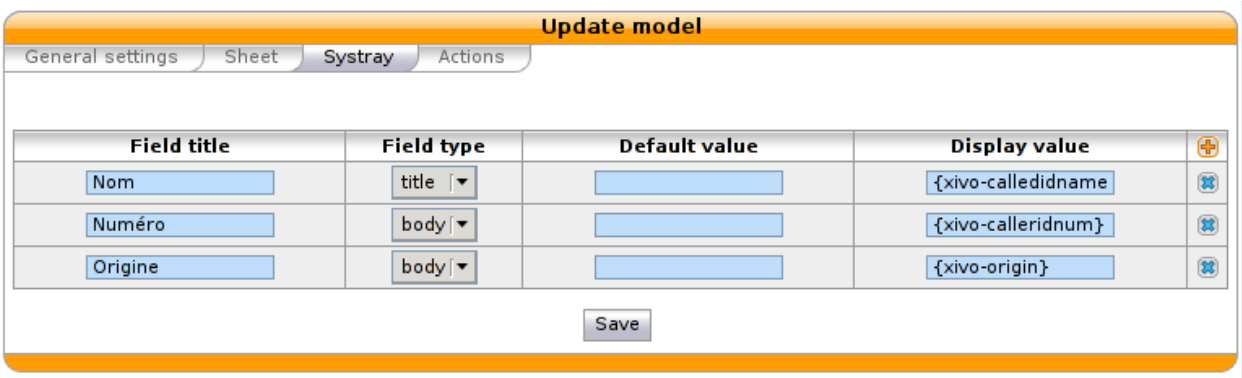
- change the maximumWidth or maximumHeight property to 0
- edit the `.ui` file and add the following property to the widget:

```
<property name="visible">
  <bool>false</bool>
</property>
```

When a CTI client submits a custom sheet, a *call_form_result* event is published on the event bus.

Systray

Mostly the same syntax as the sheet with less field types available (title, body). A Systray popup will display a single title (the last one added to the list of fields) and zero, one or more fields of type 'body'.



Warning: The popup message on MacOSX works with Growl http://growl.info. We could get simple sheet popup to work using the free Growl Fork http://www.macupdate.com/app/mac/41038/growl-fork Note that this is not officially supported.

Actions

The action is for the Wazo Client, so if you configure an action, please be sure you understand it's executed *by the client*. You need to allow this action in the client configuration too (menu *Wazo Client -> Configure*, tab *Functions*, tick option *Customer Info* and in sub-tab *Customer Info* tick the option *Allow the Automatic Opening of URL*).

The field in this tab receives the URL that will be displayed in your browser. You can also use variable substitution in this field.

- `http://example.org/foo` opens the URL on the default browser
- `http://example.org/{xivo-did}` opens the URL on the default browser, after substituting the `{xivo-did}` variable. If the substitution fails, the URL will remain `http://example.org/{xivo-did}`, i.e. the curly brackets will still be present.
- `http://example.org/{xivo-did}?origin={xivo-origin}` opens the URL on the default browser, after substituting the variables. If at least one of the substitution is successful, the failing substitutions will be replaced by an empty string. For example, if `{xivo-origin}` is replaced by 'outcall' but `{xivo-did}` is not substituted, the resulting URL will be `http://example.org/?origin=outcall`

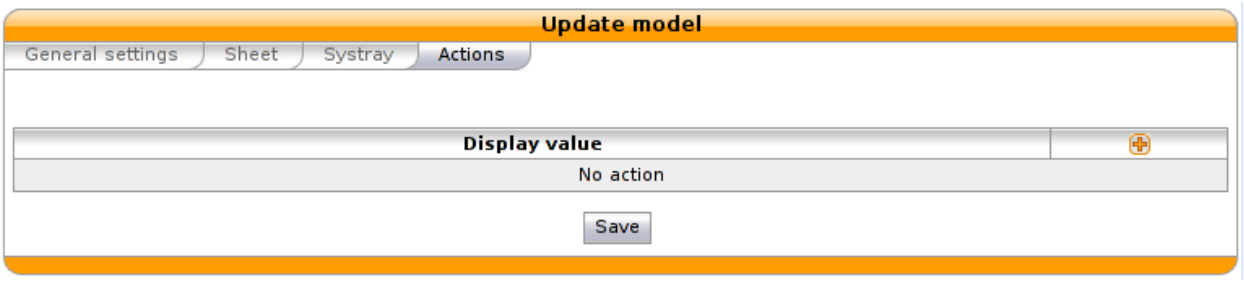
- `tcp://x.y.z.co.fr:4545/?var1=a1&var2=a2` connects to TCP port 4545 on x.y.z.co.fr, sends the string `var1=a1&var2=a2`, then closes
- `udp://x.y.z.co.fr:4545/?var1=a1&var2=a2` connects to UDP port 4545 on x.y.z.co.fr, sends the string `var1=a1&var2=a2`, then closes

---

Note: any string that would not be understood as an URL will be handled like and URL it is a process to launch and will be executed as it is written

---

For *tcp://* and *udp://*, it is a requirement that the string between */* and *?* is empty. An extension could be to define other serialization methods, if needed.

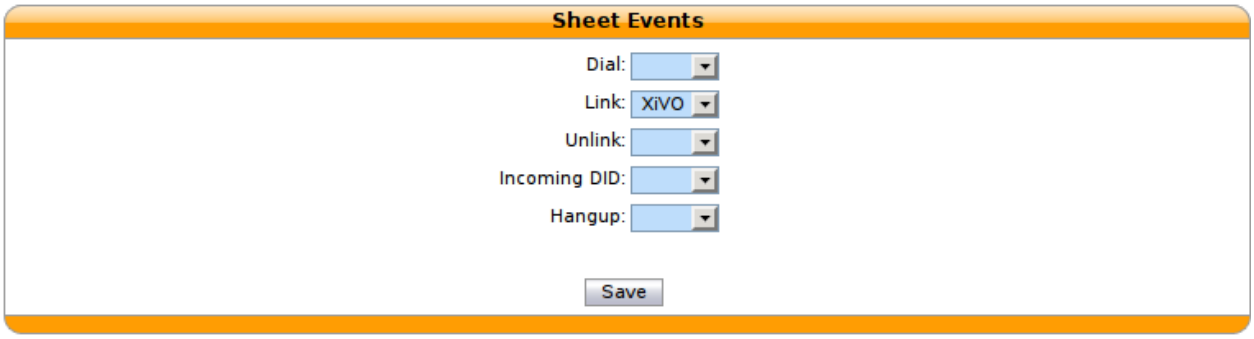


Event configuration

You can configure a sheet when a specific event is called. For example if you want to receive a sheet when an agent answers to a call, you can choose a sheet model for the Link event.

The following events are available :

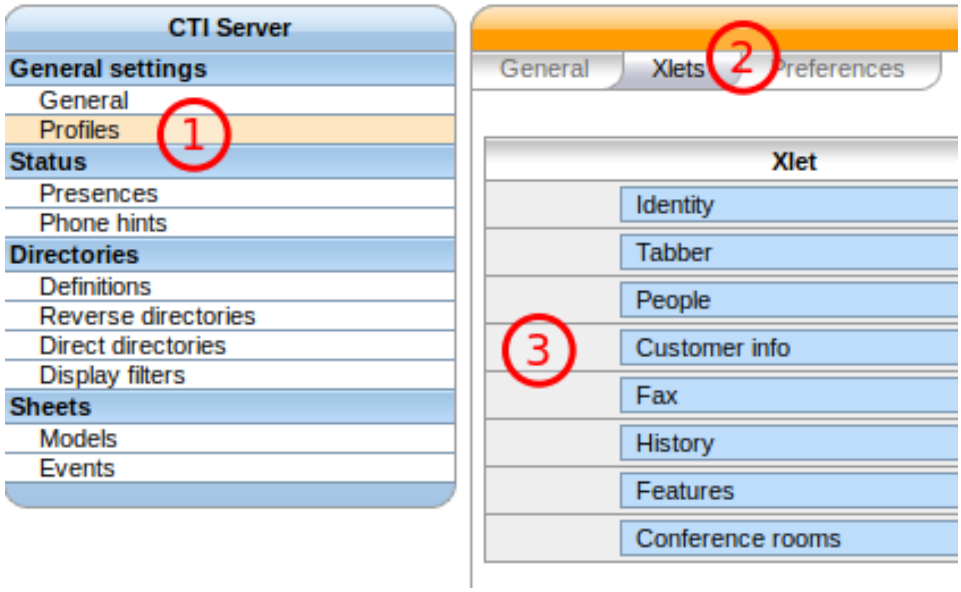
- Dial: When the member's phone starts ringing for calls on a group or queue or when the user receives a call
- Link: When a user or agent answers a call
- Unlink: When a user or agent hangup a call received from a queue
- Incoming DID: Received a call in a DID
- Hangup: Hangup the call



The informations about a call are displayed via the Wazo Client on forms called sheets. Sheets or customer info is a mechanism that allow the user to receive information about a call in its client. There are many events that can trigger a sheet to be displayed and there are many variables available for display.

Enabling the sheets for a user

To be able to receive sheets, a user must have a CTI profile with the *Customer Info* xlet.



The user must also enable *Screen Popup* in its client.

Example: Display a Web page when an agent answers a call

The first step is to assign the URL to a dialplan variable. Go in the *Services* → *IPBX* → *Configuration files* and create a new file called `setsheeturl.conf`. In this file, put the following:

```
[setsheeturl]
exten = s,1,NoOp(Starting Set Sheet URL)
same  = n,Set(SHEET_URL_CTI=http://documentation.wazo.community)
same  = n,UserEvent(dialplan2cti,UNIQUEID: ${UNIQUEID},CHANNEL: ${CHANNEL},VARIABLE:␣
↪mysheeturl,VALUE: ${SHEET_URL_CTI})
same  = n,Return()
```

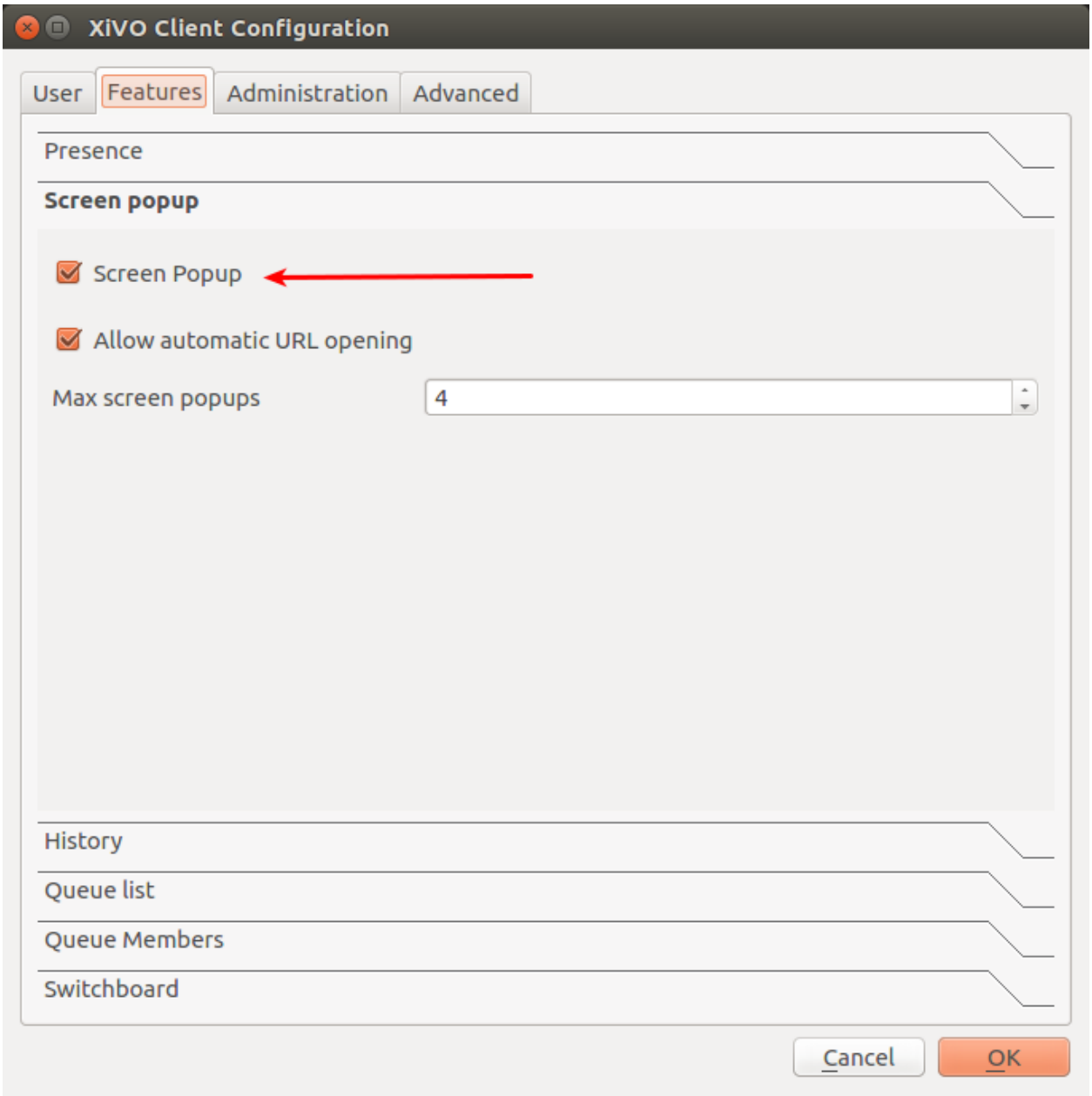
You can replace `documentation.wazo.community` by the URL you want.

The second step is to set the URL when the call is queued. To do that, we will use a preprocessing subroutine. This is configured in the queue configuration : go to *Services* → *Call center* → *Queues* and edit the queue. Set the field `Preprocessing subroutine` to `setsheeturl` (the same as above).

The third step is to configure the sheet to open the wanted URL. Go to *Services* → *CTI Server* → *Sheets* → *Models* and create a new sheet. Keep the default for everything except the Action tab, add a field and set it to `{dp-mysheeturl}` (the same as above).

The fourth and final step is to trigger the sheet when the agent answers the queued call. Go to *Services* → *CTI Server* → *Sheets* → *Events* and link the event `Agent linked` to the sheet you just created.

That's it, you can assign agents to your queue, log the agents and make them answer calls with the Wazo Client opened, and your browser should open the specified URL.


XiVO Client Configuration
User
Features
Administration
Advanced
Presence
Screen popup
Screen Popup
Allow automatic URL opening
Max screen popups
4
History
Queue list
Queue Members
Switchboard
Cancel
OK

1.9.12 Devices

Synchronize a device

First you have to display the list of devices.



Fig. 1.39: Click on the synchronize button for a device.

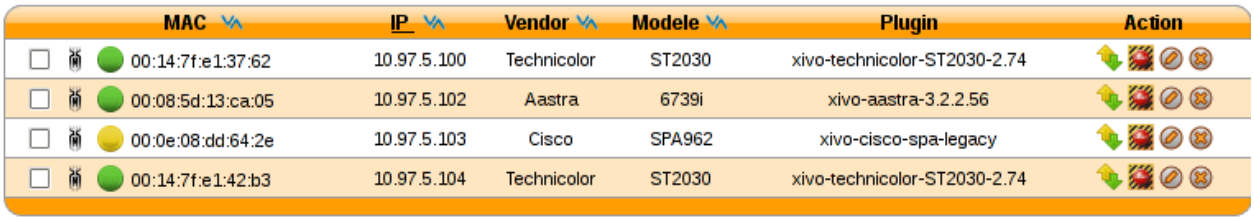


Fig. 1.40: List devices

You will see a pop-up to confirm synchronization Click on the <ok> button.

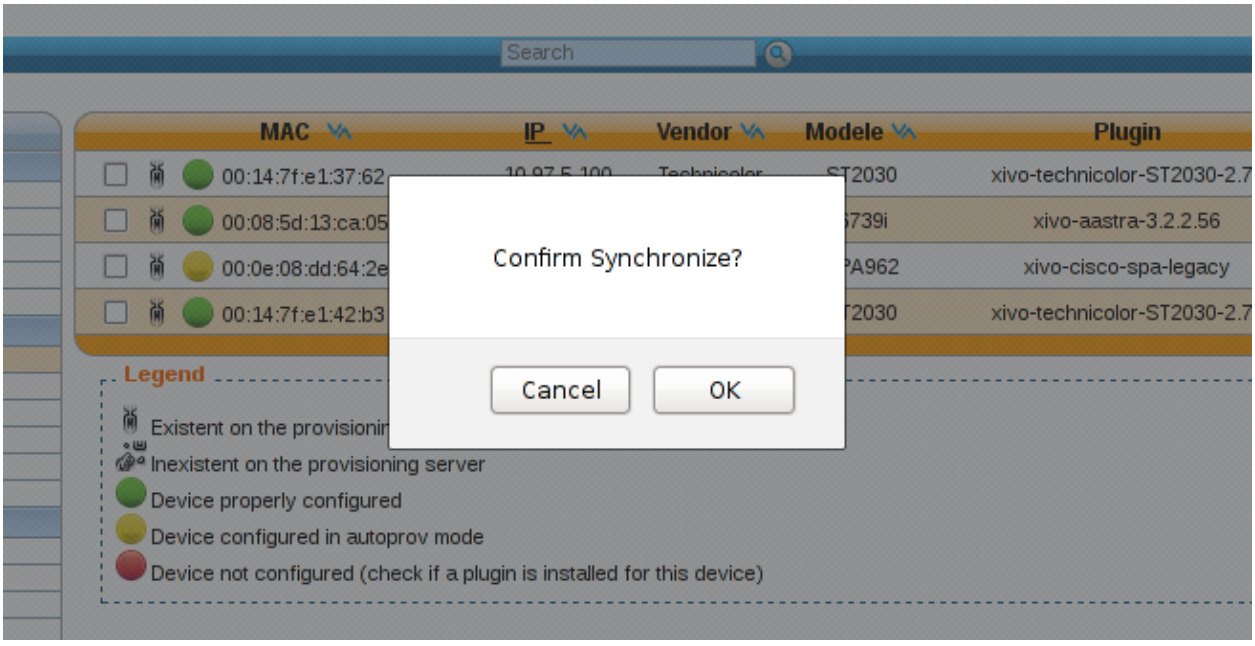


Fig. 1.41: Alert confirm synchronize

You must wait until the full synchronization process has completed to determine the state returned back from the device. This can take several seconds. It is important to wait and do nothing during this time.

If synchronization is successful, a green information balloon notifies you of success.

If synchronization fails, a red information balloon warns you of failure.

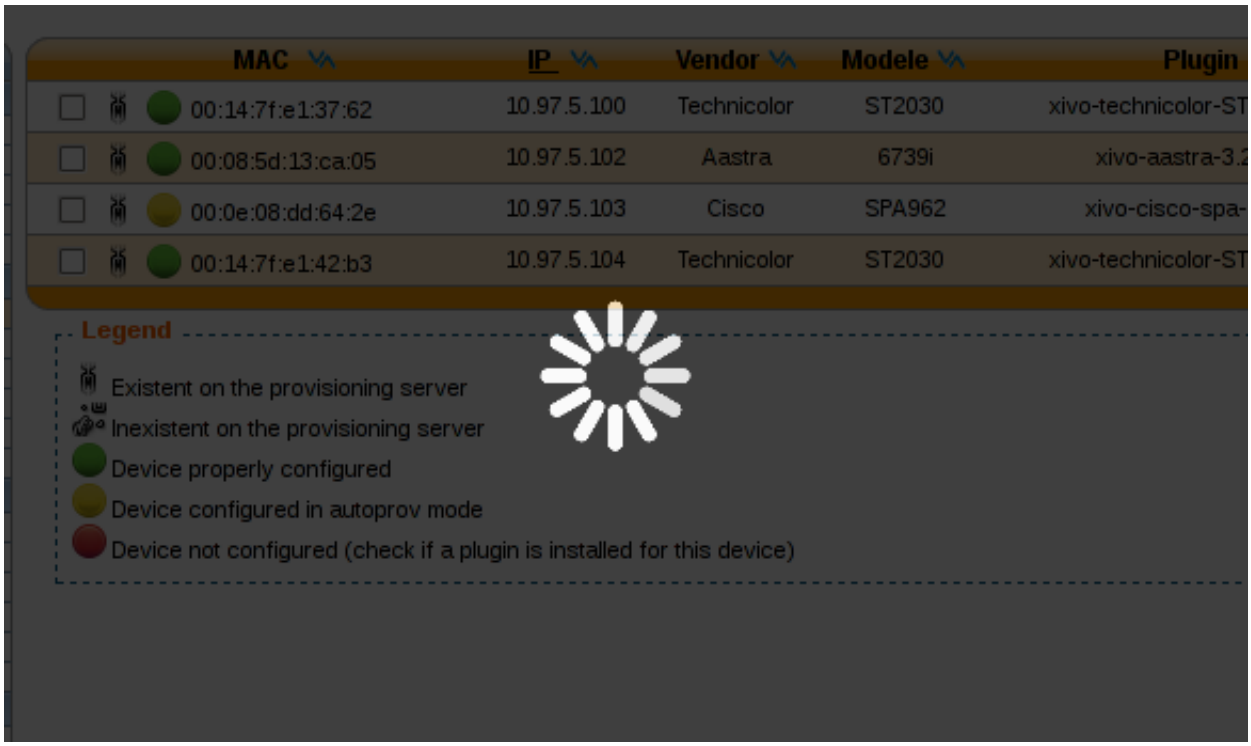


Fig. 1.42: Request synchronization processing

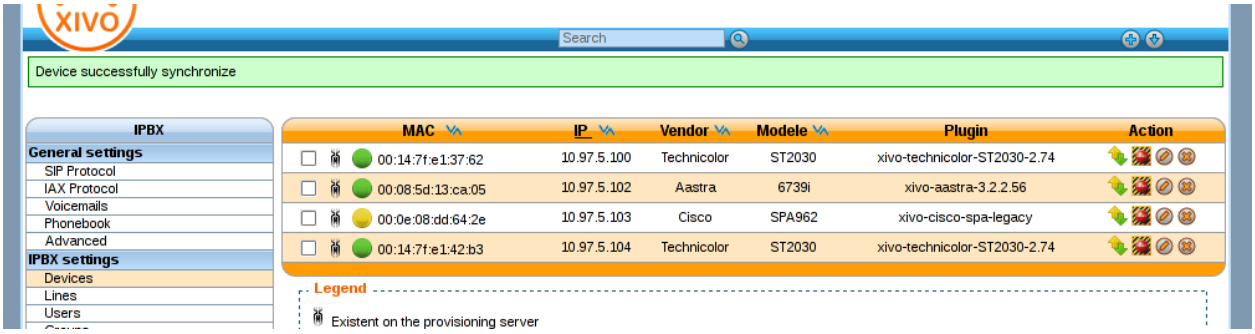


Fig. 1.43: Device successfully synchronized

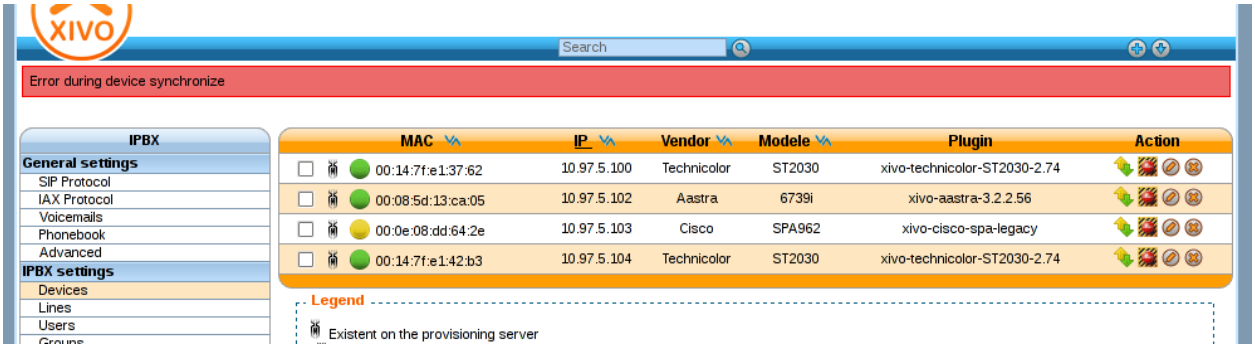


Fig. 1.44: Error during device synchronization

Synchronize multiple devices

Warning: When using multiple synchronization, the individual return states will not be displayed.

Select the devices you want to synchronize by checking the boxes.

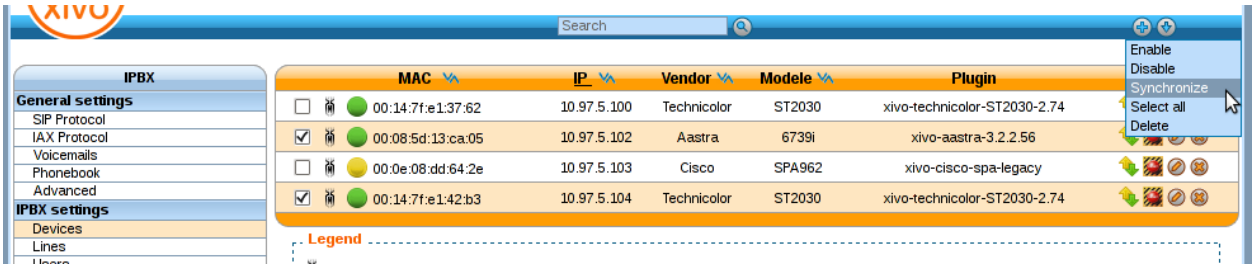


Fig. 1.45: Synchronize selected devices

A pop-up will appear requesting confirmation.

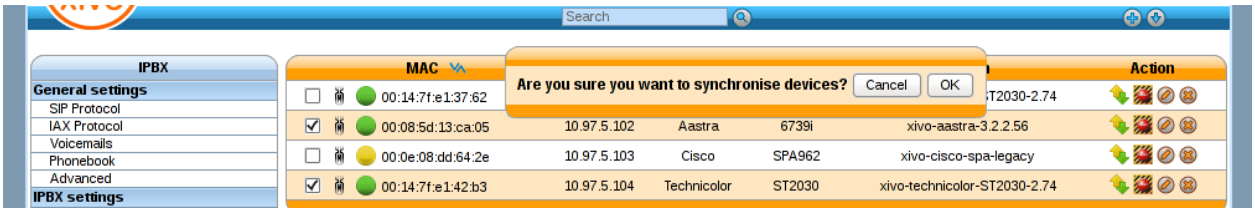


Fig. 1.46: Synchronize selected devices confirmation

If mass synchronization was successfully sent to the devices, a green information balloon notifies you of success.

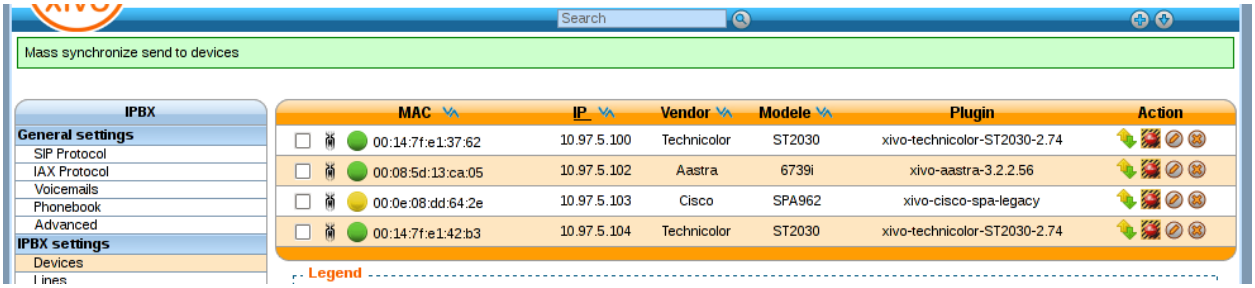


Fig. 1.47: Mass synchronization request sent successfully

1.9.13 Directories

This page documents how to add and configure directories from custom sources. Directories added from custom sources can be used for lookup via the *Wazo Client*, directory feature of phones or for *reverse lookup* on incoming calls.

An example of *adding a source* and *configuring source access* is made for each type of source:

XiVO directories

This type of directory is used to query the users of a XiVO. On a fresh install, the local XiVO is already configured. The URI field for this type of directory should be the base URL of a *xivo-confd* server.

This directory type matches the *xivo* backend in *xivo-dird*.

Available fields

- id
- agent_id
- line_id
- firstname
- lastname
- email
- exten
- context
- mobile_phone_number
- userfield
- description
- voicemail_number

Example

Adding a source

Configuring source access

Here is an example of a configuration where the userfield was used as a free field to store the DID number of the user and the description to store it's location.

CSV File directories

The source file of the directory must be in CSV format. You will be able to choose the headers and the separator in the next steps. For example, the file will look like:

```
title|firstname|lastname|displayname|society|mobilenumber|email
mr|Emmett|Brown|Brown Emmett|DMC|5555551234|emmet.brown@dmc.example.com
```

This directory type matches the *csv* backend in *xivo-dird*.

For file directories, the *Direct match* and the *Match reverse directories* must be filled with the name of the column used to match entries.

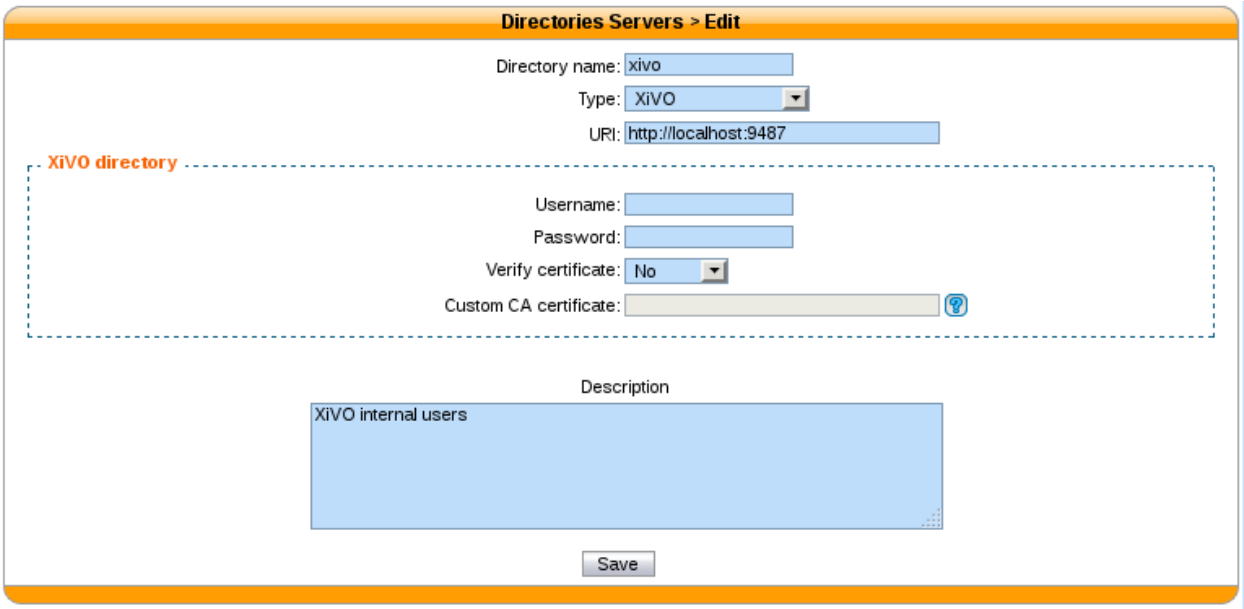


Fig. 1.48: *Configuration → Management → Directories*

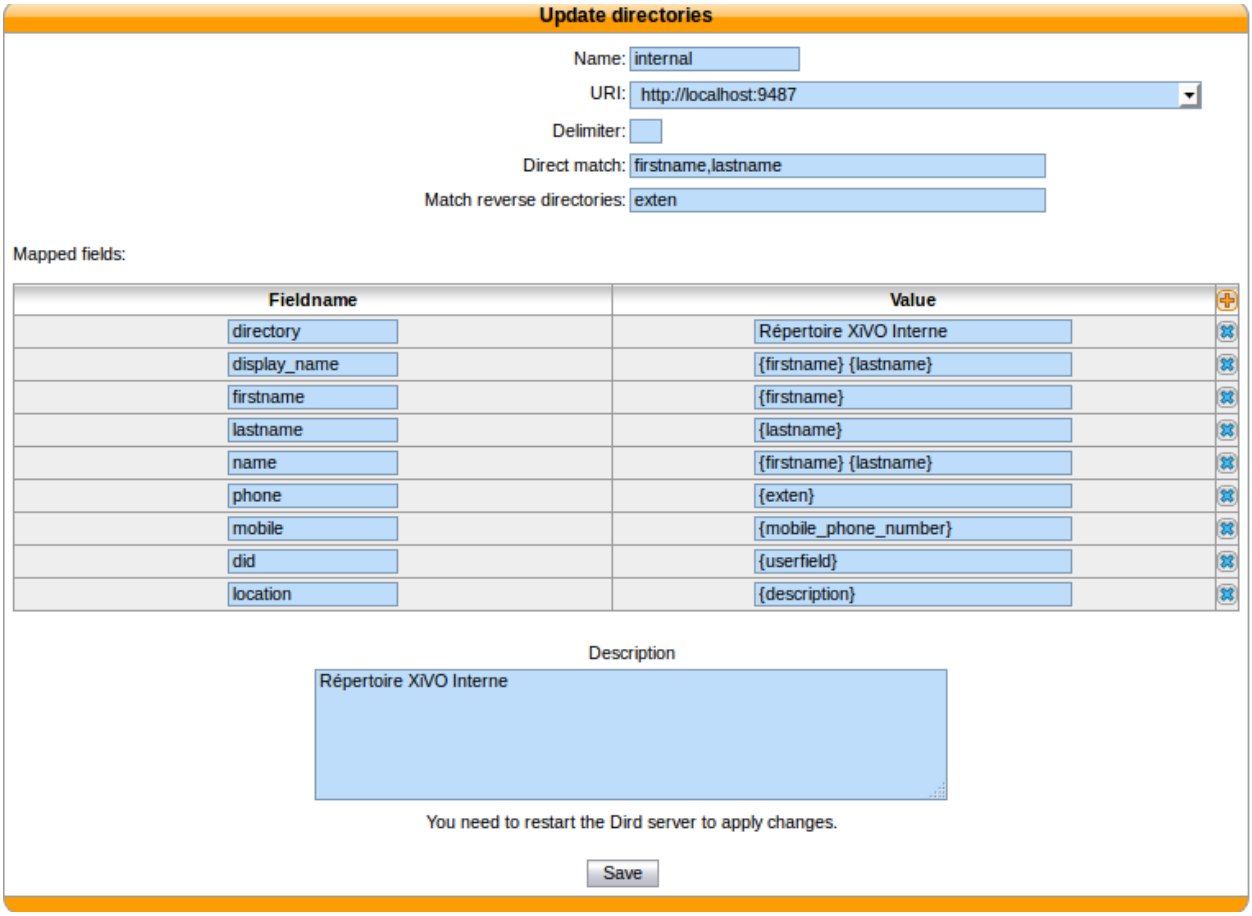


Fig. 1.49: *Services → CTI Server → Directories → Definitions*

Available fields

Available fields are the one's contained in the CSV file.

Example

csv-phonebook.csv:

```
title|firstname|lastname|displayname|society|phone|email
mr|Emmett|Brown|Brown Emmett|DMC|5555551234|emmet.brown@dmc.example.com
ms|Alice|Wonderland|Wonderland Alice|DMC|5555551235|alice.wonderland@dmc.example.com
```

Adding a source

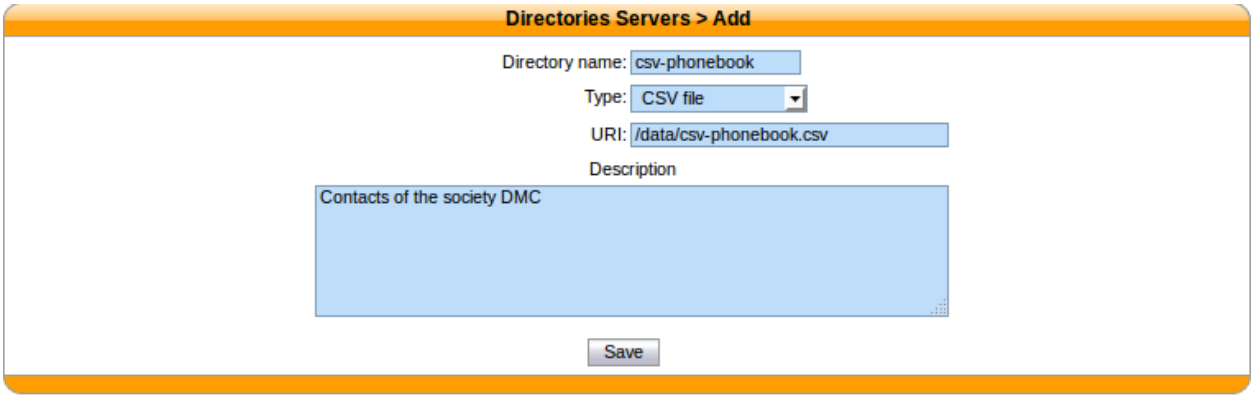


Fig. 1.50: *Configuration → Management → Directories*

Configuring source access

CSV Web service directories

The data returned by the Web service must have the same format than the file directory. In the same way, you will be able to choose the headers and the separator in the next step.

This directory type matches the *CSV web service* backend in *xivo-dird*.

For web service directories, the *Direct match* and the *Match reverse directories* must be filled with the name of the HTTP query parameter that will be used when doing the HTTP requests.

Note that the CSV returned by the Web service is not further processed.

Manual configuration needs to be done to use a secure (SSL) connection. See *CSV web service* for more details.

Available fields

Available fields are the ones contained in the CSV result.



Fig. 1.51: *Services → CTI Server → Directories → Definitions*

Example

http://example.org:8000/ws-phonebook return csv:

```
title|firstname|lastname|displayname|society|phone|email
mr|Emmett|Brown|Brown Emmett|DMC|5555551234|emmet.brown@dmc.example.com
ms|Alice|Wonderland|Wonderland Alice|DMC|5555551235|alice.wonderland@dmc.example.com
```

Adding a source

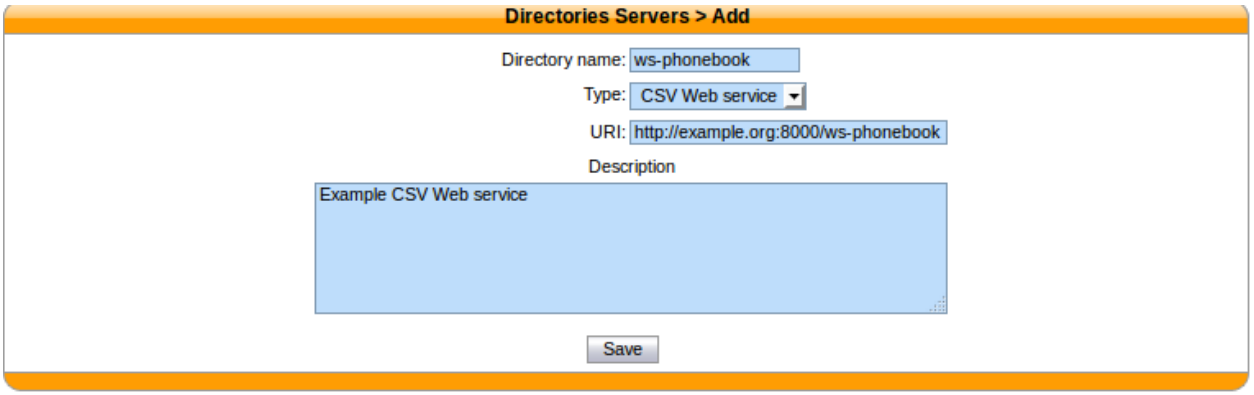


Fig. 1.52: *Configuration → Management → Directories*

Configuring source access

Given you have the following directory definition:

- *Direct match* : `search`
- *Match reverse directories* : `phone`

When a direct lookup for "Alice" is performed, then the following HTTP request:

```
GET /ws-phonebook?search=Alice HTTP/1.1
```

is emitted. When a reverse lookup for "5555551234" is performed, then the following HTTP request:

```
GET /ws-phonebook?phone=5555551234 HTTP/1.1
```

is emitted. On the reverse lookup, a filtering is performed on the result. In this example, it should have `phone` as column.

XiVO dird internal phonebook

This type of directory source is the internal phonebook of XiVO dird. The URI field is used to connect to the xivo-dird database.

This directory type matches the *dird_phonebook* backend in *xivo-dird*.



Fig. 1.53: *Services → CTI Server → Directories → Definitions*

Example

Adding a source

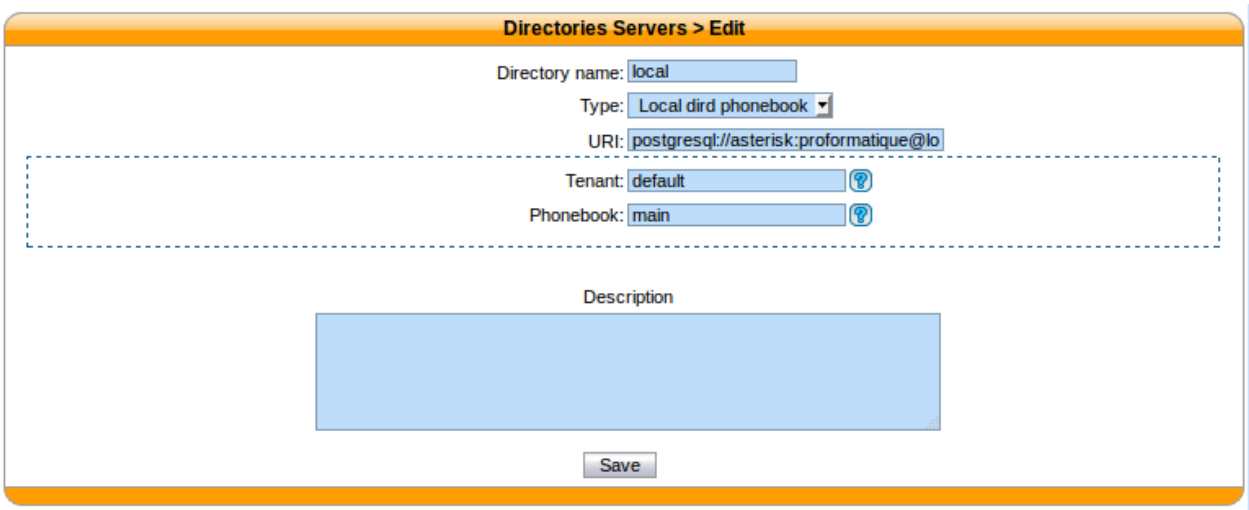


Fig. 1.54: *Configuration → Management → Directories*
URI : The URI to connect to the xivo-dird database
Tenant : Name of the tenant, the entity is used in the default configuration
Phonebook : Name of the phonebook to use

Configuring available fields

Adding the source to a profile

LDAP filter directory

This page describes how to configure Wazo to search a LDAP server from its directory service.

Adding a LDAP server

Note: SSL means TLS/SSL (doesn't mean StartTLS) and port 636 should then be used

---

Notes on SSL/TLS usage

If you are using SSL with an LDAP server that is using a CA certificate from an unknown certificate authority, you'll have to put the certificate file as a single file ending with `.crt` into `/usr/local/share/ca-certificates` and run `update-ca-certificates`.

You also need to make sure that the `/etc/ldap/ldap.conf` file contains a line `TLS_CACERT /etc/ssl/certs/ca-certificates.crt`.

After that, restart spawn-fcgi with `service spawn-fcgi restart`.

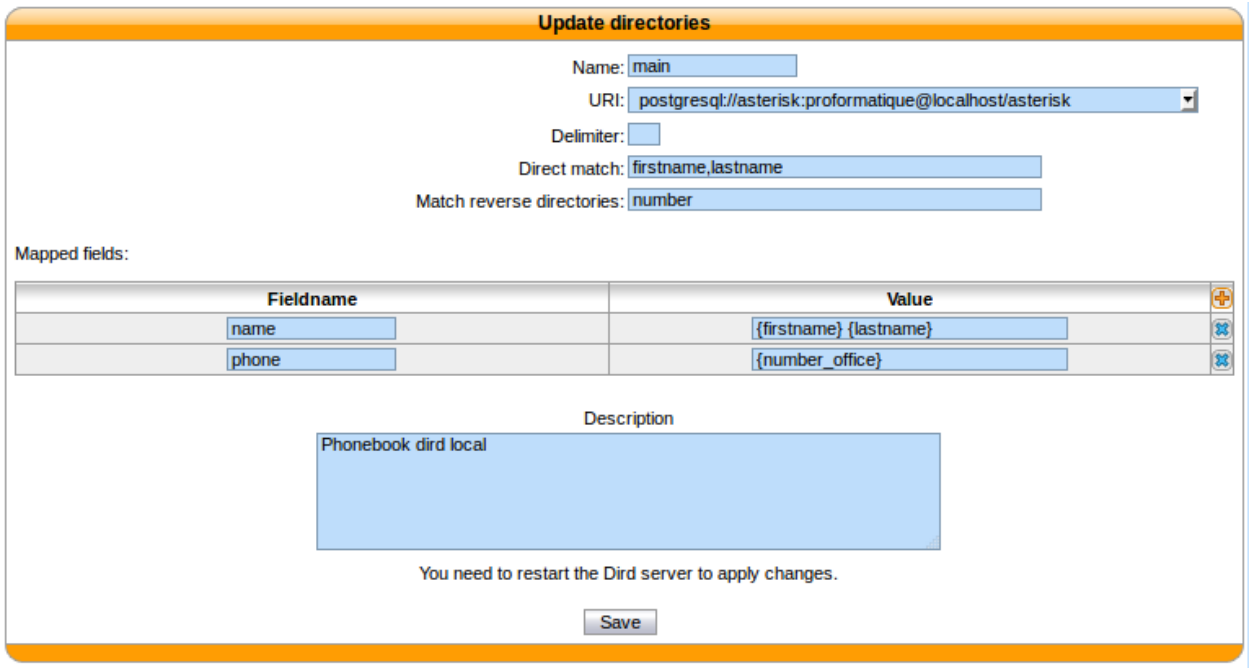


Fig. 1.55: *Services → CTI Server → Directories → Definitions*
Name : Name of this source
Direct match : Fields to match when doing a lookup
Match reverse directories : Fields to match when doing a reverse lookup
Mapped fields : Add fields to be compatible with a configured display

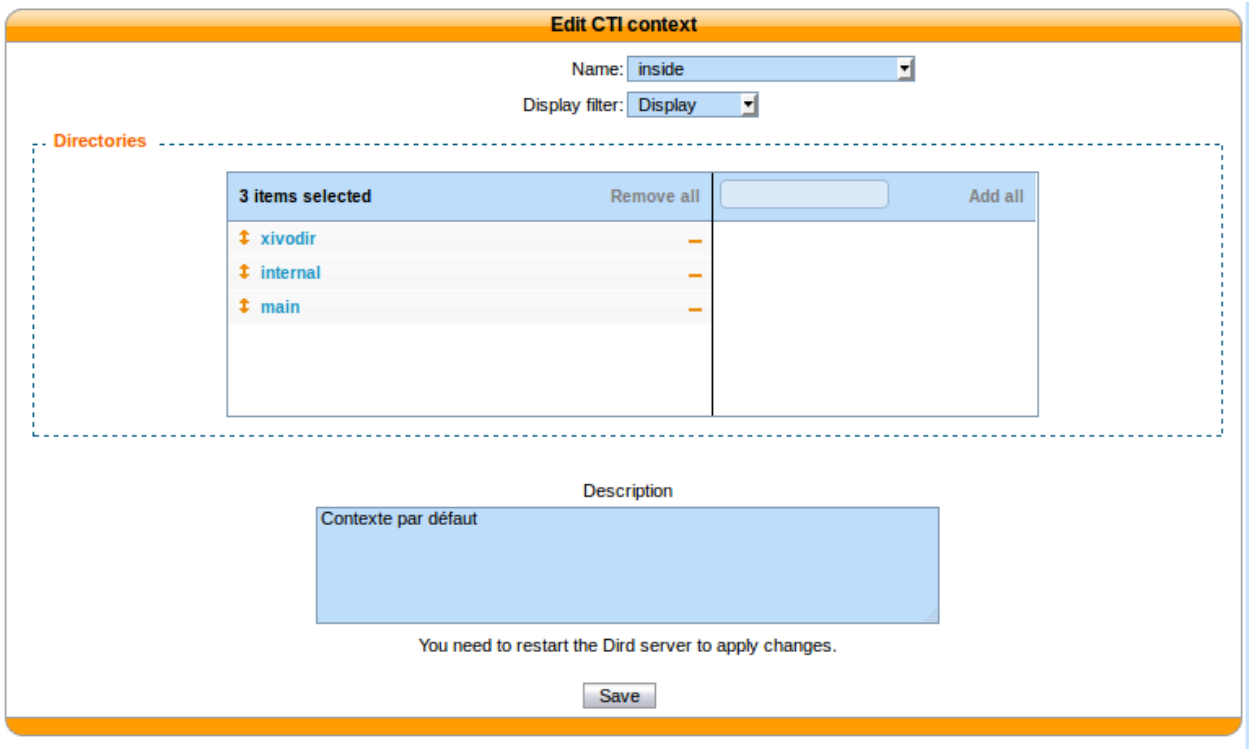


Fig. 1.56: *Services → CTI Server → Directories → Direct directories*
Directories : Add the new directory source to the profile

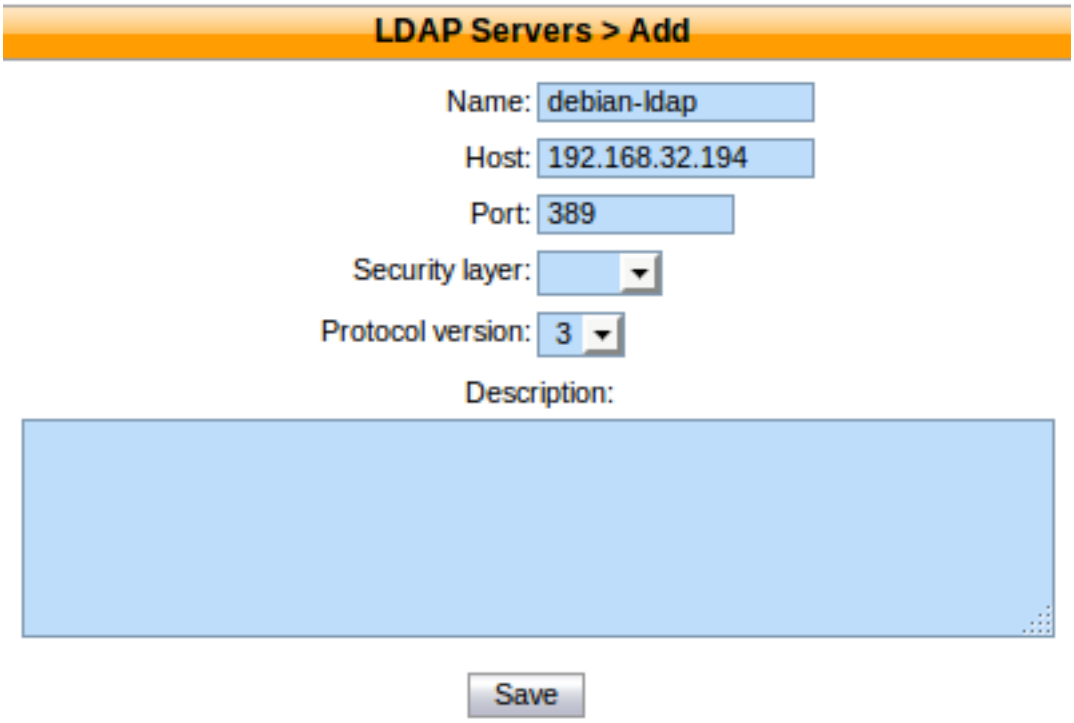


Fig. 1.57: *Configuration* → *Management* → *LDAP Servers*
Name: the server's display name
Host: the hostname or IP address
Port: the port number (default: 389)
Security layer: select SSL if it is activated on your server and you want to use it (default: disabled)
Protocol version: the LDAP protocol version (default: 3)

Also, make sure to use the FQDN of the server in the host field when using SSL. The host field must match exactly what's in the CN attribute of the server certificate.

Adding a LDAP Filter

Next thing to do after adding a LDAP server is to create a LDAP filter via the *Services → IPBX configuration → LDAP Filters* page.

You can add a LDAP filter by clicking on the add button at the top right of the page. You'll then be shown this page:

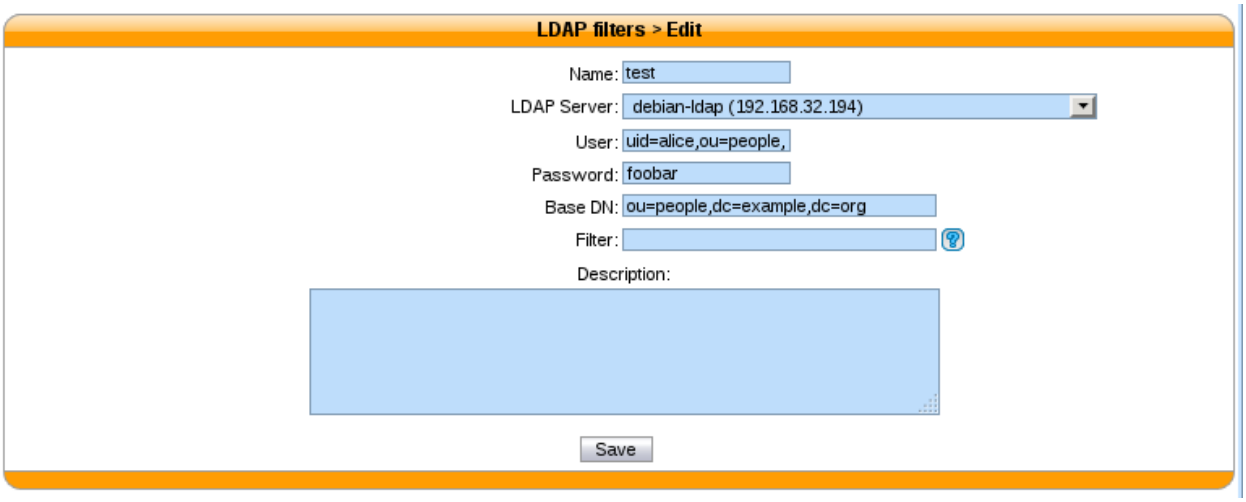


Fig. 1.58: *Services → IPBX configuration → LDAP Filters*
Name: the filter's display name
LDAP server: the LDAP server this filter applies to
User: the `dn` of the user used to do search requests
Password: the password of the given user
Base DN: the base `dn` of search requests
Filter: if specified, *it replace the default filter*

Use a Custom Filter

In some cases, you might have to use a custom filter for your search requests instead of the default filter.

In custom filters, occurrence of the pattern `%Q` is replaced by what the user entered on its phone.

Here's some examples of custom filters:

- `cn=*%Q*`
- `&(cn=*%Q*)(mail=*@example.org)`
- `|(cn=*%Q*)(displayName=*%Q*)`

Adding a source

Adding a Directory Definition

The next step is to add a directory defintion for the LDAP directory you just created. See the *directories* section for more information.

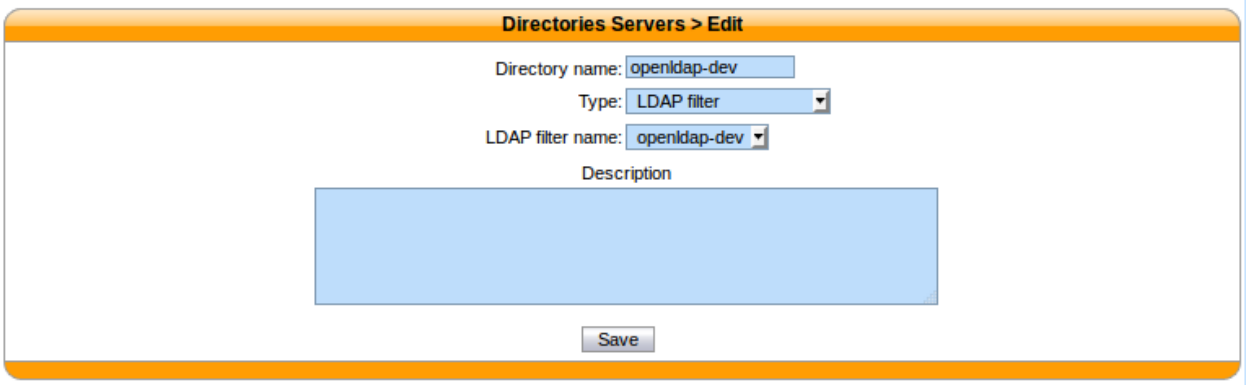


Fig. 1.59: *Configuration* → *Management* → *Directories*
LDAP filter name: The LDAP filter this directory should use.

Here's an example of an LDAP directory definition:

If a custom filter is defined in the LDAP filter configuration, the fields in *direct match* will be added to that filter using an *&*. To only use the *filter* field of your LDAP filter configuration, do not add any *direct match* fields in your directory definition.

Example:

- Given an LDAP filter with *filter* `st=Canada`
- Given a directory definition with a *direct match* `cn,o`
- Then the resulting filter when doing a search will be `&(st=Canada)(|(cn=*%Q*)(o=*%Q*))`

Note: Phone IP should be in the authorized subnet to access the directories. See *Remote directory*.

Adding a source

You can add new data sources via the *Configuration* → *Management* → *Directories* page.

- *Directory name*: the name of the directory
- *Type*: there are 4 types of directory:
  - *XiVO*
  - *CSV File*
  - *CSV Web service*
  - *XiVO dird phonebook*
  - *LDAP filter directory*
- *URI*: the data source
- *Description*: (optional) a description of the directory

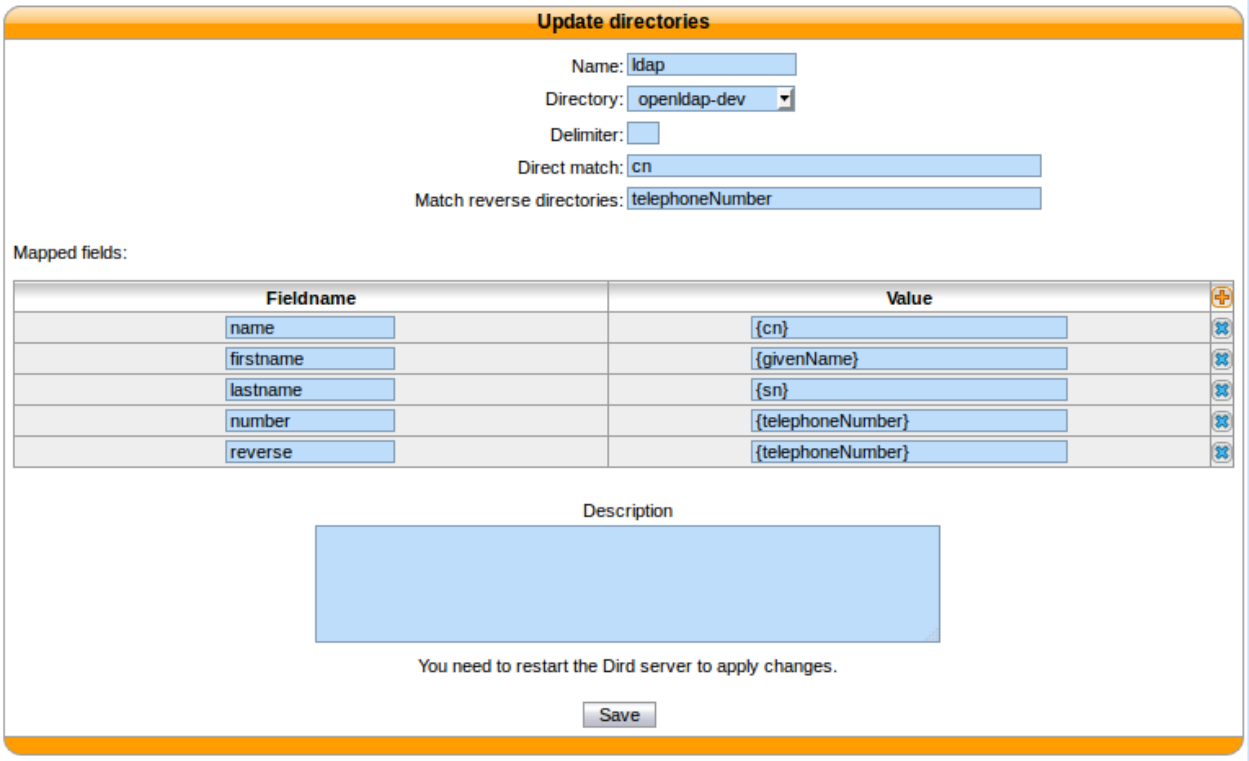


Fig. 1.60: *Services → IPBX → IPBX configuration → LDAP filters*

Configuring source access

Reverse lookup

It's possible to do reverse lookups on incoming calls to show a better caller ID name when the caller is in one of our directories.

Reverse lookup will only be tried if at least one of the following conditions is true:

- The caller ID name is the same as the caller ID number
- The caller ID name is "unknown"

Also, reverse lookup is performed after *caller ID number normalization* (since XiVO 13.11).

To enable reverse lookup, you need to add an entry in *Mapped fields*:

- *Fieldname*: `reverse`
- *Value*: the header of your data source that you want to see as the caller ID on your phone on incoming calls

Example

- *Match reverse directories*: `number_office,number_mobile`
- *Fieldname*: `reverse`
- *Value*: `{society}`

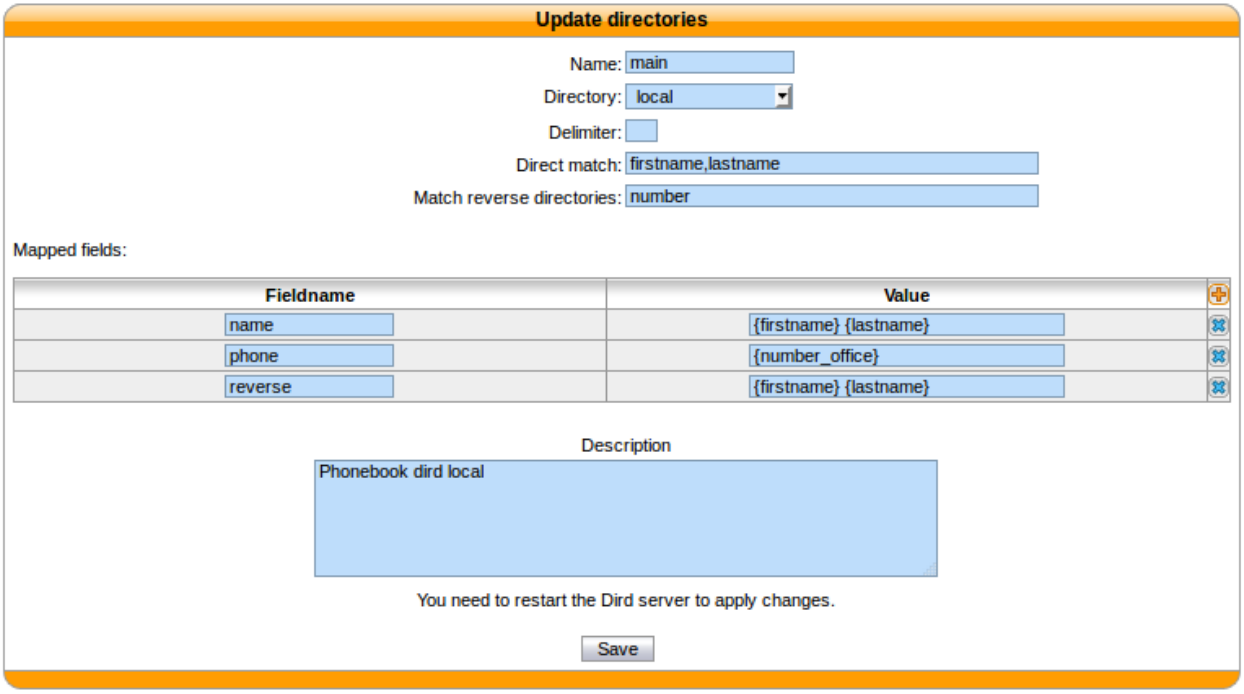


Fig. 1.61: *Services → CTI Server → Directories → Definitions*

Name: the name of the source

Directory: the name of the directory

Delimiter: (optional) the field delimiter in the data source

Direct match: the list used to match entries for direct lookup (comma separated)

Match reverse directories: (optional) the list used to match entries for reverse lookup (comma separated)

Mapped fields: used to add or modify columns in this directory source

Fieldname: the identifier for this new field

Value: a python format string that can be used to modify the data returned from a data source

This configuration will show the contact's company name on the caller ID name, when the incoming call will match office, mobile or home number.

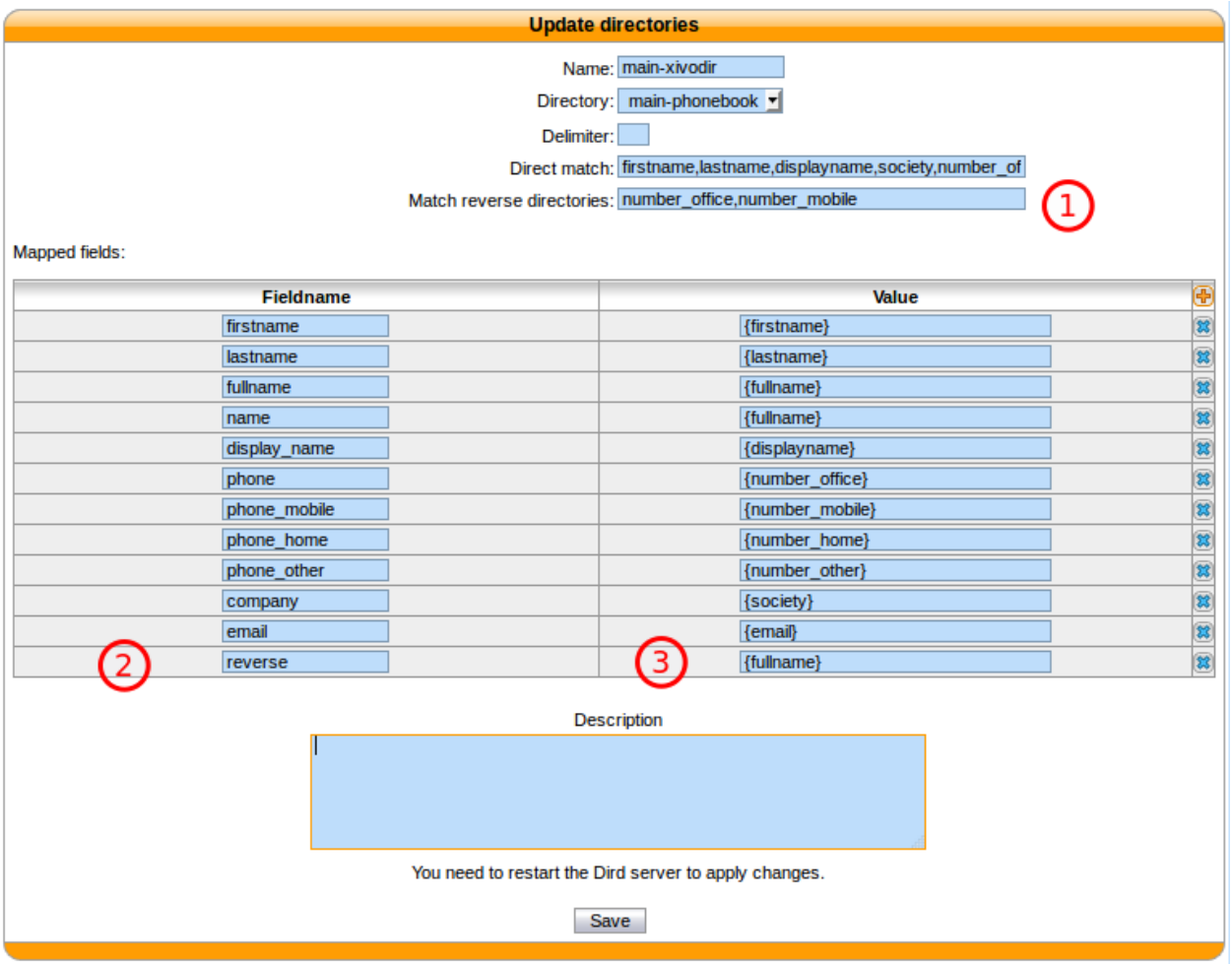


Fig. 1.62: *Services → CTI Server → Directories → Definitions*

Phone directory

Phone directory takes 2 *Fieldname* by default:

- `display_name`: the displayed name on the phone
- `phone`: the number to call

Examples:

You will find below some useful configurations of *Mapped fields*.

Adding a name field from firstname and lastname

Given a configuration where the directory source returns results with fields firstname and lastname . To add a *name* column to a directory, the administrator would add the following *Mapped fields*:

- *Fieldname*: `name`
- *Value*: `{firstname} {lastname}`

Prefixing a field

Given a directory source that need a prefix to be called, a new field can be created from an exising one. To add a prefix *9* to the numbers returned from a source, the administrator would add the following *Mapped fields*:

- *Fieldname*: `number`
- *Value*: `9{number}`

Adding a static field

Sometimes, it can be useful to add a field to the search results. A string can be added without any formatting. To add a *directory* field to the *xivodir* directory, the administrator would add the following *Mapped fields*:

- *Fieldname*: `directory`
- *Value*: `XiVO internal directory`

Configuring source display

Wazo Client

Edit the default display filter or create your own in *Services* → *CTI Server* → *Directories* → *Display filters*.

Each line in the display filter will result in a header in your Wazo Client.

- *Field title*: text displayed in the header.
- *Field type*: type of the column, this information is used by the Wazo Client. (see *type description*)
- *Default value*: value that will be used if this field is empty for one of the configured sources.
- *Field name*: name of the field in the directory definitions. The specified names should be available in the configured sources. To add new column name to a directory definition see above.

Phone

The only way to configure display phone directory is through *xivo-dird configuration*.

Adding a directory

To include a directory in direct directory definition:

1. Go to *Services* → *CTI Server* → *Directories* → *Direct directories*.
2. Edit your context.

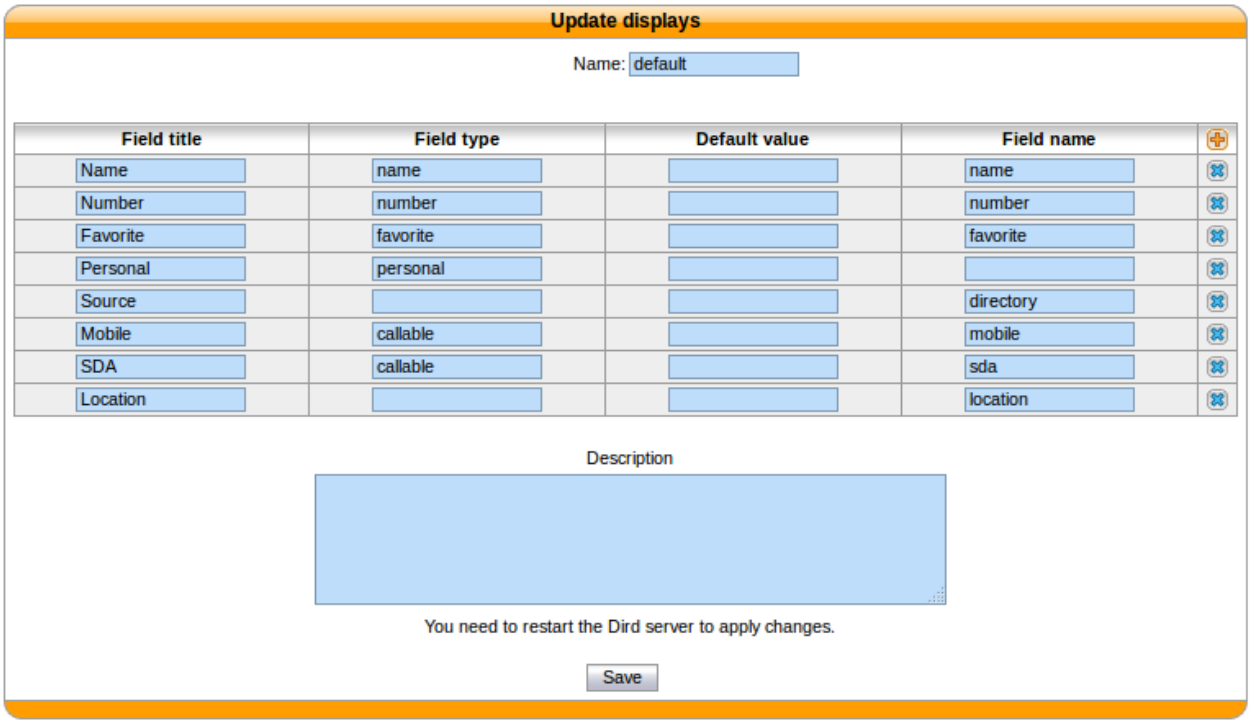


Fig. 1.63: *Services → CTI Server → Directories → Display filters*

3. Select your display filter.
4. Add the directories in the *Directories* section.

To include a directory in reverse directory definition:

1. Go to *Services → CTI Server → Directories → Reverse directories*.
2. Add the directories to include to reverse lookups in the *Related directories* section.

Applying changes

To reload the directory configuration for Wazo Client, phone lookups and reverse lookups, use *one* of these methods:

- *Services → IPBX → Control → Restart Dird server*
- console `service xivo-dird restart`

1.9.14 Directed Pickup

Directed pickup allows a user to intercept calls made to another user.

For example, if a user with number 1001 is ringing, you can dial *81001 from your phone and it will intercept (i.e. pickup) the call to this user.

The extension prefix used to pickup calls can be changed via the *Services → IPBX → IPBX services → Extensions* page.

Custom Line Limitation

There is a case where directed pickup does not work, which is the following:

```
Given you have a user U with a line of type "customized"
Given this custom line is using DAHDI technology
Given this user is a member of group G
When a call is made to group G
Then you won't be able to intercept the call made to U by pressing *8<line number of␣
↪U>
```

If you find yourself in this situation, you'll need to write a bit of dialplan.

For example, if you have the following:

- a user with a custom line with number 1001 in context default
- a custom line with interface `DAHDI/g1/5551234`

Then add the following, or similar:

```
[custom_lines]
exten = line1001,1,NoOp()
same  = n,Set(__PICKUPMARK=1001%default)
same  = n,Dial(DAHDI/g1/5551234)
same  = n,Hangup()
```

And do a `dialplan reload` in the asterisk CLI.

Then, edit the line of the user and change the interface value to `Local/line1001@custom_lines`

Note that you'll need to update your dialplan if you update the number of the line or the context.

1.9.15 Entities

Purpose

In some cases, as the telephony provider, you want different independent organisations to have their telephony served by your Wazo, e.g. different departments using the same telephony infrastructure, but you do not want each organisation to see or edit the configuration of other organisations.

Configuration

In *Configuration* → *Entities*, you can create entities, one for each independant organisation.

In *Configuration* → *Users*, you can select an entity for each administrator.

Note: Once an entity is linked with an administrator, it can not be deleted. You have to unlink the entity from all administrator to be able to delete it.

For the new entity to be useful, you need to create contexts in this entity. You may need:

- an Internal context for users, groups, queues, etc.
- an Incall context for incoming calls
- an Outcall context for outgoing calls, which should be included in the Internal context for the users to be able to call external numbers

Limitations

Global Fields

Some fields are globally unique and will collide when the same value is used in different entities:

- User CTI login
- Agent number
- Queue name
- Context name

An error message will appear when creating resources with colliding parameters, saying the resource already exists, even if the entity-linked administrator can not see them.

Affected Lists

Only the following lists may be filtered by entity:

- Lines
- Users
- Devices
- Groups
- Voicemails
- Conference Rooms
- Incoming calls
- Call filters
- Call pickups
- Schedules
- Agents
- Queues

For the devices:

- The filtering only applies to the devices associated with a line.
- The devices in autoprov mode or not configured mode are visible by every administrator.

REST API

The REST API does not have the notion of entity. When creating a resource without context via REST API, the resource will be associated to an arbitrary entity. Affected resources are:

- Contexts
- Call filters
- Group pickups
- Schedules

- Users

1.9.16 Fax

Fax transmission

It's possible to send faxes from Wazo using the fax Xlet in the Wazo Client.

The file to send must be in PDF format.

Fax reception

Adding a fax reception DID

If you want to receive faxes from Wazo, you need to add incoming calls definition with the *Application* destination and the *FaxToMail* application for every DID you want to receive faxes from.

This applies even if you want the action to be different from sending an email, like putting it on a FTP server. You'll still need to enter an email address in these cases even though it won't be used.

Note that, as usual when adding incoming call definitions, you must first define the incoming call range in the used context.

Changing the email body

You can change the body of the email sent upon fax reception by editing `/etc/xivo/mail.txt`.

The following variable can be included in the mail body:

- `%(dstnum)s`: the DID that received the fax

If you want to include a regular percent character, i.e. `%`, you must write it as `%%` in `mail.txt` or an error will occur when trying to do the variables substitution.

The `agid` service must be restarted to apply changes:

```
service xivo-agid restart
```

Changing the email subject

You can change the subject of the email sent upon fax reception by editing `/etc/xivo/asterisk/xivo_fax.conf`.

Look for the `[mail]` section, and in this section, modify the value of the `subject` option.

The available variable substitution are the same as for the email body.

The `agid` service must be restarted to apply changes:

```
service xivo-agid restart
```

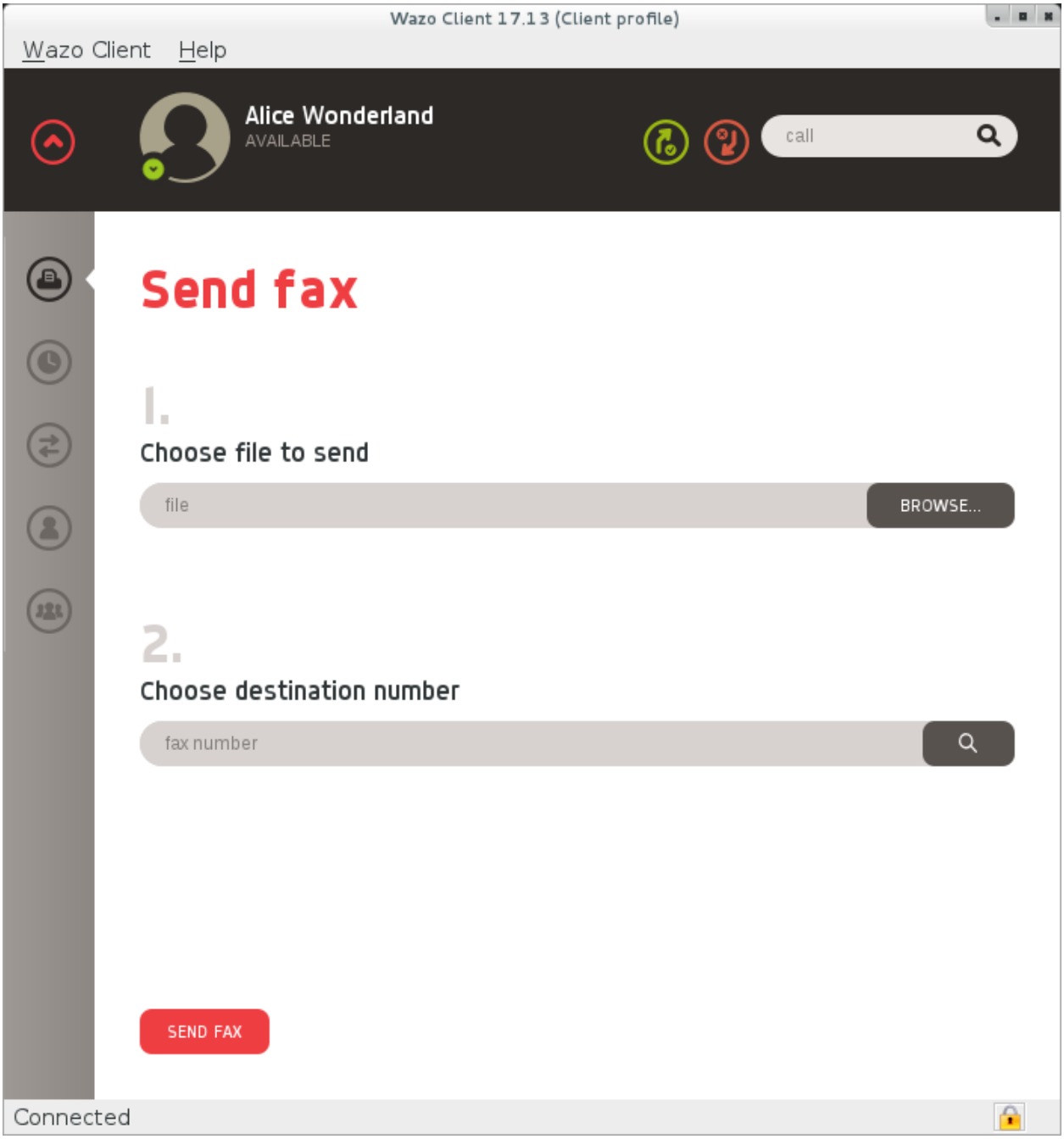
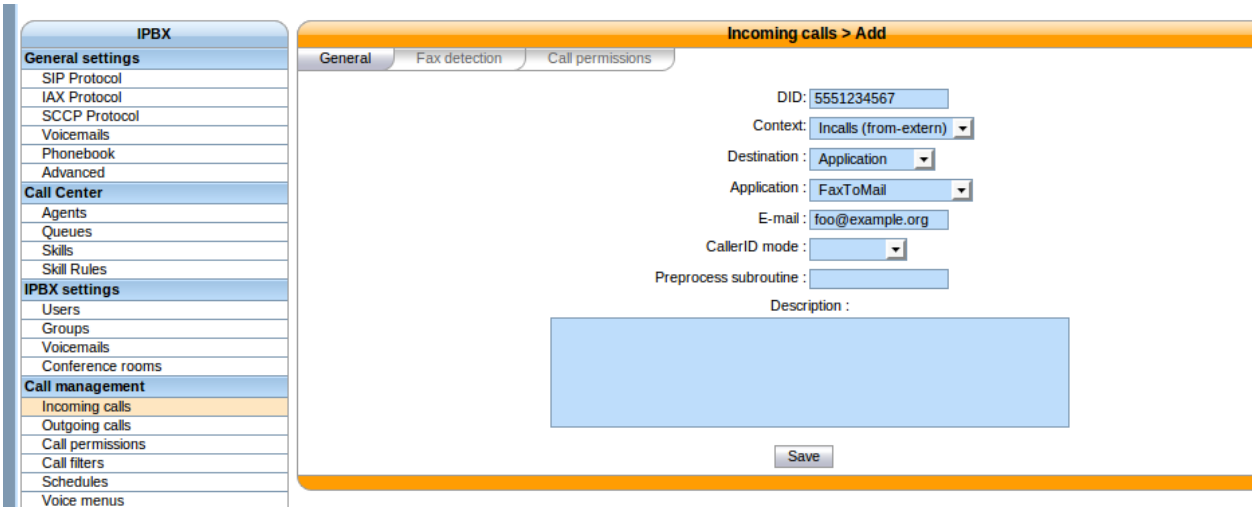


Fig. 1.64: The fax Xlet in the Wazo Client



Changing the email from

You can change the from of the email sent upon fax reception by editing `/etc/xivo/asterisk/xivo_fax.conf`.

Look for the `[mail]` section, and in this section, modify the value of the `email_from` option.

The `agid` service must be restarted to apply changes:

```
service xivo-agid restart
```

Changing the email realname

You can change the realname of the email sent upon fax reception by editing `/etc/xivo/asterisk/xivo_fax.conf`.

Look for the `[mail]` section, and in this section, modify the value of the `email_realname` option.

The `agid` service must be restarted to apply changes:

```
service xivo-agid restart
```

Using the advanced features

The following features are only available via the `/etc/xivo/asterisk/xivo_fax.conf` configuration file. They are not available from the web-interface.

The way it works is the following:

- you first declare some backends, i.e. actions to be taken when a fax is received. A backend name looks like `mail`, `ftp_example_org` or `printer_office`.
- once your backends are defined, you can use them in your destination numbers. For example, when someone calls the DID 100, you might want the `ftp_example_org` and `mail` backend to be run, but otherwise, you only want the `mail` backend to be run.

Here's an example of a valid `/etc/xivo/asterisk/xivo_fax.conf` configuration file:

```
[general]
tiff2pdf = /usr/bin/tiff2pdf
mutt = /usr/bin/mutt
lp = /usr/bin/lp

[mail]
subject = FAX reception to %(dstnum)s
content_file = /etc/xivo/mail.txt
email_from = no-reply+fax@wazo.community
email_realname = Service Fax

[ftp_example_org]
host = example.org
username = foo
password = bar
directory = /foobar

[dstnum_default]
dest = mail

[dstnum_100]
dest = mail, ftp_example_org
```

The section named `dstnum_default` will be used only if no DID-specific actions are defined.

After editing `/etc/xivo/asterisk/xivo_fax.conf`, you need to restart the agid server for the changes to be applied:

```
service xivo-agid restart
```

Using the FTP backend

The FTP backend is used to send a PDF version of the received fax to an FTP server.

An FTP backend is always defined in a section beginning with the `ftp` prefix. Here's an example for a backend named `ftp_example_org`:

```
[ftp_example_org]
host = example.org
port = 2121
username = foo
password = bar
directory = /foobar
convert_to_pdf = 0
```

The `port` option is optional and defaults to 21.

The `directory` option is optional and if not specified, the document will be put in the user's root directory.

The `convert_to_pdf` option is optional and defaults to 1. If it is set to 0, the TIFF file will not be converted to PDF before being sent to the FTP server.

The uploaded file are named like `${XIVO_SRCNUM}-${EPOCH}.pdf`.

Using the printer backend

To use the printer backend, you must have the `cups-client` package installed on your Wazo:

```
$ apt-get install cups-client
```

The printer backend uses the `lp` command to print faxes.

A printer backend is always defined in a section beginning with the `printer` prefix. Here's an example for a backend named `printer_office`:

```
[printer_office]
name = office
convert_to_pdf = 1
```

When a fax will be received, the system command `lp -d office <faxfile>` will be executed.

The `convert_to_pdf` option is optional and defaults to 1. If it is set to 0, the TIFF file will not be converted to PDF before being printed.

Warning: You need a CUPS server set up somewhere on your network.

Using the mail backend

By default, a mail backend named `mail` is defined. You can define more mail backends if you want. Just look what the default mail backend looks like.

Fax detection

Wazo **does not currently support Fax Detection**. A workaround is described in the *Fax detection* section.

Using analog gateways

Wazo is able to provision Cisco SPA122 and Linksys SPA2102, SPA3102 and SPA8000 analog gateways which can be used to connect fax equipments. This section describes the creation of custom template *for SPA3102* which modifies several parameters.

Note: **With SPA ATA plugins >= v0.8**, you **should not need** to follow this section anymore since all of these parameters are now set in the base templates of all, except for Echo_Canc_Adapt_Enable, Echo_Supp_Enable, Echo_Canc_Enable.

Note: Be aware that most of the parameters are or could be country specific, i.e. :

- Preferred Codec,
- FAX Passthru Codec,
- RTP Packet Size,
- RTP-Start-Loopback Codec,
- Ring Waveform,

- Ring Frequency,
- Ring Voltage,
- FXS Port Impedance

1. Create a custom template for the SPA3102 base template:

```
cd /var/lib/xivo-provd/plugins/xivo-cisco-spa3102-5.1.10/var/templates/
cp ../../templates/base.tpl .
```

2. Add the following content before the `</flat-profile>` tag:

```
<!-- CUSTOM TPL - for faxes - START -->

{% for line_no, line in sip_lines.iteritems() %}
<!-- Dial Plan: L{{ line_no }} -->
<Dial_Plan_{{ line_no }}_ ua="na">([x*#].)</Dial_Plan_{{ line_no }}_>

<Call_Waiting_Serv_{{ line_no }}_ ua="na">No</Call_Waiting_Serv_{{ line_no }}_>
<Three_Way_Call_Serv_{{ line_no }}_ ua="na">No</Three_Way_Call_Serv_{{ line_no }}_
↪>

<Preferred_Codec_{{ line_no }}_ ua="na">G711a</Preferred_Codec_{{ line_no }}_>
<Silence_Supp_Enable_{{ line_no }}_ ua="na">No</Silence_Supp_Enable_{{ line_no }}_
↪>
<Echo_Canc_Adapt_Enable_{{ line_no }}_ ua="na">No</Echo_Canc_Adapt_Enable_{{ line_
↪no }}_>
<Echo_Supp_Enable_{{ line_no }}_ ua="na">No</Echo_Supp_Enable_{{ line_no }}_>
<Echo_Canc_Enable_{{ line_no }}_ ua="na">No</Echo_Canc_Enable_{{ line_no }}_>
<Use_Pref_Codec_Only_{{ line_no }}_ ua="na">yes</Use_Pref_Codec_Only_{{ line_no }}
↪_>
<DTMF_Tx_Mode_{{ line_no }}_ ua="na">Normal</DTMF_Tx_Mode_{{ line_no }}_>

<FAX_Enable_T38_{{ line_no }}_ ua="na">Yes</FAX_Enable_T38_{{ line_no }}_>
<FAX_T38_Redundancy_{{ line_no }}_ ua="na">1</FAX_T38_Redundancy_{{ line_no }}_>
<FAX_Passthru_Method_{{ line_no }}_ ua="na">ReINVITE</FAX_Passthru_Method_{{ line_
↪no }}_>
<FAX_Passthru_Codec_{{ line_no }}_ ua="na">G711a</FAX_Passthru_Codec_{{ line_no }}
↪_>
<FAX_Disable_ECAN_{{ line_no }}_ ua="na">yes</FAX_Disable_ECAN_{{ line_no }}_>
<FAX_Tone_Detect_Mode_{{ line_no }}_ ua="na">caller or callee</FAX_Tone_Detect_
↪Mode_{{ line_no }}_>

<Network_Jitter_Level_{{ line_no }}_ ua="na">very high</Network_Jitter_Level_{{␣
↪line_no }}_>
<Jitter_Buffer_Adjustment_{{ line_no }}_ ua="na">disable</Jitter_Buffer_
↪Adjustment_{{ line_no }}_>
{% endfor %}

<!-- SIP Parameters -->
<RTP_Packet_Size ua="na">0.020</RTP_Packet_Size>
<RTP-Start-Loopback_Codec ua="na">G711a</RTP-Start-Loopback_Codec>

<!-- Regional parameters -->
<Ring_Waveform ua="rw">Sinusoid</Ring_Waveform> <!-- options: Sinusoid/Trapezoid -
↪->
<Ring_Frequency ua="rw">50</Ring_Frequency>
<Ring_Voltage ua="rw">85</Ring_Voltage>
```

```
<FXS_Port_Impedance ua="na">600+2.16uF</FXS_Port_Impedance>
<Caller_ID_Method ua="na">Bellcore(N.Amer,China)</Caller_ID_Method>
<Caller_ID_FSK_Standard ua="na">bell 202</Caller_ID_FSK_Standard>

<!-- CUSTOM TPL - for faxes - END -->
```

3. Reconfigure the devices with:

```
xivo-provd-cli -c 'devices.using_plugin("xivo-cisco-spa3102-5.1.10").reconfigure()
↪'
```

4. Then reboot the devices:

```
xivo-provd-cli -c 'devices.using_plugin("xivo-cisco-spa3102-5.1.10").synchronize()
↪'
```

Most of this template can be copy/pasted for a SPA2102 or SPA8000.

Using a SIP Trunk

Fax transmission, to be successful, *MUST* use G.711 codec. Fax streams cannot be encoded with lossy compression codecs (like G.729a).

That said, you may want to establish a SIP trunk using G.729a for all other communications to save bandwith. Here's a way to be able to receive a fax in this configuration.

Note: There are some prerequisites:

- your SIP Trunk must offer both G.729a and G.711 codecs
- your fax users must have a customized outgoing calleridnum (for the codec change is based on this variable)

---

1. We assume that outgoing call rules and fax users with their DID are created
2. Create the file `/etc/asterisk/extensions_extra.d/fax.conf` with the following content:

```
;; For faxes :
; The following subroutine forces inbound and outbound codec to alaw or ulaw.
; For outbound codec selection we must set the variable with inheritance.
; Must be set on each Fax DID
[pre-incall-fax]
exten = s,1,NoOp(### Force alaw,ulaw codec on both inbound (operator side) and␣
↪outbound (analog gw side) when calling a Fax ###)
exten = s,n,Set(SIP_CODEC_INBOUND=alaw,ulaw)
exten = s,n,Set(__SIP_CODEC_OUTBOUND=alaw,ulaw)
exten = s,n,Return()

; The following subroutine forces outbound codec to alaw or ulaw based on␣
↪outgoing callerid number
; For outbound codec selection we must set the variable with inheritance.
; Must be set on each outgoing call rule
[pre-outcall-fax]
exten = s,1,NoOp(### Force alaw,ulaw codec if caller is a Fax ###)
exten = s,n,GotoIf($["${CALLERID(num)}" = "0112697845"]?g711:)
exten = s,n,GotoIf($["${CALLERID(num)}" = "0112697846"]?g711:end)
```

```
exten = s,n(g711),Set(__SIP_CODEC_OUTBOUND=alaw,ulaw)
exten = s,n(end),Return()
```

3. For each Fax users' DID add the following string in the `Preprocess subroutine` field:

```
pre-incall-fax
```

4. For each Outgoing call rule add the the following string in the `Preprocess subroutine` field:

```
pre-outcall-fax
```

1.9.17 Graphics

The Services/Graphics section gives a historical overview of a Wazo system's activity based on snapshots recorded every 5 minutes. Graphics are available for the following resources :

- CPU
- Entropy
- Interruptions
- IRQ Stats
- System Load
- Memory Usage
- Open Files
- Open Inodes
- Swap Usage

Each section is presented as a series of 4 graphics : daily, weekly, monthly and yearly history. Each graphic can be clicked on to zoom. All information presented is read only.

Troubleshooting

Missing graphs

Symptoms:

- daily graphs are missing
- weekly/monthly/yearly graphs stop updating
- a mail is sent from cron every 5 minutes about a "bad magic number"

Cause:

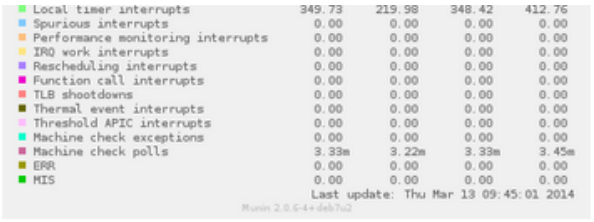
- the machine was forcefully stopped, while munin (responsible for the graphs) was running, resulting in a file corruption

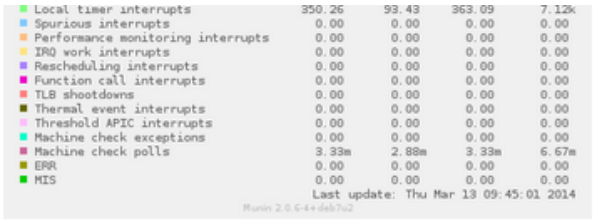
Resolution:

- Run the following command:

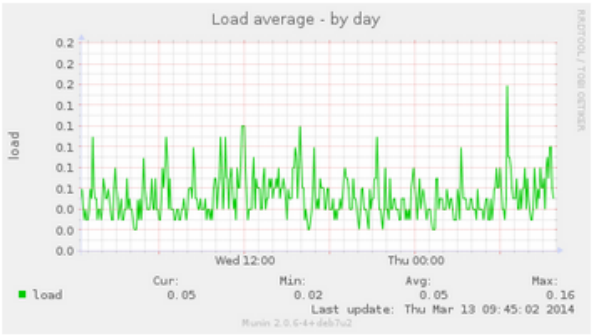
```
rm /var/lib/munin/htmlconf.storable /var/lib/munin/limits.storable
```

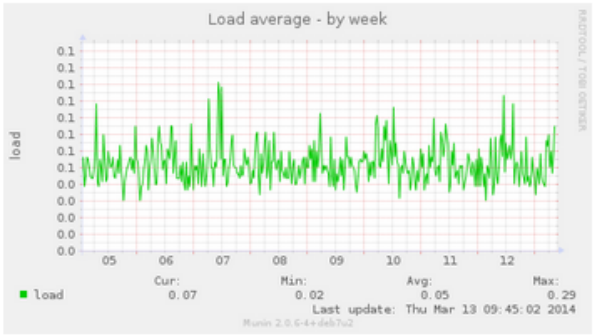
- The graphs will be restored on the next run of munin, in the next 5 minutes.

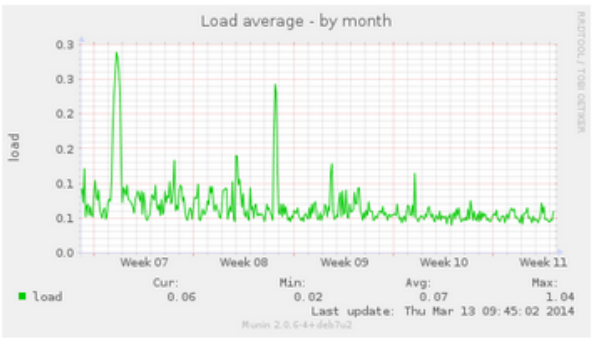


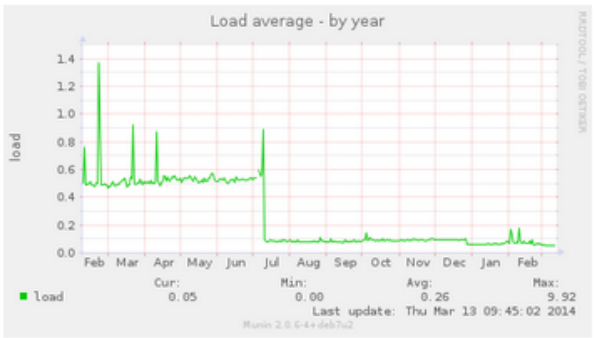


System load

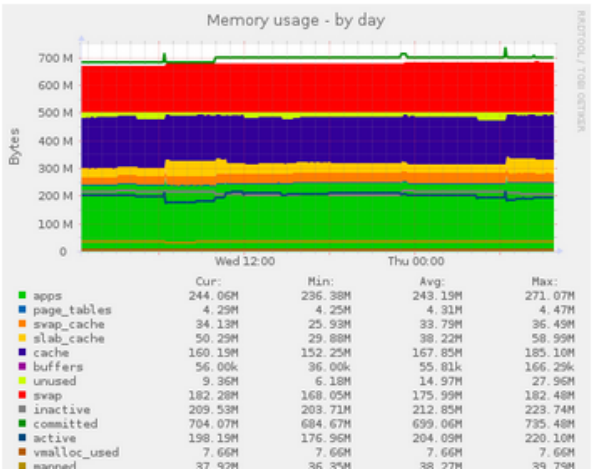


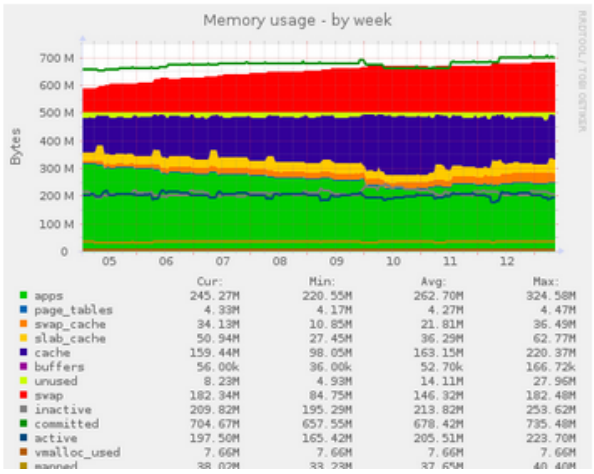






Memory usage





1.9.18 Groups

Groups are used to be able to call a set or users.

Group name cannot be `general` reserved in asterisk configuration.

1.9.19 Group Pickup

Pickup groups allow users to intercept calls directed towards other users of the group. This is done either by dialing a special extension or by pressing a function key.

Quick Summary

In order to be able to use group pickup you have to:

- Create a pickup group
- Enable an extension to intercept calls
- Add a function key to interceptors

Creating a Pickup Group

Pickup groups can be created in the *Services* → *IPBX* → *Call management* → *Call pickups* page.

In the *general* tab, you can define a name and a description for the pickup group. In the *Interceptors* tab, you can define a list of users, groups or queues that can intercept calls. In the *Intercepted* tab, you can define a list of users, groups or queues that can be intercepted.

Enabling an Interception Extension

The pickup extension can be defined in the *Services* → *IPBX* → *IPBX services* → *Extensions* page.

The extension used by group pickup is called *Group interception* it's default value is **8*.

Warning: The extension must be enabled even if a function key is used.

Adding a Function Key to an Interceptor

To assign a function to an interceptor, go to *Services* → *IPBX* → *IPBX settings* → *Users*, edit an interceptor and go to the *Func Keys* tab.

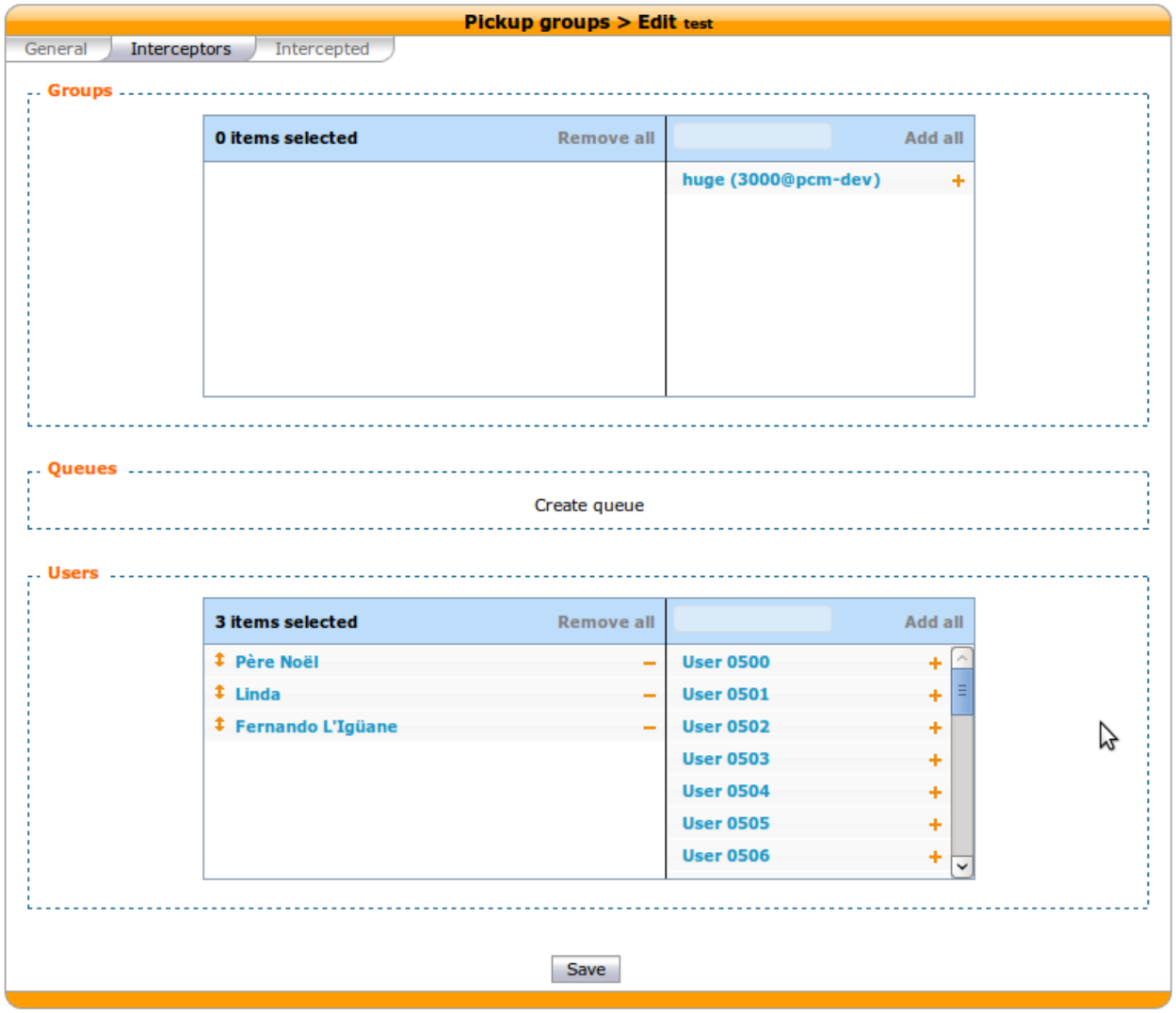
Add a new function key of type *Group Interception* and save.

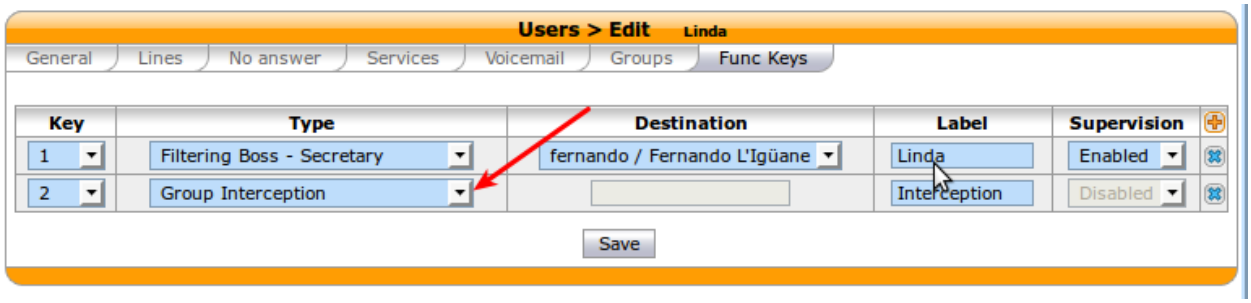
1.9.20 Server/Hardware

This section describes how to configure the telephony hardware on a Wazo server.

Note: Currently Wazo supports only Digium Telephony Interface cards

The configuration process is the following :


Pickup groups > Edit test
General
Interceptors
Intercepted
Groups
0 items selected
Remove all
Add all
huge (3000@pcm-dev)
Queues
Create queue
Users
3 items selected
Remove all
Add all
Père Noël
Linda
Fernando L'Igüane
User 0500
User 0501
User 0502
User 0503
User 0504
User 0505
User 0506
Save


Users > Edit Linda
General
Lines
No answer
Services
Voicemail
Groups
Func Keys
Key
Type
Destination
Label
Supervision
1
Filtering Boss - Secretary
fernando / Fernando L'Igüane
Linda
Enabled
2
Group Interception
Interception
Disabled
Save

Load the correct DAHDI modules

For your Digium card to work properly you must load the appropriate DAHDI kernel module. This is done via the file `/etc/dahdi/modules` and this page will guide you through its configuration.

Know which card is in your server

You can see which cards are detected by issuing the `dahdi_hardware` command:

```
dahdi_hardware
pci:0000:05:0d.0     wcb4xxp-     d161:b410 Digium Wildcard B410P
pci:0000:05:0e.0     wct4xxp-     d161:0205 Wildcard TE205P (4th Gen)
```

This command gives the card name detected and, more importantly, the DAHDI kernel module needed for this card. In the above example you can see that two cards are detected in the system:

- a Digium B410P *which needs* the `wcb4xxp` module
- and a Digium TE205P *which needs* the `wct4xxp` module

Create the configuration file

Now that we know the modules we need, we can create our configuration file:

1. Create the file `/etc/dahdi/modules`:

   ```
   touch /etc/dahdi/modules
   ```

2. Fill it with the modules name you found with the `dahdi_hardware` command (one module name per line). In our example, your `/etc/dahdi/modules` file should contain the following lines:

   ```
   wcb4xxp
   wct4xxp
   ```

Note: In the `/usr/share/dahdi/modules.sample` file you can find all the modules supported in your Wazo version.

Apply the configuration

To apply the configuration, restart the services:

```
wazo-service restart
```

Next step

Now that you have loaded the correct module for your card you must:

1. check if you need to follow one of the *Specific configuration* sections below,
2. and continue with the next configuration step which is to *configure the echo canceller*.

Specific configuration

This section lists some specific configuration. You should not follow them unless you have a specific need.

TE13x, TE23x, TE43x: E1/T1 selection

With E1/T1 cards you must select the correct *line mode* between:

- E1 : the European standard,
- and T1 : North American standard

For old generation cards (TE12x, TE20x, TE40x series) the *line mode* is selected via a physical jumper.

For new generation cards like TE13x, TE23x, TE43x series the *line mode* is selected by configuration.

If you're configuring one of these **TE13x, T23x, T43x** cards then you **MUST** create a configuration file to set the line mode to E1:

1. Create the file `/etc/modprobe.d/xivo-wcte-linemode.conf`:

```
touch /etc/modprobe.d/xivo-wcte-linemode.conf
```

2. Fill it with the following lines replacing `DAHDI_MODULE_NAME` by the correct module name (`wcte13xp`, `wcte43x` ...):

```
# set the card in E1/T1 mode
options DAHDI_MODULE_NAME default_linemode=e1
```

3. Then, restart the services:

```
wazo-service restart
```

Hardware Echo-cancellation

It is *recommended* to use telephony cards with an hardware echo-canceller module.

Warning: with **TE13x, TE23x and TE43x** cards, you **MUST** install the echo-canceller firmware. Otherwise the card won't work properly.

Know which firmware you need

If you have an hardware echo-canceller module you **have to** install its firmware.

You first need to know which firmware you have to install. The simplest way is to restart dahdi and then to lookup in the dmesg which firmware does DAHDI request at startup:

```
wazo-service restart
dmesg |grep firmware
[5461540.738209] wct4xxp 0000:01:0e.0: firmware: agent aborted loading dahdi-fw-
↪oct6114-064.bin (not found?)
[5461540.738310] wct4xxp 0000:01:0e.0: VPM450: firmware dahdi-fw-oct6114-064.bin not␣
↪available from userspace
```

In the example above you can see that the module `wct4xxp` requested the `dahdi-fw-oct6114-064.bin` firmware file but did not found it. But you now know that you need the `dahdi-fw-oct6114-064.bin` firmware.

Install the firmware

When you know which firmware you need you can install it with `xivo-fetchfw` utility.

1. Use `xivo-fetchfw` to find the name of the package. You can search for `digium` occurrences in the available packages:

```
xivo-fetchfw search digium
```

2. Find the package name which matches the firmware file you need. In our example, we need the `dahdi-fw-oct6114-064.bin` file which is supplied by the package named `digium-oct6114-064`:

```
xivo-fetchfw install digium-oct6114-064
```

Activate the Hardware Echo-cancellation

Now that you installed hardware echo-canceller firmware you must activate it in `/etc/asterisk/chan_dahdi.conf` file:

```
echocancel = 1
```

Apply the configuration

To apply the configuration, restart the services:

```
wazo-service restart
```

Next step

Now that you have loaded the correct module for your card you must:

1. check if you need to follow one of the *Specific configuration* sections below,
2. and continue with the next configuration step which is to *configure your card* according to the operator links.

Specific configuration

This section describes some specific configuration. You should not follow them unless you have a specific need.

Use the Hardware Echo-canceller for DTMF detection

If you have an hardware echo-canceller you *may* want to use it to detect the DTMF signal (instead of asterisk).

1. Create the file `/etc/modprobe.d/xivo-hwec-dtmf.conf`:

```
touch /etc/modprobe.d/xivo-hwec-dtmf.conf
```

2. Fill it with the following lines replacing `DAHDI_MODULE_NAME` by the correct module name (`wcte13xp`, `wct4xxp` ...):

```
options DAHDI_MODULE_NAME vpmdtmfsupport=1
```

3. Then, restart the services:

```
wazo-service restart
```

Card configuration

Now that you have *loaded the correct DAHDI modules* and *configured the echo canceller* you can proceed with the card configuration. Follow one of the appropriate link below :

BRI card configuration

Verifications

Verify that the `wcb4xxp` module is uncommented in `/etc/dahdi/modules`.

If it wasn't, do again the step *Load the correct DAHDI modules*.

Generate DAHDI configuration

Issue the command:

```
dahdi_genconf
```

Warning: it will erase all existing configuration in `/etc/dahdi/system.conf` and `/etc/asterisk/dahdi-channels.conf` files !

Configure

DAHDI system.conf configuration

First step is to check `/etc/dahdi/system.conf` file:

- check the span numbering,
- if needed change the clock source,

See detailed explanations of this file in the */etc/dahdi/system.conf* section.

Below is **an example** for a typical french BRI line span:

```
# Span 1: B4/0/1 "B4XXP (PCI) Card 0 Span 1" (MASTER) RED
span=1,1,0,ccs,ami
# termtype: te
bchan=1-2
hardhdlc=3
echocanceller=mg2,1-2
```

Asterisk dahdi-channels.conf configuration

Then you have to modify the `/etc/asterisk/dahdi-channels.conf` file:

- remove the unused lines like:

```
context = default
group = 63
```

- change the `context` lines if needed,
- the `signalling` should be one of:
  - `bri_net`
  - `bri_cpe`
  - `bri_net_ptmp`
  - `bri_cpe_ptmp`

See some explanations of this file in the */etc/asterisk/dahdi-channels.conf* section.

Below is **an example** for a typical french BRI line span:

```
; Span 1: B4/0/1 "B4XXP (PCI) Card 0 Span 1" (MASTER) RED
group = 0,11              ; belongs to group 0 and 11
context = from-extern     ; incoming call to this span will be sent in 'from-extern'␣
↪context
switchtype = euroisdn
signalling = bri_cpe    ; use 'bri_cpe' signalling
channel => 1-2          ; the above configuration applies to channels 1 and 2
```

Next step

Now that you have configured your BRI card:

1. you must check if you need to follow one of the *Specific configuration* sections below,
2. then, if you have another type of card to configure, you can go back to the *configure your card* section,
3. if you have configured all your card you have to configure the *DAHDI interconnections* in the web interface.

Specific configuration

You will find below 3 configurations that we recommend for BRI lines. These configurations were tested on different type of french BRI lines with success.

Note: The pre-requisites are:

- XiVO/Wazo >= 14.12,
- Use per-port dahdi interconnection (see the *DAHDI interconnections* section)

If you don't know which one to configure we recommend that you try each one after the other in this order:

1. *PTMP without layer1/layer2 persistence*
2. *PTMP with layer1/layer2 persistence*

3. *PTP with layer1/layer2 persistence*

PTMP without layer1/layer2 persistence

In this mode we will configure asterisk and DAHDI:

- to use Point-to-Multipoint (PTMP) signalling,
- and to leave Layer1 and Layer2 DOWN

Follows theses steps to configure:

1. **Before** the line `#include dahdi-channels.conf` add, in file `/etc/asterisk/chan_dahdi.conf`, the following lines:

```
layer1_presence = ignore
layer2_persistence = leave_down
```

2. In the file `/etc/asterisk/dahdi-channels.conf` use `bri_cpe_ptmp` signalling:

```
signalling = bri_cpe_ptmp
```

3. Create the file `/etc/modprobe.d/xivo-wcb4xxp.conf` to deactivate the layer1 persistence:

```
touch /etc/modprobe.d/xivo-wcb4xxp.conf
```

4. Fill it with the following content:

```
options wcb4xxp persistentlayer1=0
```

5. Then, apply the configuration by restarting the services:

```
wazo-service restart
```

Note: Expected behavior:

- The *dahdi show status* command should show the BRI spans in *RED* status if there is no call,
- For outgoing calls the layer1/layer2 should be brought back up by the Wazo (i.e. asterisk/chan_dahdi),
- For incoming calls the layer1/layer2 should be brought back up by the operator,
- You can consider that there is *a problem* only if incoming or outgoing calls are rejected.

PTMP with layer1/layer2 persistence

In this mode we will configure asterisk and DAHDI:

- to use Point-to-Multipoint (PTMP) signalling,
- and to keep Layer1 and Layer2 UP

Follows theses steps to configure:

1. **Before** the line `#include dahdi-channels.conf` add, in file `/etc/asterisk/chan_dahdi.conf`, the following lines:

```
layer1_presence = required
layer2_persistence = keep_up
```

2. In the file `/etc/asterisk/dahdi-channels.conf` use `bri_cpe_ptmp` signalling:

```
signalling = bri_cpe_ptmp
```

3. If it exists, delete the file `/etc/modprobe.d/xivo-wcb4xxp.conf`:

```
rm /etc/modprobe.d/xivo-wcb4xxp.conf
```

4. Then, apply the configuration by restarting the services:

```
wazo-service restart
```

Note: Expected behavior:

- The *dahdi show status* command should show the BRI spans in **OK** status even if there is no call,
- In asterisk CLI you may see the spans going Up/Down/Up : it is *a problem* only if incoming or outgoing calls are rejected.

PTP with layer1/layer2 persistence

In this mode we will configure asterisk and DAHDI:

- to use Point-to-Point (PTP) signalling,
- and use default behavior for Layer1 and Layer2.

Follow theses steps to configure:

1. In file `/etc/asterisk/chan_dahdi.conf` remove all occurrences of `layer1_presence` and `layer2_persistence` options.
2. In the file `/etc/asterisk/dahdi-channels.conf` use `bri_cpe` signalling:

```
signalling = bri_cpe
```

3. If it exists, delete the file `/etc/modprobe.d/xivo-wcb4xxp.conf`:

```
rm /etc/modprobe.d/xivo-wcb4xxp.conf
```

4. Then, apply the configuration by restarting the services:

```
wazo-service restart
```

Note: Expected behavior:

- The *dahdi show status* command should show the BRI spans in **OK** status even if there is no call,
- In asterisk CLI you should not see the spans going Up and Down : if it happens, it is *a problem* only if incoming or outgoing calls are rejected.

PRI card configuration

Verifications

Verify that the correct module is configured in `/etc/dahdi/modules` depending on the card you installed in your server.

If it wasn't, do again the step *Load the correct DAHDI modules*

> **Warning:** *TE13x, TE23x, TE43x* cards :
>
> - these cards need a specific dahdi module configuration. See *TE13x, TE23x, TE43x: E1/T1 selection* paragraph,
> - you **MUST** install the correct echo-canceller firmware to be able to use these cards. See *Hardware Echo-cancellation* paragraph.

Generate DAHDI configuration

Issue the command:

```
dahdi_genconf
```

> **Warning:** it will erase all existing configuration in `/etc/dahdi/system.conf` and `/etc/asterisk/dahdi-channels.conf` files !

Configure

DAHDI system.conf configuration

First step is to check `/etc/dahdi/system.conf` file:

- check the span numbering,
- if needed change the clock source,
- usually (at least in France) you should remove the `crc4`

See detailed explanations of this file in the */etc/dahdi/system.conf* section.

Below is **an example** for a typical french PRI line span:

```
# Span 1: TE2/0/1 "T2XXP (PCI) Card 0 Span 1" CCS/HDB3/CRC4 RED
span=1,1,0,ccs,hdb3
# termtype: te
bchan=1-15,17-31
dchan=16
echocanceller=mg2,1-15,17-31
```

Asterisk dahdi-channels.conf configuration

Then you have to modify the `/etc/asterisk/dahdi-channels.conf` file:

- remove the unused lines like:

```
context = default
group = 63
```

- change the `context` lines if needed,
- the `signalling` should be one of:
  - `pri_net`
  - `pri_cpe`

Below is **an example** for a typical french PRI line span:

```
; Span 1: TE2/0/1 "T2XXP (PCI) Card 0 Span 1" CCS/HDB3/CRC4 RED
group = 0,11            ; belongs to group 0 and 11
context = from-extern   ; incoming call to this span will be sent in 'from-extern'␣
↪context
switchtype = euroisdn
signalling = pri_cpe    ; use 'pri_cpe' signalling
channel => 1-15,17-31   ; the above configuration applies to channels 1 to 15 and 17␣
↪to 31
```

Next step

Now that you have configured your PRI card:

1. you must check if you need to follow one of the *Specific configuration* sections below,
2. then, if you have another type of card to configure, you can go back to the *configure your card* section,
3. if you have configured all your card you have to configure the *DAHDI interconnections* in the web interface.

Specific configuration

Multiple PRI cards and sync cable

If you have several PRI cards in your server you should link them with a synchronization cable to share the exact same clock.

To do this, you need to:

- use the coding wheel on the Digium cards to give them an order of recognition in DAHDI/Asterisk (see Digium_telephony_cards_support),
- daisy-chain the cards with a sync cable (see Digium_telephony_cards_support),
- load the DAHDI module with the `timingcable=1` option.

Create `/etc/modprobe.d/xivo-timingcable.conf` file and insert the line:

```
options DAHDI_MODULE_NAME timingcable=1
```

Where `DAHDI_MODULE_NAME` is the DAHDI module name of your card (e.g. wct4xxp for a TE205P).

Analog card configuration

Limitations

- Wazo does not support hardware echocanceller on the TDM400 card. Users of TDM400 card willing to setup an echocanceller will have to use a software echocanceller like OSLEC.

Verifications

Verify that one of the `{wctdm,wctdm24xxp}` module is uncommented in `/etc/dahdi/modules` depending on the card you installed in your server.

If it wasn't, do again the step *Load the correct DAHDI modules*

Note: Analog cards work with card module. You must add the appropriate card module to your analog card. Either:

- an FXS module (for analog equipment - phones, . . .),
- an FXO module (for analog line)

Generate DAHDI configuration

Issue the command:

```
dahdi_genconf
```

Warning: it will erase all existing configuration in `/etc/dahdi/system.conf` and `/etc/asterisk/dahdi-channels.conf` files !

Configure

DAHDI system.conf configuration

First step is to check `/etc/dahdi/system.conf` file:

- check the span numbering,

See detailed explanations of this file in the */etc/dahdi/system.conf* section.

Below is **an example** for a typical FXS analog line span:

```
# Span 2: WCTDM/4 "Wildcard TDM400P REV I Board 5"
fxoks=32
echocanceller=mg2,32
```

Asterisk dahdi-channels.conf configuration

Then you have to modify the `/etc/asterisk/dahdi-channels.conf` file:

- remove the unused lines like:

```
context = default
group = 63
```

- change the `context` and `callerid` lines if needed,
- the `signalling` should be one of:
  - `fxo_ks` for **FXS** lines -yes it is the reverse
  - `fxs_ks` for **FXO** lines - yes it is the reverse

Below is **an example** for a typical french PRI line span:

```
; Span 2: WCTDM/4 "Wildcard TDM400P REV I Board 5"
signalling=fxo_ks
callerid="Channel 32" <4032>
mailbox=4032
group=5
context=default
channel => 32
```

Next step

Now that you have configured your PRI card:

1. you must check if you need to follow one of the *Specific configuration* sections below,
2. then, if you have another type of card to configure, you can go back to the *configure your card* section,
3. if you have configured all your card you have to configure the *DAHDI interconnections* in the web interface.

Specific configuration

FXS modules

If you use **FXS** modules you should create the file `/etc/modprobe.d/xivo-tdm` and insert the line:

```
options DAHDI_MODULE_NAME fastringer=1 boostringer=1
```

Where DAHDI_MODULE_NAME is the DAHDI module name of your card (e.g. wctdm for a TDM400P).

FXO modules

If you use **FXO** modules you should create file `/etc/modprobe.d/xivo-tdm`:

```
options DAHDI_MODULE_NAME opermode=FRANCE
```

Where DAHDI_MODULE_NAME is the DAHDI module name of your card (e.g. wctdm for a TDM400P).

Voice Compression Card configuration

Verifications

Verify that the `wctc4xxp` module is uncommented in `/etc/dahdi/modules`.

If it wasn't, do again the step *Load the correct DAHDI modules*.

Configure

To configure the card you have to:

1. Install the card firmware:

```
xivo-fetchfw install digium-tc400m
```

2. Comment out the following line in `/etc/asterisk/modules.conf`:

```
noload = codec_dahdi.so
```

3. Restart asterisk:

```
service asterisk restart
```

Next step

Now that you have configured your Voice Compression card:

1. you must check if you need to follow one of the *Specific configuration* sections below,
2. then, if you have another type of card to configure, you can go back to the *configure your card* section.

Specific configuration

Select the transcoding mode

The Digium TC400 card can be used to transcode:

- 120 G.729a channels,
- 92 G.723.1 channels,
- or 92 G.729a/G.723.1 channels.

Depending on the codec you want to transcode, you can modify the `mode` parameter which can take the following value:

- mode = mixed : this the default value which activates transcoding for 92 channels in G.729a or G.723.1 (5.3 Kbit and 6.3 Kbit)
- mode = g729 : this option activates transcoding for 120 channels in G.729a
- mode = g723 : this option activates transcoding for 92 channels in G.723.1 (5.3 Kbit et 6.3 Kbit)

1. Create the file `/etc/modprobe.d/xivo-transcode.conf`:

```
touch /etc/modprobe.d/xivo-transcode.conf
```

2. And insert the following lines:

```
options wctc4xxp mode=g729
```

3. Apply the configuration by restarting the services:

```
wazo-service restart
```

4. Verify that the card is correctly seen by asterisk with the `transcoder show` CLI command - this command should show the encoders/decoders registered by the TC400 card:

```
*CLI> transcoder show
0/0 encoders/decoders of 120 channels are in use.
```

Apply configuration

If you didn't do it already, you have to restart the services to apply the configuration:

```
wazo-service restart
```

At the end of this page you will also find some general notes and DAHDI.

Notes on configuration files

/etc/dahdi/system.conf

A *span* is created for each card port. Below is an example of a standard E1 port:

```
span=1,1,0,ccs,hdb3
dchan=16
bchan=1-15,17-31
echocanceller=mg2,1-15,17-31
```

Each span has to be declared with the following information:

```
span=<spannum>,<timing>,<LBO>,<framing>,<coding>[,crc4]
```

- `spannum` : corresponds to the span number. It starts to 1 and has to be incremented by 1 at each new span. This number MUST be unique.
- `timing` : describes the how this span will be considered regarding the synchronization :
  - 0 : do not use this span as a synchronization source,
  - 1 : use this span as the primary synchronization source,
  - 2 : use this span as the secondary synchronization source etc.
- `LBO` : 0 (not used)
- `framing` : correct values are `ccs` or `cas`. For ISDN lines, `ccs` is used.
- `coding` : correct values are `hdb3` or `ami`. For example, `hdb3` is used for an E1 (PRI) link, whereas `ami` is used for T0 (french BRI) link.

- `crc4` : this is a framing option for PRI lines. For example it is rarely use in France.

Note that the `dahdi_genconf` command should usually give you the correct parameters (if you correctly set the cards jumper). All these information should be checked with your operator.

/etc/asterisk/chan_dahdi.conf

This file contains the general parameters of the DAHDI channel. It is not generated via the `dahdi_genconf` command.

/etc/asterisk/dahdi-channels.conf

This file contains the parameters of each channel. It is generated via the `dahdi_genconf` command.

Below is an example of span definition:

```
group=0,11
context=from-extern
switchtype = euroisdn
signalling = pri_cpe
channel => 1-15,17-31
```

Note that parameters are read from top to bottom in a last match fashion and are applied to the given channels when it reads a line `channel =>`.

Here the channels 1 to 15 and 17 to 31 (it is a typical E1) are set:

- in groups 0 and 11 (see *DAHDI interconnections*)
- in context `from-extern` : all calls received on these channels will be sent in the context `from-extern`
- and configured with switchtype `euroisdn` and signalling `pri_cpe`

Debug

Check IRQ misses

It's always useful to verify if there isn't any *missed IRQ* problem with the cards.

Check:

```
cat /proc/dahdi/<span number>
```

If the *IRQ misses* counter increments, it's not good:

```
cat /proc/dahdi/1
Span 1: WCTDM/0 "Wildcard TDM800P Board 1" (MASTER)
IRQ misses: 1762187
  1 WCTDM/0/0 FXOKS (In use)
  2 WCTDM/0/1 FXOKS (In use)
  3 WCTDM/0/2 FXOKS (In use)
  4 WCTDM/0/3 FXOKS (In use)
```

Digium gives some hints in their *Knowledge Base* here : http://kb.digium.com/entry/1/63/

PRI Digium cards needs 1000 interruption per seconds. If the system cannot supply them, it increment the IRQ missed counter.

As indicated in Digium *KB* you should avoid shared IRQ with other equipments (like HD or NIC interfaces).

1.9.21 Incall

General Configuration

You can configure incoming calls settings in *Services* → *IPBX* → *Call Management* → *Incoming calls*.

DID (Direct Inward Dialing) Configuration

When a "+" character is prepended a called DID, the "+" character is discarded.

Example:

> Bob has a DID with number 1000. Alice can call Bob by dialing either 1000 or +1000, without configuring another DID.

1.9.22 Interconnections

Interconnect two Wazo directly

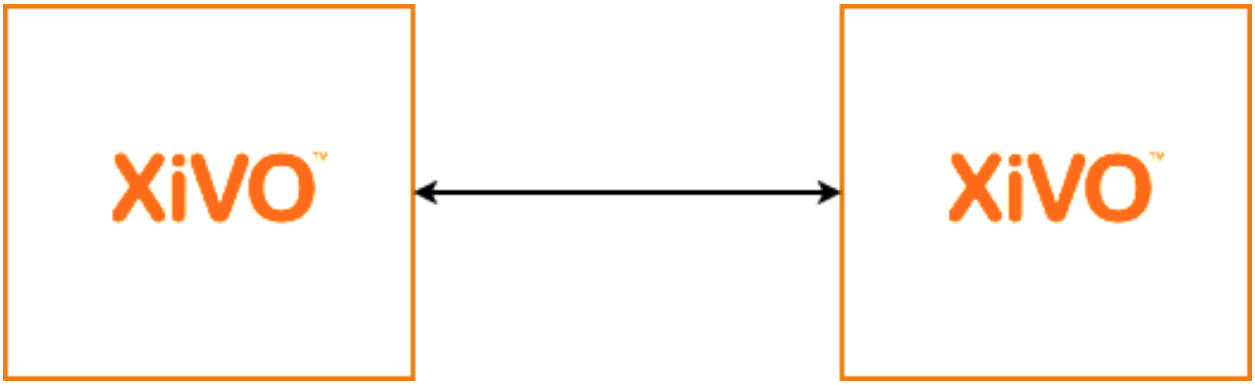


Fig. 1.65: Situation diagram

Interconnecting two Wazo will allow you to send and receive calls between the users configured on both sides.

The steps to configure the interconnections are:

- Establish the trunk between the two Wazo, that is the SIP connection between the two servers
- Configure outgoing calls on the server(s) used to emit calls
- Configure incoming calls on the server(s) used to receive calls

For now, only SIP interconnections have been tested.

Establish the trunk

The settings below allow a trunk to be used in both directions, so it doesn't matter which server is A and which is B.

Consider Wazo A wants to establish a trunk with Wazo B.

On Wazo B, go on page *Services → IPBX → Trunk management → SIP Protocol*, and create a SIP trunk:

```
Name : wazo-trunk
Username: wazo-trunk
Password: pass
Connection type: Friend
IP addressing type: Dynamic
Context: <see below>
```

Note: For the moment, Name and Username need to be the same string.

The `Context` field will determine which extensions will be reachable by the other side of the trunk:

- If `Context` is set to `default`, then every user, group, conf room, queue, etc. that have an extension if the `default` context will be reachable directly by the other end of the trunk. This setting can ease configuration if you manage both ends of the trunk.
- If you are establishing a trunk with a provider, you probably don't want everything to be available to everyone else, so you can set the `Context` field to `Incalls`. By default, there is no extension available in this context, so we will be able to configure which extension are reachable by the other end. This is the role of the incoming calls: making bridges from the `Incalls` context to other contexts.

On Wazo A, create the other end of the SIP trunk on the *Services → IPBX → Trunk management → SIP Protocol*:

```
Name: wazo-trunk
Username: wazo-trunk
Password: pass
Identified by: Friend
Connection type: Static
Address: <Wazo B IP address or hostname>
Context: Incalls
```

Register tab:

```
Register: checked
Transport: udp
Username: wazo-trunk
Password: pass
Remote server: <Wazo B IP address or hostname>
```

On both Wazo, activate some codecs, *Services → IPBX → General Settings → SIP protocol*, tab `Signaling`:

```
Enabled codecs: at least GSM (audio)
```

At that point, the Asterisk command `sip show registry` on Wazo B should print a line showing that Wazo A is registered, meaning your trunk is established.

Set the outgoing calls

The outgoing calls configuration will allow Wazo to know which extensions will be called through the trunk.

On the call emitting server(s), go on the page *Services → IPBX → Call management → Outgoing calls* and add an outgoing call.

Tab General:

```
Trunks: wazo-trunk
```

Tab Exten:

```
Exten: **99. (note the period at the end)
Stripnum: 4
```

This will tell Wazo: if any extension begins with `**99`, then try to dial it on the trunk `wazo-trunk`, after removing the 4 first characters (the `**99` prefix).

The most useful special characters to match extensions are:

```
. (period): will match one or more characters
X: will match only one character
```

You can find more details about pattern matching in Asterisk (hence in Wazo) on the Asterisk wiki.

Set the incoming calls

Now that we have calls going out from a Wazo, we need to route incoming calls on the Wazo destination.

Note: This step is only necessary if the trunk is linked to an Incoming calls context.

To route an incoming call to the right destination in the right context, we will create an incoming call in *Services* → *IPBX* → *Call management* → *Incoming calls*.

Tab General:

```
DID: 101
Context: Incalls
Destination: User
Redirect to: someone
```

This will tell Wazo: if you receive an incoming call to the extension `101` in the context `Incalls`, then route it to the user `someone`. The destination context will be found automatically, depending on the context of the line of the given user.

So, with the outgoing call set earlier on Wazo A, and with the incoming call above set on Wazo B, a user on Wazo A will dial `**99101`, and the user `someone` will ring on Wazo B.

Interconnect a Wazo to a VoIP provider

When you want to send and receive calls to the global telephony network, one option is to subscribe to a VoIP provider. To receive calls, your Wazo needs to tell your provider that it is ready and to which IP the calls must be sent. To send calls, your Wazo needs to authenticate itself, so that the provider knows that your Wazo is authorized to send calls and whose account must be credited with the call fare.

The steps to configure the interconnections are:

- Establish the trunk between the two Wazo, that is the SIP connection between the two servers
- Configure outgoing calls on the server(s) used to emit calls
- Configure incoming calls on the server(s) used to receive calls

Establish the trunk

You need the following information from your provider:

- a username
- a password
- the name of the provider VoIP server
- a public phone number

On your Wazo, go on page *Services* → *IPBX* → *Trunk management* → *SIP Protocol*, and create a SIP/IAX trunk:

```
Name : provider_username
Username: provider_username
Password: provider_password
Connection type: Peer
IP addressing type: voip.provider.example.com
Context: Incalls (or another incoming call context)
```

Register tab:

```
Register: checked
Transport: udp
Name: provider_username
Username: provider_username
Password: provider_password
Remote server: voip.provider.example.com
```

Note: For the moment, Name and Username need to be the same value.

If your Wazo is behind a NAT device or a firewall, you should set the following:

```
Monitoring: Yes
```

This option will make Asterisk send a signal to the VoIP provider server every 60 seconds (default settings), so that NATs and firewall know the connection is still alive. If you want to change the value of this cycle period, you have to select the appropriate value of the following parameter:

```
Qualify Frequency:
```

At that point, the Asterisk command `sip show registry` should print a line showing that you are registered, meaning your trunk is established.

Set the outgoing calls

The outgoing calls configuration will allow Wazo to know which extensions will be called through the trunk.

Go on the page *Services* → *IPBX* → *Call management* → *Outgoing calls* and add an outgoing call.

Tab General:

```
Trunks: provider_username
```

Tab Exten:

```
Exten: 418. (note the period at the end)
```

This will tell Wazo: if an internal user dials a number beginning with `418`, then try to dial it on the trunk `provider_username`.

The most useful special characters to match extensions are:

```
. (period): will match one or more characters
X: will match only one character
```

You can find more details about pattern matching in Asterisk (hence in Wazo) on the Asterisk wiki.

Set the incoming calls

Now that we have calls going out, we need to route incoming calls.

To route an incoming call to the right destination in the right context, we will create an incoming call in *Services* → *IPBX* → *Call management* → *Incoming calls*.

Tab General:

```
DID: your_public_phone_number
Context: Incalls (the same than configured in the trunk)
Destination: User
Redirect to: the_front_desk_guy
```

This will tell Wazo: if you receive an incoming call to the public phone number in the context `Incalls`, then route it to the user `the_front_desk_guy`. The destination context will be found automatically, depending on the context of the line of the given user.

Interconnect a Wazo to a PBX via an ISDN link

The goal of this architecture can be one of:

- start a smooth migration between an old telephony system towards IP telephony with Wazo
- bring new features to the PBX like voicemail, conference, IVR etc.

First, Wazo is to be integrated transparently between the operator and the PBX. Then users or features are to be migrated from the PBX to the Wazo.

> **Warning:** It requires a special call routing configuration on both the Wazo **and the PBX**.

Hardware

General uses

You must have an ISDN card able to support both the provider and PBX ISDN links.

Example : If you have two provider links towards the PBX, Wazo should have a 4 spans card : two towards the provider, and two towards the PBX.

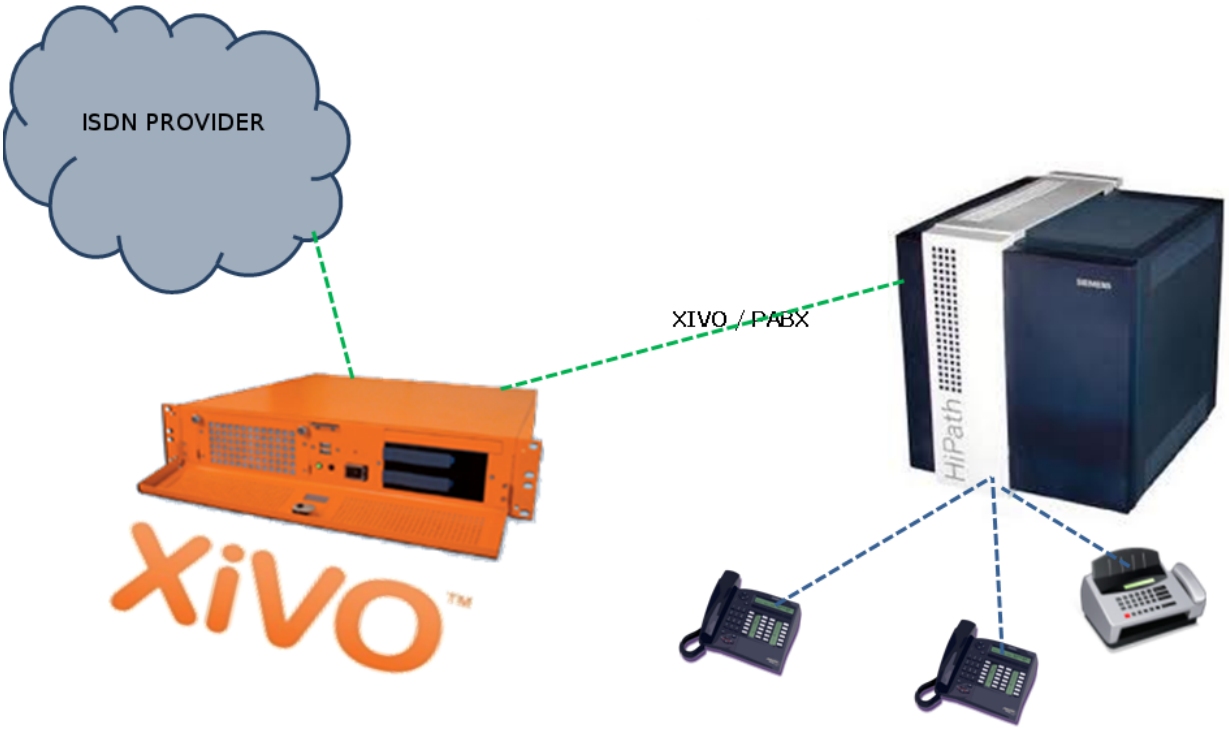


Fig. 1.66: Interconnect a Wazo to a PBX

If you use two cards

If you use two cards, you have to :

- Use a cable for clock synchronization between the cards
- Configure the *wheel* to define the cards order in the system.

Please refer to the section *Sync cable*

Configuration

You have now to configure two files :

1. `/etc/dahdi/system.conf`
2. `/etc/asterisk/dahdi-channels.conf`

system.conf

You mainly need to configure the `timing` parameter on each *span*. As a general rule :

- Provider *span* - Wazo will get the clock from the provider : the `timing` value is to be different from 0 (see */etc/dahdi/system.conf* section)
- PBX *span* - Wazo will provide the clock to the PBX : the `timing` value is to be set to 0 (see */etc/dahdi/system.conf* section)

Below is an example with two provider links and two PBX links:

```
# Span 1: TE4/0/1 "TE4XXP (PCI) Card 0 Span 1" (MASTER)
span=1,1,0,ccs,hdb3          # Span towards Provider
bchan=1-15,17-31
dchan=16
echocanceller=mg2,1-15,17-31

# Span 2: TE4/0/2 "TE4XXP (PCI) Card 0 Span 2"
span=2,2,0,ccs,hdb3          # Span towards Provider
bchan=32-46,48-62
dchan=47
echocanceller=mg2,32-46,48-62

# Span 3: TE4/0/3 "TE4XXP (PCI) Card 0 Span 3"
span=3,0,0,ccs,hdb3          # Span towards PBX
bchan=63-77,79-93
dchan=78
echocanceller=mg2,63-77,79-93

# Span 4: TE4/0/4 "TE4XXP (PCI) Card 0 Span 4"
span=4,0,0,ccs,hdb3          # Span towards PBX
bchan=94-108,110-124
dchan=109
echocanceller=mg2,94-108,110-124
```

dahdi-channels.conf

In the file `/etc/asterisk/dahdi-channels.conf` you need to adjust, for each span :

- `group` : the group number (e.g. `0` for provider links, `2` for PBX links),
- `context` : the context (e.g. `from-extern` for provider links, `from-pabx` for PBX links)
- `signalling` : `pri_cpe` for provider links, `pri_net` for PBX side

Warning: most of the PBX uses overlap dialing for some destination (digits are sent one by one instead of by block). In this case, the `overlapdial` parameter has to be activated on the PBX spans:

```
overlapdial = incoming
```

Below an example of `/etc/asterisk/dahdi-channels.conf`:

```
; Span 1: TE4/0/1 "TE4XXP (PCI) Card 0 Span 1" (MASTER)
group=0,11
context=from-extern
switchtype = euroisdn
signalling = pri_cpe
channel => 1-15,17-31

; Span 2: TE4/0/2 "TE4XXP (PCI) Card 0 Span 2"
group=0,12
context=from-extern
switchtype = euroisdn
signalling = pri_cpe
channel => 32-46,48-62

; PBX link #1
; Span 3: TE4/0/3 "TE2XXP (PCI) Card 0 Span 3"
group=2,13
context=from-pabx     ; special context for PBX incoming calls
overlapdial=incoming  ; overlapdial activation
switchtype = euroisdn
signalling = pri_net  ; behave as the NET termination
channel => 63-77,79-93

; PBX link #2
; Span 4: TE4/0/4 "T4XXP (PCI) Card 0 Span 4"
group=2,14
context=from-pabx     ; special context for PBX incoming calls
overlapdial=incoming  ; overlapdial activation
switchtype = euroisdn
signalling = pri_net  ; behave as the NET termination
channel => 94-108,110-124
```

Passthru function

Route PBX incoming calls

We first need to create a route for calls coming from the PBX

Create a file named `pbx.conf` in the directory `/etc/asterisk/extensions_extra.d/`, # Add the following lines in the file:

```
[from-pabx]
exten = _X.,1,NoOp(### Call from PBX ${CARLLERID(num)} towards ${EXTEN} ###)
exten = _X.,n,Goto(default,${EXTEN},1)
```

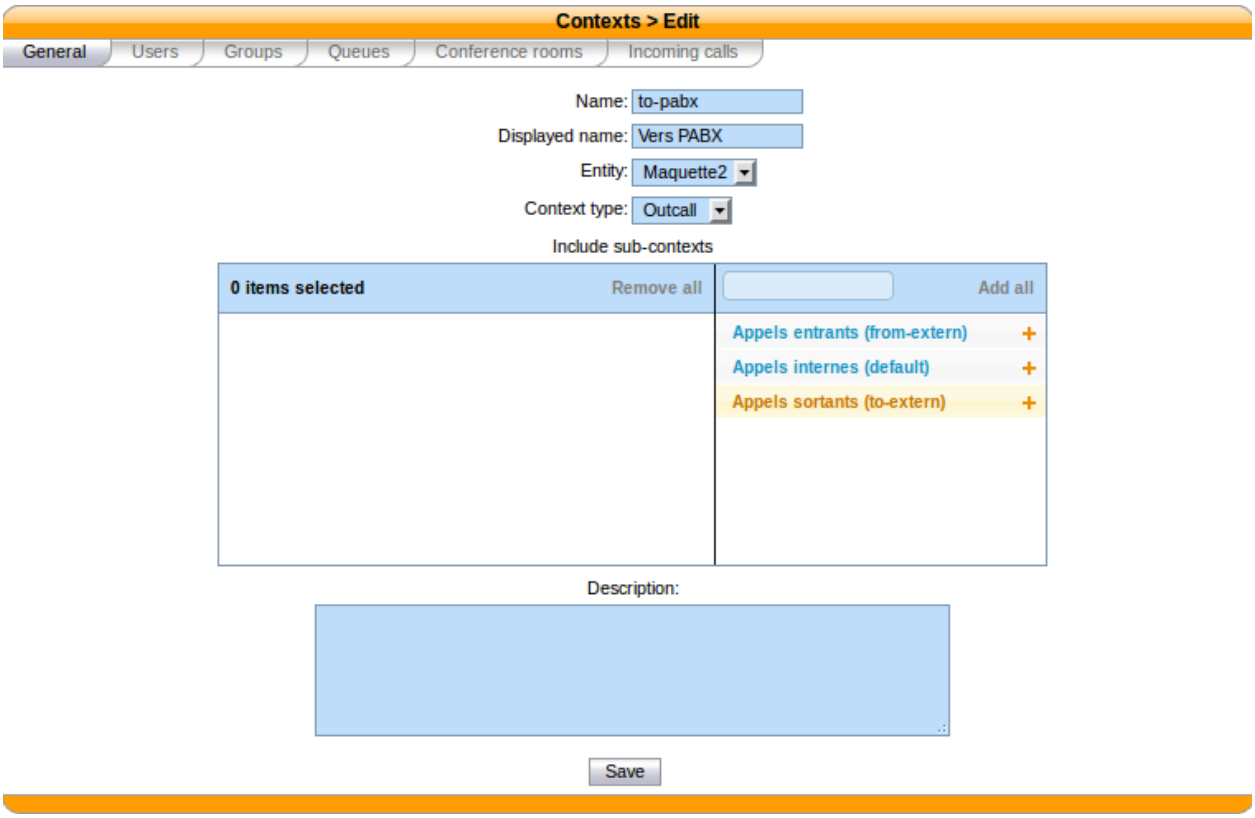
This dialplan routes incoming calls from the PBX in the `default` context of Wazo. It enables call from the PBX : * towards a SIP phone (in `default` context) * towards outgoing destniation (via the `to-extern` context included in `default` context)

Create the to-pabx context

In the webi, create a context named `to-pabx`:

- Name : to-pabx
- Display Name : TO PBX
- Context type : Outcall
- Include sub-contexts : No context inclusion

This context will permit to route incoming calls from the Wazo to the PBX.

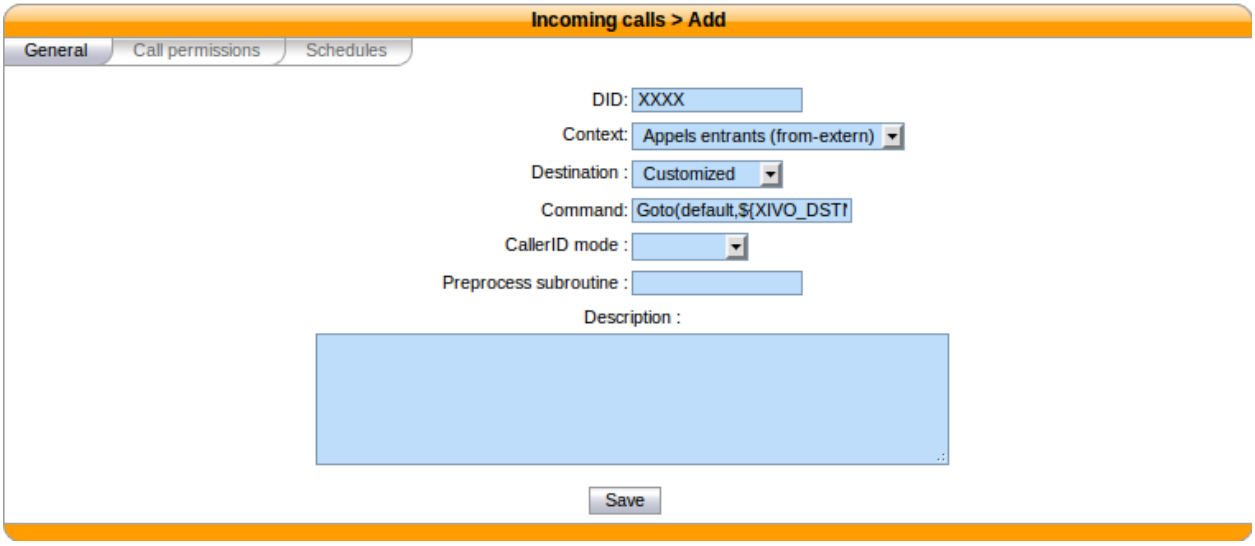


Route incoming calls to PBX

In our example, incoming calls on spans 1 and 2 (spans pluged to the provider) are routed by from-extern context. We are going to create a default route to redirect incoming calls to the PBX.

Create an incoming call as below :

- DID : XXXX (according to the number of digits sent by the provider)
- Context : Incoming calls
- Destination : Customized
- Command : Goto(to-pabx,${XIVO_DSTNUM},1)



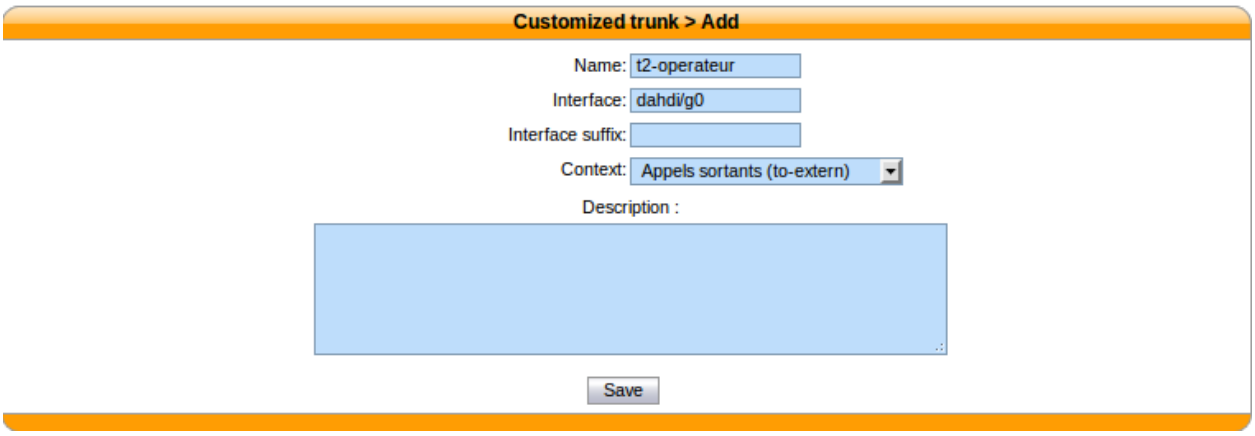
Create the interconnections

You have to create two interconnections :

- provider side : dahdi/g0
- PBX side : dahdi/g2

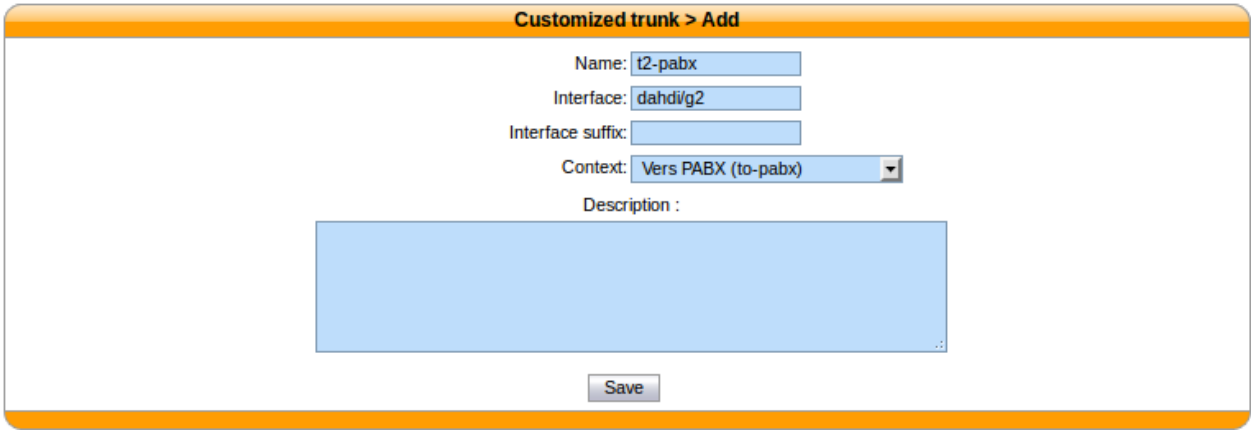
In the menu *Services* → *IPBX* → *Trunk management* → *Customized* page :

- Name : t2-operateur
- Interface : dahdi/g0
- Context : to-extern



The second interconnection :

- Name : t2-pabx
- Interface : dahdi/g2
- Context : to-pabx

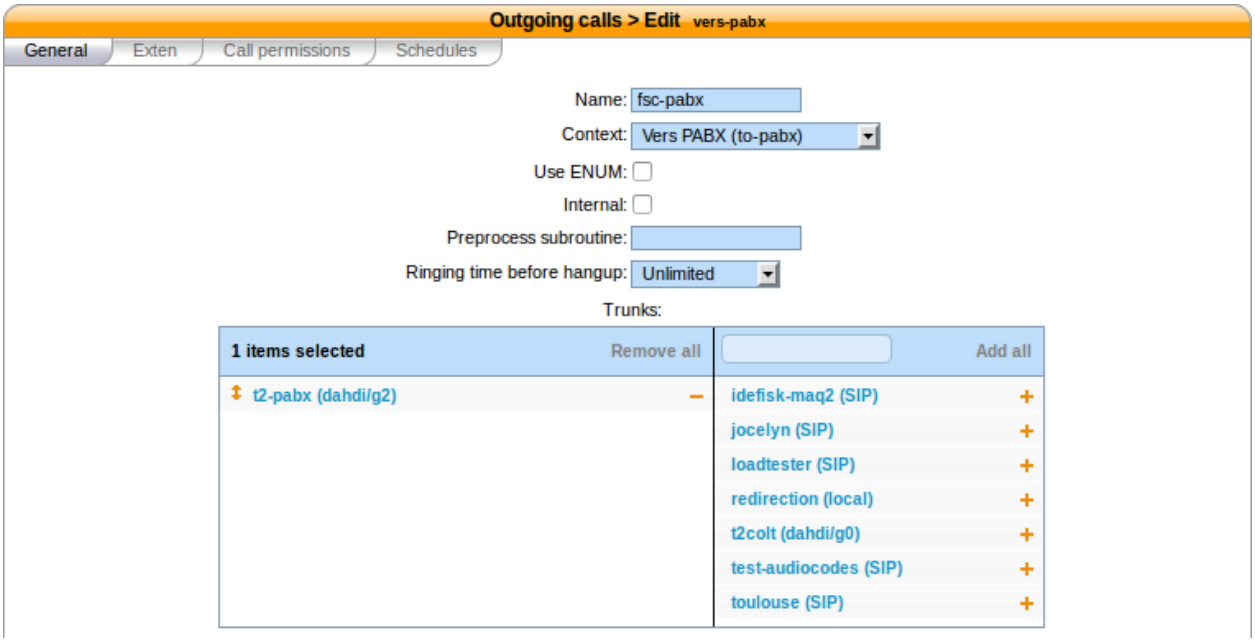


Create outgoing calls

You must create two rules of outgoing calls in the menu *Services* → *IPBX* → *Call management* → *Outgoing calls* page :

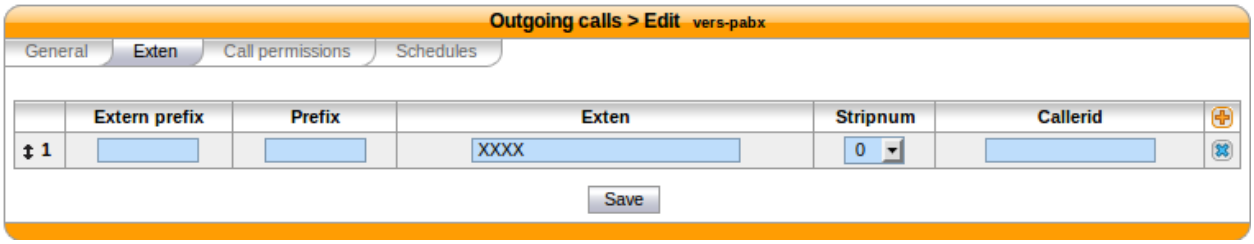
1. Redirect calls to the PBX :

- Name : fsc-pabx
- Context : to-pabx
- Trunks : choose the *t2-pabx* interconnection



In the extensions tab :

- Exten : XXXX



2. Create a rule “fsc-operateur”:

- Name : fsc-operateur
- Context : to-extern
- Trunks : choose the “t2-operateur” interconnection

In the extensions tab:

```
exten = X.
```

Specific VoIP providers

Simon Telephonics

The following configuration is based on the example found here

- username: `GV18005551212`
- password: `password`
- exten: `18005551212`
- host: `gvgw.simonics.com`

General SIP configuration

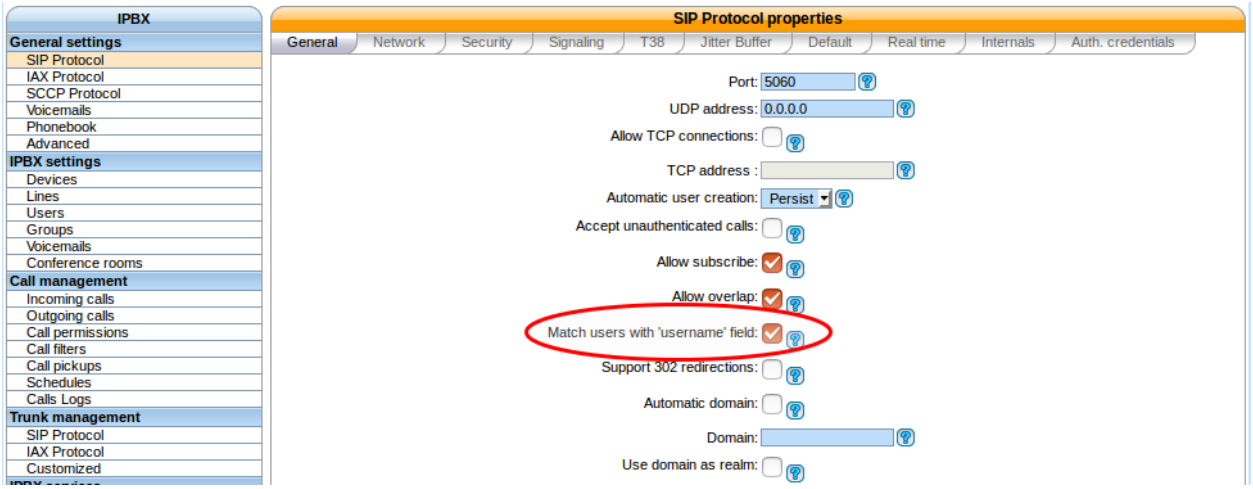
Under *Services → IPBX → General settings → SIP Protocol*.

- General
  - Match users with ‘username’ field: *checked*

Trunk settings

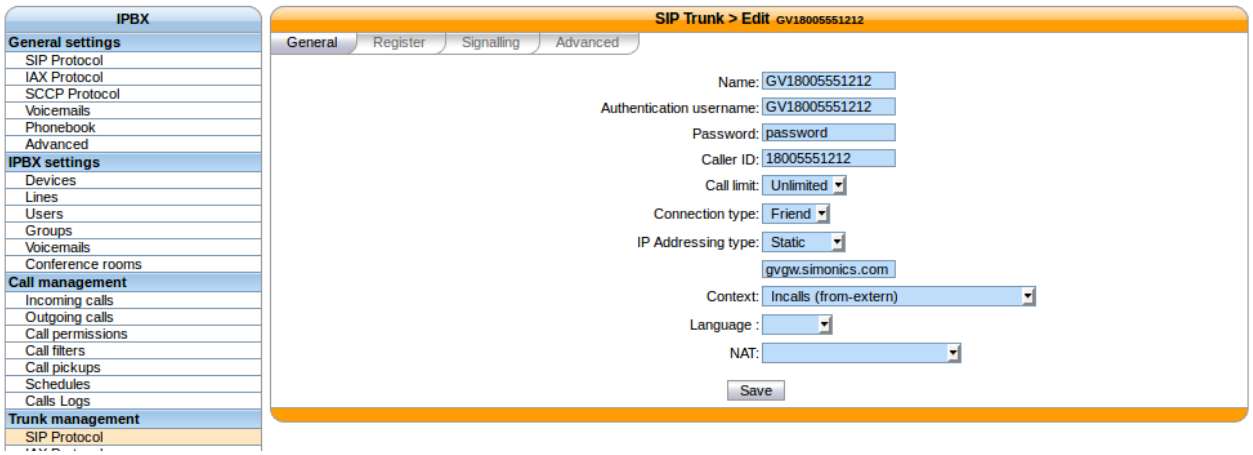
Under *Services → IPBX → Trunk management → SIP Protocol*.

- General
  - Name: `GV18005551212`
  - Authentication username: `GV18005551212`
  - Password: `password`

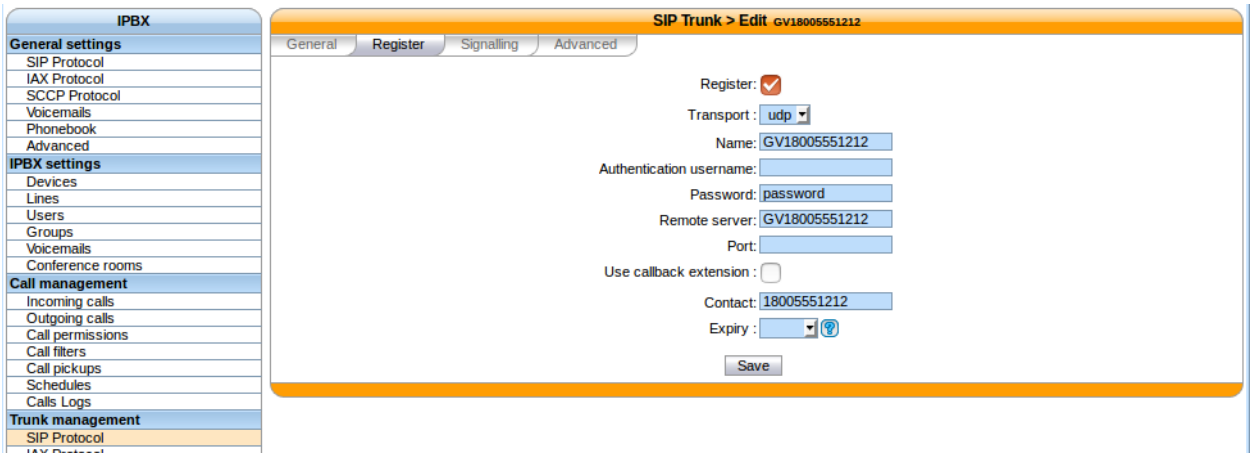


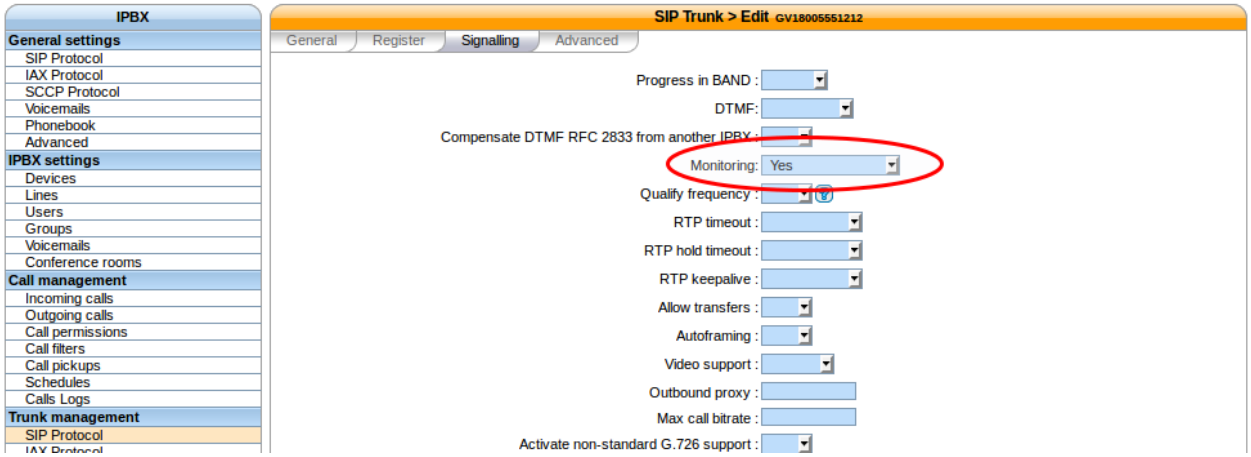
  - Caller ID: `18005551212`
  - Connection type: `Friend`
  - IP Addressing type: `static`

    * `gvgw.simonics.com`
  - Context: `<your incoming call context>`



- Register
  - Register: `checked`
  - Transport: `UDP`
  - Name: `GV18005551212`
  - Password: `password`
  - Remote Server: `GV18005551212`
  - Contact: `18005551212`
- Signaling
  - Monitoring: `Yes`


IPBX
General settings
SIP Protocol
IAX Protocol
SCCP Protocol
Voicemails
Phonebook
Advanced
IPBX settings
Devices
Lines
Users
Groups
Voicemails
Conference rooms
Call management
Incoming calls
Outgoing calls
Call permissions
Call filters
Call pickups
Schedules
Calls Logs
Trunk management
SIP Protocol
SIP Trunk > Edit GV18005551212
General
Register
Signalling
Advanced
Register:
Transport : udp
Name: GV18005551212
Authentication username:
Password: password
Remote server: GV18005551212
Port:
Use callback extension :
Contact: 18005551212
Expiry :
Save


IPBX
General settings
SIP Protocol
IAX Protocol
SCCP Protocol
Voicemails
Phonebook
Advanced
IPBX settings
Devices
Lines
Users
Groups
Voicemails
Conference rooms
Call management
Incoming calls
Outgoing calls
Call permissions
Call filters
Call pickups
Schedules
Calls Logs
Trunk management
SIP Protocol
SIP Trunk > Edit GV18005551212
General
Register
Signalling
Advanced
Progress in BAND :
DTMF:
Compensate DTMF RFC 2833 from another IPBX :
Monitoring: Yes
Qualify frequency :
RTP timeout :
RTP hold timeout :
RTP keepalive :
Allow transfers :
Autoframing :
Video support :
Outbound proxy :
Max call bitrate :
Activate non-standard G.726 support :

Outgoing calls

See the *Set the outgoing calls* section.

Incoming calls

See the *Set the incoming calls* section.

Create an interconnection

There are three types of interconnections :

- Customized
- SIP
- IAX

SIP interconnections

SIP interconnections are used to connect to a SIP provider to to another PBX that is part of your telecom infrastructure.

General SIP configurations are available in *General Settings* → *SIP Protocol* and trunk configurations are in *Trunk Management* → *SIP Protocol*

Environment with NAT

There are some configuration steps that are required when connecting to a SIP provider from a NAT environment.

In *General Settings* → *SIP Protocol* → *Network* set your *External IP address* and your *Local network*.

- External IP address: This is your public IP address
- Local network: Your internal network range

In *Trunk Management* → *SIP Protocol* set the *NAT* option to *Yes* and the *Monitoring* option to *Yes*.

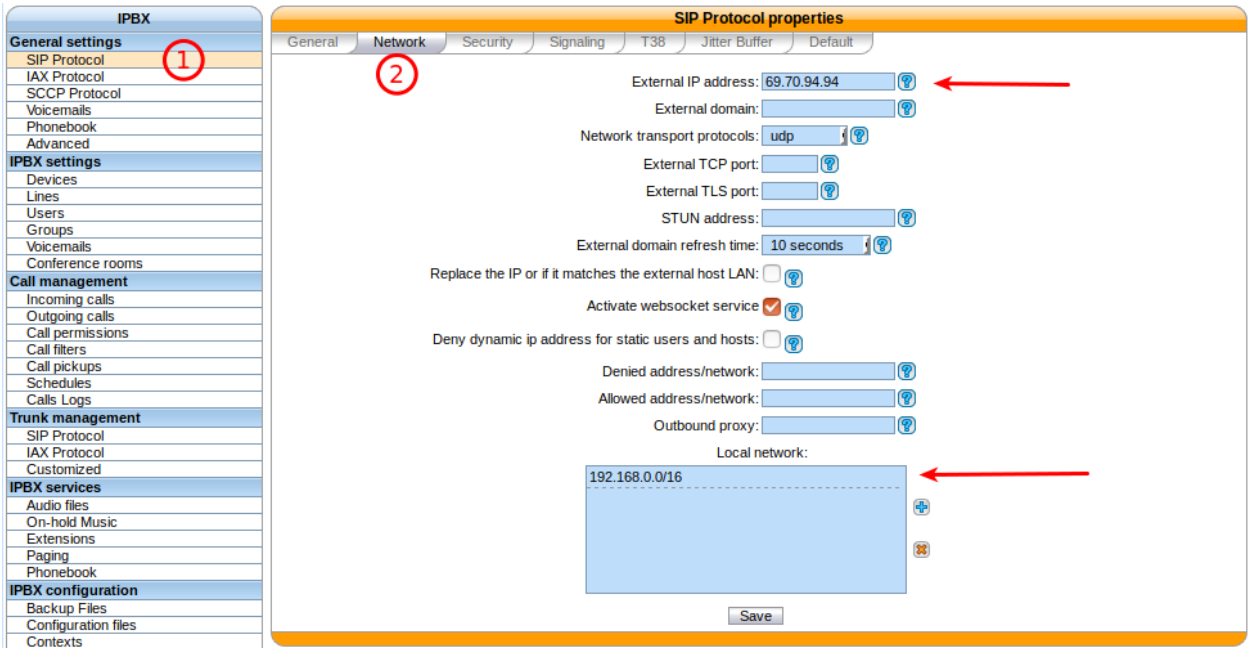
Customized interconnections

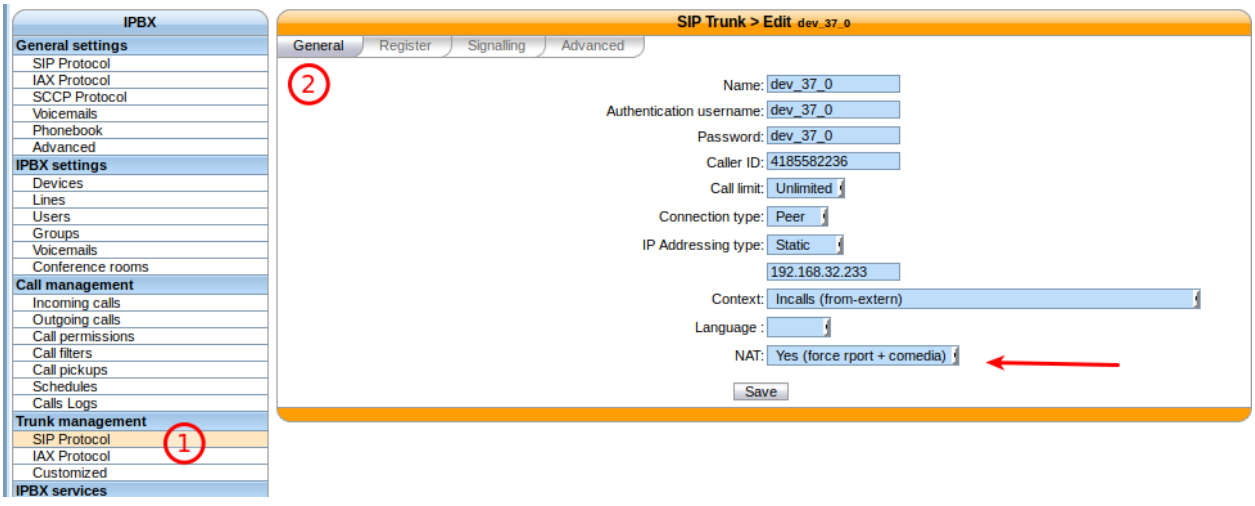
Customized interconnections are mainly used for interconnections using DAHDI or Local channels:

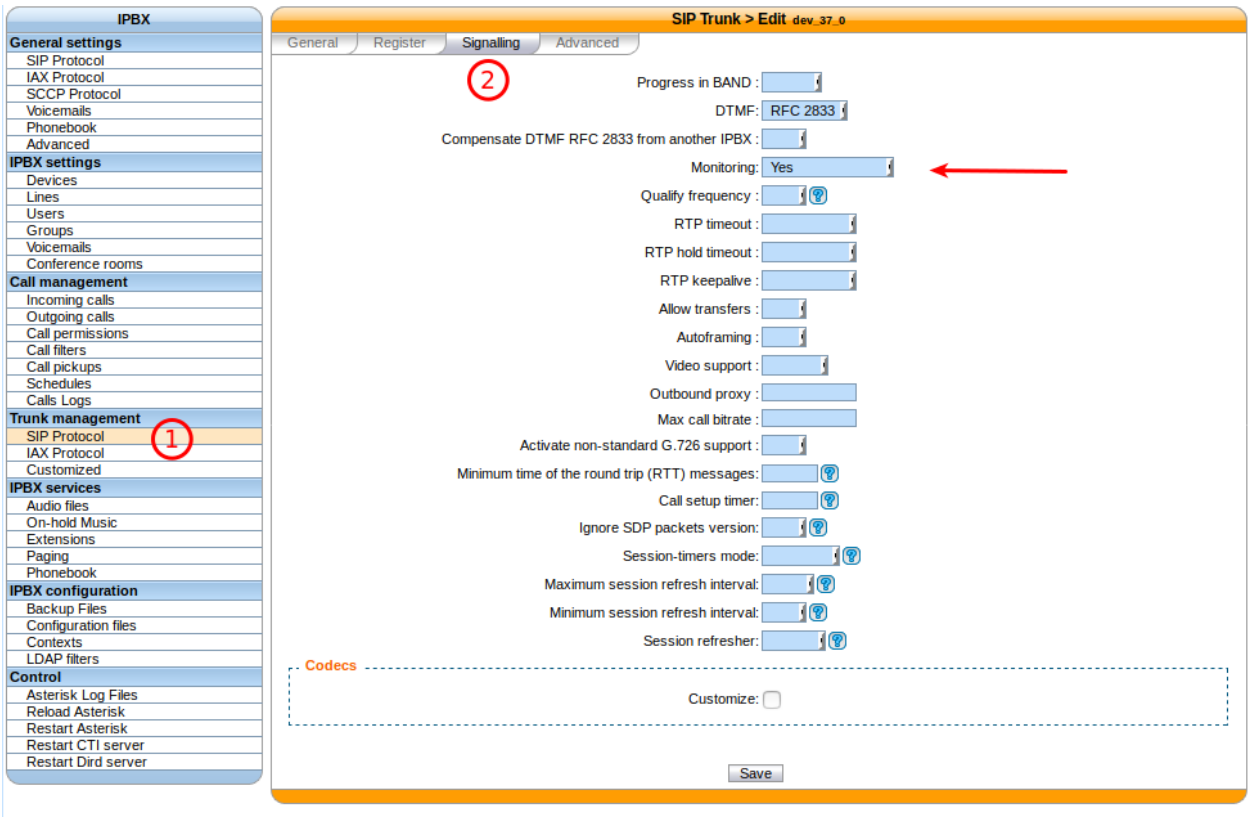
- *Name* : it is the name which will appear in the outcall interconnections list,
- *Interface* : this is the channel name (for DAHDI see *DAHDI interconnections*)
- *Interface suffix* (optional) : a suffix added after the dialed number (in fact the Dial command will dial:

```
<Interface>/<EXTEN><Interface suffix>
```

- *Context* : currently not relevant


IPBX
General settings
SIP Protocol
IAX Protocol
SCCP Protocol
Voicemails
Phonebook
Advanced
IPBX settings
Devices
Lines
Users
Groups
Voicemails
Conference rooms
Call management
Incoming calls
Outgoing calls
Call permissions
Call filters
Call pickups
Schedules
Calls Logs
Trunk management
SIP Protocol
IAX Protocol
Customized
IPBX services
Audio files
On-hold Music
Extensions
Paging
Phonebook
IPBX configuration
Backup Files
Configuration files
Contexts
1
SIP Protocol properties
General
Network
Security
Signaling
T38
Jitter Buffer
Default
2
External IP address: 69.70.94.94
External domain:
Network transport protocols: udp
External TCP port:
External TLS port:
STUN address:
External domain refresh time: 10 seconds
Replace the IP or if it matches the external host LAN:
Activate websocket service
Deny dynamic ip address for static users and hosts:
Denied address/network:
Allowed address/network:
Outbound proxy:
Local network:
192.168.0.0/16
Save


IPBX
General settings
SIP Protocol
IAX Protocol
SCCP Protocol
Voicemails
Phonebook
Advanced
IPBX settings
Devices
Lines
Users
Groups
Voicemails
Conference rooms
Call management
Incoming calls
Outgoing calls
Call permissions
Call filters
Call pickups
Schedules
Calls Logs
Trunk management
SIP Protocol
IAX Protocol
Customized
IPBX services
1
SIP Trunk > Edit dev_37_0
General
Register
Signalling
Advanced
2
Name: dev_37_0
Authentication username: dev_37_0
Password: dev_37_0
Caller ID: 4185582236
Call limit: Unlimited
Connection type: Peer
IP Addressing type: Static
192.168.32.233
Context: Incalls (from-extern)
Language :
NAT: Yes (force rport + comedia)
Save



DAHDI interconnections

To use your DAHDI links you must create a customized interconnection.

Name : the name of the interconnection like **e1_span1** or **bri_port1**

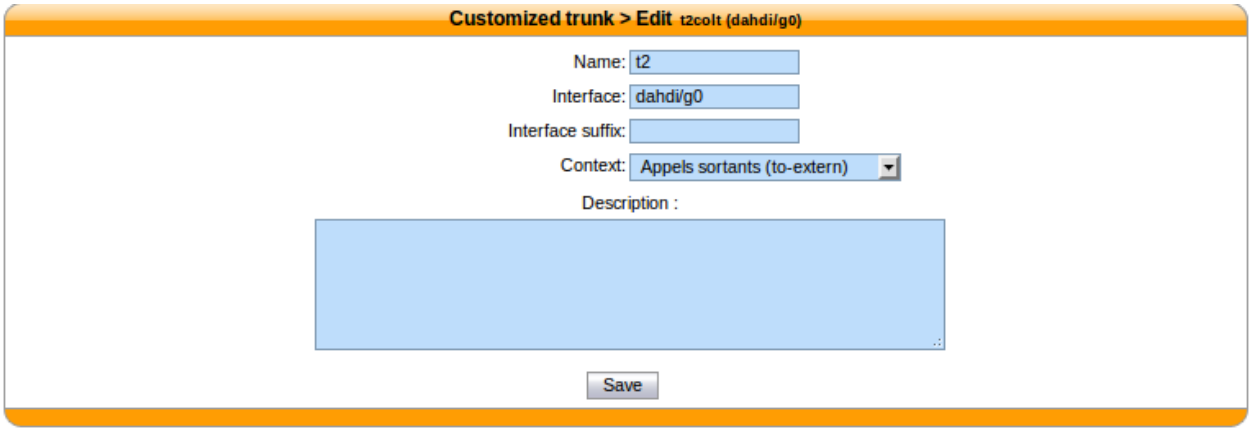
Interface : must be of the form `dahdi/[group order][group number]` where :

- `group order` is one of :
  - `g` : pick the first available channel in group, searching from lowest to highest,
  - `G` : pick the first available channel in group, searching from highest to lowest,
  - `r` : pick the first available channel in group, going in round-robin fashion (and remembering where it last left off), searching from lowest to highest,
  - `R` : pick the first available channel in group, going in round-robin fashion (and remembering where it last left off), searching from highest to lowest.
- `group number` is the group number to which belongs the span as defined in the */etc/asterisk/dahdi-channels.conf*.

> **Warning:** if you use a BRI card you MUST use per-port dahdi groups. You should not use a group like g0 which spans over several spans.

For example, add an interconnection to the menu *Services* → *IPBX* → *Trunk management* → *Customized*

```
Name : interconnection name
Interface : dahdi/g0
```



Debug

Interesting Asterisk commands:

```
sip show peers
sip show registry
sip set debug on
```

Caller ID

When setting up an interconnection with the public network or another PBX, it is possible to set a caller ID in different places. Each way to configure a caller ID has it's own use case.

The format for a caller ID is the following `"My Name" <9999>` If you don't set the number part of the caller ID, the dialplan's number will be used instead. This might not be a good option in most cases.

Outgoing call caller ID

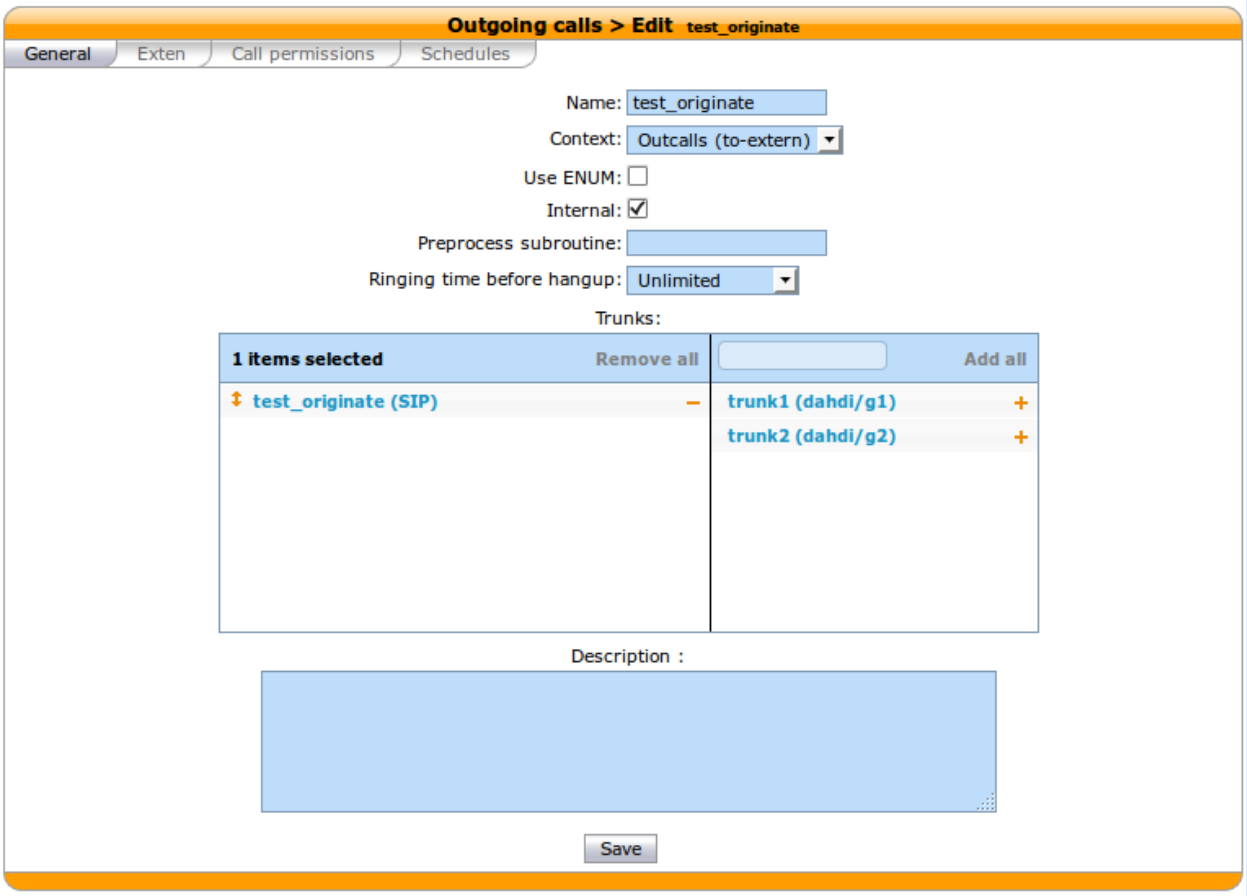
When you create an outgoing call, it's possible to set the it to internal, using the check box in the outgoing call configuration menu. When this option is activated, the caller's caller ID will be forwarded to the trunk. This option is use full when the other side of the trunk can reach the user with it's caller ID number.

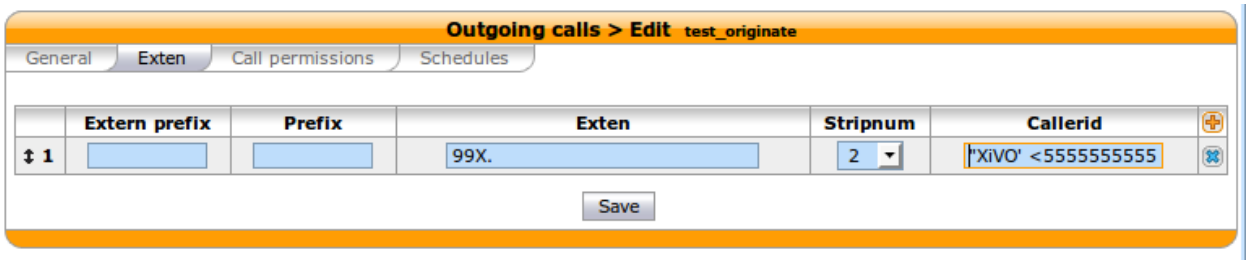
When the caller's caller ID is not usable to the called party, the outgoing call's caller id can be fixed to a given value that is more use full to the outside world. Giving the public number here might be a good idea.

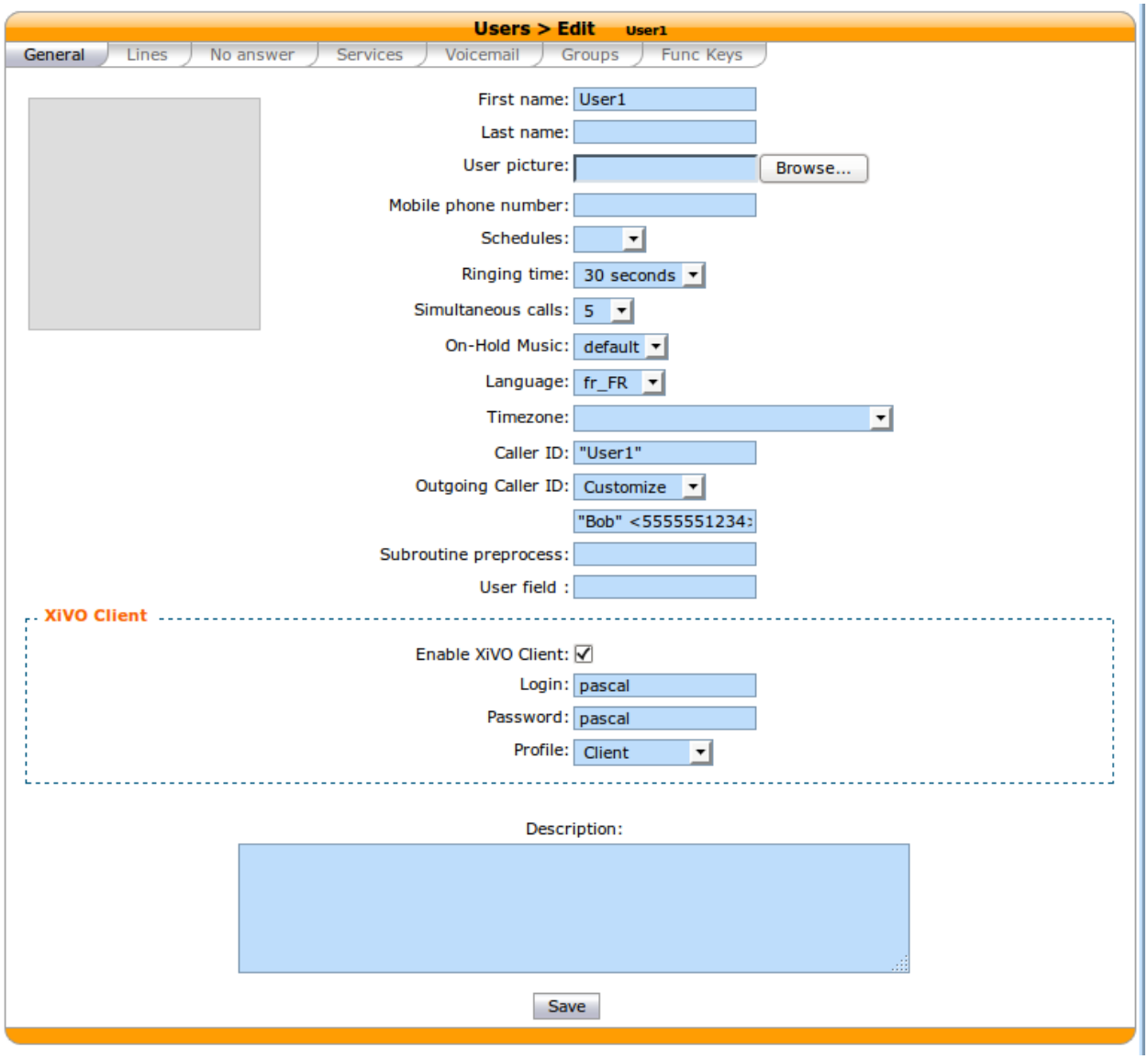
A user can also have a forced caller ID for outgoing calls. This can be use full for someone who has his own public number. This option can be set in the user's configuration page. The Outgoing Caller ID id option must be set to Customize. The user can also set his outgoing caller ID to anonymous.

The order of precedence when setting the caller ID in multiple place is the following.

1. Internal
2. User's outgoing caller ID
3. Outgoing call
4. Default caller ID


Outgoing calls > Edit test_originate
General
Exten
Call permissions
Schedules
Name: test_originate
Context: Outcalls (to-extern)
Use ENUM:
Internal:
Preprocess subroutine:
Ringing time before hangup: Unlimited
Trunks:
1 items selected
Remove all
Add all
test_originate (SIP)
trunk1 (dahdi/g1)
trunk2 (dahdi/g2)
Description :
Save


Outgoing calls > Edit test_originate
General
Exten
Call permissions
Schedules
Extern prefix
Prefix
Exten
Stripnum
Callerid
1
99X.
2
"XiVO" <5555555555
Save


Users > Edit User1
General
Lines
No answer
Services
Voicemail
Groups
Func Keys
First name: User1
Last name:
User picture:
Browse...
Mobile phone number:
Schedules:
Ringing time: 30 seconds
Simultaneous calls: 5
On-Hold Music: default
Language: fr_FR
Timezone:
Caller ID: "User1"
Outgoing Caller ID: Customize
"Bob" <5555551234>
Subroutine preprocess:
User field :
XiVO Client
Enable XiVO Client:
Login: pascal
Password: pascal
Profile: Client
Description:
Save

1.9.23 Interactive Voice Response

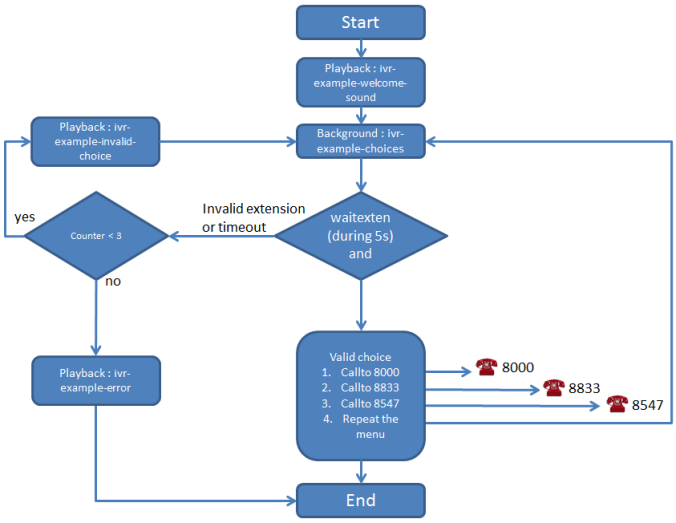
Introduction

> *Interactive voice response (IVR) is a technology that allows a computer to interact with humans through the use of voice and DTMF tones input via keypad. In telecommunications, IVR allows customers to interact with a company's host system via a telephone keypad or by speech recognition, after which they can service their own inquiries by following the IVR dialogue.*
>
> —Wikipedia

The IVR function is not yet available in graphic mode in Wazo. This functionality is currently supported only via the *xivo-confd REST API* or using scripts, also named dialplan.

Use Case: Minimal IVR

Flowchart



Configuration File and Dialplan

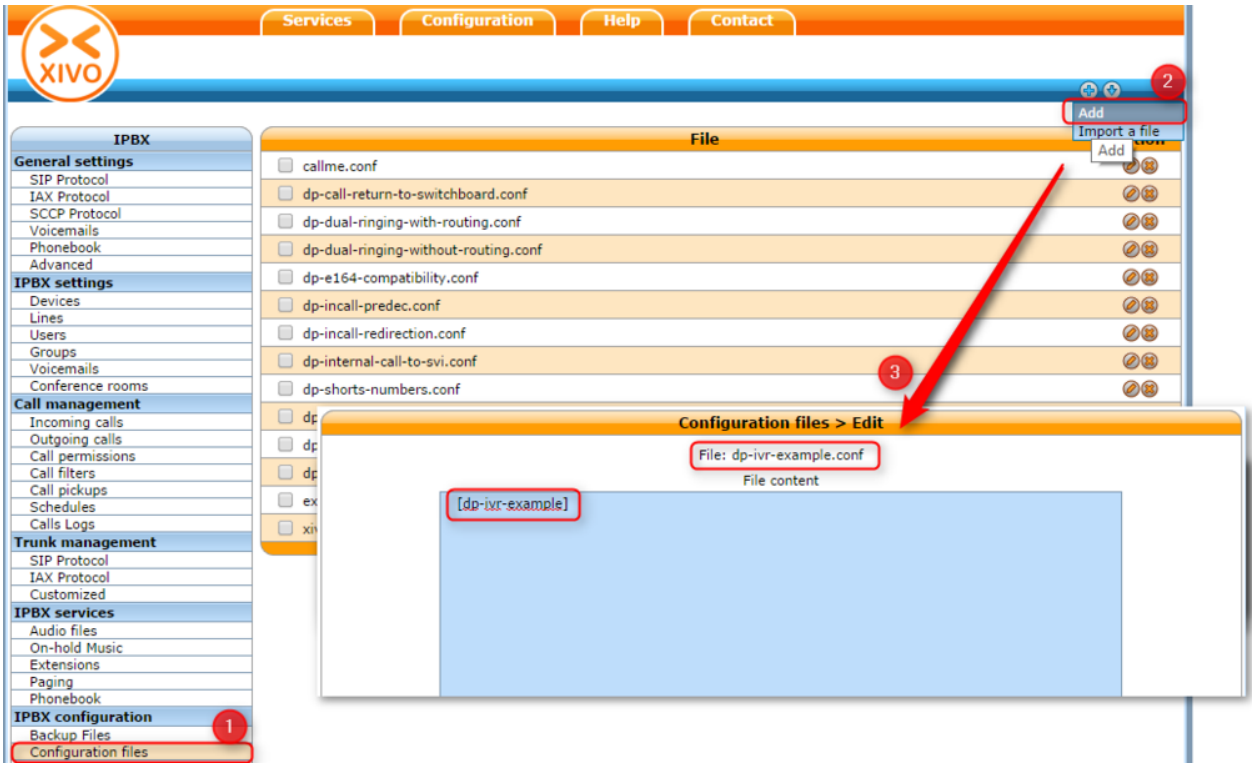
First step, you need to create a configuration file, that contain an asterisk context and your IVR dialpan. In our example, both (file and context) are named dp-ivr-example.

Copy all these lines in the newly created configuration file (in our case, dp-ivr-example) :

```
[dp-ivr-example]

exten = s,1,NoOp(### dp-ivr-example.conf ###)
same = n,NoOp(Set the context containing your ivr destinations.)
same = n,Set(IVR_DESTINATION_CONTEXT=my-ivr-destination-context)
same = n,NoOp(Set the directory containing your ivr sounds.)
same = n,Set(GV_DIRECTORY_SOUNDS=/var/lib/xivo/sounds/ivr-sounds)
same = n,NoOp(the system answers the call and waits for 1 second before continuing)
same = n,Answer(1000)

same = n,NoOp(the system plays the first part of the audio file "welcome to ...")
```




```
same = n(first),Playback(${GV_DIRECTORY_SOUNDS}/ivr-example-welcome-sound)

same = n,NoOp(variable "counter" is set to 0)
same = n(beginning),Set(counter=0)

same = n,NoOp(variable "counter" is incremented and the label "start" is defined)
same = n(start),Set(counter=$[${counter} + 1])

same = n,NoOp(counter variable is now = ${counter})
same = n,NoOp(waiting for 1 second before reading the message that indicate all␣
↪choices)
same = n,Wait(1)
same = n,NoOp(play the message ivr-example-choices that contain all choices)
same = n,Background(${GV_DIRECTORY_SOUNDS}/ivr-example-choices)
same = n,NoOp(waiting for DTMF during 5s)
same = n,Waitexten(5)

;##### CHOICE 1 #####
exten = 1,1,NoOp(pressed digit is 1, redirect to 8000 in ${IVR_DESTINATION_CONTEXT}␣
↪context)
exten = 1,n,Goto(${IVR_DESTINATION_CONTEXT},8000,1)

;##### CHOICE 2 #####
exten = 2,1,NoOp(pressed digit is 2, redirect to 8833 in ${IVR_DESTINATION_CONTEXT}␣
↪context)
exten = 2,n,Goto(${IVR_DESTINATION_CONTEXT},8833,1)

;##### CHOICE 3 #####
exten = 3,1,NoOp(pressed digit is 3, redirect to 8547 in ${IVR_DESTINATION_CONTEXT}␣
↪context)
```

```
exten = 3,n,Goto(${IVR_DESTINATION_CONTEXT},8547,1)

;##### CHOICE 4 #####
exten = 4,1,NoOp(pressed digit is 4, redirect to start label in this context)
exten = 4,n,Goto(s,start)

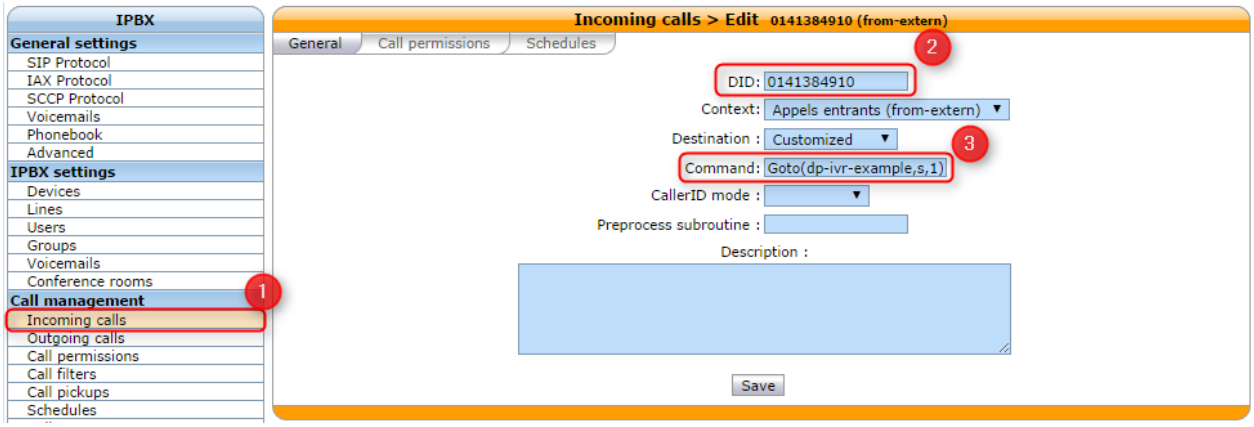
;##### TIMEOUT #####
exten = t,1,NoOp(no digit pressed for 5s, process it like an error)
exten = t,n,Goto(i,1)

;##### INVALID CHOICE #####
exten = i,1,NoOp(if counter variable is 3 or more, then goto label "error")
exten = i,n,GotoIf($[${counter}>=3]?error)
exten = i,n,NoOp(pressed digit is invalid and less than 3 errors: the guide ivr-
↪exemple-invalid-choice is now played)
exten = i,n,Playback(${GV_DIRECTORY_SOUNDS}/ivr-example-invalid-choice)
exten = i,n,Goto(s,start)
exten = i,n(error),Playback(${GV_DIRECTORY_SOUNDS}/ivr-example-error)
exten = i,n,Hangup()
```

IVR external dial

To call the script dp-ivr-example from an external phone, you must create an incoming call and redirect the call to the script dp-ivr-example with the command :

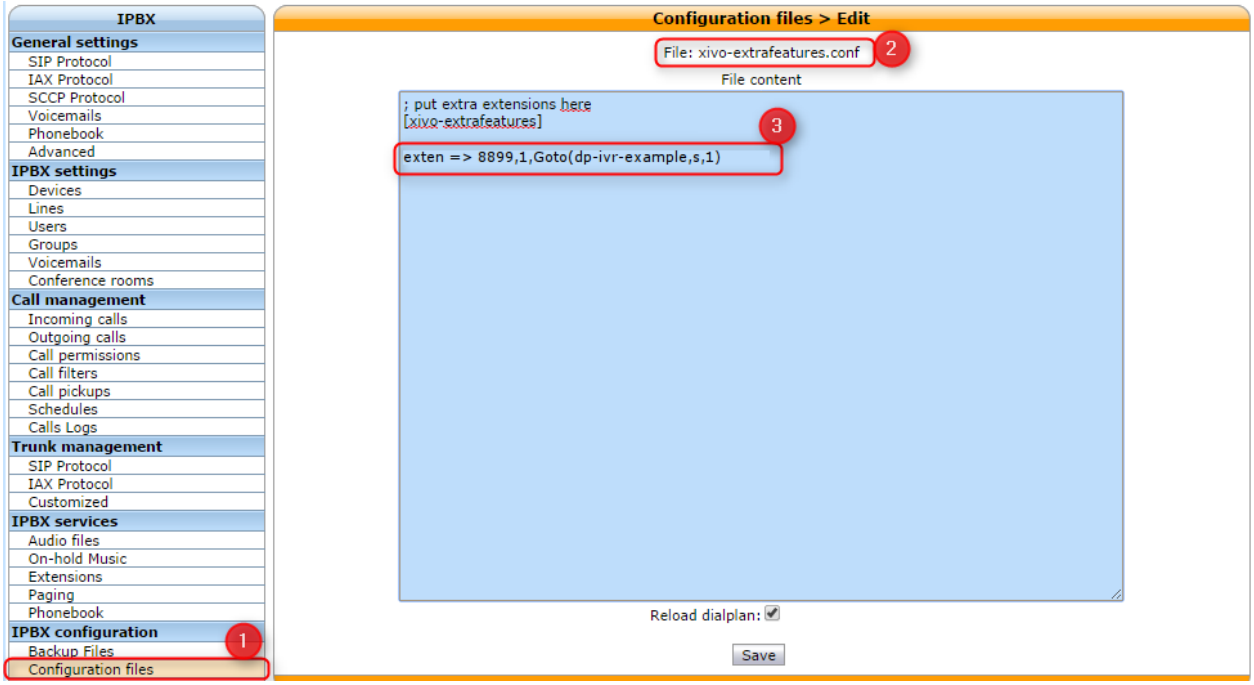
```
Goto(dp-ivr-example,s,1)
```



IVR internal dial

To call the script dp-ivr-example from an internal phone you must create an entry in the default context (`xivo-extrafeatures` is included in `default`). The best way is to add the extension in the file `xivo-extrafeatures.conf`.

```
exten => 8899,1,Goto(dp-ivr-example,s,1)
```



Use Case: IVR with a schedule

In many cases, you need to associate your IVR to a schedule to indicate when your company is closed.

Flowchart

Create Schedule

First step, create your schedule (1) from the menu *Call management* → *Schedules*. In the General tab, give a name (3) to your schedule and configure the open hours (4) and select the sound which is played when the company is closed.

In the Closed hours tab (6), configure all special closed days (7) and select the sound that indicate to the caller that the company is exceptionally closed.

The IVR script is now only available during workdays.

Assign Schedule to Incall

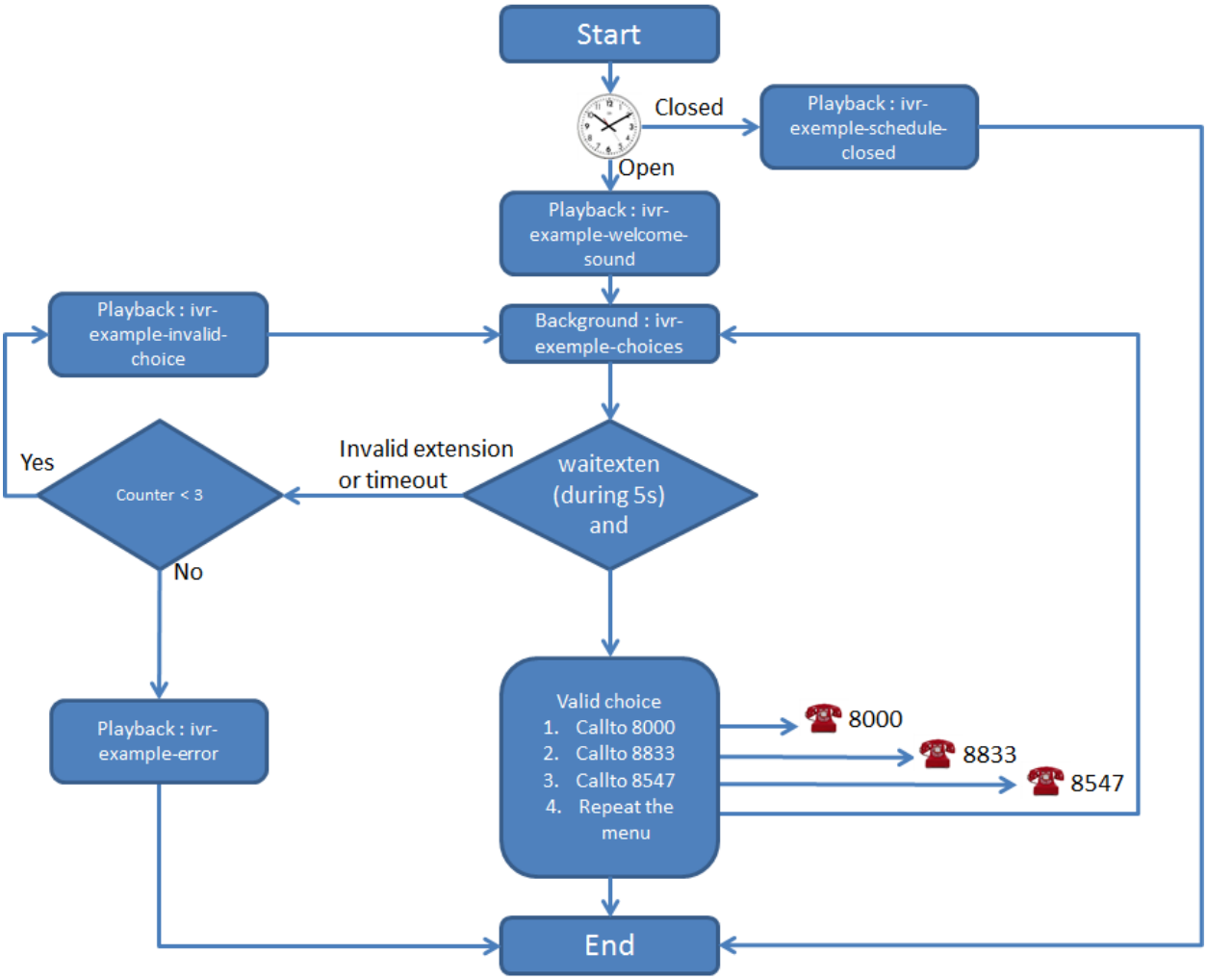
Return editing your Incall (*Call management* → *Incoming calls*) and assign the newly created schedule in the "Schedules" tab

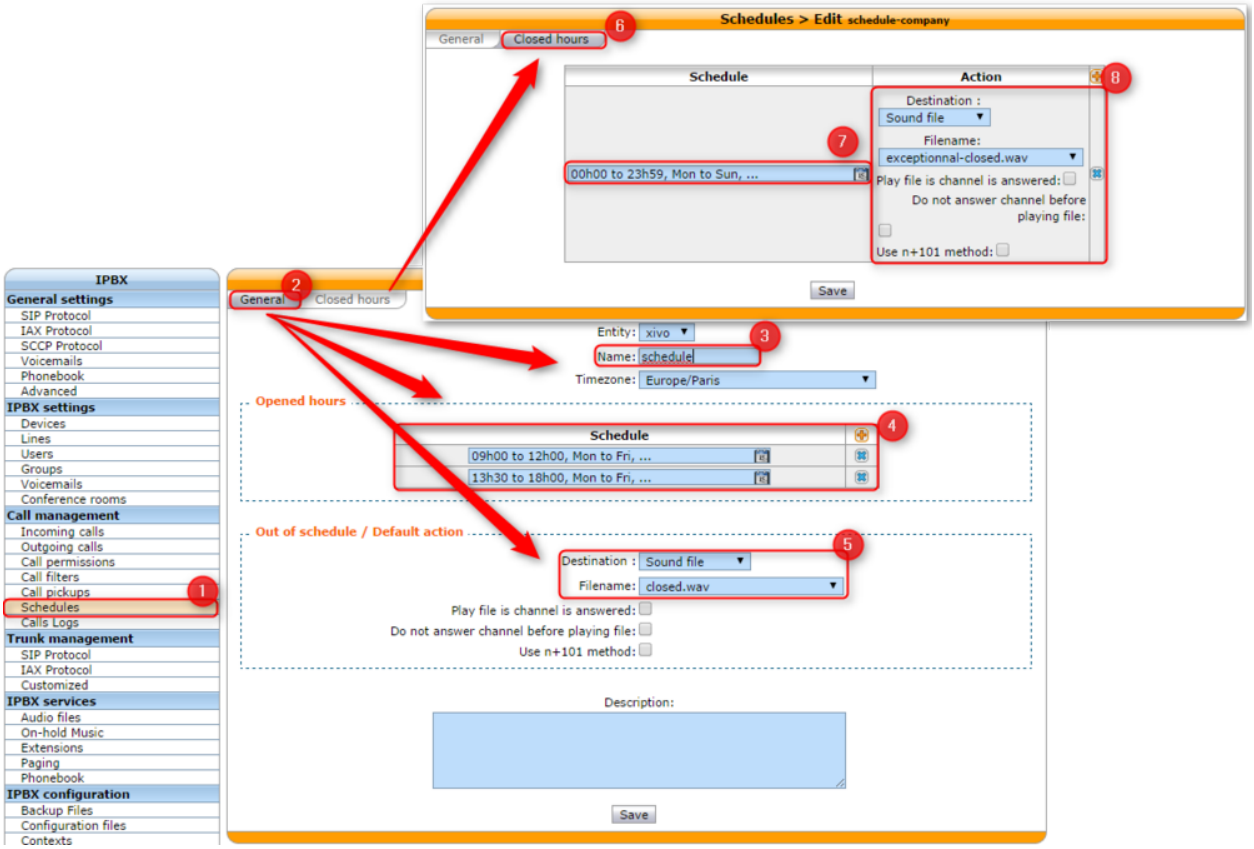
Use Case: IVR with submenu

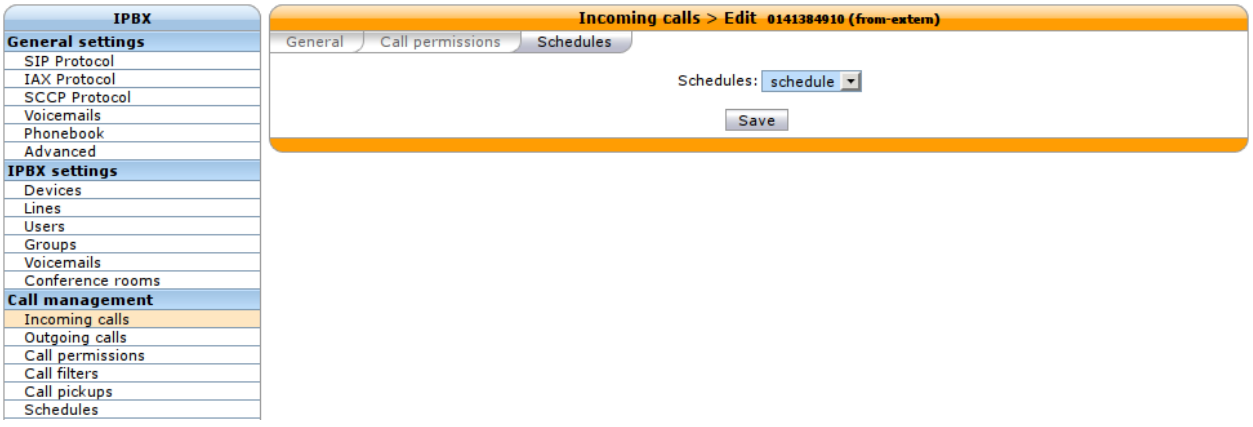
Flowchart

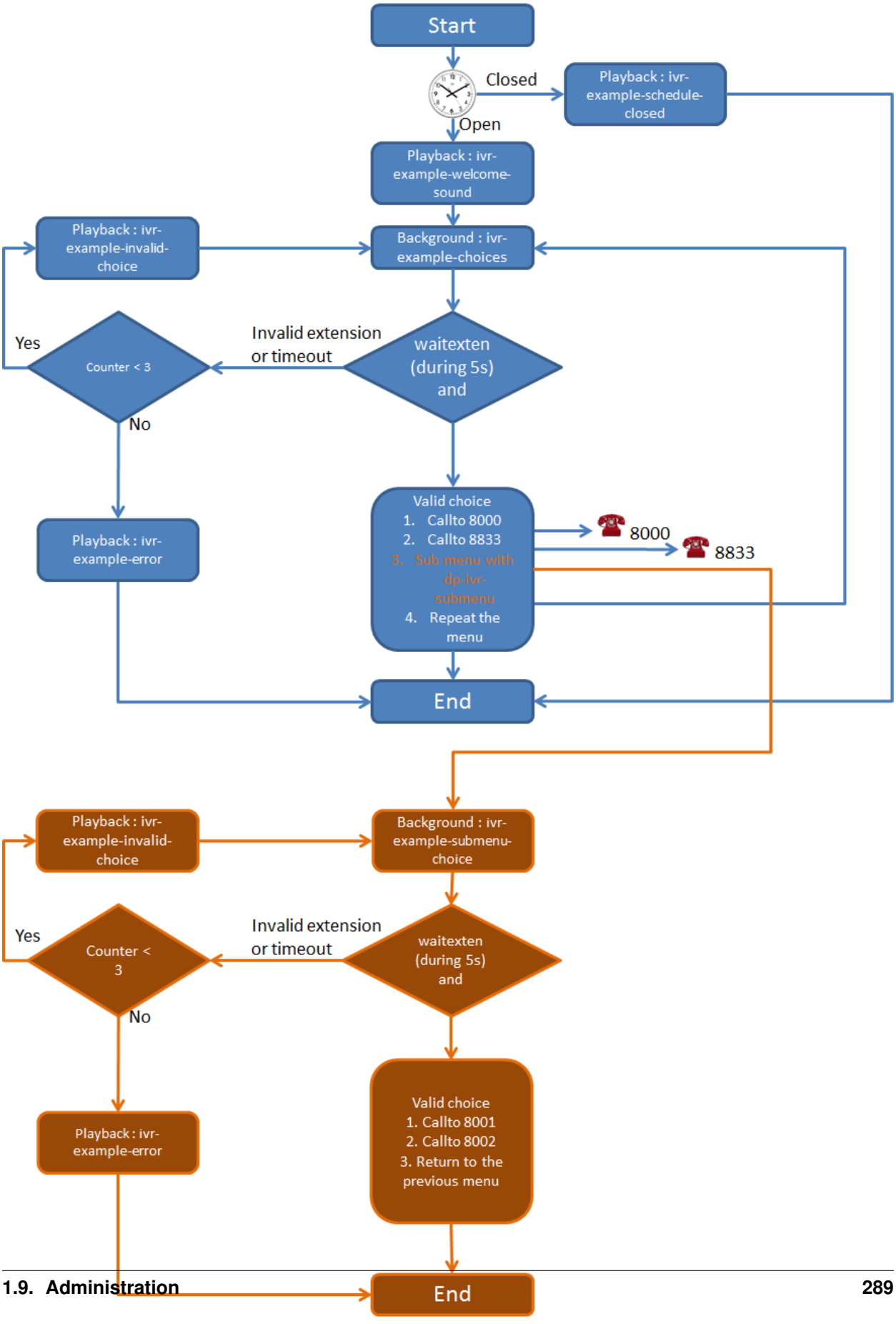
Configuration File and Dialplan

Copy all these lines (2 contexts) in a configuration file on your Wazo server :


Start
Closed
Playback : ivr-exemple-schedule-closed
Open
Playback : ivr-example-welcome-sound
Playback : ivr-example-invalid-choice
Background : ivr-exemple-choices
Yes
Counter < 3
Invalid extension or timeout
waitexten (during 5s) and
No
Playback : ivr-example-error
Valid choice
1. Callto 8000
2. Callto 8833
3. Callto 8547
4. Repeat the menu
8000
8833
8547
End


Schedules > Edit schedule-company
General
Closed hours
Schedule
Action
Destination :
Sound file
Filename:
exceptionnal-closed.wav
00h00 to 23h59, Mon to Sun, ...
Play file is channel is answered:
Do not answer channel before playing file:
Use n+101 method:
Save
IPBX
General settings
SIP Protocol
IAX Protocol
SCCP Protocol
Voicemails
Phonebook
Advanced
IPBX settings
Devices
Lines
Users
Groups
Voicemails
Conference rooms
Call management
Incoming calls
Outgoing calls
Call permissions
Call filters
Call pickups
Schedules
Calls Logs
Trunk management
SIP Protocol
IAX Protocol
Customized
IPBX services
Audio files
On-hold Music
Extensions
Paging
Phonebook
IPBX configuration
Backup Files
Configuration files
Contexts
Entity: xivo
Name: schedule
Timezone: Europe/Paris
Opened hours
Schedule
09h00 to 12h00, Mon to Fri, ...
13h30 to 18h00, Mon to Fri, ...
Out of schedule / Default action
Destination : Sound file
Filename: closed.wav
Play file is channel is answered:
Do not answer channel before playing file:
Use n+101 method:
Description:
Save
1
2
3
4
5
6
7
8


IPBX
General settings
SIP Protocol
IAX Protocol
SCCP Protocol
Voicemails
Phonebook
Advanced
IPBX settings
Devices
Lines
Users
Groups
Voicemails
Conference rooms
Call management
Incoming calls
Outgoing calls
Call permissions
Call filters
Call pickups
Schedules
Incoming calls > Edit 0141384910 (from-extern)
General
Call permissions
Schedules
Schedules: schedule
Save


Start
Closed
Playback : ivr-example-schedule-closed
Open
Playback : ivr-example-welcome-sound
Playback : ivr-example-invalid-choice
Background : ivr-example-choices
Yes
Counter < 3
Invalid extension or timeout
waitexten (during 5s) and
No
Playback : ivr-example-error
Valid choice
1. Callto 8000
2. Callto 8833
3. Sub menu with dp-ivr-submenu
4. Repeat the menu
8000
8833
End
Playback : ivr-example-invalid-choice
Background : ivr-example-submenu-choice
Yes
Counter < 3
Invalid extension or timeout
waitexten (during 5s) and
No
Playback : ivr-example-error
Valid choice
1. Callto 8001
2. Callto 8002
3. Return to the previous menu
End

```
[dp-ivr-example]

exten = s,1,NoOp(### dp-ivr-example.conf ###)
same = n,NoOp(Set the context containing your ivr destinations.)
same = n,Set(IVR_DESTINATION_CONTEXT=my-ivr-destination-context)
same = n,NoOp(Set the directory containing your ivr sounds.)
same = n,Set(GV_DIRECTORY_SOUNDS=/var/lib/xivo/sounds/ivr-sounds)
same = n,NoOp(the system answers the call and waits for 1 second before continuing)
same = n,Answer(1000)

same = n,NoOp(the system plays the first part of the audio file "welcome to ...")
same = n(first),Playback(${GV_DIRECTORY_SOUNDS}/ivr-example-welcome-sound)

same = n,NoOp(variable "counter" is set to 0)
same = n(beginning),Set(counter=0)

same = n,NoOp(variable "counter" is incremented and the label "start" is defined)
same = n(start),Set(counter=$[${counter} + 1])

same = n,NoOp(counter variable is now = ${counter})
same = n,NoOp(waiting for 1 second before reading the message that indicate all␣
↪choices)
same = n,Wait(1)
same = n,NoOp(play the message ivr-example-choices that contain all choices)
same = n,Background(${GV_DIRECTORY_SOUNDS}/ivr-example-choices)
same = n,NoOp(waiting for DTMF during 5s)
same = n,Waitexten(5)

;##### CHOICE 1 #####
exten = 1,1,NoOp(pressed digit is 1, redirect to 8000 in ${IVR_DESTINATION_CONTEXT}␣
↪context)
exten = 1,n,Goto(${IVR_DESTINATION_CONTEXT},8000,1)

;##### CHOICE 2 #####
exten = 2,1,NoOp(pressed digit is 2, redirect to 8833 in ${IVR_DESTINATION_CONTEXT}␣
↪context)
exten = 2,n,Goto(${IVR_DESTINATION_CONTEXT},8833,1)

;##### CHOICE 3 #####
exten = 3,1,NoOp(pressed digit is 3, redirect to the submenu dp-ivr-submenu)
exten = 3,n,Goto(dp-ivr-submenu,s,1)


;##### CHOICE 4 #####
exten = 4,1,NoOp(pressed digit is 4, redirect to start label in this context)
exten = 4,n,Goto(s,start)

;##### TIMEOUT #####
exten = t,1,NoOp(no digit pressed for 5s, process it like an error)
exten = t,n,Goto(i,1)

;##### INVALID CHOICE #####
exten = i,1,NoOp(if counter variable is 3 or more, then goto label "error")
exten = i,n,GotoIf($[${counter}>=3]?error)
exten = i,n,NoOp(pressed digit is invalid and less than 3 errors: the guide ivr-
↪exemple-invalid-choice is now played)
exten = i,n,Playback(${GV_DIRECTORY_SOUNDS}/ivr-example-invalid-choice)
exten = i,n,Goto(s,start)
```

```
exten = i,n(error),Playback(${GV_DIRECTORY_SOUNDS}/ivr-example-error)
exten = i,n,Hangup()


[dp-ivr-submenu]

exten = s,1,NoOp(### dp-ivr-submenu ###)
same = n,NoOp(the system answers the call and waits for 1 second before continuing)
same = n,Answer(1000)

same = n,NoOp(variable "counter" is set to 0)
same = n(beginning),Set(counter=0)

same = n,NoOp(variable "counter" is incremented and the label "start" is defined)
same = n(start),Set(counter=$[${counter} + 1])

same = n,NoOp(counter variable is now = ${counter})
same = n,NoOp(waiting for 1 second before reading the message that indicate all␣
↪choices)
same = n,Wait(1)
same = n,NoOp(play the message ivr-example-choices that contain all choices)
same = n,Background(${GV_DIRECTORY_SOUNDS}/ivr-example-submenu-choices)
same = n,NoOp(waiting for DTMF during 5s)
same = n,Waitexten(5)

;##### CHOICE 1 #####
exten = 1,1,NoOp(pressed digit is 1, redirect to 8000 in ${IVR_DESTINATION_CONTEXT}␣
↪context)
exten = 1,n,Goto(${IVR_DESTINATION_CONTEXT},8000,1)

;##### CHOICE 2 #####
exten = 2,1,NoOp(pressed digit is 2, redirect to 8001 in ${IVR_DESTINATION_CONTEXT}␣
↪context)
exten = 2,n,Goto(${IVR_DESTINATION_CONTEXT},8001,1)

;##### CHOICE 3 #####
exten = 3,1,NoOp(pressed digit is 3, redirect to the previous menu dp-ivr-example)
exten = 3,n,Goto(dp-ivr-example,s,beginning)

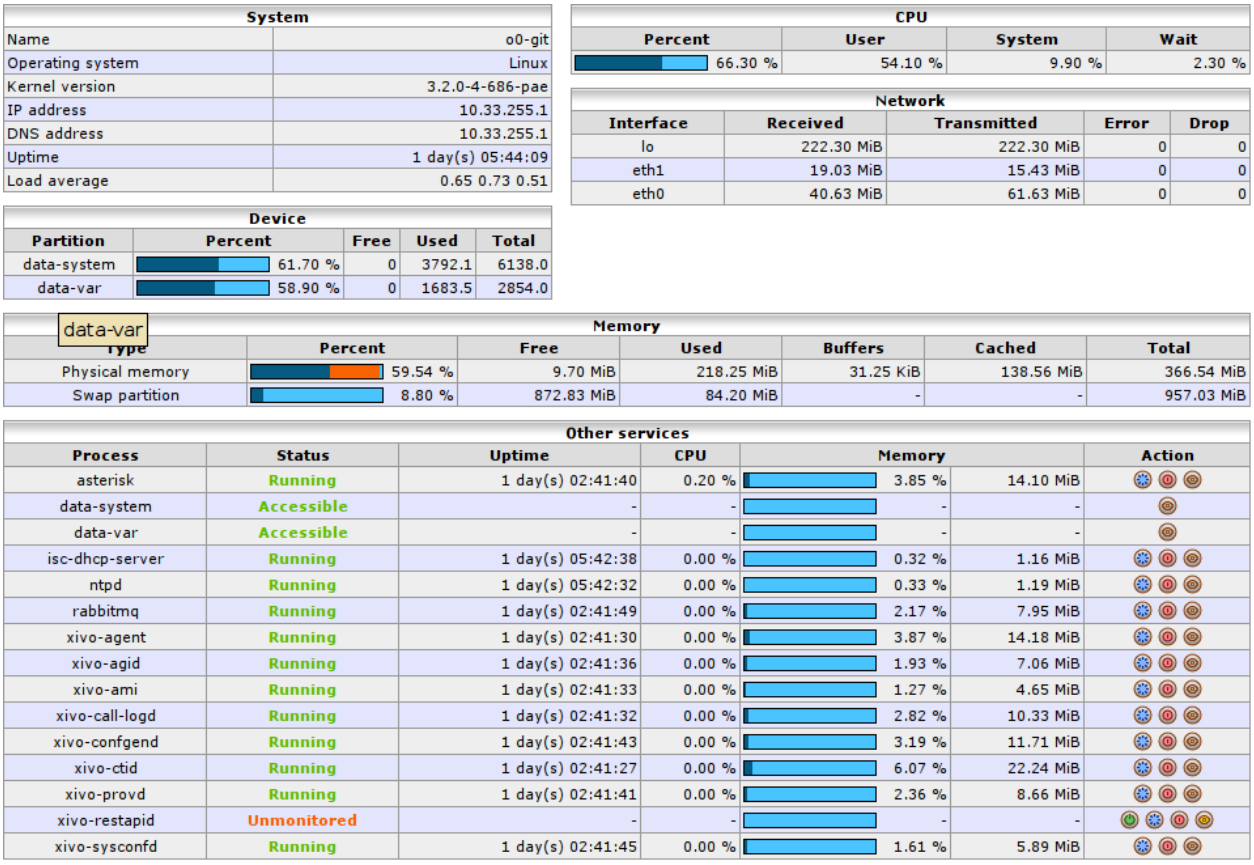

;##### TIMEOUT #####
exten = t,1,NoOp(no digit pressed for 5s, process it like an error)
exten = t,n,Goto(i,1)


;##### INVALID CHOICE #####
exten = i,1,NoOp(if counter variable is 3 or more, then goto label "error")
exten = i,n,GotoIf($[${counter}>=3]?error)
exten = i,n,NoOp(pressed digit is invalid and less than 3 errors: the guide ivr-
↪exemple-invalid-choice is now played)
exten = i,n,Playback(${GV_DIRECTORY_SOUNDS}/ivr-example-invalid-choice)
exten = i,n,Goto(s,start)
exten = i,n(error),Playback(${GV_DIRECTORY_SOUNDS}/ivr-example-error)
exten = i,n,Hangup()
```

1.9.24 Monitoring

The Monitoring section gives an overview of a Wazo system's status and of all monitored processes. It is divided into 6 sections :

- *System*
- *Device*
- *CPU*
- *Network*
- *Memory*
- *Other Services*



System

Name	o0-git
Operating system	Linux
Kernel version	3.2.0-4-686-pae
IP address	10.33.255.1
DNS address	10.33.255.1
Uptime	1 day(s) 05:44:09
Load average	0.65 0.73 0.51

CPU

Percent	User	System	Wait
66.30 %	54.10 %	9.90 %	2.30 %

Network

Interface	Received	Transmitted	Error	Drop
lo	222.30 MiB	222.30 MiB	0	0
eth1	19.03 MiB	15.43 MiB	0	0
eth0	40.63 MiB	61.63 MiB	0	0

Device

Partition	Percent	Free	Used	Total
data-system	61.70 %	0	3792.1	6138.0
data-var	58.90 %	0	1683.5	2854.0

data-var

Memory

Type	Percent	Free	Used	Buffers	Cached	Total
Physical memory	59.54 %	9.70 MiB	218.25 MiB	31.25 KiB	138.56 MiB	366.54 MiB
Swap partition	8.80 %	872.83 MiB	84.20 MiB	-	-	957.03 MiB

Other services

Process	Status	Uptime	CPU	Memory		Action
asterisk	Running	1 day(s) 02:41:40	0.20 %	3.85 %	14.10 MiB	
data-system	Accessible	-	-	-	-	
data-var	Accessible	-	-	-	-	
isc-dhcp-server	Running	1 day(s) 05:42:38	0.00 %	0.32 %	1.16 MiB	
ntpd	Running	1 day(s) 05:42:32	0.00 %	0.33 %	1.19 MiB	
rabbitmq	Running	1 day(s) 02:41:49	0.00 %	2.17 %	7.95 MiB	
xivo-agent	Running	1 day(s) 02:41:30	0.00 %	3.87 %	14.18 MiB	
xivo-agid	Running	1 day(s) 02:41:36	0.00 %	1.93 %	7.06 MiB	
xivo-ami	Running	1 day(s) 02:41:33	0.00 %	1.27 %	4.65 MiB	
xivo-call-logd	Running	1 day(s) 02:41:32	0.00 %	2.82 %	10.33 MiB	
xivo-confgend	Running	1 day(s) 02:41:43	0.00 %	3.19 %	11.71 MiB	
xivo-ctid	Running	1 day(s) 02:41:27	0.00 %	6.07 %	22.24 MiB	
xivo-provd	Running	1 day(s) 02:41:41	0.00 %	2.36 %	8.66 MiB	
xivo-restapid	Unmonitored	-	-	-	-	
xivo-sysconfd	Running	1 day(s) 02:41:45	0.00 %	1.61 %	5.89 MiB	

System

Displays generic information about the operating system, network addresses, uptime and load average. Read only.

Device

Displays free/used space on physical storage partitions. Read only.

CPU

Monitors the CPU usage. Read only.

Network

Displays network interfaces and corresponding network traffic. Read only.

Memory

Displays Physical and swap memory usage. Read only.

Other Services

Lists Wazo related processes (most of which are daemons) with their corresponding status, uptime, resource usage and controls to Restart service (blue button), stop service (red button) and stop monitoring service (grey button).

1.9.25 Music on Hold

The menu *Services* → *IPBX* → *IPBX services* → *On-hold Music* leads to the list of available on-hold musics.

Categories

Available categories are:

- files: play sound files. Formats supported:

Format Name	Filename Extension
G.719	.g719
G.723	.g723 .g723sf
G.726	.g726-40 .g726-32 .g726-24 .g726-16
G.729	.g729
GSM	.gsm
iLBC	.ilbc
Ogg Vorbis	.ogg (only mono files sampled at 8000 Hz)
G.711 A-law	.alaw .al .alw
G.711 μ-law	.pcm .ulaw .ul .mu .ulw
G.722	.g722
Au	.au
Siren7	.siren7
Siren14	.siren14
SLN	.raw .sln .sln12 .sln16 .sln24 .sln32 .sln44 .sln48 .sln96 .sln192
VOX	.vox
WAV	.wav .wav16
WAV GSM	.WAV .wav49

Only 1 audio channel must be present per file, i.e. files must be in mono.

If your music on hold files don't seem to work, you should look for errors in the asterisk logs.

The on-hold music will always play from the start.

- mp3: play MP3 files.

  Warning: The mp3 mode is deprecated and you should not use it. Instead, you should convert your MP3 files to another format and use the "files" mode.

  The on-hold music will play from an arbitrary position on the track, it will not play from the start.

- custom: do not play sound files. Instead, run an external process. That process must send on stdout the same binary format than WAV files.

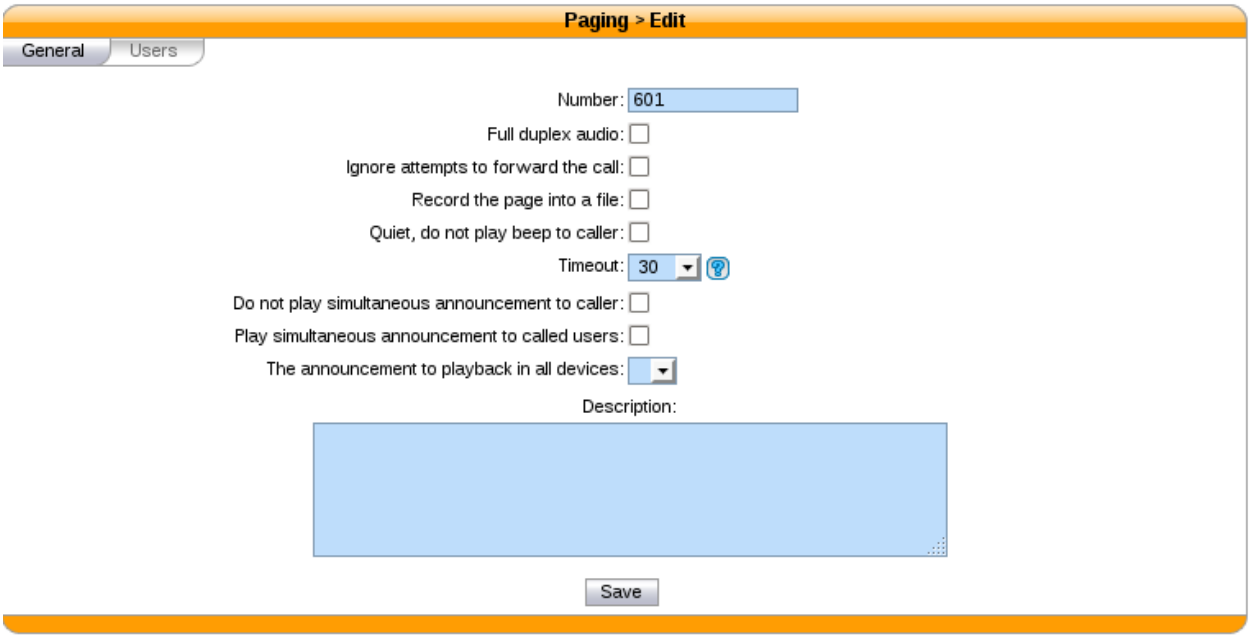
  Example process: `/usr/bin/mpg123 -s --mono -y -f 8192 -r 8000 http://streaming.example.com/stream.mp3`

  Note: Processes run by custom categories are started as soon as the category is created and will only stop when the category is deleted. This means that on-hold music fed from online streaming will constantly be receiving network traffic, even when there are no calls.

1.9.26 Paging

With Wazo, you can define paging (i.e. intercom) extensions to page a group of users. When calling a paging extension, the phones of the specified users will auto-answer, if they support it.

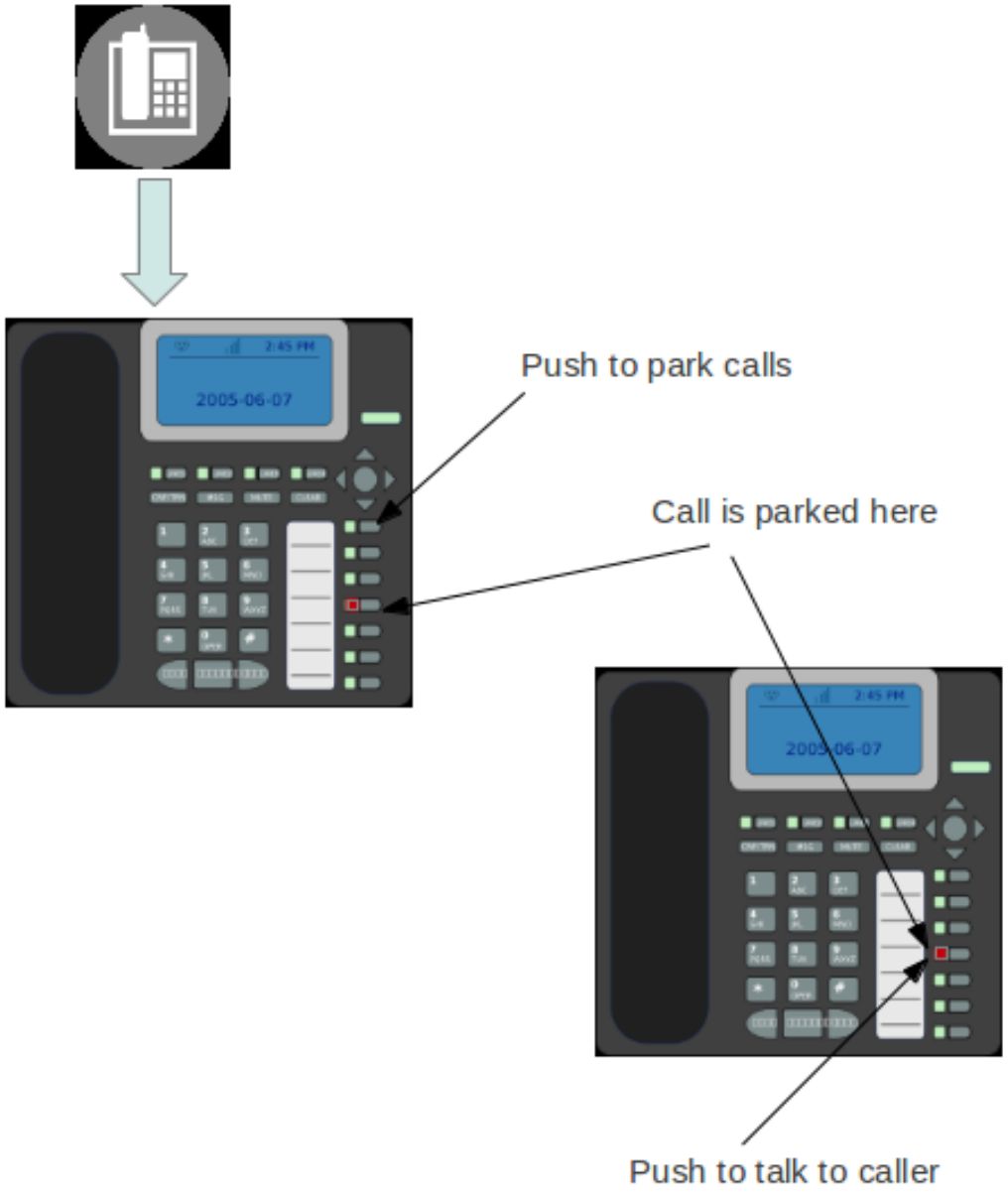
You can manage your paging extensions via the *Services* → *IPBX* → *Paging* page.



When adding a new paging extension, the number can be any numeric value; to call it, you just need to prefix the paging number with `*11`.

1.9.27 Parking

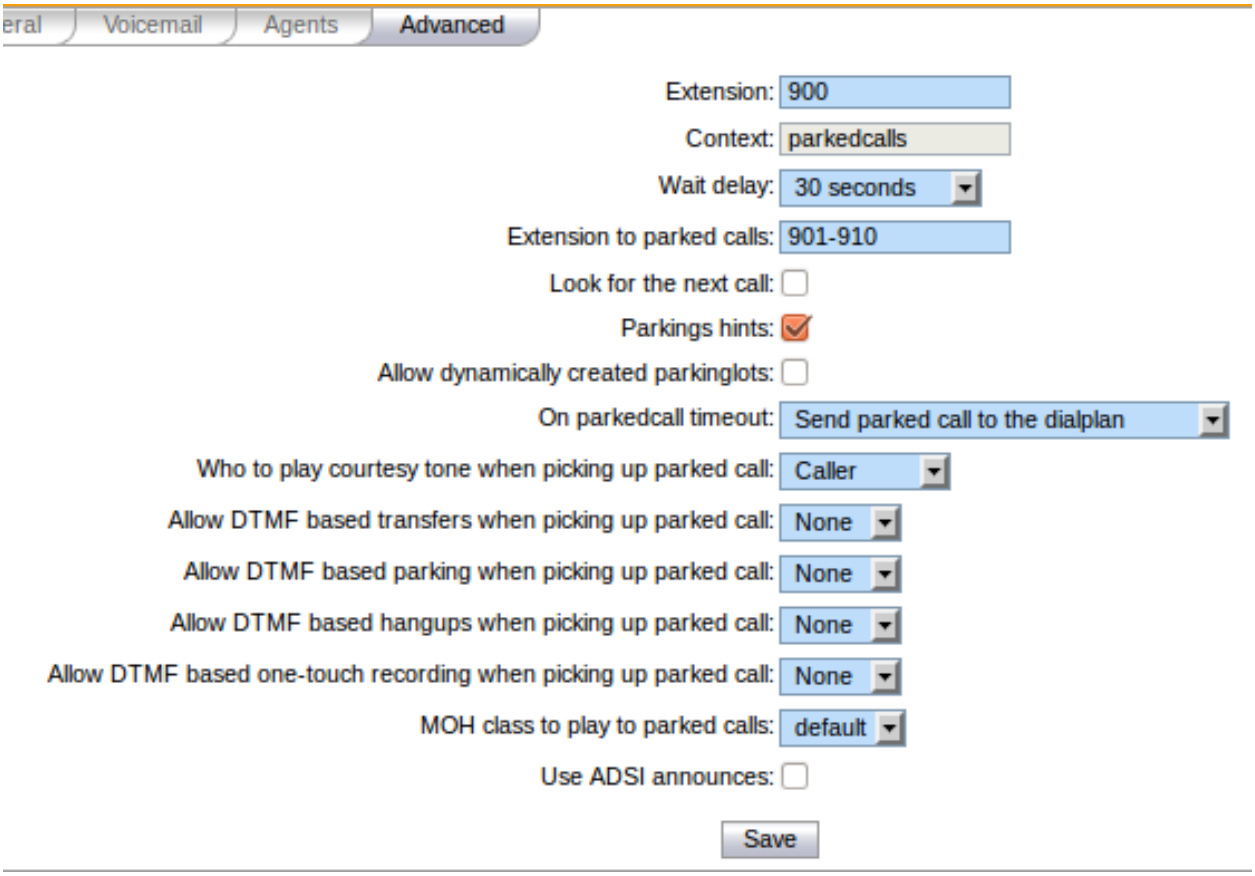
With Wazo it is possible to park calls, the same way you may park your car in a car parking. If you define supervised keys on a phone set for all the users of a system, when a call is parked, all the users are able to see that some one is waiting for an answer, push the phone key and get the call back to the phone.



There is a default parking number, 700, which is already configured when you install Wazo, but you may change the default configuration by editing the parking extension in menu *Service* → *IPBX* → *IPBX Services* → *Extensions* → *Advanced* → *Parking*

Using this extension, you may define the parking number used to park call, the parking lots, wether the sytem is rotating over the parking lots to park the calls, enable parking hint if you want to be able to supervise the parking using phone keys and other system default parameters.

You have two options in case of parking timeout :


eral
Voicemail
Agents
Advanced
Extension: 900
Context: parkedcalls
Wait delay: 30 seconds
Extension to parked calls: 901-910
Look for the next call:
Parkings hints:
Allow dynamically created parkinglots:
On parkedcall timeout: Send parked call to the dialplan
Who to play courtesy tone when picking up parked call: Caller
Allow DTMF based transfers when picking up parked call: None
Allow DTMF based parking when picking up parked call: None
Allow DTMF based hangups when picking up parked call: None
Allow DTMF based one-touch recording when picking up parked call: None
MOH class to play to parked calls: default
Use ADSI announces:
Save

- Callback the peer that parked this call

  In this case the call is sent back to the user who parked the call.

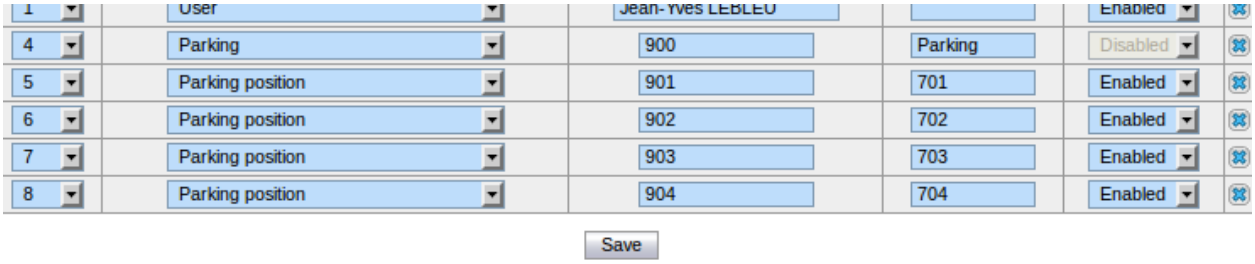
- Send park call to the dialplan

  In case you don't want to call back the user who parked the call, you have the option to send the call to any other extension or application. If the parking times out, the call is sent back to the dialplan in context `[parkedcallstimeout]`. You can define this context in a dialplan configuration file *Service → IPBX → Configuration Files* where you may define this context with dialplan commands.

  Example:

```
[parkedcallstimeout]
exten = s,1,Noop('park call time out')
same  =   n,Playback(hello-world)
same  =   n,Hangup()
```

It is also usual to define supervised phone keys to be able to park and unpark calls as in the example below.



1.9.28 Phonebook

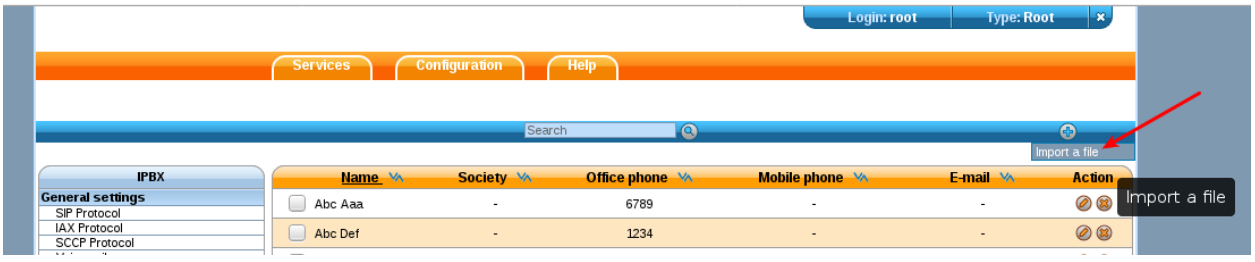
Phone books can be defined in *Services → IPBX → IPBX Services → Phonebook*. The phone books can be used from the Wazo Client, from the phones directory look key if the phone is compatible and are used to set the Caller ID for incoming calls.

You can add entries one by one or you can mass-import from a CSV file.

Note: To configure phonebook, see *Directories*.

Mass-import contacts

Go in the *Services → IPBX → IPBX Services → Phonebook* section:



The file to be imported must be a CSV file, with a comma character (`,`) as field delimiter. The file must be encoded in UTF-8.

Available fields are :

- title
- displayname
- firstname
- lastname
- society
- email
- url
- description

Address fields:

- address_<location>_address1
- address_<location>_address2
- address_<location>_city
- address_<location>_state
- address_<location>_zipcode
- address_<location>_country[1]

Available locations:

- home
- office
- other

Number fields:

- number_fax[2]
- number_office[2]
- number_home[2]
- number_mobile[2]
- number_other[2]

Example:

```
displayname,firstname,number_mobile,number_office,email,address_home_addres1,address_
↪home_address2,address_home_country
Alice,Alice,5555551234,5555557894,alice@example.com,123 wazo drv,apt. 42,CA
Bob,Bob,5556661234,5556667894,bob@example.com,123 wazo drv,apt. 42,CA
```

1.9.29 Provisioning

Wazo supports the auto-provisioning of a large number of telephony *Devices*, including SIP phones, SIP ATAs, and even softphones.

[1] These fields must contain ISO country codes. The complete list is described here.
[2] These fields must contain only phone number characters: + and * are allowed, but not space, point, etc.

Introduction

The auto-provisioning feature found in Wazo make it possible to provision, i.e. configure, a lots of telephony devices in an efficient and effortless way.

How it works

Here's a simplified view of how auto-provisioning is supported on a typical SIP hardphone:

1. The phone is powered on
2. During its boot process, the phone sends a DHCP request to obtain its network configuration
3. A DHCP server replies with the phone network configuration + an HTTP URL
4. The phone use the provided URL to retrieve a common configuration file, a MAC-specific configuration file, a firmware image and some language files.

Building on this, configuring one of the supported phone on Wazo is as simple as:

1. *Configuring the DHCP Server*
2. *Installing the required provd plugin*
3. Powering on the phone
4. Dialing the user's provisioning code from the phone

And *voila*, once the phone has rebooted, your user is ready to make and receive calls. No manual editing of configuration files nor fiddling in the phone's web interface.

Limitations

- Device synchronisation does not work in the situation where multiple devices are connected from behind a NAPT network equipment. The devices must be resynchronised manually.

External links

- Introduction to provd plugin model
- HTTP/TFTP requests processing in provd - part 1
- HTTP/TFTP requests processing in provd - part 2

Basic Configuration

You have two options to get your phone to be provisioned:

- Set up a DHCP server
- Tell manually each phone where to get the provisioning informations

You may want to manually configure the phones if you are only trying Wazo or if your network configuration does not allow the phones to access the Wazo DHCP server.

You may want to set up a DHCP server if you have a significant number of phones to connect, as no manual intervention will be required on each phone.

Configuring the DHCP Server

Wazo includes a DHCP server that facilitate the auto-provisioning of telephony devices. It is *not* activated by default.

There's a few things to know about the peculiarities of the included DHCP server:

- it only answers to DHCP requests from *supported devices*.
- it only answers to DHCP requests coming from the VoIP subnet (see *network configuration*).

This means that if your phones are on the same broadcast domain than your computers, and you would like the DHCP server on your Wazo to handle both your phones and your computers, that won't do it.

The DHCP server is configured via the *Configuration* → *Network* → *DHCP* page:

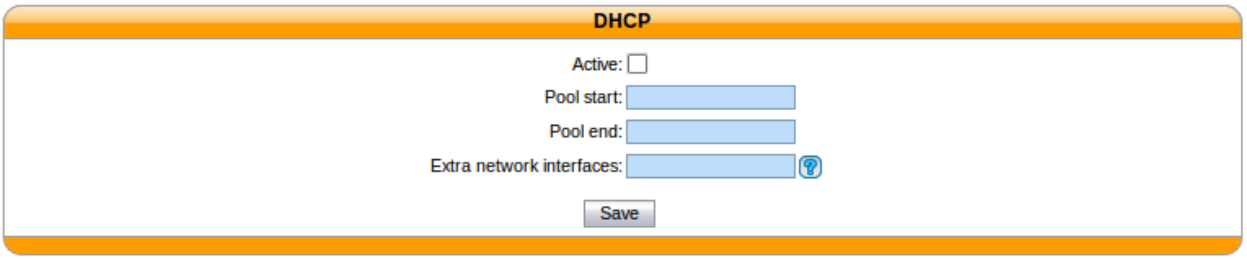


Fig. 1.67: *Configuration* → *Network* → *DHCP*

Active Activate/desactivate the DHCP server.

Pool start The lower IP address which will be assigned dynamically. This address should be in the VoIP subnet. Example: `10.0.0.10`.

Pool end The higher IP address which will be assigned dynamically. This address should be in the VoIP subnet. Example: `10.0.0.99`.

Extra network interfaces A list of space-separated network interface name. Example: `eth0`.

Useful if you have done some custom configuration in the `/etc/dhcp/dhcpd_extra.conf` file. You need to explicitly specify the additional interfaces the DHCP server should listen on.

After saving your modifications, you need to click on *Apply system configuration* for them to be applied.

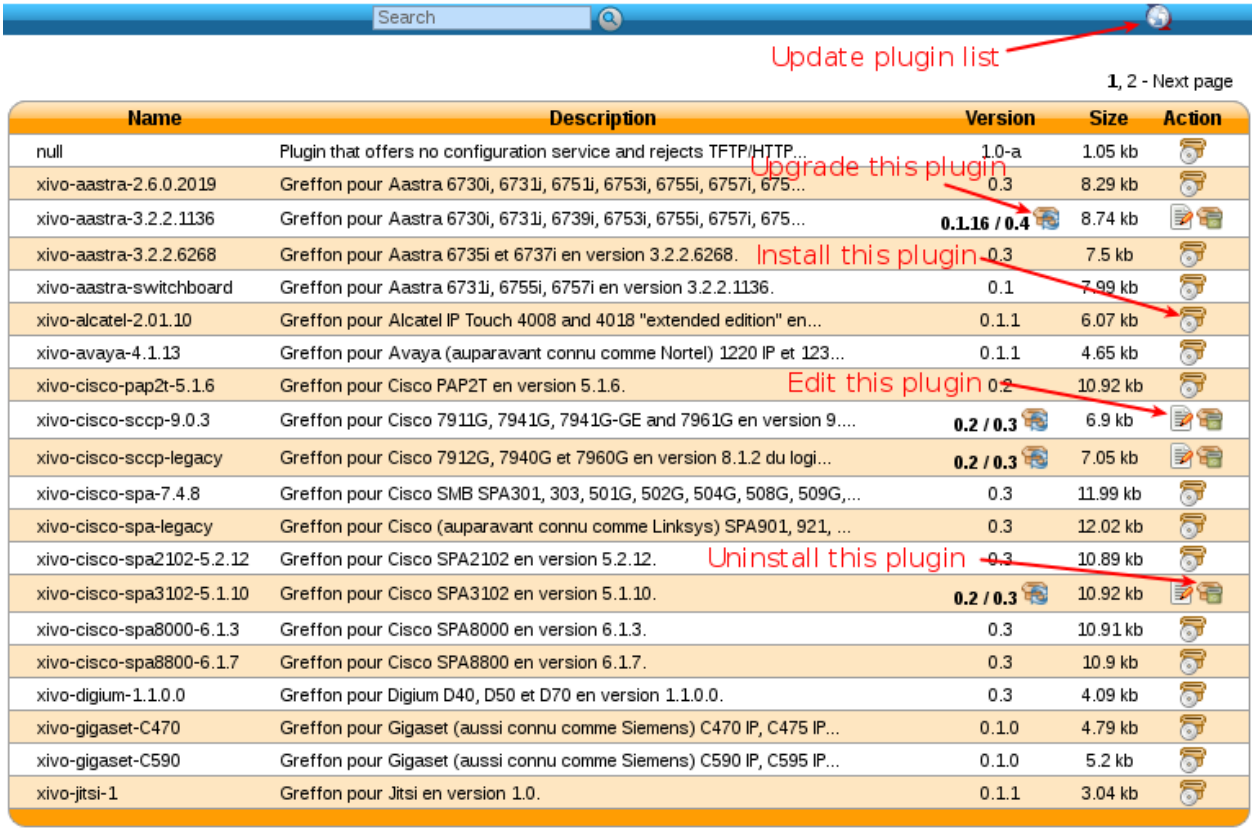
Installing `provd` Plugins

The installation and management of `provd` plugins is done via the *Configuration* → *Provisioning* → *Plugin* page:

The page shows the list of both the installed and installable plugins. You can see if a plugin is installed or not by looking at the *Action* column.

Here's the list of other things that can be done from this page:

- update the list of installable plugins, by clicking on the top right icon. On a fresh Wazo installation, this is the first thing to do.
- install a new plugin
- upgrade an installed plugin
- uninstall an installed plugin
- edit an installed plugin, i.e. install/uninstall optional files that are specific to each plugin, like firmware or language files



Name	Description	Version	Size	Action
null	Plugin that offers no configuration service and rejects TFTP/HTTP...	1.0-a	1.05 kb	
xivo-aastra-2.6.0.2019	Greffon pour Aastra 6730i, 6731i, 6751i, 6753i, 6755i, 6757i, 675...	0.3	8.29 kb	
xivo-aastra-3.2.2.1136	Greffon pour Aastra 6730i, 6731i, 6739i, 6753i, 6755i, 6757i, 675...	**0.1.16 / 0.4**	8.74 kb	
xivo-aastra-3.2.2.6268	Greffon pour Aastra 6735i et 6737i en version 3.2.2.6268.	0.3	7.5 kb	
xivo-aastra-switchboard	Greffon pour Aastra 6731i, 6755i, 6757i en version 3.2.2.1136.	0.1	7.99 kb	
xivo-alcatel-2.01.10	Greffon pour Alcatel IP Touch 4008 and 4018 "extended edition" en...	0.1.1	6.07 kb	
xivo-avaya-4.1.13	Greffon pour Avaya (auparavant connu comme Nortel) 1220 IP et 123...	0.1.1	4.65 kb	
xivo-cisco-pap2t-5.1.6	Greffon pour Cisco PAP2T en version 5.1.6.	0.2	10.92 kb	
xivo-cisco-sccp-9.0.3	Greffon pour Cisco 7911G, 7941G, 7941G-GE and 7961G en version 9....	**0.2 / 0.3**	6.9 kb	
xivo-cisco-sccp-legacy	Greffon pour Cisco 7912G, 7940G et 7960G en version 8.1.2 du logi...	**0.2 / 0.3**	7.05 kb	
xivo-cisco-spa-7.4.8	Greffon pour Cisco SMB SPA301, 303, 501G, 502G, 504G, 508G, 509G,...	0.3	11.99 kb	
xivo-cisco-spa-legacy	Greffon pour Cisco (auparavant connu comme Linksys) SPA901, 921, ...	0.3	12.02 kb	
xivo-cisco-spa2102-5.2.12	Greffon pour Cisco SPA2102 en version 5.2.12.	0.3	10.89 kb	
xivo-cisco-spa3102-5.1.10	Greffon pour Cisco SPA3102 en version 5.1.10.	**0.2 / 0.3**	10.92 kb	
xivo-cisco-spa8000-6.1.3	Greffon pour Cisco SPA8000 en version 6.1.3.	0.3	10.91 kb	
xivo-cisco-spa8800-6.1.7	Greffon pour Cisco SPA8800 en version 6.1.7.	0.3	10.9 kb	
xivo-digium-1.1.0.0	Greffon pour Digium D40, D50 et D70 en version 1.1.0.0.	0.3	4.09 kb	
xivo-gigaset-C470	Greffon pour Gigaset (aussi connu comme Siemens) C470 IP, C475 IP...	0.1.0	4.79 kb	
xivo-gigaset-C590	Greffon pour Gigaset (aussi connu comme Siemens) C590 IP, C595 IP...	0.1.0	5.2 kb	
xivo-jitsi-1	Greffon pour Jitsi en version 1.0.	0.1.1	3.04 kb	

Fig. 1.68: *Configuration → Provisioning → Plugin*

After installing a new plugin, you are automatically redirected to its edit page. You can then download and install optional files specific to the plugin. You are strongly advised to install firmware and language files for the phones you'll use although it's often not a strict requirement for the phones to work correctly.

Warning: If you uninstall a plugin that is used by some of your devices, they will be left in an unconfigured state and won't be associated to another plugin automatically.

The search box at the top comes in handy when you want to find which plugin to install for your device. For example, if you have a Cisco SPA508G, enter "508" in the search box and you should see there's 1 plugin compatible with it.

Note: If your device has a number in its model name, you should use only the number as the search keyword since this is what usually gives the best results.

It's possible there will be more than 1 plugin compatible with a given device. In these cases, the difference between the two plugins is usually just the firmware version the plugins target. If you are unsure about which version you should install, you should look for more information on the vendor website.

It's good practice to only install the plugins you need and no more.

Alternative plugins repository

By default, the list of plugins available for installation are the stable plugins for the officially supported devices.

This can be changed in the *Configuration* → *Provisioning* → *General* page, by setting the *URL* field to one of the following value:

- `http://provd.wazo.community/plugins/1/stable/` – *community supported devices* "stable" repository
- `http://provd.wazo.community/plugins/1/testing/` – officially supported devices "testing" repository
- `http://provd.wazo.community/plugins/1/archive/` – officially supported devices "archive" repository

The difference between the stable and testing repositories is that the latter might contain plugins that are not working properly or are still in developement.

The archive repository contains plugins that were once in the stable repository.

After setting a new URL, you must refresh the list of installable plugins by clicking the update icon of the *Configuration* → *Provisioning* → *Plugin* page.

How to manually tell the phones to get their configuration

If you have set up a DHCP server on Wazo and the phones can access it, you can skip this section.

The according provisioning plugins must be installed.

Aastra

On the web interface of your phone, go to *Advanced settings* → *Configuration server*, and enter the following settings:



Polycom

On the phone, go to *Menu* → *Settings* → *Advanced* → *Admin Settings* → *Network configuration* → *Server Menu* and enter the following settings:

- Server type: HTTP
- Server address: `http://<Wazo IP address>:8667/000000000000.cfg`

Then save and reboot the phone.

Snom

First, you need to run the following command on the Wazo server:

```
sed -i 's/dhcp:stop/dhcp:proceed/' /var/lib/xivo-provd/plugins/xivo-snom-8.7.5.35/var/
↪tftpboot/snom-general.xml
```

On the web interface of your phone, go to *Setup* → *Advanced* → *Update* and enter the following settings:

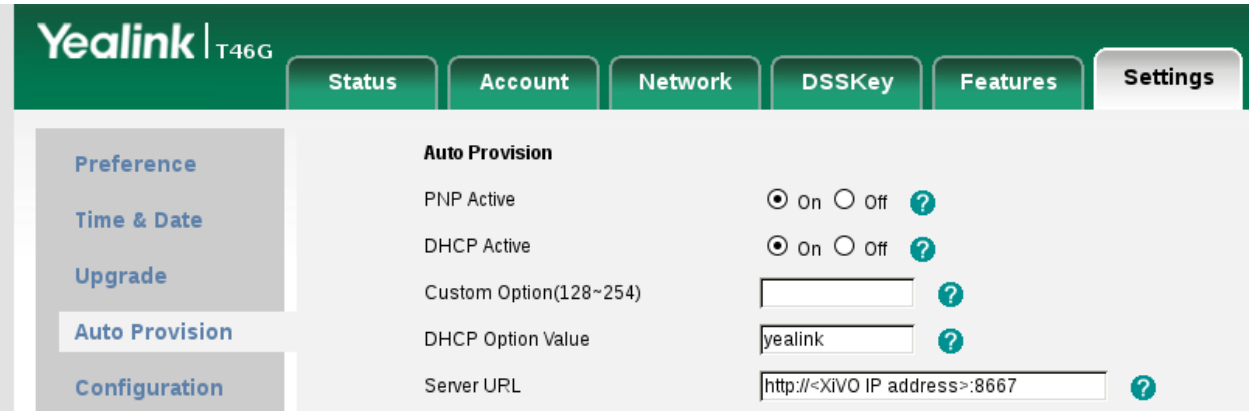
Yealink

On the web interface of your phone, go to *Settings* → *Auto Provision*, and enter the following settings:

- Server URL: `http://<Wazo IP address>:8667`

Save the changes by clicking on the *Confirm* button and then click on the *Autoprovision Now* button.





Autoprovisioning a Device

Once you have installed the proper provd plugins for your devices and setup correctly your DHCP server, you can then connect your devices to your network.

But first, go to *Services* → *IPBX* → *Devices* page. You will then see that no devices are currently known by your Wazo:

You can then power on your devices on your LAN. For example, after you power on an Aastra 6731i and give it the time to boot and maybe upgrade its firmware, you should then see the phone having its first line configured as 'autoprov', and if you refresh the devices page, you should see that your Wazo now knows about your 6731i:

You can then dial from your Aastra 6731i the provisioning code associated to a line of one of your user. You will hear a prompt thanking you and your device should then reboot in the next few seconds. Once the device has rebooted, it will then be properly configured for your user to use it. And also, if you update the device page, you'll see that the icon next to your device has now passed to green:

Resetting a Device

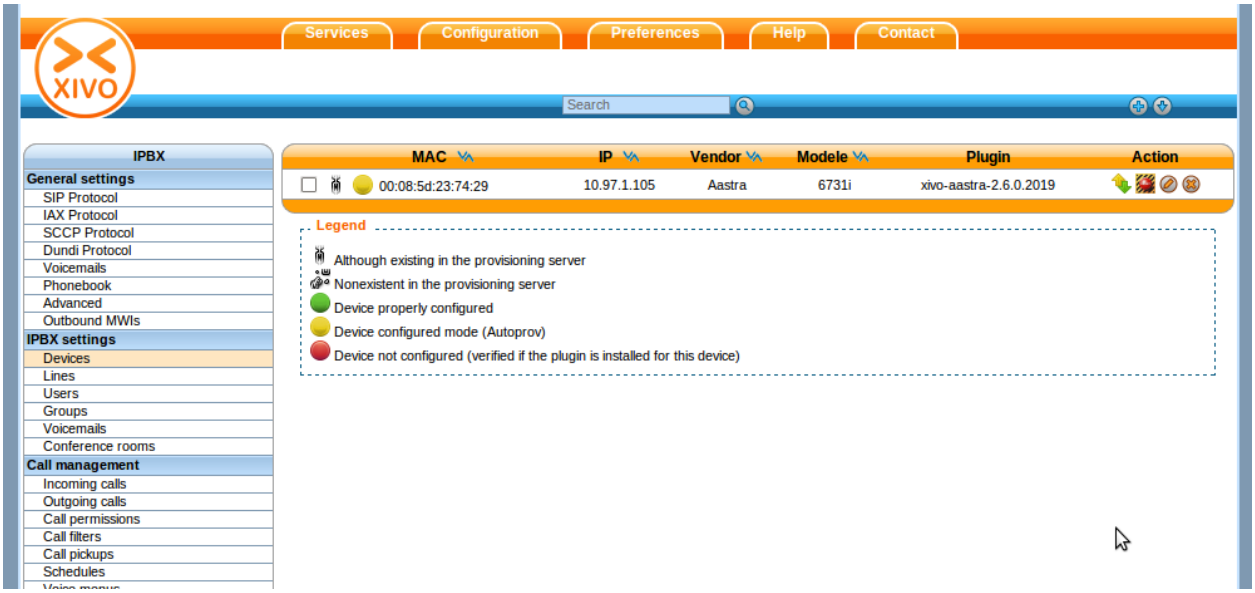
From the Device List in the Webi

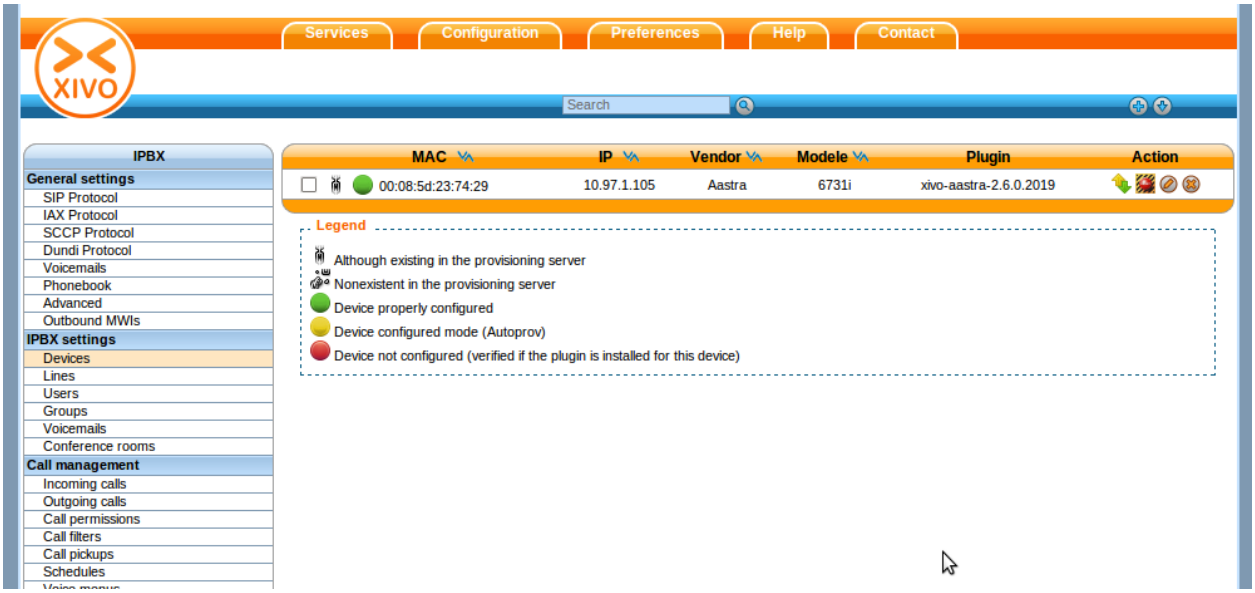
To remove a phone from Wazo or enable a device to be used for another user there are two different possibilities :

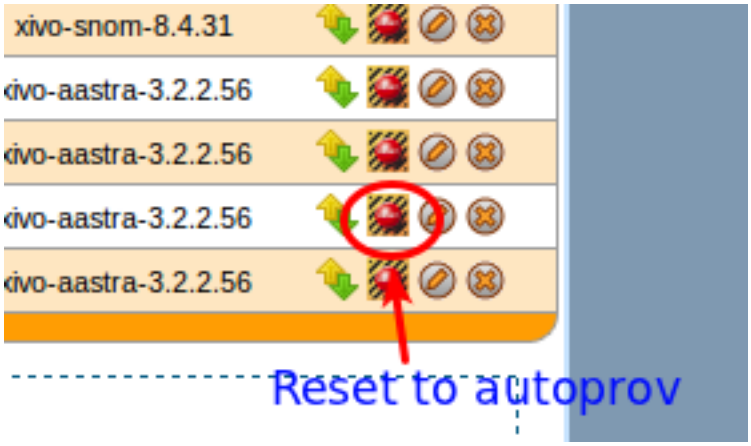
- click on the `reset to autoprov` button on the web interface

The phone will restarts and display autoprov, ready to be used for another user.


Services
Configuration
Preferences
Help
Contact
Search
IPBX
General settings
SIP Protocol
IAX Protocol
SCCP Protocol
Dundi Protocol
Voicemails
Phonebook
Advanced
Outbound MWIs
IPBX settings
Devices
Lines
Users
Groups
Voicemails
Conference rooms
Call management
Incoming calls
Outgoing calls
Call permissions
Call filters
Call pickups
Schedules
MAC
IP
Vendor
Modele
Plugin
Action
No device found
Legend
Although existing in the provisioning server
Nonexistent in the provisioning server
Device properly configured
Device configured mode (Autoprov)
Device not configured (verified if the plugin is installed for this device)


Services
Configuration
Preferences
Help
Contact
Search
IPBX
General settings
SIP Protocol
IAX Protocol
SCCP Protocol
Dundi Protocol
Voicemails
Phonebook
Advanced
Outbound MWIs
IPBX settings
Devices
Lines
Users
Groups
Voicemails
Conference rooms
Call management
Incoming calls
Outgoing calls
Call permissions
Call filters
Call pickups
Schedules
MAC
IP
Vendor
Modele
Plugin
Action
00:08:5d:23:74:29
10.97.1.105
Aastra
6731i
xivo-aastra-2.6.0.2019
Legend
Although existing in the provisioning server
Nonexistent in the provisioning server
Device properly configured
Device configured mode (Autoprov)
Device not configured (verified if the plugin is installed for this device)


Services
Configuration
Preferences
Help
Contact
Search
IPBX
General settings
SIP Protocol
IAX Protocol
SCCP Protocol
Dundi Protocol
Voicemails
Phonebook
Advanced
Outbound MWIs
IPBX settings
Devices
Lines
Users
Groups
Voicemails
Conference rooms
Call management
Incoming calls
Outgoing calls
Call permissions
Call filters
Call pickups
Schedules
MAC
IP
Vendor
Modele
Plugin
Action
00:08:5d:23:74:29
10.97.1.105
Aastra
6731i
xivo-aastra-2.6.0.2019
Legend
Although existing in the provisioning server
Nonexistent in the provisioning server
Device properly configured
Device configured mode (Autoprov)
Device not configured (verified if the plugin is installed for this device)


xivo-snom-8.4.31
xivo-aastra-3.2.2.56
xivo-aastra-3.2.2.56
xivo-aastra-3.2.2.56
xivo-aastra-3.2.2.56
Reset to autoprov

From the User Form in the Webi

Device With one User Only Associated

Edit the user associated to the device and put the device field to null.

- click on the `Save` button on the web interface

The phone doesn't restart and the phone is in autoprov mode in the device list.

You can synchronize the device to reboot it.

Device with Several Users Associated

Edit the primary user associated to the terminal (one with the line 1) and put the device field to null.

- click on the `Save` button on the web interface

The primary line of the phone has been removed, so the device will lose its funckeys associated to primary user but there others lines associated to the device will stay provisionned.

The phone doesn't restart and the phone is in autoprov mode in the device list.

You can synchronize the device for reboot it.

From a Device

- Dial ***guest** (*48378) on the phone dialpad followed by **xivo** (9486) as a password

The phone restarts and display autoprov, ready to be used for another user.

Advanced Configuration

DHCP Integration

If your phones are getting their network configuration from your Wazo's DHCP server, it's possible to activate the DHCP integration on the *Configuration* → *Provisioning* → *General* page.

What DHCP integration does is that, on every DHCP request made by one of your phones, the DHCP server sends information about the request to `provd`, which can then use this information to update its device database.

This feature is useful for phones which lack information in their TFTP/HTTP requests. For example, without DHCP integration, it's impossible to extract model information for phones from the Cisco 7900 series. Without the model information extracted, there's chance your device won't be automatically associated to the best plugin.

This feature can also be useful if your phones are not always getting the same IP addresses, for one reason or another. Again, this is useful only for some phones, like the Cisco 7900; it has no effect for Aastra 6700.

Creating Custom Templates

Custom templates comes in handy when you have some really specific configuration to make on your telephony devices.

Templates are handled on a per plugin basis. It's not possible for a template to be shared by more than one plugin since it's a design limitation of the plugin system of `provd`.

Note: When you install a new plugin, templates are not migrated automatically, so you must manually copy them from the old plugin directory to the new one. This does not apply for a plugin upgrade.

Let's suppose we have installed the `xivo-aastra-3.3.1-SP2` plugin and want to write some custom templates for it.

First thing to do is to go into the directory where the plugin is installed:

```
cd /var/lib/xivo-provd/plugins/xivo-aastra-3.3.1-SP2
```

Once you are there, you can see there's quite a few files and directories:

```
tree
.
+-- common.py
+-- entry.py
+-- pkgs
|   +-- pkgs.db
+-- plugin-info
+-- README
+-- templates
|   +-- 6730i.tpl
|   +-- 6731i.tpl
|   +-- 6739i.tpl
|   +-- 6753i.tpl
|   +-- 6755i.tpl
|   +-- 6757i.tpl
|   +-- 9143i.tpl
|   +-- 9480i.tpl
|   +-- base.tpl
+-- var
    +-- cache
    +-- installed
    +-- templates
    +-- tftpboot
        +-- Aastra
            +-- aastra.cfg
```

The interesting directories are:

templates This is where the original templates lies. You *should not* edit these files directly but instead copy the one you want to modify in the var/templates directory.

var/templates This is the directory where you put and edit your custom templates.

var/tftpboot This is where the configuration files lies once they have been generated from the templates. You should look at them to confirm that your custom templates are giving you the result you are expecting.

Warning: When you uninstall a plugin, the plugin directory is removed altogether, including all the custom templates.

A few things to know before writing your first custom template:

- templates use the Jinja2 template engine.
- when doing an `include` or an `extend` from a template, the file is first looked up in the `var/templates` directory and then in the `templates` directory.

- device in autoprov mode are affected by templates, because from the point of view of `provd`, there's no difference between a device in autoprov mode or fully configured. This means there's usually no need to modify static files in `var/tftpboot`. And this is a bad idea since a plugin upgrade will override these files.

Custom template for every devices

```
cp templates/base.tpl var/templates
vi var/templates/base.tpl
xivo-provd-cli -c 'devices.using_plugin("xivo-aastra-3.3.1-SP2").reconfigure()'
```

Once this is done, if you want to synchronize all the affected devices, use the following command:

```
xivo-provd-cli -c 'devices.using_plugin("xivo-aastra-3.3.1-SP2").synchronize()'
```

Custom template for a specific model

Let's supose we want to customize the template for our 6739i:

```
cp templates/6739i.tpl var/templates
vi var/templates/6739i.tpl
xivo-provd-cli -c 'devices.using_plugin("xivo-aastra-3.3.1-SP2").reconfigure()'
```

Custom template for a specific device

To create a custom template for a specific device you have to create a device-specific template named `<device_specific_file_with_extension>.tpl` in the `var/templates/` directory :

- for an Aastra phone, if you want to customize the file `00085D2EECFB.cfg` you will have to create a template file named `00085D2EECFB.cfg.tpl`,
- for a Snom phone, if you want to customize the file `000413470411.xml` you will have to create a template file named `000413470411.xml.tpl`,
- for a Polycom phone, if you want to customize the file `0004f2211c8b-user.cfg` you will have to create a template file named `0004f2211c8b-user.cfg.tpl`,
- and so on.

Here, we want to customize the content of a device-specific file named `00085D2EECFB.cfg`, we need to create a template named `00085D2EECFB.cfg.tpl`:

```
cp templates/6739i.tpl var/templates/00085D2EECFB.cfg.tpl
vi var/templates/00085D2EECFB.cfg.tpl
xivo-provd-cli -c 'devices.using_mac("00085D2EECFB").reconfigure()'
```

Note: The choice to use this syntax comes from the fact that `provd` supports devices that do not have MAC addresses, namely softphones.

Also, some devices have more than one file (like Snom), so this way make it possible to customize more than 1 file.

The template to use as the base for a device specific template will vary depending on the need. Typically, the model template will be a good choice, but it might not always be the case.

Changing the Plugin Used by a Device

From time to time, new firmwares are released by the devices manufacturer. This sometimes translate to a new plugin being available for these devices.

When this happens, it almost always means the new plugin obsoletes the older one. The older plugin is then considered "end-of-life", and won't receive any new updates nor be available for new installation.

Let's suppose we have the old `xivo-aastra-3.2.2.1136` plugin installed on our Wazo and want to use the newer `xivo-aastra-3.3.1-SP2` plugin.

Both these plugins can be installed at the same time, and you can manually change the plugin used by a phone by editing it via the *Services* → *IPBX* → *Devices* page.

If you are using custom templates in your old plugin, you should copy them to the new plugin and make sure that they are still compatible.

Once you take the decision to migrate all your phones to the new plugin, you can use the following command:

```
xivo-provd-cli -c 'helpers.mass_update_devices_plugin("xivo-aastra-3.2.2.1136", "xivo-
↪aastra-3.3.1-SP2")'
```

Or, if you also want to synchronize (i.e. reboot) them at the same time:

```
xivo-provd-cli -c 'helpers.mass_update_devices_plugin("xivo-aastra-3.2.2.1136", "xivo-
↪aastra-3.3.1-SP2", synchronize=True)'
```

You can check that all went well by looking at the *Services* → *IPBX* → *Devices* page.

NAT

The provisioning server has partial support for environment where the telephony devices are behind a NAT equipment.

By default, each time the provisioning server receives an HTTP/TFTP request from a device, it makes sure that only one device has the source IP address of the request. This is not a desirable behaviour when the provisioning server is used in a NAT environment, since in this case, it's normal that more than 1 devices have the same source IP address (from the point of view of the server).

If *all* your devices used on your Wazo are behind a NAT, you should disable this behaviour by setting the `NAT` option to 1 via the *Configuration* → *Provisioning* → *General* page.

Enabling the NAT option will also improve the performance of the provisioning server in this scenario.

If you have many devices behind a NAT equipment, you should also check the *security* section to make sure the IP address of your NAT equipment doesn't get banned unintentionally.

Limitations

- You must only have phones of the following brands:
  - Aastra
  - Cisco SPA
  - Yealink
- All your devices must be behind a NAT equipment (the devices may be grouped behind different NAT equipments, not necessarily the same one)

- You must provision the devices via the Web interface, i.e. associate the devices from the user form. Using the 6-digit provisioning code on the phone will produce unexpected results (i.e. the wrong device will be provisioned)

For technical information about why other devices are not supported, you can look at this issue on the Wazo bug tracker.

Security

By design, the auto-provisioning process is vulnerable to:

- Leakage of sensitive information: some files that are served by the provisioning server contains sensitive information, e.g. SIP credentials that are used by SIP phones to make calls. Depending on your network configuration and the amount of information an attacker has on your telephony ecosystem (phone vendor, MAC address, etc.), he could retrieve the content of some files containing sensitive information.
- Denial-of-service attack: in its default configuration, each time the provisioning server identify a request coming from a new device, it creates a new device object in its database. An attacker could spoof requests to the provisioning server to create a huge amount of devices, creating a denial-of-service condition.

That said, starting from XiVO 16.08, XiVO adds Fail2ban support to the provisioning server to drastically lower the likelihood of such attacks. Every time a request for a file potentially containing sensitive information is requested, a log line is appended to the `/var/log/xivo-provd-fail2ban.log` file, which is monitored by fail2ban. The same thing happens when a new device is automatically created by the provisioning server.

The fail2ban configuration for the provisioning server is located at `/etc/fail2ban/jail.d/xivo.conf`. You may want to adjust the `findtime` / `maxretry` value if you have special requirements. In particular, if you have many phones behind a NAT equipment, you'll probably have to adjust these values, since every request coming from your phones behind your NAT will appear to the provisioning server as coming from the same source IP address, and this IP address will then be more likely to get banned promptly if you, for example, reboot all your phones at the same time. Another solution would be to add your IP address to the list of ignored IP address of fail2ban. See the fail2ban(1) man page for more information.

System Requirements

XiVO/Wazo 16.08 or later is required. You also need to use compatible xivo-provd plugins. Here's the list of official plugins which are compatible:

Plugin family	Version
xivo-aastra	>= 1.6
xivo-cisco-sccp	>= 1.1
xivo-cisco-spa	>= 1.0
xivo-digium	>= 1.0
xivo-polycom	>= 1.7
xivo-snom	>= 1.6
xivo-yealink	>= 1.26

Remote directory

If you have a phone provisioned with Wazo and its one of the supported ones, you'll be able to search in your XiVO directory and place call directly from your phone.

See the list of *supported devices* to know if a model supports the XiVO directory or not.

Configuration

For the remote directory to work on your phones, the first thing to do is to go to the *Services → IPBX → (General settings) Phonebook* page.

You then have to add the range of IP addresses that will be allowed to access the directory. So if you know that your phone's IP addresses are all in the 192.168.1.0/24 subnet, just click on the small "+" icon and enter "192.168.1.0/24", then save.

You must then restart `xivo-dird-phoned` ("Restart Dird server" from the web interface will not work):

```
systemctl restart xivo-dird-phoned
```

Once this is done, on your phone, just click on the "remote directory" function key and you'll be able to do a search in the XiVO directory from it.

Jitsi

Jitsi (http://jitsi.org/) is an opensource softphone (previously SIP Communicator).

Wazo now support Jitsi sofphones provisioning. Here are the steps to follow :

Requirements

This how to needs :

1. Jitsi installed,
2. SIP line created

Add Jitsi plugin on Wazo

Open Wazo Web interface, and go to Configuration tab, Then chose *Provisioning → Plugins menu*, Install the Jitsi plugin you want to use : e.g.:

```
xivo-jitsi-1
```

You can now launch your Jitsi softphone

Configuring Jitsi

1. Launch Jitsi,
2. If you don't have any accounts configured Jitsi will launch a windows and you can click
3. Use online provisioning. Otherwise go to Tools -> Options -> Advanced -> Provisioing, Click on Enable provisioning
4. Select Manually specify a provisioning URI,
5. Enter the folowing URI where <provd_ip> is the VoIP interface IP address of your Wazo and <provd_port> is the provd port (default : 8667)

   ```
   http://<provd_ip>:<provd_port>/jitsi?uuid=${uuid}
   ```

6. When done, quit Jitsi,
7. Launch Jitsi again,
   - You should now be connected with in autoprov mode,
   - You could see a new device in the devices list,
8. You can now provision the phones by typing the provisioning code (you get it in the Lines list),
9. Quit Jitsi again (configuration syncing is not available with the Jitsi plugin)
10. And launch Jitsi again : you should now be connected with you phone account

1.9.30 Security

This page gives an overview of security best practices that should be applied to a Wazo installation. This is not an exhaustive documentation but a starting point that should be read to avoid common security issues.

Most of this page is aimed at servers that are accessible from the Internet.

fail2ban

Wazo comes with a pre-configured fail2ban. Fail2ban will block IP addresses that tried and failed to gain access to the server. There are 3 jails that a configured.

asterisk-xivo

The `asterisk-xivo` jail watches the Asterisk log file for failed registration attempts.

This jail protects against brute force attacks attempting to guess SIP accounts usernames and password.

xivo-provd

The `xivo-provd` jail will block attempts to create new devices and request for configuration files.

This jail has two goals:

- limiting DOS attacks by creating new devices repeatedly
- protecting against brute force attacks attempting to guess configuration file names.

See *Security* for more details.

sshd

The `sshd` jail protects against SSH brute force attacks.

Firewall

Wazo comes with iptables installed but does not configure any security rules. The only interaction Wazo has with iptables are:

- fail2ban

- wazo-upgrade blocks SIP trafic during an upgrade, to avoid SIP phones to become temporarily unusable after the upgrade.

It is highly recommended that you configure firewall rules on your Wazo.

Devices

Your devices, phones and VoIP gateways, should not be accessible from the Internet. If you have no choice, then the passwords should be changed. Most phones have two different passwords: admin and user passwords.

Some devices allow Wazo to change the password from the auto provisioning system. To change the default values from the web-interface go to *Configuration* → *Provisioning* → *Template device*.

For other devices, you need to change the passwords manually.

Open ports

See the list of network ports that are listening to *0.0.0.0* in the *Network* page. Change the service *configurations* for services that do not need to be accessible.

1.9.31 SCCP Configuration

Provisioning

To be able to provision SCCP phones you should :

- activate the *DHCP Server*,
- activate the *DHCP Integration*,

Then install a plugin for SCCP Phone: *Configuration* → *Provisioning* → *Plugins*

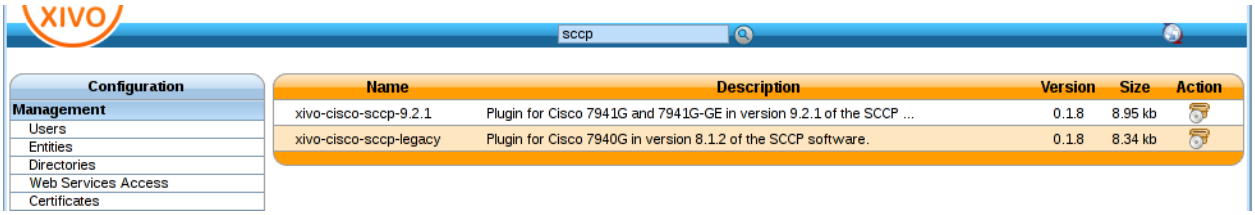


Fig. 1.69: Installing xivo-cisco-sccp plugin

At this point you should have a fully functional DHCP server that provides IP address to your phones. Depending on what type of CISCO phone you have, you need to install the plugin sccp-legacy, sccp-9.4 or both.

Note: Please refer to the *Provisioning page* for more information on how to install CISCO firmwares.

Once your plugin is installed, you'll be able to edit which firmwares and locales you need. If you are unsure, you can choose all without any problem.

Now if you connect your first SCCP phone, you should be able to see it in the device list.

Listing the detected devices: *Services* → *IPBX* → *IPBX settings* → *Devices*

When connecting a second SCCP phone, the device will be automatically detected as well.

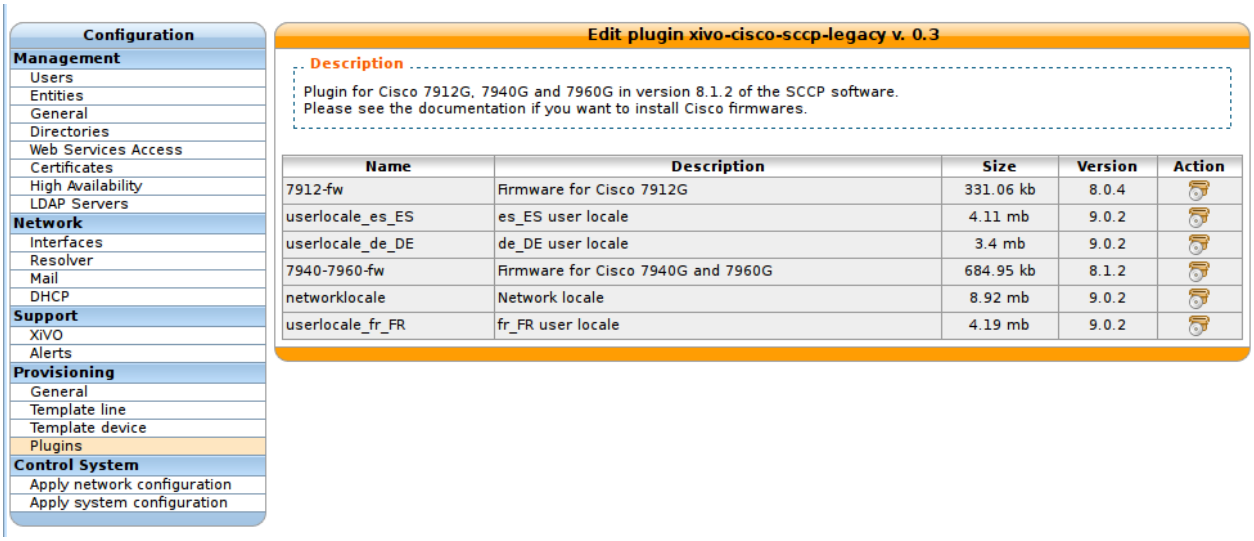


Fig. 1.70: Editing the xivo-cisco-sccp-legacy plugin



Fig. 1.71: Device list

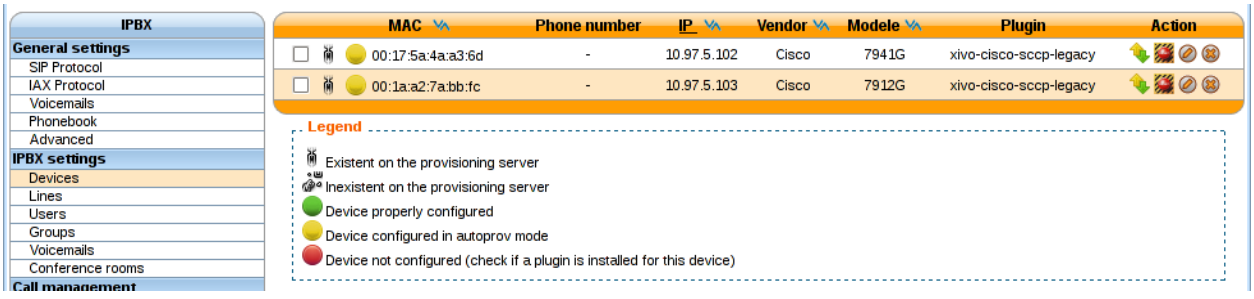


Fig. 1.72: Device list

SCCP General Settings

Review SCCP general settings: *Services → IPBX → IPBX settings → SCCP general settings*

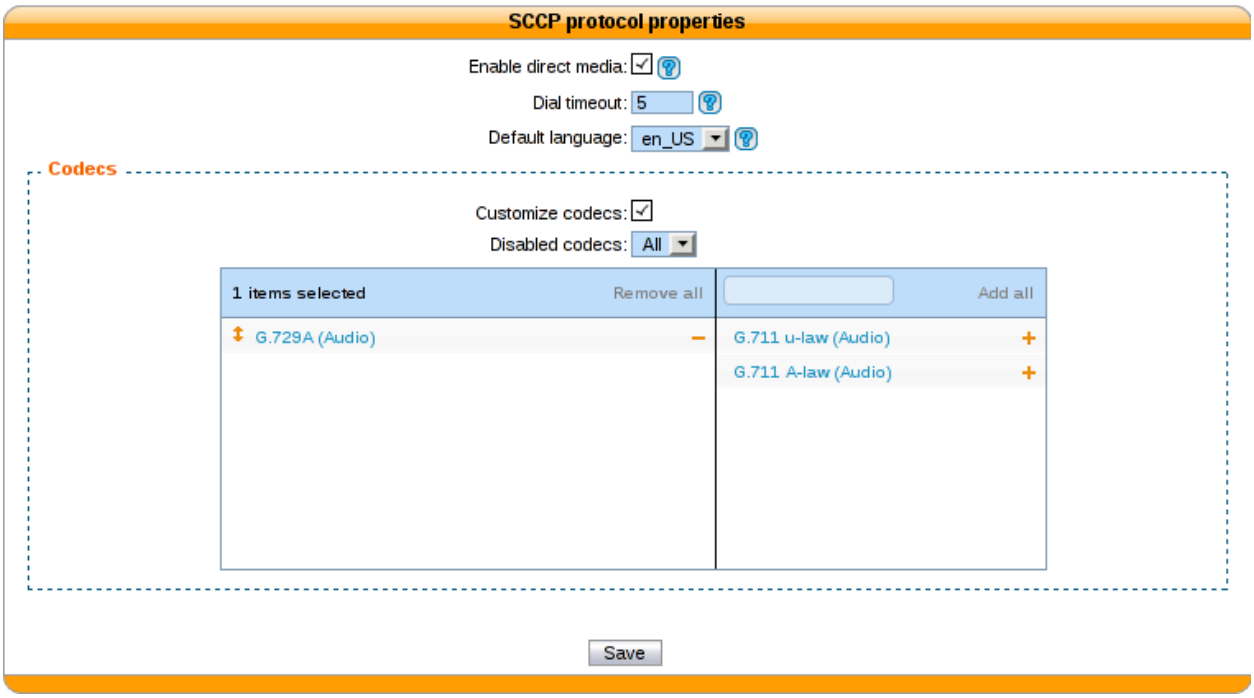


Fig. 1.73: SCCP general settings

User creation

The last step is to create a user with a **SCCP line**.

Creating a user with a SCCP line: *Services → IPBX → IPBX settings → Users*

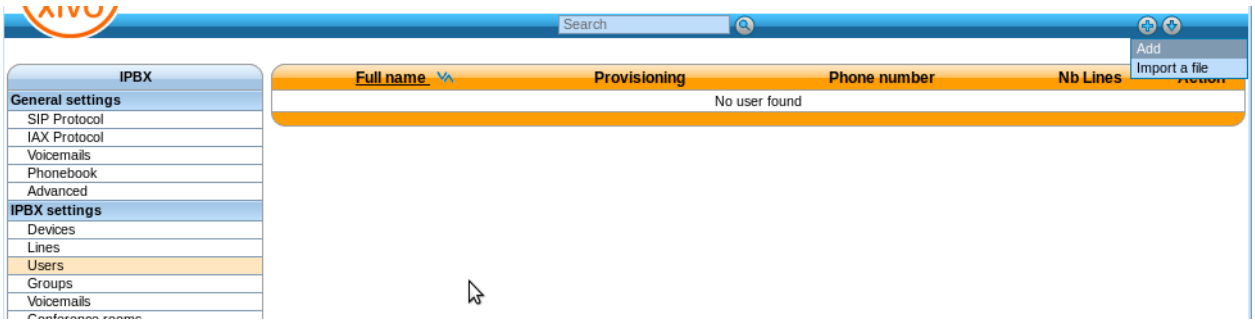


Fig. 1.74: Add a new user

Before saving the newly configured user, you need to select the *Lines* menu and add a SCCP line. Now, you can save your new user.

Congratulations ! Your SCCP phone is now ready to be called !



Fig. 1.75: Edit user informations

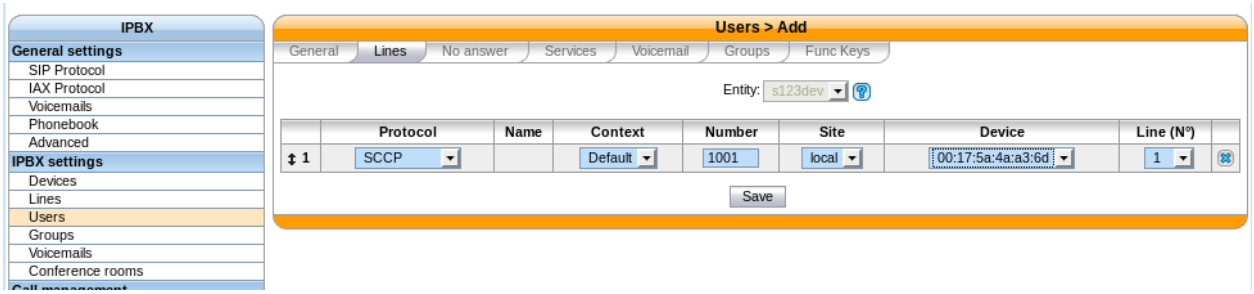


Fig. 1.76: Add a line to a user

Function keys

With SCCP phones, the only function keys that can be configured are:

- *Key*: Only the order is important, not the number
- *Type*: `Customized`; Any other type doesn't work
- *Destination*: Any valid extension
- *Label*: Any label
- *Supervision*: `Enabled` or `Disabled`

Direct Media

SCCP Phones support directmedia (direct RTP). In order for SCCP phones to use directmedia, one must enable the directmedia
Services → IPBX → IPBX settings → SCCP general settings

Features

Features	Supported
Receive call	Yes
Initiate call	Yes
Hangup call	Yes
Transfer call	Yes
Congestion Signal	Yes
Autoanswer (custom dialplan)	Yes
Call forward	Yes
Multi-instance per line	Yes
Message waiting indication	Yes
Music on hold	Yes
Context per line	Yes
Paging	Yes
Direct RTP	Yes
Redial	Yes
Speed dial	Yes
BLF (Supervision)	Yes
Resync device configuration	Yes
Do not disturb (DND)	Yes
Group listen	Yes
Caller ID	Yes
Connected line ID	Yes
Group pickup	Yes
Auto-provisioning	Not yet
Multi line	Not yet
Codec selection	Yes
NAT traversal	Not yet
Type of Service (TOS)	Manual

Telephone

Device type	Supported	Firmware version	Timezone aware
7905	Yes	8.0.3	No
7906	Yes	SCCP11.9-4-2SR1-1	Yes
7911	Yes	SCCP11.9-4-2SR1-1	Yes
7912	Yes	8.0.4(080108A)	No
7920	Yes	3.0.2	No
7921	Yes	1.4.5.3	Yes
7931	Yes	SCCP31.9-4-2SR1-1	Yes
7937	Testing		
7940	Yes	8.1(SR.2)	No
7941	Yes	SCCP41.9-4-2SR1-1	Yes
7941GE	Yes	SCCP41.9-4-2SR1-1	Yes
7942	Yes	SCCP42.9-4-2SR1-1	Yes
7945	Testing		
7960	Yes	8.1(SR.2)	No
7961	Yes	SCCP41.9-4-2SR1-1	Yes
7962	Yes	SCCP42.9-4-2SR1-1	Yes
7965	Testing		
7970	Testing		
7975	Testing		
CIPC	Yes	2.1.2	Yes

Models not listed in the table above won't be able to connect to Asterisk at all. Models listed as "Testing" are not yet officially supported in Wazo: use them at your own risk.

The "Timezone aware" column indicates if the device supports the timezone tag in its configuration file, i.e. in the file that the device request to the provisioning server when it boots. If you have devices that don't support the timezone tag and these devices are in a different timezone than the one of the Wazo, you can look at the issue #5161 for a potential solution.

1.9.32 Schedules

Schedules are specific time frames that can be defined to open or close a service. Within schedules you may specify opening days and hours or close days and hours.

A default destination as user, group ... can be defined when the schedule is in closed state.

Schedules can be applied to :

- Users
- Groups
- Inbound calls
- Outbound calls
- Queues

Creating Schedules

A schedule is composed of a name, a timezone, one or more opening hours or days that you may setup using a calendar widget, a destination to be used when the schedule state is closed.

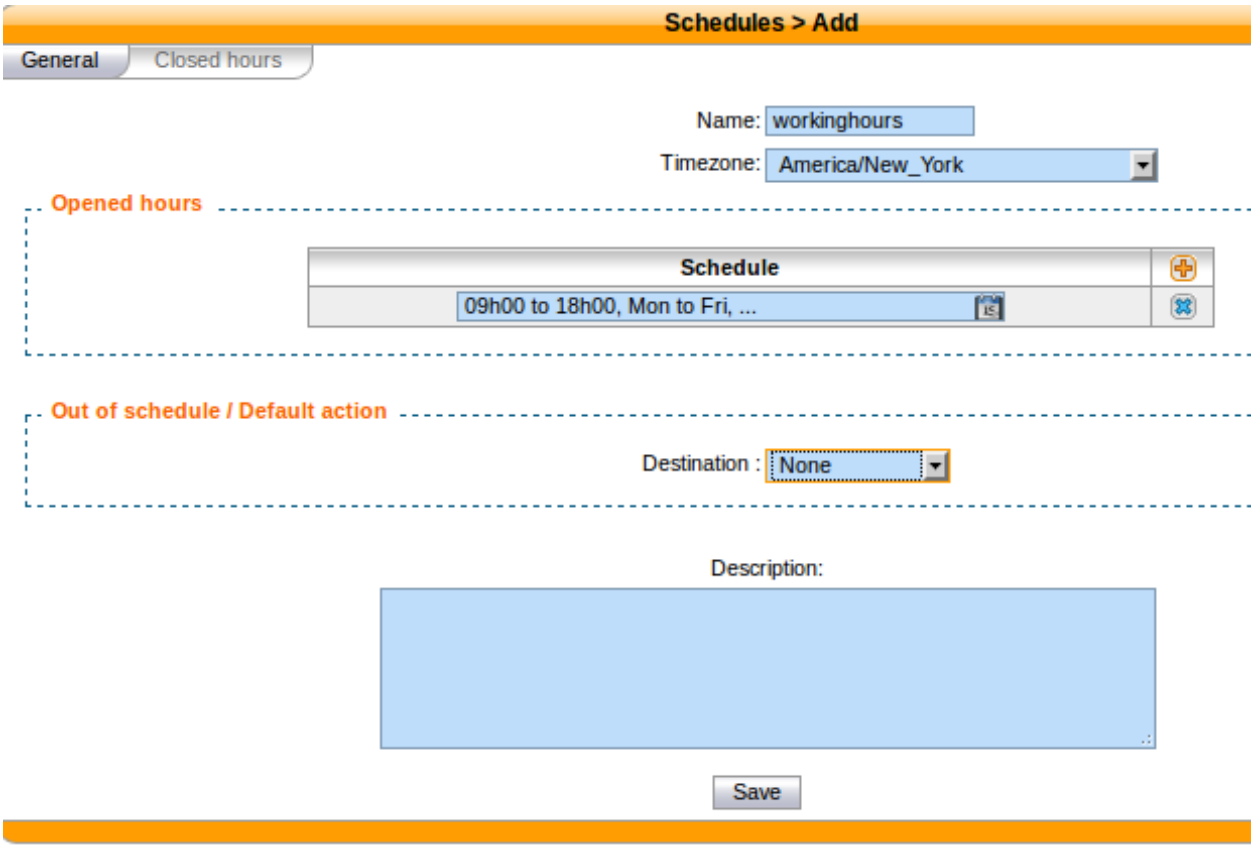


Fig. 1.77: Creating a schedule

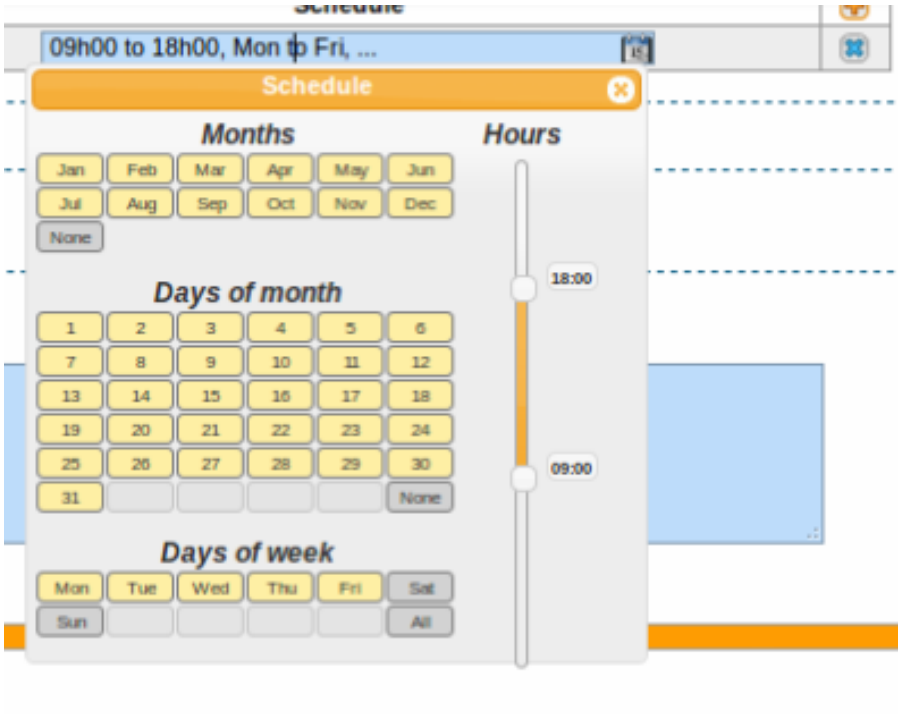


Fig. 1.78: Schedule calendar widget

With the calendar widget you may select months, days of month, days of week and opening time.

You may also optionaly select closed hours and destination to be applied when period is inside the main schedule. For example, your main schedule is opened between 08h00 and 18h00, but you are closed between 12h00 and 14h00.

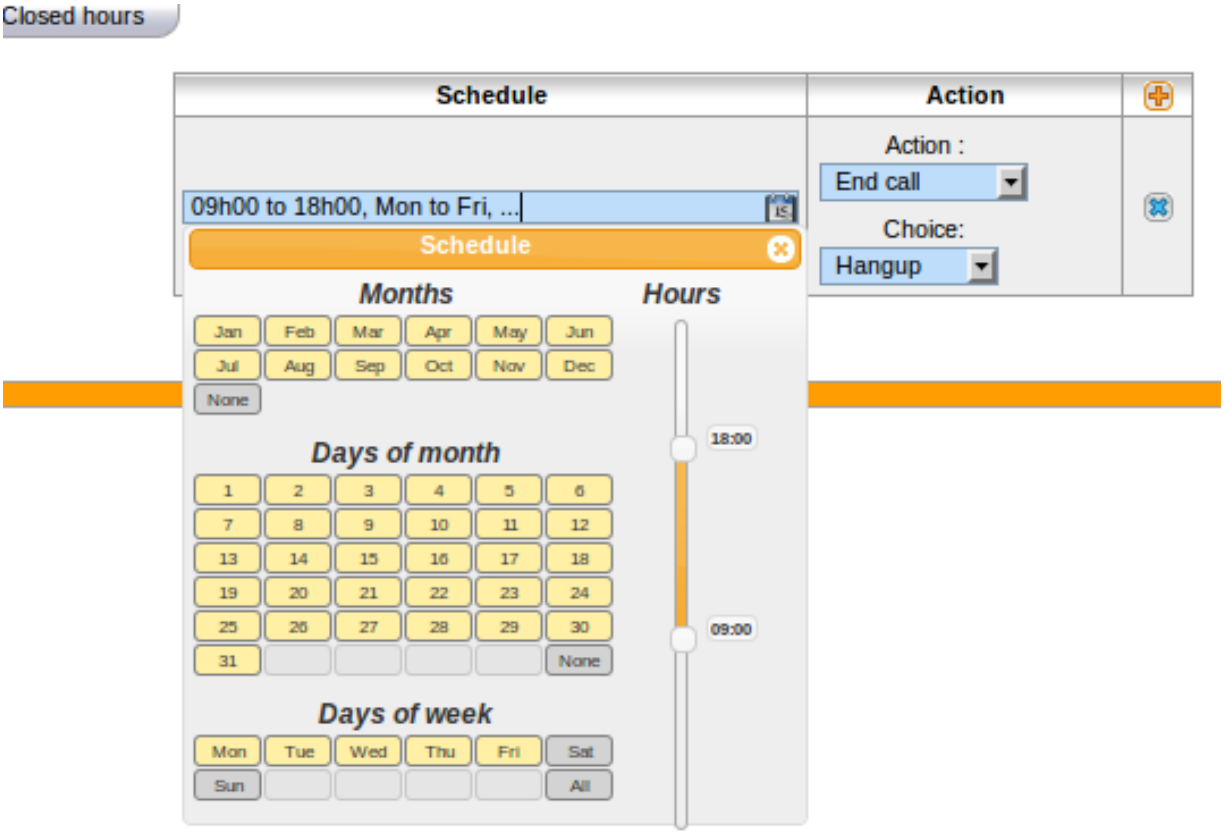


Fig. 1.79: Schedule closed hours

Using Schedule on Users

When you have a schedule associated to a user, if this user is called during a closed period, the caller will first hear a prompt saying the call is being transferred before being actually redirected to the closed action of the schedule.

If you don't want this prompt to be played, you can change the behaviour by:

1. editing the `/etc/xivo/asterisk/xivo_globals.conf` file and setting the `XIVO_FWD_SCHEDULE_OUT_ISDA` to `1`
2. reloading the asterisk dialplan with an `asterisk -rx "dialplan reload"`.

1.9.33 Sound Files

Add Sounds Files

On a fresh install, only en_US and fr_FR sounds are installed. Canadian French and German are available too.

To install Canadian French sounds you have to execute the following command:

```
apt-get install asterisk-sounds-wav-fr-ca xivo-sounds-fr-ca
```

To install German sounds you have to execute the following command:

```
apt-get install asterisk-sounds-wav-de-de xivo-sounds-de-de
```

Now you may select the newly installed language for your users.

Convert Your Wav File

Asterisk will read natively WAV files encoded in wav 8kHz, 16 bits, mono.

The following command will return the encoding format of the <file>

```
$ file <file>
RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 8000 Hz
```

The following command will re-encode the <input file> with the correct parameters for asterisk and write into the <output file>:

```
sox <input file> -b 16 -c 1 -t wav <output file> rate -I 8000
```

1.9.34 Switchboard

This page describes the configuration needed to have a switchboard on your Wazo.

Overview

Switchboard functionality is available in the Wazo Client. The goal of this page is to explain how to configure your switchboard and how to use it.

The switchboard xlet and profile allow an operator to view incoming calls, answer them, put calls on hold, view the calls on hold and pick up the calls on hold.

Limitations

Note: The shortcut keys of the switchboard do not work on the Mac version of the Wazo Client.

Note: The enter shortcut to answer a call will not work if the focus is currently on a widget that will consume the key press. ie: a text field, a drop down

Note: Attended transfers to the switchboard cannot be managed with the switchboard xlets depending on the moment at which the call was completed.

Table of Contents

Switchboard Configuration

Be sure to read the *limitations* before configuring a switchboard.

Server configuration

Quick Summary

In order to configure a switchboard on your Wazo, you need to:

- Create a queue for your switchboard
- Create a queue for your switchboard's calls on hold
- Create the users that will be operators
- Activate the switchboard option for your phone
- Create an agent for your user
- Assign the incoming calls to the switchboard queue
- For each operator, add a function key for logging in or logging out from the switchboard queue.
- Set "no answer" destinations on the switchboard queue

Supported Devices

The supported phones for the switchboard are:

Brand	Model	XiVO version	Plugin version
Aastra	6755i	>= 14.07	>= xivo-aastra-3.3.1-SP2, v1.0
Aastra	6757i	>= 14.07	>= xivo-aastra-3.3.1-SP2, v1.0
Aastra	6735i	>= 14.07	>= xivo-aastra-3.3.1-SP2, v1.2
Aastra	6737i	>= 14.07	>= xivo-aastra-3.3.1-SP2, v1.2
Polycom	VVX 400	>= 15.11	>= xivo-polycom-5.3.0, v1.3
Polycom	VVX 410	>= 15.11	>= xivo-polycom-5.3.0, v1.3
Snom	720	>= 14.14	>= xivo-snom-8.7.3.25.5, v1.0
Snom	D725	>= 14.14	>= xivo-snom-8.7.5.17, v1.4
Yealink	T46G	>= 15.01	>= xivo-yealink-72.0, v1.22.1

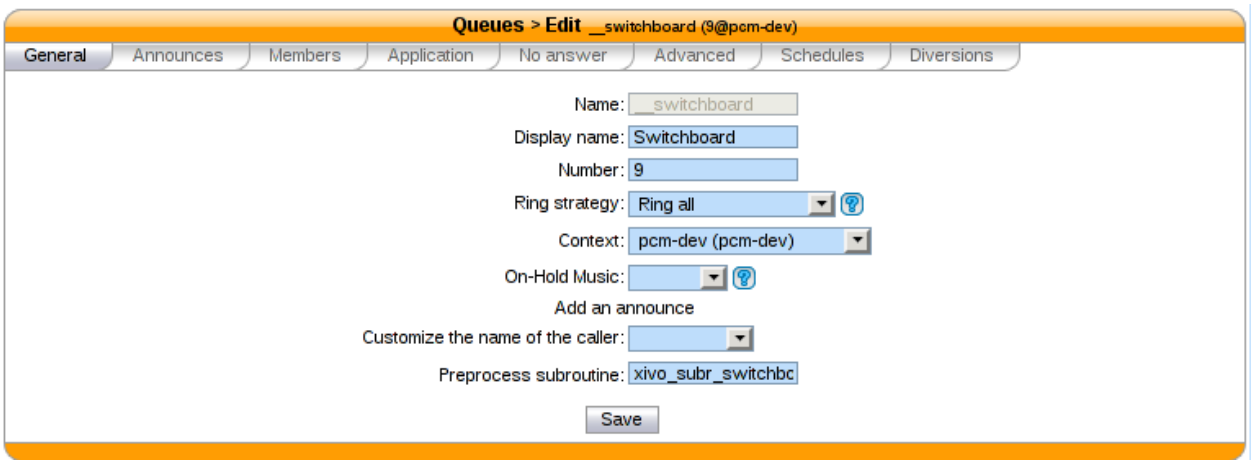
Create a Queue for Your Switchboard

All calls to the switchboard will first be distributed to a switchboard queue.

To create this queue, go to *Services* → *Call center* → *Queues* and click the add button.

The following configuration is mandatory

- The *General* → *Name* field has to be *__switchboard*
- The *General* → *Ring strategy* field has to be *Ring all*
- The *General* → *Preprocess subroutine* field has to be *xivo_subr_switchboard*



- The *Application* → *Enable DTMF hangup by caller* option has to be *enabled*
- The *Application* → *Enable DTMF transfers by callee* option has to be *enabled*
- The *Advanced* → *Member reachability timeout* option has to be *disabled*
- The *Advanced* → *Time before retrying a call to a member* option has to be *1 second*
- The *Advanced* → *Delay before reassigning a call* option has to be *disabled*
- The *Advanced* → *Call a member already on* option has to be *disabled*
- The *Advanced* → *Autopause agents* option has to be *No*

Other important fields

- The *General* → *Display name* field is the name displayed in the Wazo Client xlets and in the statistics
- The *General* → *Number* field is the number that will be used to reach the switchboard internally (typically *9*)

Create a Queue for Your Switchboard on Hold

The switchboard uses a queue to track its calls on hold.

To create this queue, go to *Services* → *Call center* → *Queues* and click the add button.

The following configuration is mandatory

- The *General* → *Name* field has to be *__switchboard_hold*
- The *General* → *Number* field has to be a valid number in a context reachable by the switchboard

Other important fields

- The *General* → *Display name* field is the name displayed in the Wazo Client xlets and in the statistics

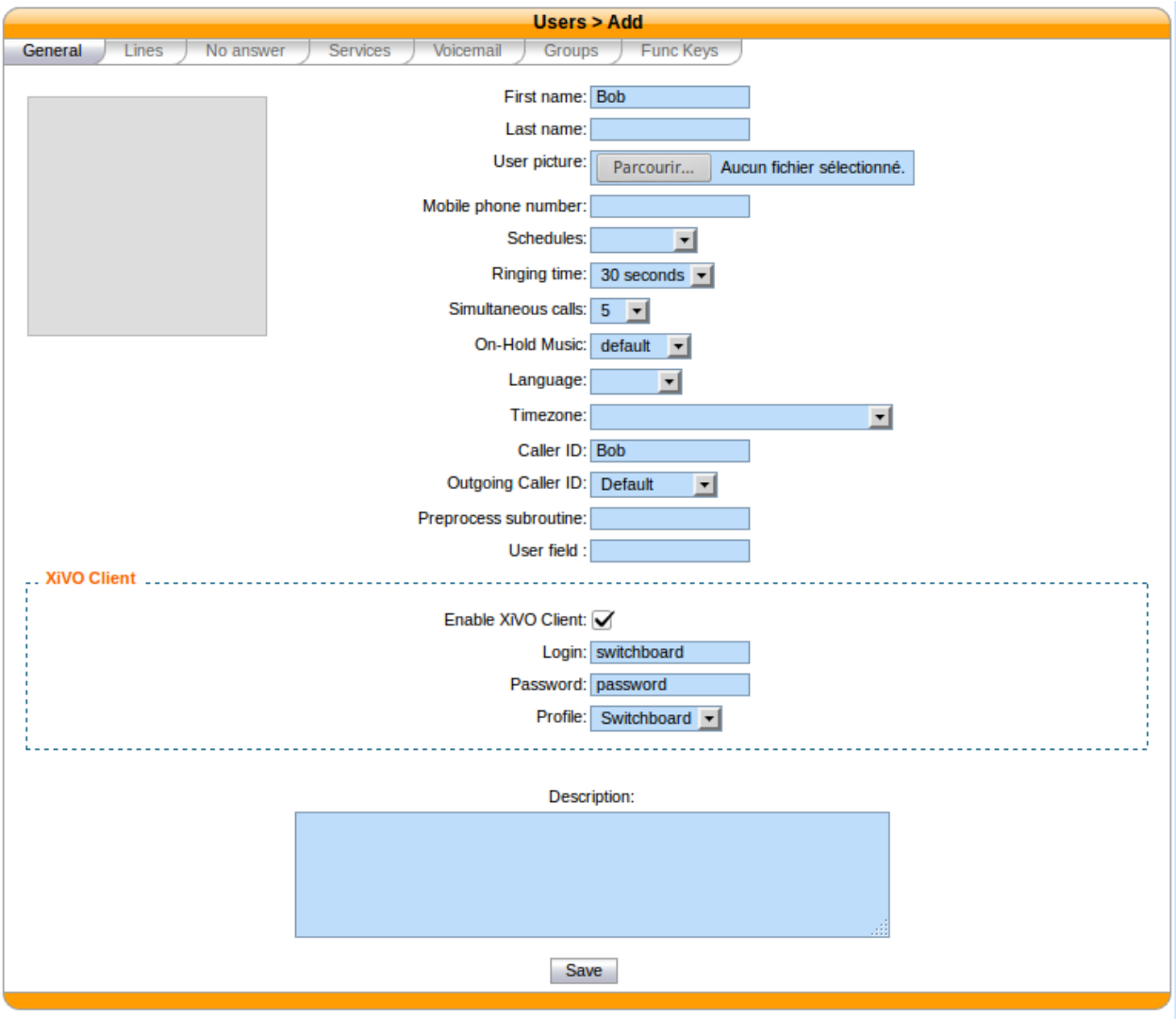
> **Warning:** This queue MUST have **NO** members

Create the Users that Will be Operators

Each operator needs to have a user configured with a line. The Wazo Client profile has to be set to *Switchboard*.

The following configuration is mandatory for switchboard users

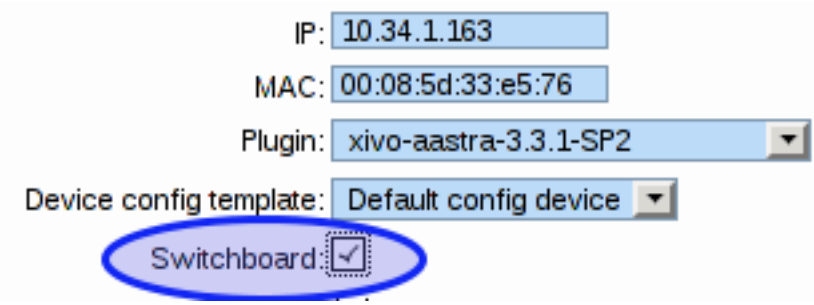
- The *General* → *First name* field has to be set
- The *General* → *Enable Wazo Client* option has to be *enabled*
- The *General* → *Login* field has to be set
- The *General* → *Password* field has to be set
- The *General* → *Profile* field has to be set to *Switchboard*
- The *Lines* → *Number* field has to have a valid extension
- The *Lines* → *Device* field has to be a *supported device*
- The *Services* → *Enable DTMF transfers* option has to be *enabled*
- The *Services* → *Enable supervision* option has to be *enabled*



Activate the Switchboard Option for your Phone

The switchboard option must be activated on the phone. It's possible to activate this option only on *supported phones* and plugins.

- Edit device associated to your user in *Services* → *Devices*
- Check the switchboard checkbox and save
- Synchronize your phone to apply the changes



Polycom Phones

To be able to use a Polycom phone for the switchboard, the Wazo must be able to do HTTP requests to the phone. This might be problematic if there's a NAT between your Wazo and your phone.

It's possible to configure the Polycom switchboard via the *configuration files* of xivo-ctid. The following options are available:

```
switchboard_polycom:
    username: xivo_switchboard
    password: xivo_switchboard
    answer_delay: 0.5
```

You will also need to change the XML API username/password by creating a *custom template* for your phone.

Snom Phones

When using a Snom switchboard, you must not configure a function key on position 1.

To be able to use a Snom phone for the switchboard, the Wazo must be able to do HTTP requests to the phone. This might be problematic if there's a NAT between your Wazo and your phone. The following command should work from your Wazo's bash command line `wget http://guest:guest@<phone IP address>/command.htm?key=SPEAKER`. If this command does not activate the phone's speaker, your network configuration will have to be *fixed* before you can use the Snom switchboard.

It's possible to configure the Snom switchboard via the *configuration files* of xivo-ctid. The following options are available:

```
switchboard_snom:
    username: guest
    password: guest
    answer_delay: 0.5
```

You have to change the username and password option if you have changed the administrator username or administrator password for your phone in *Configuration* → *Provisioning* → *Template Device*.

Yealink Phones

When using a Yealink switchboard, you must not configure a function key on position 1.

Create an Agent for the Operator

Each operator needs to have an associated agent.

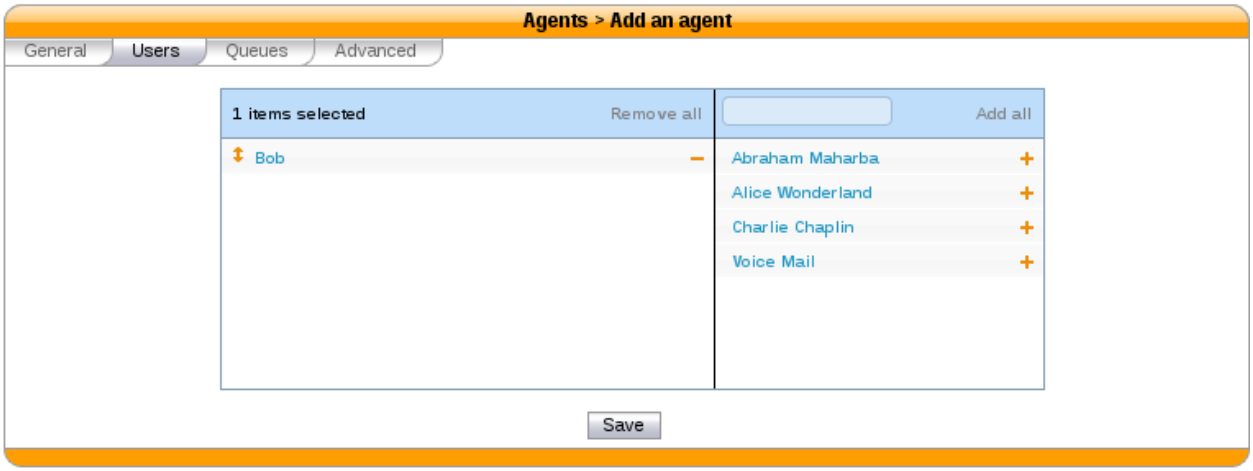
> **Warning:** Each agent MUST ONLY be a member of the Switchboard queue

To create an agent:

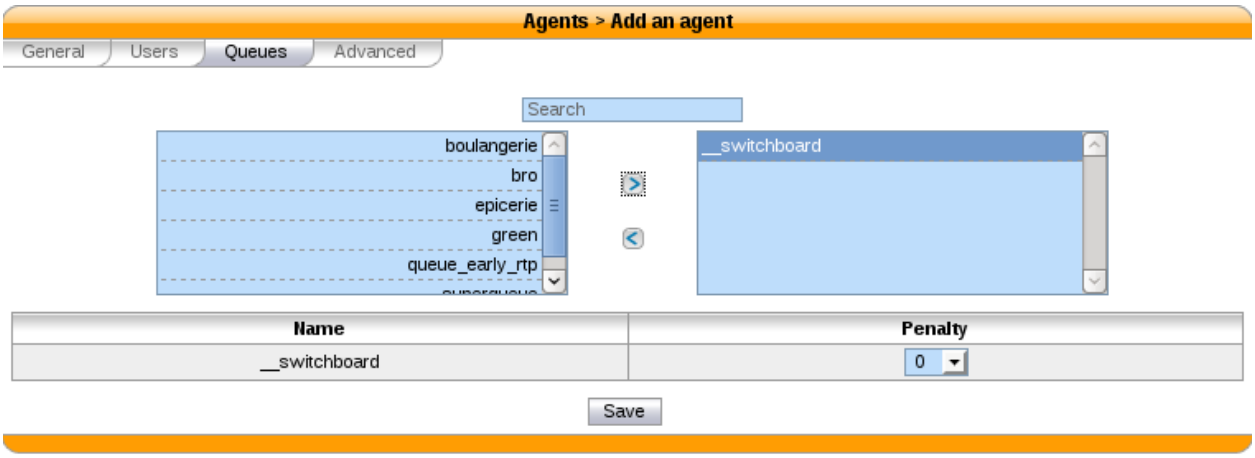
- Go to *Services* → *Call center* → *Agents*
- Click on the group *default*
- Click on the *Add* button



- Associate the user to the agent in the *Users* tab



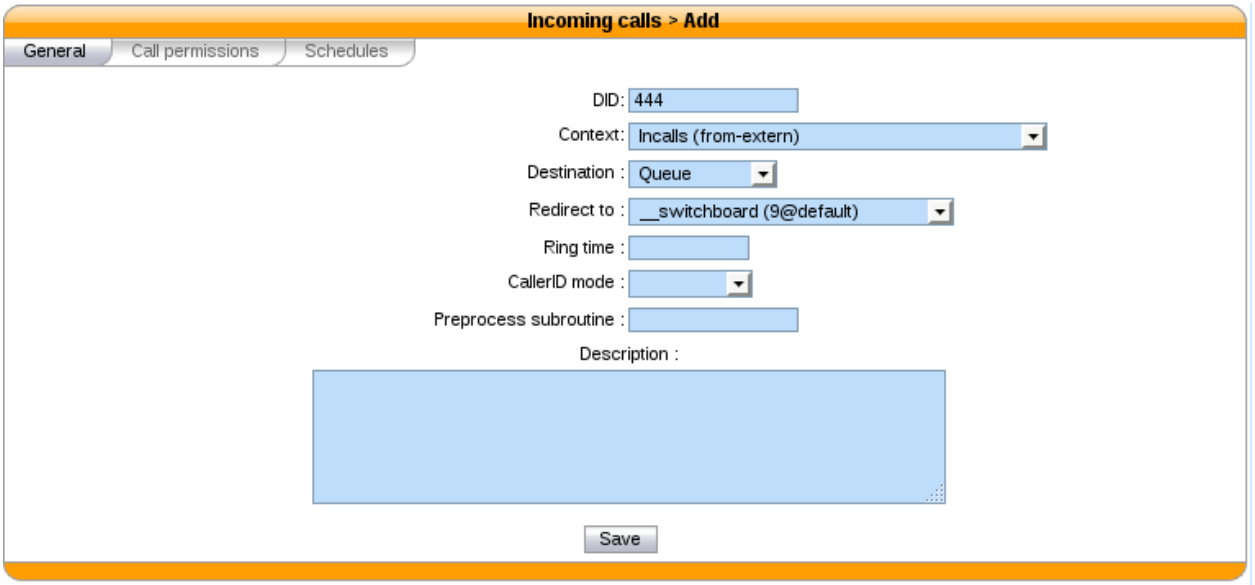
- Assign the Agent to the *Switchboard* Queue (**and ONLY to the Switchboard queue**)



Send Incoming Calls to the *Switchboard* Queue

Incoming calls must be sent to the *Switchboard* queue to be distributed to the operators. To do this, we have to change the destination of our incoming call for the switchboard queue.

In this example, we associate our incoming call (DID *444*) to our *Switchboard* queue:



Set "No Answer" Destinations on the *Switchboard* Queue

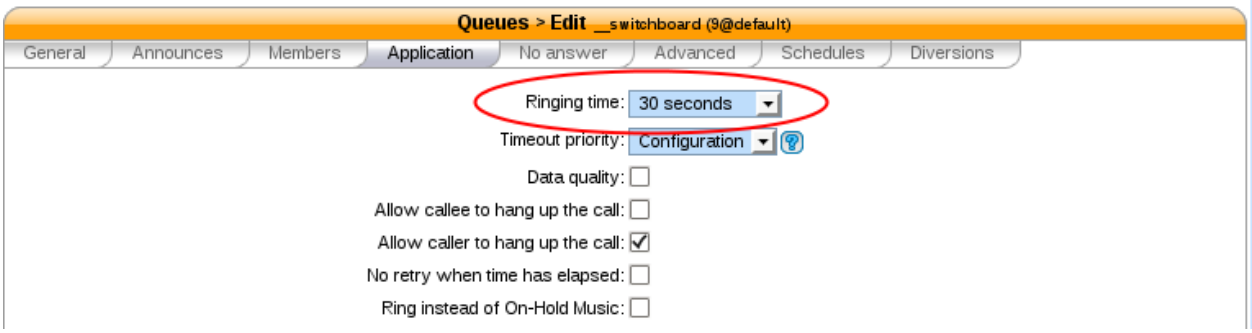
When there are no operators available to answer a call, "No Answer" destinations should be used to redirect calls towards another destination.

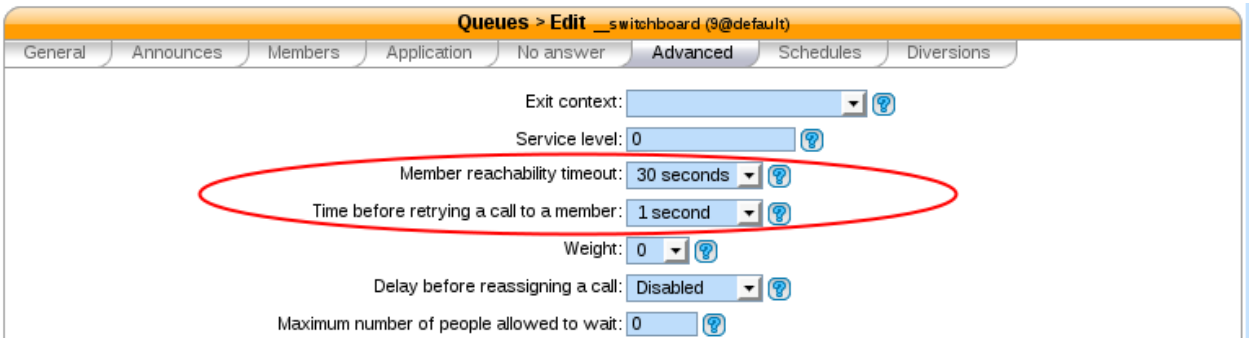
You also need to set the timeout of the Switchboard queue to know when calls will be redirected.

The reachability timeout must not be disabled nor be too short.

The time before retrying a call to a member should be as low as possible (1 second).

In this example we redirect "No Answer", "Busy" and "Congestion" calls to the *everyone* group and "Fail" calls to the *guardian* user.





You can also choose to redirect all the calls to another user or a voice mail.

Wazo Client configuration

Directory xlet

The transfer destination is chosen in the Directory xlet. You **must** follow the *Directory Xlet* section to be able to use it.

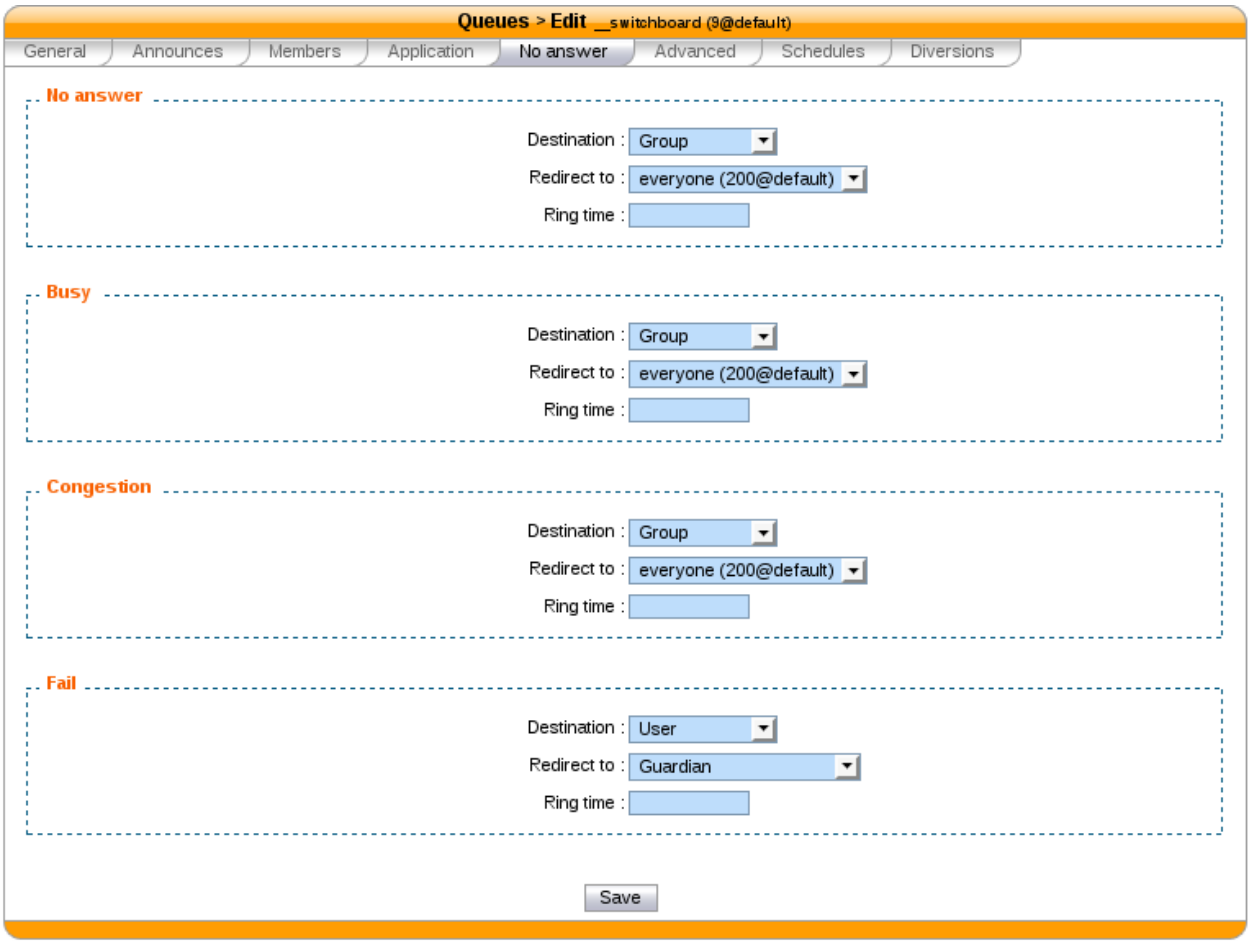
Configuration for multiple switchboards

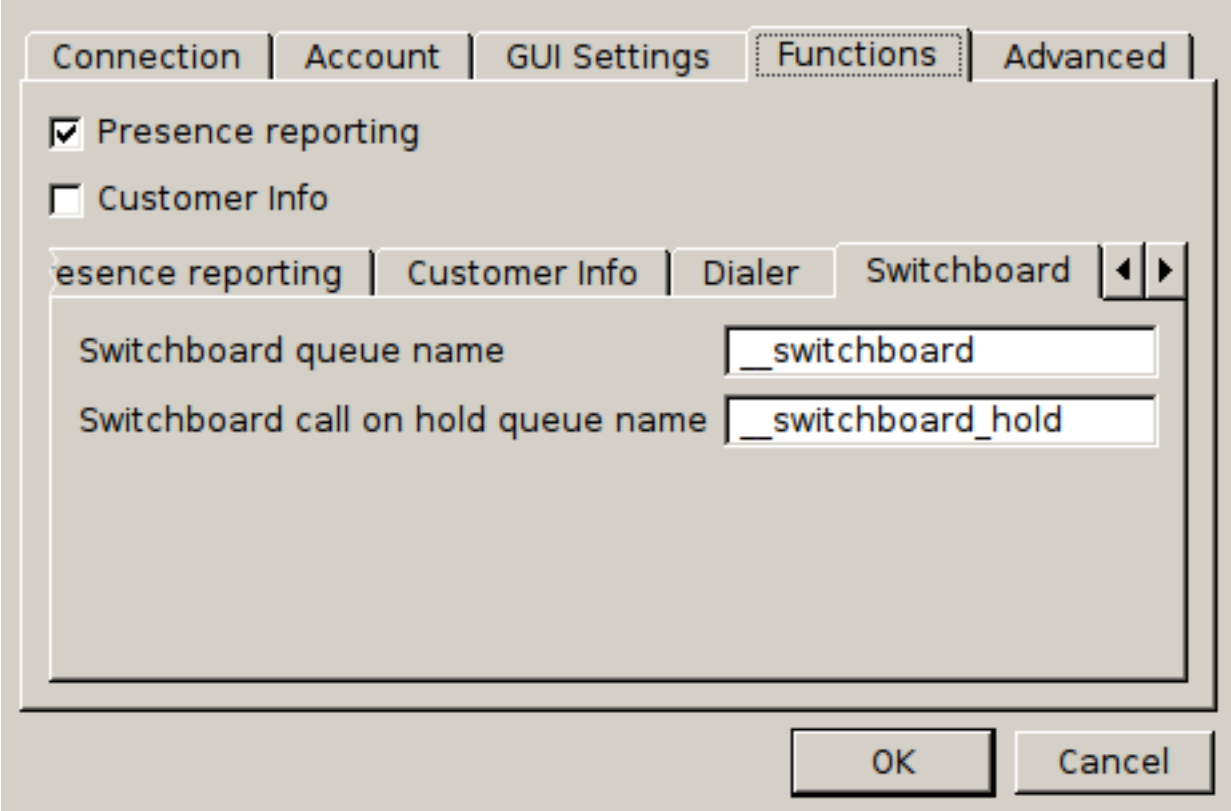
The above documentation can be used for multiple switchboards on the same Wazo by replacing the *__switchboard* and *__switchboard_hold* queues name and configuring the operators Wazo Client accordingly in the *Wazo Client* → *Configure* → *Functions* → *Switchboard* window.

All switchboard queues should be added to the xivo-ctid configuration. New queues can be added by adding a file in `/etc/xivo-ctid/conf.d`. For example, the following content should be used for a new switchboard queue names *__switchboard_two* and an hold queue names *__switchboard_hold_two*.

```
{"switchboard_queues": {"__switchboard_two": true},
 "switchboard_hold_queues": {"__switchboard_hold_two": true}}
```

Switchboard Usage


Queues > Edit _switchboard (9@default)
General
Announces
Members
Application
No answer
Advanced
Schedules
Diversions
No answer
Destination : Group
Redirect to : everyone (200@default)
Ring time :
Busy
Destination : Group
Redirect to : everyone (200@default)
Ring time :
Congestion
Destination : Group
Redirect to : everyone (200@default)
Ring time :
Fail
Destination : User
Redirect to : Guardian
Ring time :
Save



Warning: The *switchboard configuration* must be completed before using the switchboard. This includes :

- Device, User, Agent and Queues configuration (see above),
- Directory xlet configuration (see *Directory Xlet*)

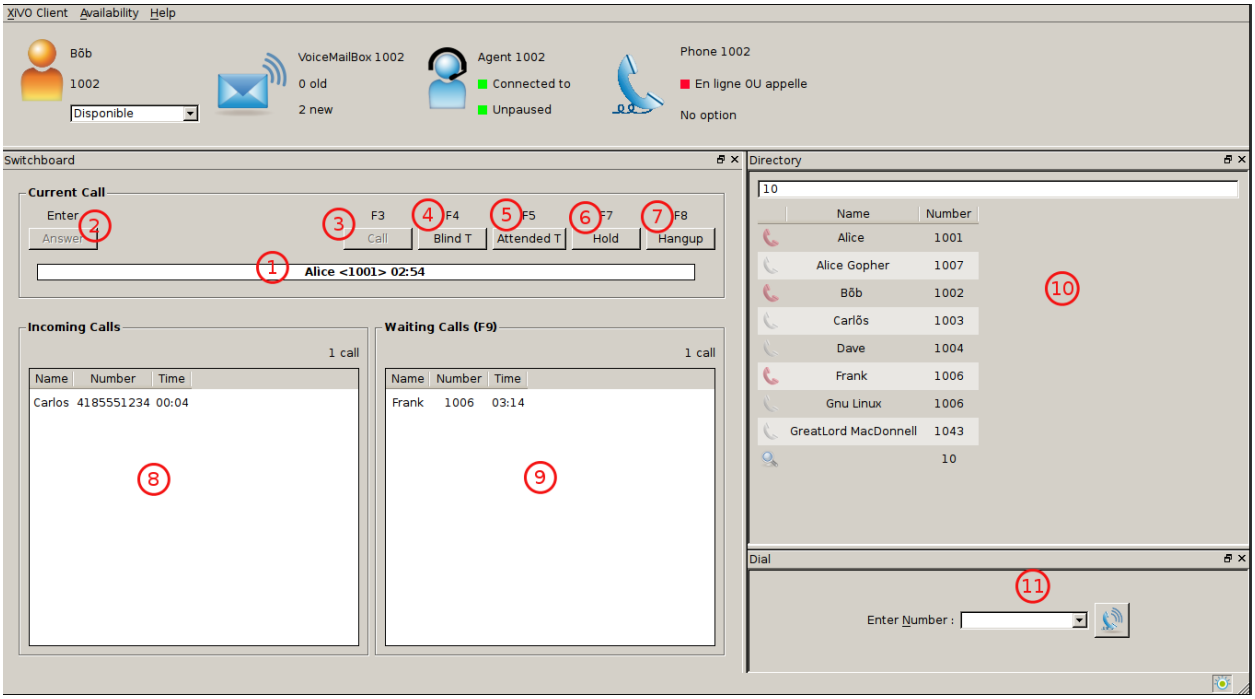
If it's not the case, the user must disconnect his Wazo Client and reconnect.

Be sure to read the *limitations* before using the switchboard.

The Wazo Client Switchboard Profile

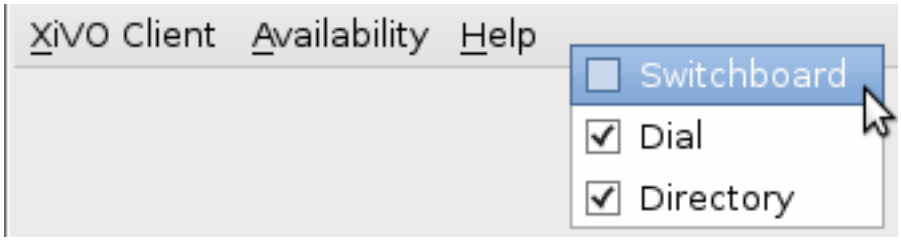
When the user connects with his Wazo Client, he gets the Switchboard profile.

1. *Current Call* frame
2. *Answer* button
3. *Call* button
4. *Blind transfer* button
5. *Attended transfer* button
6. *Hold* button
7. *Hangup* button
8. *Incoming Calls* list
9. *Waiting Calls* list



10. *Directory* Xlet
11. *Dial* Xlet

Note: If you don't see the Switchboard Xlet, right-click on the grey bar at the right of the *Help* menu and check *Switchboard*:



The operator can login his agent using a function key or an extension to start receiving calls.

Call flow

Answering an incoming call

When the switchboard receives a call, the new call is added to the *Incoming Calls* list on the left and the phone starts ringing. The user can answer this call by:

- clicking on any call in the list
- clicking the *Answer* button
- pressing the *Enter* key

Note: The Wazo Client must be the active window for the keyboard shortcuts to be handled

The operator can select which call to answer by:

- clicking directly on the incoming call
- pressing *F6* to select the incoming calls frame and pressing the up and down arrow keys

Selecting a call to answer while talking will not answer the call.

Once the call has been answered, it is removed from the incoming calls list and displayed in the *Current Call* frame.

Making a Call

The switchboard operator can do the following operations:

- Press the *Call* button or press *F3*
- Search for the call destination in the directory xlet
- Press to confirm the selection and start the call

Hanging Up a Call

The switchboard operator can hang up its current call by either:

- Clicking the *Hangup* button
- Pressing the *F8* key

If the operator has placed a new call via the *Directory* or *Dial* xlet and that call has not yet been answered, he can cancel it in the same way.

Distributing a call

Once the call has been answered and placed in the current call frame, the operator has 3 choices:

- transfer the call to another user
  - using the *Blind transfer* button or the *F4* key.
  - using the *Attended transfer* button or the *F5* key
- put the call on hold using the *Hold* button or the *F7* key
- end the call using the *Hangup* button or the *F8* key.

Transferring a call

Transfer buttons allow the operator to select towards which destination he wishes to transfer the call. This is made through the *Directory* xlet. For defails about the xlet *Directory* usage and configuration see *Directory Xlet*.

Once the destination name has been entered, press *Enter*. If multiple destinations are displayed, you can choose by:

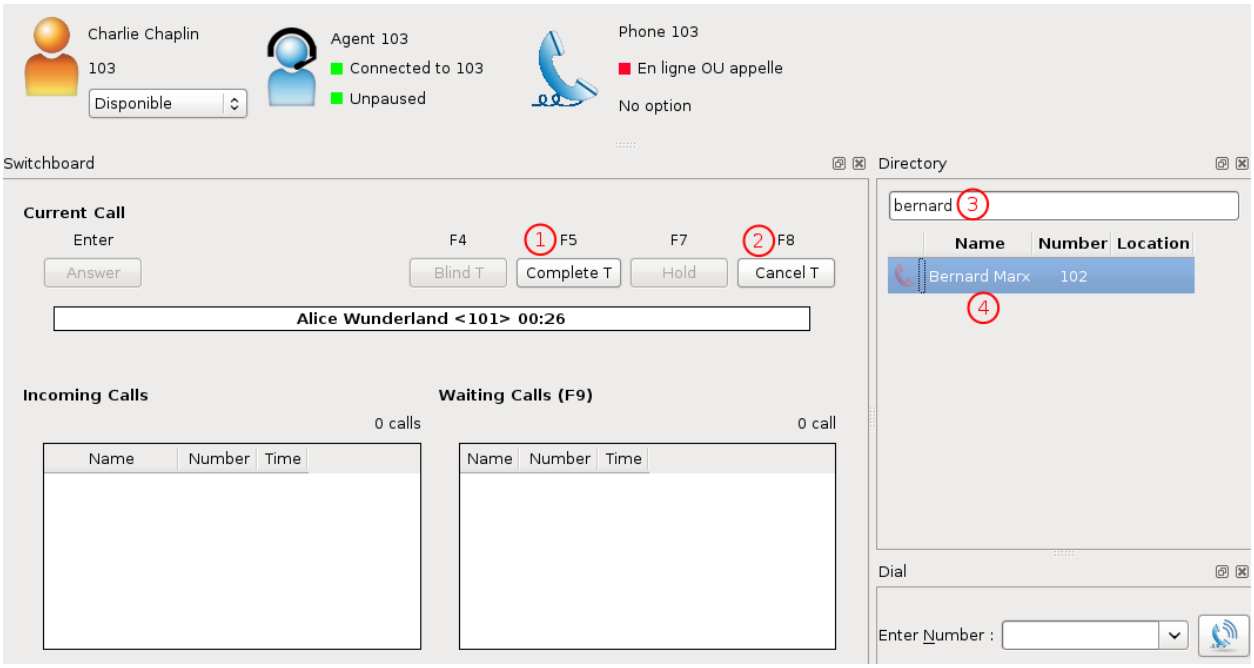
- double-clicking on the destination
- using *Up*/*Down* arrows then:

- pressing *Enter*
- pressing the transfer button again

Blind transfers are straightforward: once the call is transferred, the operator is free to manage other calls.

Attended transfers are a bit more complicated: the operator needs to wait for the transfer destination to answer before completing the transfer.

In this example, the operator is currently asking *Bernard Marx* if he can transfer *Alice Wonderland* to him.



1. *Complete transfer* button
2. *Cancel transfer* button
3. Transfer destination filtering field (xlet *Directory*)
4. Transfer destination list (xlet *Directory*)

Once the destination has answered, you can:

- cancel the transfer with *F8* key
- complete the transfer with *F5* key

Note: The operator can not complete an attended transfer while the transfer destination is ringing. In this case, the operator must cancel the attended transfer and use the *Blind transfer* action.

Putting a call on hold

If the user places the call on hold, it will be removed from the *Current call* frame and displayed in the *Waiting calls* list. The time counter shows how long the call has been waiting, thus it will be reset each time the call returns in the *Waiting calls* list. The calls are ordered from the oldest to the newest.

Retrieving a call on hold

Once a call has been placed on hold, the operator will most certainly want to retrieve that call later to distribute it to another destination.

To retrieve a call on hold:

- click the desired call in the *Waiting calls* list
- with the keyboard:
  - move the focus to the *Waiting calls* list (*F9* key)
  - choose the desired call with the arrow keys
  - press the *Enter* key.

Once a call has been retrieved from the *Waiting calls* list, it is moved back into the *Current Call* frame, ready to be distributed.

1.9.35 Users

Users Configuration.

User Import and Export

CSV Import

Users can be imported and associated to other resources by use of a CSV file. CSV Importation can be used in situations where you need to modify many users at the same in an efficient manner, or for migrating users from one system to another. A CSV file can be created and edited by spreadsheet tools such as Excel, LibreOffice/OpenOffice Calc, etc.

CSV file

The first line of a CSV file contains a list of field names (also sometimes called "columns"). Each new line afterwards are users to import. CSV data must respect the following conditions:

- Files must be encoded in UTF-8
- Fields must be separated with a `,`
- Fields can be optionally quoted with a `"`
- Double-quotes can be escaped by writing them twice (e.g. `Robert ""Bob"" Jenkins`)
- Empty fields or headers that are not defined will be considered null.
- Fields of type *bool* must be either `0` for false, or `1` for true.
- Fields of type *int* must be a positive number

In the following tables, columns have been grouped according to their resource. Each resource is created and associated to its user when all required fields for that resource are present.

User

Field	Type	Required	Values	Description
entity_id	int	Yes		Entity ID (Defined in menu *Configuration* → *Management* → *Entities*)
firstname	string	Yes		User's firstname
lastname	string			User's lastname
email	string			User's email
language	string		de_DE, en_US, es_ES, fr_FR, fr_CA	User's language
mobile_phone_number	string			Mobile phone number
outgoing_caller_id	string			Customize outgoing caller id for this user
enabled	bool			Enable/Disable the user
supervision_enabled	bool			Enable/Disable supervision
call_transfer_enabled	bool			Enable/Disable call transfers by DTMF
dtmf_hangup_enabled	bool			Enable/Disable hangup by DTMF
simultaneous_calls	int			Number of calls a user can have on his phone simultaneously
ring_seconds	int		Must be a multiple of 5	Number of seconds a call will ring before ending
call_permission_password	string			Overwrite all passwords set in call permissions associated to the user

CTI Profile

Field	Type	Required	Values	Description
cti_profile_enabled	bool	No		Activates the Wazo Client account for this user
username	string	Yes, if profile enabled		Wazo Client username
password	string	Yes, if profile enabled		Wazo Client password
cti_profile_name	string	Yes, if profile enabled		Wazo Client profile (Defined in menu *Services* → *CTI server* → *Profiles*)

Phone

Field	Type	Required	Values	Description
exten	string	Yes		Number for calling the user. The number must be inside the range of acceptable numbers defined for the context
context	string	Yes		Context
line_protocol	string	Yes	sip, sccp	Line protocol
sip_username	string			SIP username
sip_secret	string			SIP secret

Incoming call

Field	Type	Required	Values	Description
incall_exten	string	Yes		Number for calling the user from an incoming call (i.e outside of Wazo). The number must be inside the range of acceptable numbers defined for the context.
incall_context	string	Yes		context used for calls coming from outside of Wazo
incall_ring_seconds	int			Number of seconds a call will ring before ending

Voicemail

Field	Type	Required	Values	Description
voicemail_name	string	Yes		Voicemail name
voicemail_number	string	Yes		Voicemail number
voicemail_context	string	Yes		Voicemail context
voicemail_password	string		A sequence of digits or #	Voicemail password
voicemail_email	string			Email for sending notifications of new messages
voicemail_attach_audio	bool			Enable/Disable attaching audio files to email message
voicemail_delete_messages	bool			Enable/Disable deleting message after notification is sent
voicemail_ask_password	bool			Enable/Disable password checking

Call permissions

Field	Type	Required	Values	Description
call_permissions	string		list separated by semicolons (;)	Names of the call permissions to assign to the user

Importing a file

Once your file is ready, you can import it via *Services* → *IPBX* → *IPBX settings* → *Users*. At the top of the page there is a plus button. A submenu will appear when the mouse is on top. Click on Import a file.

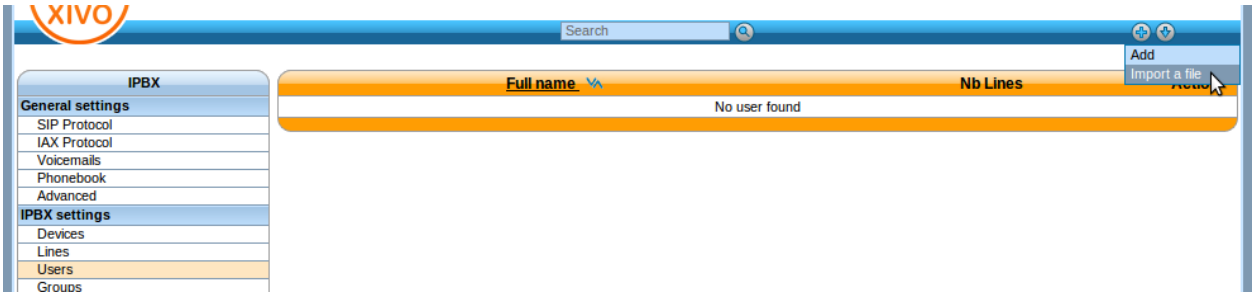


Fig. 1.80: Import Users

Examples

The following example defines 3 users who each have a phone number. The first 2 users have a SIP line, where as the last one uses SCCP:

```
entity_id,firstname,lastname,exten,context,line_protocol
1,John,Doe,1000,default,sip
1,George,Clinton,1001,default,sip
1,Bill,Bush,1002,default,sccp
```

The following example imports a user with a phone number and a voicemail:

```
entity_id,firstname,lastname,exten,context,line_protocol,voicemail_name,voicemail_
↪number,voicemail_context
1,John,Doe,1000,default,sip,Voicemail for John Doe,1000,default
```

The following exmple imports a user with both an internal and external phone number (e.g. incoming call):

```
entity_id,firstname,lastname,exten,context,line_protocol,incall_exten,incall_context
1,John,Doe,1000,default,sip,2050,from-extern
```

CSV Update

The field list for an update is the same as for an import with the addition of the column uuid, which is mandatory. For each line in the CSV file, the updater goes through the following steps:

1. Find the user, using the uuid

2. For each resource (line, voicemail, extension, etc) find out if it already exists.
3. If an existing resource was found, associate it with the user. Otherwise, create it.
4. Update all remaining fields

The following restrictions must also be respected during update:

- Columns that are not included in the CSV header will not be updated.
- A field that is empty (i.e, "") will be converted to NULL, which will unset the value.
- A line's protocol cannot be changed (i.e you cannot go from "sip" to "sccp" or vice-versa).
- An incall cannot be updated if the user has more than one incall associated.

Updating is done through the same menu as importing (*Services → IPBX → IPBX settings → Users*). A submenu will appear when the mouse is on top. Click on *Update from file* in the submenu.

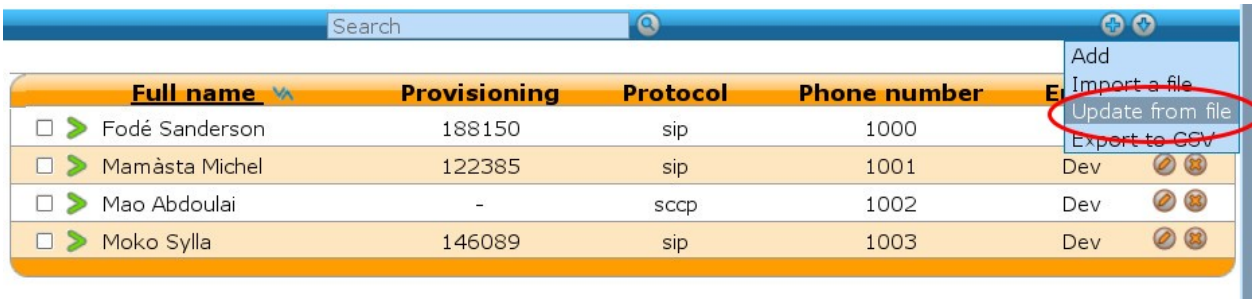


Fig. 1.81: *Services → IPBX → IPBX settings → Users → Update from file*

CSV Export

CSV exports can be used as a scaffold for updating users, or as a means of importing users into another system. An export will generate a CSV file with the same list of columns as an import, with the addition of uuid and provisioning_code.

Exports are done through the same menu as importing (*Services → IPBX → IPBX settings → Users*). Click on *Export to CSV* in the submenu. You will be asked to download a file.

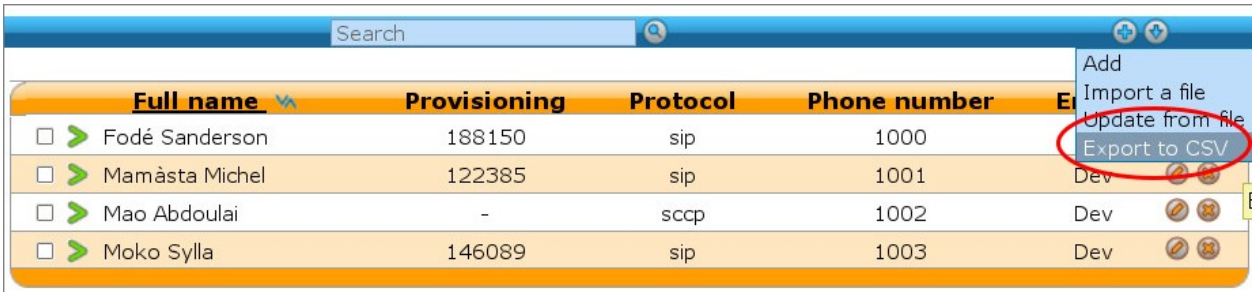


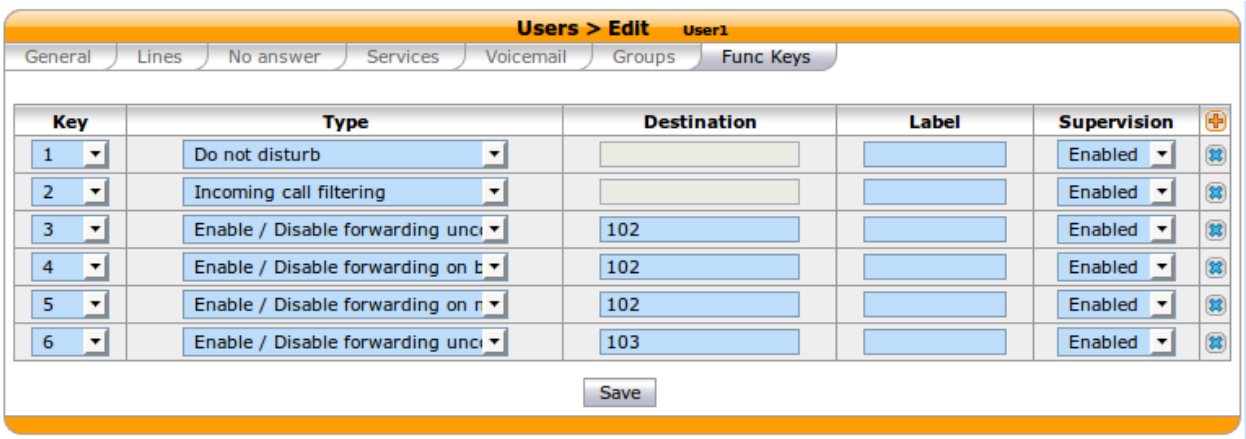
Fig. 1.82: *Services → IPBX → IPBX settings → Users → Export to CSV*

Function keys

Function keys can be configured to customize the user's phone keys. Key types are pre-defined and can be browsed through the Type drop-down list. The Supervision field allows the key to be supervised. A supervised key will light

up when enabled. In most cases, a user cannot add multiple times exactly the same function key (example : two user function keys pointing to the same user). Adding the same function key multiple times can lead to undefined behavior and generally will delete one of the two function keys.

Warning: SCCP device only supports type "Customized".



For User keys, start to key in the user name in destination, Wazo will try to complete with the corresponding user.

If the forward unconditional function key is used with no destination the user will be prompted when the user presses the function key and the BLF will monitor *ALL* unconditional forward for this user.

Extensions

*3 (online call recording)

To enable online call recording, you must check the "Enable online call recording" box in the user form.

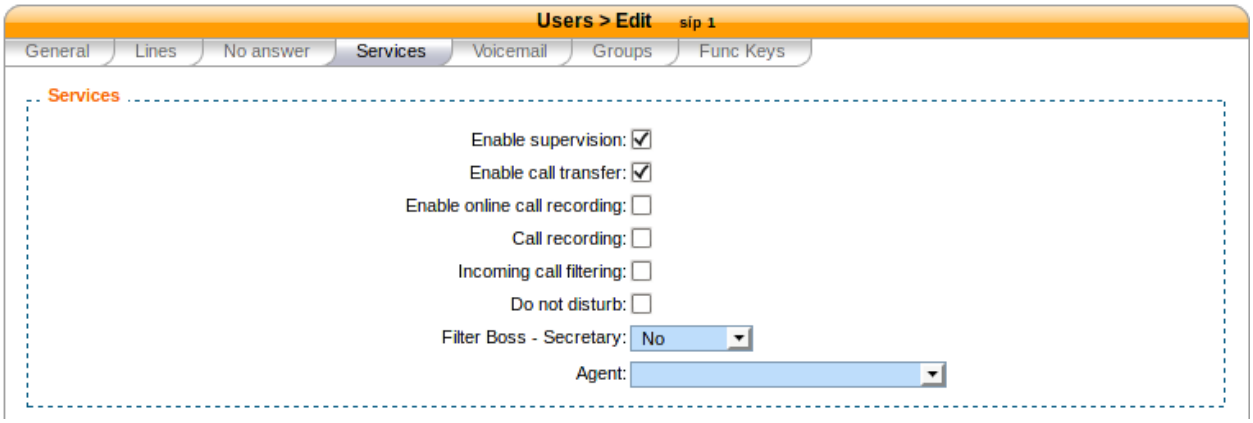


Fig. 1.83: Users Services

When this option is activated, the user can press `*3` during a conversation to start/stop online call recording. The recorded file will be available in the `monitor` directory of the *Services* → *IPBX* → *Audio files* menu.

*26 (call recording)

You can enable/disable the recording of all calls for a user in 2 different way:

1. By checking the "Call recording" box of the user form.

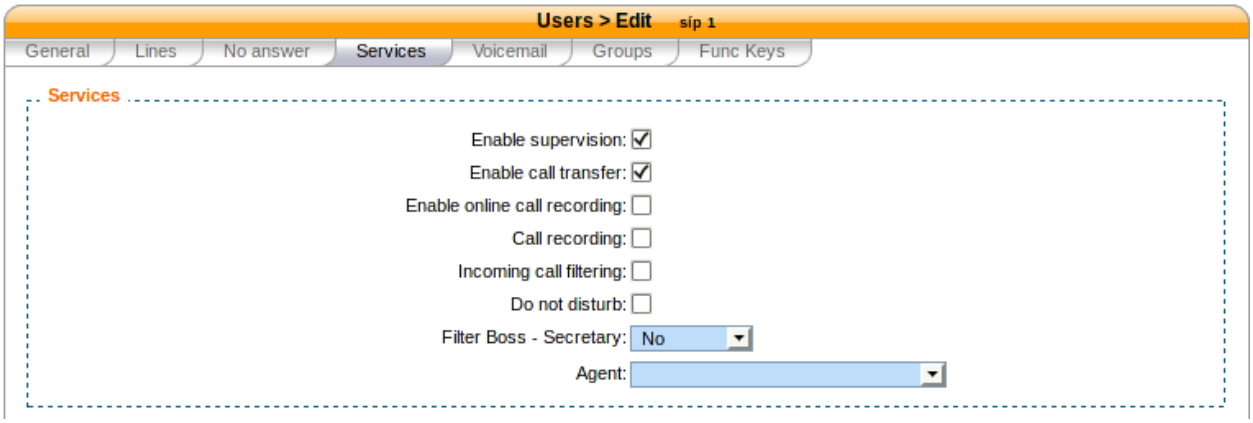


Fig. 1.84: Users Services

2. By using the extension **26* from your phone (the "call recording" option must be activated in *Services → IPBX → Extensions*).

When this option is activated, all calls made to or made by the user will be recorded in the `monitor` directory of the *Services → IPBX → Audio files* menu.

1.9.36 Voicemails

Voicemail Configuration.

General Configuration

The global voicemail configuration is located under *Services → IPBX → General Settings → Voicemails*.

To customize the email sent when a voicemail is received, you can use a few variables. The complete list is available on the Asterisk wiki.

Adding voicemails

There are 2 ways to add a voicemail:

- *Using Services → IPBX → IPBX settings → Voicemails*
- *Using the user's configuration*

Using *Services → IPBX → IPBX settings → Voicemails*

New voicemails can be added using the + button.

Once your voicemail is configured, you have to edit the user configuration and search the voicemail previously created and then associate it to your user.

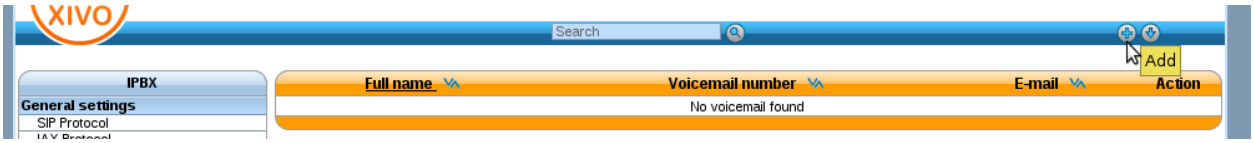


Fig. 1.85: Add voicemails from voicemail menu

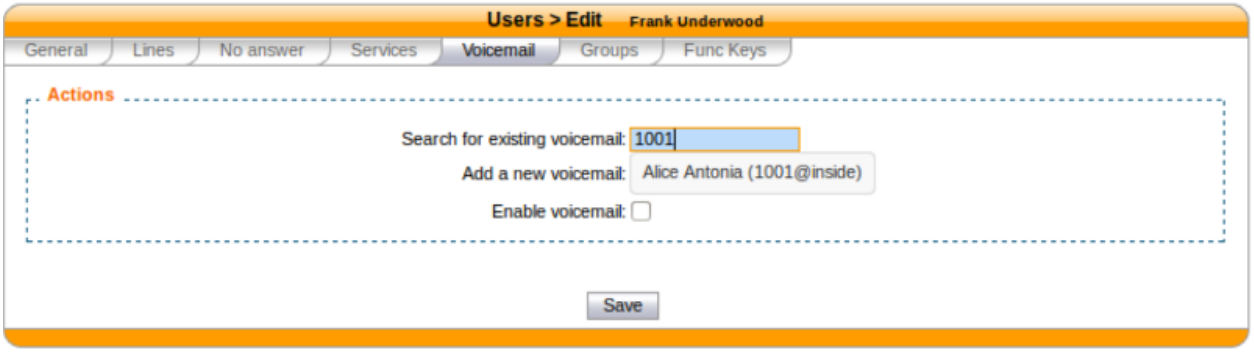


Fig. 1.86: Search for a voicemail in the user's configuration

Using the user's configuration

The other way is to add the voicemail from user's configuration in the 'voicemail' tab by

1. Clicking the + button
2. Filling the voicemail form
3. Saving

Note: The user's language *must* be set in the *general* tab

Disabling a voicemail

You can disable a user's voicemail by un-checking the 'Enable voicemail' option on the Voicemail tab from user's configuration.

Deleting a voicemail

Delete voicemail is done on *Services* → *IBX* → *IPBX settings* → *Voicemails* or from the user's *voicemail* tab.

Note:

- Deleting a voicemail is irreversible. It deletes all messages associated with that voicemail.
- If the voicemail contains messages, the message waiting indication on the phone will not be deactivated until the next phone reboot.

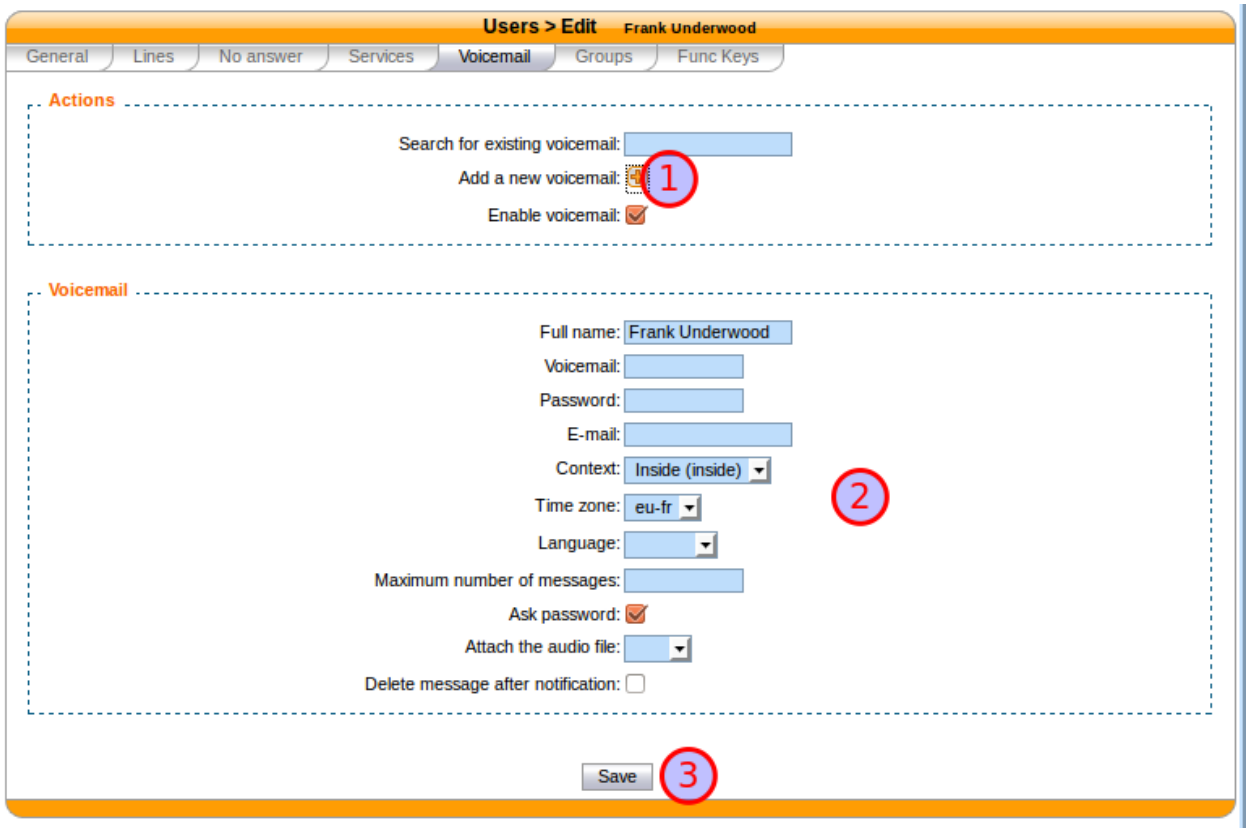


Fig. 1.87: Add a voicemail from the user's configuration

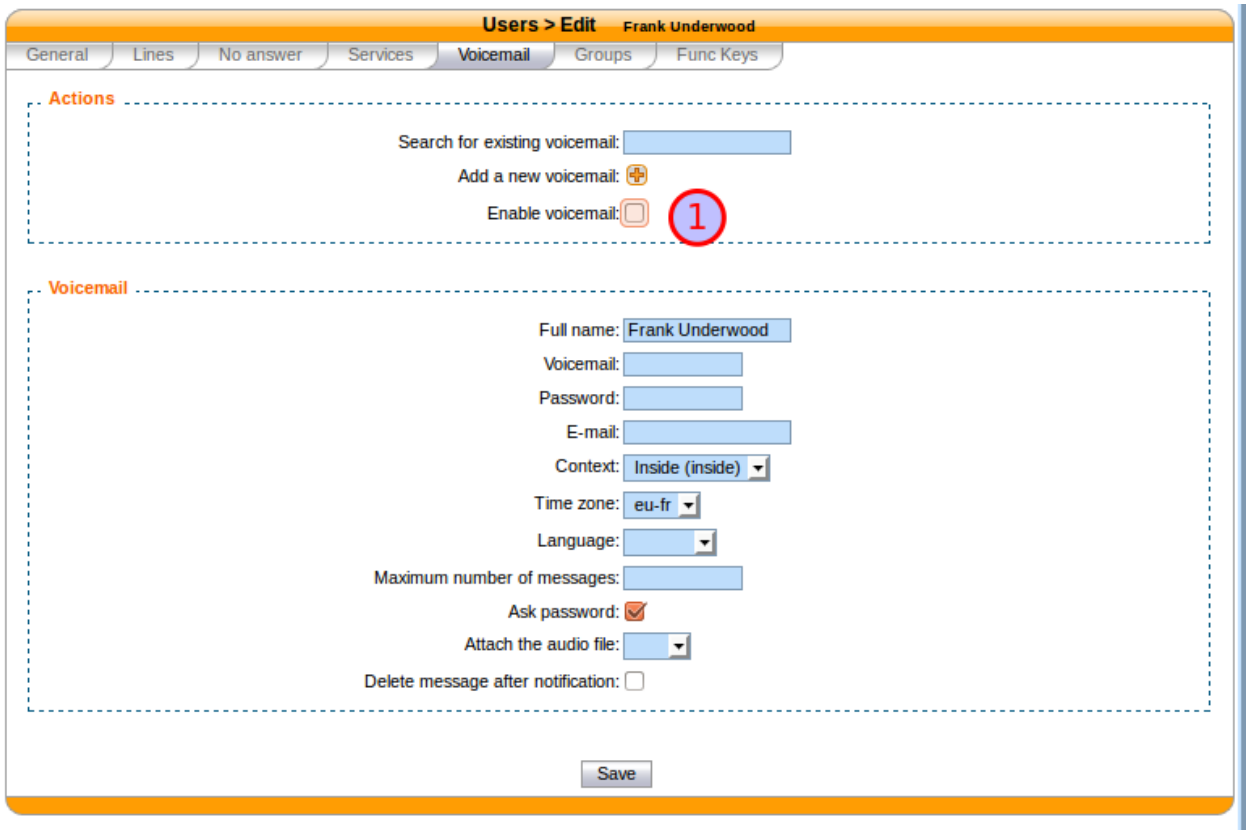


Fig. 1.88: Deactivate user's voicemail

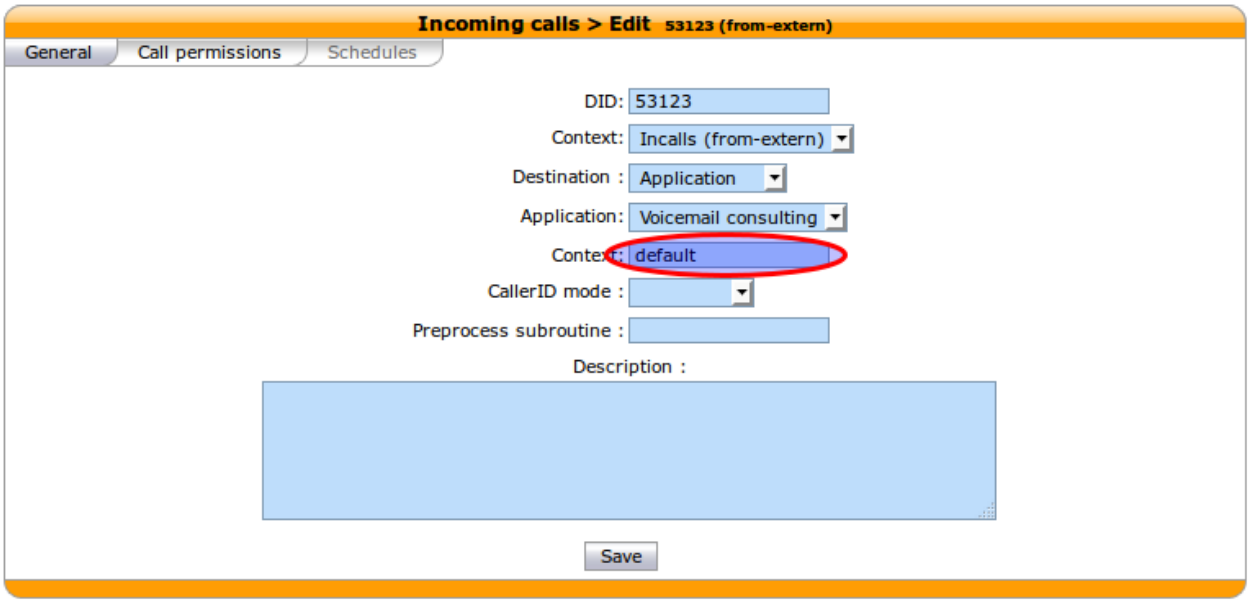
Disable password checking

Unchecking the option `Ask password` allows you to skip password checking for the voicemail only when it is consulted from an internal context.

- when calling the voicemail with *98
- when calling the voicemail with *99<voicemail number>

Warning: If the the *99 extension is enabled and a user does not have a password on its voicemail, anyone from the same context will be able to listen to its messages, change its password and greeting messages.

However, the password will be asked when the voicemail is consulted through an incoming call. For instance, let's consider the following incoming call:



With such a configuration, when calling this incoming call from the outside, we will be asked for:

- the voicemail number we want to consult
- the voicemail password, **even if the "Disable password checking option" is activated**

And then, we will be granted access to the voicemail.

Take note that the second "context" field contains the context of the voicemail. Voicemails of other contexts will not be accessible through this incoming call.

Warning: For security reasons, such an incoming call should be avoided if a voicemail in the given context has no password.

Advanced configuration

Remote *xivo-confd*

If *xivo-confd* is on a remote host, *xivo-confd-client* configuration will be required to be able to change the voicemail passwords using a phone.

This configuration should be done:

```
mkdir -p /etc/systemd/system/asterisk.service.d
cat >/etc/systemd/system/asterisk.service.d/remote-confd-voicemail.conf <<EOF
[Service]
Environment=CONFD_HOST=localhost
Environment=CONFD_PORT=9486
Environment=CONFD_HTTPS=true
Environment=CONFD_USERNAME=<username>
Environment=CONFD_PASSWORD=<password>
EOF
systemctl daemon-reload
```

1.9.37 Web Services Access

You may configure Web Services / REST API permissions in *Configuration* → *Management* → *Web Services Access*.

Web services access may have two different meanings:

- Who may access REST APIs of various Wazo daemons, and which resources in those REST APIs?
- Who may access PHP web services under `https://wazo.example.com/xivo/configuration/json.php/*`?

REST API access and permissions

Those REST API interfaces are documented on http://api.wazo.community. They all require an authorization token, obtained by giving valid credentials to the REST API of wazo-auth. The relevant settings are:

- Login/Password: the wazo-auth credentials (for the wazo-auth *backend* `xivo_service`)
- ACL: The list of authorized REST API resources. See *REST API Permissions*.

Unlike PHP web services, there is no host-based authorization, so the `Host` setting is not relevant.

A few REST API access are automatically generated during the installation of Wazo, so that Wazo services may authenticate each other.

You will probably only need to create such a REST API access when you want another non-Wazo service to communicate with Wazo via REST API.

PHP web services

Warning: DEPRECATED

Those web services are deprecated. There is no documentation about their usage, and the goal is to remove them.

They are still protected with HTTP authentication, requiring a login and password. The relevant settings are:

- Login/Password: the HTTP authentication credentials
- Host: the authorized hosts that are allowed to make HTTP requests:
  - Empty value: HTTP authentication
  - Non-empty value: no HTTP authentication, all requests coming from this host will be accepted. Valid hosts may be: a hostname, an IP address, a CIDR block.

There is no fine-grained permissions: either the user has access to every PHP web services, or none.

xivo-confd

Warning: DEPRECATED

There is also a special case for authentication with xivo-confd. See *REST API* for more details.

1.10 Contact Center

In Wazo, the contact center is implemented to fulfill the following objectives :

- Call routing

  Includes basic call distribution using call queues and skills-based routing

- Agent and Supervisor workstation.

  Provides the ability to execute contact center actions such as: agent login, agent logout and to receive real time statistics regarding contact center status

- Statistics reporting

  Provides contact center management reporting on contact center activities

- Advanced functionalities

  Call recording

- Screen Pop-up

1.10.1 Agents

Introduction

> *A call center agent is the person who handles incoming or outgoing customer calls for a business. A call center agent might handle account inquiries, customer complaints or support issues. Other names for a call center agent include customer service representative (CSR), telephone sales or service representative (TSR), attendant, associate, operator, account executive or team member.*
>
> —SearchCRM

In this respect, agents in Wazo have no fixed line and can login from any registered device.

Getting Started

- Create a user with a SIP line and a provisioned device.
- Create agents.
- Create a queue adding created agent as member of queue.

Creating agents

Service > Call center > Agents > General

These settings are specific for a given agent.

Service > Call center > Agents > Users

These settings are specific for a given agent.

Service > Call center > Agents > Queues

These settings are specific for a given agent.

Service > Call center > Agents > Advanced

These settings are specific for a given agent.

Service > IPBX > General settings > Advanced > Agent

These settings are global for all agents.

1.10.2 Queues

Call queues are used to distribute calls to the agents subscribed to the queue. Queues are managed on the *Services* → *Call Center* → *Queues* page.

A queue can be configured with the following options:

- Name: used as an unique id, cannot be `general`
- Display name: Displayed on the supervisor screen
- On-Hold music: The music the caller will hear. The music is played when waiting and when the call is on hold.

A ring strategy defines how queue members are called when a call enters the queue. A queue can use one of the following ring strategies:

- Linear: For each call, in the same order, starting from the same member
  - For agents: In login order
  - For static members: In definition order
- Least recent: call the member who least recently hung up a call

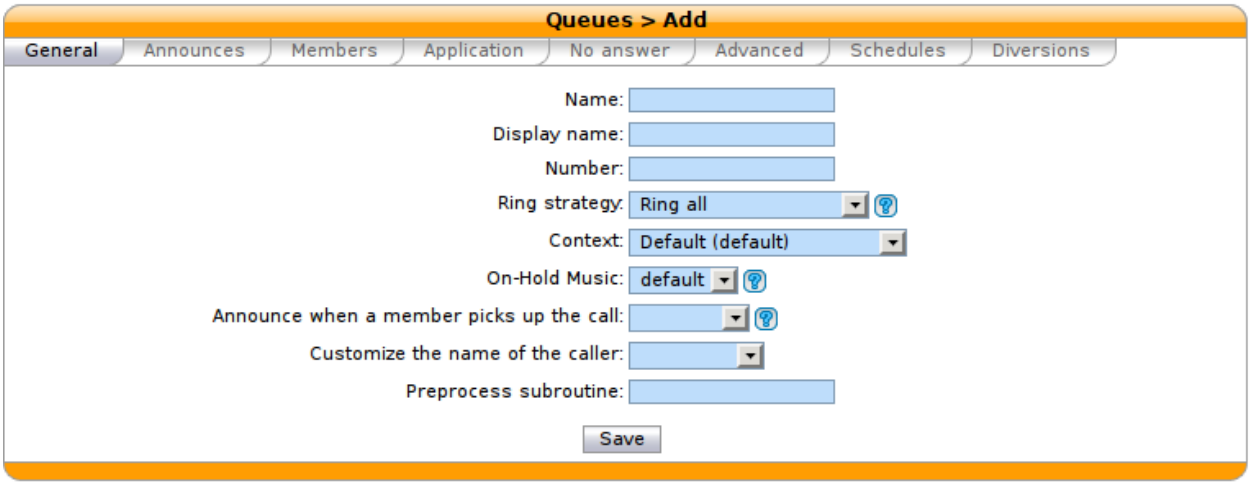


Fig. 1.89: *Services → Call Center → Queues → Add*

- Fewest calls: call the member with the fewest completed calls
- Round robin memory: call the "next" member after the one who answered last
- Random: call a member at random
- Weight random: same as random, but taking the member penalty into account
- Ring all: call all members at the same time

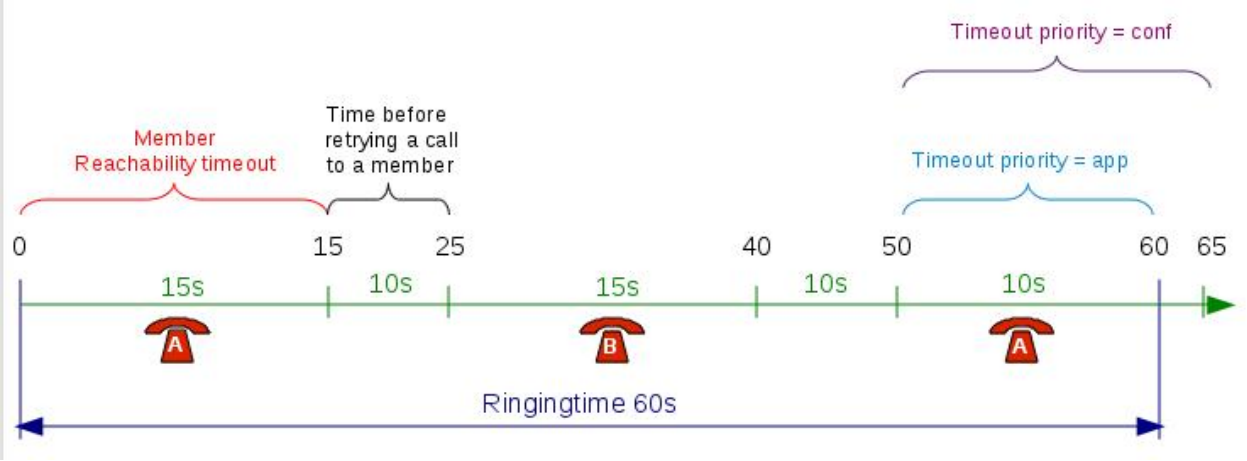
> **Warning:** When editing a queue, you can't change the ring strategy to linear. This is due to an asterisk limitation. Unfortunately, if you want to change the ring strategy of a queue to linear, you'll have to delete it first and then create a new queue with the right strategy.

Note: When an agent is a member of many queues the order of call distribution between multiple queues is nondeterministic and cannot be configured.

Timers

You may control how long a call will stay in a queue using different timers:

- Member reachabillity time out (Advanced tab): Maximum number of seconds a call will ring on an agent's phone. If a call is not answered within this time, the call will be forwarded to another agent.
- Time before retrying a call to a member (Advanced tab): Used once a call has reached the "Member reachability time out". The call will be put on hold for the number of seconds alloted before being redirected to another agent.
- Ringing time (Application tab): The total time the call will stay in the queue.
- Timeout priority (Application tab): Determines which timeout to use before ending a call. When set to "configuration", the call will use the "Member reachability time out". When set to "dialplan", the call will use the "Ringing time".



No Answer

Calls can be diverted on no answer:

- No answer: The call reached the "Ringing time" in Application tab and no agent answered the call
- Congestion: The number of calls waiting has reached the "Maximum number of people allowed to wait" limit specified on the advanced tab
- Fail: No agent was available to answer the call when the call entered the queue ("Join an empty queue" condition on the advanced tab) or the call was queued and no agents were available to answer ("Remove callers if there are no agents" on the advanced tab)

Diversions

Diversions can be used to redirect calls to another destination when a queue is very busy. Calls are redirected using one of the two following scenarios:

The diversion check is done only once per call, before the *preprocess subroutine* is executed and before the call enters the queue.

In the following sections, a waiting call is a call that has entered the queue but has not yet been answered by a queue member.

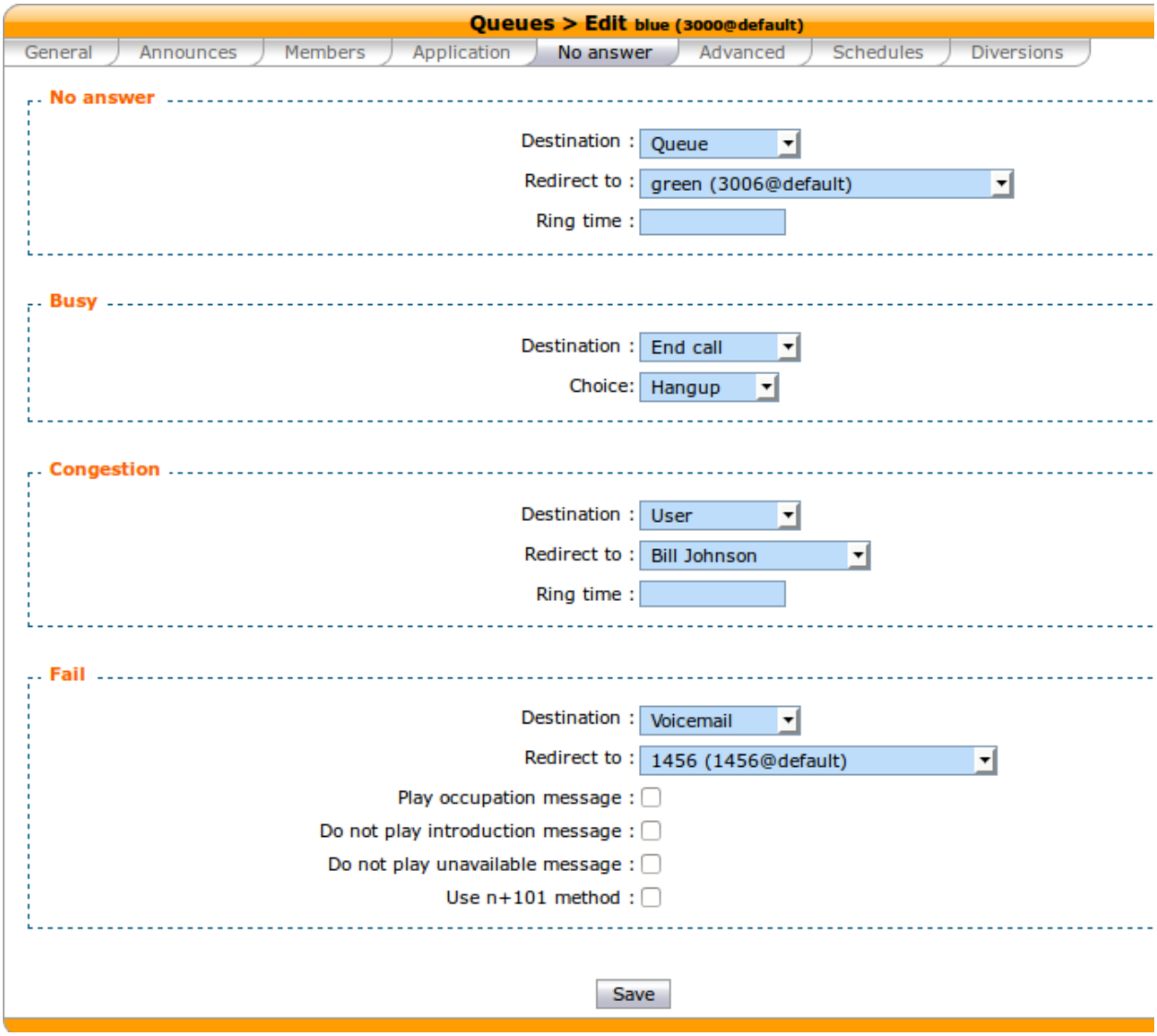
Estimated Wait Time Overrun

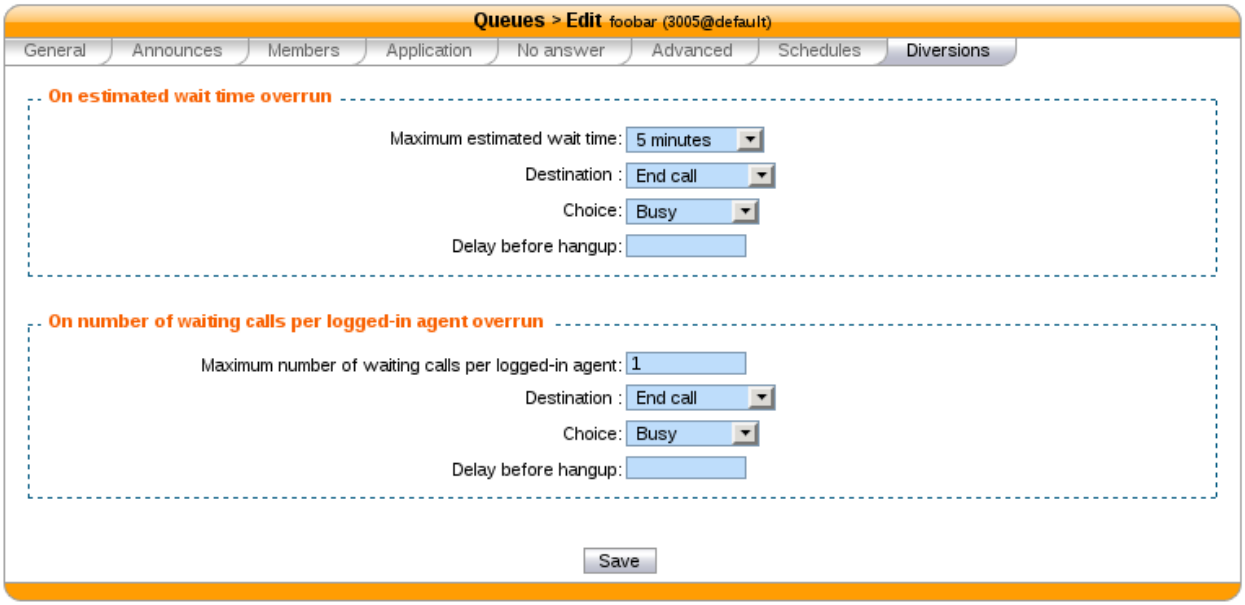
When this scenario is used, the administrator can set a destination for calls to be sent to when the estimated waiting time is over the threshold.

Note that if a new call arrives when there are no waiting calls in the queue, the call will **always** be allowed to enter the queue.

Note:

- this *estimated* waiting time is computed from the **actual hold time** of all **answered** calls in the queue (since last asterisk restart) according to an exponential smoothing formula
- the estimated waiting time of a queue is updated only when a queue member answers a call.


Queues > Edit blue (3000@default)
General
Announces
Members
Application
No answer
Advanced
Schedules
Diversions
No answer
Destination : Queue
Redirect to : green (3006@default)
Ring time :
Busy
Destination : End call
Choice: Hangup
Congestion
Destination : User
Redirect to : Bill Johnson
Ring time :
Fail
Destination : Voicemail
Redirect to : 1456 (1456@default)
Play occupation message :
Do not play introduction message :
Do not play unavailable message :
Use n+101 method :
Save



Number of Waiting Calls per Logged-In Agent Overrun

When this scenario is used, the administrator can set a destination for calls to be sent to when the number of waiting calls per logged-in agent is over the threshold.

The number of waiting calls includes the call for which the check is currently being performed.

The number of logged-in agents is the sum of user members and currently logged-in agent members. An agent only needs to be logged in and a member of the queue to participate towards the count of logged-in agents, regardless of whether he is available, on call, on pause or on wrapup.

The maximum number of waiting calls per logged-in agent can have a fractional part.

Here are a few examples:

```
Maximum number of waiting calls per logged-in agent: 1
Current number of waiting calls: 2
Current number of logged-in agents: 2
Number of waiting calls per logged-in agent when a new call arrives: 3 / 2 = 1.5
Call will be redirected

Maximum number of waiting calls per logged-in agent: 0.5
Number of waiting calls: 5
Number of logged-in agents: 12
Number of waiting calls per logged-in agent when a new call arrives: 6 / 12 = 0.5
Call will not be redirected
```

Note that if a new call arrives when there are no waiting calls in the queue, the call will **always** be allowed to enter the queue. For example, in the following scenario:

```
Maximum number of waiting calls per logged-in agent: 0.5
Current number of waiting calls: 0
Current number of logged-in agents: 1
Number of waiting calls per logged-in agent when a new call arrives: 1 / 1 = 1
```

Even if the number of waiting calls per logged-in agent (1) is greater than the maximum (0.5), the call will still be accepted since there are currently no waiting calls.

Music on Hold

The music on hold of the queue will be played:

- When the caller is waiting to be answered.
- When the caller is put on hold by an agent who already answered.

If you want a different music to be played when the caller is put on hold after being answered, you need to make some more configuration:

1. Write an AGI script that will set the channel variable `CHANNEL(musicclass)` to the name of the music-on-hold class you want the caller to hear when he is put on hold by the agent. Save this script to e.g. `/usr/local/bin/agi-agent-hold-moh`.
2. Add the following *preprocess subroutine* on the queue:

```
[setup-agent-hold-moh]
exten = s,1,NoOp(Setting AGI script for custom agent hold music)
same  =   n,Set(XIVO_QUEUEAGI=/usr/local/bin/agi-agent-hold-moh)
same  =   n,Return
```

This configuration will give the following scenario:

- The caller calls the queue
- The caller hears the music on hold of the queue
- The agent answers the call
- Wazo calls the AGI script, setting the new music on hold
- The caller and the agent talk together
- The agent puts the caller on hold
- The caller hears the new music on hold, set by the AGI script

1.10.3 Supervision

Introduction

Allows a contact center supervisor to monitor contact center activities such as:

- Monitoring real time information from call queues
- Agent activities per call queues
- Agent detailed activities

Wazo Client as a Supervision Platform

Configuration

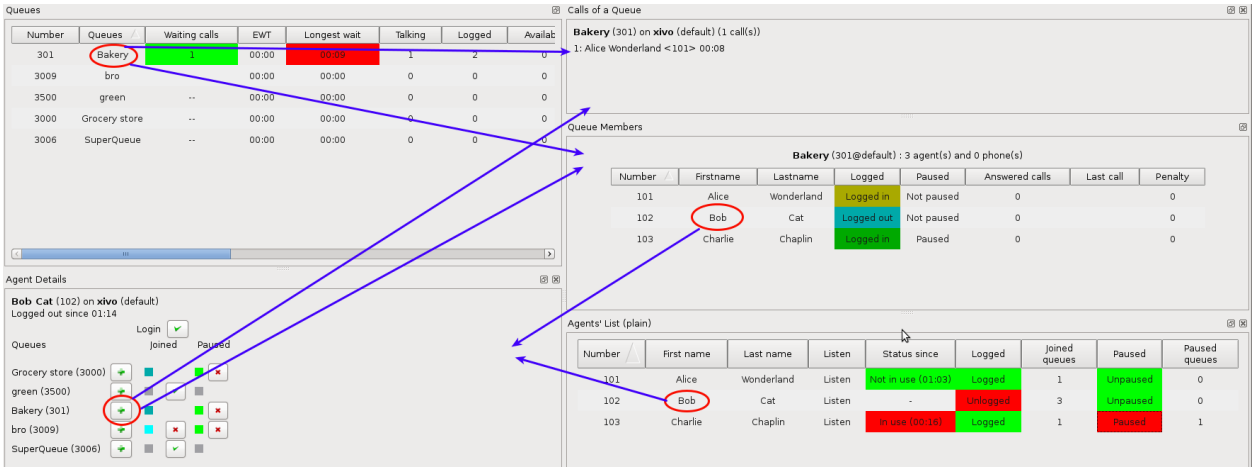
A supervisor profile defined in *Service* → *CTI Server* → *Profiles* menu usually contains the following Xlets :

- Identity
- Queues
- Queue members

- Queues (entries detail)
- Agents (list)
- Agents (detail)

  Note You may also see the *Agent Status Dashboard*

Supervision Panel

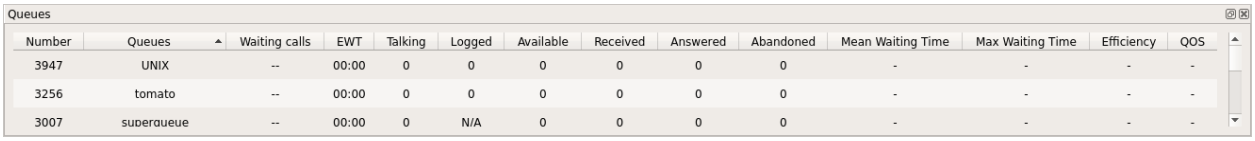


- Clicking on a queue's name in the queue list will display the agent list in the xlet *Queue Members* and show waiting calls in the *Calls of a Queue* xlet.
- Clicking on an agent's name in the agent list will display information on the agent in the *Agent Details* xlet
- Clicking on the + icon in the *Agent Details* xlet will display information about the selected queue in the *Calls of a Queue* and *Queue Members* xlets.

Queue List

General information

The queue list is a dashboard displaying queue statistics and real-time counters for each queue configured on the Wazo.



Real-time Columns

The data of following columns display real-time information.

Queues queue name and number if configured to be displayed

Waiting calls The number of calls currently waiting for an agent in this queue. The background color can change depending of the configured thresholds

EWT Estimated waiting time

Longest wait The longest waiting time for currently waiting calls. The background color can change depending of the configured thresholds

Talking The number of agents currently in conversation in the queue. This column is set to 0 when the queue has just been created and no members have been added.

Logged The number of logged agents in the queue. This column is set to "N/A" when the queue has just been created and no members have been added.

Available The number of available agents ready to take a call in the queue. This column is set to N/A when the queue has just been created and no members have been added.

Last Period Columns

The data of following columns are based on statistics fetched from a fixed-width window of time, e.g. the last 60 minutes or the last 10 minutes. See below to configure the width of the window for each queue.

Received The number of calls received in this queue during the configured statistical window

Answered The number of calls answered in this queue during the configured statistical window

Abandoned The number of calls abandoned in this queue during the configured statistical window

Mean waiting time The mean wait time in the statistical time window, in mm:ss If no calls are received, "-" is displayed

Max waiting time The longest wait time in the statistical time window, in mm:ss If no calls are received, "-" is displayed

Efficiency Answered calls over received calls during the configured statistical window (unanswered calls that are still waiting are not taken into account). If no calls are received, "-" is displayed

QOS Percentage of calls taken within X seconds over answered calls during the configured statistical window. If no calls are received, "-" is displayed

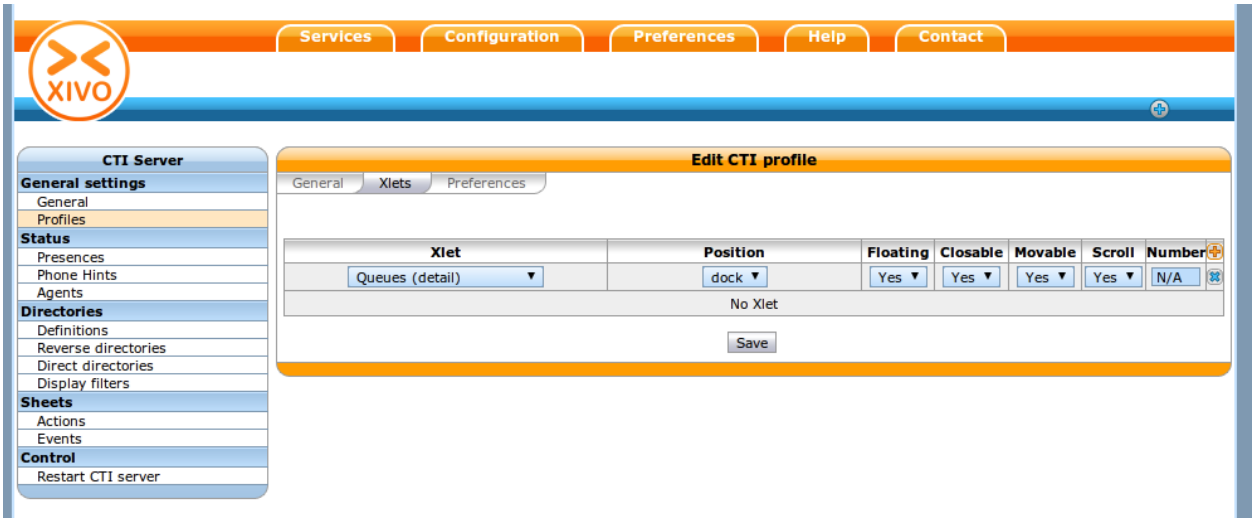
Counter availability

When the Wazo Client is started, "na" is diplayed for counters that have not been initialized.

When the Wazo Client is restarted, the counters are always displayed and calculated as if the application was not restarted. When the server is restarted, counters are reinitialized.

Enabling the xlet

The xlet can be added to any CTI profile from the web interface.



Configuration

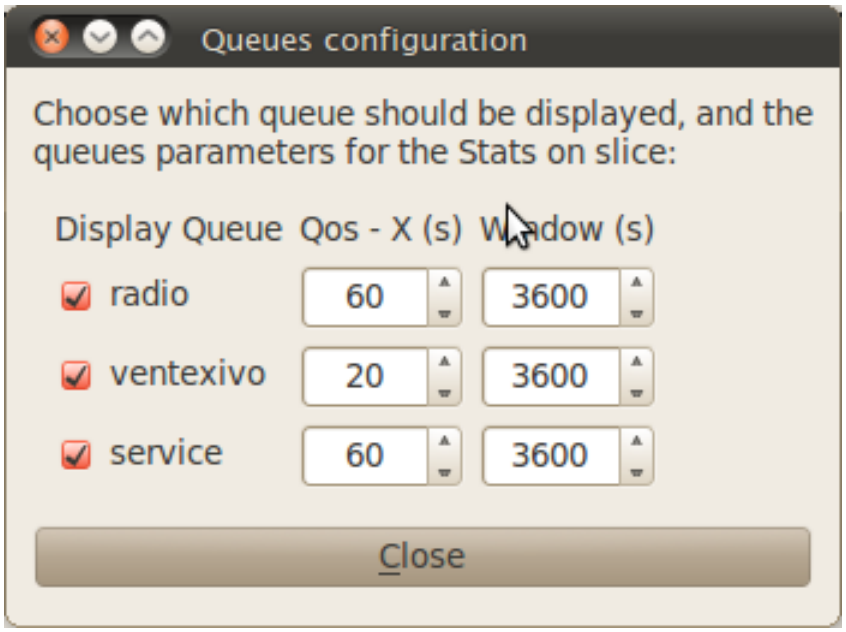
Some values can be configured for the xlet. The statistic fetch timer can be set in the CTI profile preferences. This option is expressed in seconds and the default is 30 seconds.



The statistical period can be configured through the Wazo Client once logged in by right-clicking on the Queue's name in the *Queues* xlet. For each queue, you can configure the following information:

- Qos: maximum wait time for a call, in seconds.
- Window: period of time used for accumulating statistics, in seconds.

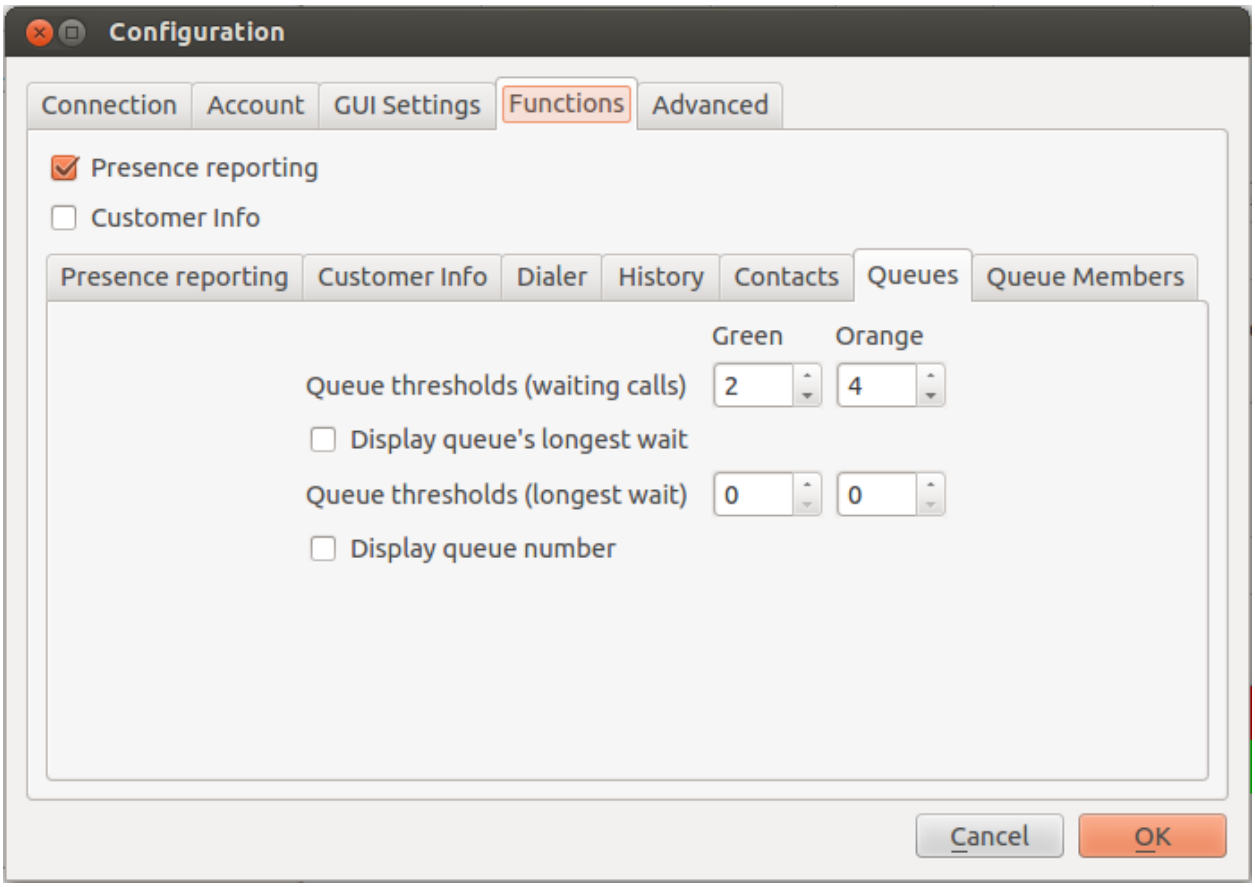
The data used to compute statistics on the Wazo server is only kept for a maximum of 3 hours. The window period cannot be configured to go beyond this limit.



Display options can also be set on the client side. A threshold can be configured to change the color of a column using the following parameters:

- Queue thresholds (waiting calls): number of waiting calls in the queue.
- Display queue's longest wait: Add a column displaying the number of seconds the longest call has waited.

- Queue thresholds (longest wait): number of seconds for the longest waiting call in the queue.
- Display queue number: Add a column displaying the queue's number.



Monitoring queues on high dimension screens

You may want to display the queue list on one big screen, visible by multiple people. However, the default font will not be large enough, so the information will not be readable.

You can change the font size of this Xlet by giving a configuration file when launching the Wazo Client:

```
> wazoclient -stylesheet big_fonts.qss   # Windows and Mac
$ wazoclient -- -stylesheet big_fonts.qss   # GNU/Linux
```

The `big_fonts.qss` file should contain:

```
QueuesView {font-size: 40px;}
QueuesView QHeaderView {font-size: 40px;}
```

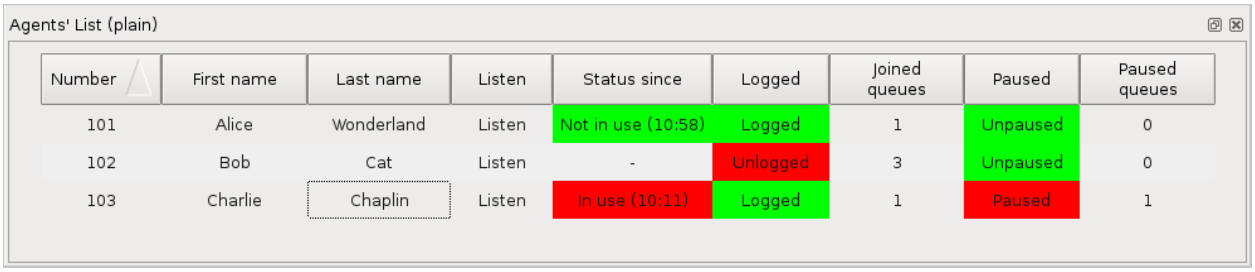
Units of size that can be used are described on the Qt documentation.

Agent List

General information

The queue list is a dashboard displaying each agent configured on the Wazo.

Columns



Number The agent's number

First name & Last name The agent's first name and last name

Listen A *clickable cell* to listen to the agent's current call.

Clicking on the cell will make your phone ring. When you'll answer, you'll hear the conversation the agent is having.

You'll then be able to press the following digits on your phone to switch between the different "listen" modes:

- 4 - spy mode (default). No one hears you.
- 5 - whisper mode. Only the agent hears you.
- 6 - barge mode. Both the agent and the person he's talking to hear you.

Status since Shows the agent's status and the time spent in this status. An agent can have three statuses:

- *Not in use* when he is ready to answer an ACD call
- *Out of queue* when he called or answered a call not from the queue
- *In use* when he is either on call from a queue, on pause or on wrapup

Logged A *clickable cell* to log or unlog the agent

Joined queues The number of queues the agent will be receiving calls from

Paused A *clickable cell* to pause or unpause the agent

Paused queues The number of queues in which the agent is paused

Agent Details

General information

Display advanced informations of an agent and enable to login/logoff, add/remove to a queue, and pause/unpause.

1. This is the status information of agent
2. Button to login/logoff agent
3. Supervision button of the Xlet "Calls of a queue"
4. Add/Remove agent for given queue
5. Pause/Unpause button for given queue

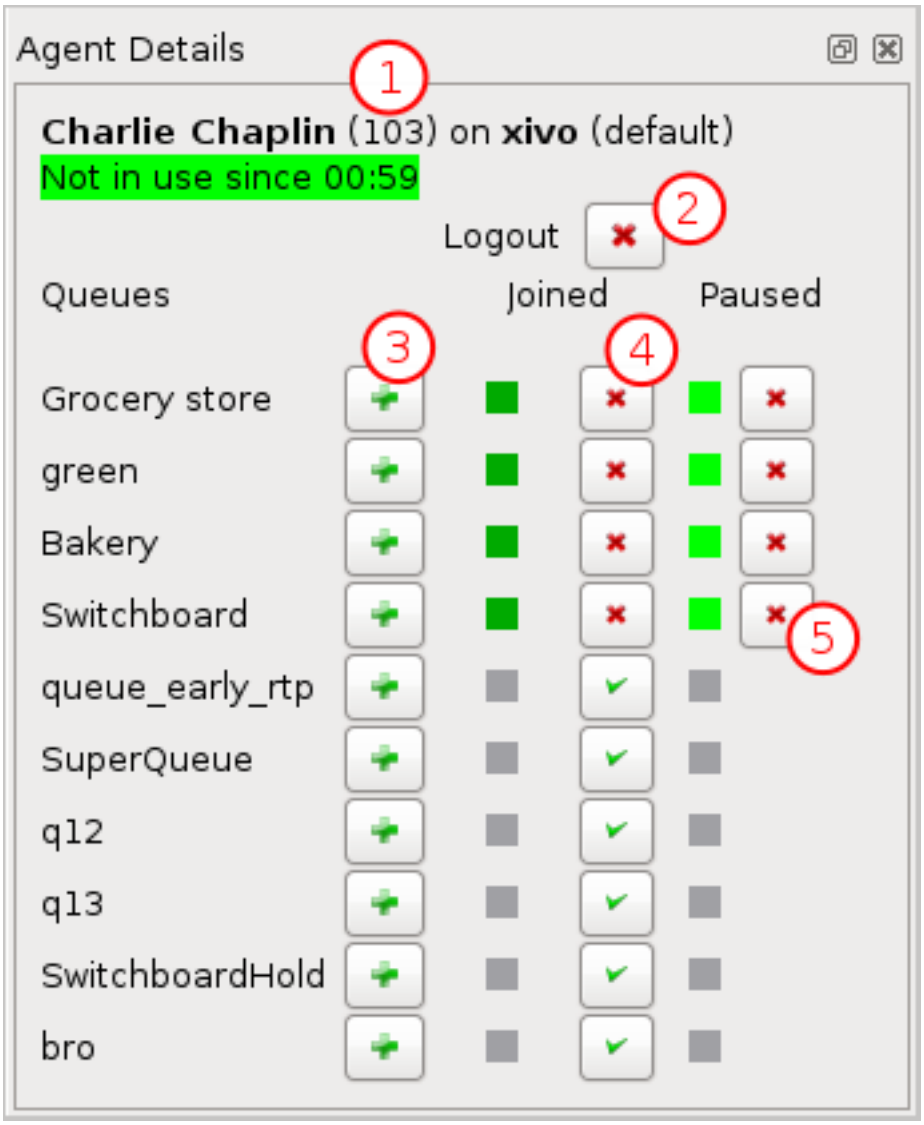
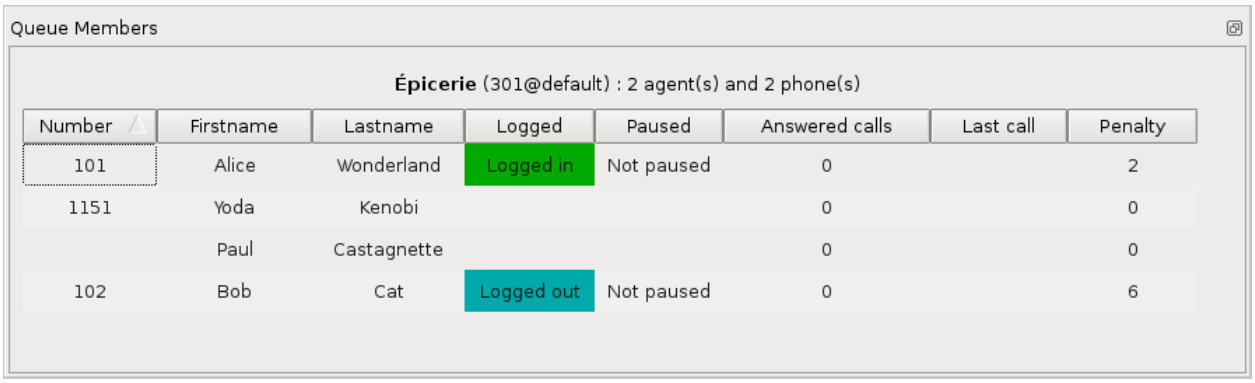


Fig. 1.90: Agent Details



Queue members

The queue members lists which agents or phones will receive calls from the selected queue and some of their attributes.

Columns

Number The agent number or the phone number of the queue member.

Firstname and Lastname First name and last name of the agent or the user to which the phone belongs.

Logged Whether the agent is logged or not. Blank for a phone.

Paused Whether the agent is paused or not. Blank for a phone.

Answered calls Number of calls answered by the member since last login (for an agent), or restart or configuration reload.

Last call Hangup time of the last answered calls.

Penalty Penalty of the queue member.

Link Wazo Client presence to agent presence

You can configure Wazo to have the following scenario:

- The agent person leaves temporarily his office (lunch, break, . . .)
- He sets his presence in the Wazo Client to the according state
- The agent will be automatically set in pause and his phone will not ring from queues
- He comes back to his office and set his presence to 'Available'
- The pause will be automatically cancelled

You can *configure the presence states* of CTI profiles and attach `Actions` to them, such as *Set in pause* or *Enable DND*.

You can then attach an action *Set in pause* for multiple presence states and attach an action *Cancel the pause* for the presence state *Available*.

For now, the actions attached to the mandatory presence *Disconnected* will not be taken into account.

1.10.4 Agent Status Dashboard

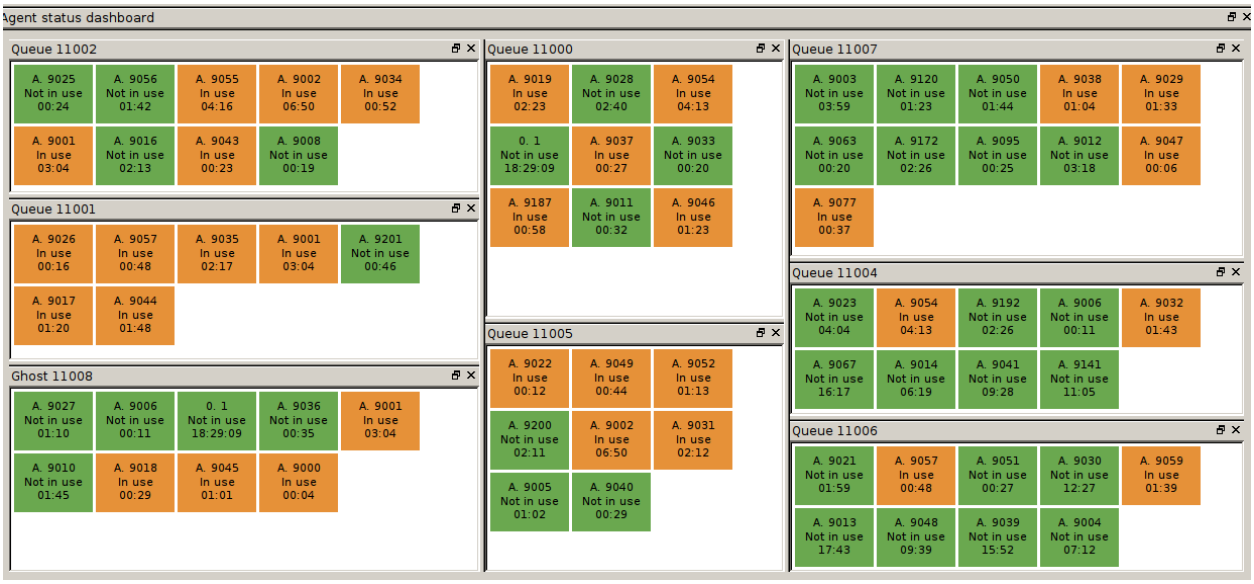
Overview

The goal of the agent status dashboard xlet is to give contact center supervisors a better overview of agent status evolution in active queues.

Usage

The xlet is *read-only* and presents a list of queues. For each queue, the xlet displays a status box for each logged in agent. Each status box gives the following information:

- Agent name
- Agent status: Shows the agent's status. An agent can have six statuses:
  - *Not in use* when he is ready to answer an ACD call



  - *Int. Incoming* when he answered an internal call not from a queue
  - *Int. Outgoing* when he emitted an internal call not from a queue
  - *Ext. Incoming* when he answered an external call not from a queue
  - *Ext. Outgoing* when he emitted an external call not from a queue
  - *In use* when he is either on call a from a queue, on pause or on wrapup
- Agent status since: Shows the time spent in the current status
- Background color:
  - green if *Not in use*
  - purple if *Int. Incoming* or *Int. Outgoing*
  - pink if *Ext. Incoming* or *Ext. Outgoing*
  - orange if *In use*

Note that the agent status will only change when the communication is established, not when phones are ringing.

Known bugs

1. If an agent emits a call via his Wazo Client, the status will change to *Int. Outgoing* or *Ext. Outgoing* when the destination phone rings, instead of when the destination answers.
2. - Given the agent is on an ACD call
   - When the agent logs out
   - When the agent hangs up the ACD call
   - When the agent logs back in via CTI Client
   - Then the agent may be seen as outgoing non-ACD communication, whether there is a non-ACD communication or not

   To make the agent *Not in use* again, make a non-ACD call and hangup.

3. - Given the agent is on ACD call
   - When the agent calls someone else (e.g. his supervisor)
   - When the ACD call hangs up (while the agent talks to his supervisor)
   - Then the agent is seen as available, instead of in outgoing non-ACD communication.

   This applies to all kinds of non-ACD calls.

Changing the disposition

The disposition of the Xlet can be changed in two ways:

- Placement of queues
- Which queues are displayed

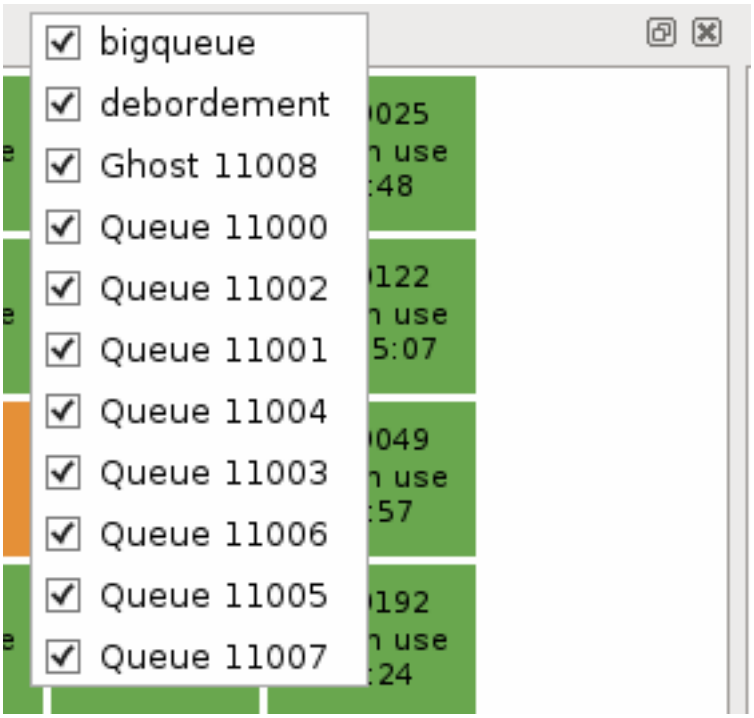
The disposition is saved whenever the Wazo Client is closed and restored when it is opened again.

Changing the placement of queues

The little windows containing each queue can be resized and moved around. That way, any layout can be achieved, according to the size and importance of each queue.

Choosing which queues are displayed

There is a little contextual menu when right-clicking on the title bar of every queue window. Checking/unchecking the lines of this menu shows/hides the associated queue.



Known issues

- There is no profile containing this xlet. The profile must be created manually.
- There is no sorting on agents in a queue.
- An empty queue will display an empty box with no message specifying the queue has no logged agents.

Configuration

No special configuration is necessary other than creating a CTI profile in which the Agent Status Dashboard is added.

1.10.5 Skills-Based Routing

Introduction

> *Skills-based routing (SBR), or Skills-based call routing, is a call-assignment strategy used in call centres to assign incoming calls to the most suitable agent, instead of simply choosing the next available agent. It is an enhancement to the Automatic Call Distributor (ACD) systems found in most call centres. The need for skills-based routing has arisen, as call centres have become larger and dealt with a wider variety of call types.*
>
> —Wikipedia

In this respect, skills-based routing is also based on call distribution to agents through waiting queues, but one or many skills can be assigned to each agent, and call can be distributed to the most suitable agent.

In skills-based routing, you will have to find a way to be able to tag the call for a specific skill need. This can be done for example by entering the call distribution system using different incoming call numbers, using an IVR to let the caller do his own choice, or by requesting to the information system database the customer profile.

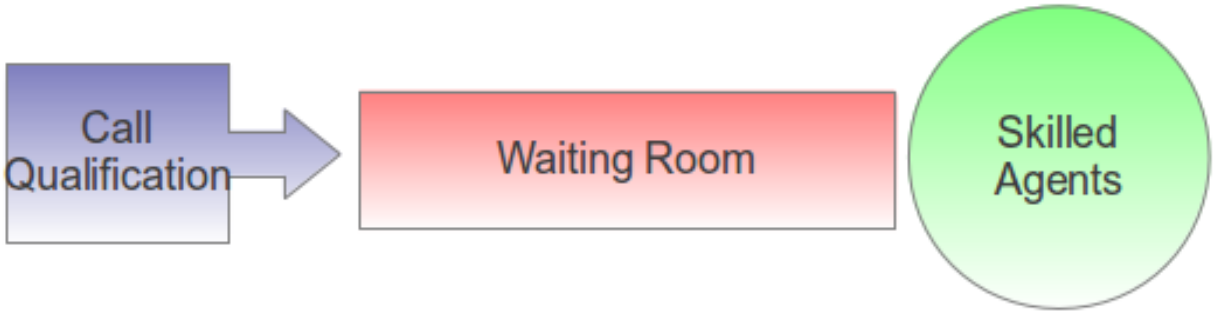


Fig. 1.91: Skills-Based Routing

Getting Started

- Create the skills
- Apply the skills to the agents
- Create the skill rule sets
- Assign the skill rule sets using a configuration file
- Apply the skill rule sets to call qualification, i.e. incoming calls by using the preprocess subroutine field

Note that you shouldn't use skill based routing on a queue with queue members of type user because the behaviour is not defined and might change in a future Wazo version.

Skills

Skills are created using the menu *Services* → *Call center* → *Skills*. Each skill belongs to a category. First create the category, and in this category create different skills.

Note that a skill name can't contain upper case letters and must be globally unique (i.e. the same name can't be used in two different categories).

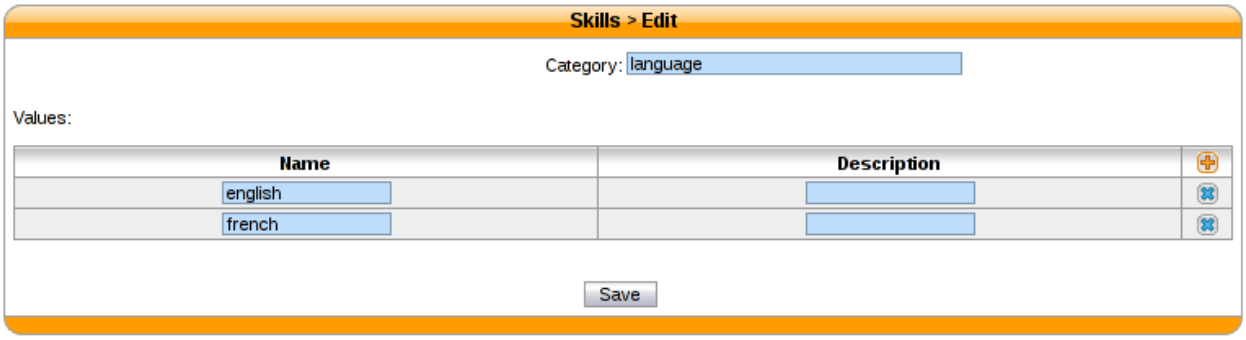


Fig. 1.92: Skills Creation

Once all the skills are created you may apply them to agents. Agents may have one or more skills from different categories.

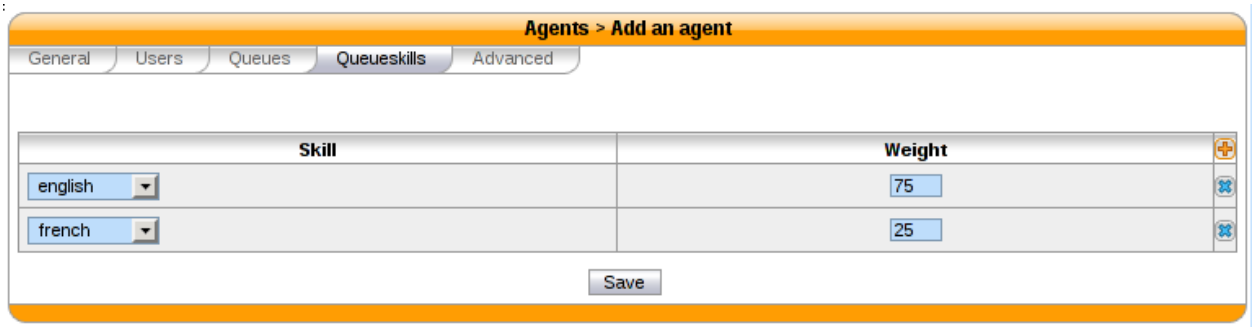


Fig. 1.93: Apply Skills to Agents

It is typical to use a value between 0 and 100 inclusively as the weight of a skill, although any integer is accepted.

Skill Rule Sets

Once skills are created, rule sets can be defined.

A rule set is a list of rules. Here's an example of a rule set containing 2 rules:

1. WT < 60, english > 50
2. english > 0

The first rule of this rule set can be read as:

> If the caller has been waiting for less than 60 seconds (WT < 60), only try to call agents which have the skill "english" set to a value higher than 50; otherwise, go to the next rule.

And the second rule can be read as:

> Only try to call agents which have the skill "english" set to a value higher than 0.

Let's examine some simple scenarios, because there's actually some subtilities on how calls are distributed. We will suppose that we have a queue with the default settings and the following members:

- Agent A, with skill english set to 75
- Agent B, with skill english set to 25

Scenario 1

Given:

- Agent A is logged and not in use
- Agent B is logged and not in use
- There is no call in the queue

When a new call enters the queue, then it is distributed to Agent A. As long as Agent A is available and doesn't answer the call, the call will never be distributed to Agent B, even after 60 seconds of waiting time.

When another call enters the queue, then after 60 seconds of waiting time, this call will be distributed to Agent B (and the first call will still be distributed only to Agent A).

The reason is that there's a difference between a call that is being distributed (i.e. that is making agents ring) and a call that is waiting for being distributed. When a call is being distributed to a set of members, no other rule is tried as long as there's at least 1 of these members available.

Scenario 2

Given:

- Agent A is not logged
- Agent B is logged and not in use
- There is no call in the queue

When a new call enters the queue, then it is *immediately* distributed to Agent B.

The reason is that when there's no logged agent matching a rule, the next rule is immediately tried.

Rules

Each rule set is composed of rules, and each rule has two parts, separated by a comma:

- the first part (optional) is the *"dynamic part"*
- the second part is the *"skill part"*

Each part contains an expression composed of operators, variables and integer constants.

Operators

The following operators can be used inside rules:

Comparison operators:

- operand1 ! operand2 (is not equal)
- operand1 = operand2 (is equal)
- operand1 > operand2 (is greater than)
- operand1 < operand2 (is lesser than)

Logical operators:

- operand1 & operand2 (both are true)
- operand1 | operand2 (at least one of them are true)

'!' is the operator with the higher priority, and '|' the one with the lower priority. You can use parentheses '()' to change the priority of operations.

Dynamic Part

The dynamic part can reference the following variables:

- WT
- EWT

The waiting time (WT) is the elapsed time since the call entered the queue. The time the call pass in an IVR or another queue is not taken into account.

The estimated waiting time (EWT) has never fully worked. It is mentioned here only for historical reason. You should not use it. It might be removed in a future Wazo version.

Examples

- WT < 60

Skill Part

The skill part can reference any skills name as variables.

You can also use meta-variables, starting with a '$', to substitute them with data set on the Queue() call. For example, if you call Queue() with the skill rule set argument equal to:

```
select_lang(lang=german)
```

Then every `$lang` occurrence will be replaced by 'german'.

Examples

- english > 50
- technic ! 0 & ($os > 29 & $lang > 39 | $os > 39 & $lang > 19)



Fig. 1.94: Create Skill Rule Sets

Evaluation

Note that the expression:

> english | french

is equivalent to:

> english ! 0 | french ! 0

Sometimes, a rule references a skill which is not defined for every agent. For example, given the following rule:

> english > 0 | english < 1

Then, for an agent which has the skill english defined, the result of this expression is always true. For an agent which does not have the skill english defined, the result of this expression is always false.

Said differently, an agent without a skill X is not the same as an agent with the skill X set to the value 0.

Technically, this is what is happening when evaluating the rule "english > 0" for an agent without the skill english:

```
  english > 0
=     <Substituing english with the agent value>
  "undefined" > 0
=     <A comparison with "undefined" in at least one operand yields undefined>
  "undefined"
=     <In a boolean context, "undefined" is equal to false>
  false
```

This behaviour applies to every comparison operators.

Also, the syntax that is currently accepted for comparison is always of the form:

```
variable cmp_op constant
```

Where "variable" is a variable name, "cmp_op" is a comparison operator and "constant" is an integer constant. This means the following expressions are not accepted:

- 10 < english (but english > 10 is accepted)
- english < french (the second operand must be a constant)
- 10 < 11 (the first operand must be a variable name)

Apply Skill Rule Sets

A skill rule set is attached to a call using a bit of dialplan. This dialplan is stored in a configuration file you may edit using menu *Services* → *IPBX* → *Configuration Files*.

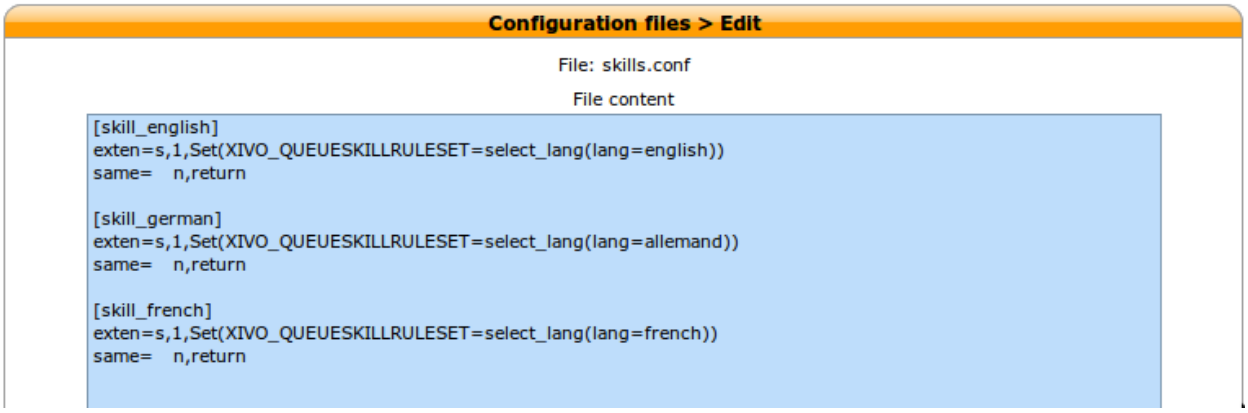


Fig. 1.95: Use Rule Set In Dialplan

In the figure above, 3 different languages are selected using three different subroutines.

Each of this different selections of subroutines can be applied to the call qualifying object. In the following example language selection is applied to incoming calls.

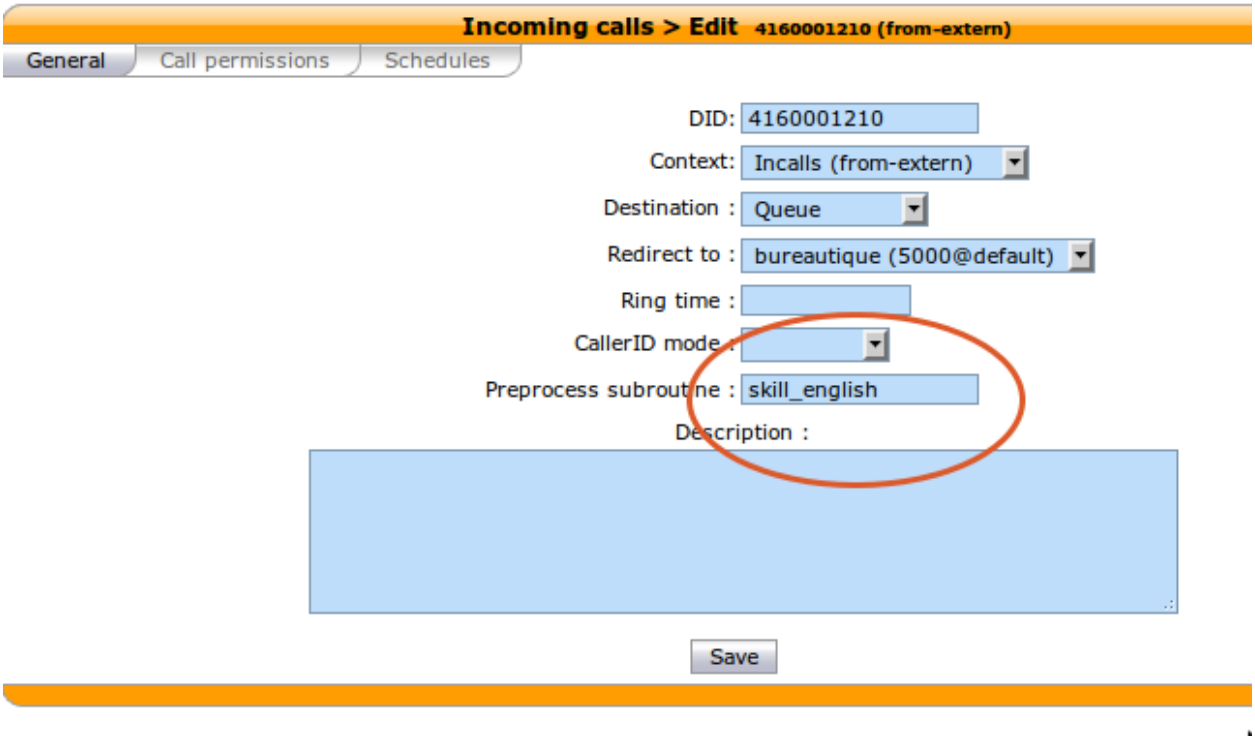


Fig. 1.96: Apply Rule Set to Incoming Call

Example

Configuration file for simple skill selection :

```
[simple_skill_english]
exten = s,1,Set(XIVO_QUEUESKILLRULESET=english_rule_set)
same  =   n,Return()

[simple_skill_french]
exten = s,1,Set(XIVO_QUEUESKILLRULESET=french_rule_set)
same  =   n,Return()
```

Monitoring

You may monitor your waiting calls with skills using the asterisk CLI and the command `queue show <queue_name>`:

```
wazo*CLI> queue show services
services has 1 calls (max unlimited) in 'ringall' strategy (0s holdtime, 2s talktime),
↪ W:0, C:1, A:10, SL:0.0% within 0s
  Members:
     Agent/2000 (Not in use) (skills: agent-1) has taken no calls yet
     Agent/2001 (Unavailable) (skills: agent-4) has taken no calls yet
  Virtual queue english:
  Virtual queue french:
     1. SIP/jyl-dev-assur-00000017 (wait: 0:05, prio: 0)
  Callers:
```

You may monitor your skills groups with the command `queue show skills groups <agent_name>`:

```
wazo*CLI> queue show skills groups <PRESS TAB>
agent-2   agent-3   agent-4   agent-48  agent-7   agent-1
wazo*CLI> queue show skills groups agent-1
Skill group 'agent-1':
  - bank           : 50
  - english        : 100
```

You may monitor your skills rules with the command `queue show skills rules <rule_name>`:

```
wazo*CLI> queue show skills rules <PRESS TAB>
english      french       select_lang
wazo*CLI> queue show skills rules english
Skill rules 'english':
  => english>90
```

1.10.6 Statistics

Overview

The statistics page is used to monitor the efficiency of queues and agents. Statistics are automatically generated every six hours. They can also be generated manually.

Note: The contact center statistics do not apply to switchboard queues. See *Switchboard* for more details.

Note: The oldest statistics are periodically removed. See *xivo-purge-db* for more details.

Configuration

In order to display call center statistics, you must create at least one configuration profile.

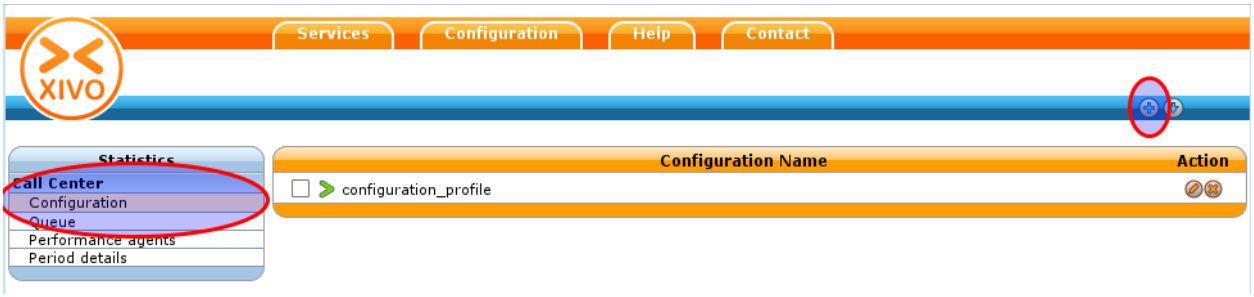


Fig. 1.97: Statistics Configuration

The configuration profile is used to generate reports from the cache. The cache is generated independently from the configuration so adding a new configuration does not require a new cache generation.

Configuration options

Field	Values	Description
name	string	Configuration name, useful for remembering what the configuration is used for
interval	enum [0-999] [day, week, month]	Default time interval used when displaying statistics. Examples: "-1 day": show statistics for yesterday "-3 weeks": show statistics for the last 3 weeks
show on summary page		Display this configuration on the summary page
timezone	America/Montreal	Your time zone
Period cache		Maximum and minimum dates that can be used for displaying statistics
start	YYYY-MM	Start date
end	YYYY-MM	End date. If left to 0, use the servers' current date
Working Hours		Work hours for agents
start	hh:mm	Beginning of working hours.
end	hh:mm	End of working hours
Periods		Number of calls answered for a time period
Period 1	number of seconds (Example: 20)	Show number of calls answered within 20 seconds in column "P1"
Period n	number of seconds (Example: 20)	Show number of calls answered within 20 seconds in column "Pn"

Note: Calls outside of working hours will not be in the cache. e.g. if working hours are from 8:00 AM to 16:00 PM, a call at 7:55 AM will not show up in the reports.

Note: Statistics are computed on the hour. e.g. If work hours are from 8:30 to 16:15, working hours should be set from 8:00 to 17:00.

Note: Period includes both bounds, if the same number is used for the higher bound and the lower bound of next period, some calls will be counted twice. i.e period 1 : 0-30 period 2 : 31-60 period 3 : 61

How to generate the cache

The cache must be generated before using reports. By default, the cache is automatically generated every six hours.

However, you can safely generate it manually. The script to generate the cache is *xivo-stat fill_db*. When this script is run, statistics will be regenerated for the last 8 hours starting from the previous hour. e.g. If you run xivo-stat on 2012-08-04 11:47:00, statistics will be regenerated from 2012-08-04 03:00:00 to 2012-08-04 11:47:00

Note: *xivo-stat fill_db* can be a long operation when used for the first time or after a *xivo-stat clean_db*

Warning: The current events have an end date of the launch date of the script xivo-stat as the end date.

Clearing the cache

If for some reason the cache generation fails or the cache becomes unusable, the administrator can safely clean the cache using *xivo-stat clean_db* and then regenerate it. This operation will only clear the cache and does *not* erase any other data.

Queue statistics

Queue statistics can be viewed in *Services* → *Statistics* → *Queue*.

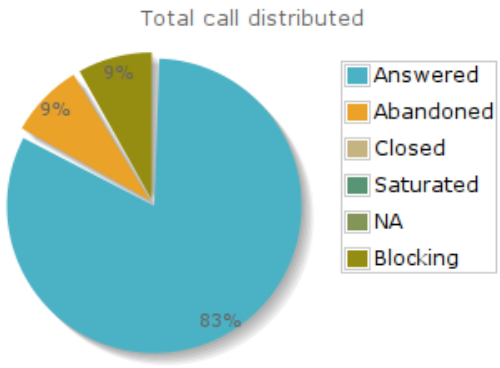
The first table displays a list of queues with all the counters for the period choosen from the Dashboard panel

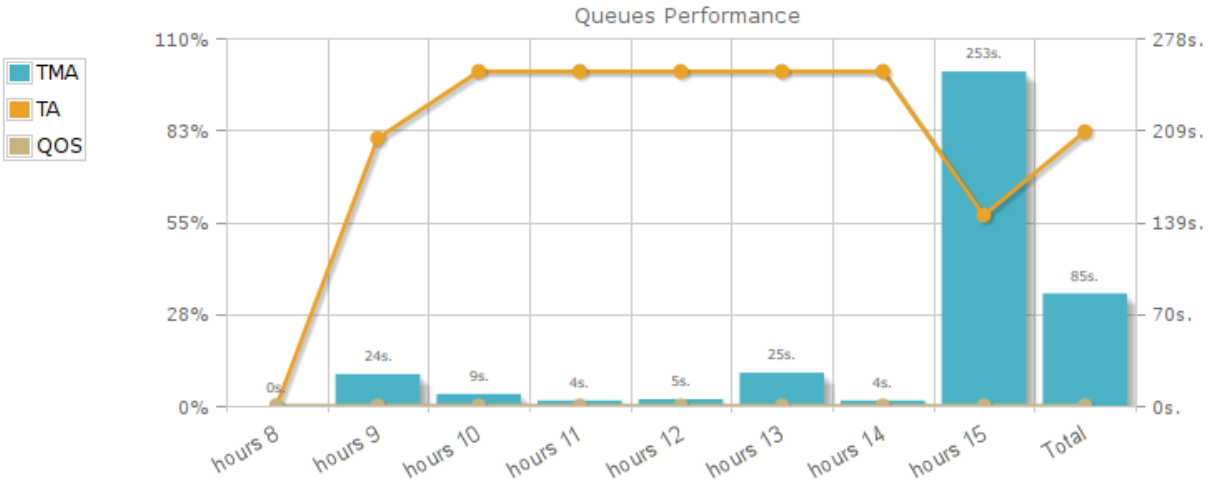
				Dissuaded or Overflowed						
	Received	Answered	Abandoned	Closed	Saturated	NA	Blocking	AWT	Answered rate	QoS
STA1	1749	1521	84	21	0	0	123	00:00:13	88 %	0 %
STA2	1713	1454	73	38	0	0	148	00:00:11	86 %	0 %
STA3	1529	1367	76	23	0	0	63	00:00:10	90 %	0 %
STA4	2147	1776	115	17	0	0	239	00:00:16	83 %	0 %
STA5	1800	1594	93	28	0	0	85	00:00:13	89 %	0 %

By clicking on a queue name you may display detailed queue statistics

Statistics can be displayed :

				Dissuaded or Overflowed						
	Received	Answered	Abandoned	Closed	Saturated	NA	Blocking	AWT	Answered rate	QoS
8h-9h	0	0	0	0	0	0	0			
9h-10h	5	4	1	0	0	0	0	00:00:24	80 %	0 %
10h-11h	2	2	0	0	0	0	0	00:00:09	100 %	0 %
11h-12h	1	1	0	0	0	0	0	00:00:04	100 %	0 %
12h-13h	3	3	0	0	0	0	0	00:00:05	100 %	0 %
13h-14h	1	1	0	0	0	0	0	00:00:25	100 %	0 %
14h-15h	4	4	0	0	0	0	0	00:00:04	100 %	0 %
15h-16h	7	4	1	0	0	0	2	00:04:13	57 %	0 %
Total	23	19	2	0	0	0	2	00:01:25	82 %	0 %





By week

				Dissuaded or Overflowed						
	Received	Answered	Abandoned	Closed	Saturated	NA	Blocking	AWT	Answered rate	QoS
Monday 7	28	27	0	0	0	0	1	00:00:10	96 %	0 %
Tuesday 8	22	20	1	0	0	0	1	00:00:04	90 %	0 %
Wednesday 9	23	19	2	0	0	0	2	00:01:25	82 %	0 %
Thursday 10	21	21	0	0	0	0	0	00:00:07	100 %	0 %
Friday 11	34	30	1	0	0	0	3	00:00:08	88 %	0 %
Total	128	117	4	0	0	0	7	00:00:21	91 %	0 %



By month

By year

Counters

- Received: Total number of received calls
- Answered: Calls answered by an agent
- Abandoned: Calls that were hung up while waiting for an answer
- Dissuaded or Overflowed:
  - Closed: Calls received when the queue was closed
  - No answer (NA): Calls that reached the ring timeout delay
  - Satured: Calls received when the queue was already full ("Maximum number of people allowed to wait:" limit of advanced tab) or when one of the diversion parameters were reached
  - Blocking : Calls received when no agents were available or when there were no agents to take the call, join an empty queue condition or remove callers if there are no agents condition is reached (advanced queue parameter tab).
- Average waiting time (AWT): The average waiting time of call on wait
- Answered rate (HR): The ratio of answered calls over (received calls - closed calls)
- Quality of service (QoS): Percentage of calls answered in less than x seconds over the number of answered calls, where x is defined in the configuration

Agent performance

Agent performance statistics can be viewed in *Services* → *Statistics* → *Performance agents*.

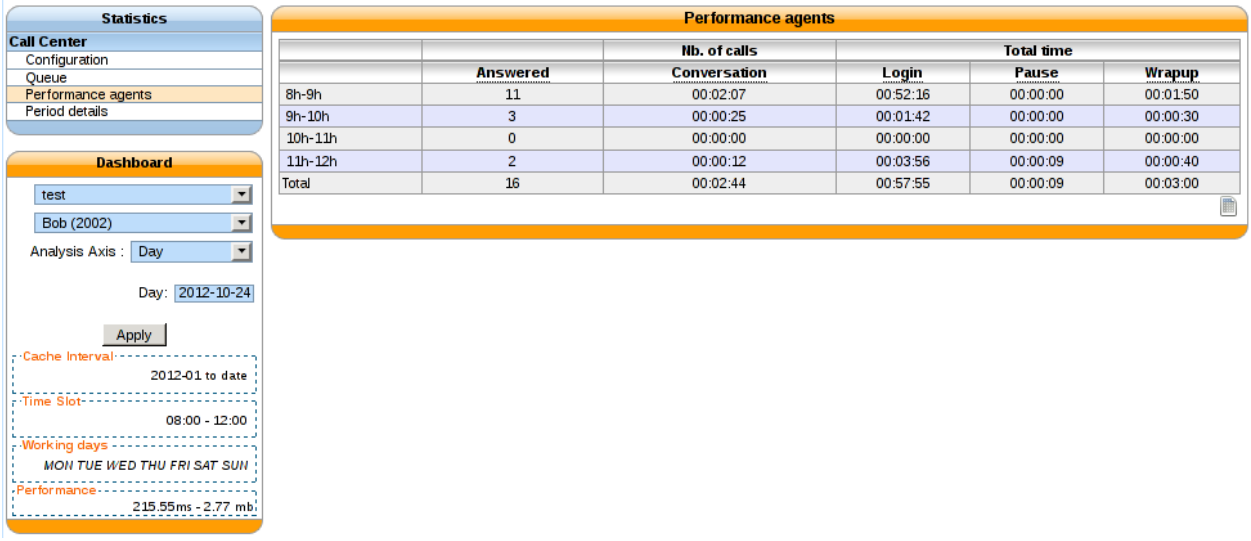
Note: The agent performance counters do not take into account transfer between agents: if agent A processes a call and transfers it to agent B, only the counters of agent A will be updated. Ignoring any info after the call transfer.

				Dissuaded or Overflowed						
	Received	Answered	Abandoned	Closed	Saturated	NA	Blocking	AWT	Answered rate	QoS
1 week										
Tuesday	0	0	0	0	0	0	0			
Wednesday	10	0	0	0	0	0	10	00:00:00	0 %	
Thursday	13	12	1	0	0	0	0	00:00:23	92 %	0 %
Friday	19	17	2	0	0	0	0	00:00:25	89 %	0 %
2 week										
Monday	28	27	0	0	0	0	1	00:00:10	96 %	0 %
Tuesday	22	20	1	0	0	0	1	00:00:04	90 %	0 %
Wednesday	23	19	2	0	0	0	2	00:01:25	82 %	0 %
Thursday	21	21	0	0	0	0	0	00:00:07	100 %	0 %
Friday	34	30	1	0	0	0	3	00:00:08	88 %	0 %
3 week										
Monday	36	35	0	0	0	0	1	00:00:11	97 %	0 %
Tuesday	40	36	4	0	0	0	0	00:00:07	90 %	0 %
Wednesday	35	35	0	0	0	0	0	00:00:07	100 %	0 %
Thursday	51	51	0	0	0	0	0	00:00:05	100 %	0 %
Friday	16	16	0	0	0	0	0	00:00:04	100 %	0 %
4 week										
Monday	24	24	0	0	0	0	0	00:00:04	100 %	0 %
Tuesday	25	24	1	0	0	0	0	00:00:08	96 %	0 %
Wednesday	28	24	4	0	0	0	0	00:00:26	85 %	0 %
Thursday	40	37	0	3	0	0	0	00:00:09	100 %	0 %
Friday	39	37	2	0	0	0	0	00:00:06	94 %	0 %
5 week										
Monday	43	43	0	0	0	0	0	00:00:07	100 %	0 %
Tuesday	35	32	1	0	0	0	2	00:00:06	91 %	0 %
Wednesday	31	28	1	0	0	0	2	00:00:07	90 %	0 %
Thursday	27	15	1	0	0	0	11	00:00:49	55 %	0 %
Total	640	583	21	3	0	0	33	00:00:13	91 %	0 %



				Dissuaded or Overflowed						
	Received	Answered	Abandoned	Closed	Saturated	NA	Blocking	AWT	Answered rate	QoS
January	0	0	0	0	0	0	0			
February	0	0	0	0	0	0	0			
March	0	0	0	0	0	0	0			
April	0	0	0	0	0	0	0			
May	0	0	0	0	0	0	0			
June	0	0	0	0	0	0	0			
July	119	98	7	13	0	0	1	00:00:08	92 %	1 %
August	95	72	6	3	0	0	14	00:00:10	78 %	0 %
September	181	153	9	0	0	0	19	00:00:11	84 %	0 %
October	214	159	9	0	0	0	46	00:00:13	74 %	0 %
November	177	169	6	1	0	0	1	00:00:06	96 %	0 %
December	194	180	5	1	0	0	8	00:00:07	93 %	0 %
Total	973	824	42	18	0	0	89	00:00:09	86 %	0 %





		Nb. of calls	Total time		
	Answered	Conversation	Login	Pause	Wrapup
8h-9h	11	00:02:07	00:52:16	00:00:00	00:01:50
9h-10h	3	00:00:25	00:01:42	00:00:00	00:00:30
10h-11h	0	00:00:00	00:00:00	00:00:00	00:00:00
11h-12h	2	00:00:12	00:03:56	00:00:09	00:00:40
Total	16	00:02:44	00:57:55	00:00:09	00:03:00

Counters

- Answered: Number of calls answered by the agent
- Conversation: Total time spent for calls answered during a given period
- Login: Total login time of an agent.
- Wrapup: Total time spent in wrapup by an agent.
- Pause: Total pause time of an agent

Note: The Pause time counter only supports **PAUSEALL** and **UNPAUSEALL** command from cticlient. The agent must also be a member of a least 1 queue.

Agent summary

Performance agents

	Nb. of calls	Total time			
	Answered	Conversation	Login	Pause	Wrapup
Karine Boudoix (108)	87	09:59:57	30:58:51	10:53:16	00:21:30
Fred Epric (100)	45	01:27:52	34:59:45	05:32:45	00:07:30
Hipolyte Marroussou (102)	13	00:24:23	27:42:47	97:50:42	00:02:10
Gérard Mensour (101)	0	00:00:00	00:00:00	00:00:00	00:00:00
Irène Pourtox (106)	39	03:52:40	34:08:47	22:39:31	00:09:45
Juliette Queriau (107)	78	09:03:51	34:06:24	91:23:52	00:19:30

Agent per day

	Nb. of calls	Total time			
	Answered	Conversation	Login	Pause	Wrapup
9h-10h	1	00:07:31	00:28:45	00:55:13	00:00:00
10h-11h	5	00:30:50	01:00:00	00:00:00	00:01:15
11h-12h	4	00:24:15	01:00:00	00:00:20	00:01:15
12h-13h	0	00:00:00	00:33:34	00:59:07	00:00:00
13h-14h	0	00:00:00	01:00:00	00:34:20	00:00:00
14h-15h	3	01:23:38	01:00:00	00:00:00	00:00:30
15h-16h	2	00:05:24	01:00:00	00:00:00	00:00:30
16h-17h	3	00:26:21	01:00:00	00:00:00	00:01:00
17h-18h	6	00:37:50	01:00:00	00:00:00	00:01:15
Total	24	03:35:49	08:02:20	02:29:01	00:05:45

	Nb. of calls	Total time			
	Answered	Conversation	Login	Pause	Wrapup
Monday 1	24	03:35:49	08:02:20	02:29:01	00:05:45
Tuesday 2	22	02:17:11	07:31:53	03:36:46	00:05:30
Wednesday 3	19	01:40:33	07:27:13	02:34:03	00:04:45
Thursday 4	22	02:26:24	07:57:25	02:13:23	00:05:30
Friday 5	0	00:00:00	00:00:00	00:00:00	00:00:00
Total	87	09:59:57	30:58:51	10:53:16	00:21:30

	Nb. of calls	Total time			
	Answered	Conversation	Login	Pause	Wrapup
23 week					
Monday	0	00:00:00	09:00:00	00:00:00	00:00:00
Tuesday	0	00:00:00	09:00:00	00:00:00	00:00:00
Wednesday	0	00:00:00	09:00:00	00:00:00	00:00:00
Thursday	0	00:00:00	09:00:00	00:00:00	00:00:00
Friday	0	00:00:00	09:00:00	00:00:00	00:00:00
24 week					
Monday	0	00:00:00	09:00:00	00:00:00	00:00:00
Tuesday	0	00:00:00	03:54:46	00:00:00	00:00:00
Wednesday	0	00:00:00	00:00:00	00:00:00	00:00:00
Thursday	0	00:00:00	04:35:05	04:56:29	00:00:00
Friday	10	01:23:58	08:24:59	03:11:31	00:01:35
25 week					
Monday	15	01:20:19	08:17:50	03:02:58	00:02:30
Tuesday	3	00:13:52	08:22:51	07:06:44	00:00:20
Wednesday	3	00:20:02	08:28:03	06:12:39	00:00:40
Thursday	0	00:00:00	08:24:06	08:24:01	00:00:00
Friday	1	00:09:22	08:27:27	05:27:33	00:00:15
26 week					
Monday	10	00:36:41	08:23:33	04:48:20	00:02:30
Tuesday	11	01:08:46	08:30:00	04:34:26	00:02:45
Wednesday	3	00:07:48	08:29:34	04:58:42	00:00:45
Thursday	5	00:40:10	04:00:58	07:26:52	00:01:15
Friday	9	01:12:33	08:19:18	04:27:04	00:02:15
Total	70	07:13:31	150:38:3	64:37:24	00:14:50

Agent per week

Agent per month

Agent per year

	Nb. of calls	Total time			
	Answered	Conversation	Login	Pause	Wrapup
January	0	00:00:00	00:00:00	00:00:00	00:00:00
February	0	00:00:00	00:00:00	00:00:00	00:00:00
March	0	00:00:00	00:00:00	00:00:00	00:00:00
April	0	00:00:00	00:00:00	00:00:00	00:00:00
May	1	00:00:34	97:17:07	00:00:00	00:00:05
June	89	09:01:48	159:03:1	69:17:25	00:19:35
July	39	03:52:40	34:08:47	22:39:31	00:09:45
August	0	00:00:00	00:00:00	00:00:00	00:00:00
September	0	00:00:00	00:00:00	00:00:00	00:00:00
October	0	00:00:00	00:00:00	00:00:00	00:00:00
November	0	00:00:00	00:00:00	00:00:00	00:00:00
December	0	00:00:00	00:00:00	00:00:00	00:00:00
Total	110	11:06:45	282:04:2	87:16:55	00:24:40

Period details

Display by period defined in configuration, i.e. between 0 and 10s, 10s and 30s etc ... the number of handled calls and the number of abandonned calls

Details by time

P1: 0-15s | P2: 16-20s | P3: 21-30s | P4: 31-60s | P5: 61s |

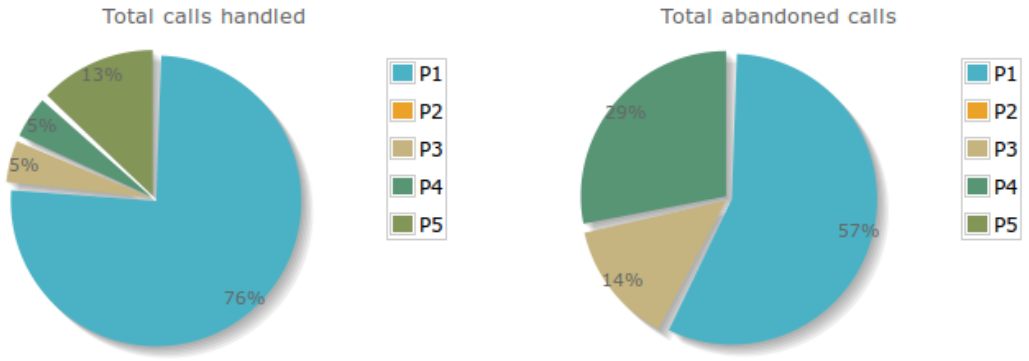
	Calls handled					Abandoned calls				
	P1	P2	P3	P4	P5	P1	P2	P3	P4	P5
blue	1992	25	140	152	250	38	3	6	15	26
green	0	0	0	0	0	0	0	0	0	0
red	0	0	0	0	0	0	0	0	0	0
yellow	640	2	18	7	2	10	1	0	2	1

You may click on a queue name to get more information for this queue

Details by time

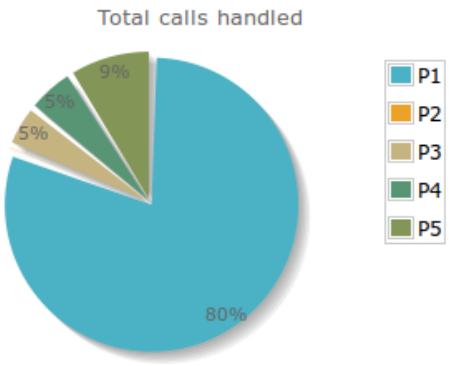
P1: 0-15s | P2: 16-20s | P3: 21-30s | P4: 31-60s | P5: 61s |

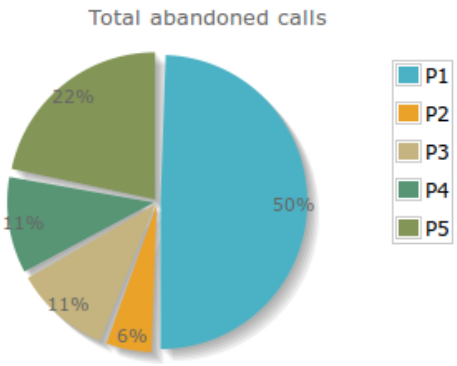
	Calls handled					Abandoned calls				
	P1	P2	P3	P4	P5	P1	P2	P3	P4	P5
8h-9h	0	0	0	0	0	0	0	0	0	0
9h-10h	13	0	1	5	12	0	0	0	0	0
10h-11h	16	0	2	0	5	2	0	0	2	0
11h-12h	20	0	0	0	1	1	0	0	0	0
12h-13h	0	0	0	0	0	0	0	0	0	0
13h-14h	10	0	1	0	3	0	0	1	0	0
14h-15h	26	0	0	1	0	0	0	0	0	0
15h-16h	26	0	5	2	2	1	0	0	0	0
16h-17h	18	0	1	0	1	0	0	0	0	0
17h-18h	18	0	0	2	2	0	0	0	0	0
18h-19h	0	0	0	0	0	0	0	0	0	0
19h-20h	0	0	0	0	0	0	0	0	0	0
Total	147	0	10	10	26	4	0	1	2	0



P1: 0-15s | P2: 16-20s | P3: 21-30s | P4: 31-60s | P5: 61s |

	Calls handled					Abandoned calls				
	P1	P2	P3	P4	P5	P1	P2	P3	P4	P5
Monday 1	147	0	10	10	26	4	0	1	2	0
Tuesday 2	145	2	8	8	6	2	0	0	0	1
Wednesday 3	128	0	8	7	7	1	0	1	0	2
Thursday 4	122	2	5	11	23	2	1	0	0	1
Friday 5	0	0	0	0	0	0	0	0	0	0
Total	542	4	31	36	62	9	1	2	2	4





Period details by day

Period details by week

Period details by month

Period details by year

1.10.7 Reporting

You may use your own reporting tools to be able to produce your own reports provided **you do not use the Wazo server original tables**, but copy the tables to your own data server. You may use the following procedure as a template :

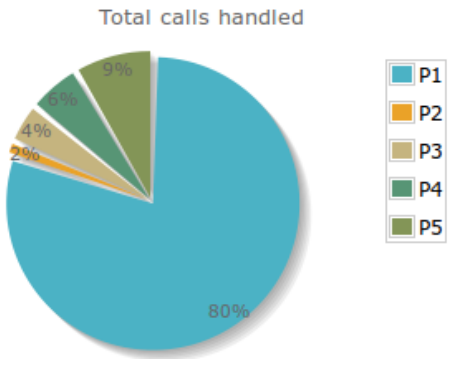
- Allow remote database access on Wazo
- Create a postgresql account read only on asterisk database
- Create target tables in your database located on the data server
- Copy the statistic table content to your data server

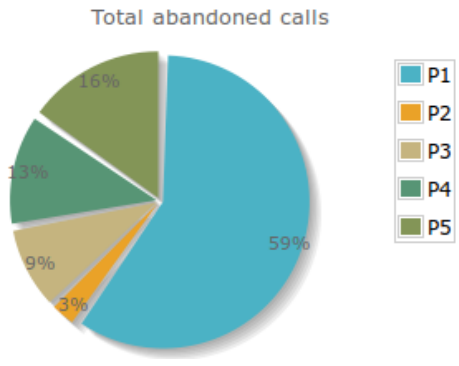
General Architecture

1. The *queue_log* table of the *asterisk* database is filled by events from Asterisk and by custom dialplan events
2. *xivo-stat fill_db* is then used to read data from the *queue_log* table and generate the tables *stat_call_on_queue* and *stat_queue_periodic*
3. The web interface generate tables and graphics from the *stat_call_on_queue* and *stat_queue_periodic* tables depending on the selected configuration

P1: 0-15s | P2: 16-20s | P3: 21-30s | P4: 31-60s | P5: 61s |

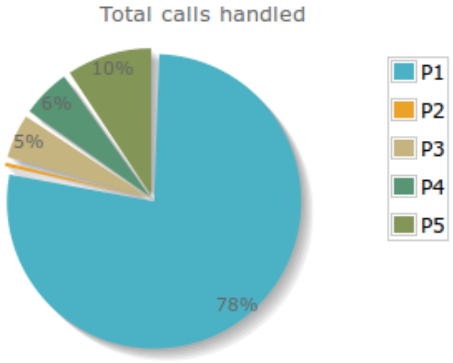
	Calls handled					Abandoned calls				
	P1	P2	P3	P4	P5	P1	P2	P3	P4	P5
27 week										
Monday	147	0	10	10	26	4	0	1	2	0
Tuesday	145	2	8	8	6	2	0	0	0	1
Wednesday	128	0	8	7	7	1	0	1	0	2
Thursday	122	2	5	11	23	2	1	0	0	1
Friday	0	0	0	0	0	0	0	0	0	0
28 week										
Monday	0	0	0	0	0	0	0	0	0	0
Tuesday	0	0	0	0	0	0	0	0	0	0
Wednesday	0	0	0	0	0	0	0	0	0	0
Thursday	0	0	0	0	0	0	0	0	0	0
Friday	0	0	0	0	0	0	0	0	0	0
29 week										
Monday	0	0	0	0	0	0	0	0	0	0
Tuesday	0	0	0	0	0	0	0	0	0	0
Wednesday	27	7	0	5	0	10	0	1	2	1
Thursday	0	0	0	0	0	0	0	0	0	0
Friday	0	0	0	0	0	0	0	0	0	0
30 week										
Monday	0	0	0	0	0	0	0	0	0	0
Tuesday	0	0	0	0	0	0	0	0	0	0
Wednesday	0	0	0	0	0	0	0	0	0	0
Thursday	0	0	0	0	0	0	0	0	0	0
Friday	0	0	0	0	0	0	0	0	0	0
31 week										
Monday	0	0	0	0	0	0	0	0	0	0
Tuesday	0	0	0	0	0	0	0	0	0	0
Wednesday	0	0	0	0	0	0	0	0	0	0
Total	569	11	31	41	62	19	1	3	4	5

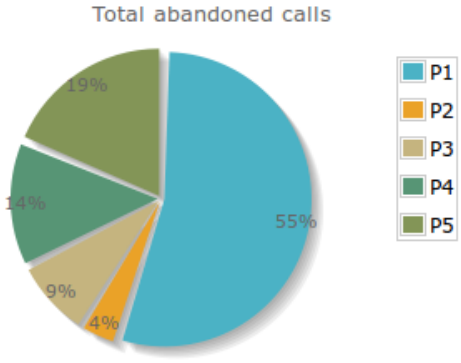




P1: 0-15s | P2: 16-20s | P3: 21-30s | P4: 31-60s | P5: 61s |

	Calls handled					Abandoned calls				
	P1	P2	P3	P4	P5	P1	P2	P3	P4	P5
January	0	0	0	0	0	0	0	0	0	0
February	21	0	0	1	7	28	4	9	9	5
March	10	0	0	0	0	10	0	0	0	0
April	4	0	0	0	0	16	0	0	0	0
May	1	0	0	0	0	3	0	0	0	2
June	1570	14	119	121	214	23	2	4	13	21
July	569	11	31	41	62	19	1	3	4	5
August	0	0	0	0	0	0	0	0	0	0
September	0	0	0	0	0	0	0	0	0	0
October	0	0	0	0	0	0	0	0	0	0
November	0	0	0	0	0	0	0	0	0	0
December	0	0	0	0	0	0	0	0	0	0
Total	2028	25	140	153	257	95	7	15	24	33





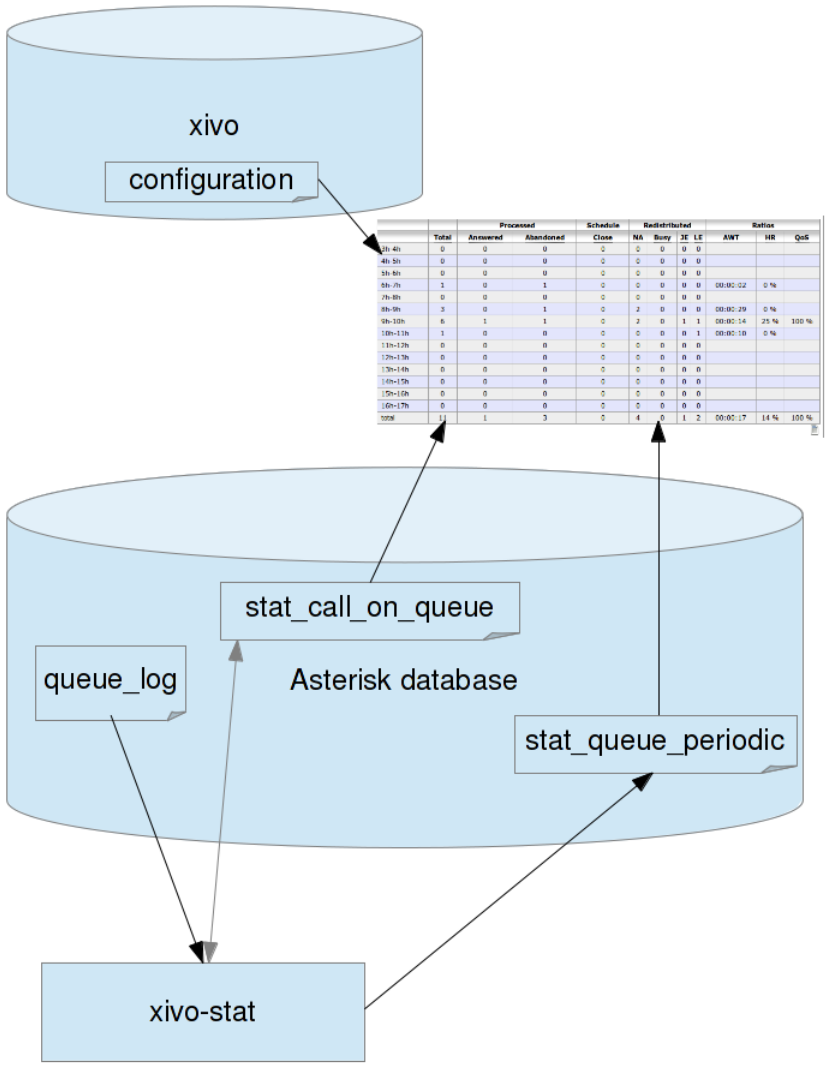


Fig. 1.98: Statistics Architecture

Statistic Data Table Content

stat_call_on_queue

This table is used to store each call individually. Each call received on a queue generates a single entry in this table containing time related fields and a foreign key to the agent who answered the call and another on the queue on which the call was received.

It also contains the status of the call ie. answered, abandoned, full, etc.

Field	Values	Description
id	generated	
callid	numeric value	This call id is also used in the CEL table and can be used to get call detail information
time	Call time	
ringtime		Ringing duration time in seconds
talktime		Talk time duration in seconds
waittime		Wait time duration in seconds
status		See status description below
queue_id		Id of the queue, the name of the queue can be found in table `stat_queue`, using this name queue details can be found in table `queuefeatures`
agent_id		Id of the agent, the agent name can be found in table `stat_agent`, using this name agent details can be found in table `agentfeatures` using the number in the second part of the name Exemple : Agent/1002 is agent with number 1002 in table `agentfeatures`

Queue Call Status

Status	Description
full	Call was not queued because queue was full, happens when the number of calls is greater than the maximum number of calls allowed to wait
closed	Closed due to the schedule applied to the queue
joinempty	No agents were available in the queue to take the call (follows the join empty parameter of the queue)
leaveempty	No agents available while the call was waiting in the qeuue
divert_ca_ratio	Call diverted because the ratio number of agent number of calls waiting configured was exceeded
divert_waittime	Call diverted because the maximum expected waiting time configured was exceeded
answered	Call was answered
abandoned	Call hangup by the caller
timeout	Call stayed longer than the maximum time allowed in queue parameter

stat_queue_periodic Table

This table is an aggregation of the queue_log table.

This table contains counters on each queue for each given period. The granularity at the time of this writing is an hour and is not configurable. This table is then used to compute statistics for a given range of hours, days, week, month or year.

Field	Description
id	Generated id
time	time period, all counters are aggregated for an hour
answered	Number of answered calls during the period
abandoned	Number of abandoned calls during the period
total	Total calls received during the period
full	Number of calls received when queue was full
closed	Number of calls received on close
joinempty	Number of calls received no agents available
leaveempty	Number of calls diverted agents not available during the wait
divert_ca_ratio	Number of calls diverted due to the number of agent number versus calls waiting configured was exceeded
divert_waittime	Number of calls diverted because the maximum expected waiting time configured was exceeded
timeout	Number of calls diverted because the maximum time allowed in queue parameter was exceeded
queue_id	

stat_agent

This table is used to match agents to an id that is different from the id in the agent configuration table. This is necessary to avoid loosing statistics on a deleted agent. This also means that if an agent changes number ie. Agent/1001 to Agent/1202, the supervisor will have to take this information into account when viewing the statistics. Affecting an old number to a another agent also means that the supervisor will have to ignore entries for this given agent for the period before the number assignment to the new agent.

stat_queue

This table is used to store queues in a table that is different from the queue configuration table. This is necessary to avoid losing statistics on a deleted queue. Renaming a queue is also not handled at this time.

1.11 High Availability (HA)

The HA (High Availability) solution in Wazo makes it possible to maintain basic telephony function whether your main Wazo server is running or not. When running a Wazo HA cluster, users are guaranteed to never experience a downtime of more than 5 minutes of their basic telephony service.

The HA solution in Wazo is based on a 2-nodes "master and slave" architecture. In the normal situation, both the master and slave nodes are running in parallel, the slave acting as a "hot standby", and all the telephony services are provided by the master node. If the master fails or must be shutdown for maintenance, then the telephony devices automatically communicate with the slave node instead of the master one. Once the master is up again, the telephony devices failback to the master node. Both the failover and the failback operation are done automatically, i.e. without

any user intervention, although an administrator might want to run some manual operations after failback as to, for example, make sure any voicemail messages that were left on the slave are copied back to the master.

1.11.1 Prerequisites

The HA in Wazo only works with telephony devices (i.e. phones) that support the notion of a primary and backup telephony server.

- Phones must be able to reach the master and the slave (take special care if master and slave are not in the same subnet)
- If firewalling, the master must be allowed to join the slave on ports 22 and 5432
- If firewalling, the slave must be allowed to join the master with an ICMP ping
- Trunk registration timeout (`expiry`) should be less than 300 seconds (5 minutes)
- The slave must have no provisioning plugins installed.

The HA solution is guaranteed to work correctly with *the following devices*.

1.11.2 Quick Summary

- You need two configured Wazo (wizard passed)
- Configure one Wazo as a master -> setup the slave address (VoIP interface)
- Restart services (wazo-service restart) on master
- Configure the other Wazo as a slave -> setup the master address (VoIP interface)
- Configure file synchronization by runnning the script `xivo-sync -i` on the master
- Start configuration synchronization by running the script `xivo-master-slave-db-replication <slave_ip>` on the master
- Resynchronize all your devices
- Configure the Wazo Clients

That's it, you now have a HA configuration, and every hour all the configuration done on the master will be reported to the slave.

1.11.3 Configuration Details

First thing to do is to *install 2 Wazo*.

Important: When you upgrade a node of your cluster, you must also upgrade the other so that they both are running the same version of Wazo. Otherwise, the replication might not work properly.

You must configure the HA in the Web interface (*Configuration* → *Management* → *High Availability* page).

You can configure the master and slave in whatever order you want.

You must also run `xivo-sync -i` on the master to setup file synchronization. Running `xivo-sync -i` will create a passwordless SSH key on the master, stored under the `/root/.ssh` directory, and will add it to the `/root/.ssh/authorized_keys` file on the slave. The following directories will then be rsync'ed every hour:

- /etc/asterisk/extensions_extra.d

- /etc/xivo/asterisk
- /var/lib/asterisk/agi-bin
- /var/lib/asterisk/moh
- /var/lib/xivo/certificates
- /var/lib/xivo/sounds/acd
- /var/lib/xivo/sounds/playback

Warning: When the HA is configured, some changes will be automatically made to the configuration of Wazo.

SIP expiry value on master and slave will be automatically updated:

- min: 3 minutes
- max: 5 minutes
- default: 4 minutes

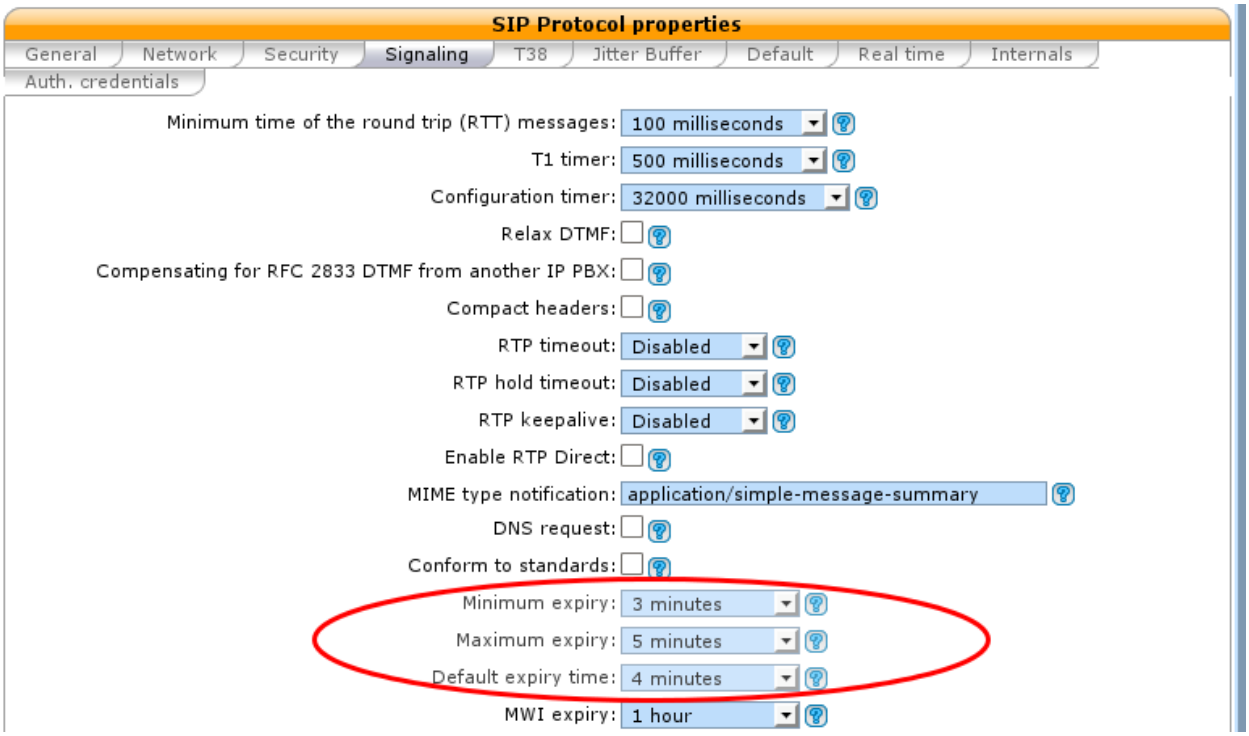


Fig. 1.99: *Services → IPBX → General Settings → SIP Protocol*

The provisioning server configuration will be automatically updated in order to allow phones to switch from Wazo power failure.

Warning: Do not change these values when the HA is configured, as this may cause problems. These values will be reset to blank when the HA is disabled.

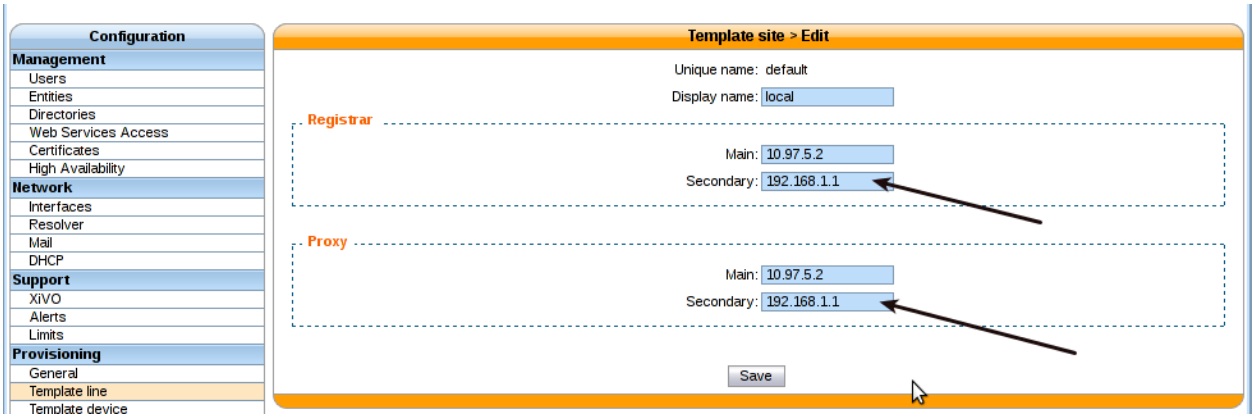


Fig. 1.100: *Configuration → Provisioning → Template Line → Edit default*

Important: For the telephony devices to take the new proxy/registrar settings into account, you must *resynchronize the devices* or restart them manually.

Disable node

Default status of HIGH AVAILABILITY (HA) is disabled:

Note: You can reset at any time by choosing a server mode (disabled)

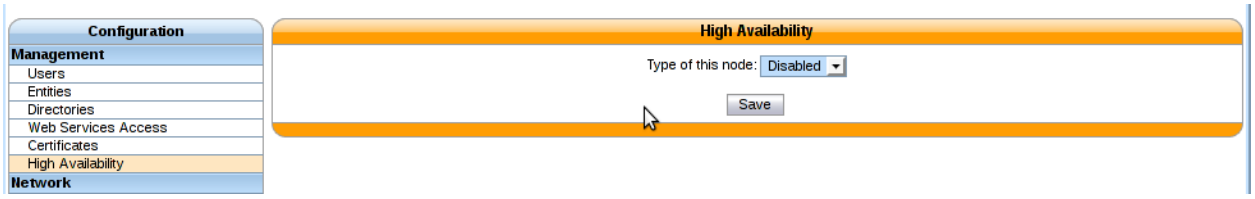


Fig. 1.101: HA Dashboard Disabled (default state)

Important: You have to restart services (wazo-service restart) once the master node is disabled.

Master node

In choosing the method `Master` you must enter the IP address **of the VoIP interface** of the slave node.

Important: You have to restart all services (wazo-service restart) once the master node is configured.

Slave node

In choosing the method `Slave` you must enter the IP address **of the VoIP interface** of the master node.

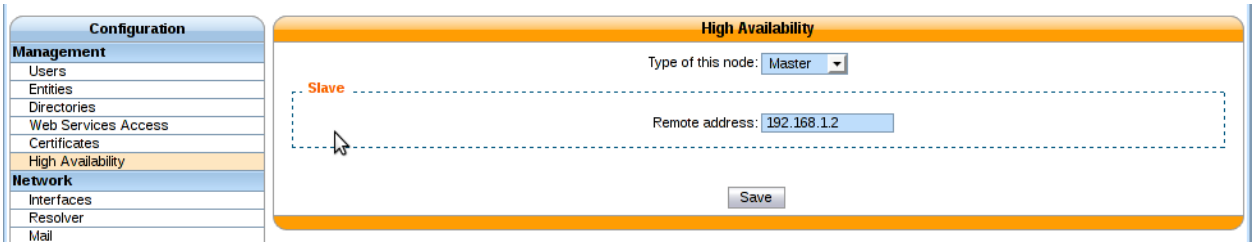


Fig. 1.102: HA Dashboard Master

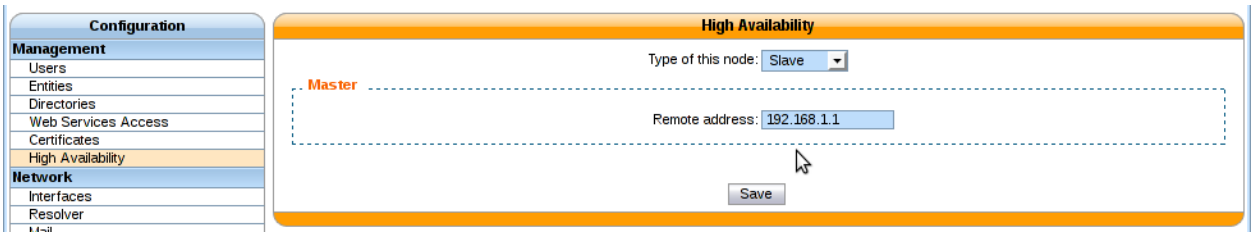


Fig. 1.103: HA Dashboard Slave

Replication Configuration

Once master slave configuration is completed, Wazo configuration is replicated from the master node to the slave every hour (:00).

Replication can be started manually by running the replication scripts on the master:

```
xivo-master-slave-db-replication <slave_ip>
xivo-sync
```

The replication does not copy the full Wazo configuration of the master. Notably, these are excluded:

- All the network configuration (i.e. everything under the *Configuration* → *Network* section)
- All the support configuration (i.e. everything under the *Configuration* → *Support* section)
- Call logs
- Call center statistics
- Certificates
- HA settings
- Provisioning configuration
- Voicemail messages

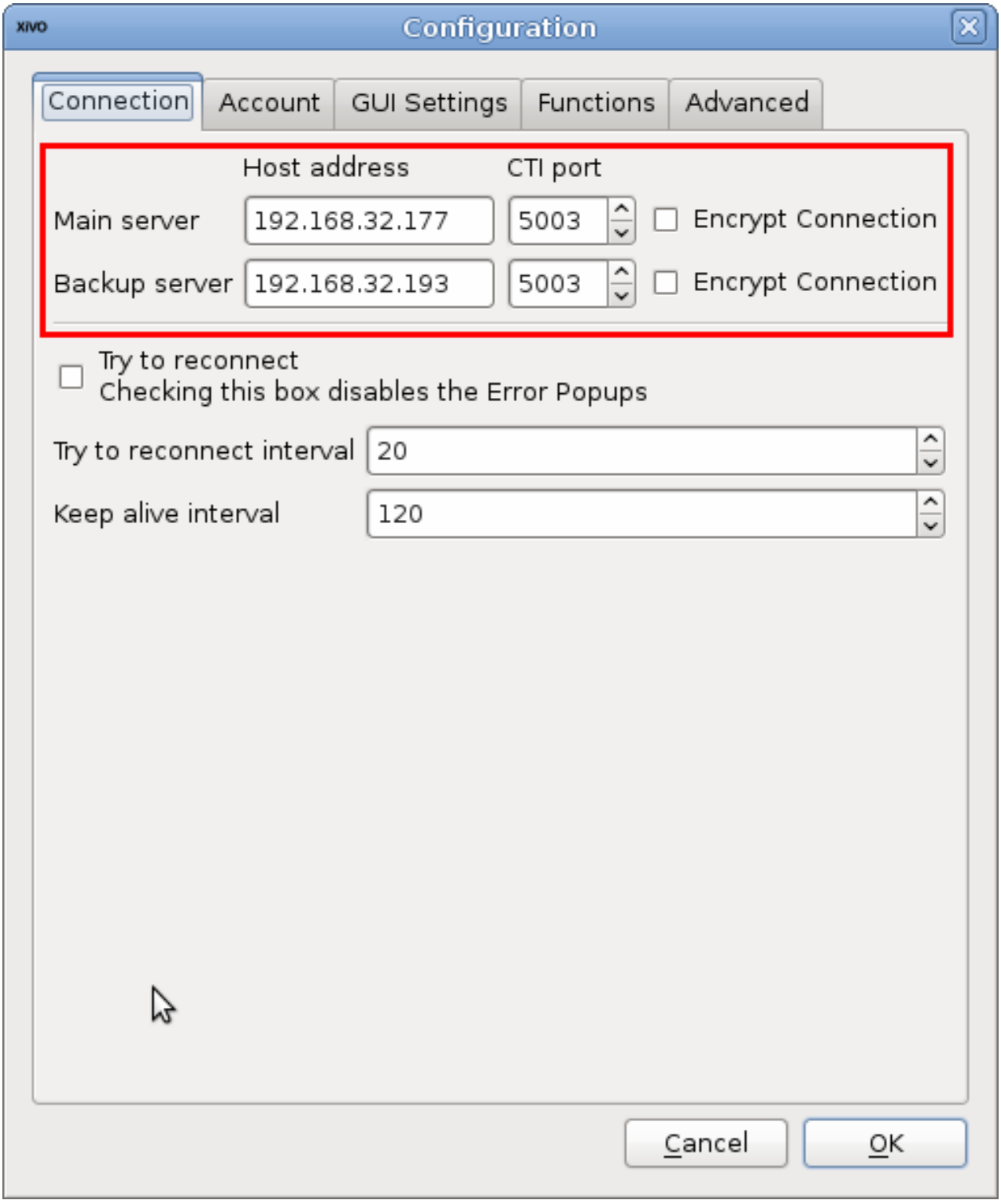
Less importantly, these are also excluded:

- Queue logs
- CELs

Wazo Client

You have to enter the master and slave address in the `Connection` tab of the Wazo Client configuration :

The main server is the master node and the backup server is the slave node.


XIVO
Configuration
Connection
Account
GUI Settings
Functions
Advanced
Host address
CTI port
Main server
192.168.32.177
5003
Encrypt Connection
Backup server
192.168.32.193
5003
Encrypt Connection
Try to reconnect
Checking this box disables the Error Popups
Try to reconnect interval
20
Keep alive interval
120
Cancel
OK

When connecting the Wazo Client with the main server down, the login screen will hang for 3 seconds before connecting to the backup server.

1.11.4 Internals

4 scripts are used to manage services and data replication.

- xivo-master-slave-db-replication <slave_ip> is used on the master to replicate the master's data on the slave server. It runs on the master.
- xivo-manage-slave-services {start,stop} is used on the slave to start, stop monit and asterisk. The services won't be restarted after an upgrade or restart.
- xivo-check-master-status <master_ip> is used to check the status of the master and enable or disable services accordingly.
- xivo-sync is used to sync directories from master to slave.

1.11.5 Limitations

When the master node is down, some features are not available and some behave a bit differently. This includes:

- Call history / call records are not recorded.
- Voicemail messages saved on the master node are not available.
- Custom voicemail greetings recorded on the master node are not available.
- Phone provisioning is disabled, i.e. a phone will always keep the same configuration, even after restarting it.
- Phone remote directory is not accessible, because provisioned IP address points to the master.

Note that, on failover and on failback:

- DND, call forwards, call filtering, . . . , statuses may be lost if changed recently.
- If you are connected as an agent, then you might need to reconnect as an agent when the master goes down. Since it's hard to know when the master goes down, if your CTI client disconnects and you can't reconnect it, then it's a sign the master might be down.

Additionally, only on failback:

- Voicemail messages are not copied from the slave to the master, i.e. if someone left a message on your voicemail when the master was down, you won't be able to consult it once the master is up again.
- More generally, custom sounds are not copied back. This includes recordings.

Here's the list of limitations that are more relevant on an administrator standpoint:

- The master status is up or down, there's no middle status. This mean that if Asterisk is crashed the Wazo is still up and the failover will NOT happen.

1.11.6 Berofos Integration

Berofos Integration

Wazo offers the possibility to integrate a berofos failover switch within a HA cluster.

This is useful if you have one or more ISDN lines (i.e. T1/E1 or T0 lines) that you want to use whatever the state of your Wazo HA cluster. To use a berofos within your Wazo HA installation, you need to properly configure both your

berofos and your Wazo, then the berofos will automatically switch your ISDN lines from your master node to your slave node if your master goes down, and vice-versa when it comes back up.

You can also use a Berofos failover switch to secure the ISDN provider lines when installing a Wazo in front of an existing PBX. The goal of this configuration is to mitigate the consequences of an outage of the Wazo : with this equipment the ISDN provider links could be switched to the PBX directly if the Wazo goes down.

Wazo **does not offer natively** the possibility to configure Berofos in this failover mode. The *Berofos Integration with PBX* section describes a workaround.

Installation and Configuration

Master Configuration

There is nothing to be done on the master node.

Slave Configuration

First, install the bntools package:

```
apt-get install bntools
```

This will make the `bnfos` command available.

You can then connect your berofos to your network and power it on. By default, the berofos will try to get an IP address via DHCP. If it is not able to get such address from a DHCP server, it will take the 192.168.0.2/24 IP address.

Note: The DHCP server on Wazo does not offer IP addresses to berofos devices by default.

Next step is to create the `/etc/bnfos.conf` file via the following command:

```
bnfos --scan -x
```

If no berofos device is detected using this last command, you'll have to explicitly specify the IP address of the berofos via the -h option:

```
bnfos --scan -x -h <berofos ip>
```

At this stage, your `/etc/bnfos.conf` file should contains something like this:

```
[fos1]
mac = 00:19:32:00:12:1D
host = 10.34.1.50
#login = <user>:<password>
```

It is advised to configure your berofos with a static IP address. You first need to put your berofos into *flash mode* :

- press and hold the black button next to the power button,
- power on your berofos,
- release the black button when the red LEDs of port D start blinking.

Then, you can issue the following command, by first replacing the network configuration with your one:

```
bnfos --netconf -f fos1 -i 10.34.1.20 -n 255.255.255.0 -g 10.34.1.1 -d 0
```

Note:

- `-i` is the IP address
- `-n` is the netmask
- `-g` is the gateway
- `-d 0` is to disable DHCP

You can then update your berofos firmware to version 1.53:

```
wget http://www.beronet.com/downloads/berofos/bnfos_v153.bin
bnfos --flash bnfos_v153.bin -f fos1
```

Once this is done, you'll have to reboot your berofos in operationnal mode (that is in normal mode).

Then you must rewrite the `/etc/bnfos.conf` (mainly if you changed the IP address):

```
bnfos --scan -x -h <berofos ip>
```

Now that your berofos has proper network configuration and an up to date firmware, you might want to set a password on your berofos:

```
bnfos --set apwd=<password> -f fos1
bnfos --set pwd=1 -f fos1
```

You must then edit the `/etc/bnfos.conf` and replace the login line to something like:

```
login = admin:<password>
```

Next, configure your berofos for it to work correctly with the Wazo HA:

```
bnfos --set wdog=0 -f fos1
bnfos --set wdogdef=0 -f fos1
bnfos --set scenario=0 -f fos1
bnfos --set mode=1 -f fos1
bnfos --set modedef=1 -f fos1
```

This, among other things, disable the watchdog. The switching from one relay mode to the other will be done by the Wazo slave node once it detects the master node is down, and vice-versa.

Finally, you can make sure everything works fine by running the xivo-berofos command:

```
xivo-berofos master
```

The green LEDs on your berofos should be lighted on ports A and B.

Connection

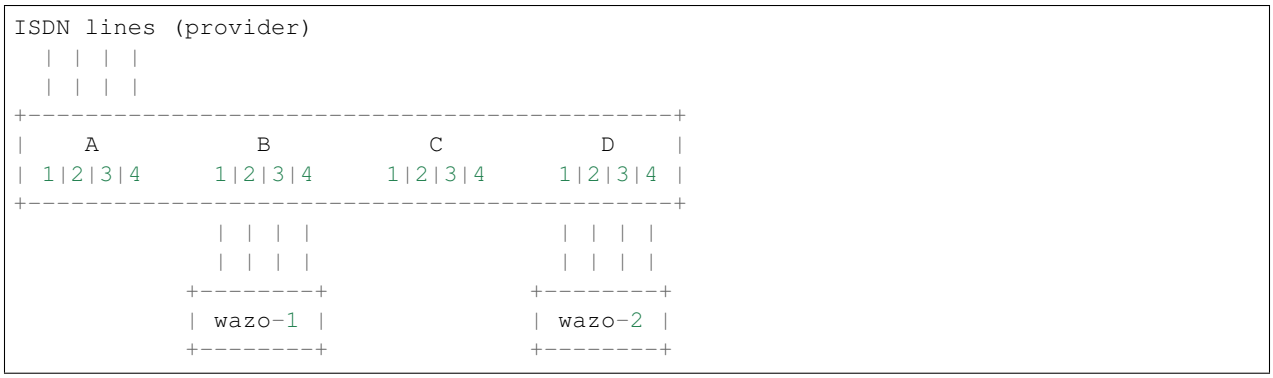
Two Wazo

Here's how to connect the ISDN lines between your berofos with:

- two Wazo in high availability

In this configuration you can protect **up two 4** ISDN lines. If more than 4 ISDN lines to protect, you must set up a *Multiple berofos* configuration.

Here's an example with 4 ISDN lines coming from your telephony provider:



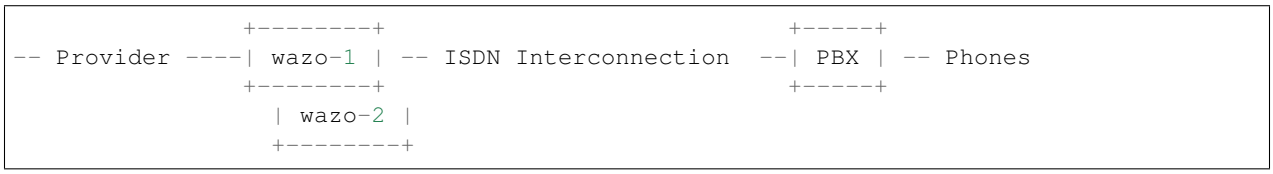
Two Wazo and one PBX

Here's how to connect your berofos with:

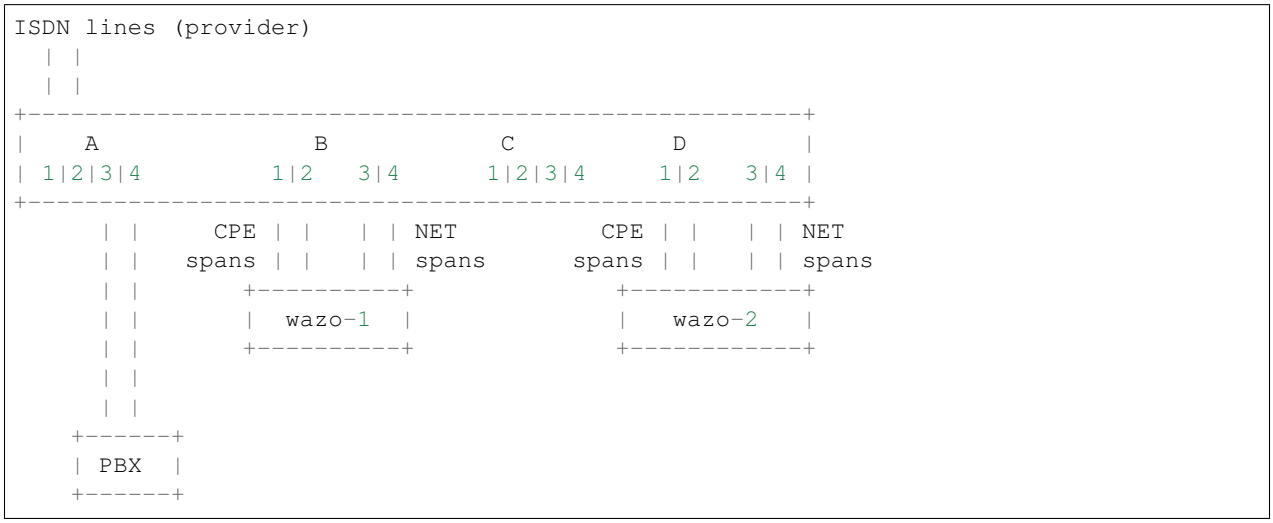
- two Wazo in high availability,
- one PBX.

In this configuration you can protect **up two 2** ISDN lines. If more than 2 ISDN lines to protect, you must set up a *Multiple berofos* configuration.

Logical view:



This example shows the case where there are 2 ISDN lines coming from your telephony provider:



One Wazo and one PBX

This case is not currently supported. You'll find a workaround in the *Berofos Integration with PBX* section.

Multiple berofos

It's possible to use more than 1 berofos with Wazo.

For each supplementary berofos you want to use, you must first configure it properly like you did for the first one. The only difference is that you need to add a berofos declaration to the `/etc/bnfos.conf` file instead of creating/overwriting the file. Here's an example of a valid config file for 2 berofos:

```
[fos1]
mac = 00:19:32:00:12:1D
host = 10.100.0.201
login = admin:foobar

[fos2]
mac = 00:11:22:33:44:55
host = 10.100.0.202
login = admin:barfoo
```

Warning: berofos name must follow the pattern `fosX` where X is a number starting with 1, then 2, etc. The `bnfos` tool won't work properly if it's not the case.

Operation

When your Wazo switch the relay mode of your berofos, it logs the event in the `/var/log/syslog` file.

Default mode

Note that when the berofos is off, the A and D ports are connected together. This behavior is not customizable.

Uninstallation

It is important to remove the `/etc/bnfos.conf` file on the slave node when you don't want to use anymore your berofos with your Wazo.

Reset the Berofos

You can reset the berofos configuration :

1. Power on the berofos,
2. When red and green LEDs are still lit, press & hold the black button,
3. Release it when the red LEDs of the D port start blinking fast
4. Reboot the beronet, it should have lost its configuration.

External links

- berofos user manual

1.11.7 Troubleshooting

When replicating the database between master and slave, if you encounter problems related to the system locale, see *PostgreSQL localization errors*.

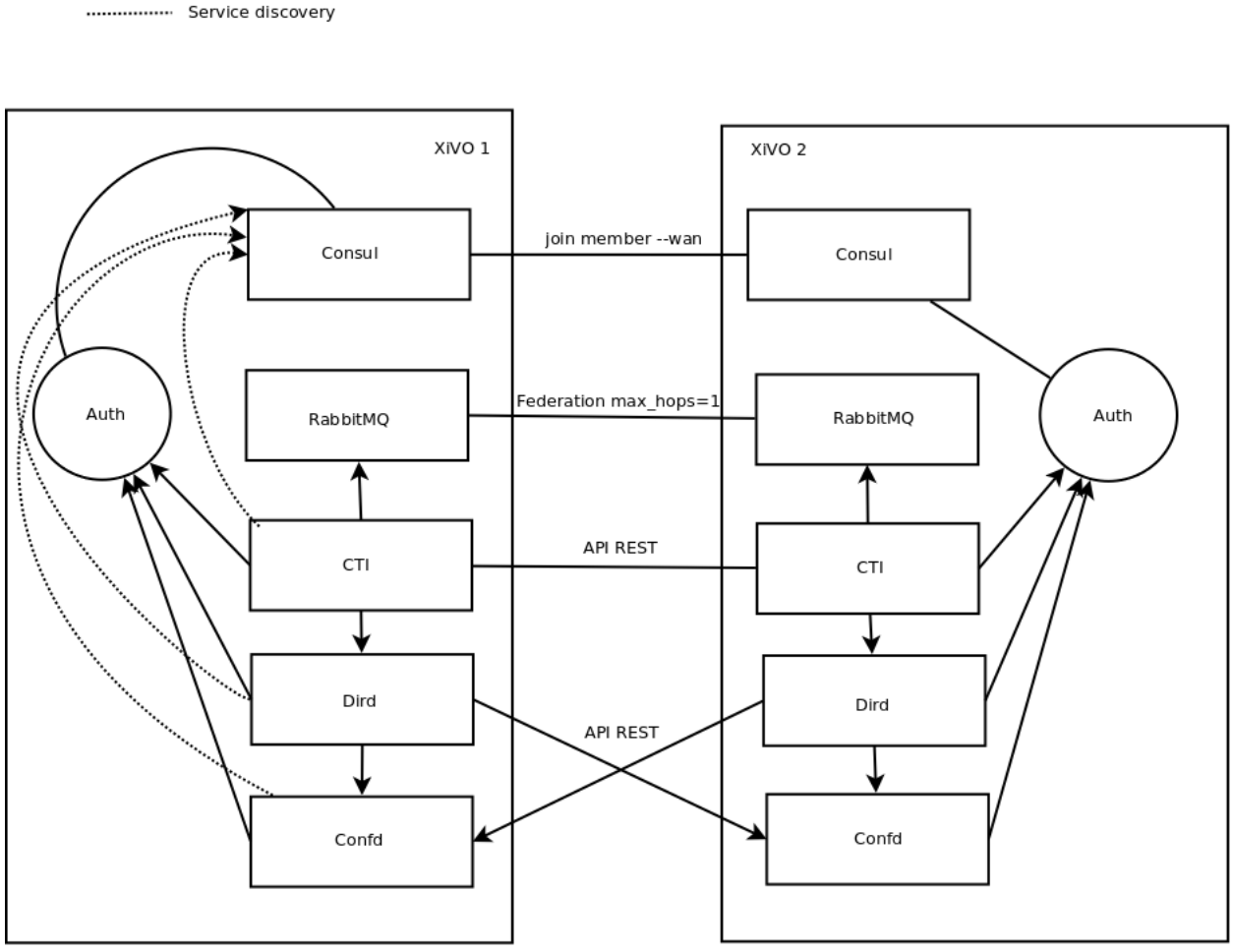
1.12 Scalability and Distributed Systems

This section gathers configuration that are possible using Wazo to feature rich scalable communication systems.

1.12.1 Contact and Presence Sharing

Wazo allow the administrator to share presence and statuses between multiple installations. For example, an enterprise could have a Wazo in each office and still be able to search, contact and view the statuses of colleagues in other offices.

This page will describe the steps that are required to configure such use case.



Prerequisite

1. All Wazo that you interconnect should be on the same version
2. All ports necessary for communication should be open *Network*

Warning: If you are cloning a virtual machine or copy the database, the UUID of the two Wazo will be the same, you must regenerate them in the *infos* table of the *asterisk* database and restart all services. You must also remove all consul data that included the old UUID.

Warning: Telephony will be interrupted during the configuration period.

Warning: The configuration must be applied to each Wazo you want to interconnect. For example, if 6 different Wazo are to be connected, the configuration for all other Wazo should be added. This does not apply to the message bus which can use a ring policy, each Wazo talking to its two neighbours.

Warning: You should use your firewall to restrict access to the HTTP ports of consul and xivo-ctid, because they don't have any authentication mechanism enabled.

Note: In an architecture with a lot of Wazo, we recommend that you centralize some services, such as xivo-dird, to make your life easier. Don't forget redundancy. This applies also to RabbitMQ and Consul. In this case, the configuration will have to be done entirely manually in YAML config files.

For this procedure, the following name and IP addresses will be used:

- Wazo 1: 192.168.1.124
- Wazo 2: 192.168.1.125

Configure service discovery on all wazo-services

The default configuration for each services in wazo uses the IP address of the "eth0" interface as it's advertised IP address. To advertise a different address you will have to override the default configuration.

To do so you have to create a file in the `conf.d` directory of all services.

First, create a configuration file that will be used for all services.

```
service_discovery:
  advertise_address: 192.168.1.124
```

or

```
service_discovery:
  advertise_address: auto
  advertise_address_interface: "<interface of the address to advertise. ex: enp0s3>"
```

Save that file and create a symlink such that each service will be able to used that file.

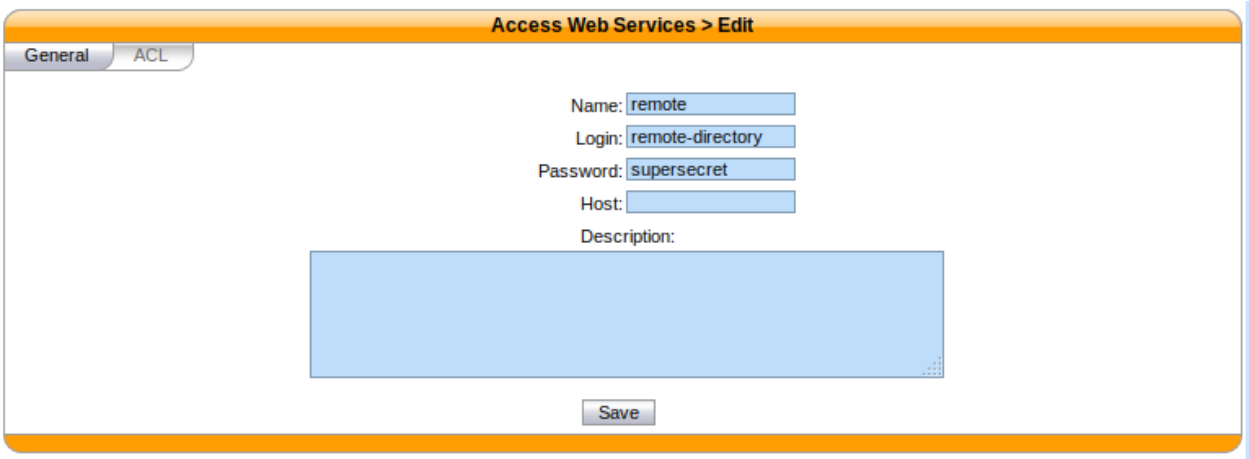
```
for dir in /etc/{xivo,wazo}-*/conf.d; do ln -sf /root/config/service_discovery.yml
↪$dir/service_discovery.yml; done
```

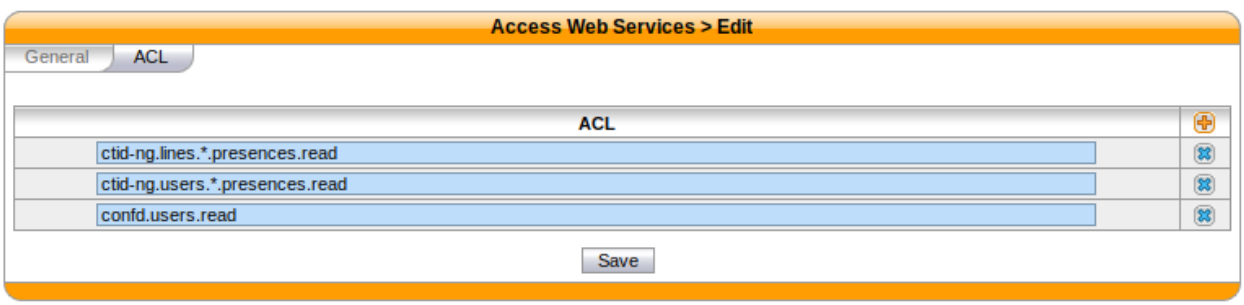
Add a Web Service User

The first thing to do is to create a new web service access to be able to search users and get there presences using the following ACL.

- ctid-ng.lines.*.presences.read
- ctid-ng.users.*.presences.read
- confd.users.read

This can be done in *Configuration* → *Management* → *Web Services Access*





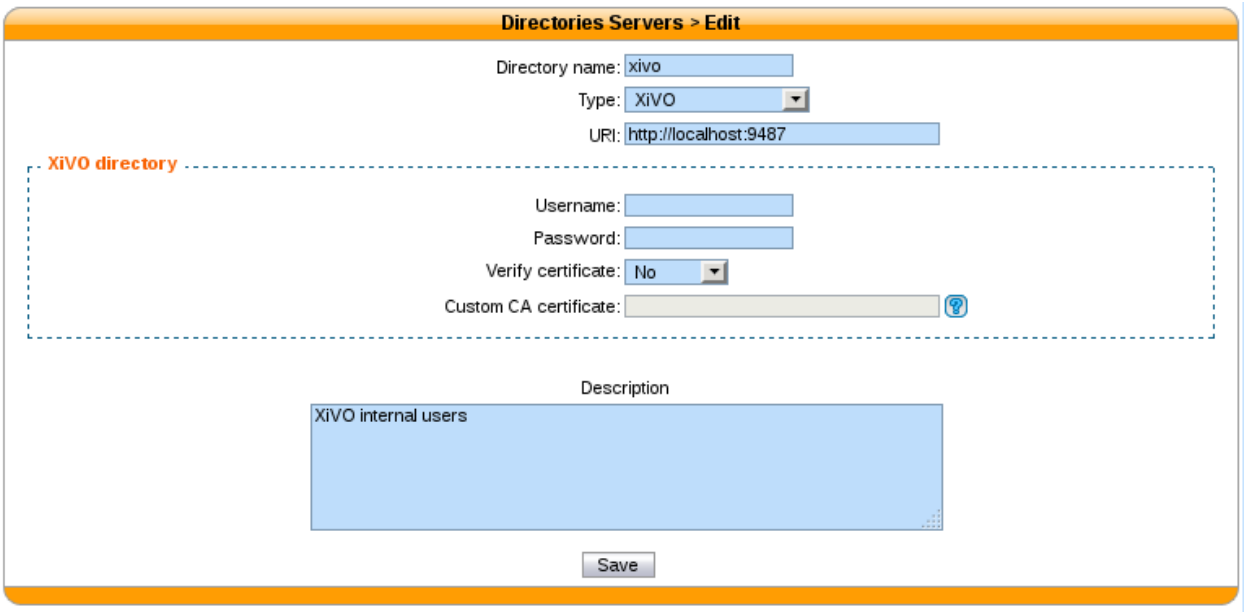
Configuring the directories

Add New Directory Sources for Remote Wazo

For each remote Wazo a new directory has to be created in *Configuration* → *Management* → *Directories*

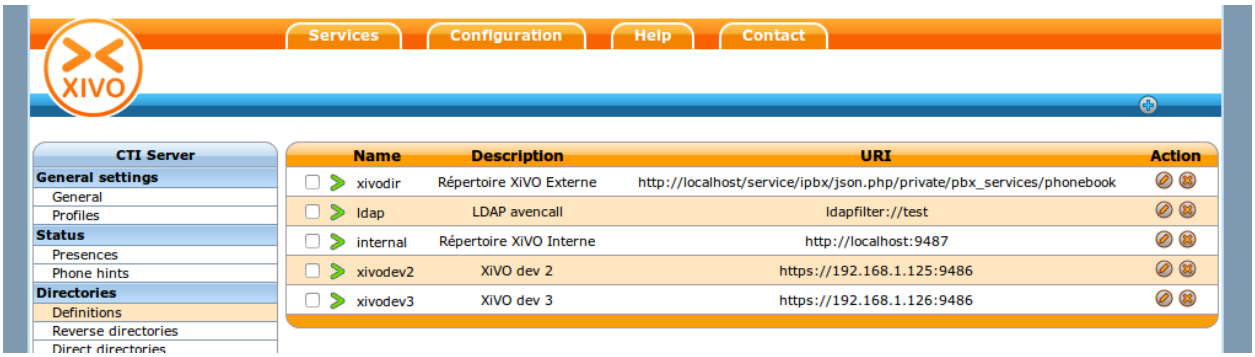
Note: We recommend doing a working configuration without certificate verification first. Once you get it working, enable certificate verification.


Services
Configuration
Help
Contact
Configuration
Management
Users
Entities
General
Directories
Web Services Access
Certificates
High Availability
LDAP Servers
Network
Interfaces
Name
Uri
Action
phonebook
http://localhost/service/ipbx/json.php/private/pbx_services/phonebook
XiVO
http://localhost:9487
xivo-dev-2
https://192.168.1.125:9486
xivo-dev-3
https://192.168.1.126:9486

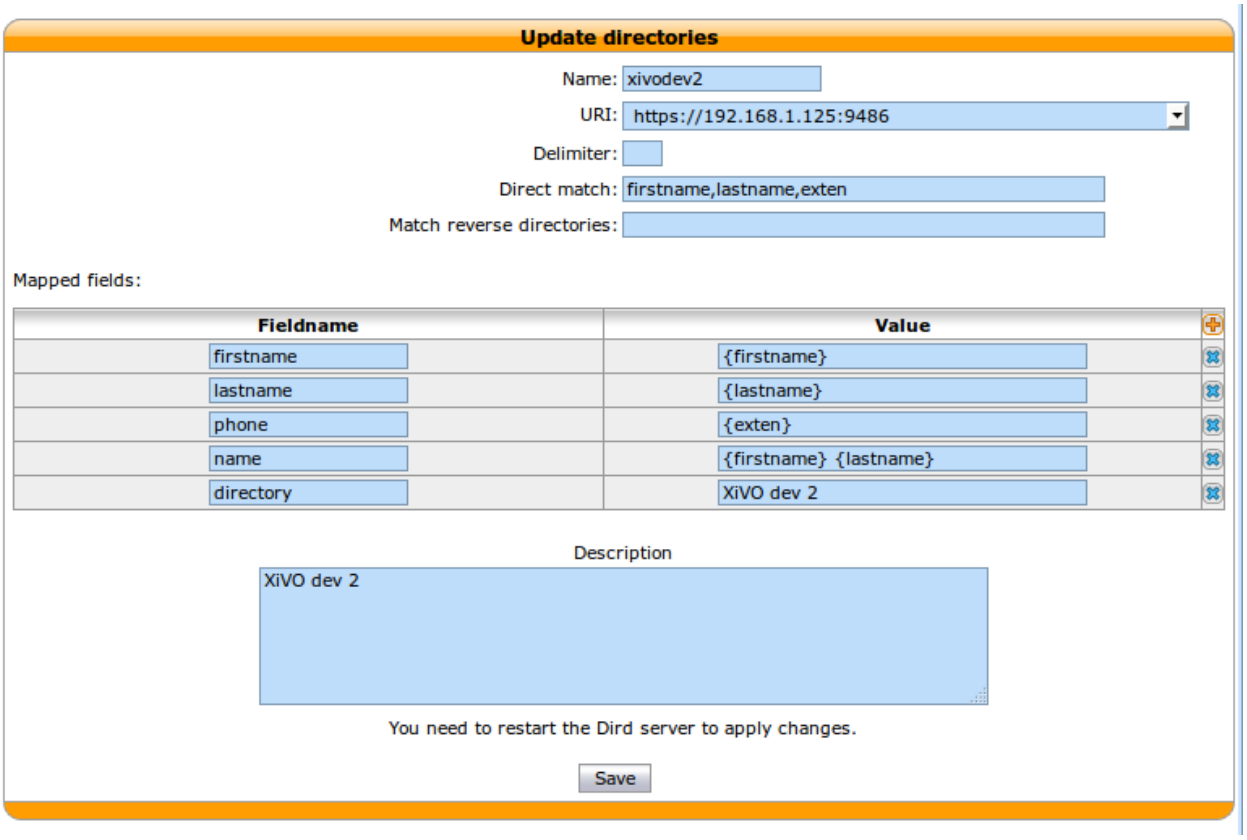

Directories Servers > Edit
Directory name:
xivo
Type:
XiVO
URI:
http://localhost:9487
XiVO directory
Username:
Password:
Verify certificate:
No
Custom CA certificate:
Description
XiVO internal users
Save

Add a Directory Definition for Each New Directories

To add a new directory definition, go to *Services* → *CTI Server* → *Directories* → *Definitions*



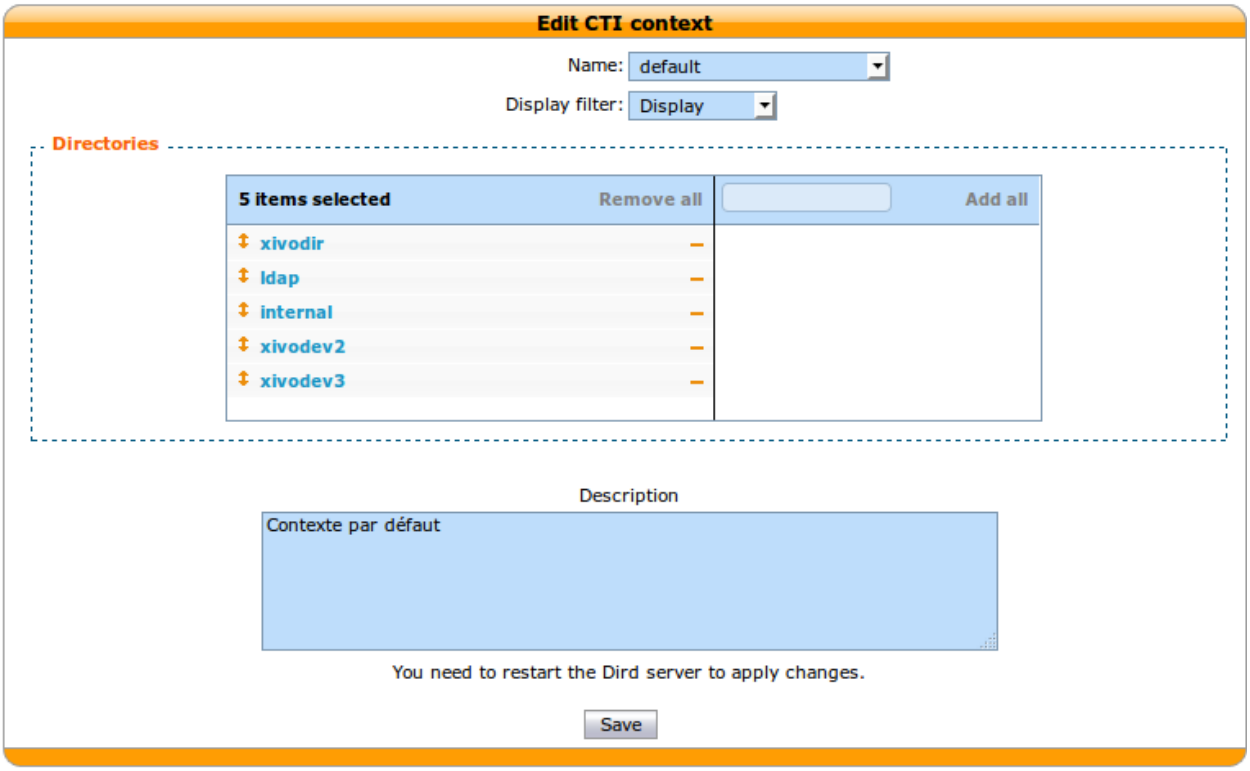
In each directory definition, add the fields to match the configured *Display filters*



Add the New Definitions to Your Dird Profiles

At the moment of this writing xivo-dird profiles are mapped directly to the user's profile. For each internal context where you want to be able to see user's from other Wazo, add the new directory definitions in *Services* → *CTI Server* → *Directories* → *Direct directories*.


Services
Configuration
Help
Contact
CTI Server
General settings
General
Profiles
Status
Presences
Phone hints
Directories
Definitions
Reverse directories
Direct directories
Display filters
Sheets
Context
Description
Action
__switchboard_directory
default
Contexte par défaut


Edit CTI context
Name:
default
Display filter:
Display
Directories
5 items selected
Remove all
Add all
xivodir
ldap
internal
xivodev2
xivodev3
Description
Contexte par défaut
You need to restart the Dird server to apply changes.
Save

Restart xivo-dird

To apply the new directory configuration, you can either restart from:

- *Services* → *IPBX*
- on the command line *service xivo-dird restart*

Check that the Configuration is Working

At this point, you should be able to search for users on other Wazo from the *People Xlet*.

Configuring RabbitMQ

Create a RabbitMQ user

```
rabbitmqctl add_user xivo xivo
rabbitmqctl set_user_tags xivo administrator
rabbitmqctl set_permissions -p / xivo ".*" ".*" ".*"
rabbitmq-plugins enable rabbitmq_federation
```

Restart RabbitMQ

```
systemctl restart rabbitmq-server
```

Setup Message Federation

```
rabbitmqctl set_parameter federation-upstream xivo-dev-2 '{"uri":"amqp://
↪xivo:xivo@192.168.1.125","max-hops":1}'  # remote IP address
rabbitmqctl set_policy federate-xivo 'xivo' '{"federation-upstream-set":"all"}' --
↪priority 1 --apply-to exchanges
```

Check That Service Discovery is Working

```
apt-get install consul-cli
```

```
consul-cli agent services --ssl --ssl-verify=false
```

The output should include a service names *xivo-ctid* with an address that is reachable from other XiVO.

```
{"consul": {"ID": "consul",
            "Service": "consul",
            "Tags": [],
            "Port": 8300,
            "Address": ""},
 "e546a652-e290-47e2-8519-ec3642daa6e6": {"ID": "e546a652-e290-47e2-8519-ec3642daa6e6
↪",
                                         "Service": "xivo-ctid",
```

```
                                                "Tags": ["xivo-ctid",
                                                         "607796fc-24e2-4e26-8009-
↪cbb48a205512"],
                                                "Port": 9495,
                                                "Address": "192.168.1.124"}}
```

Configure Ctid-ng

Add a configuration file on ctid-ng conf.d directory named discovery.yml with your configuration.

The *service_id* and *service_key* are the ones you defined *earlier* in the web interface.

```
remote_credentials:
  xivo-2:
    xivo_uuid: 1cc7fbf2-5f13-4898-9869-986990cb9b0a
    service_id: remote-directory
    service_key: supersecret
```

To get the xivo_uuid information on your second Wazo, use the command:

```
echo $XIVO_UUID
```

Restart xivo-ctid-ng

```
systemctl restart xivo-ctid-ng
```

Configure Consul

Stop Wazo

```
wazo-service stop
systemctl stop consul
```

Remove All Consul Data

```
rm -rf /var/lib/consul/raft/
rm -rf /var/lib/consul/serf/
rm -rf /var/lib/consul/services/
rm -rf /var/lib/consul/tmp/
rm -rf /var/lib/consul/checks/
```

Configure Consul to be Reachable from Other Wazo

Add a new configuration file */etc/consul/xivo/interconnection.json* with the following content where *advertise_addr* is reachable from other Wazo.

```
{
"client_addr": "0.0.0.0",
"bind_addr": "0.0.0.0",
"advertise_addr": "192.168.1.124"  // The IP address of this Wazo, reachable from␣
↪outside
}
```

Check that the Configuration is Valid

```
consul configtest --config-dir /etc/consul/xivo/
```

No output means that the configuration is valid.

Start Consul

```
systemctl start consul
```

Start Wazo

```
wazo-service start
```

Join the Consul Cluster

Join another member of the Consul cluster. Only one join is required as members will be propagated.

```
consul join -wan 192.168.1.125
```

Check that Consul Sees other Consul

List other members of the cluster with the following command

```
consul members -wan
```

Check consul logs for problems

```
consul monitor
```

Check That Everything is Working

There is no further configuration needed, you should now be able to connect your Wazo Client and search contacts from the People Xlet. When looking up contacts of another Wazo, you should see their phone status, their user availability, and agent status dynamically.

Troubleshooting

Chances are that everything won't work the first time, here are some interesting commands to help you debug the problem.

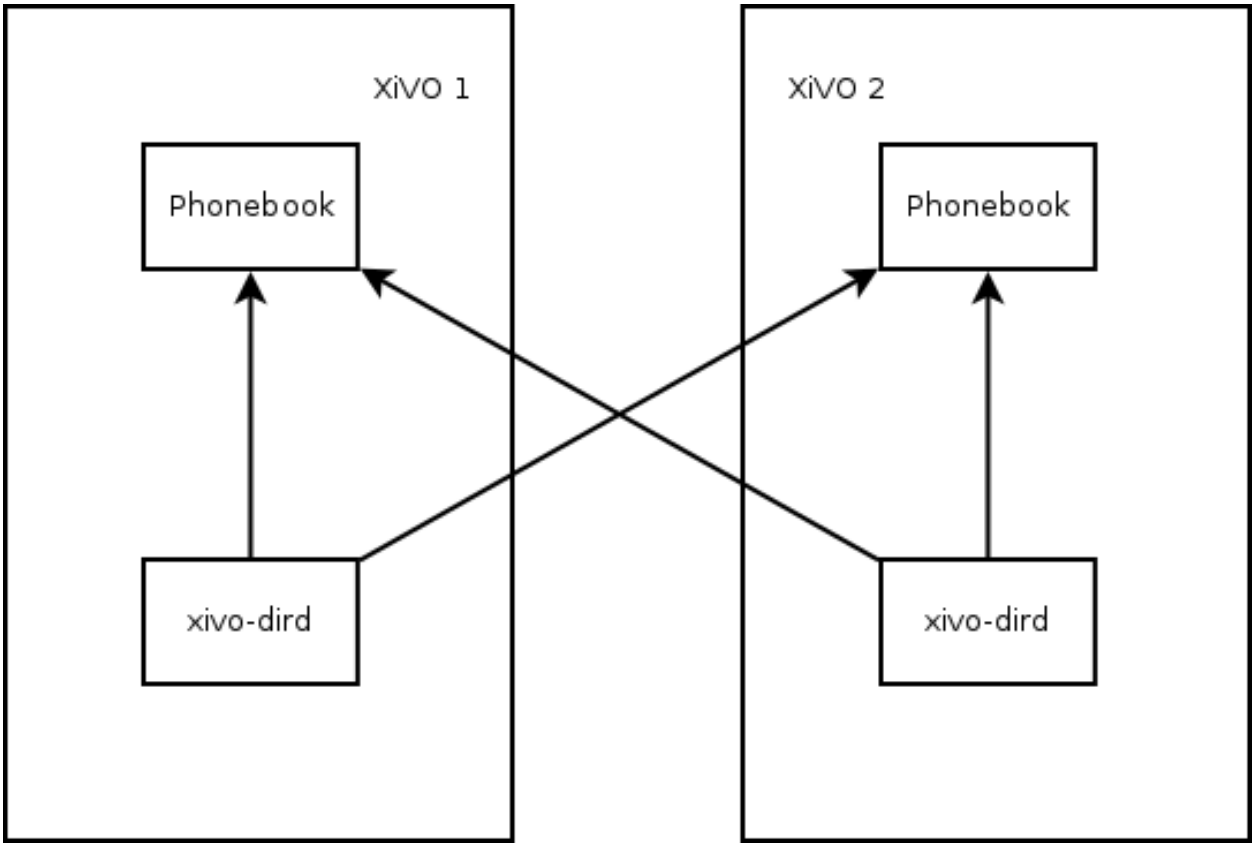
```
tail -f /var/log/xivo-dird.log
tail -f /var/log/xivo-ctid-ng.log
tail -f /var/log/xivo-confd.log
consul monitor
consul members -wan
consul-cli agent services --ssl --ssl-verify=false
rabbitmqctl eval 'rabbit_federation_status:status().'
```

What's next?

One you get this part working, check out *Phonebook Sharing*.

1.12.2 Phonebook Sharing

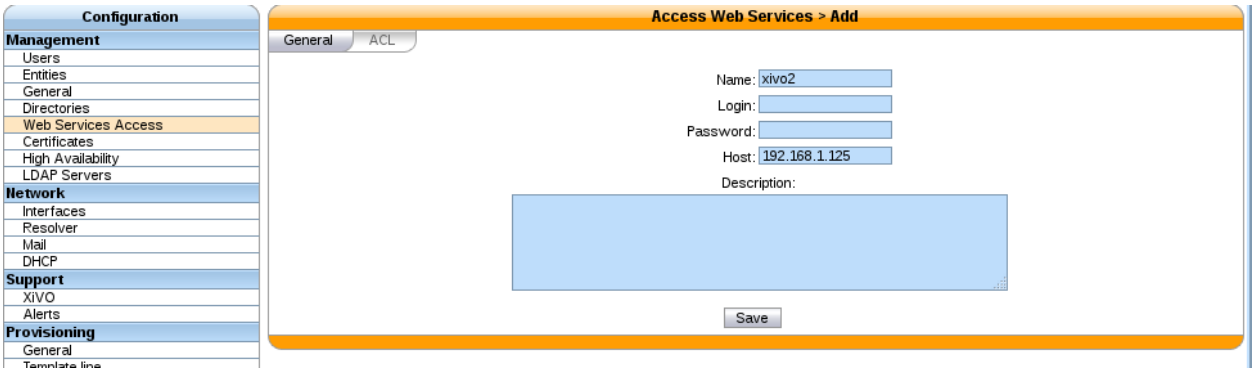
Sharing phonebooks allows users of different Wazo servers to access the contacts in the *phonebooks* of the other Wazo servers.



This procedure follows the *Contact and Presence Sharing* (but it's not mandatory), so we will use the same conventions.

Open Phonebook Access

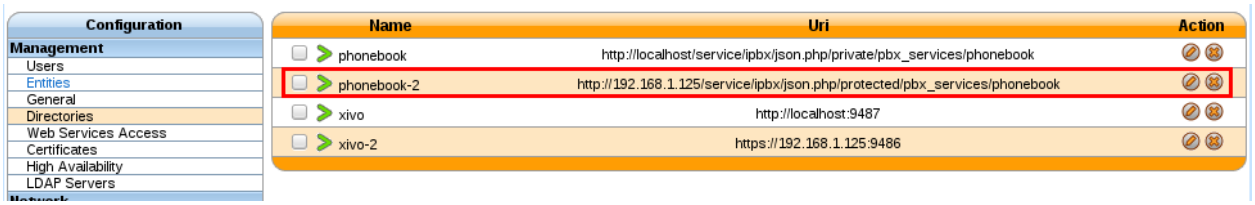
On each Wazo, you must have a *Web Services User* that authorizes access from another host (not by login/password). The phonebook access does not support login/password authorization.



This Web Services user will allow other Wazo servers to access the phonebook of this Wazo.

Configuring the Directories

For each remote Wazo a new phonebook has to be created in *Configuration* → *Management* → *Directories*





Note that the URL of the directory must contain `restricted`, not `private`, e.g:

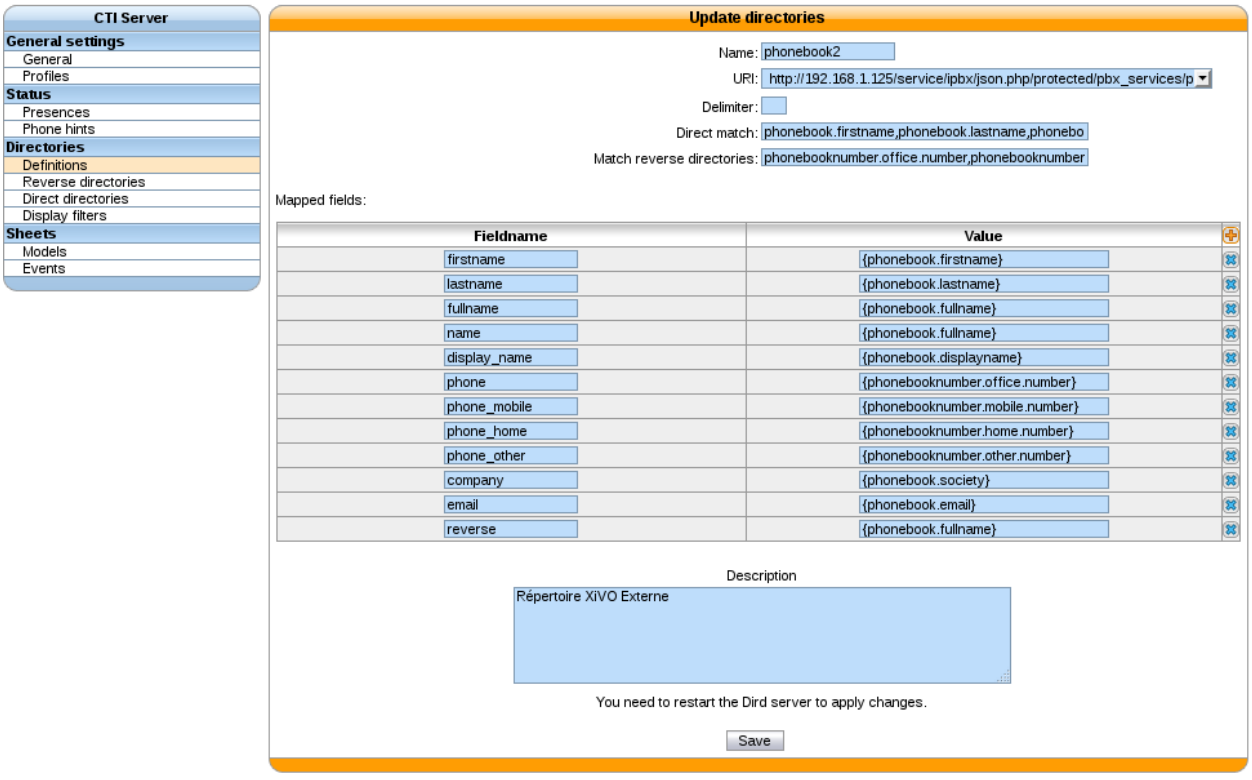
```
http://192.168.1.125/service/ipbx/json.php/restricted/pbx_services/phonebook
```

Add a Directory Definition for Each Phonebook

To add a new directory definition, go to *Services* → *CTI Server* → *Directories* → *Definitions*

In each directory definition, add the fields to match the other phonebooks:

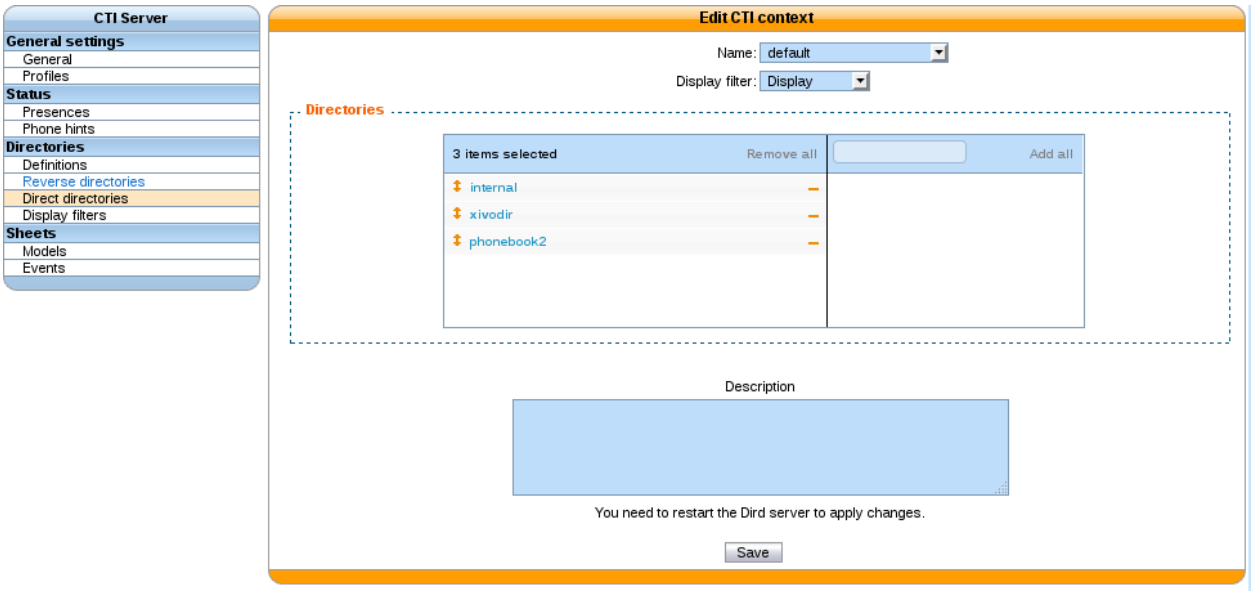

CTI Server
General settings
General
Profiles
Status
Presences
Phone hints
Directories
Definitions
Reverse directories
Name
Description
URI
Action
internal
http://localhost:9487
xivodir
http://localhost/service/ipbx/json.php/private/pbx_services/phonebook
phonebook2
http://192.168.1.125/service/ipbx/json.php/protected/pbx_services/phonebook


CTI Server
General settings
General
Profiles
Status
Presences
Phone hints
Directories
Definitions
Reverse directories
Direct directories
Display filters
Sheets
Models
Events
Update directories
Name: phonebook2
URI: http://192.168.1.125/service/ipbx/json.php/protected/pbx_services/p
Delimiter:
Direct match: phonebook.firstname,phonebook.lastname,phonebo
Match reverse directories: phonebooknumber.office.number,phonebooknumber
Mapped fields:
Fieldname
Value
firstname {phonebook.firstname}
lastname {phonebook.lastname}
fullname {phonebook.fullname}
name {phonebook.fullname}
display_name {phonebook.displayname}
phone {phonebooknumber.office.number}
phone_mobile {phonebooknumber.mobile.number}
phone_home {phonebooknumber.home.number}
phone_other {phonebooknumber.other.number}
company {phonebook.society}
email {phonebook.email}
reverse {phonebook.fullname}
Description
Répertoire XiVO Externe
You need to restart the Dird server to apply changes.
Save

Add the New Definitions to Your Users

We just defined the directories, now let's use them:





Check That Everything is Working

There is no further configuration needed, you should now be able to connect your Wazo Client and search phonebook contacts from the People Xlet.

Troubleshooting

Chances are that everything won't work the first time, here are some interesting commands to help you debug the problem.

```
tail -f /var/log/xivo-dird.log
tail -f /var/log/nginx/xivo.access.log
```

1.13 API and SDK

1.13.1 Message Bus

The message bus is used to receive events from Wazo. It is provided by an AMQP 0-9-1 broker (namely, RabbitMQ) that is integrated in Wazo.

Usage

Websocket

The easiest way to listen for events is to use the *Wazo WebSocket*.

Direct AMQP connection

At the moment, the AMQP broker only listen on the 127.0.0.1 address. This means that if you want to connect to the AMQP broker from a distant machine, you must modify the RabbitMQ server configuration, which is not yet an officially supported operation. All events are sent to the *xivo* exchange.

Otherwise, the default connection information is:

- Virtual host: /
- User name: guest
- User password: guest
- Port: 5672
- Exchange name: xivo
- Exchange type: topic

Example

Here's an example of a simple client, in python, listening for the *call_form_result* CTI events:

```
import kombu

from kombu.mixins import ConsumerMixin

EXCHANGE = kombu.Exchange('xivo', type='topic')
ROUTING_KEY = 'call_form_result'


class C(ConsumerMixin):

    def __init__(self, connection):
        self.connection = connection

    def get_consumers(self, Consumer, channel):
        return [Consumer(kombu.Queue(exchange=EXCHANGE, routing_key=ROUTING_KEY),
                callbacks=[self.on_message])]

    def on_message(self, body, message):
```

```
        print('Received:', body)
        message.ack()


def main():
    with kombu.Connection('amqp://guest:guest@localhost:5672//') as conn:
        try:
            C(conn).run()
        except KeyboardInterrupt:
            return


main()
```

If you are new to AMQP, you might want to look at the RabbitMQ tutorial.

Notes

Things to be aware when writing a client/consumer:

- The published messages are not persistent. When the AMQP broker stops, the messages that are still in queues will be lost.

Changelog

18.02

- The following message has been added:
  - *auth_user_external_auth_authorized*

17.17

- The following messages have been added:
  - *auth_user_external_auth_added*
  - *auth_user_external_auth_deleted*

17.16

- The following messages have been added:
  - *relocate_initiated*
  - *relocate_answered*
  - *relocate_completed*
  - *relocate_ended*

17.14

- The *chat_message_sent* bus message has been added.
- The *chat_message_received* bus message has been added.
- The *chat_message_event* bus message has been deprecated.

17.08

- The *plugin_install_progress* bus message has been added.
- The *plugin_uninstall_progress* bus message has been added.

17.01

- The *favorite_added* bus message has been added.
- The *favorite_deleted* bus message has been added.

16.08

- The *call_held* bus message has been added.
- The *call_resumed* bus message has been added.
- The *user_status_update* bus message now uses the user's UUID instead of the user's ID.

16.07

- The *user_created* bus message has been added.
- The *user_edited* bus message has been added.
- The *user_deleted* bus message has been added.

15.20

- The *chat_message_event* bus message has been added.

15.17

- The *service_registered_event* and *service_deregistered_event* bus messages have been added.

Events

Events that are sent to the bus use a JSON serialization format with the content-type *application/json*. For example, the CTI call_form_result event looks like this:

```
{"name": "call_form_result",
 "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
 "data": {...}}
```

All events have the same basic structure, namely, a JSON object with 4 keys:

name A string representing the name of the event. Each event type has a unique name.

required_acl (optional) Either a string or null. Currently used by xivo-websocketd to determine if a client can receive the event or not. See the *Events Access Control* section for more information.

origin_uuid The uuid to identify the message producer.

data The data specific part of the event. This is documented on a per event type; if not this is assumed to be null.

AMI events

All AMI events are broadcasted on the bus.

- routing key: ami.<event name>
- event specific data: a dictionary with the content of the AMI event

Example event with binding key QueueMemberStatus:

```
{
    "name": "QueueMemberStatus",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "Status": "1",
        "Penalty": "0",
        "CallsTaken": "0",
        "Skills": "",
        "MemberName": "sip\/m3ylhs",
        "Queue": "petak",
        "LastCall": "0",
        "Membership": "static",
        "Location": "sip\/m3ylhs",
        "Privilege": "agent,all",
        "Paused": "0",
        "StateInterface": "sip\/m4ylhs"
    }
}
```

auth_user_external_auth_added

This event is sent when a user adds an external authentication to its account.

- routing_key: auth.users.{user_uuid}.external.{external_auth_name}.created
- event specific data:
  - user_uuid: The user's UUID
  - external_auth_name: The name of the external service

Example:

```
{
  "name": "auth_user_external_auth_added",
  "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
  "data": {
    "user_uuid": "a1e05585-1421-4397-bd59-9cf9725888e9",
    "external_auth_name": "zoho"
  }
}
```

auth_user_external_auth_authorized

This event is sent when a user authorizes an oauth2 request on an external authentication plugin.

- routing_key: auth.users.{user_uuid}.external.{external_auth_name}.authorized
- event specific data:
  - user_uuid: The user's UUID
  - external_auth_name: The name of the external service

Example:

```
{
  "name": "auth_user_external_auth_authorized",
  "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
  "data": {
    "user_uuid": "a1e05585-1421-4397-bd59-9cf9725888e9",
    "external_auth_name": "zoho"
  }
}
```

auth_user_external_auth_deleted

This event is sent when a user removes an external authentication from its account.

- routing_key: auth.users.{user_uuid}.external.{external_auth_name}.deleted
- event specific data:
  - user_uuid: The user's UUID
  - external_auth_name: The name of the external service

Example:

```
{
  "name": "auth_user_external_auth_deleted",
  "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
  "data": {
    "user_uuid": "a1e05585-1421-4397-bd59-9cf9725888e9",
    "external_auth_name": "zoho"
  }
}
```

call_form_result

The call_form_result event is sent when a *custom call form* is submitted by a CTI client.

- routing key: call_form_result
- event specific data: a dictionary with 2 keys:
  - user_id: an integer corresponding to the user ID of the client who saved the call form
  - variables: a dictionary holding the content of the form

Example:

```
{
    "name": "call_form_result",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "user_id": 40,
        "variables": {
            "firstname": "John",
            "lastname": "Doe"
        }
    }
}
```

agent_status_update

The agent_status_update is sent when an agent is logged in or logged out.

- routing key: status.agent
- required ACL: events.statuses.agents
- event specific data: a dictionary with 3 keys:
  - agent_id: an integer corresponding to the agent ID of the agent who's status changed
  - status: a string identifying the status
  - xivo_id: the uuid of the xivo

Example:

```
{
    "name": "agent_status_update",
    "required_acl": "events.statuses.agents",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "agent_id": 42,
        "xivo_id": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
        "status": "logged_in"
    }
}
```

call_created, call_updated, call_ended

The events `call_created`, `call_updated`, `call_ended` are sent when a call handled by xivo-ctid-ng is received, connected or hung up.

- routing key: calls.call.created, calls.call.updated, calls.call.ended
- required ACL: events.calls.<user_uuid>
- event specific data: a dictionary with the same fields as the REST API model of Call (See http://api.wazo.community, section xivo-ctid-ng)

Example:

```
{
    "name": "call_created",
    "required_acl": "events.calls.2e752722-0864-4665-887d-a78a024cf7c7",
    "origin_uuid": "08c56466-8f29-45c7-9856-92bf1ba89b82",
    "data": {
        "bridges": [],
        "call_id": "1455123422.8",
        "caller_id_name": "Some One",
        "caller_id_number": "1001",
        "creation_time": "2016-02-10T11:57:02.592-0500",
        "status": "Ring",
        "talking_to": {},
        "user_uuid": "2e752722-0864-4665-887d-a78a024cf7c7"
    }
}
```

call_held

This message is sent when a call is placed on hold

- routing key: calls.hold.created
- event specific data:
  - call_id: The asterisk channel unique ID

Example:

```
{"name": "call_held",
 "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
 "data": {"call_id": "1465572129.31"}}
```

call_resumed

This message is sent when a call is resumed from hold

- routing key: calls.hold.deleted
- event specific data:
  - call_id: The asterisk channel unique ID

Example:

```
{"name": "call_resumed",
 "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
 "data": {"call_id": "1465572129.31"}}
```

chat_message_received, chat_message_sent

- routing key: `chat.message.<wazo-uuid>.<user_id>`. The `wazo-uuid` and `user-uuid` are the sender for `chat_message_sent` and the recipient for `chat_message_received`.
- event specific data:
  - alias: The nickname of the chatter
  - to: The destination's Wazo UUID and user UUID
  - from: The chatter's Wazo UUID and user UUID
  - msg: The message

Example:

```
{
    "name": "chat_message_received",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "alias": "Alice"
        "to": ["ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3", "fcb36731-c50a-453e-92c7-
↪571297d41616"],
        "from": ["ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3", "4f2e2249-ae2b-4bc2-b5fc-
↪ad42ee01ddaf"],
        "msg": "Hi!"
    }
}
```

Note: The message named `chat_message_event` is deprecated since Wazo 17.14. You should not use it anymore. If you want to send a new chat message, you should use the *xivo-ctid-ng REST API* instead.

endpoint_status_update

The endpoint_status_update is sent when an end point status changes. This information is based on asterisk hints.

- routing key: status.endpoint
- required ACL: events.statuses.endpoints
- event specific data: a dictionary with 3 keys
  - xivo_id: the uuid of the xivo
  - endpoint_id: an integer corresponding to the endpoint ID
  - status: an integer corresponding to the asterisk device state

Example:

```
{
    "name": "endpoint_status_update",
    "required_acl": "events.statuses.endpoints",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "endpoint_id": 67,
        "xivo_id": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
        "status": 0
```

```
    }
}
```

favorite_added

The *favorite_added* event is published when a contact is marked as a favorite by a user.

- routing key: directory.<user_uuid>.favorite.created
- required ACL: events.directory.<user_uuid>.favorite.created
- event specific data:
  - xivo_id: The user's Wazo server UUID
  - user_uuid: The user's UUID
  - source: The source in which this contact can be found
  - source_entry_id: The ID of the contact within this source

Example:

```
{
    "name": "favorite_added",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "xivo_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
        "user_uuid": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2",
        "source": "internal",
        "source_entry_id": 42
    }
}
```

favorite_deleted

The *favorite_deleted* event is published when a favorited contact is marked a not favorite by a user

- routing key: directory.<user_uuid>.favorite.deleted
- required ACL: events.directory.<user_uuid>.favorite.deleted
- event specific data:
  - xivo_id: The user's Wazo server UUID
  - user_uuid: The user's UUID
  - source: The source in which this contact can be found
  - source_entry_id: The ID of the contact within this source

Example:

```
{
    "name": "favorite_deleted",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "xivo_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
        "user_uuid": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2",
```

```
        "source": "internal",
        "source_entry_id": 42
    }
}
```

plugin_install_progress

The *plugin_install_progress* event is published during the installation of a plugin.

- routing key: *plugin.install.<uuid>.<status>*
- required ACL: *events.plugin.install.<uuid>.<status>*
- event specific data:
  - uuid: The installation task UUID
  - status: The status of the installation

Example:

```
{
    "name": "plugin_install_progress",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "uuid": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2",
        "status": "completed"
    }
}
```

plugin_uninstall_progress

The *plugin_uninstall_progress* event is published during the removal of a plugin.

- routing key: *plugin.uninstall.<uuid>.<status>*
- required ACL: *events.plugin.uninstall.<uuid>.<status>*
- event specific data:
  - uuid: The removal task UUID
  - status: The status of the removal

Example:

```
{
    "name": "plugin_uninstall_progress",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "uuid": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2",
        "status": "removing"
    }
}
```

relocate_initiated, relocate_answered, relocate_completed, relocate_ended

Those events are published during the different steps of a relocate operation.

- routing key: `calls.relocate.XXX` where `XXX` is the event, e.g. `calls.relocate.completed`
- headers:
  - `"user_uuid:XXX": True` where `XXX` is the initiator's user UUID
- required ACL: `events.relocates.XXX` where XXX is the initiator's user UUID
- event specific data: a relocate object, see http://api.wazo.community, section `xivo-ctid-ng`.

Example:

```
{
    "name": "relocate_completed",
    "origin_uuid": "cc5d0d76-687e-40a7-81cf-75e0540d1787",
    "data": {
        "uuid": "2fb9efc0-95d3-463b-9042-e2cf2183a303",
        "completions": [
          "answer"
        ],
        "relocated_call": "132456789.1",
        "initiator_call": "132456789.2",
        "recipient_call": "132456789.3",
        "initiator": "b459e3c9-b0a9-43a6-86ff-b4f7d00f6737",
    }
}
```

user_created

The *user_created* event is published when a new user is created.

- routing key: *config.user.created*
- event specific data: a dictionary with 2 keys
  - id: the ID of the created user
  - uuid: the UUID of the created user

Example:

```
{
    "name": "user_created",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "id": 42,
        "uuid": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2"
    }
}
```

user_deleted

The *user_deleted* event is published when a user is deleted.

- routing key: *config.user.deleted*

- event specific data: a dictionary with 2 keys
  - id: the ID of the deleted user
  - uuid: the UUID of the deleted user

Example:

```
{
    "name": "user_deleted",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "id": 42,
        "uuid": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2"
    }
}
```

user_edited

The *user_edited* event is published when a user is modified.

- routing key: *config.user.edited*
- event specific data: a dictionary with 2 keys
  - id: the ID of the modified user
  - uuid: the UUID of the modified user

Example:

```
{
    "name": "user_edited",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "id": 42,
        "uuid": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2"
    }
}
```

user_status_update

The user_status_update is sent when a user changes his CTI presence using the Wazo Client.

- routing key: status.user
- required ACL: events.statuses.users
- event specific data: a dictionary with 3 keys
  - xivo_id: the uuid of the xivo
  - user_uuid: the user's UUID
  - status: a string identifying the status

Example:

```
{
    "name": "user_status_update",
    "required_acl": "events.statuses.users",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "user_uuid": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2",
        "xivo_id": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
        "status": "busy"
    }
}
```

users_forwards_<forward_name>_updated

The users_forwards_<forward_name>_updated is sent when a user changes his forward using REST API.

- forward_name:
  - busy
  - noanswer
  - unconditional
- routing key: config.users.<user_uuid>.forwards.<forward_name>.updated
- required ACL: events.config.users.<user_uuid>.forwards.<forward_name>.updated
- event specific data: a dictionary with 3 keys
  - user_uuid: the user uuid
  - enabled: the state of the forward
  - destination: the destination of the forward

Example:

```
{
    "name": "users_forwards_busy_updated",
    "required_acl": "events.config.users.a1223fe6-bff8-4fb6-a982-f9157dea5094.
↪forwards.busy.updated",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "user_uuid": "a1223fe6-bff8-4fb6-a982-f9157dea5094",
        "enabled": true
        "destination": "1234"
    }
}
```

users_services_<service_name>_updated

The users_services_<service_name>_updated is sent when a user changes his service using REST API.

- service_name:
  - dnd
  - incallfilter
- routing key: config.users.<user_uuid>.services.<service_name>.updated

- required ACL: events.config.users.<user_uuid>.services.<service_name>.updated
- event specific data: a dictionary with 2 keys
  - user_uuid: the user uuid
  - enabled: the state of the service

Example:

```
{
    "name": "users_services_dnd_updated",
    "required_acl": "events.config.users.a1223fe6-bff8-4fb6-a982-f9157dea5094.
↪services.dnd.updated",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "user_uuid": "a1223fe6-bff8-4fb6-a982-f9157dea5094",
        "enabled": true
    }
}
```

service_registered_event

The service_registered_event is sent when a service is started.

- routing key: service.registered.<service_name>
- event specific data: a dictionary with 5 keys
  - service_name: The name of the started service
  - service_id: The consul ID of the started service
  - address: The advertised address of the started service
  - port: The advertised port of the started service
  - tags: The advertised Consul tags of the started service

Example:

```
{
    "name": "service_registered_event",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "service_name": "xivo-ctid",
        "service_id": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2",
        "address": "192.168.1.42",
        "port": 9495,
        "tags": ["xivo-ctid", "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3", "Québec"]
    }
}
```

service_deregistered_event

The service_deregistered_event is sent when a service is stopped.

- routing key: service.deregistered.<service_name>
- event specific data: a dictionary with 3 keys

  - service_name: The name of the stopped service
  - service_id: The consul ID of the stopped service
  - tags: The advertised Consul tags of the stopped service

Example:

```
{
    "name": "service_deregistered_event",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "service_name": "xivo-ctid",
        "service_id": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2",
        "tags": ["xivo-ctid", "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3", "Québec"]
    }
}
```

user_voicemail_message_created

The events `user_voicemail_message_created`, `user_voicemail_message_updated`, `user_voicemail_message_deleted` are sent when a message is left, updated or deleted from a voicemail. A distinct message is generated for each user associated to the voicemail: if the voicemail is not associated to any user, no message is generated.

- routing key: voicemails.messages.created, voicemails.messages.updated, voicemails.messages.deleted
- required ACL: events.users.<user_uuid>.voicemails
- event specific data: a dictionary with the same fields as the REST API model of VoicemailMessage (See http://api.wazo.community, section xivo-ctid-ng)

Example:

```
{
    "name": "user_voicemail_message_created",
    "required_acl": "events.users.8a709eb7-897f-4183-aa3b-ffa2a74e7e37.voicemails",
    "origin_uuid": "3b13295f-9f93-4c19-bd52-015a928a8a2a",
    "data": {
        "voicemail_id": 1,
        "message": {
            "timestamp": 1479226725,
            "caller_id_num": "1001",
            "caller_id_name": "Alice",
            "duration": 0,
            "folder": {
                "type": "new",
                "id": 1,
                "name": "inbox"
            },
            "id": "1479226725-00000003"
        },
        "user_uuid": "8a709eb7-897f-4183-aa3b-ffa2a74e7e37",
        "message_id": "1479226725-00000003"
    }
}
```

1.13.2 Queue logs

Queue logs are events logged by Asterisk in the queue_log table of the asterisk database. Queue logs are used to generate Wazo call center statistics.

Queue log sample

Agent callback login

```
           time             |      callid      | queuename |   agent    |        event
↪       |        data1        |      data2       |     data3     | data4 | data5
--------------------------+------------------+-----------+------------+---------------
↪------+---------------------+------------------+---------------+-------+-------
2012-07-03 15:27:23.896208 | 1341343640.4     | NONE      | Agent/3001 |
↪AGENTCALLBACKLOGIN  | 1002@pcm-dev        |                  |               |
↪|
```

Agent callback logoff

Agent/3001 is logged in queues q1 and q2.

```
           time             |      callid      | queuename |   agent    |        event
↪       |        data1        |      data2       |     data3     | data4 | data5
--------------------------+------------------+-----------+------------+---------------
↪------+---------------------+------------------+---------------+-------+-------
2012-07-03 15:28:07.348244 | NONE             | q2        | Agent/3001 | UNPAUSE
↪       |                     |                  |               |       |
2012-07-03 15:28:07.346320 | NONE             | q1        | Agent/3001 | UNPAUSE
↪       |                     |                  |               |       |
2012-07-03 15:28:07.327425 | NONE             | NONE      | Agent/3001 | UNPAUSEALL
↪       |                     |                  |               |       |
2012-07-03 15:28:06.249357 | NONE             | NONE      | Agent/3001 |
↪AGENTCALLBACKLOGOFF | 1002@pcm-dev        | 43               | CommandLogoff |
↪|
```

Call on a Queue with join empty conditions met

```
           time             |      callid      | queuename |   agent    |        event
↪       |        data1        |      data2       |     data3     | data4 | data5
--------------------------+------------------+-----------+------------+---------------
↪------+---------------------+------------------+---------------+-------+-------
2012-07-04 07:27:55.640421 | 1341401275.9     | q1        | NONE       | JOINEMPTY
↪       |                     |                  |               |       |
```

Enter the queue and get answered by an agent

```
           time             |      callid      | queuename |   agent    |        event
↪       |        data1        |      data2       |     data3     | data4 | data5
--------------------------+------------------+-----------+------------+---------------
↪------+---------------------+------------------+---------------+-------+-------
```

```
2012-07-04 07:33:23.085718 | 1341401601.24    | q1         | Agent/3001 | CONNECT
↪       | 2                    | 1341401601.27    | 1             |        |
2012-07-04 07:33:21.165823 | 1341401601.24    | q1         | NONE       | ENTERQUEUE
↪       |                      | 1000             | 1             |        |
```

Agent or caller ends the call after 12 seconds

```
           time             |      callid      | queuename |   agent    |        event
↪       |        data1         |      data2       |     data3      | data4 | data5
---------------------------+------------------+-----------+------------+---------------
↪------+----------------------+------------------+---------------+-------+-------
2012-07-04 07:37:46.601754 | 1341401851.34    | q1         | Agent/3001 | COMPLETEAGENT
↪       | 2                    | 12               | 1             |        |
```

Call on a full queue

```
           time             |      callid      | queuename |   agent    |        event
↪       |        data1         |      data2       |     data3      | data4 | data5
---------------------------+------------------+-----------+------------+---------------
↪------+----------------------+------------------+---------------+-------+-------
2012-07-04 07:40:17.339945 | 1341402016.44    | q1         | NONE       | FULL
↪       |                      |                  |               |        |
```

Call on a closed queue

```
           time             |      callid      | queuename |   agent    |        event
↪       |        data1         |      data2       |     data3      | data4 | data5
---------------------------+------------------+-----------+------------+---------------
↪------+----------------------+------------------+---------------+-------+-------
2012-07-04 07:48:03.455999 | 1341402482.49    | q1         | NONE       | CLOSED
↪       |                      |                  |               |        |
```

Caller abandon before an answer

```
           time             |      callid      | queuename |   agent    |        event
↪       |        data1         |      data2       |     data3      | data4 | data5
---------------------------+------------------+-----------+------------+---------------
↪-------+----------------------+------------------+---------------+-------+-------
 2012-07-04 07:49:52.939802 | 1341402586.51    | q1         | NONE       | ABANDON
↪       | 1                    | 1                | 6             |        |
```

1.13.3 REST API

The Wazo REST APIs are the privileged way to programmatically interact with Wazo.

REST API Quickstart

Introduction

Wazo REST APIs are HTTP interfaces that allow you to programmatically interact with Wazo. In order to access the REST APIs of Wazo, you need:

- a Wazo server up and running
- a browser
- somewhere you can copy-paste text (ids, tokens, etc.)

REST API Permissions

First of all, you must have permission to use the REST API. In your Wazo web interface, go to *Configuration* → *Management* → *Web Services Access* and create a new user:

- Name: `rest-api-test`
- Login: `rest-api-test`
- Password: some secret password
- Host: nothing
- ACL tab: add a line containing only `#`. `#` is a wildcard that gives access to every REST API. You may want to delete this account when you're done, to reduce risks of unauthorized access.

Save the form, and store the login/password somewhere for later use.

Swagger UI

In this article, we will use the Swagger Web UI, a small web application available in every Wazo installation since XiVO 15.10.

In your browser, go to `http://<wazo>/api`. You should see:

- a list of available APIs
- input boxes on the top, we will ignore those for now

The list of available APIs reflects the different modules of Wazo. Each module is a Python process that serves its own REST API. We will concentrate on two of them:

- wazo-auth
- xivo-confd

wazo-auth is the daemon responsible for authentication. Every API is protected by a token-based authentication mechanism. In order to use any REST API, we will need a valid authentication token, obtained from wazo-auth.

xivo-confd is the daemon responsible for Wazo configuration. Its REST API allows you to read and modify users, lines, extensions, groups, etc. This is the programatic equivalent of the Wazo web interface. However, the xivo-confd REST API is not yet complete, and not all aspects of Wazo configuration are available in xivo-confd.

HTTPS certificates

Almost all REST APIs use encryption and are available via HTTPS. Unfortunately, Wazo does not come with a trusted certificate. So you have to manually trust the self-signed certificate of your Wazo. To that end:

1. Click on wazo-auth in the menu on the left.
2. You should see an error like:

   ```
   Can't read from server. It may not have the appropriate access-control-origin␣
   ↪settings.
   ```

   This is expected. This is the kind of error (quite misleading, admittedly) you get when the certificate is not trusted.
3. Copy the URL you see in the text box at the top of the page, something like: `https://wazo:9497/1.1/api/api.yml` and paste it in your browser.
4. Accept the HTTPS certificate validation exception.
5. You should see a YAML text file describing the xivo-confd API.
6. Go back to `http://wazo/api`.
7. Click on wazo-auth again.
8. Now you should see a list of sections for the wazo-auth REST API, like `backends` or `token`
9. Repeat the whole procedure for xivo-confd (the port in the URL will be different, and the REST API description will take longer to load), and you should be ready to go.

Authentication token

Let's ask wazo-auth for an authentication token:

1. Choose the `wazo-auth` service in the list of REST APIs
2. In the top-right text box of the page (left to the "Explore" button), fill "token" with the `rest-api-test:password`: those credentials are the ones from the Web Services Access you created earlier.
3. Go to the `POST /tokens` section and click on the yellow box to the right of the `body` parameter. This will pre-fill the `body` parameter.
4. In the `body` parameter, set:
   - `backend` to `xivo_service`
   - `expiration` to the number of seconds for the token to be valid (e.g. 3600 for one hour). After the expiration time, you will need to re-authenticate to get a new token.
5. Click `Try it out` at the end of the section. This will make an HTTP request to wazo-auth.
6. You should see a response to your HTTP request, containing a JSON object. In the response, you should see a `token` attribute. That little string is your authentication token. Save it somewhere, in case you need it later.
7. Copy-paste the `token` attribute in the top-right input box, replacing the `rest-api-test:password`. Note that you don't need to click the Explore button to accept the change of token.

Use the xivo-confd REST API

Now that we have an authentication token, we are ready to use the REST API.

1. Click on xivo-confd in the left menu
2. Choose a REST API endpoint, like *users* → *GET /users* and click `Try it out`

And that's it, you are ready to use any REST API with your authentication token.

Note: Be aware that this token will expire, and that you will need to get a new one when that happens. You can take a look at https://auth.wazo.community for an easier manual token generation process. Note that the `auth.wazo.community` server will never know the tokens that you generate, you browser will ask your Wazo directly.

Warning: Also, note that this authentication token gives **all permissions** to anyone who knows it. Same goes for the account password we created earlier. Remember to delete this account, or at least restrict permissions when you're done.

What's next

- Check our *REST API Examples* for more elaborate examples of how to use the REST APIs of Wazo.
- *REST API Conventions* are also a good read
- Explore the REST API in Swagger, it also serves as the reference documentation for REST API.

Something went wrong...

Check *REST API Troubleshooting*.

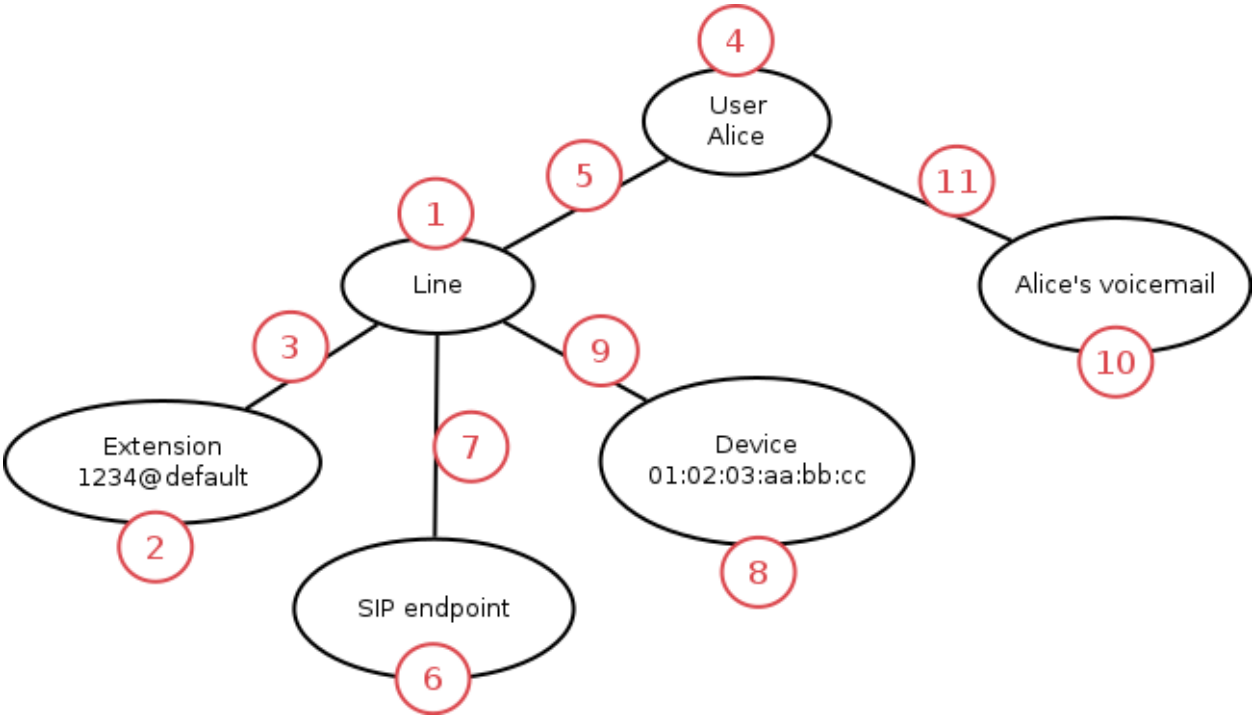
REST API Examples

CURL Examples (xivo-confd)

```
# Get the list of users
curl --insecure \
-H 'Accept: application/json' \
-H 'X-Auth-Token: 17496bfa-4653-9d9d-92aa-17def0fa9826' \
https://wazo:9486/1.1/users

# Create a user
# When sending data, you need the Content-Type header.
curl --insecure \
-X POST \
-d '{"firstname": "hello-world"} \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-H 'X-Auth-Token: 17496bfa-4653-9d9d-92aa-17def0fa9826' \
https://wazo:9486/1.1/users
```

Create a user with a phone and a voicemail (xivo-confd)



1. Create a line:

```
POST /lines/sip
{
  "context": "default"
}
```

Response:

```
{
  "id": 11
  ...
}
```

2. Create an extension:

```
POST /extensions
{
  "exten": "1234",
  "context": "default"
}
```

Response:

```
{
  "id": 22
  ...
}
```

3. Associate the line-extension:

```
PUT /lines/11/extensions/22
```

4. Create a user:

```
POST /users
{
  "firstname": "Alice"
}
```

Response:

```
{
  "uuid": "44444444-4444-4444-4444-444444444444"
  ...
}
```

5. Associate the user-line:

```
PUT /users/44444444-4444-4444-4444-444444444444/lines/11
```

6. Create the SIP endpoint:

```
POST /endpoints/sip
{
}
```

Response:

```
{
  "id": 66
  ...
}
```

7. Associate the line-endpoint:

```
PUT /lines/11/endpoints/sip/66
```

8. Create the device. This is usually done automatically when the device is plugged in and put in autoprov mode. However, you need to get the device ID:

```
GET /devices?search=88:88:88:88:88:88  or  GET /devices?search=192.168.88.88
{
  "id": "88888888888888888888888888888888",
  ...
}
```

9. Associate the line-device:

```
PUT /lines/11/devices/88888888888888888888888888888888
```

You may also want to re-synchronize the device:

```
PUT /devices/88888888888888888888888888888888/synchronize
```

10. Create the voicemail:

```
POST /voicemails
{
  "name": "Alice's voicemail",
  "number": "1234"
  "context": "default"
}
```

Response:

```
{
  "id": 1010
  ...
}
```

11. Associate the user-voicemail:

```
PUT /users/44444444-4444-4444-4444-444444444444/voicemails/1010
```

REST API Troubleshooting

Here is a list of common problems you can encounter with Wazo REST APIs.

Swagger UI: Can't read from server...

Problem

When trying to access Swagger UI via `http://wazo/api`, I get:

```
Can't read from server. It may not have the appropriate access-control-origin␣
↪settings.
```

Answer

This is a very generic error message from Swagger UI. It can have a variety of causes, most commonly:

- the HTTPS certificate of the API you're trying to get is not trusted
- the daemon that serves the API is not running

What you can do:

- check that the Swagger API spec is accessible: when choosing an API in the Swagger menu, copy-paste the URL of the top text box ending with `api.yml` into your browser.
- check the HTTP requests/answers in your browser debugging tools
- check that the daemon is running: in a console, type: `wazo-service status`
- check the log files of the daemon in `/var/log/<daemon>.log` (see also: *Log Files*)

REST API Reference

Access

Each REST API is available via HTTPS on *different ports*.

Most of them can also be reached by default via *Nginx* using the port TCP/443.

API reference

wazo-webhookd REST API

API reference

API documentation is available on http://api.wazo.community.

More specific documentation:

Filtering webhook events on user

When configuring a webhook, you can set the `user_uuid` parameter. Doing so makes the webhook being only triggered when events are related to the specified user.

For example, given a webhook on event `user_status_update`, and `user_uuid` is set to user A, the webhook will only be triggered when user A changes its presence status, not when user B does.

Supported events

The current list of events that is supported by the `user_uuid` parameter is:

- `agent_paused`
- `agent_status_update`
- `agent_unpaused`
- `call_created`
- `call_ended`
- `call_log_user_created`
- `call_updated`
- `chat_message_sent`
- `chat_message_received`
- `endpoint_status_update`
- `favorite_added`
- `favorite_deleted`
- `relocate_initiated`
- `relocate_answered`
- `relocate_completed`

- relocate_ended
- user_status_update
- user_voicemail_message_created
- user_voicemail_message_deleted
- user_voicemail_message_updated
- users_forwards_busy_updated
- users_forwards_noanswer_updated
- users_forwards_unconditional_updated
- users_services_dnd_updated
- users_services_incallfilter_updated

Unsupported events will always trigger the webhook, regardless of the related user.

wazo-webhookd HTTP templates

When creating a webhook (i.e. a subscription), you can customize parts of the HTTP request that will be triggered. For this, subscriptions are defined using a templating "language", that indicates where to use variables that will be replaced with event data.

Templates use the Jinja2 syntax. See the Jinja documentation for more details.

The following parts of the request are templated:

- the request's URL
- the request's body

Example

Given a subscription:

```
{
  "name": "Hello subscription",
  "service": "http",
  "events": [
    "hello"
  ],
  "config": {
    "content_type": "text/plain",
    "method": "POST",
    "url": "https://example.com/event_handler?v=1.0",
    "verify_certificate": "true",
    "body": "I just received an event named {{ event_name }},
             from the Wazo server {{ wazo_uuid }}.
             The event contained the following data:
             hello = \"{{ event['hello'] }}\",
             bye = \"{{ event['bye'] }}\"."
  }
}
```

When an event is emitted:

```
{
  "name": "hello_event",
  "origin_uuid": "my-wazo",
  "data": {
    "hello": "world",
    "bye": "bye"
  }
}
```

Then a HTTP request is sent to https://example.com:

```
POST /event_handler?v=1.0
Content-Type: text/plain

I just received an event named hello_event,
from the Wazo server my-wazo.
The event contained the following data:
hello = "world",
bye = "bye".
```

Reference

Available variables:

- `event_name`: the name of the event.
- `wazo_uuid`: the UUID of the Wazo server who sent the event.
- `event`: the body of the event. Details may be accessed like: `event['detail']`. Further nested details may be accessed like: `event['detail']['subdetail']`.

Tips

Query string

If you want to create a query string from an event, you can use Jinja's builtin filter feature:

The template:

```
https://example.com/query?{{ event|urlencode }}
```

gives an URL:

```
https://example.com/query?key1=value1&key2=value2
```

when triggered with an event:

```
{"key1": "value1",
 "key2": "value2"}
```

xivo-confd REST API

API reference

API documentation is available on http://api.wazo.community. This section contains extended documentation for certain aspects of the API.

Function Keys

Function keys can be used as shortcuts for dialing a number, or accomplishing other menial tasks, by pushing a button on the phone. A function key's action is determined by its destination.

Function keys can be added directly on a user, or in a template. Templates are useful for creating a set of common function keys that can be used by the same group of people.

This page only describes the data models used by the REST API. Consult the API documentation for further details on URLs.

Function Key Template

Parameters

Field	Type	Required	Description
name	string	No	A name for the template.
keys	*Function Key*	No	A collection of function keys under the form `{"position": "funckey"}`. See the example for more details.

Example

```
{
    "name": "Example template",
    "keys": {
         "1": {
            "destination": {
                 "type": "user",
                 "user_id": 34
             }
         },
         "2": {
             "blf": true,
             "label": "Call mom",
             "destination": {
                 "type": "custom",
                 "exten": "5551234567"
             }
         }
     }
 }
```

Function Key

Description

Field	Type	Required	Description
blf	boolean	No	Turn on BLF when there is activity on the destination
label	string	No	Label to display next to the function key
destination	*Destination*	Yes	Destination to call

Example

```
{
    "blf": True,
    "label": "Call john",
    "destination": {
         "type": "user",
         "user_id": 34
     }
 }
```

Destination

A destination determines the number to dial when using a function key. Destinations are composed of a parameter named `type` and any additional parameters required by its type.

Available destination types:

agent An agent

bsfilter Boss/Secretary filter

conference Conference room

custom A custom number to dial

forward Forward a call towards another number

group A group

onlinerec Record a conversation during a call

paging A paging

park Park a call

park_position Pick up a parked call

queue Call queue

service A call service

transfer Transfer a call

user A User

Here are the parameters required for each destination:

Agent

Field	Type	Value
agent_id	numeric	Agents's id

BSFilter

Field	Type	Value
filter_member_id	numeric	ID of the filter member

Conference

Field	Type	Value
conference_id	numeric	Conference's id

Custom

Field	Type	Value
exten	string	Number to dial

Forward

Field	Type	Value
forward	string	Type of forward. Possible values: busy, noanswer, unconditional
exten	string	Number to dial (optional)

Group

Field	Type	Value
group_id	numeric	Group's id

Online call recording

No parameters are required for this destination

Paging

Field	Type	Value
paging_id	numeric	Pagings's id

Parking

No parameters are required for this destination

Parking Position

Field	Type	Value
position	numeric string	Position of the parking to pick up

Queue

Field	Type	Value
queue_id	numeric	User's id

Service

Field	Type	Value
service	string	Name of the service

Currently supported services:

phonestatus Phone Status

recsnd Sound Recording

callrecord Call recording

incallfilter Incoming call filtering

enablednd Enable "Do not disturb" mode

pickup Group Interception

calllistening Listen to online calls

directoryaccess Directory access

fwdundoall Disable all forwaring

enablevm Enable Voicemail

vmusermsg Consult the Voicemail

vmuserpurge Delete messages from voicemail

Transfer

Field	Type	Value
transfer	string	Type of transfer. Possible values: blind, attended

User

Field	Type	Value
user_id	numeric	User's id

CSV User Import

Users and common related resources can be imported onto a Wazo server by sending a CSV file with a predefined *set of fields*.

This page only documents additional notes useful for API users. Consult the API documentation for more details.

Uploading files

Files may be uploaded as usual through the web interface, or from a console by using HTTP utilities and the REST API. When uploading through the API, the header *Content-Type: text/csv charset=utf-8* must be set and the CSV data must be sent in the body of the request. A file may be uploaded using *curl* as follows:

```
curl -k -H "Content-Type: text/csv; charset=utf-8" -u username:password --data-binary
↪"@file.csv" https://wazo:9486/1.1/users/import
```

The response can be reindented in a more readable format by piping the output through *python -m json.tool* in the following way:

```
curl (...) | python -m json.tool
```

Migration from 1.0

The API version 1.0 is no longer supported and has been removed. In most cases, code that used the old API can be migrated to version 1.1 without much hassle by updating the URL. For example, in 1.0, the URL to list users was:

```
   /1.0/users/

In 1.1, it is::

   /1.1/users
```

Please note that there are no trailing slashes in URLs for version 1.1.

For further details consult the documentation at http://api.wazo.community

xivo-provd REST API

This section describes the REST API provided by the xivo-provd application.

If you want to interact with the REST API of the xivo-provd daemon that is executing as part of Wazo, you should be careful on which operation you are doing as to not cause stability problem to other parts of the Wazo ecosystem. Mostly, this means being careful when editing or deleting devices and configs.

By default, the REST API of xivo-provd is accessible only from localhost on port 8666. No authentication is required.

Warning: Major changes could happen to this API.

API

The description of the API has been split into these sections:

Provd Management

Get the Provd Manager

The provd manager resource represents the main entry point to the xivo-provd REST API.

It links to the following resources:

- The `dev` relation links to a *device manager*.
- The `cfg` relation links to a *config manager*.
- The `pg` relation links to a *plugin manager*.
- The `srv.configure` relation links to the provd manager *configuration service*.

Query

```
GET /provd
```

Example request

```
GET /provd HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "links": [
        {
            "href": "/provd/dev_mgr",
            "rel": "dev"
        },
        {
            "href": "/provd/cfg_mgr",
            "rel": "cfg"
        },
        {
            "href": "/provd/pg_mgr",
```

```
            "rel": "pg"
        },
        {
            "href": "/provd/configure",
            "rel": "srv.configure"
        }
    ]
}
```

Devices Management

Get the Device Manager

The device manager links to the following resources:

- The `dev.synchronize` relation links to the *device synchronization service*.
- The `dev.reconfigure` relation links to the *device reconfiguration service*.
- The `dev.dhcpinfo` relation links to the *device DHCP information service*.
- The `dev.devices` relation links to the *list of devices*.

Query

```
GET /provd/dev_mgr
```

Example request

```
GET /provd/dev_mgr HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "links": [
        {
            "href": "/provd/dev_mgr/synchronize",
            "rel": "dev.synchronize"
        },
        {
            "href": "/provd/dev_mgr/reconfigure",
            "rel": "dev.reconfigure"
        },
        {
            "href": "/provd/dev_mgr/dhcpinfo",
            "rel": "dev.dhcpinfo"
```

```
        },
        {
            "href": "/provd/dev_mgr/devices",
            "rel": "dev.devices"
        }
    ]
}
```

List Devices

Query

```
GET /provd/dev_mgr/devices
```

Query Parameters

Field	Description
q	A selector, encoded in JSON, describing which device should be returned. All devices are returned if not specified. Example: `q={"ip":"10.34.1.119"}`
fields	A list of fields, separated by comma. Example: `fields=mac,ip`
skip	An integer specifing the number of devices to skip. Example: `skip=10`
sort	The key on which to sort the results. Example: `sort=id`
sort_ord	The order of sort; either `ASC` or `DESC`.

Example request

```
GET /provd/dev_mgr/devices HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "devices": [
        {
            "added": "auto",
            "config": "38e5e08ffe804b468f5aa53b9536bb25",
            "configured": true,
            "description": "",
            "id": "38e5e08ffe804b468f5aa53b9536bb25",
            "ip": "10.34.1.122",
            "mac": "00:08:5d:33:e5:76",
            "model": "6731i",
            "plugin": "xivo-aastra-3.3.1-SP2",
```

```
            "remote_state_sip_username": "je5qtq",
            "vendor": "Aastra",
            "version": "3.3.1.2235"
        }
    ]
}
```

Create a Device

Query

```
POST /provd/dev_mgr/devices
```

Example request

```
POST /provd/dev_mgr/devices HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json

{
    "device": {
        "ip": "192.168.1.1",
        "mac": "00:11:22:33:44:55",
        "plugin": "xivo-aastra-3.3.1-SP2"
    }
}
```

Example response

```
HTTP/1.1 201 Created
Content-Type: application/vnd.proformatique.provd+json
Location: /provd/dev_mgr/devices/68b10c99945b4fb889f22a7559fc3271

{"id": "68b10c99945b4fb889f22a7559fc3271"}
```

If the `id` field is not given, then an ID is automatically generated by the server.

Get a Device

Query

```
GET /provd/dev_mgr/devices/<device_id>
```

Example request

```
GET /provd/dev_mgr/devices/68b10c99945b4fb889f22a7559fc3271 HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "device": {
        "added": "auto",
        "config": "38e5e08ffe804b468f5aa53b9536bb25",
        "configured": true,
        "description": "",
        "id": "38e5e08ffe804b468f5aa53b9536bb25",
        "ip": "10.34.1.122",
        "mac": "00:08:5d:33:e5:76",
        "model": "6731i",
        "plugin": "xivo-aastra-3.3.1-SP2",
        "remote_state_sip_username": "je5qtq",
        "vendor": "Aastra",
        "version": "3.3.1.2235"
    }
}
```

Update a Device

Query

```
PUT /provd/dev_mgr/devices/<device_id>
```

Example request

```
PUT /provd/dev_mgr/devices/68b10c99945b4fb889f22a7559fc3271 HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json

{
    "device": {
        "added": "auto",
        "config": "38e5e08ffe804b468f5aa53b9536bb25",
        "configured": true,
        "description": "",
        "id": "38e5e08ffe804b468f5aa53b9536bb25",
        "ip": "10.34.1.122",
        "mac": "00:08:5d:33:e5:76",
        "model": "6731i",
        "plugin": "xivo-aastra-3.4",
        "remote_state_sip_username": "je5qtq",
        "vendor": "Aastra",
```

```
        "version": "3.3.1.2235"
    }
}
```

Example response

```
HTTP/1.1 204 No Content
```

Delete a Device

Query

```
DELETE /provd/dev_mgr/devices/<device_id>
```

Example request

```
DELETE /provd/dev_mgr/devices/68b10c99945b4fb889f22a7559fc3271 HTTP/1.1
Host: wazoserver
```

Example response

```
HTTP/1.1 204 No Content
```

Synchronize a Device

Query

```
POST /provd/dev_mgr/synchronize
```

Example request

```
POST /provd/dev_mgr/synchronize HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json

{
    "id": "d035bccaf0dd4a8396fc57a3329ca0a4"
}
```

Example response

```
HTTP/1.1 201 Created
Location: /provd/dev_mgr/synchronize/42
```

The URI returned in the `Location` header points to an *operation in progress* resource.

Reconfigure a Device

Query

```
POST /provd/dev_mgr/reconfigure
```

Errors

Error code	Error message	Description
400	invalid device ID	

Example request

```
POST /provd/dev_mgr/reconfigure HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json

{
    "id": "d035bccaf0dd4a8396fc57a3329ca0a4"
}
```

Example response

```
HTTP/1.1 204 No Content
```

Push DHCP Request Information

Query

```
POST /provd/dev_mgr/dhcpinfo
```

Example request

```
POST /provd/dev_mgr/dhcpinfo HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json
```

```
{
    "dhcp_info": {
        "ip": "192.168.1.100",
        "mac": "00:11:22:33:44:55",
        "op": "commit",
        "options": [
            "06066.6f.6f.62.61.72.a"
        ]
    }
}
```

Example response

```
HTTP/1.1 204 No Content
```

Configs Management

Get the Config Manager

The config manager links to the following resources:

- The `cfg.configs` relation links to the *list of configs*.
- The `cfg.autocreate` relation links to the *config autocreate service*.

Query

```
GET /provd/cfg_mgr
```

Example request

```
GET /provd/cfg_mgr HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json


{
    "links": [
        {
            "href": "/provd/cfg_mgr/configs",
            "rel": "cfg.configs"
        },
```

```
        {
            "href": "/provd/cfg_mgr/autocreate",
            "rel": "cfg.autocreate"
        }
    ]
}
```

List Configs

Query

```
GET /provd/cfg_mgr/configs
```

Query Parameters

These are the *same parameters as for the list devices* action.

Example request

```
GET /provd/cfg_mgr/configs HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "configs": [
        {
            "configdevice": "defaultconfigdevice",
            "deletable": true,
            "id": "38e5e08ffe804b468f5aa53b9536bb25",
            "parent_ids": [
                "base",
                "defaultconfigdevice"
            ],
            "raw_config": {
                "X_key": "",
                "exten_dnd": "*25",
                "exten_fwd_busy": "*23",
                "exten_fwd_disable_all": "*20",
                "exten_fwd_no_answer": "*22",
                "exten_fwd_unconditional": "*21",
                "exten_park": null,
                "exten_pickup_call": "*8",
                "exten_pickup_group": null,
                "exten_voicemail": "*98",
```

```
                "funckeys": {
                    "1": {
                        "label": "",
                        "line": 1,
                        "type": "speeddial",
                        "value": "1005"
                    }
                },
                "protocol": "SIP",
                "sip_dtmf_mode": "SIP-INFO",
                "sip_lines": {
                    "1": {
                        "auth_username": "je5qtq",
                        "display_name": "El\u00e8s 01",
                        "number": "1001",
                        "password": "T2S7C0",
                        "proxy_ip": "10.34.1.11",
                        "registrar_ip": "10.34.1.11",
                        "username": "je5qtq"
                    }
                }
            }
        }
    ]
}
```

Create a Config

Query

```
POST /provd/cfg_mgr/configs
```

Example request

```
POST /provd/cfg_mgr/configs HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json

{
    "config": {
        "parent_ids": [
            "base"
        ],
        "raw_config": {
            "sip": {
                "lines": {
                    "1": {
                        "auth_username": "100",
                        "display_name": "Foo",
                        "password": "100",
                        "username": "100"
                    }
                }
```

```
            }
        }
    }
}
```

Example response

```
HTTP/1.1 201 Created
Content-Type: application/vnd.proformatique.provd+json
Location: /provd/cfg_mgr/configs/77839d0f05c84662864b0ae5c27b33e4

{"id": "77839d0f05c84662864b0ae5c27b33e4"}
```

If the `id` field is not given, then an ID id automatically generated by the server.

Get a Config

Query

```
GET /provd/cfg_mgr/configs/<config_id>
```

Example request

```
GET /provd/cfg_mgr/configs/77839d0f05c84662864b0ae5c27b33e4 HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "config": {
        "id": "77839d0f05c84662864b0ae5c27b33e4",
        "parent_ids": [
            "base"
        ],
        "raw_config": {
            "sip": {
                "lines": {
                    "1": {
                        "auth_username": "100",
                        "display_name": "Foo",
                        "password": "100",
                        "username": "100"
                    }
                }
            }
```

```
        }
    }
}
```

Get a Raw Config

Query

```
GET /provd/cfg_mgr/configs/<config_id>/raw
```

Example request

```
GET /provd/cfg_mgr/configs/77839d0f05c84662864b0ae5c27b33e4/raw HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "raw_config": {
        "X_xivo_phonebook_ip": "10.34.1.11",
        "http_port": 8667,
        "ip": "10.34.1.11",
        "ntp_enabled": true,
        "ntp_ip": "10.34.1.11",
        "sip": {
            "lines": {
                "1": {
                    "auth_username": "100",
                    "display_name": "John",
                    "password": "100",
                    "username": "100"
                }
            }
        },
        "tftp_port": 69
    }
}
```

Update a Config

Query

```
PUT /provd/cfg_mgr/configs/<config_id>
```

Example request

```
PUT /provd/cfg_mgr/configs/77839d0f05c84662864b0ae5c27b33e4 HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json

{
    "config": {
        "id": "77839d0f05c84662864b0ae5c27b33e4",
        "parent_ids": [
            "base"
        ],
        "raw_config": {
            "sip": {
                "lines": {
                    "1": {
                        "auth_username": "100",
                        "display_name": "John",
                        "password": "100",
                        "username": "100"
                    }
                }
            }
        }
    }
}
```

Example response

```
HTTP/1.1 204 No Content
```

Delete a Config

Query

```
DELETE /provd/cfg_mgr/configs/<config_id>
```

Example request

```
DELETE /provd/cfg_mgr/configs/77839d0f05c84662864b0ae5c27b33e4
Host: wazoserver
```

Example response

```
HTTP/1.1 204 No Content
```

Autocreate a Config

This service is used to create a new config from the config that has the `autocreate` role.

Query

```
POST /provd/cfg_mgr/autocreate
```

Example request

```
POST /provd/cfg_mgr/autocreate HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json

{}
```

Example response

```
HTTP/1.1 201 Created
Content-Type: application/vnd.proformatique.provd+json
Location: /provd/cfg_mgr/configs/autoprov1411400365

{"id":"autoprov1411400365"}
```

Plugins Management

Get the Plugin Manager

The plugin manager links to the following resources:

- The `srv.install` relation links to the plugin manager *installation service*. This installation service permits installing/uninstalling plugins.
- The `pg.plugins` relation links to the *list of plugins*.
- The `pg.reload` relation links to the *plugin reload service*.

Query

```
GET /provd/pg_mgr
```

Example request

```
GET /provd/pg_mgr HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "links": [
        {
            "href": "/provd/pg_mgr/install",
            "rel": "srv.install"
        },
        {
            "href": "/provd/pg_mgr/plugins",
            "rel": "pg.plugins"
        },
        {
            "href": "/provd/pg_mgr/reload",
            "rel": "pg.reload"
        }
    ]
}
```

List Plugins

List the installed plugins.

If you want to install/uninstall plugins, you need to go trough the plugin installation service.

Query

```
GET /provd/pg_mgr/plugins
```

Example request

```
GET /provd/pg_mgr/plugins HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "plugins": {
        "xivo-aastra-3.3.1-SP2": {
            "links": [
                {
                    "href": "/provd/pg_mgr/plugins/xivo-aastra-3.3.1-SP2",
                    "rel": "pg.plugin"
```

```
                }
            ]
        },
        "xivo-cisco-sccp-9.0.3": {
            "links": [
                {
                    "href": "/provd/pg_mgr/plugins/xivo-cisco-sccp-9.0.3",
                    "rel": "pg.plugin"
                }
            ]
        }
    }
}
```

Get a Plugin

The plugin links to the following resources:

- The `pg.info` relation links to the *plugin information*.
- The `srv.install` relation links to the plugin *installation service*. Plugins usually provided this service to install/uninstall firmware and language files.

Query

```
GET /provd/pg_mgr/plugins/<plugin_id>
```

Example request

```
GET /provd/pg_mgr/plugins/xivo-aastra-3.3.1-SP2 HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "links": [
        {
            "href": "/provd/pg_mgr/plugins/xivo-aastra-3.3.1-SP2/info",
            "rel": "pg.info"
        },
        {
            "href": "/provd/pg_mgr/plugins/xivo-aastra-3.3.1-SP2/install",
            "rel": "srv.install"
        }
    ]
}
```

Get Information of a Plugin

Query

```
GET /provd/pg_mgr/plugins/<plugin_id>/info
```

Example request

```
GET /provd/pg_mgr/plugins/xivo-aastra-3.3.1-SP2/info HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "plugin_info": {
        "capabilities": {
            "Aastra, 6730i, 3.3.1.5089": {
                "sip.lines": 6
            },
            "Aastra, 6731i, 3.3.1.2235": {
                "sip.lines": 6,
                "switchboard": true
            },
            "Aastra, 6735i, 3.3.1.5089": {
                "sip.lines": 9
            },
            "Aastra, 6737i, 3.3.1.5089": {
                "sip.lines": 9
            },
            "Aastra, 6739i, 3.3.1.2235": {
                "sip.lines": 9
            },
            "Aastra, 6753i, 3.3.1.2235": {
                "sip.lines": 9
            },
            "Aastra, 6755i, 3.3.1.2235": {
                "sip.lines": 9,
                "switchboard": true
            },
            "Aastra, 6757i, 3.3.1.2235": {
                "sip.lines": 9,
                "switchboard": true
            },
            "Aastra, 9143i, 3.3.1.2235": {
                "sip.lines": 9
            },
            "Aastra, 9480i, 3.3.1.2235": {
                "sip.lines": 9
            }
```

```
        },
        "description": "Plugin for Aastra 6730i, 6731i, 6735i, 6737i, 6739i, 6753i,␣
↪6755i, 6757i, 6757i CT, 9143i, 9480i, 9480i CT in version 3.3.1 SP2.",
        "version": "1.1"
    }
}
```

Reload a Plugin

Reload the given plugin. This is mostly useful during plugin development, after changing the code of the plugin, instead of restarting the xivo-provd application.

Query

```
POST /provd/pg_mgr/reload
```

Example request

```
POST /provd/pg_mgr/reload HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json

{
    "id": "xivo-aastra-3.3.1-SP2"
}
```

Example response

```
HTTP/1.1 204 No Content
```

General Resources

This section describes the resources that are available from more than one URI or are generic enough to not fit in a more specific section.

Operation In Progress

This resource represents an operation in progress and is used to follow the progress of an underlying operation. Said differently, it is a monitor on an operation that can change over time.

Get Current Status

Query

```
GET <uri>
```

Example request

```
GET <uri> HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "status": "progress"
}
```

The `status` field describe the current status of the operation. The format is `[label|]state[;current[/end]](\(sub_oips\))*`. Here's some examples:

- progress
- download|progress
- download|progress;10
- download|progress;10/100
- download|progress(file_1|progress;20/100)(file_2|waiting;0/50)
- download|progress;20/150(file_1|progress)(file_2|waiting)
- op|progress(op1|progress(op11|progress)(op12|waiting))(op2|progress)

The state of an operation is either `waiting`, `progress`, `success` or `fail`.

Delete

Delete the "operation in progress" resource.

This does not cancel the underlying operation; it only deletes the monitor. Every monitor that is created should be deleted, else they won't be freed by the process and they will accumulate, taking memory.

Query

```
DELETE <uri>
```

Example request

```
DELETE <uri> HTTP/1.1
Host: wazoserver
```

Example response

```
HTTP/1.1 204 No Content
```

Configuration Service

Get the Configuration

Query

```
GET <uri>
```

Example request

Example request for the configuration service of the *provd manager*.

```
GET /provd/configure HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "params": [
        {
            "description": "The plugins repository URL",
            "id": "plugin_server",
            "links": [
                {
                    "href": "/provd/configure/plugin_server",
                    "rel": "srv.configure.param"
                }
            ],
            "value": "http://provd.wazo.community/plugins/1/stable"
        },
        {
            "description": "The proxy for HTTP requests. Format is \"http://
↪[user:password@]host:port\"",
            "id": "http_proxy",
            "links": [
```

```
                {
                    "href": "/provd/configure/http_proxy",
                    "rel": "srv.configure.param"
                }
            ],
            "value": null
        },
        {
            "description": "The proxy for FTP requests. Format is \"http://
↪[user:password@]host:port\"",
            "id": "ftp_proxy",
            "links": [
                {
                    "href": "/provd/configure/ftp_proxy",
                    "rel": "srv.configure.param"
                }
            ],
            "value": null
        },
        {
            "description": "The proxy for HTTPS requests. Format is \"host:port\"",
            "id": "https_proxy",
            "links": [
                {
                    "href": "/provd/configure/https_proxy",
                    "rel": "srv.configure.param"
                }
            ],
            "value": null
        },
        {
            "description": "The current locale. Example: fr_FR",
            "id": "locale",
            "links": [
                {
                    "href": "/provd/configure/locale",
                    "rel": "srv.configure.param"
                }
            ],
            "value": null
        },
        {
            "description": "Set to 1 if all the devices are behind a NAT.",
            "id": "NAT",
            "links": [
                {
                    "href": "/provd/configure/NAT",
                    "rel": "srv.configure.param"
                }
            ],
            "value": 0
        }
    ]
}
```

Get the Value of a Parameter

Query

```
GET <uri>
```

Example request

Example request for the NAT option of the configuration service of the provd entry point.

```
GET /provd/configure/NAT HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "param": {
        "value": 0
    }
}
```

Set the Value of a Parameter

Query

```
PUT <uri>
```

Example request

Example request for the NAT option of the configuration service of the *provd manager*.

```
PUT /provd/configure/NAT HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json

{
    "param": {
        "value": 1
    }
}
```

Example response

```
HTTP/1.1 204 No Content
Content-Type: application/vnd.proformatique.provd+json
```

Installation Service

Get the Installation Service

Query

```
GET <uri>
```

Example request

Example request for the installation service of the *plugin manager*.

```
GET /provd/pg_mgr/install HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "links": [
        {
            "href": "/provd/pg_mgr/install/install",
            "rel": "srv.install.install"
        },
        {
            "href": "/provd/pg_mgr/install/uninstall",
            "rel": "srv.install.uninstall"
        },
        {
            "href": "/provd/pg_mgr/install/installed",
            "rel": "srv.install.installed"
        },
        {
            "href": "/provd/pg_mgr/install/installable",
            "rel": "srv.install.installable"
        },
        {
            "href": "/provd/pg_mgr/install/upgrade",
            "rel": "srv.install.upgrade"
        },
        {
            "href": "/provd/pg_mgr/install/update",
```

```
            "rel": "srv.install.update"
        }
    ]
}
```

The upgrade and update services are optional and not all installation service provide them.

Install a Package

Query

```
POST <uri>
```

Example request

Example request for the installation service of the plugin manager.

```
POST /provd/pg_mgr/install/install HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json

{
    "id": "xivo-polycom-4.0.4"
}
```

Example response

```
HTTP/1.1 201 Created
Location: /provd/pg_mgr/install/install/1
Content-Type: application/vnd.proformatique.provd+json
```

The URI returned in the `Location` header points to an *operation in progress* resource.

Uninstall a Package

Query

```
POST <uri>
```

Example request

Example request for the installation service of the plugin manager.

```
POST /provd/pg_mgr/install/uninstall HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json
```

```
{
    "id": "xivo-polycom-4.0.4"
}
```

Example response

```
HTTP/1.1 204 No Content
Content-Type: application/vnd.proformatique.provd+json
```

Upgrade a Package

Query

```
POST <uri>
```

Example request

Example request for the installation service of the plugin manager.

```
POST /provd/pg_mgr/install/upgrade HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json

{
    "id": "xivo-polycom-4.0.4"
}
```

Example response

```
HTTP/1.1 201 Created
Location: /provd/pg_mgr/install/upgrade/1
Content-Type: application/vnd.proformatique.provd+json
```

The URI returned in the `Location` header points to an *operation in progress* resource.

Update the List of Installable Packages

Query

```
POST <uri>
```

Example request

Example request for the installation service of the plugin manager.

```
POST /provd/pg_mgr/install/update HTTP/1.1
Host: wazoserver
Content-Type: application/vnd.proformatique.provd+json

{}
```

Example response

```
HTTP/1.1 201 Created
Location: /provd/pg_mgr/install/update/1
Content-Type: application/vnd.proformatique.provd+json
```

The URI returned in the `Location` header points to an *operation in progress* resource.

List Installable Packages

Query

```
GET <uri>
```

Example request

Example request for the installation service of the plugin manager.

```
GET /provd/pg_mgr/install/installable HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "pkgs": {
        "null": {
            "capabilities": {
                "*, *, *": {
                    "sip.lines": 0
                }
            },
            "description": "Plugin that offers no configuration service and rejects␣
↪TFTP/HTTP requests.",
            "dsize": 1073,
            "sha1sum": "90b2fb6c2b135a9d539488b6a85779dd95e0e876",
            "version": "1.0"
        },
        "xivo-aastra-3.3.1-SP2": {
            "capabilities": {
```

```
                "Aastra, 6730i, 3.3.1.5089": {
                    "sip.lines": 6
                },
                "Aastra, 6731i, 3.3.1.2235": {
                    "sip.lines": 6,
                    "switchboard": true
                },
                "Aastra, 6735i, 3.3.1.5089": {
                    "sip.lines": 9
                },
                "Aastra, 6737i, 3.3.1.5089": {
                    "sip.lines": 9
                },
                "Aastra, 6739i, 3.3.1.2235": {
                    "sip.lines": 9
                },
                "Aastra, 6753i, 3.3.1.2235": {
                    "sip.lines": 9
                },
                "Aastra, 6755i, 3.3.1.2235": {
                    "sip.lines": 9,
                    "switchboard": true
                },
                "Aastra, 6757i, 3.3.1.2235": {
                    "sip.lines": 9,
                    "switchboard": true
                },
                "Aastra, 9143i, 3.3.1.2235": {
                    "sip.lines": 9
                },
                "Aastra, 9480i, 3.3.1.2235": {
                    "sip.lines": 9
                }
            },
            "description": "Plugin for Aastra 6730i, 6731i, 6735i, 6737i, 6739i,␣
↪6753i, 6755i, 6757i, 6757i CT, 9143i, 9480i, 9480i CT in version 3.3.1 SP2.",
            "dsize": 9397,
            "sha1sum": "68dbed6afa87cf624a89166bdc6bdf7413cb84df",
            "version": "1.1"
        }
    }
}
```


List Installed Packages

Query

```
GET <uri>
```

Example request

Example request for the installation service of the plugin manager.

```
GET /provd/pg_mgr/install/installed HTTP/1.1
Host: wazoserver
Accept: application/vnd.proformatique.provd+json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.proformatique.provd+json

{
    "pkgs": {
        "xivo-aastra-3.3.1-SP2": {
            "capabilities": {
                "Aastra, 6730i, 3.3.1.5089": {
                    "sip.lines": 6
                },
                "Aastra, 6731i, 3.3.1.2235": {
                    "sip.lines": 6,
                    "switchboard": true
                },
                "Aastra, 6735i, 3.3.1.5089": {
                    "sip.lines": 9
                },
                "Aastra, 6737i, 3.3.1.5089": {
                    "sip.lines": 9
                },
                "Aastra, 6739i, 3.3.1.2235": {
                    "sip.lines": 9
                },
                "Aastra, 6753i, 3.3.1.2235": {
                    "sip.lines": 9
                },
                "Aastra, 6755i, 3.3.1.2235": {
                    "sip.lines": 9,
                    "switchboard": true
                },
                "Aastra, 6757i, 3.3.1.2235": {
                    "sip.lines": 9,
                    "switchboard": true
                },
                "Aastra, 9143i, 3.3.1.2235": {
                    "sip.lines": 9
                },
                "Aastra, 9480i, 3.3.1.2235": {
                    "sip.lines": 9
                }
            },
            "description": "Plugin for Aastra 6730i, 6731i, 6735i, 6737i, 6739i,␣
↪6753i, 6755i, 6757i, 6757i CT, 9143i, 9480i, 9480i CT in version 3.3.1 SP2.",
            "version": "1.1"
        }
    }
}
```

xivo-sysconfd REST API

This service provides a public API that can be used to change the configuration that are on a Wazo.

> **Warning:** The 0.1 API is currently in development. Major changes could still happen and new resources will be added over time.

API reference

Asterisk Voicemail

Delete voicemail

Query

```
GET /delete_voicemail
```

Parameters

Mandatory

name the voicemail name

Optional

context the voicemail context (default is 'default')

Errors

Error code	Error message	Description
404	Not found	The voicemail does not exist

Example requests

```
GET /delete_voicemail HTTP/1.1
Host: wazoserver
Accept: application/json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    nothing
}
```

Common configuration

Apply configuration

Query

```
GET /commonconf_apply
```

Generate configuration

Query

```
POST /commonconf_generate
```

Dhcpd configuration

Update configuration

Query

```
GET /dhcpd_update
```

Ethernet configuration

Discover interfaces

Query

```
GET /discover_netifaces
```

Example request

```
GET /discover_netifaces HTTP/1.1
Host: wazoserver
Accept: application/json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
   "lo":
   {
        "hwaddress": "00:00:00:00:00:00",
        "typeid": 24,
        "alias-raw-device": null,
        "network": "127.0.0.0",
        "family": "inet",
        "physicalif": false,
        "vlan-raw-device": null,
        "vlanif": false,
        "dummyif": false,
        "mtu": 65536,
        "broadcast": "127.255.255.255",
        "hwtypeid": 772,
        "netmask": "255.0.0.0",
        "carrier": true,
        "flags": 9,
        "address": "127.0.0.1",
        "vlan-id": null,
        "type": "loopback",
        "options": null,
        "aliasif": false,
        "name": "lo"
   },
   "eth0":
   {
        "alias-raw-device": null,
        "family": "inet",
        "hwaddress": "36:76:70:29:69:c2",
        "vlan-id": null,
        "network": "172.17.0.0",
        "physicalif": false,
        "vlan-raw-device": null,
        "vlanif": false,
        "type": "eth",
        "aliasif": false,
        "broadcast": "172.17.255.255",
        "netmask": "255.255.0.0",
        "address": "172.17.0.101",
        "typeid": 6,
        "name": "eth0",
        "hwtypeid": 1,
        "dummyif": false,
        "mtu": 1500,
        "carrier": true,
        "flags": 3,
        "options": null
   }
}
```

Get interface

Query

```
GET /netiface/<interface>
```

Example request

```
GET /netiface/eth0 HTTP/1.1
Host: wazoserver
Content-Type: application/json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
   "eth0":
   {
        "alias-raw-device": null,
        "family": "inet",
        "hwaddress": "36:76:70:29:69:c2",
        "vlan-id": null,
        "network": "172.17.0.0",
        "physicalif": false,
        "vlan-raw-device": null,
        "vlanif": false,
        "type": "eth",
        "aliasif": false,
        "broadcast": "172.17.255.255",
        "netmask": "255.255.0.0",
        "address": "172.17.0.101",
        "typeid": 6,
        "name": "eth0",
        "hwtypeid": 1,
        "dummyif": false,
        "mtu": 1500,
        "carrier": true,
        "flags": 3,
        "options": null
   }
}
```

Modify interface

Description

Field	Values	Description
iface	string	Interface name like eth0
method	list	static or dhcp
address	string	
netmask	string	
broadcast	string	
gateway	string	
mtu	int	
auto	boolean	
up	boolean	
options	list	dns-search and dns-nameservers

Query

```
PUT /modify_physical_eth_ipv4
```

Example request

```
PUT /modify_physical_eth_ipv4 HTTP/1.1
Host: wazoserver
Content-Type: application/json
{
    "ifname': "eth0",
    "method': "dhcp",
    "auto": "True"
}
```

Replace virtual interface

Query

```
PUT /replace_virtual_eth_ipv4
```

Example request

```
PUT /replace_virtual_eth_ipv4 HTTP/1.1
Host: wazoserver
Content-Type: application/json
{
    "ifname": "eth0:0",
    "new_ifname": "eth0:1",
    "method": "dhcp",
```

```
    "auto": "True"
}
```

Modify interface

Query

```
PUT /modify_eth_ipv4
```

Example request

```
PUT /modify_eth_ipv4 HTTP/1.1
Host: wazoserver
Content-Type: application/json
{
    'ifname' : 'eth0'
    'address': '192.168.0.1',
    'netmask': '255.255.255.0',
    'broadcast': '192.168.0.255',
    'gateway': '192.168.0.254',
    'mtu': 1500,
    'auto': True,
    'up': True,
    'options': [['dns-search', 'toto.tld tutu.tld'],
                ['dns-nameservers', '127.0.0.1 192.168.0.254']]
}
```

Change state

Query

```
PUT /change_state_eth_ipv4
```

Example request

```
PUT /change_state_eth_ipv4 HTTP/1.1
Host: wazoserver
Content-Type: application/json
{
    'ifname' : 'eth0',
    'state': True
}
```

Delete interface ipv4

Query

```
GET /delete_eth_ipv4/<interface>
```

Example request

```
GET /delete_eth_ipv4/eth0 HTTP/1.1
Host: wazoserver
Content-Type: application/json
```

HA configuration

Get HA configuration

Query

```
GET /get_ha_config
```

Update HA configuration

Query

```
POST /update_ha_config
```

network configuration

Get network configuration

Query

```
GET /network_config
```

Rename ethernet interface

Query

```
POST /rename_ethernet_interface
```

swap ethernet interface

Query

```
POST /swap_ethernet_interfaces
```

Routes

Query

```
POST /routes
```

OpenSSL configuration

List certificates

Query

```
GET /openssl_listcertificates
```

Get certificate infos

Query

```
GET /openssl_certificateinfos
```

Export public key

Query

```
GET /openssl_exportpubkey
```

Export SSL certificate

Query

```
GET /openssl_export
```

Create CA certificate

Query

```
POST /openssl_createcacertificate
```

Create certificate

Query

```
POST / openssl_createcertificate
```

Delete certificate

Query

```
GET /openssl_deletecertificate
```

Import SSL certificate

Query

```
POST /openssl_import
```

DNS configuration

Host configuration

Query

```
POST /hosts
```

Resolv.conf configuration

Query

```
POST /resolv_conf
```

Services daemon

Reload services

Query

```
POST /services
```

Xivo Services

Reload Wazo services

Query

```
POST /xivoctl
```

Handlers

Execute handlers

Query

```
POST /exec_request_handlers
```

Status check

Status

Query

```
GET /status_check
```

Example request

```
GET /status_check HTTP/1.1
Host: wazoserver
Content-Type: application/json
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "status": "up"
}
```

For other services, see http://api.wazo.community. See also the *REST API Quickstart* for an interactive web UI.

REST API Conventions

Authentication

For all REST APIs, the main way to authenticate is to use an access token obtained from *wazo-auth*. This token should be given in the `X-Auth-Token` header in your request. For example:

```
curl <options...> -H 'X-Auth-Token: 17496bfa-4653-9d9d-92aa-17def0fa9826' https://
↪<wazo_address>:9486/1.1/users
```

Also, your token needs to have the right ACLs to give you access to the resource you want. See *REST API Permissions*.

REST API Permissions

The tokens delivered by *wazo-auth* have a list of permissions associated (ACL), that determine which REST resources are authorized for this token. Each REST resource has an associated required ACL. When you try to access to a REST resource, this resource requests wazo-auth with your token and the required ACL to validate the access.

Syntax

An ACL contains 3 parts separated by dot (`.`)

- *service*: name of service, without prefix `xivo-` (e.g. `xivo-confd` -> `confd`).
- *resource*: name of resource separated by dot (`.`) (e.g. `/users/17/lines` -> `users.17.lines`).
- *action*: action performed on resource. Generally, this is the following schema:
  - `get` -> `read`
  - `put` -> `update`
  - `post` -> `create`
  - `delete` -> `delete`

Substitutions

There are 3 substitution values for an ACL.

- `*`: replace only one word between dot.
- `#`: replace one or multiple words.
- `me`: replace the `user_uuid` from sent token.

Example

The ACL `confd.users.me.#.read` will have access to the following REST resources:

```
GET /users/{user_id}/cti
GET /users/{user_id}/funckeys
GET /users/{user_id}/funckeys/{position}
GET /users/{user_id}/funckeys/templates
GET /users/{user_id}/lines
GET /users/{user_id}/lines/{line_id}
GET /users/{user_id}/voicemail
```

- *service*: `confd`
- *resource*: `users.me.#`
- *action*: `read`

The ACL `confd.users.me.funckeys.*.*` will have access to the following REST resources:

```
DELETE /users/{user_id}funckeys/{position}
GET /users/{user_id}funckeys/{position}
PUT /users/{user_id}funckeys/{position}
GET /users/{user_id}funckeys/templates
```

- *service*: `confd`
- *resource*: `users.me.funckeys.*`
- *action*: `*`

Where `{user_id}` is the user uuid from the token.

Available ACLs

The ACL corresponding to each resource is documented in http://auth.wazo.community. Some resources may not have any associated ACL yet, so you must use `{service}.#` instead.

See also *Service Authentication* for details about the token-based authentication process.

Other methods (xivo-confd)

Warning: DEPRECATED

For compatibility reason, xivo-confd may accept requests without an access token. For this, you must create a web-services user in the web interface (*Configuration* → *Management* → *Web Services Access*):

- if an IP address is specified for the user, no authentication is needed
- if you choose not to specify an IP address for the user, you can connect to the REST API with a HTTP Digest authentication, using the user name and password you provided. For instance, the following command line allows to retrieve Wazo users through the REST API, using the login **admin** and the password **passadmin**:

```
curl <options...> --digest --cookie '' -u admin:passadmin https://<wazo_address>
↪:9486/1.1/users
```

HTTP status codes

Standard HTTP status codes are used. For the full definition see IANA definition.

- 200: Success
- 201: Created
- 400: Incorrect syntax
- 404: Resource not found
- 406: Not acceptable
- 412: Precondition failed
- 415: Unsupported media type
- 500: Internal server error

See also *Errors* for general explanations about error codes.

General URL parameters

Example usage of general parameters:

```
GET http://<wazo_address>:9486/1.1/voicemails?limit=X&offset=Y
```

Parameters

order Sort the list using a column (e.g. "number"). See specific resource documentation for columns allowed.

direction 'asc' or 'desc'. Sort list in ascending (asc) or descending (desc) order

limit total number of resources to show in the list. Must be a positive integer

offset number of resources to skip over before starting the list. Must be a positive integer

search Search resources. Only resources with a field containing the search term will be listed.

Data representation

Data retrieved from the REST server

JSON is used to encode returned or sent data. Therefore, the following headers are needed:

- **when the request is supposed to return JSON:** Accept = application/json
- **when the request's body contains JSON:** Content-Type = application/json

---

Note: Optional properties can be added without changing the protocol version in the main list or in the object list itself. Properties will not be removed, type and name will not be modified.

Getting object lists

`GET /1.1/objects`

When returning lists the format is as follows:

- total - number of items in total in the system configuration (optional)
- items - returned data as an array of object properties list.

Other optional properties can be added later.

`Response data format`

```
{
    "total": 2,

    "items":
    [
        {
            "id": "1",
            "prop1": "test"
        },
        {
            "id": "2",
            "prop1": "ssd"
        }
    ]
}
```

Getting An Object

Format returned is a list of properties. The object should always have the same attributes set, the default value being the equivalent to NULL in the content-type format.

`GET /1.1/objects/<id>`

`Response data format`

```
{
   "id": "1",
   "prop1": "test"
}
```

Data sent to the REST server

The Wazo REST server implements POST and PUT methods for item creation and update respectively. Data is created using the POST method via a root URL and is updated using the PUT method via a root URL suffixed by /<id. The server expects to receive JSON encoded data. Only one item can be processed per request. The data format and required data fields are illustrated in the following example:

`Request data format`

```
{
   "id": "1",
   "prop1": "test"
}
```

When updating, only the id and updated properties are needed, omitted properties are not updated. Some properties can also be optional when creating an object.

Errors

A request to the web services may return an error. An error will always be associated to an HTTP error code, and eventually to one or more error messages. The following errors are common to all web services:

Error code	Error message	Description
406	empty	Accept header missing or contains an unsupported content type
415	empty	Content-Type header missing or contains an unsupported content type
500	list of errors	An error occured on the server side; the content of the message depends of the type of errors which occured

The 400, 404 and 412 errors depend on the web service you are requesting. They are separately described for each of them.

The error messages are contained in a JSON list, even if there is only one error message:

```
[ message_1, message_2, ... ]
```

REST API changelog

The changelog of REST API can be found in GitHub repository of each project:

- wazo-auth changelog
- wazo-call-logd changelog
- wazo-plugind changelog
- wazo-webhookd changelog
- xivo-agentd changelog
- xivo-confd changelog
- xivo-ctid changelog
- xivo-ctid-ng changelog
- xivo-dird changelog

1.13.4 Subroutine

What is it ?

The preprocess subroutine allows you to enhance Wazo features through the Asterisk dialplan. Features that can be enhanced are :

- User
- Group
- Queue

- Meetme
- Incoming call
- Outgoing call

There are three possible categories :

- Subroutine for one feature
- Subroutine for global forwarding
- Subroutine for global incoming call to an object

Subroutines are called at the latest possible moment in the dialplan, so that the maximum of variables have already been set: this way, the variables can be read and modified at will before they are used.

Here is an example of the dialplan execution flow when an external incoming call to a user being forwarded to another external number (like a forward to a mobile phone):

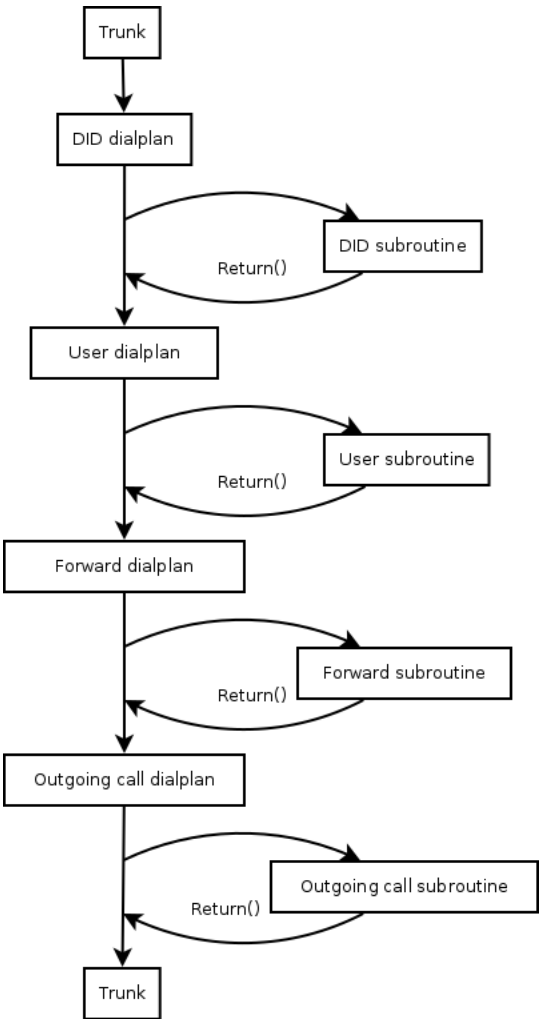


Fig. 1.104: Where subroutines are called in dialplan

Adding new subroutine

Where

You can write the subroutine in two locations, depending on what you prefer:

- add or edit a file in the web interface *Services → IPBX → IPBX Configuration → Configuration files*
- add/edit a file directly on the server in `/etc/asterisk/extensions_extra.d`

The two operations are strictly equivalent.

Note: Since all configuration files will be merged together in the end, it does not matter in which file you write your subroutine. The different files are only here to find your way back more quickly than one big configuration file. So don't be afraid to create new files!

What

An example:

```
[myexample]
exten = s,1,NoOp(This is an example)
same  =   n,Return()
```

Subroutines should always end with a `Return()`. You may replace `Return()` by a `Goto()` if you want to completely bypass the Wazo dialplan, but this is not recommended.

To plug your subroutine into the Wazo dialplan, you must add `myexample` in the subroutine field in the web interface, e.g. *Services → IPBX → PBX Settings → Users → Edit → tab General → Preprocess subroutine*.

Global subroutine

There is predefined subroutine for this feature, you can find the name and the activation in the `/etc/xivo/asterisk/xivo_globals.conf`. The variables are:

```
; Global Preprocess subroutine
XIVO_PRESUBR_GLOBAL_ENABLE = 1
XIVO_PRESUBR_GLOBAL_USER = xivo-subrgbl-user
XIVO_PRESUBR_GLOBAL_AGENT = xivo-subrgbl-agent
XIVO_PRESUBR_GLOBAL_GROUP = xivo-subrgbl-group
XIVO_PRESUBR_GLOBAL_QUEUE = xivo-subrgbl-queue
XIVO_PRESUBR_GLOBAL_MEETME = xivo-subrgbl-meetme
XIVO_PRESUBR_GLOBAL_DID = xivo-subrgbl-did
XIVO_PRESUBR_GLOBAL_OUTCALL = xivo-subrgbl-outcall
XIVO_PRESUBR_GLOBAL_PAGING = xivo-subrgbl-paging
```

So if you want to add a subroutine for all of your Wazo users you can do this:

```
[xivo-subrgbl-user]
exten = s,1,NoOp(This is an example for all my users)
same  =   n,Return()
```

Forward subroutine

You can also use a global subroutine for call forward.

```
; Preprocess subroutine for forwards
XIVO_PRESUBR_FWD_ENABLE = 1
XIVO_PRESUBR_FWD_USER = xivo-subrfwd-user
XIVO_PRESUBR_FWD_GROUP = xivo-subrfwd-group
XIVO_PRESUBR_FWD_QUEUE = xivo-subrfwd-queue
XIVO_PRESUBR_FWD_MEETME = xivo-subrfwd-meetme
XIVO_PRESUBR_FWD_VOICEMAIL = xivo-subrfwd-voicemail
XIVO_PRESUBR_FWD_SCHEDULE = xivo-subrfwd-schedule
XIVO_PRESUBR_FWD_SOUND = xivo-subrfwd-sound
XIVO_PRESUBR_FWD_CUSTOM = xivo-subrfwd-custom
XIVO_PRESUBR_FWD_EXTENSION = xivo-subrfwd-extension
```

Dialplan variables

Some of the Wazo variables can be used and modified in subroutines (non exhaustive list):

- `WAZO_CHANNEL_DIRECTION`: can have two values:
  - `from-wazo` when the channel was initiated by Wazo: the channel links Wazo to the called party. From Asterisk, this is an outbound channel. From the peer, this is an incoming call
  - `to-wazo` when the channel was initiated by the user: the channel links Wazo to the calling party. From Asterisk, this is an inbound channel. From the peer, this is an outgoing call.

  The default value is `from-wazo`. If you write scripts using originates to place new calls, you should set `WAZO_CHANNEL_DIRECTION` to `to-wazo` on the originator channel.
- `XIVO_CALLOPTIONS`: the value is a list of options to be passed to the Dial application, e.g. `hHtT`. This variable is available in agent, user and outgoing call subroutines. Please note that it may not be set earlier, because it will be overwritten.
- `XIVO_CALLORIGIN`: can have two values:
  - `intern` when the call does not involve DID or trunks, e.g. a simple call between two phones or one phone and its voicemail
  - `extern` when the call is received via a DID or emitted through an Outgoing Call

  This variable is used by xivo-agid when *selecting the ringtone* for ringing a user. This variable is available only in user subroutines.
- `XIVO_DSTNUM`: the value is the extension dialed, as received by Wazo (e.g. an internal extension, a DID, or an outgoing extension including the local prefix). This variable is available in all subroutines.
- `XIVO_GROUPNAME`: the value is the name of the group being called. This variable is only available in group subroutines.
- `XIVO_GROUPOPTIONS`: the value is a list of options to be passed to the Queue application, e.g. `hHtT`. This variable is only available in group subroutines.
- `XIVO_INTERFACE`: the value is the *Technology/Resource* pairs that are used as the first argument of the Dial application. This variable is only available in the user subroutines.
- `XIVO_MOBILEPHONENUMBER`: the value is the phone number of a user, as set in the web interface. This variable is only available in user subroutines.

- XIVO_QUEUENAME: the value is the name of the queue being called. This variable is only available in queue subroutines.
- XIVO_QUEUEOPTIONS: the value is a list of options to be passed to the Queue application, e.g. hHtT. This variable is only available in queue subroutines.
- XIVO_RINGSECONDS: the value is the number of seconds a user will ring before the call is forwarded elsewhere, or hungup if no forwards are configured. This variable can only be used in a User subroutine.
- XIVO_SRCNUM: the value is the callerid number of the originator of the call: the internal extension of a user (outgoing callerid is ignored), or the public extension of an external incoming call. This variable is available in all subroutines.

1.13.5 WebSocket Event Service

Wazo offers a service to receive messages published on the *bus (e.g. RabbitMQ)* over an encrypted WebSocket connection. This ease in building dynamic web applications that are using events from your Wazo.

The service is provided by the xivo-websocketd component.

Getting Started

To use the service, you need to:

1. connect to it on port 9502 using an encrypted WebSocket connection.
2. authenticate to it by providing a wazo-auth token that has the websocketd ACL. If you don't know how to obtain a wazo-auth token from your Wazo, consult the *documentation on wazo-auth*.

For example, if you want to use the service located at example.org with the token some-token-id, you would use the URL wss://example.org:9502/?token=some-token-id.

The *SSL/TLS certificate* that is used by the WebSocket server is the same as the one used by the Wazo web interface and the REST APIs. By default, this is a self-signed certificate, and web browsers will prevent connections from being successfully established for security reasons. On most web browsers, this can be circumvented by first visiting the https://<wazo-ip>:9502/ URL and adding a security exception. Other solutions to this problem are described in the *connection section*.

After a succesful connection and authentication to the service, the server will send the following message:

```
{"op": "init", "code": 0, "msg": ""}
```

This indicate that the server is ready to accept more commands from the client. Had an error happened, the server would have closed the connection, possibly with one of the *service specific WebSocket close code*.

The message you see is part of the small *JSON-based protocol* that is used for the client/server interaction.

To receive events on your WebSocket connection, you need to tell the server which type of events you are interested in, and then tell it to start sending you these events. For example, if you are interested in the *"endpoint_status_update" events*, you send the following command:

```
{"op": "subscribe", "data": {"event_name": "endpoint_status_update"}}
```

If all goes well, the server will respond with:

```
{"op": "subscribe", "code": 0, "msg": ""}
```

Once you have subscribed to all the events you are interested in, you ask the server to start sending you the matching events by sending the following command:

```
{"op": "start"}
```

The server will respond with:

```
{"op": "start", "code": 0, "msg": ""}
```

Once you have received this message, all the other messages you'll receive will be messages originating from the bus, in the same format as they were on the bus.

Example

Here's a rudimentary example of a web page accessing the service:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Wazo WebSocket Example</title>
<script>
var socket = null;
var started = false;

function connect() {
    if (socket != null) {
        console.log("socket already connected");
        return;
    }

    var host = document.getElementById("host").value;
    var token_id = document.getElementById("token").value;
    socket = new WebSocket("wss://" + host + ":9502/?token=" + token_id);
    socket.onclose = function(event) {
        socket = null;
        console.log("websocketd closed with code " + event.code + " and reason '" +
↪event.reason + "'");
    };
    socket.onmessage = function(event) {
        if (started) {
            console.log("message received: " + event.data);
            return;
        }

        var msg = JSON.parse(event.data);
        switch (msg.op) {
            case "init":
                subscribe("*");
                start();
                break;
            case "start":
                started = true;
                console.log("waiting for messages");
                break;
        }
    };
```

```
41      started = false;
42  }
43
44  function subscribe(event_name) {
45      var msg = {
46          op: "subscribe",
47          data: {
48            event_name: event_name
49          }
50      };
51      socket.send(JSON.stringify(msg));
52  };
53
54  function start() {
55      var msg = {
56          op: "start"
57      };
58      socket.send(JSON.stringify(msg));
59  }
60  </script>
61  </head>
62  <body>
63    <p>Open the web console to see what's happening.</p>
64    <form>
65      <div>
66        <label for="host">Host:</label>
67        <input type="text" id="host" autofocus>
68      </div>
69      <div>
70        <label for="token">Token ID:</label>
71        <input type="text" id="token" size="35">
72      </div>
73      <div>
74        <button type="button" onclick="connect();">Connect</button>
75      </div>
76    </form>
77  </body>
78  </html>
```

The page has a form for the user to enter a host and token ID, and has a connect button. When the button is clicked, the `connect` function is called, and the WebSocket connection is created at line 18 (using the WebSocket API):

```
socket = new WebSocket("wss://" + host + ":9502/?token=" + token_id);
```

Then, at line 23, a `onmessage` callback is set on the WebSocket object:

```
socket.onmessage = function(event) {
    if (started) {
        console.log("message received: " + event.data);
        return;
    }

    var msg = JSON.parse(event.data);
    switch (msg.op) {
        case "init":
            subscribe("endpoint_status_update");
            subscribe("user_status_update");
            start();
```

```
            break;
        case "start":
            started = true;
            console.log("waiting for messages");
            break;
    }
};
```

After a successful connection to the service, an "init" message will be received by the client. When the client receives this message, it sends two subscribe commands (e.g. `subscribe("endpoint_status_update")`) and a start command (e.g. `start()`). When the client receives the "start" message, it sets the `started` flag. After that, all the other messages it receives will be logged to the console.

Reference

The WebSocket service is provided by `xivo-websocketd`, and its behaviour can be configured via its *configuration files* located under the `/etc/xivo-websocketd` directory. After modifying the configuration files, you need to restart `xivo-websocketd` with `systemctl restart xivo-websocketd`.

Connection

The service is available on port 9502 on all network interfaces by default. This can be changed in the configuration file.

The canonical URL to reach the service is `wss://<host>:9502/`.

The connection is always encrypted. The certificate and private key used by the server can be changed in the configuration file. By default, since the certificate is self-signed, you'll have to either:

- add a security exception on the client machines that access the service
- use a certificate signed by an untrusted CA and add the CA bundle on the system that access the service
- use a trusted certificate

See the *Certificates for HTTPS* section for more information on certificate configuration.

Authentication

Authentication is done by passing a wazo-auth token ID in the `token` query parameter. Authentication is mandatory.

The token must have the `websocketd` ACL.

When the token expires, the server close the connection with the status code 4003. There is currently no way to change the token of an existing connection. A new connection must be made when the token expires.

Events Access Control

Clients connected to `xivo-websocketd` only receive events that they are authorized to receive. For example, a client connected with a token obtained from the "xivo_user" `wazo-auth` backend will *not* receive call events of other users.

When a message is received from the bus by `xivo-websocketd`, it extracts the ACL from the `required_acl` key of the event. If the field is missing, no clients will receive the event. If the value is null, all subscribed clients will receive the event. If the value is a string, then all subscribed clients which have a matching ACL will receive the event.

No authorization check is done at subscription time. Checks are only done when an event is received by the server. This mean a client can subscribe to an event "foo", but will never receive any of these events if it does not have the matching ACL.

See the *Events* section for more information on the required ACL of events which are available by default on Wazo.

Status Code

The WebSocket connection might be closed by the server using one of following status code:

- 4001: No token ID was provided.
- 4002: Authentication failed. Either the token ID is invalid, expired, or does not have the necessary ACL.
- 4003: Authentication expired. The token has expired or was deleted.
- 4004: Protocol error. The server received a frame that it could not understand. For example, the content was not valid JSON, or was requesting an unknown operation, or a mandatory argument to an operation was missing.

The server also uses the pre-defined WebSocket status codes.

Protocol

A JSON-based protocol is used over the WebSocket connection to control which events are received by the client.

Client Messages

The format of the messages sent by the client are all of the same format:

```
{"op": "<operation-name>", "data": <operation-specific-value>}
```

The "op" key is mandatory, and the value is either "subscribe" or "start". The "data" key is mandatory for the "subscribe" operation.

The "subscribe" message ask the server to subscribe the client to the given event. When a message with the same name is published on the "xivo" exchange of the bus, the server forwards the message to all the subscribed clients that are authorized to receive it. For this command, the "data" value is a dictionary with an "event_name" key (mandatory). Example:

```
{"op": "subscribe", "data": {"event_name": "endpoint_status_update"}}
```

You can subscribe to any event. The special event name * can be used to match all events.

See the *Events* section for more information on the events which are available by default on Wazo.

The "start" message ask the server to start sending messages from the bus to the client. Example:

```
{"op": "start"}
```

The server won't forward messages from the bus to the client until it receives the "start" message from the client.

If the client send a message that the server doesn't understand, the server closes the connection.

Server Messages

The format of the messages sent by the server are all of the same format (until the server receives a "start" command):

```
{"op": "<operation-name>", "code": <status-code>, "msg": "<error message>"}
```

The 3 keys are always present. The value of the "op" key can be one of "init", "subscribe" or "start". The value of the "code" key is an integer representing the status of the operation, 0 meaning there was no error, other values meaning there was an error. The "msg" is an empty string unless "code" is non-zero, in which case it's a human-readable message of the error.

The "init" message is only sent after the connection is successfully established between the client and the server. It's code is always zero; if the connection could not be established, the connection is simply closed. Example:

```
{"op": "init", "code": 0, "msg": ""}
```

The "subscribe" message is sent as a response to a client "subscribe" message. The code is always zero. Example:

```
{"op": "subscribe", "code": 0, "msg": ""}
```

The "start" message is sent as a response to a client "start" message. The code is always zero. Example:

```
{"op": "start", "code": 0, "msg": ""}
```

After receiving the "start" message, the server switch into the "bus/started" mode, where all messages that the server will ever sent will be the body of the messages it received on the bus on behalf of the client.

Note that a client can subscribe to more events after sending its "start" message, but it won't receive any response from the server, e.g. the server won't send a corresponding "subscribe" message. Said differently, once the client has sent a "start" message, every message the client will ever receive are messages coming from the bus.

1.14 Contributors

General information:

1.14.1 Contributing to the Documentation

Wazo documentation is generated with Sphinx. The source code is available on GitHub at https://github.com/wazo-pbx/xivo-doc

Provided you already have Python installed on your system. You need first to install Sphinx : `easy_install -U Sphinx`[1].

Quick Reference

- http://docutils.sourceforge.net/docs/user/rst/cheatsheet.txt
- http://docutils.sourceforge.net/docs/user/rst/quickref.html
- http://openalea.gforge.inria.fr/doc/openalea/doc/_build/html/source/sphinx/rest_syntax.html

Documentation guideline

Here's the guideline/conventions to follow for the Wazo documentation.

[1] `easy_install` can be found in the debian package python-setuptools : `sudo apt-get install python-setuptools`

Language

The documentation must be written in english, and only in english.

Sections

The top section of each file must be capitalized using the following rule: capitalization of all words, except for articles, prepositions, conjunctions, and forms of to be.

Correct:

```
The Vitamins are in My Fresh California Raisins
```

Incorrect:

```
The Vitamins Are In My Fresh California Raisins
```

Use the following punctuation characters:

- * with overline, for "file title"
- =, for sections
- -, for subsections
- ^, for subsubsections

Punctuation characters should be exactly as long as the section text.

Correct:

```
Section1
========
```

Incorrect:

```
Section2
==========
```

There should be 2 empty lines between sections, except when an empty section is followed by another section.

Correct:

```
Section1
========

Foo.


Section2
========

Bar.
```

Correct:

```
Section1
========
```

```
Foo.


.. _target:

Section2
========

Bar.
```

Correct:

```
Section1
========

Subsection1
-----------

Foo.
```

Incorrect:

```
Section1
========

Foo.

Section2
========

Bar.
```

Lists

Bullet lists:

```
* First item
* Second item
```

Autonumbered lists:

```
#. First item
#. Second item
```

Literal blocks

Use `::` on the same line as the line containing text when possible.

The literal blocks must be indented with three spaces.

Correct:

```
Bla bla bla::

   apt-get update
```

Incorrect:

```
Bla bla bla:

::

   apt-get update
```

Inline markup

Use the following roles when applicable:

- `:file:` for file, i.e.:

  ```
  The :file:`/dev/null` file.
  ```

- `:menuselection:` for the web interface menu:

  ```
  The :menuselection:`Configuration --> Management --> Certificates` page.
  ```

- `:guilabel:` for designating a specific GUI element:

  ```
  The :guilabel:`Action` column.
  ```

Others

- There must be no warning nor error messages when building the documentation with `make html`.
- There should be one and only one newline character at the end of each file
- There should be no trailing whitespace at the end of lines
- Paragraphs must be wrapped and lines should be at most 100 characters long

1.14.2 Debugging Asterisk

Precondition

To debug asterisk crashes or freezes, you need the following debug packages on your Wazo:

General rule	XiVO < 14.18	XiVO >= 14.18	Wazo >= 16.16
Example version	14.12	14.18	17.15

Commands	`apt-get install␣ ↪xivo-fai-14.12`
	`apt-get update`
	`apt-get install gdb`
	`apt-get install -t␣ ↪xivo-14.12␣ ↪asterisk-dbg␣ ↪xivo-libsccp-dbg`
		`xivo-dist xivo-14. ↪18`
		`apt-get update`
		`apt-get install gdb`
		`apt-get install -t␣ ↪xivo-14.18␣ ↪asterisk-dbg␣ ↪xivo-libsccp-dbg`
		`xivo-dist xivo-five`
			`xivo-dist wazo-17. ↪15`
			`apt-get update`
			`apt-get install␣ ↪gdb libc6-dbg`
			`apt-get install -t␣ ↪wazo-17.15␣ ↪asterisk-dbg␣ ↪xivo-libsccp-dbg`
			`xivo-dist phoenix`

So There is a Problem with Asterisk. Now What ?

1. Find out the time of the incident from the people most likely to know
2. Determine if there was a segfault
   (a) The command `grep segfault /var/log/syslog` should return a line such as the following:

```
Oct 16 16:12:43 xivo-1 kernel: [10295061.047120] asterisk[1255]: segfault at␣
↪e ip b751aa6b sp b5ef14d4 error 4 in libc-2.11.3.so[b74ad000+140000]
```

   (b) Note the exact time of the incident from the segfault line.
   (c) Follow the *Debugging Asterisk Crash* procedure.
3. If you observe some of the following common symptoms, follow the *Debugging Asterisk Freeze* procedure.
   - The output of command `service asterisk status` says Asterisk PBX is running.
   - No more calls are distributed and phones go to `No Service`.
   - Command `core show channels` returns only headers (no data) right before returning
4. Fetch Asterisk logs for the day of the crash (make sure file was not already logrotated):

```
cp -a /var/log/asterisk/full /var/local/`date +"%Y%m%d"`-`hostname`-asterisk-full.
↪log
```

5. Fetch xivo-ctid logs for the day of the crash (make sure file was not already logrotated):

```
cp -a /var/log/xivo-ctid.log /var/local/`date +"%Y%m%d"`-`hostname`-xivo-ctid.log
```

6. Open a new issue on the bugtracker with following information
   - Tracker: Bug
   - Status: New
   - Category: Asterisk
   - In versions: The version of your Wazo installation where the crash/freeze happened
   - Subject : `Asterisk Crash` or `Asterisk Freeze`
   - Description : Add as much context as possible, if possible, a scenario that lead to the issue, the date and time of issue, where we can fetch logs and backtrace
   - Attach logs and backtrace (if available) to the ticket (issue must be saved, then edited and files attached to a comment).

Debugging Asterisk Crash

When asterisk crashes, it usually leaves a core file in `/var/spool/asterisk/`.

You can create a backtrace from a core file named `core_file` with:

```
gdb -batch -ex "bt full" -ex "thread apply all bt" /usr/sbin/asterisk core_file > bt-
↪threads.txt
```

Debugging Asterisk Freeze

You can create a backtrace of a running asterisk process with:

```
gdb -batch -ex "thread apply all bt" /usr/sbin/asterisk $(pidof asterisk) > bt-
↪threads.txt
```

If your version of asterisk has been compiled with the DEBUG_THREADS flag, you can get more information about locks with:

```
asterisk -rx "core show locks" > core-show-locks.txt
```

Note: Debugging freeze without this information is usually a lot more difficult.

Optionally, other information that can be interesting:

- the output of `asterisk -rx 'core show channels'`
- the verbose log of asterisk just before the freeze

Recompiling Asterisk

It's relatively straightforward to recompile the asterisk version of your Wazo with the DEBUG_THREADS and DONT_OPTIMIZE flag, which make debugging an asterisk problem easier.

The steps are:

1. Uncomment the `deb-src` line for the Wazo sources:

   ```
   sed -i 's/^# *deb-src/deb-src/' /etc/apt/sources.list.d/xivo*
   ```

2. Fetch the asterisk source package:

   ```
   mkdir -p ~/ast-rebuild
   cd ~/ast-rebuild
   apt-get update
   apt-get install -y build-essential
   apt-get source asterisk
   ```

3. Install the build dependencies:

   ```
   apt-get build-dep -y asterisk
   ```

4. Enable the DEBUG_THREADS and DONT_OPTIMIZE flag:

   ```
   cd <asterisk-source-folder>
   vim debian/rules
   ```

5. Update the changelog by appending `+debug1` in the package version:

   ```
   vim debian/changelog
   ```

6. Rebuild the asterisk binary packages:

   ```
   dpkg-buildpackage -us -uc
   ```

This will create a couple of .deb files in the parent directory, which you can install via dpkg.

Recompiling a vanilla version of Asterisk (Wazo < 17.17)

It is sometimes useful to produce a "vanilla" version of Asterisk, i.e. a version of Asterisk that has none of the Wazo patches applied, to make sure that the problem is present in the original upstream code. This is also sometimes necessary before opening a ticket on the Asterisk issue tracker.

The procedure is similar to the one described above. Before calling `dpkg-buildpackage`, you just need to:

1. Make sure `quilt` is installed:

```
apt-get install -y quilt
```

2. Unapply all the currently applied patches:

```
quilt pop -a
```

3. Remove all the lines in the `debian/patches/series` file:

```
truncate -s0 debian/patches/series
```

When installing a vanilla version of Asterisk on a XiVO 16.08 or earlier, you'll need to stop monit, otherwise it will restart asterisk every few minutes.

Recompiling a vanilla version of Asterisk (Wazo >= 17.17)

It is sometimes useful to produce a "vanilla" version of Asterisk, i.e. a version of Asterisk that has none of the Wazo patches applied, to make sure that the problem is present in the original upstream code. This is also sometimes necessary before opening a ticket on the Asterisk issue tracker.

Wazo offers a vanilla version of Asterisk, compiled with the DONT_OPTIMIZE flag. This makes filing bug reports to Asterisk much easier.

Note that this version of Asterisk loses some features that are specific to Wazo. The removed features include:

- Queue skill-based routing
- Voicemail message consultation via REST API
- Call transfers via Wazo Client or REST API

To install the vanilla version of Asterisk (replace 17.17 with your current version of Wazo):

```
xivo-dist wazo-17.17
apt-get update
apt-get install -t wazo-17.17 asterisk-vanilla asterisk-vanilla-dbg
xivo-fix-paths-rights
xivo-dist phoenix
```

This command should replace the `asterisk` package with `asterisk-vanilla`.

Once the packages are installed, you can reproduce the crash and extract the backtrace logs from the core dump file. Those file may then be used to file a bug report to Asterisk.

To revert this modification, reinstall `asterisk` (replace 17.17 with your current version of Wazo):

```
xivo-dist wazo-17.17
apt-get update
apt-get install -t wazo-17.17 asterisk
xivo-fix-paths-rights
xivo-dist phoenix
```

Running Asterisk under Valgrind

1. Install valgrind:

```
apt-get install valgrind
```

2. Recompile asterisk with the DONT_OPTIMIZE flag.
3. Edit `/etc/asterisk/modules.conf` so that asterisk doesn't load unnecessary modules. This step is optional. It makes asterisk start (noticeably) faster and often makes the output of valgrind easier to analyze, since there's less noise.
4. Edit `/etc/asterisk/asterisk.conf` and comment the `highpriority` option. This step is optional.
5. Stop monit and asterisk:

```
monit quit
service asterisk stop
```

6. Stop all unneeded Wazo services. For example, it can be useful to stop xivo-ctid, so that it won't interact with asterisk via the AMI.
7. Copy the valgrind.supp file into /tmp. The valgrind.supp file is located in the contrib directory of the asterisk source code.
8. Execute valgrind in the /tmp directory:

```
cd /tmp
valgrind --leak-check=full --log-file=valgrind.txt --suppressions=valgrind.supp --
↪vgdb=no asterisk -G asterisk -U asterisk -fnc
```

Note that when you terminate asterisk with Control-C, asterisk does not unload the modules before exiting. What this means is that you might have lots of "possibly lost" memory errors due to that. If you already know which modules is responsible for the memory leak/bug, you should explicitly unload it before terminating asterisk.

Running asterisk under valgrind takes a lots of extra memory, so make sure you have enough RAM.

External links

- https://wiki.asterisk.org/wiki/display/AST/Debugging
- http://blog.wazo.community/visualizing-asterisk-deadlocks.html
- https://wiki.asterisk.org/wiki/display/AST/Valgrind

1.14.3 Debugging Daemons

To activate debug mode, add `debug: true` in the daemon *configuration file*). The output will be available in the daemon's *log file*.

It is also possible to run the Wazo daemon, in command line. This will allow to run in foreground and debug mode. To see how to use it, type:

```
xivo-{name} -h
```

Note that it's usually a good idea to stop monit before running a daemon in foreground:

```
systemctl stop monit.service
```

xivo-confgend

```
twistd -no -u xivo-confgend -g xivo-confgend --python=/usr/bin/xivo-confgend --logger␣
↪xivo_confgen.bin.daemon.twistd_logs
```

No debug mode in confgend.

xivo-provd

```
twistd -no -u xivo-provd -g xivo-provd -r epoll --logger provd.main.twistd_logs xivo-
↪provd -s -v
```

- -s for logging to stderr
- -v for verbose

consul

```
sudo -u consul /usr/bin/consul agent -config-dir /etc/consul/xivo -pid-file /var/run/
↪consul/consul.pid
```

Consul logs its output to `/var/log/syslog` to get the output of consul only use consul monitor:

```
consul monitor -ca-file=/usr/share/xivo-certs/server.crt -http-addr=https://
↪localhost:8500
```

```
2015/08/03 09:48:25 [INFO] consul: cluster leadership acquired
2015/08/03 09:48:25 [INFO] consul: New leader elected: this-xivo
2015/08/03 09:48:26 [INFO] raft: Disabling EnableSingleNode (bootstrap)
2015/08/03 11:04:08 [INFO] agent.rpc: Accepted client: 127.0.0.1:41545
```

Note: The ca-file can be different when using custom HTTPS certificates

mongooseim

```
echo "{loglevel, 5}." >> /etc/mongooseim/wazo.cfg
systemctl restart mongooseim
```

Log file: `/var/log/mongooseim/ejabberd.log`. This file contains almost nothing before the loglevel is set.

You should not leave the loglevel enabled in production for too long. To disable it:

```
sed -i '/loglevel, 5/d' /etc/mongooseim/wazo.cfg
systemctl restart mongooseim
```

1.14.4 Generate your own prompts

If you want your Wazo to speak in your language that is not supported by Wazo, and you don't want to record the whole package of sounds in a studio, you may generate them yourself with some text-to-speech services.

The following procedure will generate prompts for `pt_BR` (portuguese from Brazil) based on the Google TTS service.

Note: There are two sets of prompts: the Asterisk prompts and the Wazo prompts. This procedure only covers the Wazo prompts, but it may be adapted for Asterisk prompts.

---

1. Create an account on Transifex and join the team of translation of Wazo.
2. Translate the prompts in the wazo-prompt resource.
3. Go to https://www.transifex.com/wazo/wazo/wazo-prompt/pt_BR/download/for_use/ and download the file on your Wazo. You should have a file named like `for_use_wazo_wazo-prompt_pt_BR.ini`.
4. On your Wazo, download the tool to automate the use of Google TTS:

```
wget https://github.com/zaf/asterisk-googletts/raw/master/cli/googletts-cli.pl
chmod +x googletts-cli.pl
```

5. Then run the following script to generate the sound files (set `LANGUAGE` and `COUNTRY` to your own language):

```
LANGUAGE=pt
COUNTRY=BR
mkdir -p wav/{digits,letters}
cat for_use_wazo_wazo-prompt_${LANGUAGE}_${COUNTRY}.ini | while IFS='=' read file␣
↪text ; do
  echo $file
  ./googletts-cli.pl -t "$text" -l ${LANGUAGE}-${COUNTRY} -s 1.4 -r 8000 -o wav/
↪$file.wav
done
```

6. Install the prompts on your system:

```
mv wav /usr/share/asterisk/sounds/${LANGUAGE}_${COUNTRY}
```

7. Make your language available in the web interface:

```
sed -i "s/'nl_NL'/\0, '${LANGUAGE}_${COUNTRY}'/" /usr/share/xivo-web-interface/
↪lib/i18n.inc
systemctl restart spawn-fcgi
```

Note that this last modification may be erased after running `wazo-upgrade`.

And that's it, you can configure a user to use your new language and he will hear the prompts in your language. You may also want to use the *xivo-confd HTTP API* to mass-update your users.

1.14.5 Wazo Guidelines

Inter-process communication

Our current goal is to use only two means of communication between Wazo processes:

- a REST API over HTTP for synchronous commands

- a software bus (RabbitMQ) for asynchronous events

Each component should have its own REST API and its own events and can communicate with every other component from across a network only via those means.

Service API

The current xivo-dao Git repository contains the basis of the future services Python API. The API is split between different resources available in Wazo, such as users, groups, schedules... For each resource, there are different modules :

- service: the public module, providing possible actions. It contains only business logic and no technical logic. There must be no file name, no SQL queries and no URLs in this module.
- dao: the private Data Access Object. It knows where to get data and how to update it, such as SQL queries, file names, URLs, but has no business logic.
- model: the public class used to represent the resource. It must be self-contained and have almost no methods, except for computed fields based on other fields in the same object.
- notifier: private, it knows to whom and in which format events must be sent.
- validator: private, it checks input parameters from the service module.

Definition of Wazo Daemon

The goal is to make Wazo as elastic as possible, i.e. the Wazo services need to be able to run on separate machines and still talk to each other.

To be in accordance with our goal, a Wazo daemon must (if applicable):

- Offer a REST API (with encryption, authentication and accepting cross-site requests)
- Be able to read and send events on a software bus
- Be able to run inside a container, such as Docker, and be separated from the Wazo server
- Offer a configuration file in YAML format.
- Access the Wazo database through the `xivo-dao` library
- Have a configurable level of logging
- Have its own log file
- Be extendable through the use of plugins
- Not run with system privileges
- Be installable from source
- Service discovery with consul

Currently, none of the Wazo daemons meet these expectations; it is a work in progress.

1.14.6 Network

Configuration for daemon

Network Flow table (IN) :

Daemon Name	Service	Protocol	Port	Listen	Authentication	Enabled
-	ICMP	ICMP	-	0.0.0.0	no	yes
postfix	SMTP	TCP	25	0.0.0.0	yes	yes
isc-dhcpd	DHCP	UDP	67	0.0.0.0	no	no
isc-dhcpd	DHCP	UDP	68	0.0.0.0	no	no
xivo-provd	TFTP	UDP	69	0.0.0.0	no	yes
ntpd	NTP	UDP	123	0.0.0.0	yes	yes
monit	HTTP	TCP	2812	127.0.0.1	no	yes
asterisk	SIP	UDP	5060	0.0.0.0	yes	yes
asterisk	IAX	UDP	4569	0.0.0.0	yes	yes
asterisk	SCCP	TCP	2000	0.0.0.0	yes	yes
asterisk	AMI	TCP	5038	127.0.0.1	yes	yes
asterisk	HTTP	TCP	5039	127.0.0.1	yes	yes
asterisk	HTTPS	TCP	5040	127.0.0.1	yes	yes
sshd	SSH	TCP	22	0.0.0.0	yes	yes
nginx	HTTP	TCP	80	0.0.0.0	yes	yes
nginx	HTTPS	TCP	443	0.0.0.0	yes	yes
munin	HTTP	TCP	4949	127.0.0.1	no	yes
xivo-ctid	XiVO-CTI/S	TCP	5003	0.0.0.0	yes	yes
postgresql	SQL	TCP	5432	127.0.0.1	yes	yes
rabbitMQ	AMQP	TCP	5672	0.0.0.0	yes	yes
consul	Consul RPC	TCP	8300	127.0.0.1	yes	yes
consul	Consul Serf LAN	TCP/UDP	8301	127.0.0.1	yes	yes
consul	Consul Serf WAN	TCP/UDP	8302	127.0.0.1	yes	yes
consul	Consul HTTPS	TCP	8500	127.0.0.1	both	yes
xivo-provd	HTTP	TCP	8666	127.0.0.1	no	yes
xivo-provd	HTTP	TCP	8667	0.0.0.0	no	yes
xivo-confgend	HTTP	TCP	8669	127.0.0.1	no	yes
xivo-sysconfd	HTTP	TCP	8668	127.0.0.1	no	yes
wazo-admin-ui	HTTPS	TCP	9296	0.0.0.0	yes	yes
wazo-auth	HTTPS	TCP	9497	0.0.0.0	both	yes
wazo-call-logd	HTTPS	TCP	9298	0.0.0.0	yes	yes
wazo-webhookd	HTTPS	TCP	9300	0.0.0.0	yes	yes
xivo-confd	HTTPS	TCP	9486	0.0.0.0	yes	yes
xivo-confd	HTTP	TCP	9487	127.0.0.1	no	yes
xivo-dird	HTTPS	TCP	9489	0.0.0.0	yes	yes
xivo-amid	HTTPS	TCP	9491	0.0.0.0	yes	yes
xivo-agentd	HTTPS	TCP	9493	0.0.0.0	yes	yes
xivo-ctid	HTTP	TCP	9495	127.0.0.1	no	yes
xivo-dird-phoned	HTTP	TCP	9498	0.0.0.0	IP filtering	yes
xivo-dird-phoned	HTTPS	TCP	9499	0.0.0.0	IP filtering	yes
xivo-ctid-ng	HTTPS	TCP	9500	0.0.0.0	yes	yes
xivo-websocketd	WSS	TCP	9502	0.0.0.0	yes	yes
wazo-plugind	HTTPS	TCP	9503	0.0.0.0	yes	yes

1.14.7 Plugins

This section cover the preferred way to extend the functionalities of a Wazo server. There are many extension point in Wazo, all of them can be used in combination to add complete features to you favorite PBX.

For more information about installing, updating and removing plugins see the *Plugins* section.

What is a plugin

A plugin is a set of additions made to a custom Wazo installation to add a new functionality.

What can be done with a plugin

Wazo plugins allow a third party to add almost anything to Wazo. Most of our services have extension points that can be used together to create a complete feature as a plugin.

Here's a non exhaustive list of what can be done with plugins

- Add configuration files to wazo services in `/etc/*/conf.d/`
- Add configuration files and dialplan files to Asterisk
- Reload services to complete the installation
- Extend wazo services using the available extension points
  - wazo-auth
  - xivo-confd
  - xivo-confgend
  - xivo-ctid-ng
  - xivo-dird

Creating a plugin

A plugin has the following structure:

- `wazo/plugin.yml`
- `wazo/rules`

plugin.yml

The `plugin.yml` file contains all the metadata of plugin. It should contains the following fields:

- `description`: The description of the plugin
- `name`: The name of the plugin
- `namespace`: An identifier for the author of the plugin
- `version`: The version of the plugin
- `plugin_format_version`: The version of the plugin specification implemented by this plugin.
- `depends`: Other plugins which this plugin depends on
- `debian_depends`: Debian packages which this plugin depends on

Example:

```
name: foobar
namespace: foocorp
version: 0.0.1
description: This plugin adds some foo to your Wazo
plugin_format_version: 1
```

```
depends:
  - name: foobaz
    namespace: foocorp
  - name: admin-ui-context
    namespace: official
debian_depends:
  - golang-go
```

rules

The *rules* file is an executable that will accept the following commands

- build
- package
- install
- uninstall
- postrm

Hello World

This example will create a plugin that adds an extension ****42* that says *Hello World* when called.

`wazo/plugin.yml`:

```
namespace: demo
name: helloworld
description: Adds the extension "***42" to you dialplan to greet users
version: 0.0.1
plugin_format_version: 0
```

`wazo/rules`:

```
#!/bin/sh

case "$1" in
    build)
        ;;
    package)
        mkdir -p ${pkgdir}/etc/asterisk/extensions_extra.d
        cp helloworld.conf ${pkgdir}/etc/asterisk/extensions_extra.d/
        ;;
    install)
        asterisk -x 'dialplan reload'
        ;;
    uninstall)
        ;;
    *)
        echo "$0 called with unknown argument '$1'" >&2
        exit 1
        ;;
esac
```

`helloworld.conf`:

```
[xivo-extrafeatures]
exten = ***42,1,Playback(hello-world)
same = n,Return()
```

Plugin format version

0 (default)

A plugin in version *0* should implement the following requirements:

- an executable name `wazo/rules` that returns *0* on success for the following commands:
  - build
  - package
  - install
  - uninstall

1 (recommended)

Version *1* adds support for the postrm instruction in the rules file.

rules commands

build The *build* command is used to compile or generate files that will be included in the package.

package The *package* command is used to copy all files required by the plugin in the `<pkgdir>` directory.

The *pkgdir* environment variable holds the prefix that will be used to build the package. If the plugin needs to install a file in `/etc/foo/bar` do the following commands

```
mkdir -p ${pkgdir}/etc/foo
cp bar ${pkgdir}/etc/foo/bar
```

install The *install* command is used at the end of the installation to execute instructions that are usually not related to the file system. It will be used as the postinst of the generated debian package.

uninstall The *uninstall* command is used before the debian package is removed. It will be used as the prerm of the generated debian package.

postrm (added in version 1) The *postrm* command is used at the end of the debian package removal. It will be used as the postrm of the generated debian package.

Dependencies

There are 2 kinds of dependencies that can be added on a plugin, “depends” and “debian_depends”.

depends

The *depends* section of the *plugin.yml* file contains dependencies that are other plugins built for wazo-plugind. Those dependencies should be already installed or available on the market.

There's no version requirements for this kind of dependencies, they are used to make plugin installation less of a hassle.

When installing a plugin if a dependency is already satisfied, the package will not be upgraded.

Example:

```
Given a plugin "A" depending on plugin "B".
Given "B" is already installed in an older version.
When installing "A".
Then "B" will not be upgraded.
```

depends also generate an entry in the *debian_depends* section.

debian_depends

The *debian_depends* section of the *plugin.yml* file contains dependencies that will be added to the debian control file. This means that the debian packages listed here will be installed during the plugin installation. This also means that removing that dependency from the system will also remove all plugins depending on it.

1.14.8 Profiling Python Programs

Profiling CPU/Time Usage

Here's an example on how to profile xivo-ctid for CPU/time usage:

1. Stop the monit daemon:

   ```
   service monit stop
   ```

2. Stop the process you want to profile, i.e. xivo-ctid:

   ```
   service xivo-ctid stop
   ```

3. Start the service in foreground mode running with the profiler:

   ```
   python -m cProfile -o test.profile /usr/bin/xivo-ctid -f
   ```

   This will create a file named `test.profile` when the process terminates.

   To profile xivo-confgend, you must use this command instead of the one above:

   ```
   twistd -p test.profile --profiler=cprofile --savestats -no --python=/usr/bin/xivo-
   ↪confgend
   ```

   Note that profiling multi-threaded program (xivo-agid, xivo-confd) doesn't work reliably.

   The *Debugging Daemons* section documents how to launch the various Wazo services in foreground/debug mode.

4. Examine the result of the profiling:

```
$ python -m pstats test.profile
Welcome to the profile statistics browser.
% sort time
% stats 15
...
% sort cumulative
% stats 15
```

Measuring Code Coverage

Here's an example on how to measure the code coverage of xivo-ctid.

This can be useful when you suspect a piece of code to be unused and you want to have additional information about it.

1. Install the following packages:

   ```
   apt-get install python-pip build-essential python-dev
   ```

2. Install coverage via pip:

   ```
   pip install coverage
   ```

3. Run the program in foreground mode with `coverage run`:

   ```
   service monit stop
   service xivo-ctid stop
   coverage erase
   coverage run /usr/bin/xivo-ctid -f
   ```

   The *Debugging Daemons* section documents how to launch the various Wazo service in foreground/debug mode.

4. After the process terminates, use `coverage html` to generate an HTML coverage report:

   ```
   coverage html --include='*xivo_cti*'
   ```

   This will generate an `htlmcov` directory in the current directory.

5. Browse the coverage report.

   Either copy the directory onto your computer and open it with a web browser, or start a web server on the Wazo:

   ```
   cd htmlcov
   python -m SimpleHTTPServer
   ```

   Then open the page from your computer (i.e. not on the Wazo):

   ```
   firefox http://<wazo-hostname>:8000
   ```

External Links

- Official python documentation
- PyMOTW
- coverage.py

1.14.9 Style Guide

Syntax

License

Python files start with a UTF8 encoding comment and the GPLv3 license. A blank line should separate the license from the imports

Example:

```
# -*- coding: utf-8 -*-
# Copyright 2016 The Wazo Authors  (see the AUTHORS file)
# SPDX-License-Identifier: GPL-3.0+

import argparse
```

Spacing

- Lines should not go further than 80 to 100 characters.
- In python, indentation blocks use 4 spaces
- In PHP, indentation blocks use tabs
- Imports should be ordered alphabetically
- Separate module imports and `from` imports with a blank line

Example:

```
import argparse
import datetime
import os
import re
import shutil
import tempfile

from StringIO import StringIO
from urllib import urlencode
```

PEP8

When possible, use pep8 to validate your code. Generally, the following errors are ignored :

- E501 (max 80 chars per line)

Example:

```
pep8 --ignore=E501 xivo_cti
```

When possible, avoid using backslashes to separate lines.

Bad Example:

```
user = session.query(User).filter(User.firstname == firstname)\
                          .filter(User.lastname == lastname)\
                          .filter(User.number == number)\
                          .all()
```

Good Example:

```
user = (session.query(User).filter(User.firstname == firstname)
                           .filter(User.lastname == lastname)
                           .filter(User.number == number)
                           .all())
```

Strings

Avoid using the + operator for concatenating strings. Use string interpolation instead.

Bad Example:

```
phone_interface = "SIP" + "/" + username + "-" + password
```

Good Example:

```
phone_interface = "SIP/%s-%s" % (username, password)
```

Comments

Redundant comments should be avoided. Instead, effort should be put on making the code clearer.

Bad Example:

```
#Add the meeting to the calendar only if it was created on a week day
#(monday to friday)
if meeting.day > 0 and meeting.day < 7:
    calendar.add(meeting)
```

Good Example:

```
def created_on_week_day(meeting):
    return meeting.day > 0 and meeting.day < 7

if created_on_week_day(meeting):
    calendar.add(meeting)
```

Conditions

Avoid using parenthesis around if statements, unless the statement expands on multiple lines or you need to nest your conditions.

Bad Examples:

```
if(x == 3):
    print "condition is true"
```

```
if(x == 3 and y == 4):
    print "condition is true"
```

Good Examples:

```
if x == 3:
    print "condition is true"

if x == 3 and y == 4:
    print "condition is true"

if (extremely_long_variable == 3
    and another_long_variable == 4
    and yet_another_variable == 5):

    print "condition is true"

if (2 + 3 + 4) - (1 + 1 + 1) == 6:
    print "condition is true"
```

Consider refactoring your statement into a function if it becomes too long, or the meaning isn't clear.

Bad Example:

```
if price * tax - bonus / reduction + fee < money:
    product.pay(money)
```

Good Example:

```
def calculate_price(price, tax, bonus, reduction, fee):
    return price * tax - bonus / reduction + fee

final_price = calculate_price(price, tax, bonus, reduction, fee)

if final_price < money:
    product.pay(money)
```

Naming

- Class names are in `CamelCase`
- File names are in `lower_underscore_case`

Conventions for functions prefixed by *find*:

- Return None when nothing is found
- Return an object when a single entity is found
- Return the first element when multiple entities are found

Example:

```
def find_by_username(username):
    users = [user1, user2, user3]
    user_search = [user for user in users if user.username == username]

    if len(user_search) == 0:
        return None
```

```
    return user_search[0]
```

Conventions for functions prefixed by *get*:

- Raise an Exception when nothing is found
- Return an object when a single entity is found
- Return the first element when multiple entities are found

Example:

```
def get_user(userid):
    users = [user1, user2, user3]
    user_search = [user for user in users if user.userid == userid]

    if len(user_search) == 0:
        raise UserNotFoundError(userid)

    return user_search[0]
```

Conventions for functions prefixed by *find_all*:

- Return an empty list when nothing is found
- Return a list of objects when multiple entites are found

Example:

```
def find_all_users_by_username(username):
    users = [user1, user2, user3]
    user_search = [user for user in users if user.username == username]

    return user_search
```

Magic numbers

Magic numbers should be avoided. Arbitrary values should be assigned to variables with a clear name

Bad example:

```
class TestRanking(unittest.TestCase):

    def test_ranking(self):
        rank = Rank(1, 2, 3)

        self.assertEquals(rank.position, 1)
        self.assertEquals(rank.grade, 2)
        self.assertEquals(rank.session, 3)
```

Good example:

```
class TestRanking(unittest.TestCase):

    def test_ranking(self):
        position = 1
        grade = 2
        session = 3
```

```
        rank = Rank(position, grade, session)

        self.assertEquals(rank.position, position)
        self.assertEquals(rank.grade, grade)
        self.assertEquals(rank.session, session)
```

Tests

Tests for a package are placed in their own folder named "tests" inside the package.

Example:

```
package1/
__init__.py
mod1.py
tests/
    __init__.py
    test_mod1.py
package2/
__init__.py
mod9.py
tests/
    __init__.py
    test_mod9.py
```

Unit tests should be short, clear and concise in order to make the test easy to understand. A unit test is separated into 3 sections :

- Preconditions / Preparations
- Thing to test
- Assertions

Sections are separated by a blank line. Sections that become too big should be split into smaller functions.

Example:

```
class UserTestCase(unittest.TestCase):

    def test_fullname(self):
        user = User(firstname='Bob', lastname='Marley')
        expected = 'Bob Marley'

        fullname = user.fullname()

        self.assertEquals(expected, fullname)

    def _prepare_expected_user(self, firstname, lastname, number):
        user = User()
        user.firstname = firstname
        user.lastname = lastname
        user.number = number

        return user

    def _assert_users_are_equal(expected_user, actual_user):
```

```
        self.assertEquals(expected_user.firstname, actual_user.firstname)
        self.assertEquals(expected_user.lastname, actual_user.lastname)
        self.assertEquals(expected_user.number, actual_user.number)

    def test_create_user(self):
        expected = self._prepare_expected_user('Bob', 'Marley', '4185551234')

        user = create_user('Bob', 'Marley', '4185551234')

        self._assert_users_are_equal(expected, user)
```

Exceptions

Exceptions should not be used for flow control. Raise exceptions only for edge cases, or when something that isn't usually expected happens.

Bad Example:

```
def is_user_available(user):
    if user.available():
        return True
    else:
        raise Exception("User isn't available")

try:
    is_user_available(user)
except Exception:
    disable_user(user)
```

Good Example:

```
def is_user_available(user):
    if user.available():
        return True
    else:
        return False


if not is_user_available(user):
    disable_user(user)
```

Avoid throwing `Exception`. Use one of Python's built-in Exceptions, or create your own custom Exception. A list of exceptions is available on the Python documentation website.

Bad Example:

```
def get_user(userid):
    user = session.query(User).get(userid)

    if not user:
        raise Exception("User not found")
```

Good Example:

```
class UserNotFoundError(LookupError):

    def __init__(self, userid):
```

```
        message = "user with id %s not found" % userid
        LookupError.__init__(self, message)

def get_user(userid):
    user = session.query(User).get(userid)

    if not user:
        raise UserNotFoundError(userid)
```

Never use `except:` without specifying any exception type. The reason is that it will also catch important exceptions, such as `KeyboardInterrupt` and `OutOfMemory` exceptions, making your program unstoppable or continuously failing, instead of stopping when wanted.

Bad Example:

```
try:
    get_user(user_id)
except:
    logger.exception("There was an error")
```

Good Example:

```
try:
    get_user(user_id)
except UserNotFoundError as e:
    logger.error(e.message)
    raise
```

1.14.10 Translating Wazo

French and English are maintained by the Wazo authors. Other languages are provided by the community.

Asterisk and Wazo Prompts

Languages and prompts are recorded by several studios. The information for those languages are:

- French : Super Sonic productions (supersonicprod@wanadoo.fr)
- English : Asterisk voice (allison@theasteriskvoice.com)
- German : ATS studio
- Italian : ATS studio

Prompts transcripts are listed in Transifex (*-prompts). You may translate them there.

The prompts used in Wazo are stored in xivo-sounds git repository. You may also want to *generate your own sound files*.

Wazo Client

All translations are in Transifex (wazo-client). The source language is English. Translations are synchronised with the code before every release.

Web Interface

Translations are currently available in French and English. There are no plans to translate the Web interface in other languages.

1.14.11 Wazo Package File Structure

Package naming

Let's assume we want to organise the files for xivo-confd.

- Git repo name: `xivo-confd`
- Binary file name: `xivo-confd`
- Python package name: `xivo_confd`

```
xivo-confd
|-- bin
|   `-- xivo-confd
|-- contribs
|   `-- docker
|       |-- ...
|       `-- prod
|           `-- ...
|-- debian
|   `-- ...
|-- Dockerfile
|-- docs
|   `-- ...
|-- etc
|   `-- ...
|-- integration-tests
|   `-- ...
|-- LICENSE
|-- README.md
|-- requirements.txt
|-- setup.cfg
|-- setup.py
|-- test-requirements.txt
|-- .travis.yml
`-- xivo_confd
    `-- ...
```

Sources

`etc/` Contains default configuration files.

`docs/` Contains technical documentation for this package: API doc, architecture doc, diagrams, ... Should be in RST format using Sphinx.

`bin/` Contains the binaries. Not applicable for pure libraries.

`integration_tests/` Contains the tests bigger than unit-tests. Tests should be runnable simply, e.g. `nosetests integration_tests`.

`README.md` Read me in markdown (Github flavor).

LICENSE License (GPLv3)

.travis.yml Travis CI configuration file

Python

Standard files:

- setup.py
- setup.cfg
- requirements.txt
- test-requirements.txt
- xivo_confd/ (the main sources)

Debian

debian/ Contains the Debian packaging files (`control`, `rules`, ...)

Docker

Dockerfile Used to build a docker image for a working production version

contribs/docker/prod/ Contains the files necessary for running xivo-confd inside a production Docker image

contribs/docker/other/ Contains the Dockerfile and other files to run xivo-confd inside Docker with specific configuration

File naming

- PID file: `/var/run/xivo-confd/xivo-confd.pid`
- WSGI socket file: `/var/run/xivo-confd/xivo-confd.sock`
- Config file: `/etc/xivo-confd/config.yml`
- Log file: `/var/log/xivo-confd.log`
- Static data files: `/usr/share/xivo-confd`
- Storage data files: `/var/lib/xivo-confd`

Component specific information:

1.14.12 Wazo Client

Building the Wazo Client

Building the Wazo Client for Windows platforms

This page explains how to build an executable of the Wazo Client from its sources for Windows. It uses cross-compilation from a Docker container, so you do not need a Windows machine to build. The following instructions are made to run on a Linux host.

Prerequisites

- Docker installed on your machine, and access to Internet.
- A copy of the Wazo Client source code including the Git repository. This directory must be clean of binary files (`.o`, `.dll`, `.exe`, etc.).

Building the package

Go to the `wazo-client-qt` cloned directory and run (first step will take a long time, about 1 hour):

```
docker build -t wazopbx/windows-x86-qt:5.5.0 -f packaging/windows/Dockerfile-windows-
↪x86-qt packaging/windows
docker build -t wazopbx/windows-x86-wazo-client-qt:qt-5.5.0 -f packaging/windows/
↪Dockerfile-windows-x86-wazo-client-qt packaging/windows
docker rm wazo-client-qt-build-win32
docker run --name wazo-client-qt-build-win32 -v $PWD:/usr/src/wazo-client-qt wazopbx/
↪windows-x86-wazo-client-qt:qt-5.5.0
mkdir -p ./bin
docker cp wazo-client-qt-build-win32:/usr/src/wazo-client-qt-build/bin/package-win32 .
↪/bin
```

You will find the installer ready to install in `./bin/package-win32`.

Build options

For more options, you have to edit `packaging/windows/Dockerfile-windows-x86-wazo-client-qt`.

To add a console:

```
qmake CONFIG+=console
```

To generate debug symbols:

```
make DEBUG=yes
```

Clean

```
make distclean
```

Building the Wazo Client on GNU/Linux platforms

This page explains how to build an executable of the Wazo Client from its sources for GNU/Linux.

Prerequisites

- Qt5 library development files: Qt website (Ubuntu packages `qt5-default qt5-qmake qttools5-dev-tools qttools5-dev libqt5svg5-dev`). The currently supported Qt version is 5.5.0.
- openGL development library - libGL (Debian package `libgl1-mesa-dev`)

- Git (Debian package `git`)
- Generic software building tools : `make`, `g++` ... (Debian package `build-essential`)

Get sources

In a bash shell:

```
$ git clone git://github.com/wazo-pbx/wazo-client-qt.git
```

Building

You need to have the Qt5 binaries (qmake, lrelease, ...) in your $PATH.

Launch qmake to generate the Makefile:

```
$ cd wazo-client-qt
$ /path/to/qt5/bin/qmake
```

This will also generate a file `versions.mak` that contains version informations about the code being compiled. It is necessary for compilation and packaging.

You can then launch `make`:

```
$ make
```

Binaries are available in the `bin` directory.

The version of the executable is taken from the `git describe` command.

Build options

To generate debug symbols:

```
$ make DEBUG=yes
```

To compile the unit tests of the Wazo Client:

```
$ qmake CONFIG+=tests
```

or, if you have a recent version of Google Mock:

```
$ qmake CONFIG+=tests CONFIG+=gmock
```

To compile the Wazo Client ready for functional tests:

```
$ make FUNCTESTS=yes
```

Cleaning

```
$ make distclean
```

Launch

You can launch the built executable with:

```
$ LD_LIBRARY_PATH=bin bin/wazoclient
```

Package

To create the Debian package, usable on Debian and Ubuntu, you first need to modify `build-deps` to locate the Qt 5 installation directory:

```
$ /path/to/qt5/bin/qmake -spec linux-g++
$ make
$ make pack
```

This will result in a `.deb` file in the current directory.

The version of the package is taken from the `git describe` command.

Building the Wazo Client on Mac OS

This page explains how to build an executable of the Wazo Client from its sources for Mac OS.

Mac OS Prerequisites

Developer tools

You will need an Apple developer account to get development tools, such as GCC. To log in or sign in, go to the Developer portal of Apple. In the Downloads section, get the Command line Tools for XCode and install them. You might want to get XCode too, but it is rather big.

Qt SDK

You need the development files of the Qt 5 library, available on the Qt website. The currently supported Qt version is 5.5.0.

Get sources

In a bash shell, enter:

```
$ git clone git://github.com/wazo-pbx/wazo-client-qt.git
```

Building

Launch qmake to generate the Makefile:

```
$ cd wazo-client-qt
$ /path/to/qt5/bin/qmake
```

This will also generate a file `versions.mak` that contains version informations about the code being compiled. It is necessary for compilation and packaging.

You can then launch `make`:

```
$ make
```

Binaries are available in the `bin` directory.

The version of the executable is taken from the `git describe` command.

Debug build

Add `DEBUG=yes` on the command line:

```
$ make DEBUG=yes
```

Cleaning

```
$ make distclean
```

Launch

You can launch the built executable with:

```
$ DYLD_LIBRARY_PATH=bin bin/wazoclient.app/Contents/MacOS/wazoclient
```

Package

You need to have the bin directory of Qt in your $PATH.

To create the app bundle:

```
$ make pack
```

This will result in a `.dmg` file in the current directory.

The version of the package is taken from the `git describe` command.

Coding the Wazo Client

Project folder map

baselib

The folder *baselib* contains all files necessary to build the baselib. It contains the necessary code and data structures to communicate with the Wazo CTI server.

This library is designed to be reusable by other Wazo CTI clients. If you want to build it without the rest of the Wazo Client, go in its folder and type:

```
$ qmake && make
```

The library will be available in the new bin folder.

wazoclient

The folder *wazoclient* contains all other source files included in the Wazo Client.

src contains the source code files, *images* contains the images, *i18n* contains the translation files and *qtaddons* contains some Qt addons used by the Wazo Client.

src

The source files are separated in three categories :

- the Wazo Client itself, the source files are directly in *src*.
- the XLet library (*xletlib*) contains the code common to multiple XLets (plugins), like the XLet base class and mainly GUI stuff.
- the XLets themselves (*xlets*), each one is in a *xlets/something* subfolder.

Each XLet is compiled into a dynamic library, but some XLets are still compiled within the wazoclient executable instead of in a separated library. They are marked with a **-builtin* subfolder name.

delivery

This folder contains all license informations necessary for the Wazo Client to be redistributed, i.e. the GNU GPLv3 and the additional requirements.

Configuration access

The settings of the application are stored in BaseEngine for runtime and in files when the client is closed :

- *~/.config/Wazo* on GNU/Linux systems
- (what about other platforms?)

There are now 3 sets of functions from BaseEngine that you can use to read/store settings :

getConfig() / setConfig()

They are proxy methods to use the BaseConfig object inside BaseEngine. They use QVariantMap to store the settings values. They are currently used to store/retrieve options used in the ConfigWidget.

You can find the available keys to access data in the detailed Doxygen documentation of BaseEngine, or in *baseengine.h*.

Note that the settings stored in BaseConfig won't be written in the configuration file if BaseEngine is not aware of their existence (loaded in *loadSettings* and saved in *saveSettings*).

getSettings()

Through this function, you can access the lowest level of configuration storage, QSettings. It also contains the options stored in BaseConfig, but is less easy to use.

This direct access is used for purely graphical settings, only used to remember the appearance of the GUI until the next launch. These settings don't have to be shared with other widgets, and storing them directly in QSettings avoids writing code to import/export to/from BaseConfig.

getProfileSetting() / setProfileSetting()

This pair of methods allow you to read/write settings directly in QSettings, but specifically for the current configuration profile.

Configuration profiles

When starting Wazo Client with an argument, this argument is interpreted as a profile name. This profile name allows you to separate different profiles, with different configuration options.

For example, configuration profile "profileA" will auto-connect with user A and password B and "profileB" will not auto-connect, but is set to connect with user C, no password remembered. To invoke these profiles, use :

```
$ wazoclient profileA
$ wazoclient profileB
```

The default configuration profile is default-user.

Recognizing / extracting phone numbers

Of course, working on Wazo Client implies working with phone numbers. But how to interpret them easily, when we are not sure of the format they're in?

You can use the PhoneNumber namespace (*baselib/src/phonenumber.h*) to do that, it contains routines for recognition/extraction of phone numbers, that way you don't have to parse manually.

These subroutines are pretty basic for the moment, if you need/want to improve them, feel free to do it.

Retrieving CTI server infos

Informations are synchronized from the server to the BaseEngine when the client connects.

It is stored in BaseEngine in "lists". It is stored in a format close to the one used to transmit it, so you can see the CTI protocol definition for further documentation.

Each list contains objects of different type. These types are :

- channel
- user
- phone
- trunk
- agent

- queue
- group
- meetme
- voicemail
- queuemember
- parking

Each type corresponds to a class derived from XInfo, e.g. channel infos are stored in ChannelInfo objects.

The basic attributes of all objects are 3 strings: the IPBX ID, the XiVO object ID and the extended ID of the object, which is the two previous attributes linked with a "/".

Listen to IPBX events

If you want your XLet to receive IPBX/CTI events, you can do so by inheriting the IPBXListener interface.

You must specify which type of events you want to listen. This depends of the implemented functions in the CTI server. You can register to listen these events by calling the IPBXListener method :

registerListener(xxx);

For now, *xxx*, the event type, can take take the values : * chitchat * history * records_campaign * queuestats

On reception of the specified type of event, BaseEngine will call the *IPBXListener* method *parseCommand(QVariantMap)*.

You should then reimplement this method to make it process the event data, stored in the *QVariantMap* parameter.

The parking XLet

There are two concepts here : * Parked calls: These calls have been parked by a switchboard or an operator. They are waiting to be answered by a specific person, unlike a queue, where calls will be answered by one of the agents of the group associated to the queue. Each parked call is given a phone number so that the call can be answered by everyone.

- Parking lots: They are containers for parked calls. Each parking lot has a phone number, used to identify where to send the call we want to park.

ParkingWidget represents a parking lot and contains a table that stores all parked calls.

Adding new XLets

When you want to add a new XLet, you can use the basic XLetNull, that only prints "Hello World". Here is a little script to accelerate the copy from XLetNull.

```
#!/usr/bin/env sh

newname="newname" # Replaces xletnull
NewName="NewName" # Replaces XLetNull & XletNull
NEWNAME="NEWNAME" # Replaces XLETNULL

if [ ! -d xletnull ] ; then

    echo "Please execute this script in wazo-client-qt/wazoclient/src/xlets"
    echo $newname
```

```
    exit 1
fi

cp -r xletnull $newname
cd $newname
rm -f moc* *.o Makefile

for f in $(find . -type f -print) ; do
    mv $f `echo $f | sed s/xletnull/$newname/`
done

find . -type f -exec sed -i "s/xletnull/$newname/g;s/X[Ll]etNull/$NewName/g;s/
↪XLETNULL/$NEWNAME/g" {} \;</nowiki>
```

Before executing the script, just replace the first three variables with the name of the new XLet.

Then, you must add a line in wazoclient/xlets.pro to add your new directory to the SUBDIRS variable.

Then you can start implementing your new class. The <xletname>Plugin class is only an interface between the main app and your XLet.

Translations

If you want to localize your XLet, there are four steps.

Modify the sources

In the <xletname>Plugin constructor, add the line :

b_engine->registerTranslation(":/<xletname>_%1");

before the return instruction.

Modify the project file

Add these lines in the .pro file in your XLet directory :

TRANSLATIONS = <xletname>_fr.ts TRANSLATIONS += <xletname>_nl.ts

RESOURCES = res.qrc

Replace fr and nl with the languages you want.

Create the resource file

In a file res.qrc in your XLet directory, put these lines :

```
<!DOCTYPE RCC><RCC version="1.0">
    <qresource>
        <file><xletname>_fr.qm</file>
        <file><xletname>_nl.qm</file>
    </qresource>
</RCC>
```

These files will be embedded in the Xlet library binary.

Create the translation files

In your XLet directory, run :

lupdate <xletname>.pro

This creates as much .ts translation files as specified in the .pro file. You can now translate strings in these file.

The XLet will now be compiled and translated.

Add a new XLet

For now, it is not possible to add easily an XLet without changing the CTI server configuration files.

If you just want to test your new XLet, you can add the following line in baseengine.cpp :

m_capaxlets.push_back(QVariantList() << QVariant("<xletname>") << QVariant("tab"));

right after the line

m_capaxlets = datamap.value("capaxlets").toList();

You can replace "tab" with "grid" or "dock".

Add a translation

This is definitely not something funny and not easy to automatize.

You have to add, in every .pro file of the project (except xlets.pro and all those that don't need translations), a line

TRANSLATIONS += <project>_<lang>.ts

Replace <project> with the project name (wazoclient, baselib, xlet) and <lang> by the identifier of your language (en, fr, nl, . . .) Then you have to add, in every .qrc file, the .qm files corresponding to the ones you added in the .pro files, such as :

<file><project>_<lang>.qm</file>

in the <qresource> section of these XML .qrc files.

After that, you have to run, in the Wazo Client root directory, something like :

find . -name *.pro -exec lupdate {} ;

This will create or update all .ts translation files registered in the .pro files.

You can then start translating the strings in these files, in the *wazoclient/i18n* folder.

Code modification

If you want to be able to select your new language from within the Wazo Client, you have to add it in the interface.

For that, you can add your new language in the *m_locale_cbox* QCombobox in ConfigWidget.

CTI debugging tool

If you have a problem and you want to see what is going on between the CTI server and client, you can use a specific script, designed specifically for Wazo, instead of using something like Wireshark to listen network communications.

Profiling

To get profiling informations on the Wazo Client:

- Compile the Wazo Client with debugging symbols
- Run the command:

```
LD_LIBRARY_PATH=bin valgrind --tool=callgrind bin/wazoclient
```

- Quit the client
- Open the generated file `callgrind.out.<pid>` with KCacheGrind

Automatic checking tools

We use two tools to check the source code of the Wazo Client: CppCheck et Valgrind.

CppCheck

Usage:

```
cppcheck -I baselib/src -I wazoclient/src .
```

Valgrind (Memcheck)

Usage:

```
LD_LIBRARY_PATH=bin valgrind --leak-check=full --suppressions=valgrind.supp --num-
↪callers=30 --gen-suppressions=yes bin/wazoclient
```

You need to fill a file `valgrind.supp` with Valgrind suppressions, to avoid displaying errors in code you have no control over.

Here is a template `valgrind.supp` you can use. All memory in the Wazo Client is allocated using the new operator, so all calls to `malloc` and co. must come from libraries:

```
{
   malloc
   Memcheck:Leak
   fun:malloc
   ...
}

{
   calloc
   Memcheck:Leak
   fun:calloc
   ...
}

{
   realloc
   Memcheck:Leak
```

```
   fun:realloc
   ...
}

{
   memalign
   Memcheck:Leak
   fun:memalign
   ...
}
```

Figures

Here's a call graph for the presence features. Not complete, but gives a good global view of the internal mechanism.

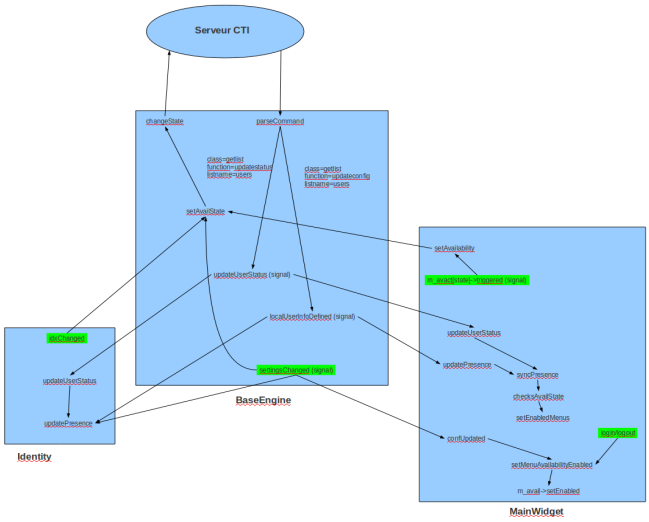


Fig. 1.105: Wazo Client presence call graph

Here's a call graph describing the chaining of calls when the Wazo Client connects to the server.

Manage Translations of the Wazo Client

This sections describes how to manage Wazo Client translations from a developer point of view. If you want to help translate the Wazo Client, see *Translating Wazo*

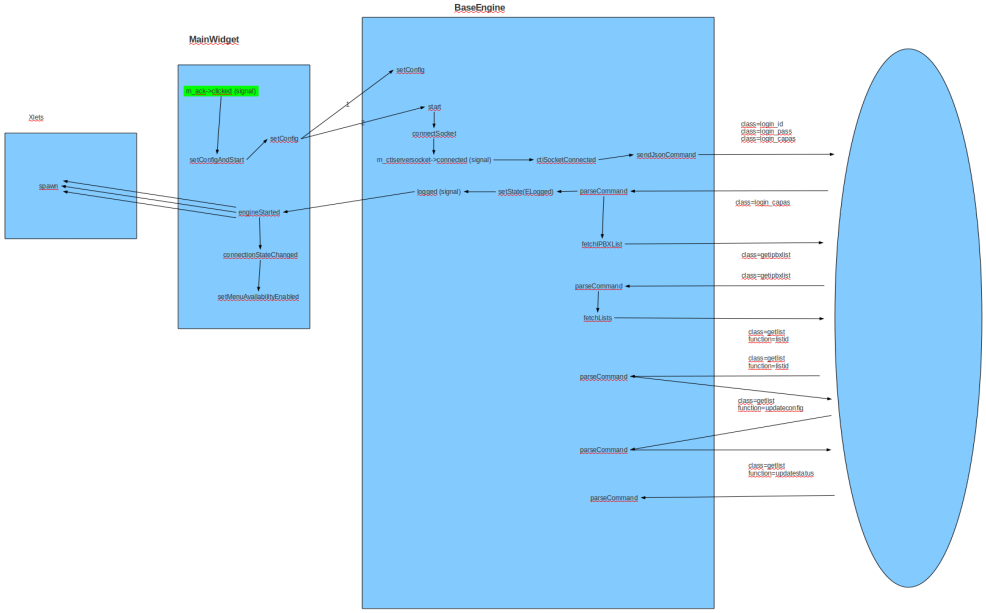


Fig. 1.106: Wazo Client login call graph

You need to install these tools:

```
pip install transifex-client
apt-get install qt4-dev-tools
```

How to Add a New Translated String

String to be translated is marked using the tr macro in the source code.

Example:

```
tr("Number");
```

Updating translations on transifex

Run the following commands from the root of the wazo-client-qt project:

```
make pushtr
```

After this command, you can visit Transifex, and check that the wazo-client is 100% translated for your language. Once all the translations have been checked, run the 3 following commands:

```
make pulltr
git commit
git push
```

Warning: Under Arch Linux, you must have qt5 installed and prepend `QT_PATH=/usr/bin` before `make {pull,push}tr`.

Add a new Wazo Client locale

Localizing the Wazo Client goes through four steps :

- Creating the new translation in Transifex
- Generatint the translation files
- Embedding the translation in the binaries
- Displaying the new locale to be chosen

Creating the new translation in Transifex

Log into Transifex and click the `Create language` option.

Generate translation files

The translation files will be automatically generated from the source code.

For the command to create files for your locale, you need to ensure it is listed in the project file.

There are a few project files you should edit, each one will translate a module of the Wazo Client :

- `baselib/baselib.pro`
- `wazoclient/wazoclient.pro`
- `wazoclient/xletlib.pro`
- `wazoclient/src/xlets/*/*.pro`

In these files, you should add a line like this one:

```
TRANSLATIONS += $$ROOT_DIR/i18n/wazoclient_fr.ts
```

This line adds a translation file for french. Please replace fr by the code of your locale. The `$$ROOT_DIR` variable references either wazoclient or baselib.

You can use a command like the following to automate this ($LANG is the new language):

```
find . -name '*.pro' -exec sed -i -e 's|^TRANSLATIONS += $${\?ROOT_DIR}\?/i18n/\(.*\)_
↪en.ts|\0\nTRANSLATIONS += $$ROOT_DIR/i18n/\1_$LANG.ts|' {} \;
```

To actually create the files, you will have to use the translation managing script. But first, you must tell the script about your new locale. Edit the `utils/translations.sh` file and add your locale to the `LOCALES` variable. Then, you can run the script:

```
$ make pulltr
```

Embed the translation files

For each project previously edited, you should have a corresponding `.qrc` file. These resource files list all files that will be embedded in the Wazo Client binaries. You should then add the corresponding translation files like below:

```
<file>obj/wazoclient_fr.qm</file>
```

This embeds the French translation of the `wazoclient` module, corresponding to the translation file above. The path is changed to `obj/` because the `.qm` file will be generated from the `.ts` file.

You can use a command like the following to automate this ($LANG is the new language):

```
find . -name '*.qrc' -exec sed -i -e 's|^\( *\)<file>\(.*\)obj/\(.*\)_fr.qm</file>
↪|\0\n\1<file>\2obj/\3_$LANG.qm</file>|' {} \;
```

Display the new locale

You have to edit the source file `wazoclient/src/configwidget.cpp` and add the entry corresponding to your locale in the locale-choosing combobox.

1.14.13 CTI Server

This section describes the informations and tools for CTI Server.

CTI Proxy

Here's how to run the various CTI client-server development/debugging tools. These tools can be found on GitHub, in the Wazo project.

You can get the scripts by using Git:

```
$ git clone https://github.com/wazo-pbx/xivo-tools.git
```

General Information

Both the ctispy, ctisave and ctistat tools work in a similar way. They both are proxies that need to be inserted between the CTI client and the CTI server message flow.

To do this, you first start the given tool on your development machine, giving it the CTI server hostname as the first argument. You then configure your CTI client to connect to the tool on port 50030 (notice the trailing 0). The tool should then accept the connection from the client, and once this is done, will make a connection to the server, thereby being able to process all the information sent between the client and the server.

In the following examples, we suppose that the CTI server is located on the host named wazo-new.

Tools

ctispy

`ctispy` can be used to see the message flow between the client and the server in "real-time".

The simplest invocation is:

```
$ cti-proxy/ctispy wazo-new
```

You can pretty-print the messages if you want by using the `--pretty-print` option:

```
$ cti-proxy/ctispy wazo-new --pretty-print
```

By default, each message is displayed separately even though more than one message can be in a single TCP packet. You can also use the `--raw` option if you want to see the raw traffic between the client and the server:

```
$ cti-proxy/ctispy wazo-new --raw
```

Note that when using the `--raw` option, some other option doesn't work because the messages are not decoded/analyzed.

If you want to remove some fields from the messages, you can use the `--strip` option:

```
$ cti-proxy/ctispy wazo-new --strip timenow --strip commandid --strip replyid
```

If you want to see only messages matching a certain key and value, use the `--include` option:

```
$ cti-proxy/ctispy wazo-new --include class=getlist
```

Finally, you can ignore all the messages from the client or the server by using the `--no-client` or `--no-server` option respectively.

By default, ctispy will exit after the connection with the client is closed. You can bypass this behavior with the `--loop` option, that will make the CTI proxy continue, whether the client is connected or not.

ctisave

`ctisave` save the messages from the client and the server in two separate files. This is useful to do more careful post-analysis.

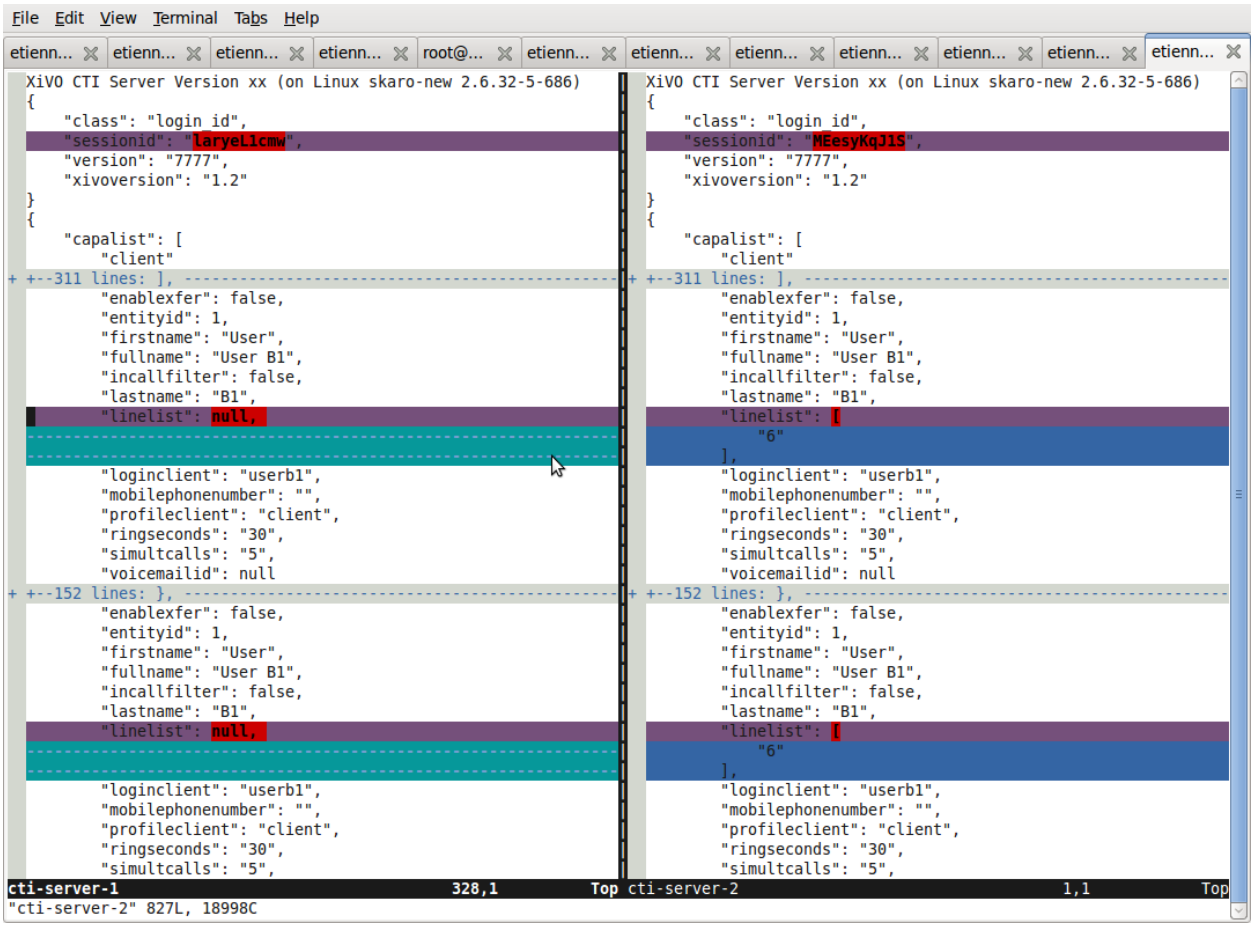
The simplest invocation is:

```
$ cti-proxy/ctisave wazo-new /tmp/cti-client /tmp/cti-server
```

To do comparison, it's often useful to strip some fields:

```
$ cti-proxy/ctisave wazo-new /tmp/cti-client /tmp/cti-server --strip timenow
--strip commandid --strip replyid
```

One useful thing to do with files generated from different ctisave invocation is to compare them with a tool like vimdiff, for example:



ctistat

`ctistat` display various statistic about a CTI "session" when it ends.

The simplest invocation is:

```
$ cti-proxy/ctistat wazo-new
```

CTI Protocol

Protocol Changelog

The versions below indicate the Wazo version followed by the protocol version.

Warning: The CTI server protocol is subject to change without any prior warning. If you are using this protocol in your own tools please be sure to check that the protocol did not change before upgrading Wazo

17.16 - 2.3

- the *Login ID* now uses the *wazoversion* instead of the *xivoversion*

16.11 - 2.2

- the *user_id* field has been added back to the *User status update*

16.09 - 2.2

- the *Register user status update* now uses the *user_uuid* instead of the *user_id*
- the *User status update* now uses the *user_uuid* instead of the *user_id*

16.04 - 2.1

- the *Chitchat* command *to* and *from* fields are now a list of two strings, *xivo_uuid* and *user_uuid*.

16.01 - 2.0

- the *lastconnswins* field has been removed from the *Login capas* command
- the *loginkind* field has been removed from the *Login capas* command
- the *ipbxcommands* and *regcommands* capakinds have been removed from *Login capas* command
- the *Login password* command has been modified. The *hashedpassword* has been replaced by the *password* field which is now sent verbatim.

15.20 - 1.2

- the *STARTTLS* command has been added

15.19 - 1.2

- the *Chitchat* command *to* field is now a list of two elements, *xivo_uuid* and *user_id*.
- the `getlist` command has been removed for the *channels* listname.

- many fields have been removed from the `getlist` command.
  - users list
    - enableclient
    - profileclient
  - phones
    - context
    - protocol
    - simultcalls
    - channels
  - voicemails
    - email
    - fullname
    - old
    - waiting
  - agents
    - phonenumber
- some ipbxcommands have been removed:
  - mailboxcount
  - atxfer
  - transfer
  - hangup
  - originate

15.18 - 1.2

- add the *Attended transfer to voicemail* command
- add the *Blind transfer to voicemail* command
- the *Send fax* command now include the size and data field.
- the *filetransfer* command has been removed.

15.16 - 1.2

- the *Get relations* command was added.
- the *Relations* message was added.

15.14 - 1.2

- the `people_purge_personal_contacts` message was added.
- the `people_personal_contacts_purged` message was added.
- the `people_personal_contact_raw` message was added.
- the `people_personal_contact_raw_result` message was added.
- the `people_edit_personal_contact` message was added.
- the `people_personal_contact_raw_update` message was added.
- the `people_import_personal_contacts_csv` message was added.
- the `people_import_personal_contacts_csv_result` message was added.
- the `people_export_personal_contacts_csv` message was added.
- the `people_export_personal_contacts_csv_result` message was added.
- for messages `people_personal_contact_deleted` and `people_favorite_update` there are no longer `data` sub-key.

15.13 - 1.2

- for `channel status update` message:
  - the value of `commstatus` have been changed from `linked-caller` and `linked-called` to `linked`.
  - the key `direction` have been removed.
  - the key `talkingto_kind` have been removed.
- the `people_personal_contacts` message was added.
- the `people_personal_contacts_result` message was added.
- the `people_create_personal_contact` message was added.
- the `people_personal_contact_created` message was added.
- the `people_delete_personal_contact` message was added.
- the `people_personal_contact_deleted` message was added.

15.12 - 1.2

- `people_search_result` has a new key in `relations`: `source_entry_id`
- the `people_favorites` message was added.
- the `people_favorites_result` message was added.
- the `people_set_favorite` message was added.
- the `people_favorite_update` message was added.

15.11 - 1.2

- the `fax_progress` message was added.

15.09 - 1.2

- for messages of class `history` the client cannot request by mode anymore. The server returns all calls and the mode is now metadata for each call.

14.24 - 1.2

- for messages of class `ipbxcommand`, the command `record` and `sipnotify` have been removed.
- the `logfromclient` message has been removed

14.22 - 1.2

- for messages of class `faxsend`, the steps `file_decoded` and `file_converted` have been removed.

14.06 - 1.2

- the `dial_success` message was added

14.05 - 1.2

- the `unhold_switchboard` command was renamed `resume_switchboard`.

13.22 - 1.2

- the `actionfiche` message was renamed `call_form_result`.

13.17 - 1.2

- for messages of class `login_capas` from server to client: the key `presence` has been removed.

13.14 - 1.2

- for messages of class `getlist`, list `agents` and function `updatestatus`: the key `availability` in the `status` object/dictionary has changed values:
  - deleted values: `on_call_non_acd_incoming` and `on_call_non_acd_outgoing`
  - added values: * `on_call_non_acd_incoming_internal` * `on_call_non_acd_incoming_external` * `on_call_non_acd_outgoing_internal` * `on_call_non_acd_outgoing_external`

13.12 - 1.2

- for messages of class `getlist`, list `agents` and function `updatestatus`: the key `availability` in the `status` object/dictionary has changed values:
  - deleted value: `on_call_non_acd`
  - added values: `on_call_non_acd_incoming` and `on_call_non_acd_outgoing`

13.10 - 1.2

- for messages of class `getlist` and function `updateconfig`, the `config` object/dictionary does not have a `rules_order` key anymore.

Commands

Objects have the format: "<type>:<xivoid>/<typeid>"

- <type> can take any of the following values: user, agent, queue, phone, group, meetme, ...
- <xivoid> indicates on which server the object is defined
- <typeid> is the object id, type dependant

e.g. user:xivo-test/5 I'm looking for the user that has the ID 5 on the xivo-test server.

Here is a non exaustive list of types:

- exten
- user
- vm_consult
- voicemail

Agent

Login agent

`Client -> Server`

```
{"agentphonenumber": "1000", "class": "ipbxcommand", "command": "agentlogin",
↪"commandid": 733366597}
```

agentphonenumber is the physical phone set where the agent is going to log on.

`Server > Client`

- Login successfull :

```
{"function": "updateconfig",
 "listname": "queuemembers",
 "tipbxid": "xivo",
 "timenow": 1362664323.94,
 "tid": "Agent/2002,blue",
 "config": {"paused": "0",
            "penalty": "0",
```

```
            "membership": "static",
            "status": "1",
            "lastcall": "",
            "interface": "Agent/2002",
            "queue_name": "blue",
            "callstaken": "0"},
 "class": "getlist"}

{"function": "updatestatus",
 "listname": "agents",
 "tipbxid": "xivo",
 "timenow": 1362664323.94,
 "status": {"availability_since": 1362664323.94,
            "queues": [],
            "on_call": false,
            "availability": "available",
            "channel": null},
   "tid": 7,
   "class": "getlist"}
```


- The phone number is already used by an other agent :


```
{"class": "ipbxcommand", "error_string": "agent_login_exten_in_use", "timenow":␣
↪1362664158.14}
```


Logout agent

```
Client -> Server
```


```
{"class": "ipbxcommand", "command": "agentlogout", "commandid": 552759274}
```


Pause

On all queues

```
Client -> Server
```


```
{"class": "ipbxcommand", "command": "queuepause", "commandid": 859140432, "member":
↪"agent:xivo/1", "queue": "queue:xivo/all"}
```


Un pause agent

On all queues

```
Client -> Server
```


```
{"class": "ipbxcommand", "command": "queueunpause", "commandid": 822604987, "member":
↪"agent:xivo/1", "queue": "queue:xivo/all"}
```


Add an agent in a queue

```
Client -> Server
```

```
{"class": "ipbxcommand", "command": "queueadd", "commandid": 542766213, "member":
↪"agent:xivo/3", "queue": "queue:xivo/2"}
```

Remove an agent from a queue

```
Client -> Server
```

```
{"class": "ipbxcommand", "command": "queueremove", "commandid": 742480296, "member":
↪"agent:xivo/3", "queue": "queue:xivo/2"}
```

Listen to an agent

```
Client -> Server
```

```
{"class": "ipbxcommand", "command": "listen", "commandid": 1423579492, "destination":
↪"xivo/1", "subcommand": "start"}
```

Configuration

The following messages are used to retrieve Wazo configuration.

Common fields

- class : getlist
- function : listid
- commandid
- tipbxid
- listname : Name of the list to be retreived : users, phones, agents, queues, voicemails, queuemembers

```
{
   "class": "getlist",
   "commandid": 489035169,
   "function": "listid",
   "tipbxid": "xivo",
   "listname": "........."
}
```

Users configuration

Return a list of configured user id's

```
Client -> Server
```

```
{"class": "getlist", "commandid": 489035169, "function": "listid", "listname": "users
↪", "tipbxid": "xivo"}
```

```
Server -> Client
```

```
{
   "class": "getlist",
   "function": "listid", "listname": "users",
   "list": ["11", "12", "14", "17", "1", "3", "2", "4", "9"],
   "tipbxid": "xivo","timenow": 1362735061.17
   }
```

User configuration

Return a user configuration

- tid is the userid returned by *Users configuration* message

`Client -> Server`

```
{
  "class": "getlist",
  "function": "updateconfig",
  "listname": "users",
  "tid": "17",
  "tpbxid": "xivo",  "commandid": 5}
```

`Server -> Client`

```
{
   "class": "getlist",
   "function": "updateconfig",
   "listname": "users",
   "tid": "17",
   "tipbxid": "xivo",
   "timenow": 1362741166.4,
   "config": {
         "enablednd": 0, "destrna": "", "enablerna": 0,  "enableunc": 0, "destunc": "
↪", "destbusy": "", "enablebusy": 0, "enablexfer": 1,
         "firstname": "Alice",  "lastname": "Bouzat", "fullname": "Alice Bouzat",
         "voicemailid": null, "incallfilter": 0,  "enablevoicemail": 0,   "agentid":␣
↪2, "linelist": ["7"], "mobilephonenumber": ""}
    }
```

Phones configuration

`Client -> Server`

```
{"class": "getlist", "commandid": 495252308, "function": "listid", "listname": "phones
↪", "tipbxid": "xivo"}
```

`Server > Client`

```
{"class": "getlist", "function": "listid", "list": ["1", "3", "2", "5", "14", "7", "6
↪", "9", "8"],
   "listname": "phones", "timenow": 1364994093.38, "tipbxid": "xivo"}
```

Individual phone configuration request:

```
{"class": "getlist", "commandid": 704096693, "function": "updateconfig", "listname":
↪"phones", "tid": "3", "tipbxid": "xivo"}
```

Server > Client

```
{"class": "getlist",
   "config": {"allowtransfer": null, "identity": "SIP/ihvbur", "iduserfeatures": 1,
                  "initialized": null, "number": "1000"},
   "function": "updateconfig", "listname": "phones", "tid": "3", "timenow":␣
↪1364994093.43, "tipbxid": "xivo"}
```

Agents configuration

Client -> Server

```
{"class": "getlist", "commandid": 1431355191, "function": "listid", "listname":
↪"agents", "tipbxid": "xivo"}
```

Queues configuration

Client -> Server

```
{"class": "getlist", "commandid": 719950939, "function": "listid", "listname": "queues
↪", "tipbxid": "xivo"}
```

Server -> Client

```
{"function": "listid", "listname": "queues", "tipbxid": "xivo",
      "list": ["1", "10", "3", "2", "5", "4", "7", "6", "9", "8"], "timenow":␣
↪1382704649.64, "class": "getlist"}
```

Queue configuration

tid is the id returned in the list field of the getlist response message

Client -> Server

```
{"commandid":7,"class":"getlist","tid":"3","tipbxid":"xivo","function":"updateconfig",
↪"listname":"queues"}
```

Server -> Client

```
{
 "function": "updateconfig", "listname": "queues", "tipbxid": "xivo", "timenow":␣
↪1382704649.69, "tid": "3",
   "config":
      {"displayname": "red", "name": "red", "context": "default", "number": "3002"},
 "class": "getlist"}
```

Voicemails configuration

```
Client -> Server
```

```
{"class": "getlist", "commandid": 1034160761, "function": "listid", "listname":
↪"voicemails", "tipbxid": "xivo"}
```

Queue members configuration

```
Client -> Server
```

```
{"class": "getlist", "commandid": 964899043, "function": "listid", "listname":
↪"queuemembers", "tipbxid": "xivo"}
```

```
Server -> Client
```

```
{"function": "listid", "listname": "queuemembers", "tipbxid": "xivo",
   "list": ["Agent/2501,blue", "Agent/2500,yellow", "Agent/2002,yellow", "Agent/2003,_
↪_switchboard",
            "Agent/2003,blue", "Agent/108,blue", "Agent/2002,blue"],
   "timenow": 1382717016.23,
   "class": "getlist"}
```

Fax

Send fax

```
Client -> Server
```

```
{"class": "faxsend",
 "filename": "contract.pdf",
 "destination", 41400,
 "size": 100000,
 "data": "<base64 of the fax content>"}
```

Fax status

```
Server -> Client
```

- pages: number of pages sent (`NULL` if FAILED)
- status
  - FAILED: Failed to send fax.
  - PRESENDFAX: Fax number exist and converting pdf->tiff has been done.
  - SUCCESS: Fax sent with success.

```
{"class": "fax_progress", "status": "SUCCESS", "pages": 2 }
```

Call control commands

Dial

- destination can be any number
- destination can be a pseudo URL of the form "type:ibpx/id"

```
Client -> Server
```

```
{
   "class": "ipbxcommand",
   "command": "dial",
   "commandid": <commandid>,
   "destination": "exten:xivo/<extension>"
}
```

For example :

```
{
    "class": "ipbxcommand",
    "command": "dial",
    "commandid": 1683305913,
    "destination": "exten:xivo/1202"
}
```

The server will answer with either an error or a success:

```
{
    "class": "ipbxcommand",
    "error_string": "unreachable_extension:1202",
}

{
    "class": "dial_success",
    "exten": "1202"
}
```

Attended transfer to voicemail

Transfer the current call to a given voicemail and listen to the message before completing the transfer.

```
Client -> Server
```

```
{
    "class": "attended_transfer_voicemail",
    "voicemail": "<voicemail number>"
}
```

Blind transfer to voicemail

Transfer the current call to a given voicemail.

```
Client -> Server
```

```
{
    "class": "blind_transfer_voicemail",
    "voicemail": "<voicemail number>"
}
```

Login

Once the network is connected at the socket level, the login process requires three steps. If one of these steps is omitted, the connection is reset by the cti server.

- login_id, the username is sent as a login to the cti server, cti server answers by giving a sessionid
- login_pass, the password is sent to the cti server, cti server answers by giving a capaid
- login_capas, the capaid is returned to the server with the user's availability, cti server answers with a list of info relevant to the user

```
{
"commandid": <commandid>,
"class": "login_id",
}
```

- class: defined what class of command use.
- commandid : a unique integer number.

Login ID

`Client -> Server`

```
{
"class": "login_id",
"commandid": 1092130023,
"company": "default",
"ident": "X11-LE-24079",
"lastlogout-datetime": "2013-02-19T11:13:36",
"lastlogout-stopper": "disconnect",
"userlogin": <userlogin>,
"wazoversion": "<cti protocol version>"
}
```

`Server -> Client`

```
{
    "class": "login_id",
    "sessionid": "21UaGDfst7",
    "timenow": 1361268824.64,
    "wazoversion": "<cti protocol version>"
}
```

Note: sessionid is used to calculate the hashed password in next step

Login password

```
Client -> Server
```

```
{
    "class": "login_pass",
    "password": "secret",
    "commandid": <commandid>
}
```

```
Server -> Client
```

```
{
    "capalist": [
        2
    ],
    "class": "login_pass",
    "replyid": 1646064863,
    "timenow": 1361268824.68
}
```

If no CTI profile is defined on XiVO for this user, the following message will be sent:

```
{
    "error_string": "capaid_undefined",
    "class": "login_pass",
    "replyid": 1646064863,
    "timenow": 1361268824.68
}
```

Note: the first element of the capalist is used in the next step login_capas

Login capas

```
Client -> Server
```

```
{
"capaid": 3,
"commandid": <commandid>,
"state": "available",
"class": "login_capas"
}
```

```
Server -> Client
```

First message, describes all the capabilities of the client, configured at the server level

- presence : actual presence of the user
- userid : the user id, can be used as a reference
- **capas**
  - **userstatus** [a list of available statuses]
    - status name

        * color
        * selectionnable status from this status
        * default action to be done when this status is selected
        * long name
    - services : list of availble services
    - phonestatus : list of available phonestatuses with default colors and descriptive names
    - capaxlets : List of xlets configured for this profile
    - appliname

```
{
   "class": "login_capas"
   "presence": "available",
   "userid": "3",
   "ipbxid": "xivo",
   "timenow": 1361440830.99,
   "replyid": 3,
   "capas": {
            "preferences": false,
            "userstatus": {
                        "available": { "color": "#08FD20",
                                       "allowed": ["available", "away", "outtolunch",
↪"donotdisturb", "berightback"],
                                       "actions": {"enablednd": "false"}, "longname":
↪"Disponible"
                                     },
                        "berightback": {  "color": "#FFB545",
                                          "allowed": ["available", "away", "outtolunch
↪", "donotdisturb", "berightback"],
                                          "actions": {"enablednd": "false"}, "longname
↪": "Bient\u00f4t de retour"
                                       },
                        "disconnected": { "color": "#202020",
                                          "actions": {"agentlogoff": ""}, "longname":
↪"D\u00e9connect\u00e9"
                                       },
                       /* a list of other status depends on the cti server␣
↪configuration */
            },
         "services": ["fwdrna", "fwdbusy", "fwdunc", "enablednd"],
         "phonestatus": {
                           "16": {"color": "#F7FF05", "longname": "En Attente"},
                           "1":  {"color": "#FF032D", "longname": "En ligne OU appelle
↪"},
                           "0":  {"color": "#0DFF25", "longname": "Disponible"},
                           "2":  {"color": "#FF0008", "longname": "Occup\u00e9"},
                           "-1": {"color": "#000000", "longname": "D\u00e9sactiv\u00e9
↪"},
                           "4":  {"color": "#FFFFFF", "longname": "Indisponible"},
                           "-2": {"color": "#030303", "longname": "Inexistant"},
                           "9":  {"color": "#FF0526", "longname": "(En Ligne OU␣
↪Appelle) ET Sonne"},
                           "8":  {"color": "#1B0AFF", "longname": "Sonne"}
                        }
      },
```

```
   "capaxlets": [["identity", "grid"], ["search", "tab"], ["customerinfo", "tab", "1
↪"], ["fax", "tab", "2"], ["dial", "grid", "2"], ["tabber", "grid", "3"], ["history",
↪ "tab", "3"], ["remotedirectory", "tab", "4"], ["features", "tab", "5"], ["people",
↪"tab", "6"], ["conference", "tab", "7"]],
   "appliname": "Client",
}
```

Second message describes the current user configuration

```
{
   "function": "updateconfig",
   "listname": "users",
   "tipbxid": "xivo",
   "timenow": 1361440830.99,
   "tid": "3",
   "config": {"enablednd": false},
   "class": "getlist"
}
```

Third message describes the current user status

```
{
   "function": "updatestatus",
   "listname": "users",
   "status": {"availstate": "available"},
   "tipbxid": "xivo",
   "tid": "3",
   "class": "getlist",
   "timenow": 1361440830.99
}
```

Others

call_form_result

This message is received when a *call form* is submitted from a client to the Wazo.

`Client -> Server`

```
{
    "class": "call_form_result",
    "commandid": <commandid>,
    "infos": {"buttonname": "saveandclose",
              "variables": {"XIVOFORM_varname1": "value1",
                            "XIVOFORM_varname2": "value2"}}
}
```

History

- size : Size of the list to be sent by the server

`Client -> Server`

```
{
   "class": "history",
   "commandid": <commandid>
   "size": "8",
   "xuserid": "<xivoid>/<userfeaturesid>",
}
```

`Server > Client`

Send back a table of calls :

- duration in seconds
- extension: caller/destination extension
- fullname: caller ID name
- mode
  - 0 : sent calls
  - 1 : received calls
  - 2 : missed calls

```
{
   "class": "history",
   "history": [
      {"calldate": "2013-03-29T08:44:35.273998",
       "duration": 30.148765,
       "extension": "*844201",
       "fullname": "Alice Wonderland",
       "mode": 0},
      {"calldate": "2013-03-28T16:56:48.071213",
       "duration": 58.134744,
       "extension": "41400",
       "fullname": "41400"}
       "mode": 1},
   ],
   "replyid": 529422441,
   "timenow": 1364571477.33
}
```

Chitchat

`Client > Server`

```
{
   "class": "chitchat",
   "alias": "Alice",
   "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse␣
↪venenatis velit nibh, ac condimentum felis rutrum id.",
   "to": [<xivo_uuid>, <user_uuid>],
   "commandid": <commandid>
}
```

`Server > Client`

The following message is received by the remote Wazo Client

```
{
    "class": "chitchat",
    "from": [<xivo_uuid>, <user_uuid>],
    "to": [<xivo_uuid>, <user_uuid>]
    "alias": "Alice",
    "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse␣
↪venenatis velit nibh, ac condimentum felis rutrum id.",
}
```

Directory

Request directory information, names matching pattern ignore case.

`Client -> Server`

```
{
   "class": "directory",
   "commandid": 1079140548,
   "pattern": "pau"
}
```

`Server > Client`

```
{
   "class": "directory",
   "headers": ["Nom", "Num\u00e9ro", "Mobile", "Autre num\u00e9ro", "E-mail",
↪"Fonction", "Site", "Source"],
   "replyid": 1079140548,
   "resultlist": ["Claire Mapaurtal;;+33644558899;31256;cmapaurtal@societe.com;;;",
                  "Paul Salvadier;+33445236988;+33678521430;31406;psalvadier@societe.
↪com;;;"],
   "status": "ok",
   "timenow": 1378798928.26
}
```

parking

keepalive

availstate

getipbxlist

```
{
    "class": "getipbxlist",
    "commandid": <commandid>
}
```

People

Get relations

This command will trigger a *Relations* message.

`Client -> Server`

```
{
    "class": "get_relations"
}
```

People headers

Client -> Server

```
{
  "class": "people_headers",
}
```

Server -> Client

```
{
  "class": "people_headers_result",
  "column_headers": ["Status", "Name", "Number"],
  "column_types": [null, null, "number"],
}
```

People Search

Client -> Server

```
{
  "class": "people_search",
  "pattern": <pattern>,
}
```

Server -> Client

```
{
  "class": "people_search_result",
  "term": "Bob",
  "column_headers": ["Firstname", "Lastname", "Phone number", "Mobile", "Fax", "Email
↪", "Agent"],
  "column_types": [null, "name", "number_office", "number_mobile", "fax", "email",
↪"relation_agent"],
  "results": [
    {
      "column_values": ["Bob", "Marley", "5555555", "5556666", "5553333",
↪"mail@example.com", null],
      "relations": {
        "agent_id": null,
        "user_id": null,
        "endpoint_id": null,
        "source_entry_id": null
      },
      "source": "my_ldap_directory"
    }, {
      "column_values": ["Charlie", "Boblin", "5555556", "5554444", "5552222",
↪"mail2@example.com", null],
      "relations": {
        "agent_id": 12,
```

```
        "user_id": 34,
        "endpoint_id": 56,
        "source_entry_id": "34"
      },
      "source": "internal"
    }
  ]
}
```

Relations

This message can currently only be received as a response to the *Get relations* command.

- The *xivo_uuid* is the id of the server
- The *user_id* is the id of the current user.
- The *endpoint_id* is the id of the line of the current user or null.
- The *agent_id* is the id of the agent of the current user or null.

`Server -> Client`

```
{
    "class": "relations",
    "data": {"xivo_uuid": <the xivo uuid>,
             "user_id": <the user id>,
             "endpoint_id": <the endpoint id>,
             "agent_id": <the agent id>}
}
```

Favorites list

`Client -> Server`

```
{
  "class": "people_favorites",
}
```

`Server -> Client`

```
{
  "class": "people_favorites_result",
  "column_headers": ["Firstname", "Lastname", "Phone number", "Mobile", "Fax", "Email
↪", "Agent", "Favorites"],
  "column_types": [null, "name", "number_office", "number_mobile", "fax", "email",
↪"relation_agent", "favorite"],
  "results": [
    {
      "column_values": ["Bob", "Marley", "5555555", "5556666", "5553333",
↪"mail@example.com", null, true],
      "relations": {
        "agent_id": null,
        "user_id": null,
        "endpoint_id": null,
        "source_entry_id": "55"
```

```
      },
      "source": "my_ldap_directory"
    }, {
      "column_values": ["Charlie", "Boblin", "5555556", "5554444", "5552222",
↪"mail2@example.com", null, true],
      "relations": {
        "agent_id": 12,
        "user_id": 34,
        "endpoint_id": 56,
        "source_entry_id": "34"
      },
      "source": "internal"
    }
  ]
}
```

Set favorite

Client -> Server

```
{
  "class": "people_set_favorite",
  "source": "my_ldap_directory"
  "source_entry_id": "55"
  "favorite": true
}
```

Server -> Client

```
{
  "class": "people_favorite_update",
  "source": "my_ldap_directory"
  "source_entry_id": "55"
  "favorite": true
}
```

STARTTLS

The STARTTLS command is used to upgrade a connection to use SSL. Once connected, the server send a starttls offer to the client which can reply with a starttls message including the status field. The server will then send a starttls message back to the client with the same status and start the handshake if the status is true.

Server -> Client

```
{
    "class": "starttls"
}
```

Client -> Server -> Client

```
{
    "class": "starttls",
    "status": true
}
```

Note: a client which does not reply to the starttls offer will keep it's unencrypted connection.

Personal contacts list

```
Client -> Server
```

```
{
  "class": "people_personal_contacts"
}
```

```
Server -> Client
```

```
{
  "class": "people_personal_contacts_result",
  "column_headers": ["Firstname", "Lastname", "Phone number", "Mobile", "Fax", "Email
↪", "Agent", "Favorites", "Personal"],
  "column_types": [null, "name", "number_office", "number_mobile", "fax", "email",
↪"relation_agent", "favorite", "personal"],
  "results": [
    {
      "column_values": ["Bob", "Marley", "5555555", "5556666", "5553333",
↪"mail@example.com", null, false, true],
      "relations": {
        "agent_id": null,
        "user_id": null,
        "endpoint_id": null,
        "source_entry_id": "abcd-12"
      },
      "source": "personal"
    }, {
      "column_values": ["Charlie", "Boblin", "5555556", "5554444", "5552222",
↪"mail2@example.com", null, false, true],
      "relations": {
        "agent_id": null,
        "user_id": null,
        "endpoint_id": null,
        "source_entry_id": "efgh-34"
      },
      "source": "personal"
    }
  ]
}
```

Personal contact purge

```
Client -> Server
```

```
{
  "class": "people_purge_personal_contacts",
}
```

```
Server -> Client
```

```
{
  "class": "people_personal_contacts_purged",
}
```

Personal contact raw

```
Client -> Server
```

```
{
  "class": "people_personal_contact_raw",
  "source": "personal",
  "source_entry_id": "abcd-1234"
}
```

```
Server -> Client
```

```
{
  "class": "people_personal_contact_raw_result",
  "source": "personal",
  "source_entry_id": "abcd-1234",
  "contact_infos": {
      "firstname": "Bob",
      "lastname": "Wonderland"
      ...
  }
}
```

Create personal contact

```
Client -> Server
```

```
{
  "class": "people_create_personal_contact",
  "contact_infos": {
      "firstname": "Bob",
      "lastname": "Wonderland",
      ...
  }
}
```

```
Server -> Client
```

```
{
  "class": "people_personal_contact_created"
}
```

Delete personal contact

```
Client -> Server
```

```
{
  "class": "people_delete_personal_contact",
```

```
  "source": "personal",
  "source_entry_id": "abcd-1234"
}
```

Server -> Client

```
{
  "class": "people_personal_contact_deleted",
  "source": "personal",
  "source_entry_id": "abcd-1234"
}
```

Edit personal contact

Client -> Server

```
{
  "class": "people_edit_personal_contact",
  "source": "personal",
  "source_entry_id": "abcd-1234",
  "contact_infos": {
      "firstname": "Bob",
      "lastname": "Wonderland",
      ...
  }
}
```

Server -> Client

```
{
  "class": "people_personal_contact_raw_update",
  "source": "personal",
  "source_entry_id": "abcd-1234"
}
```

Import personal contacts

Client -> Server

```
{
  "class": "people_import_personal_contacts_csv",
  "csv_contacts": "firstname,lastname\r\nBob,the Builder\r\n,Alice,Wonderland\r\n,
↪BobMissingFields\r\n"
}
```

Server -> Client

```
{
  "class": "people_import_personal_contacts_csv_result",
  "created_count": 2,
  "failed": [
      {
          "line": 3,
          "errors": [
```

```
                "missing fields"
                ]
        }

    ]
}
```

Export personal contacts

Client -> Server

```
{
  "class": "people_export_personal_contacts_csv",
}
```

Server -> Client

```
{
  "class": "people_export_personal_contacts_csv_result",
  "csv_contacts": "firstname,lastname\r\nBob,the Builder\r\n,Alice,Wonderland\r\n"
}
```

Service

- class : featuresput

Call Filtering

- function : incallfilter
- value : true, false activate deactivate filtering

Client -> Server

```
{"class": "featuresput", "commandid": 1326845972, "function": "incallfilter", "value
↪": true}
```

Server > Client

```
{
   "class": "getlist",
   "config": {"incallfilter": true},
   "function": "updateconfig",
   "listname": "users",
   "tid": "2",
   "timenow": 1361456398.52, "tipbxid": "xivo"  }
```

DND

- function : enablednd
- value : true, false activate deactivate DND

Client -> Server

```
{"class": "featuresput", "commandid": 1088978942, "function": "enablednd", "value":␣
↪true}
```

Server > Client

```
{
   "class": "getlist",
   "config": {"enablednd": true},
   "function": "updateconfig",
   "listname": "users",
   "tid": "2",
   "timenow": 1361456614.55, "tipbxid": "xivo"}
```

Recording

- function : enablerecording
- value : true, false

Activate / deactivate recording for a user, extension call recording has to be activated : *Services->IPBX->IPBX services->Extension*

Client -> Server

```
{"class": "featuresput", "commandid": 1088978942, "function": "enablerecording",
↪"value": true, "target" : "7" }
```

Server > Client

```
{
   "class": "getlist",
   "config": {"enablerecording": true},
   "function": "updateconfig",
   "listname": "users",
   "tid": "7",
   "timenow": 1361456614.55, "tipbxid": "xivo"}
```

Unconditional Forward

Forward the call at any time, call does not reach the user

- function : fwd

Client -> Server

```
{
   "class": "featuresput", "commandid": 2082138822, "function": "fwd",
   "value": {"destunc": "1002", "enableunc": true}
}
```

Server > Client

```
{
   "class": "getlist",
```

```
   "config": {"destunc": "1002", "enableunc": true},
   "function": "updateconfig",
   "listname": "users",
   "tid": "2",
   "timenow": 1361456777.98, "tipbxid": "xivo"}
```

Forward On No Answer

Forward the call to another destination if the user does not answer

- function : fwd

```
Client -> Server
```

```
{
   "class": "featuresput", "commandid": 1705419982, "function": "fwd",
   "value": {"destrna": "1003", "enablerna": true}
   }
```

```
Server > Client
```

```
{
   "class": "getlist",
   "config": {"destrna": "1003", "enablerna": true},
   "function": "updateconfig",
   "listname": "users",
   "tid": "2",
   "timenow": 1361456966.89, "tipbxid": "xivo" }
```

Forward On Busy

Forward the call to another destination when the user is busy

- function : fwd

```
Client -> Server
```

```
{
   "class": "featuresput", "commandid": 568274890, "function": "fwd",
   "value": {"destbusy": "1009", "enablebusy": true}
   }
```

```
Server > Client
```

```
{
   "class": "getlist",
   "config": {"destbusy": "1009", "enablebusy": true},
   "function": "updateconfig",
   "listname": "users",
   "tid": "2",
   "timenow": 1361457163.77, "tipbxid": "xivo"
   }
```

Statistics

Subscribe to queues stats

This message can be sent from the client to enable statitics update on queues

```
Client -> Server
```

```
  {"commandid":36,"class":"subscribetoqueuesstats"}

``Server > Client``
```

Get queues stats

When statistic update is enable by sending message *Subscribe to queues stats*.

The first element of the message is the queue id

```
{"stats": {"10": {"Xivo-LoggedAgents": 0}},
   "class": "getqueuesstats", "timenow": 1384509582.88}
{"stats": {"1": {"Xivo-WaitingCalls": 0}},
   "class": "getqueuesstats", "timenow": 1384509582.89}
{"stats": {"1": {"Xivo-TalkingAgents": "0", "Xivo-AvailableAgents": "1", "Xivo-EWT":
↪"6"}},
   "class": "getqueuesstats", "timenow": 1384512350.25}
```

Status

These messages can also be received without any request as unsolicited messages.

User status

User status is to manage user presence

- Request user status update

```
Client -> Server
```

```
{"class": "getlist", "commandid": 107712156,
   "function": "updatestatus",
   "listname": "users",
   "tid": "14", "tipbxid": "xivo"}
```

```
Server > Client
```

```
{"class": "getlist",
   "function": "updatestatus",
   "listname": "users",
   "status": {"availstate": "outtolunch", "connection": "yes"},
         "tid": "1", "timenow": 1364994093.48, "tipbxid": "xivo"}
```

- Change User status

```
Client -> Server
```

```
{"availstate": "away",
    "class": "availstate",
    "commandid": 1946092392,
    "ipbxid": "xivo",
        "userid": "1"}
```

Server > Client

```
{"class": "getlist",
    "function": "updatestatus",
    "listname": "users",
    "status": {"availstate": "away"},
    "tid": "1", "timenow": 1370523352.6, "tipbxid": "xivo"}
```

Phone status

- tid is the line id, found in linelist from message *User configuration*

Client -> Server

```
{"class": "getlist", "commandid": 107712156,
   "function": "updatestatus",
   "listname": "phones", "tid": "8", "tipbxid": "xivo"}
```

Server > Client

```
{"class": "getlist",
 "function": "updatestatus",
 "listname": "phones",
 "status": {"hintstatus": "0"},
 "tid": "1",
 "timenow": 1364994093.48,
 "tipbxid": "xivo"}
```

Queue status

Client -> Server

```
{"commandid":17,"class":"getlist","tid":"8","tipbxid":"xivo","function":"updatestatus
↪","listname":"queues"}
```

Server > Client

```
{"function": "updatestatus", "listname": "queues", "tipbxid": "xivo", "timenow":␣
↪1382710430.54,
   "status": {"agentmembers": ["1","5"], "phonemembers": ["8"]},
   "tid": "8", "class": "getlist"}
```

Agent status

- tid is the agent id.

Client -> Server

```
{"class": "getlist",
 "commandid": <random_integer>,
 "function": "updatestatus",
 "listname": "agents",
 "tid": "635",
 "tipbxid": "xivo"}
```

Server > Client

```
{"class": "getlist",
 "listname": "agents",
 "function": "updatestatus",
 "tipbxid": "xivo",
 "tid": 635,
 "status": {
     "availability": "logged_out",
      "availability_since": 1370868774.74,
      "channel": null,
      "groups": [],
      "on_call_acd": false,
      "on_call_nonacd": false,
      "on_wrapup": false,
      "phonenumber": null,
      "queues": [
          "113"
      ]
  }}
```

- availability can take the values:
  - logged_out
  - available
  - unavailable
  - on_call_nonacd_incoming_internal
  - on_call_nonacd_incoming_external
  - on_call_nonacd_outgoing_internal
  - on_call_nonacd_outgoing_external
- availability_since is the timestamp of the last availability change
- queues is the list of queue ids from which the agent receives calls

Switchboard

Answer

This allows the switchboard operator to answer an incoming call or unhold a call on-hold.

```
{"class": "answer", "uniqueid": "12345667.89"}
```

Unsolicited Messages

These messages are received whenever one of the following corresponding event occurs: sheet message on incoming calls, or updatestatus when a phone status changes.

Sheet

This message is received to display customer information if configured at the server side

```
{
   "timenow": 1361444639.61,
   "class": "sheet",
   "compressed": true,
   "serial": "xml",
   "payload": "AAADnnicndPBToNAEAbgV1n3XgFN1AP...................",
   "channel": "SIP/e6fhff-00000007"
}
```

How to decode payload :

```
>>> b64content = base64.b64decode(<payload content>)
>>> # 4 first cars are the encoded lenght of the xml string (in Big Endian format)
>>> xmllen = struck.unpack('>I',b64content[0:4])
>>> # the rest is a compressed xml string
>>> xmlcontent = zlib.decompress(toto[4:])
>>> print xmlcontent

<?xml version="1.0" encoding="utf-8"?>
   <profile>
      <user>
         <internal name="ipbxid"><![CDATA[xivo]]></internal>
         <internal name="where"><![CDATA[dial]]></internal>
         <internal name="channel"><![CDATA[SIP/barometrix_jyldev-00000009]]></
↪internal>
         <internal name="focus"><![CDATA[no]]></internal>
         <internal name="zip"><![CDATA[1]]></internal>
         <sheet_qtui order="0010" name="qtui" type="None"><![CDATA[]]></sheet_qtui>
         <sheet_info order="0010" name="Nom" type="title"><![CDATA[0230210083]]></
↪sheet_info>
         <sheet_info order="0030" name="Origine" type="text"><![CDATA[extern]]></
↪sheet_info>
         <sheet_info order="0020" name="Num\xc3\xa9ro" type="text"><!
↪[CDATA[0230210083]]></sheet_info>
         <systray_info order="0010" name="Nom" type="title"><![CDATA[Maric\xc3\xa9␣
↪Sapr\xc3\xaftch\xc3\xa0]]></systray_info>
         <systray_info order="0030" name="Origine" type="body"><![CDATA[extern]]></
↪systray_info>
         <systray_info order="0020" name="Num\xc3\xa9ro" type="body"><!
↪[CDATA[0230210083]]></systray_info>
      </user>
   </profile>
```

The xml file content is defined by the following xsd file: `xivo-javactilib/src/main/xsd/sheet.xsd` (online version)

Phone status update

Received when a phone status change

- class : getlist
- function : updatestatus
- listname : phones

```
{
   "class": "getlist",
   "function": "updatestatus",
   "listname": "phones",
   "tipbxid": "xivo",
   "timenow": 1361447017.29,
   .........
}
```

tid is the the object identification

Example of phone messages received when a phone is ringing :

```
{.... "status": {"hintstatus": "0"}, "tid": "3"}
{.... "status": {"hintstatus": "8"}, "tid": "3"}
```

Update notification

Register agent status update

The *register_agent_status_update* command is used to register to the status updates of a list of agent. Once registered to a agent's status, the client will receive all *Agent status update* events for the registered agents.

This command should be sent when an agent is displayed in the people xlet to be able to update the agent status icon.

The *Unregister agent status update* command should be used to stop receiving updates.

`Client -> Server`

```
{
  "class": "register_agent_status_update",
  "agent_ids": [["<xivo-uuid>", "<agent-id1>"],
                ["<xivo-uuid>", "<agent-id2>"],
                ...,
                ["<xivo-uuid>", "<agent-idn>"]],
  "commandid": <commandid>
}
```

Unregister agent status update

The *unregister_agent_status_update* command is used to unregister from the status updates of a list of agent.

Once unregistered, the client will stop receiving the *Agent status update* events for the specified agents.

`Client -> Server`

```
{
  "class": "unregister_agent_status_update",
  "agent_ids": [["<xivo-uuid>", "<agent-id1>"],
                ["<xivo-uuid>", "<agent-id2>"],
                ...,
                ["<xivo-uuid>", "<agent-idn>"]],
  "commandid": <commandid>
}
```

Agent status update

The *agent_status_update* event is received when the presence of an agent changes.

To receive this event, the user must first register to the event for a specified agent using the *Register agent status update* command.

To stop receiving this event, the user must send the *Unregister agent status update* command.

- data, a dictionary containing 3 fields:
  - agent_id, is an integer containing the ID of the user affected by this status change
  - xivo_uuid: a string containing the UUID of the Wazo that sent the status update
  - status: a string containing the new status, “logged_in” or “logged_out”

`Server -> Client`

```
{
  "class": "agent_status_update",
  "data": {
    "agent_id": 42,
    "xivo_uuid": "<the-xivo-uuid>",
    "status": "<status-name>"
  }
}
```

The *agent_status_update* event contains the same data as the *agent_status_update*. The latter should be preferred to the former for uses that do not require a persistent connection to xivo-ctid.

Register endpoint status update

The *register_endpoint_status_update* command is used to register to the status updates of a list of lines. Once registered to a endpoint’s status, the client will receive all *Endpoint status update* events for the registered agents.

This command should be sent when a endpoint is displayed in the people xlet to be able to update the agent status icon.

The *Unregister endpoint status update* command should be used to stop receiving updates.

`Client -> Server`

```
{
  "class": "register_endpoint_status_update",
  "endpoint_ids": [["<xivo-uuid>", "<endpoint-id1>"],
                   ["<xivo-uuid>", "<endpoint-id2>"],
                   ...,
                   ["<xivo-uuid>", "<endpoint-idn>"]],
```

```
  "commandid": <commandid>
}
```

Unregister endpoint status update

The *unregister_endpoint_status_update* command is used to unregister from the status updates of a list of agent.

Once unregistered, the client will stop receiving the *Endpoint status update* events for the specified agents.

`Client -> Server`

```
{
  "class": "unregister_endpoint_status_update",
  "endpoint_ids": [["<xivo-uuid>", "<endpoint-id1>"],
                   ["<xivo-uuid>", "<endpoint-id2>"],
                   ...,
                   ["<xivo-uuid>", "<endpoint-idn>"]],
  "commandid": <commandid>
}
```

Endpoint status update

The *endpoint_status_update* event is received when the status of a line changes.

To receive this event, the user must first register to the event for a specified endpoint using the *Register endpoint status update* command.

To stop receiving this event, the user must send the *Unregister endpoint status update* command.

- data, a dictionary containing 3 fields:
  - endpoint_id, is an integer containing the ID of the line affected by this status change
  - xivo_uuid: a string containing the UUID of the Wazo that sent the status update
  - status: an integer matching an entry in the cti hint configuration

`Server -> Client`

```
{
  "class": "endpoint_status_update",
  "data": {
    "endpoint_id": 42,
    "xivo_uuid": "<the-xivo-uuid>",
    "status": <hint-status>
  }
}
```

The *endpoint_status_update* event contains the same data as the *endpoint_status_update*. The latter should be preferred to the former for uses that do not require a persistent connection to xivo-ctid.

Register user status update

The *register_user_status_update* command is used to register to the status updates of a list of user. Once registered to a user's status, the client will receive all *User status update* events for the registered users.

This command should be sent when a user is displayed in the people xlet to be able to update the presence status icon.

The *Unregister user status update* command should be used to stop receiving updates.

```
Client -> Server
```

```
{
  "class": "register_user_status_update",
  "user_ids": [["<xivo-uuid>", "<user-uuid1>"],
               ["<xivo-uuid>", "<user-uuid2>"],
               ...,
               ["<xivo-uuid>", "<user-uuidn>"]],
  "commandid": <commandid>
}
```

Unregister user status update

The *unregister_user_status_update* command is used to unregister from the status updates of a list of user.

Once unregistered, the client will stop receiving the *User status update* events for the specified users.

```
Client -> Server
```

```
{
  "class": "unregister_user_status_update",
  "user_ids": [["<xivo-uuid>", "<user-uuid1>"],
               ["<xivo-uuid>", "<user-uuid2>"],
               ...,
               ["<xivo-uuid>", "<user-uuidn>"]],
  "commandid": <commandid>
}
```

User status update

The *user_status_update* event is received when the presence of a user changes.

To receive this event, the user must first register to the event for a specified user using the *Register user status update* command.

To stop receiving this event, the user must send the *Unregister user status update* command.

- data, a dictionary containing the following fields:
  - user_uuid, a string containing the UUID of the user.
  - user_id, an integer containing the ID of the user.
  - xivo_uuid: a string containing the UUID of the Wazo that sent the status update
  - status: a string containing the new status of the user based on the cti profile configuration

Note: When multiple Wazo share user statuses, the cti profile configuration for presences and phone statuses should match on all Wazo to be displayed properly

```
Server -> Client
```

```
{
  "class": "user_status_update",
  "data": {
    "user_uuid": "<the-user-uuid>",
    "user_id": <the-user-id>,
    "xivo_uuid": "<the-xivo-uuid>",
    "status": "<status-name>"
  }
}
```

Warning: The *user_id* field is **DEPRECATED** and **should not be used**. Use the *user_uuid* field instead.

CTI server implementation

In the git repository `git://github.com/wazo-pbx/xivo-ctid.git`

- *cti_config* handles the configuration coming from the WEBI
- *interfaces/interface_ami*, together with *asterisk_ami_definitions*, *amiinterpret* and *xivo_ami* handle the AMI connections (asterisk)
- *interfaces/interface_info* handles the CLI-like connections
- *interfaces/interface_webi* handles the requests and signals coming from the WEBI
- *interfaces/interface_cti* handles the clients' connections, with the help of *client_connection*, and it often involves *cti_command* too
- *innerdata* is meant to be the place where all statuses are computed and stored

The main loop uses *select()* syscall to dispatch the tasks according to miscellaneous incoming requests.

Requirements for *innerdata*:

- the properties fetched from the WEBI configuration shall be stored in the relevant *xod_config* structure
- the properties fetched from elsewhere shall be stored in the relevant *xod_status* structure
- at least two kinds of objects are not "predefined" (as are the phones or the queues, for instance)
  - the channels (in the asterisk SIP/345-0x12345678 meaning)
  - the group and queue members shall be handled in a special way each

The purpose of the 'relations' field, in the various structures is to keep track of relations and cross-relations between different objects (a phone logged in as an agent, itself in a queue, itself called by some channels belonging to phones ...).

CTI server Message flow

Messages sent from the CTI clients to the server are received by the CTIServer class. The CTIServer then calls `interface_cti.CTI` class `manage_connection` method. The `interface_cti` uses his `_cti_command_handler` member to parse and run the command. The `CTICommandHandler` get a list of classes that handle this message from the `CTICommandFactory`. Then the the `interface_cti.CTI` calls `run_commands` on the handler, which returns a list of all commands replies.

To implement a new message in the protocol you have to create a new class that inherits the `CTICommand` class. Your new class should have a static member caller `required_fields` which is a list of required fields for this

class. Your class should also have a `conditions` static member which is a list of tupples of conditions to detect that an incoming message matches this class. The `__init__` of your class is responsible for the initialization of it's fields and should call `super(<ClassName>, self).__init__(msg)`. Your class should register itself to the `CTICommandFactory`.

```
from xivo_cti.cti.cti_command import CTICommand
from xivo_cti.cti.cti_command_factory import CTICommandFactory

class InviteConfroom(CTICommand):
    required_fields = ['class', 'invitee']
    conditions = [('class', 'invite_confroom')]
    def __init__(self):
        super(InviteConfroom, self).__init__(msg)
        self._invitee = msg['invitee']

CTICommandFactory.register_class(InviteConfroom)
```

Each CTI commands has a callback list that you can register to from anywhere. Each callback function will be called when this message is received with the command as parameter.

Refer to `MeetmeList.__init__` for a callback registration example and to `MeetmeList.invite` for the implementation of a callback.

```
from xivo_cti.cti.commands.invite_confroom import InviteConfroom

class MySuperClass(object):
    def __init__(self):
        InviteConfroom.register_callback(self.invite_confroom_handler)

    def invite_confroom_handler(self, invite_confroom_command):
        # Do your stuff here.
        if ok:
            return invite_confroom_command.get_message('Everything is fine')
        else:
            return invite_confroom_command.get_warning('I don't know you, go away',␣
↪True)
```

Note: The client's connection is injected in the command instance before calling callbacks functions. The client's connection is an `interface_cti.CTI` instance.

1.14.14 Database

Adding a Migration Script

Strating with XiVO 14.08, the database migration is handled by alembic.

The Wazo migration scripts can be found in the xivo-manage-db repository.

On a XiVO, they are located in the `/usr/share/xivo-manage-db` directory.

To add a new migration script from your developer machine, go into the root directory of the xivo-manage-db repository. There should be an `alembic.ini` file in this directory. You can then use the following command to create a new migration script:

```
alembic revision -m "<description>"
```

This will create a file in the `alembic/versions` directory, which you'll have to edit.

When the migration scripts are executed, they use a connection to the database with the role/user `asterisk`. This means that new objects that are created in the migration scripts will be owned by the `asterisk` role and it is thus not necessary (nor recommended) to explicitly grant access to objects to the asterisk role (i.e. no "GRANT ALL" command after a "CREATE TABLE" command).

1.14.15 Diagrams

Agent states

Graphs representing states and transitions between agent states. Used in Agent status dashboard and agent list.

`Download (DIA)`

Architecture

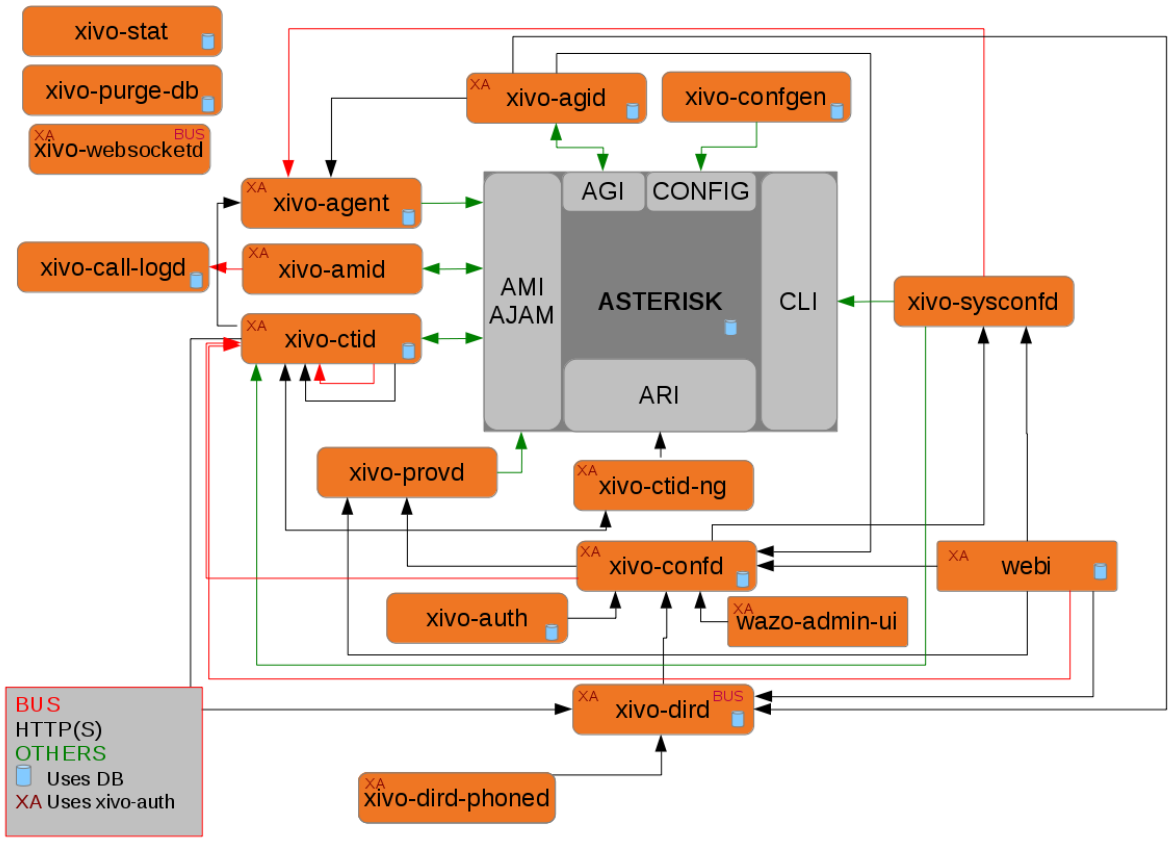


Fig. 1.107: Relationships between the components of Wazo. `(source)`

1.14.16 Provisioning

This section describes the informations and tools for xivo-provd.

Managing DHCP server configuration

This page considers the configuration files of the DHCP server in `/etc/dhcp/dhcpd_update/`.

Who modifies the files

The files are updated with the command `dhcpd-update`, which is also run when updating the provisioning plugins. This commands fetches configurations files from the `provd.wazo.community` server.

How to update the source files

Ensure your modifications are working

- On a Wazo, edit manually the file `/etc/dhcp/dhcpd_update/*.conf`
- `service isc-dhcp-server restart`
- If errors are shown in `/var/log/daemon.log`, check your modifications

Edit the files

- Edit the files in the Git repo `xivo-provd-plugins`, directory `dhcp/`
- Push your modifications
- Go in `dhcp/`
- Run `make upload` to push your modifications to `provd.wazo.community`. There is no `testing` version of these files. Once the files are uploaded, they are available for all Wazo installations.

Managing Plugins

Git Repository

Most plugin-related files are available in the xivo-provd-plugins repository. Following examples are relative to the repository directory tree. Any modifications should be preceeded by a *git pull*.

Updating a Plugin

We will be using the *xivo-cisco-spa* plugins family as an example on this page

There is one directory per family. Here is the directory structure for `xivo-cisco-spa`:

```
plugins/xivo-cisco-spa/
+-- model_name_xxx
+-- model_name_xxx
+-- common
+-- build.py
```

Every plugin has a folder called `common` which regoups common ressources for each model. Every model has its own folder with its version number.

After modifying a plugin, you must increment the version number. You can modifiy the file `plugin-info` to change the version number:

```
plugins/xivo-cisco-spa/
+-- model_name_xxx
    +-- plugin-info
```

Important: If ever you modify the folder `common`, you must increment the version number of all the models.

Use Case: Update Firmwares for a given plugin

Let us suppose we want to update firmwares for xivo-snom from 8.7.3.25 to 8.7.3.25 5. Here are the steps to follow :

1. Copy folder plugins/xivo-snom/8.7.3.25 to plugins/xivo-snom/8.7.3.25.5
2. Update VERSION number in plugins/xivo-snom/8.7.3.25.5/entry.py
3. Update VERSION number in plugins/xivo-snom/8.7.3.25.5/plugin-info
4. Download new firmwares (.bin files from snom website)
5. Update VERSION number and URIs in plugins/xivo-snom/8.7.3.25.5/pkgs/pkgs.db (with uris of downloaded files from snom website)
6. Update sizes and sha1sums in plugins/xivo-snom/8.7.3.25.5/pkgs/pkgs.db (using helper script xivo-tools/dev-tools/check_fw)
7. Update plugins/xivo-snom/build.py (duplicate and update section 8.7.3.25 > 8.7.3.25.5)

Test your changes

You have three different methods to test your changes on your development machine.

Always increase plugin version (easiest)

If the production version is 0.4, change the plugin version to 0.4.01, make your changes and upload to testing (see below).

Next modification will change the plugin version to 0.4.02, etc. When you are finished making changes, change the version to 0.5 and upload one last time.

Edit directly on Wazo

Edit the files in `/var/lib/xivo-provd/plugins`.

To apply your changes, go in `xivo-provd-cli` and run:

```
plugins.reload('xivo-cisco-spa-7.5.4')
```

Disable plugin caching

Edit `/etc/xivo/provd/provd.conf` and add the line:

```
cache_plugin: True
```

Empty `/var/cache/xivo-provd` and restart provd.

Make your changes in provd-plugins, update the plugin version to the new one and upload to testing (see below). Now, every time you uninstall/install the plugin, the new plugin will be fetched from testing, instead of being cached, even without changing the version.

Uploading to testing

Before updating a plugin, it must be passed through the testing phase. Once it has been approved it can be uploaded to the production server.

In the `xivo-provd-plugins` repo, you must merge your changes in the `testing` branch before uploading the plugins to `provd.wazo.community`:

```
git checkout testing
git pull
git merge my-new-branch
git push  # this step is important: it validates that your build is up-to-date and␣
↪will not remove anything
make upload
```

Afterwards, in the web-interface, you must modify the URL in section *Configuration* → *Provisioning* → *General* to:

```
`http://provd.wazo.community/plugins/1/testing/`
```

You can then update the list of plugins and check the version number for the plugin that you modified. Don't forget to install the plugin to test it.

Mass-install all firmwares related to a given plugin

Using xivo-provd-cli on a Wazo server, one can mass-install firmwares. Following example installs all firmwares for xivo-snom 8.7.3.25.5 plugin (note the auto-completion):

```
xivo-provd-cli> plugins.installed().keys()
[u'xivo-snom-8.7.3.15',
 u'xivo-cisco-sccp-legacy',
 u'xivo-snom-8.4.35',
 u'xivo-snom-8.7.3.25',
 u'xivo-aastra-switchboard',
 u'xivo-aastra-3.2.2-SP3',
```

```
 u'xivo-aastra-3.2.2.1136',
 u'xivo-cisco-sccp-9.0.3',
 u'null',
 u'xivo-snom-8.7.3.25.5']
xivo-provd-cli> p = plugins['xivo-snom-8.7.3.25.5']
xivo-provd-cli> p.install_all()
```

Uploading to stable

Once checked, you must synchronize the plugin from *testing* to *stable*. If applicable, you should also update the archive repo.

To download the stable and archive plugins:

```
$ make download-stable
$ make download-archive
```

Go to the *plugins/_build* directory and delete the plugins that are going to be updated. Note that if you are not updating a plugin but you are instead removing it "once and for all", you should instead move it to the archive directory:

```
$ rm -fi stable/xivo-cisco-spa*
```

Copy the files from the directory *testing* to *stable*:

```
$ cp testing/xivo-cisco-spa* stable
```

Go back to the *plugins* directory and upload the files to the stable and archive repo:

```
$ make upload-stable
$ make upload-archive
```

The file are now up to date and you can test by putting back the *stable* url in the web-interface's configuration:

```
`http://provd.wazo.community/plugins/1/stable/`
```

Testing a new SIP phone

Let's suppose you have received a brand new SIP phone that is not supported by the provisioning system of Wazo. You would like to know if it's possible to add auto-provisioning support for it. That said, you have never tested the phone before.

This guide will help you get through the different steps that are needed to add auto-provisioning support for a phone to Wazo.

Prerequisites

Before continuing, you'll need the following:

- a private LAN where only your phones and your test machines are connected to it, i.e. a LAN that you fully control.

Configuring a test environment

Although it's possible to do all the testing directly on a Wazo, it's more comfortable and usually easier to do on a separate, dedicated machine.

That said, you'll still need a Wazo near, since we'll be doing the call testing part on it and not on a separate asterisk.

So, for the rest of this guide, we'll suppose you are doing your tests on a *Debian* server with the following configuration:

- Installed packages:

```
isc-dhcp-server tftpd-hpa apache2
```

- Example content of the `/etc/dhcp/dhcpd.conf` file (restart `isc-dhcp-server` after modification):

```
ddns-update-style none;

default-lease-time 7200;
max-lease-time 86400;

log-facility local7;

subnet 10.34.1.0 netmask 255.255.255.0 {
    authoritative;

    range 10.34.1.200 10.34.1.250;

    option subnet-mask 255.255.255.0;
    option broadcast-address 10.34.1.255;
    option routers 10.34.1.6;

    option ntp-servers 10.34.1.6;
    option domain-name "my-domain.example.org";
    option domain-name-servers 10.34.1.6;

    log(concat("[VCI: ", option vendor-class-identifier, "]"));
}
```

- Example content of the `/etc/default/tftpd-hpa` file (restart `tftpd-hpa` after modifcation):

```
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/srv/tftp"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="--secure --verbose"
```

With this configuration, files served via TFTP will be in the `/srv/tftp` directory and those served via HTTP in the `/var/www` directory.

Testing

Adding auto-provisioning support for a phone is mostly a question of finding answers to the following questions.

1. *Is it worth the time adding auto-provisioning support for the phone ?*

   Indeed. Adding quality auto-provisioning support for a phone to Wazo requires a non negligible amount of work, if you don't meet any real problem and are comfortable with provisioning in Wazo. Not all phones are

born equal. Some are cheap. Some are old and slow. Some are made to work on proprietary system and will only work in degraded mode on anything else.

That said, if you are uncertain, testing will help you clarifying your idea.

2. *What is the vendor, model, MAC address and firmware version (if available) of your phone ?*

   Having the vendor and model name is essential when looking for documentation or other information. The MAC address will be needed later on for some tests, and it's always good to know the firmware version of the phone if you are trying to upgrade to a newer firmware version and you're having some troubles, and when reading the documentation.

3. *Is the official administrator guide/documentation available publicly on the vendor web site ? Is it available only after registering and login to the vendor web site ?*

   Having access to the administrator guide/documentation of the phone is also essential. Once you've found it, download it and keep the link to the URL. If you can't find it, it's probably not worth going further.

4. *Is the latest firmware of the phone available publicly on the vendor web site ? Is it available only after registering and login to the vendor web site ?*

   Good auto-provisioning support means you need to have an easy way to download the latest firmware of the phone. Ideally, this mean the firmware is downloadable from an URL, with no authentication whatsoever. In the worst case, you'll need to login on some web portal before being able to download the firmware, which will be cumbersome to automatize and probably fragile. If this is the case, it's probably not worth going further.

5. *Does the phone need other files, like language files ? If so, are these files available publicly on the vendor web site ? After registering ?*

   Although you might not be able to answer to this question yet because you might not know if the phone needs such files to be either in English or in French (the two officially supported language in Wazo), you'll need to have an easy access to these files if its the case.

6. *Does the phone supports auto-provisioning via DHCP + HTTP (or TFTP) ?*

   The provisioning system in Wazo is based on the popular method of using a DHCP server to tell the phone where to download its configuration files, and a HTTP (or TFTP) server to serve these configuration files. Some phones support other methods of provisioning (like TR-069), but that's of no use here. Also, if your phone is only configurable via its web interface, although it's technically possible to configure it automatically by navigating its web interface, it's an **extremely bad** idea since it's impossible to guarantee that you'll still be able to provision the phone on the next firmware release.

   If the phone supports both HTTP and TFTP, pick HTTP, it usually works better with the provisioning server of Wazo.

7. *What are the default usernames/passwords on the phone to access administrator menus (phone UI and web UI) ? How do you do a factory reset of the phone ?*

   Although this step is optional, it might be handy later to have these kind of information. Try to find them now, and note them somewhere.

8. *What are the DHCP options and their values to send to the phones to tell it where its configuration files are located ?*

   Once you know that the phone supports DHCP + HTTP provisioning, the next question is what do you need to put in the DHCP response to tell the phone where its configuration files are located. Unless the admin documentation of the phone is really poor, this should not be too hard to find.

   Once you have found this information, the easiest way to send it to the phone is to create a custom host declaration for the phone in the `/etc/dhcp/dhcpd.conf` file, like in this example:

```
host my-phone {
   hardware ethernet 00:11:22:33:44:55;
   option tftp-server-name "http://169.254.0.1/foobar.cfg";
}
```

9. *What are the configuration files the phone needs (filename and content) and what do we need to put in it for the phone to minimally be able to make and receive calls on Wazo ?*

   Now that you are able to tell your phone where to look for its configuration files, you need to write these files with the right content in it. Again, at this step, you'll need to look through the documentation or examples to answer this question.

   Note that you only want to have the most basic configuration here, i.e. only configure 1 line, with the right SIP registrar and proxy, and the associated username and password.

10. *Do basic telephony services, like transfer, works correctly when using the phone buttons ?*

    On most phones, it's possible to do transfer (both attended and direct), three-way conferences or put someone on hold directly from the phone. Do some tests to see if it works correctly.

    Also at this step, it's a good idea to check how the phone handle non-ascii characters, either in the caller ID or in its configuration files.

11. *Does other "standard" features work correctly on the phone ?*

    For quality auto-provisioning support, you must find how to configure and make the following features work:

    - NTP server
    - MWI
    - function keys (speed dial, BLF, directed pickup / call interception)
    - timezone and DST support
    - multi language
    - DTMF
    - hard keys, like the voicemail hard key on some phone
    - non-ASCII labels (line name, function key label)
    - non-ASCII caller ID
    - backup proxy/registrar
    - paging

Once you have answered all these questions, you'll have a good idea on how the phone works and how to configure it. Next step would be to start the development of a new provd plugin for your phone for a specific firmware version.

IOT Phones

FK = Funckey

HK = HardKey

Y = Supported

MN = Menu

N = Not supported

NT = Not tested

NYT = Not yet tested

SK = SoftKey

	model
Provisioning	Y
H-A	Y
Directory XIVO	Y
Funckeys	8
Supported programmable keys	
User with supervision function	Y
Group	Y
Queue	Y
Conference Room with supervision function	Y
General Functions	
Online call recording	N
Phone status	Y
Sound recording	Y
Call recording	Y
Incoming call filtering	Y
Do not disturb	Y
Group interception	Y
Listen to online calls	Y
Directory access	Y
Filtering Boss - Secretary	Y
Transfers Functions	
Blind transfer	HK
Indirect transfer	HK
Forwards Functions	
Disable all forwarding	Y
Enable/Disable forwarding on no answer	Y
Enable/Disable forwarding on busy	Y
Enable/Disable forwarding unconditional	Y
Voicemail Functions	
Enable voicemail with supervision function	Y
Reach the voicemail	Y
Delete messages from voicemail	Y
Agent Functions	
Connect/Disconnect a static agent	Y
Connect a static agent	Y
Disconnect a static agent	Y
Parking Functions	
Parking	Y
Parking position	Y
Paging Functions	
Paging	Y

Configuring a NAT Environment

This is a configuration example to simulate the case of a hosted Wazo, i.e. an environment where:

- the Wazo has a public IP address

- the phones are behind a NAT

In this example, we'll reproduce the following environment:

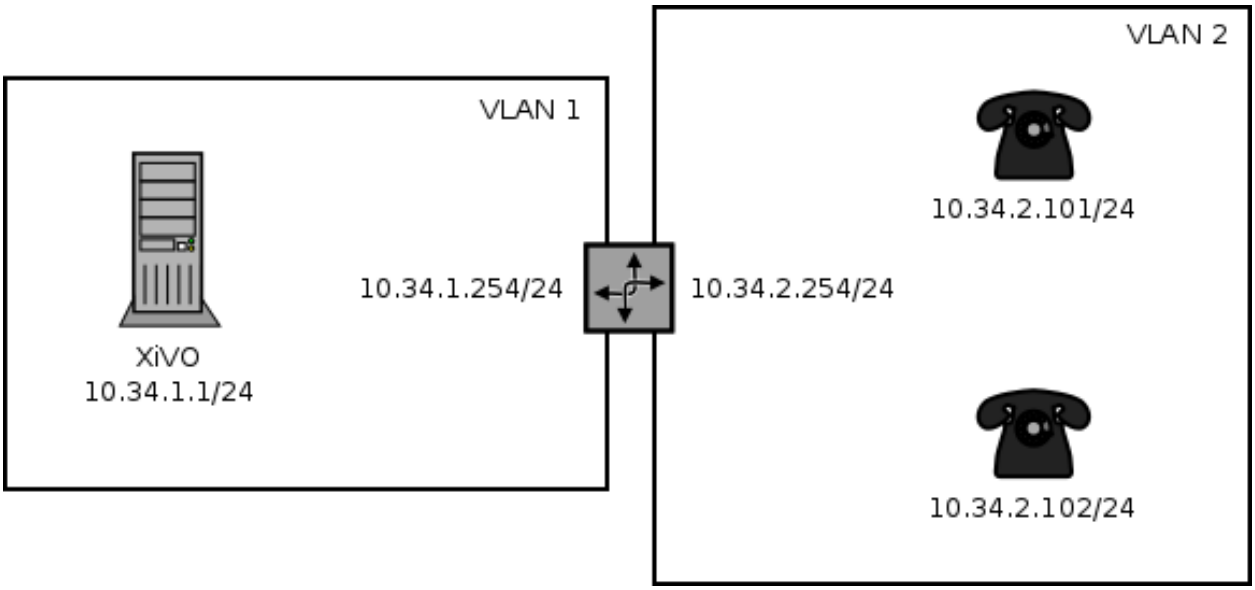


Fig. 1.108: Phones behind a NAT

Where:

- the Wazo is installed inside a virtual machine
- the host machine is used as a router, a NAT and a DHCP server for the phones
- the phones are in a separate VLAN than the Wazo, and when they want to interact with it, they must pass through the NAT

With this setup, we could also put some phones in the same VLAN as the Wazo. We would then have a mixed environment, where some phones are behind the NAT and some phones aren't.

Also, it's easy to go from a non-NAT environment to a NAT environment with this setup. What you usually have to do is only to switch your phone from the "Wazo" VLAN to the "phones" VLAN, and reconfiguring the lines on your Wazo.

Prerequisite

On the host machine:

- 1 VLAN network interface for the Wazo. In our example, this will be `eth0.341`, with IP 10.34.1.254/24.
- 1 VLAN network interface for the phones. In our example, this will be `eth0.342`, with IP 10.34.2.254/24.

On the guest machine, i.e. on the Wazo:

- 1 network adapter attached to the "Wazo" VLAN network interface. In our example, this interface inside the virtual machine will have the IP 10.34.1.1/24.

Configuration

1. On the host, install the ISC DHCP server:

```
apt-get install isc-dhcp-server
```

2. If you do not want it to always be started:

```
systemctl disable isc-dhcp-server.service
```

3. Edit the DHCP server configuration file `/etc/dhcp/dhcpd.conf`. We need to configure the DHCP server to serve network configuration for the phones (Aastra and Snom in this case):

```
ddns-update-style none;

default-lease-time 3600;
max-lease-time 86400;

log-facility daemon;

option space Aastra6700;
option Aastra6700.cfg-server-name code 2 = text;
option Aastra6700.contact-rcs code 3 = boolean;

class "Aastra" {
    match if substring(option vendor-class-identifier, 0, 6) = "Aastra";

    vendor-option-space Aastra6700;
    option Aastra6700.cfg-server-name = "http://10.34.1.1:8667/Aastra";
    option Aastra6700.contact-rcs false;
}

class "Snom" {
    match if substring(option vendor-class-identifier, 0, 4) = "snom";

    option tftp-server-name = "http://10.34.1.1:8667";
    # the domain-name-servers option must be provided for the Snom 715 to work␣
↪properly
    option domain-name-servers 10.34.1.1;
}

subnet 192.168.32.0 netmask 255.255.255.0 {
}

subnet 10.34.1.0 netmask 255.255.255.0 {
}

subnet 10.34.2.0 netmask 255.255.255.0 {
    authoritative;

    range 10.34.2.100 10.34.2.199;

    option subnet-mask 255.255.255.0;
    option broadcast-address 10.34.2.255;
    option routers 10.34.2.254;

    option ntp-servers 10.34.1.1;
}
```

4. If you have many network interfaces on your host machine, you might also want to edit `/etc/default/isc-dhcp-server` to only include the "phones" VLAN network interface in the "INTERFACES" variable.

5. Start the isc-dhcp-server:

```
systemctl start isc-dhcp-server.service
```

6. Add an iptables rules to do NAT:

```
iptables -t nat -A POSTROUTING -o eth0.341 -j MASQUERADE
```

7. Make sure that IP forwarding is enabled:

```
sysctl -w net.ipv4.ip_forward=1
```

8. Put all the phones in the "phones" VLAN on your switch
9. Activate the `NAT` and `Monitoring` options on the *Services → IPBX → General settings → SIP Protocol* page of your Wazo.

Note that the iptables rules and the IP forwarding setting are not persistent. If you don't make them persistent (not documented here), don't forget to reactivate them each time you want to recreate a NAT environment.

Developing Provisioning Plugins

Here is an example of how to develop a provisioning plugin for Digium phones. You can find all the code on Github.

Phone Analysis

Here's a non-exhaustive list of what a phone may or may not support:

- Language
- Timezone
- UTF-8
- Reboot of the phone (SIP notify ?)
- Simple call
- Blind transfer
- Attended transfer
- Firmware upgrade
- Multiple lines
- DTMF (RTP ? SIP ?)
- MWI (voicemail indication)
- Voicemail button
- Call on hold
- Function keys
- Call interception (with BLF)
- NTP

DHCP Configuration

In `xivo-provd-plugins/provisioning/dhcpd-update/dhcp/dhcpd_update`:

```
group {
    option tftp-server-name = concat(config-option VOIP.http-server-uri, "/Digium");
    class "DigiumD40" {
        match if substring(option vendor-class-identifier, 0, 10) = "digium_D40";
        log(concat("[", binary-to-ascii(16, 8, ":", hardware), "] ", "BOOT Digium D40
↪"));
    }
    class "DigiumD50" {
        match if substring(option vendor-class-identifier, 0, 10) = "digium_D50";
        log(concat("[", binary-to-ascii(16, 8, ":", hardware), "] ", "BOOT Digium D50
↪"));
    }
    class "DigiumD70" {
        match if substring(option vendor-class-identifier, 0, 10) = "digium_D70";
        log(concat("[", binary-to-ascii(16, 8, ":", hardware), "] ", "BOOT Digium D70
↪"));
    }
}
```

In `xivo-provd-plugins/provisioning/dhcpd-update/dhcp/dhcpd_subnet.conf.middle`:

```
# Digium
allow members of "DigiumD40";
allow members of "DigiumD50";
allow members of "DigiumD70";
```

You can check the logs in `/var/log/syslog`:

```
dhcpd: [1:0:f:d3:5:48:48] [VENDOR-CLASS-IDENTIFIER: digium_D40_1_1_0_0_48178]
dhcpd: [1:0:f:d3:5:48:48] POOL VoIP
dhcpd: [1:0:f:d3:5:48:48] BOOT Digium D40
dhcpd: DHCPDISCOVER from 00:0f:d3:05:48:48 via eth0
dhcpd: DHCPOFFER on 10.42.1.100 to 00:0f:d3:05:48:48 via eth0
dhcpd: [1:0:f:d3:5:48:48] [VENDOR-CLASS-IDENTIFIER: digium_D40_1_1_0_0_48178]
dhcpd: [1:0:f:d3:5:48:48] POOL VoIP
dhcpd: [1:0:f:d3:5:48:48] BOOT Digium D40
dhcpd: DHCPREQUEST for 10.42.1.100 (10.42.1.1) from 00:0f:d3:05:48:48 via eth0
dhcpd: DHCPACK on 10.42.1.100 to 00:0f:d3:05:48:48 via eth0
```

Update the DHCP configuration

To upload the new DHCP configuration on `provd.wazo.community`, in `xivo-provd-plugins/dhcpd-update`:

```
make upload
```

To download the DHCP configuration on the Wazo server, run:

```
dhcpcd-update -d
```

Plugin creation

In `xivo-provd-plugins/plugins`, create the directory tree:

```
xivo-digium/
    build.py
    1.1.0.0/
        plugin-info
        entry.py
        pkgs/
            pkgs.db
    common/
        common.py
        var/
            tftpboot/
                Digium/
```

In `build.py`:

```
# -*- coding: UTF-8 -*-

from subprocess import check_call

@target('1.1.0.0', 'xivo-digium-1.1.0.0')
def build_1_1_0_0(path):
    check_call(['rsync', '-rlp', '--exclude', '.*',
                'common/', path])
    check_call(['rsync', '-rlp', '--exclude', '.*',
                '1.1.0.0/', path])
```

In `1.1.0.0/plugin-info`:

```
{
    "version": "0.3",
    "description": "Plugin for Digium D40, D50 and D70 in version 1.1.0.0.",
    "description_fr": "Greffon pour Digium D40, D50 et D70 en version 1.1.0.0.",
    "capabilities": {
        "Digium, D40, 1.1.0.0": {
            "sip.lines": 2
        },
        "Digium, D50, 1.1.0.0": {
            "sip.lines": 4
        },
        "Digium, D70, 1.1.0.0": {
            "sip.lines": 6
        }
    }
}
```

In `1.1.0.0/entry.py`:

```
# -*- coding: UTF-8 -*-
common = {}
execfile_('common.py', common)
VERSION = u'1.1.0.0.48178'
class DigiumPlugin(common['BaseDigiumPlugin']):
    IS_PLUGIN = True
    pg_associator = common['DigiumPgAssociator'](VERSION)
```

In `1.1.0.0/pkgs/pkgs.db`, put the informations needed to download the firmwares:

```
[pkg_firmware]
description: Firmware for all Digium phones
description_fr: Micrologiciel pour tous les téléphones Digium
version: 1.1.0.0
files: firmware
install: digium-fw

[install_digium-fw]
a-b: untar $FILE1
b-c: cp */*.eff firmware/

[file_firmware]
url: http://downloads.digium.com/pub/telephony/res_digium_phone/firmware/firmware_1_1_
↪0_0_package.tar.gz
size: 100111361
sha1sum: 1d44148b996eaf270fd35995f3c5d69ff0438c5b
```

In `common/common.py`, put the code needed to extract informations about the phone:

```
class DigiumDHCPDeviceInfoExtractor(object):

    _VDI_REGEX = re.compile(r'^digium_(D\d\d)_([\d_]+)$')

    def extract(self, request, request_type):
        return defer.succeed(self._do_extract(request))

    def _do_extract(self, request):
        options = request['options']
        if 60 in options:
            return self._extract_from_vdi(options[60])

    def _extract_from_vdi(self, vdi):
        # Vendor Class Identifier:
        #   digium_D40_1_0_5_46476
        #   digium_D40_1_1_0_0_48178
        #   digium_D70_1_0_5_46476
        #   digium_D70_1_1_0_0_48178
        match = self._VDI_REGEX.match(vdi)
        if match:
            model = match.group(1).decode('ascii')
            fw_version = match.group(2).replace('_', '.').decode('ascii')
            dev_info = {u'vendor': u'Digium',
                        u'model': model,
                        u'version': fw_version}
            return dev_info


class DigiumHTTPDeviceInfoExtractor(object):

    _PATH_REGEX = re.compile(r'^/Digium/(?:([a-fA-F\d]{12})\.cfg)?')

    def extract(self, request, request_type):
        return defer.succeed(self._do_extract(request))

    def _do_extract(self, request):
        match = self._PATH_REGEX.match(request.path)
        if match:
```

```
            dev_info = {u'vendor': u'Digium'}
            raw_mac = match.group(1)
            if raw_mac and raw_mac != '000000000000':
                mac = norm_mac(raw_mac.decode('ascii'))
                dev_info[u'mac'] = mac
            return dev_info
```

You should see in the logs (`/var/log/xivo-provd.log`):

```
provd[1090]: Processing HTTP request: /Digium/000fd3054848.cfg
provd[1090]: <11> Extracted device info: {u'ip': u'10.42.1.100', u'mac': u
↪'00:0f:d3:05:48:48', u'vendor': u'Digium'}
provd[1090]: <11> Retrieved device id: 254374beec8d40209ff70393326b0b13
provd[1090]: <11> Routing request to plugin xivo-digium-1.1.0.0
```

Still in `common/common.py`, put the code needed to associate the phone with the plugin:

```
class DigiumPgAssociator(BasePgAssociator):

    _MODELS = [u'D40', u'D50', u'D70']

    def __init__(self, version):
        BasePgAssociator.__init__(self)
        self._version = version

    def _do_associate(self, vendor, model, version):
        if vendor == u'Digium':
            if model in self._MODELS:
                if version == self._version:
                    return FULL_SUPPORT
                return COMPLETE_SUPPORT
            return PROBABLE_SUPPORT
        return IMPROBABLE_SUPPORT
```

Then, the last piece: the generation of the phone configuration:

```
class BaseDigiumPlugin(StandardPlugin):

    _ENCODING = 'UTF-8'
    _CONTACT_TEMPLATE = 'contact.tpl'

    def __init__(self, app, plugin_dir, gen_cfg, spec_cfg):
        StandardPlugin.__init__(self, app, plugin_dir, gen_cfg, spec_cfg)

        self._tpl_helper = TemplatePluginHelper(plugin_dir)
        self._digium_dir = os.path.join(self._tftpboot_dir, 'Digium')

        downloaders = FetchfwPluginHelper.new_downloaders(gen_cfg.get('proxies'))
        fetchfw_helper = FetchfwPluginHelper(plugin_dir, downloaders)

        self.services = fetchfw_helper.services()
        self.http_service = HTTPNoListingFileService(self._tftpboot_dir)

    dhcp_dev_info_extractor = DigiumDHCPDeviceInfoExtractor()

    http_dev_info_extractor = DigiumHTTPDeviceInfoExtractor()

    def configure(self, device, raw_config):
```

```
        self._check_device(device)

        filename = self._dev_specific_filename(device)
        contact_filename = self._dev_contact_filename(device)

        tpl = self._tpl_helper.get_dev_template(filename, device)
        contact_tpl = self._tpl_helper.get_template(self._CONTACT_TEMPLATE)

        raw_config['XX_mac'] = self._format_mac(device)
        raw_config['XX_main_proxy_ip'] = self._get_main_proxy_ip(raw_config)
        raw_config['XX_funckeys'] = self._transform_funckeys(raw_config)
        raw_config['XX_lang'] = raw_config.get(u'locale')

        path = os.path.join(self._digium_dir, filename)
        contact_path = os.path.join(self._digium_dir, contact_filename)
        self._tpl_helper.dump(tpl, raw_config, path, self._ENCODING)
        self._tpl_helper.dump(contact_tpl, raw_config, contact_path, self._ENCODING)

    def deconfigure(self, device):
        filenames = [
            self._dev_specific_filename(device),
            self._dev_contact_filename(device)
        ]

        for filename in filenames:
            path = os.path.join(self._digium_dir, filename)
            try:
                os.remove(path)
            except OSError as e:
                logger.info('error while removing file %s: %s', path, e)

    if hasattr(synchronize, 'standard_sip_synchronize'):
        def synchronize(self, device, raw_config):
            return synchronize.standard_sip_synchronize(device)

    else:
        # backward compatibility with older xivo-provd server
        def synchronize(self, device, raw_config):
            try:
                ip = device[u'ip'].encode('ascii')
            except KeyError:
                return defer.fail(Exception('IP address needed for device␣
↪synchronization'))
            else:
                sync_service = synchronize.get_sync_service()
                if sync_service is None or sync_service.TYPE != 'AsteriskAMI':
                    return defer.fail(Exception('Incompatible sync service: %s' %␣
↪sync_service))
                else:
                    return threads.deferToThread(sync_service.sip_notify, ip, 'check-
↪sync')

    def get_remote_state_trigger_filename(self, device):
        if u'mac' not in device:
            return None

        return self._dev_specific_filename(device)
```

```
    def is_sensitive_filename(self, filename):
        return bool(self._SENSITIVE_FILENAME_REGEX.match(filename))

    def _check_device(self, device):
        if u'mac' not in device:
            raise Exception('MAC address needed to configure device')

    def _get_main_proxy_ip(self, raw_config):
        if raw_config[u'sip_lines']:
            line_no = min(int(x) for x in raw_config[u'sip_lines'].keys())
            line_no = str(line_no)
            return raw_config[u'sip_lines'][line_no][u'proxy_ip']
        else:
            return raw_config[u'ip']


    def _format_mac(self, device):
         return format_mac(device[u'mac'], separator='', uppercase=False)

    _SENSITIVE_FILENAME_REGEX = re.compile(r'^[0-9a-f]{12}\.cfg$')

    def _dev_specific_filename(self, device):
        filename = '%s.cfg' % self._format_mac(device)
        return filename

    def _dev_contact_filename(self, device):
        contact_filename = '%s-contacts.xml' % self._format_mac(device)
        return contact_filename

    def _transform_funckeys(self, raw_config):
        return dict(
            (int(k), v) for k, v in raw_config['funckeys'].iteritems()
        )
```

Then you can create the configuration templates with Jinja syntax. Here are some examples:

- base.tpl
- contact.tpl
- D40.tpl

Upload the plugin on `provd.wazo.community`

First, change the source of your plugins in *Configuration -> Provisioning -> General* (cf. *Alternative plugins repository*)

For a development version:

```
cd xivo-skaro/provisioning/plugins
make upload
```

For a stable version:

```
cd xivo-skaro/provisioning/plugins
make download-stable
cd _build
cp dev/xivo-digium-1.1.0.0-0.3.tar.bz2 stable/
```

```
cd ..
make upload-stable
```

More details about this in *Managing Plugins*.

1.14.17 SCCP

xivo-libsccp is an alternative SCCP channel driver for Asterisk. It was originally based on chan_skinny.

This page is intended for developers and people interested in using xivo-libsccp on something other than Wazo.

Installation from the git repository

Warning: If you just want to use your SCCP phones with Wazo, refer to *SCCP Configuration* instead.

The following packages are required to compile xivo-libsccp on Debian.

- build-essential
- asterisk-dev

```
apt-get update && apt-get install build-essential asterisk-dev
```

```
git clone https://github.com/wazo-pbx/xivo-libsccp.git
cd xivo-libsccp
make
make install
```

Configuration

Warning: If you just want to use your SCCP phones with Wazo, refer to *SCCP Configuration* instead.

See sccp.conf.sample for a configuration file example.

FAQ

```
Q. When is this *feature X* will be available?
A. The order in which we implement features is based on our client needs. Write
   us an email that clearly explain your setup and what you would like to do and we
   will see what we can do. We don't provide any timeline.
```

```
Q. I want to use the Page() application to call many phones at the same time.
A. Here a Page() example for a one way call (half-duplex):

exten => 1000,1,Verbose(2, Paging to external cisco phone)
same => n,Page(sccp/100/autoanswer&sccp/101/autoanswer,i,120 )

...for a two-way call (full-duplex):
```

```
exten => 1000,1,Verbose(2, Paging to external cisco phone)
same => n,Page(sccp/100/autoanswer&sccp/101/autoanswer,di,120 )
```

Network Configuration for 7920/7921

Here's how to to configure a hostapd based AP on a Debian host so that both a 7920 and 7921 Wi-Fi phone can connect to it.

The 7920 is older than the 7921 and is pretty limited in its Wi-Fi functionnality:

- 802.11b
- WPA (no WPA2)
- TKIP (no CCMP/AES)

Which means that the most secure WLAN you can set up if you want both phones to connect to it is not that secure.

1. Make sure you have a wireless NIC capable of master mode.
2. If needed, install the firmware-<vendor> package. For example, if you have a ralink card like I do:

   ```
   apt-get install firmware-ralink
   ```

3. Install the other dependencies:

   ```
   apt-get install wireless-tools hostapd bridge-utils
   ```

4. Create an hostapd configuration file in `/etc/hostapd/hostapd.sccp.conf` with content: `hostapd.sccp.conf`
5. Update the following parameters (if applicable) in the configuration file:
   - interface
   - ssid
   - channel
   - wpa_passphrase
6. Create a new stanza in `/etc/network/interfaces`:

   ```
   iface wlan-sccp inet manual
        hostapd /etc/hostapd/hostapd.sccp.conf
   ```

7. Up the interface:

   ```
   ifup wlan0=wlan-sccp
   ```

8. Configure your 7920/7921 to connect to the network.

   To unlock the phone's configuration menu on the 7921:
   - Press the Navigation Button downwards to enter SETTINGS mode
   - Navigate to and select Network Profiles
   - Unlock the IP phone's configuration menu by pressing **#. The padlock icon on the top-right of the screen will change from closed to open.

When asked for the authentication mode, select something like “Auto” or “AKM”.

You don’t have to enter anything for the username/password.

9. You’ll probably want to bridge your wlan0 interface with another interface, for example a VLAN interface:

```
brctl addbr br0
brctl addif br0 wlan0
brctl addif br0 eth0.341
ip link set br0 up
```

10. If you are using virtualbox and your guest interface is bridged to eth0.341, you’ll need to change its configuration and bridge it with br0 instead, else it won’t work properly.

Adding Support for a New Phone

This section describes the requirements to consider that a SCCP phone is working with Wazo libsccp.

Basic functionality

- Register on Asterisk
- SCCP reset [restart]
- Call history
- Date time display
- HA

Telephony

These test should be done with and without direct media enabled

- Emit a call
- Receive a call
- Receive and transfer a call
- Emit a call and transfer the call
- Hold and resume a call
- Features (*0 and others)
- Receive 2 calls simultaneously
- Emit 2 calls simultaneously
- DTMF on an external IVR

Function keys

- Redial
- DND
- Hold

- Resume
- New call
- End call
- Call forward (Enable)
- Call forward (Disable)
- Try each button in each mode (on hook, in progress, etc)

Optional options to test and document

- Phone book
- Caller ID and other display i18n
- MWI
- Speeddial/BLF

1.14.18 Web Interface

Configuration for development

Default error level for Wazo web interface is E_ALL & ~E_DEPRECATED & ~E_USER_DEPRECATED & ~E_RECOVERABLE_ERROR & ~E_STRICT

If you want to display warning or other error in your browser, edit the `/etc/xivo/web-interface/xivo.ini` and replace report_type level to 3:

```
[error]
level = E_ALL
report_type = 3
report_mode = 1
report_func = 1
email = john.doe@example.com
file = /var/log/xivo-web-interface/error.log
```

You may also edit `/etc/xivo/web-interface/php.ini` and change the error level, but you will need to restart the cgi:

```
service spawn-fcgi restart
```

Interactive debugging in Eclipse

Instructions for Eclipse 4.5.

On your Wazo:

1. Install php5-xdebug:

   ```
   apt-get install php5-xdebug
   ```

2. Edit the `/etc/php5/cgi/conf.d/20-xdebug.ini` (or `/etc/php5/conf.d/20-xdebug.ini` on wheezy) and add these lines at the end:

```
xdebug.remote_enable=1
xdebug.remote_host="<dev_host_ip>"
```

where <dev_host_ip> is the IP address of your machine where Eclipse is installed.

3. Restart spawn-fcgi:

```
service spawn-fcgi restart
```

On your machine where Eclipse is installed:

1. Make sure you have Eclipse PDT installed
2. Create a PHP project named xivo-web-interface:
   - Choose "Create project at existing location", using the xivo-web-interface directory
3. In the Window / Preferences / PHP menu:
   - Add a new PHP server with the following information:
     - Name: anything you want
     - Base URL: https://<wazo_ip>
     - Path Mapping:
       * Path on Server: /usr/share/xivo-web-interface
       * Path in Workspace: /xivo-web-interface/src
4. Create a new PHP Web Application debug configuration:
   - Choose the PHP server you created in last step
   - Pick some file, which can be anything if you don't "break at first line"
   - Uncheck "Auto Generate", and set the path you want your browser to open when you'll launch this debug configuration.

Then, to start a debugging session, set some breakpoints in the code and launch your debug configuration. This will open the page in your browser, and when the code will hit your breakpoints, you'll be able to go through the code step by step, etc.

1.15 Quality assurance

1.15.1 Testing architecture

Legend:

- assu is our production Wazo, used to make calls in the company. We also use it as a source of "external" calls to the test servers.
- dev-gateway is a simple gateway, to link all other servers.
- xivo-daily is reinstalled every day and runs all the automatic tests in xivo-acceptance.
- xivo-load handles a lot of calls all day long, and we monitor the system metrics while it does.
- callgen makes the calls towards xivo-load
- xivo-test and xivo-test-slave are used for manual tests we run before each release

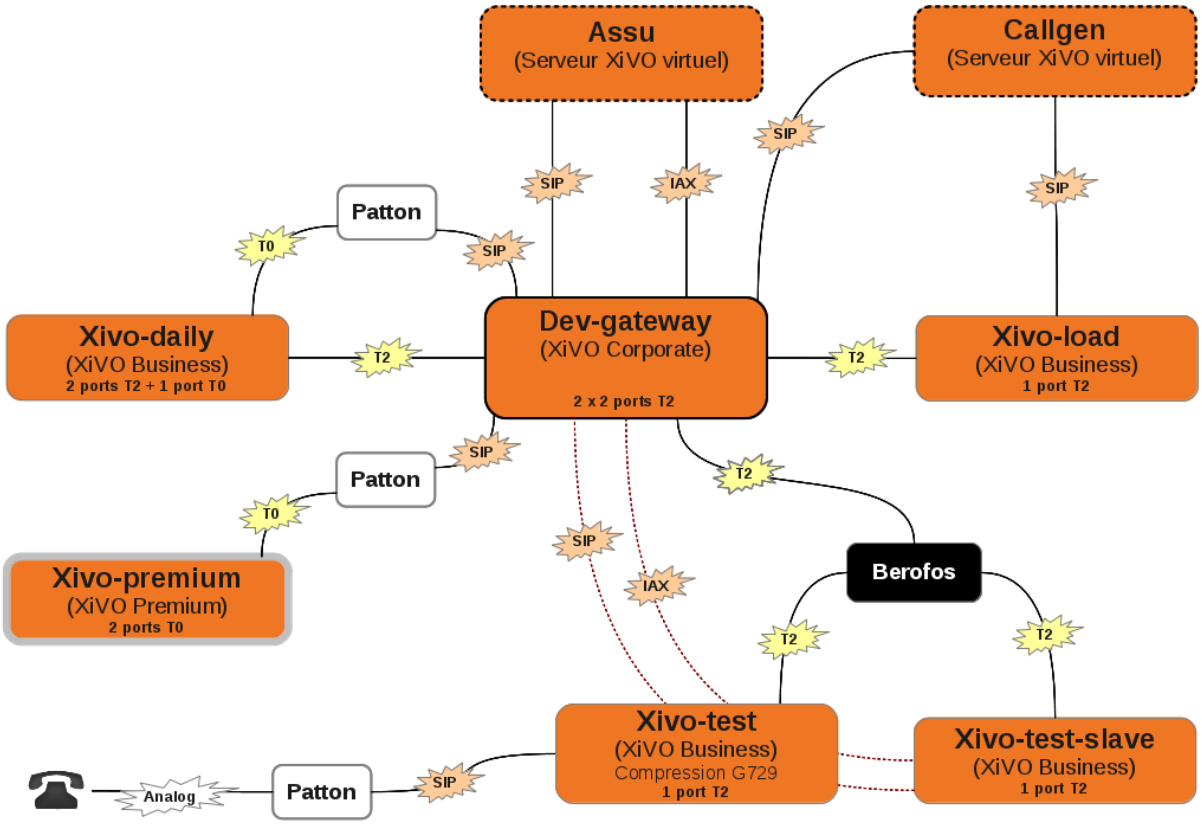


Fig. 1.109: Testing architecture

- xivo-premium (not yet installed) will allow us to test the new xivo-premium hardware

1.16 Troubleshooting

The list of current bugs can be found on the official Wazo issue tracker.

1.16.1 Transfers using DTMF

When transfering a call using DTMF (*1) you get an *invalid extension* error when dialing the extension.

The workaround to this problem is to create a preprocess subroutine and assign it to the destinations where you have the problem.

Under *Services → IPBX → IPBX configuration → Configuration files* add a new file containing the following dialplan:

```
[allow-transfer]
exten = s,1,NoOp(## Setting transfer context ##)
same = n,Set(__TRANSFER_CONTEXT=<internal-context>)
same = n,Return()
```

Do not forget to substitute <internal-context> with your internal context.

Some places where you might want to add this preprocess subroutine is on queues and outgoing calls to be able to transfer the called person to another extension.

1.16.2 Fax detection

Wazo **does not currently support Fax detection**. The following describe a workaround to use this feature. The behavior is to answer all incoming (external) call, wait for a number of seconds (4 in this example) : if a fax is detected, receive it otherwise route the call normally.

Note: This workaround works only :

- on incoming calls towards an User (and an User only),
- if the incoming trunk is a DAHDI or a SIP trunk,
- if the user has a voicemail which is activated and with the email field filled
- XiVO/Wazo >= 13.08 (needs asterisk 11)

Be aware that this workaround will probably not survive any upgrade.

---

1. In the Web Interface and under *Services → IPBX → IPBX configuration → Configuration files* add a new file named *fax-detection.conf* containing the following dialplan:

```
;; Fax Detection
[pre-user-global-faxdetection]
exten = s,1,NoOp(Answer call to be able to detect fax if call is external AND␣
↪user has an email configured)
same  =   n,GotoIf($["${XIVO_CALLORIGIN}" = "extern"]?:return)
same  =   n,GotoIf(${XIVO_USEREMAIL}?:return)
same  =   n,Set(FAXOPT(faxdetect)=yes) ; Activate dynamically fax detection
same  =   n,Answer()
same  =   n,Wait(4) ; You can change the number of seconds it will wait for fax␣
↪(4 to 6 is good)
```

```
same  =   n,Set(FAXOPT(faxdetect)=no) ; If no fax was detected deactivate␣
↪dyamically fax detection (needed if you want directmedia to work)
same  =   n(return),Return()

exten = fax,1,NoOp(Fax detected from ${CALLERID(num)} towards ${XIVO_DSTNUM} -␣
↪will be sent upon reception to ${XIVO_USEREMAIL})
same  =     n,GotoIf($["${CHANNEL(channeltype)}" = "DAHDI"]?
↪changeechocan:continue)
same  =     n(changeechocan),Set(CHANNEL(echocan_mode)=fax) ; if chan type is␣
↪dahdi set echo canceller in fax mode
same  =     n(continue),Gosub(faxtomail,s,1(${XIVO_USEREMAIL}))
```

2. In the file `/etc/xivo/asterisk/xivo_globals.conf` set the global user subroutine to `pre-user-global-faxdetection` : this subroutine will be executed each time a user is called:

```
XIVO_PRESUBR_GLOBAL_USER = pre-user-global-faxdetection
```

3. Reload asterisk configuration (both for dialplan and dahdi):

```
asterisk -rx 'core reload'
```

1.16.3 Berofos Integration with PBX

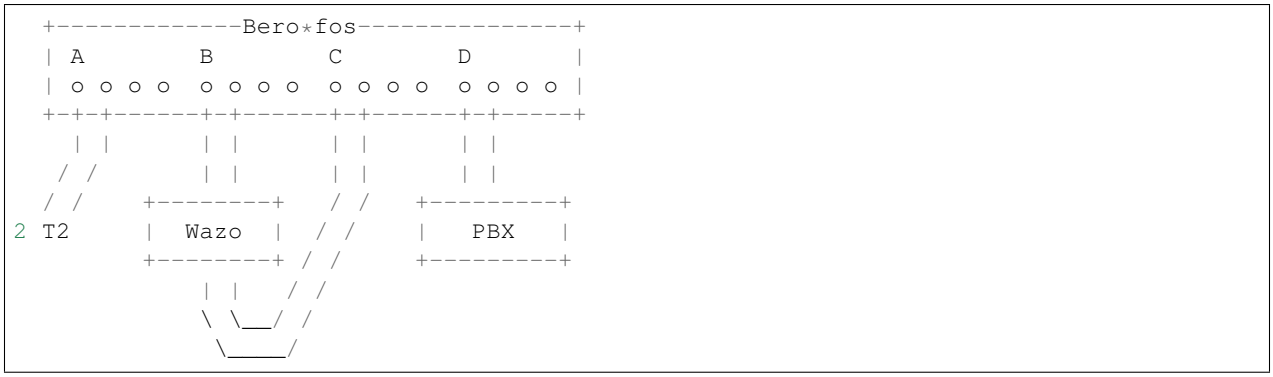
You can use a Berofos failover switch to secure the ISDN provider lines when installing a Wazo in front of an existing PBX. The goal of this configuration is to mitigate the consequences of an outage of the Wazo : with this equipment the ISDN provider links could be switched to the PBX directly if the Wazo goes down.

Wazo **does not offer natively** the possibility to configure Berofos in this failover mode. This section describes a workaround.

Logical view:



Connection:



The following describes how to configure your Wazo and your Berofos.

1. Follow the Berofos general configuration (firmware, IP, login/password) described in the the *Berofos Installation and Configuration* page.

2. When done, apply these specific parameters to the berofos:

```
bnfos --set scenario=1   -h 10.105.2.26 -u admin:berofos
bnfos --set mode=1       -h 10.105.2.26 -u admin:berofos
bnfos --set modedef=1    -h 10.105.2.26 -u admin:berofos
bnfos --set wdog=1       -h 10.105.2.26 -u admin:berofos
bnfos --set wdogdef=1    -h 10.105.2.26 -u admin:berofos
bnfos --set wdogitime=60 -h 10.105.2.26 -u admin:berofos
```

3. Add the following script `/usr/local/sbin/berofos-workaround`:

```
#!/bin/bash
# Script workaround for berofos integration with a Wazo in front of PABX

res=$(/usr/sbin/service asterisk status)
does_ast_run=$?
if [ $does_ast_run -eq 0 ]; then
    /usr/bin/logger "$0 - Asterisk is running"
    # If asterisk is running, we (re)enable wdog and (re)set the mode
    /usr/bin/bnfos --set mode=1 -f fos1 -s
    /usr/bin/bnfos --set modedef=1 -f fos1 -s
    /usr/bin/bnfos --set wdog=1 -f fos1 -s

    # Now 'kick' berofos ten times each 5 seconds
    for ((i == 1; i <= 10; i += 1)); do
        /usr/bin/bnfos --kick -f fos1 -s
        /bin/sleep 5
    done
else
    /usr/bin/logger "$0 - Asterisk is not running"
fi
```

4. Add execution rights to script:

```
chmod +x /usr/local/sbin/berofos-workaround
```

5. Create a cron to launch the script every minutes `/etc/cron.d/berofos-cron-workaround`:

```
# Workaround to berofos integration
MAILTO=""

*/1 * * * * root /usr/local/sbin/berofos-workaround
```

1.16.4 CTI server is unexpectedly terminating

If you observes that your CTI server is sometimes unexpectedly terminating with the following message in `/var/log/xivo-ctid.log`:

```
(WARNING) (main): AMI: CLOSING
```

Then you might be in the case where asterisk generates lots of data in a short period of time on the AMI while the CTI server is busy processing other thing and is not actively reading from its AMI connection. If the CTI server takes too much time before consuming some data from the AMI connection, asterisk will close the AMI connection. The CTI server will terminate itself once it detects the connection to the AMI has been lost.

There's a workaround to this problem called the ami-proxy, which is a process which buffers the AMI connection between the CTI server and asterisk. This should only be used as a last resort solution, since this increases the latency between the processes and does not fix the root issue.

To enable the ami-proxy, you must:

1. Add a file `/etc/systemd/system/xivo-ctid.service.d/ami-proxy.conf`:

```
mkdir -p /etc/systemd/system/xivo-ctid.service.d
cat >/etc/systemd/system/xivo-ctid.service.d/ami-proxy.conf <<EOF
[Service]
Environment=XIVO_CTID_AMI_PROXY=1
EOF
systemctl daemon-reload
```

2. Restart the CTI server:

```
systemctl restart xivo-ctid.service
```

If you are on a Wazo cluster, you must do the same procedure on the slave if you want the ami-proxy to also be enabled on the slave.

To disable the ami-proxy:

```
rm /etc/systemd/system/xivo-ctid.service.d/ami-proxy.conf
systemctl daemon-reload
systemctl restart xivo-ctid.service
```

1.16.5 Agents receiving two ACD calls

An agent can sometimes receive more than 1 ACD call at the same time, even if the queues he's in have the "ringinuse" parameter set to no (default).

This behaviour is caused by a bug in asterisk: https://issues.asterisk.org/jira/browse/ASTERISK-16115

It's possible to workaround this bug in Wazo by adding an agent *subroutine*. The subroutine can be either set globally or per agent:

```
[pre-limit-agentcallback]
exten = s,1,NoOp()
same  =   n,Set(LOCKED=${LOCK(agentcallback-${XIVO_AGENT_ID})})
same  =   n,GotoIf(${LOCKED}?:not-locked,1)
same  =   n,Set(GROUP(agentcallback)=${XIVO_AGENT_ID})
same  =   n,Set(COUNT=${GROUP_COUNT(${XIVO_AGENT_ID}@agentcallback)})
same  =   n,NoOp(${UNLOCK(agentcallback-${XIVO_AGENT_ID})})
same  =   n,GotoIf($[ ${COUNT} <= 1 ]?:too-many-calls,1)
same  =   n,Return()

exten = not-locked,1,NoOp()
same  =   n,Log(ERROR,Could not obtain lock)
same  =   n,Wait(0.5)
same  =   n,Hangup()

exten = too-many-calls,1,NoOp()
same  =   n,Log(WARNING,Not calling agent ID/${XIVO_AGENT_ID} because already in use)
same  =   n,Wait(0.5)
same  =   n,Hangup()
```

This workaround only applies to queues with agent members; it won't work for queues with user members.

Also, the subroutine prevent asterisk from calling an agent twice by hanguping the second call. In the agent statistics, this will be shown as a non-answered call by the agent.

1.16.6 PostgreSQL localization errors

The database and the underlying database cluster used by Wazo is sensitive to the system locale configuration. The locale used by the database and the database cluster is set when Wazo is installed. If you change your system locale without particular attention to PostgreSQL, you might make the database and database cluster temporarily unusable.

When working with locale and PostgreSQL, there's a few useful commands and things to know:

- `locale -a` to see the list of currently available locales on your system
- `locale` to display information about the current locale of your shell
- `grep ^lc_ /etc/postgresql/9.6/main/postgresql.conf` to see the locale configuration of your database cluster
- `sudo -u postgres psql -l` to see the locale of your databases
- the `/etc/locale.gen` file and the associated `locale-gen` command to configure the available system locales
- `systemctl restart postgresql.service` to restart your database cluster
- the PostgreSQL log file located at `/var/log/postgresql/postgresql-9.6-main.log`

Note: You can use any locale with Wazo as long as it uses an UTF-8 encoding.

Database cluster is not starting

If the database cluster doesn't start and you have the following errors in your log file:

```
LOG:  invalid value for parameter "lc_messages": "en_US.UTF-8"
LOG:  invalid value for parameter "lc_monetary": "en_US.UTF-8"
LOG:  invalid value for parameter "lc_numeric": "en_US.UTF-8"
LOG:  invalid value for parameter "lc_time": "en_US.UTF-8"
FATAL:  configuration file "/etc/postgresql/9.6/main/postgresql.conf" contains errors
```

Then this usually means that the locale that is configured in `postgresql.conf` (here `en_US.UTF-8`) is not currently available on your system, i.e. does not show up the output of `locale -a`. You have two choices to fix this issue:

- either make the locale available by uncommenting it in the `/etc/locale.gen` file and running `locale-gen`
- or modify the `/etc/postgresql/9.6/main/postgresql.conf` file to set the various `lc_*` options to a locale that is available on your system

Once this is done, restart your database cluster.

Can't connect to the database

If the database cluster is up but you get the following error when trying to connect to the `asterisk` database:

```
FATAL:  database locale is incompatible with operating system
DETAIL:  The database was initialized with LC_COLLATE "en_US.UTF-8",  which is not␣
↪recognized by setlocale().
HINT:  Recreate the database with another locale or install the missing locale.
```

Then this usually means that the database locale is not currently available on your system. You have two choices to fix this issue:

- either make the locale available by uncommenting it in the `/etc/locale.gen` file, running `locale-gen` and restarting your database cluster
- or *recreate the database using a different locale*

Error during the upgrade

Then you are mostly in one of the cases described above. Check your log file.

Error while restoring a database backup

If during a database restore, you get the following error:

```
pg_restore: [archiver (db)] Error while PROCESSING TOC:
pg_restore: [archiver (db)] Error from TOC entry 4203; 1262 24745 DATABASE asterisk␣
↪asterisk
pg_restore: [archiver (db)] could not execute query: ERROR:  invalid locale name: "en_
↪US.UTF-8"
    Command was: CREATE DATABASE asterisk WITH TEMPLATE = template0 ENCODING = 'UTF8'␣
↪LC_COLLATE = 'en_US.UTF-8' LC_CTYPE = 'en_US.UTF-8';
```

Then this usually means that your database backup has a locale that is not currently available on your system. You have two choices to fix this issue:

- either make the locale available by uncommenting it in the `/etc/locale.gen` file, running `locale-gen` and restarting your database cluster
- or if you want to restore your backup using a different locale (for example `fr_FR.UTF-8`), then restore your backup using the following commands instead:

```
sudo -u postgres dropdb asterisk
sudo -u postgres createdb -l fr_FR.UTF-8 -O asterisk -T template0 asterisk
sudo -u postgres pg_restore -d asterisk asterisk-*.dump
```

Error during master-slave replication

Then the slave database is most likely not using an UTF-8 encoding. You'll need to *recreate the database using a different locale*

Changing the locale (LC_COLLATE and LC_CTYPE) of the database

If you have decided to change the locale of your database, you must:

- make sure that you have enough space on your hard drive, more precisely in the file system holding the `/var/lib/postgresql` directory. You'll have, for a moment, two copies of the `asterisk` database.
- prepare for a service interruption. The procedure requires the services to be restarted twice, and the system performance will be degraded while the database with the new locale is being created, which can take a few hours if you have a really large database.
- make sure the new locale is available on your system, i.e. shows up in the output of `locale -a`

Then use the following commands (replacing `fr_FR.UTF-8` by your locale):

```
wazo-service restart all
sudo -u postgres createdb -l fr_FR.UTF-8 -O asterisk -T template0 asterisk_newlocale
sudo -u postgres pg_dump asterisk | sudo -u postgres psql -d asterisk_newlocale
wazo-service stop
sudo -u postgres psql <<'EOF'
DROP DATABASE asterisk;
ALTER DATABASE asterisk_newlocale RENAME TO asterisk;
EOF
wazo-service start
```

You should also modify the `/etc/postgresql/9.6/main/postgresql.conf` file to set the various `lc_*` options to the new locale value.

For more information, consult the official documentation on PostgreSQL localization support.

1.16.7 Originate a call from the Asterisk console

It is sometimes useful to ring a phone from the asterisk console. For example, if you want to call the `1234` extension in context `default`:

```
channel originate Local/1234@default extension 42@xivo-callme
```

1.16.8 Network packets capture

In some extreme cases, packet capture may be very useful to find out what is happening between Wazo and other equipment (phones, trunks, etc.)

Local capture, for later analysis:

```
# change interface eth0 and filter 'udp port 5060' as you wish
tcpdump -i eth0 -w /tmp/wazo.pcap udp port 5060
```

Remote packet capture, streamed to Wireshark via SSH:

```
# install dumpcap on the server wazo.example.com
ssh wazo.example.com apt-get install -y wireshark-common

# run the capture on interface eth0, for SIP packets only (UDP port 5060)
wireshark -k -i <(ssh wazo.example.com "dumpcap -P -i eth0 -w - -f 'udp port 5060'")
```

1.16.9 Getting help

Sometimes it's just not possible to fix a problem by yourself. In that case, you will most likely need to get help from someone outside your network.

ngrok can be used to give access to someone outside your network to your Wazo server.

To make that possible, you will have to follow these 4 easy steps.

- Create an account on ngrok
- Install ngrok on your Wazo server:

On a 32 bit server:

```
wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-386.zip
unzip ngrok-stable-linux-386.zip
```

On a 64 bit server:

```
wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
unzip ngrok-stable-linux-amd64.zip
```

- Add your ngrok token (given when you signed up on ngrok)

```
./ngrok authtoken <YOUR AUTH TOKEN>
```

- Add SSH and HTTPS access in your *ngrok* config

```
cat << EOF >> ~/.ngrok2/ngrok.yml
tunnels:
  webi:
    addr: 443
    proto: tcp
  ssh:
    addr: 22
    proto: tcp
EOF
```

- Start *ngrok*

```
./ngrok start --all
```

The output will show the public URL and ports that are now available to access you server. For example:

```
tcp://0.tcp.ngrok.io:12345 -> localhost:22
tcp://0.tcp.ngrok.io:9876 -> localhost:443
```

means:

- anyone can use this command to SSH into your machine: `ssh root@0.tcp.ngrok.io -p 12345`
- anyone can access the web interface via: `https://0.tcp.ngrok.io:12346`.

To stop *ngrok* hit Ctrl-C.

Note: The ngrok tunnel will not survive a reboot of the server, you'll have to set it up again after restart.

Warning: This setup is a typical scenario for a man-in-the-middle attack. If you don't trust the Ngrok servers, you should ensure that:

- the HTTPS certificate is the right one, i.e. it has the same fingerprint:
  - on the server: `openssl x509 -text -noout -in /usr/share/xivo-certs/server.crt -sha256 -fingerprint | grep Fingerprint`
  - in the browser, check the details of the certificate to see the fingerprint
- the SSH key fingerprint of the server is correct, when SSH asks you upon the first connection (TOFU)

1.16.10 Collecting logs

When troubleshooting a problem, you may need to send logs for analysis.

`wazo-debug-collect` simplifies the gathering of logs:

```
apt-get update
apt-get install wazo-debug
wazo-debug-collect -o /tmp/logs.tar.gz
```

`wazo-debug-collect` will gather all the logs of the different Wazo daemons, including Asterisk, and bundle them into a tarball. You may then send this tarball for analysis.

Warning: Be careful before sending the logs in a public place: they may contain sensible information, that can be used to connect to your Wazo.

1.17 Community Documentation

This page provides links to resources on various topics around Wazo. They have been generously created by people from the community.

1.17.1 Tutorials

Please note that these resources are provided on an "as is basis". They have not been reviewed by the Wazo team, therefore the information presented may be innaccurate. We also accept resources provided in other languages besides English.

Unless specified, the license is CC BY-SA.

Tutorial	Language	Level	Author
What is XiVO and tutorial (video)	English	Beginner	XiVO
Xivo pour les nuls	French	Beginner	Nicolas
Installing XiVO (YouTube series)	English	Beginner	VoIP-Nui
Start: how to create a user with a SIP line (YouTube series)	French	Beginner	VoIP-Nui
Start: how to popup an URL (Document)	French	Beginner	
Start: how to create a context, users, voicemails, ring group, music on hold, conf.call	French	Beginner	Networkl
Tips: post-installation of XiVO on Kimsufi	French	Intermediate	LabCella
Tips: username and password on XiVO	French	Intermediate	LabCella
Tips: self-hosting and telephony with XiVO	French	Intermediate	LabCella
XiVO provisioning + pfSense + siproxd + OVH	French	Intermediate	LabCella
SCCP provisioning, unsupported phones and no DHCP	French	Intermediate	LabCella
Date format on SCCP 7941	French	Intermediate	LabCella
Installing XiVO on Raspberry Pi (Raspivo)	French	Intermediate	Iris Netw
`How to popup an url with CTIClient`	French	Intermediate	Assonanc
`How to backup XiVO to external FTP` with `backup-ftp.sh`	French	Intermediate	Yohan Vi
`How to create a Wazo Client`	French	Intermediate	Yohan Vi
`How to configure a C610P IP on XiVO`	French	Intermediate	Yohan Vi
`How to export the phonebook of XiVO` with phonebook_csv_export.py	French	Intermediate	Yohan Vi
`How to use openVPN on XiVO`	French	Expert	Yohan Vi

Table 1.11 – continued from previous page

Tutorial	Language	Level	Author
How configure SNOM M700 DECT	French	Intermediate	Jonathan
Scripted provisioning for SNOM M700 DECT with specific scripts	French	Intermediate	Jonathan
How to configure XiVO with Untangle firewall	English	Intermediate	Scott Mc
How to use Keepalived with XiVO (high availability)	English	Expert	Eric Viel
Getting Started with XiVO	English	Beginner	Nerd Vitt
Function key redirects calls to a DID/user towards sound file	French	Intermediate	Yohan Vi
Function key redirects calls to a DID/user towards extension	French	Intermediate	Yohan Vi
Function key redirects calls to a DID/user towards voicemail	French	Intermediate	Yohan Vi
Play music when user is called from DID	French	Intermediate	Yohan Vi
Reverse lookup from a text file	French	Intermediate	TiJof & Y
Wazo star codes (en)	English	Intermediate	Ward Mu
Wazo star codes (fr)	French	Intermediate	Thomas I
Configuring FOP2 with Wazo	English	Intermediate	Richard C

1.17.2 Contribute

We gladly accept new contributions. There are two ways to contribute:

- The preferred way: open a pull request on Github and add a line to this page (see: *Contributing to the Documentation*).
- You can also open a contribution ticket on the bug tracker.

Note that we only accept documents in open formats, such as PDF or ODF.

1.18 Documentation changelog

1.19 Attribution Notice

The major part of this documentation has been copied (2016-11-25) from the XiVO documentation. That documentation was licensed under the Create Commons Attribution-ShareAlike 4.0 International License and was copyrighted 2012-2016 Avencall.

CHAPTER 2

Changelog

The *Documentation changelog* is available.

CHAPTER 3

Indices and tables

- genindex
- search

Index

C

ctiserver, 203

D

devices, 217

I

Identity, 78
interconnections, 263, 265, 267
interconnections/simonics, 274

M

mail, 93

N

network, 94

P

People, 81

S

Service, 84

U

users, 335

V

VLAN, 97

W

Wazo Client, 70, 516, 517, 519
wizard, 5

X

Xlets, 71