Wazo Documentation

Release 19.16

The Wazo Authors

Nov 18, 2019

Contents

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 existing 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 new documentation at http://www.wazo-platform.org/install

1.2.2 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:

- `PUT /asterisk/sip/general`
  - `trustrpid`: `yes`
  - `sendrpid`: `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 wazo-agid:

```
service wazo-agid restart
```

Time and date

- Configure your locale and default time zone device template with `wazo-provd` endpoint `/provd/cfg_mgr/config` by editing the default template
- 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* `alaw` instead of both `alaw` and `ulaw` by default):

- `PUT /asterisk/sip/general`
  - `allow`: `alaw,g722,g729,h264`
- `PUT /asterisk/iax/general`
  - `allow`: `alaw,g722,g729,h264`

1.3 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:

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

1.3.1 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.

1.3.2 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`.

1.3.3 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)

1.3.4 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 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-platform/wazo-
↪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-platform/wazo-
↪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
   ```

Note: Upgrading to a specific version between 18.03 and 19.12 is not supported

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-platform/wazo-
↪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
   ```

Note: Upgrading to a specific version between 18.03 and 19.12 is not supported

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 < 19.04

Those procedures are valid if your Wazo installation is newer than 18.01 and older than 19.04.

Upgrade to Wazo <= 18.03

Example to upgrade to Wazo 18.03:

```
wazo-dist -a wazo-18.03
apt-get update
apt-get install xivo-upgrade/wazo-18.03
wazo-upgrade
wazo-dist -m phoenix-stretch
```

Upgrade to Wazo < 19.12

Not supported

Upgrade to Wazo < 19.13

Example to upgrade to Wazo 19.12:

```
wazo-upgrade
```

Upgrade to Wazo >= 19.13

Example to upgrade to Wazo 19.13:

```
wazo-upgrade
```

This will upgrade your Wazo to 19.12. From there:

1. Read the *upgrade notes*
2. Upgrade to Wazo 19.13:

   ```
   wazo-dist-upgrade -t wazo-19.13
   wazo-dist -m pelican-buster
   ```

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 pelican-stretch
wazo-upgrade
```

Upgrade from Wazo > 19.12

Those procedures are valid if your Wazo installation is newer than 19.12

Upgrade to Wazo > 19.12

Example to upgrade to Wazo 19.13:

```
wazo-dist -a wazo-19.13
apt-get update
apt-get install wazo-upgrade/wazo-19.13
wazo-upgrade
wazo-dist -m pelican-buster
```

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 pelican-buster
wazo-upgrade
```

1.3.5 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.

1.3.6 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) 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

1.3.7 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.

1.3.8 Upgrade notes

Upgrade notes

19.16

- `xivo-amid-client` has been renamed to `wazo-amid-client`
- `wazo-auth` http configuration section have been moved onto the rest_api section, eg:

```
rest_api:
  https:
    listen: <ip>
    port: <port>
    certificate: </path/to/cert>
    private_key: </path/to/key>
```

  becomes:

```
rest_api:
  listen: <ip>
  port: <port>
  certificate: </path/to/cert>
  private_key: </path/to/key>
```

- The default value for Asterisk PJSIP configuration parameter `rtptimeout` has been set to 7200 seconds on new installs only. The change was done to automatically delete ghost calls that might get stuck. If you wish to modify this value, use the `/asterisk/sip/general` endpoint in `wazo-confd` API.

19.15

- We have standardize the stevedore entry point namespace for our python client. If you have custom plugins, Be sure to use the full client name for the namespace. (e.g. `auth_client.commands` –> `wazo_auth_client.commands`)
- The directed call pickup extension `*8XXXX` has been disabled by default on new installations, because it made it possible for any user to pickup any other user, including users for whom it should not be possible. This does not apply to upgrades, but if you wish to disable this feature, you can do it with `wazo-confd /extensions/features` API endpoint.

Consult the 19.15 Roadmap for more information.

19.14

- A new version (v2) of websocket protocol has been created. See *Wazo WebSocket* for more information

  The v1 is now deprecated and should not be used anymore. Also it does not return the attribute `msg` in all payloads as it was always empty.

- `xivo-confgend` has been renamed to `wazo-confgend`
  - The custom configuration files have been moved to `/etc/wazo-confgend/conf.d`
  - The log file has been renamed to `wazo-confgend.log`
  - The plugin entry points have been renamed from `xivo` to `wazo`. Plugins enabled in custom configuration files should use the new name.
  - The entry point identifier has been changed from `xivo_confgend` to `wazo_confgend`. If you have developed custom plugins for confgend you should use the new identifier in your `setup.py`.
- `xivo-confgend-client` has been renamed to `wazo-confgend-client`

  - If you used the `xivo-confgen` CLI tool you will now have to use `wazo-confgen`

- If you are upgrading a Wazo that was originally installed in 18.03 or earlier, the old directory configuration is now replaced with a new profile `default` for each tenant. The migration of the old directory configuration must be done manually, since there is no way to automatically detect the tenant for each directory configuration. To allow this migration, the old configuration is dumped in `/var/backups/xivo/dird_sources.yml` during the upgrade to Wazo Platform 19.14. The administrator must then recreate the directory configuration manually using the API or web interface.
- There is a known bug that will remove pre-recorded sound files provided by the `xivo-sounds-*`, .e.g `xivo-sounds-fr-ca`. If you had installed one of these packages manually, you need to install the corresponding `wazo-sounds-*` package manually, e.g. `wazo-sounds-fr-ca`. Upgrades to Wazo >= 19.15 are not affected by this bug.

Consult the 19.14 Roadmap for more information.

19.13

- **Debian has been upgraded from version 9 (stretch) to 10 (buster).** Please consult the following detailed upgrade notes for more information:

Debian 10 (Buster) Upgrade Notes

The upgrade to Wazo 19.13 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 9.6 to version 11 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 version of Wazo is at least 18.01. You can run `wazo-upgrade` to check the version currently installed. If your version of Wazo is older that 18.01, you should first upgrade your Wazo to Debian Stretch, following the procedure described in *Debian 9 (stretch) Upgrade Notes*.
- 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.
- Remove the `freeradius` package. If you have recompiled Asterisk on you server you most likely installed the `libfreeradius-dev` package, which pulled `freeradius`. This package cannot be confiugred on Debian Buster under some circunstances that are not under our control. You can remove it with the following command `apt purge freeradius`
- If you have customized the Debian system of your Wazo 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 `stretch-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:

- `wazo-dist -m pelican-stretch`: 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`: Upgrades to the latest version of Wazo with Debian 10 (Buster). 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.

External Links

- Official Debian 10 release notes

- `xivo-amid` has been renamed to `wazo-amid`
  - The custom configuration has been moved to `/etc/wazo-amid/conf.d/`.
  - The log file has been renamed to `wazo-amid.log`.

  - The NGINX proxy has been recreated in `/etc/nginx/locations/https-enabled/wazo-amid`.

Consult the 19.13 Roadmap for more information.

19.12

General

- All administration interfaces `xivo-web-interface` and `wazo-admin-ui` have been removed. They are replaced by `wazo-ui`. To install it, run the following command after the upgrade: `apt install wazo-ui`.
- The Wazo Client and `xivo-ctid` have been removed.
- `wazo-dird` is now configured using its REST API. The previous configuration files have been removed and a new profile `default` is now created for each new tenant.
- *Entity* concept has been replaced by *Tenant*. The previous concept was not completely sealed and we have fixed it with the *tenant*.
  - Existing devices are migrated automatically to the tenant of their first associated line. If a device is in autoprov mode, it will be migrated to the default tenant. See *Introduction* for more information on how device tenants are handled.
  - Agents are now multi-tenant. Agents created using the rest API that were not logged into a queue and that were not associated to a user have been deleted.
  - Skills are migrated to the tenant of the agent with whom they are associated. If a skill was not associated with an agent, it has been deleted.
  - All the existing skill rules have been associated to the tenant of the first queue found in the database. If there were no queue configured in the system, the skill rules have been deleted.
  - Call logs are now multi-tenant. Each call log that cannot be associated to a tenant has been associated to the `master` tenant. Also for all call logs created after the upgrade, if the tenant cannot be extracted from call informations, they will be associated to the master tenant.

Migration of sound files to tenants

In Wazo 19.03, sound files are now segregated by tenant (a.k.a entity). However, Wazo has no way to know which entity owns which sound file. Thus a manual intervention is required to make those sound files available to tenants.

Sound files include:

  - queue announces (`acd`)
  - telephony feature sounds, like autoprovisioning message, transfer messages, etc. (`features`)
  - recordings of sounds, conversations and conferences (`recordings` and `monitor`)
  - custom sounds used for IVR or dialplan (`playback`)

How to migrate

The sound files are stored in `/var/lib/xivo/sounds`, for example:

```
root@wazo:~# tree /var/lib/xivo/sounds
/var/lib/xivo/sounds
├── acd
│   ├── tenant-bakery-queue-announce.wav
│   └── tenant-grocery-queue-announce.wav
├── features
│   ├── tenant-bakery-autoprov.wav
│   └── tenant-vacuumcleaners-autoprov.wav
├── monitor -> ../../../spool/asterisk/monitor
├── playback
└── recordings -> ../../asterisk/sounds/custom
```

In order to make the sound files available to tenants, you need to move the files in a `tenants` subdirectory, like this:

```
root@wazo:~# tree /var/lib/xivo/sounds
/var/lib/xivo/sounds
└── tenants
    ├── 3176b5c5-a765-4dcc-81a6-e69e29081d66
    │   ├── acd
    │   │   └── tenant-bakery-queue-announce.wav
    │   ├── features
    │   │   └── tenant-bakery-autoprov.wav
    │   ├── monitor
    │   ├── playback
    │   └── recordings
    ├── 62770df9-4451-4b99-a1d3-ccf48881b173
    │   ├── acd
    │   │   └── tenant-grocery-queue-announce.wav
    │   ├── features
    │   ├── monitor
    │   ├── playback
    │   └── recordings
    └── cc85438a-8e79-417f-b713-f05e1529d132
        ├── acd
        ├── features
        │   └── tenant-vacuumcleaners-autoprov.wav
        ├── monitor
        ├── playback
        └── recordings
```

Each subdirectory of the `tenants` directory must be named like the UUID of each tenant. In order to know the UUID of tenants, you can use the `wazo-auth-cli` command:

```
root@wazo:~# wazo-auth-cli tenant list -c uuid -c name
+--------------------------------------+----------------+
| uuid                                 | name           |
+--------------------------------------+----------------+
| 80ef6d2e-2f70-4934-a02b-bdabcdf48495 | master         |
| 3176b5c5-a765-4dcc-81a6-e69e29081d66 | bakery         |
| 62770df9-4451-4b99-a1d3-ccf48881b173 | grocery        |
| cc85438a-8e79-417f-b713-f05e1529d132 | vacuumcleaners |
+--------------------------------------+----------------+
```

You can safely ignore the `master` tenant, which is used internally by Wazo.

You should move sounds files of each tenant for the following directories:

- `acd`

  - `features`
  - `monitor`
  - `playback`
  - `recordings`
  - We needed to do some guesswork for ambiguous resources that shared other resources from different entities. These resources have been migrated to the most logical tenants. However, it may be possible that they are still associated to resources that were migrated to different tenants. When this happens, you need to fix them manually and to make sure to remove the affected resources or to recreate them in the right tenants. Even if they still work, these configurations are invalid and shall be removed automatically in future upgrades. Therefore, you should review the following resources:
    * call permissions
    * ivr
    * moh
    * pagings
- User authentication has been updated with the following changes:
  - User passwords cannot be returned in plain text anymore.
  - Users export (export CSV) cannot export passwords anymore.
  - Processing time to import users (import CSV) has been increased significantly if the field `password` is provided.
  - Fields `username` and `password` in `wazo-confd` API `/users` are now ignored for authentication and must be considered invalid. They have been replaced by `wazo-auth` API.
  - Field `enabled` for `wazo-confd` API `/users/<user_id>/cti` is now ignored for authentication and must be considered invalid. It has been replaced by `wazo-auth` API.
- Invalid user email address (e.g. `invalid@`) have been deleted automatically during upgrade.
- All agents will have to log out and log back in to receive calls from queues. You may use the command `wazo-agentd-cli -c "relog all"` to do this.
- The procedure for custom certificates, especially for Let's Encrypt certificates, has been simplified. See *Certificates for HTTPS*.
- People using the `xivo-aastra-2.6.0.2019` will have to upgrade to plugin version 1.9.2 or later
- `wazo-provd` now uses YAML configuration. The defaults can be overridden in the `/etc/wazo-provd/conf.d/` directory. See *Configuration Files*.
- The provisioning option *DHCP Integration* is now enabled by default. There is no REST API to disable this feature.
- Call pickups that have been created using the REST API or `wazo-admin-ui` have the interceptors and targets mixed up. Since call pickups created using the "orange" web-interface did not have that bug, we could not fix the existing configuration automatically. Faulty call pickups have to be edited and users moved from interceptors to targets and vice versa.
- Since the feature for managing certificates from the "orange" web-interface is gone, all certificates must now be managed manually. The directory to access to certificates is `/var/lib/xivo/certificates` and is not backuped or synchronized for HA anymore.
- If a group or queue was named `general`, then it has been renamed with one or more suffix `_` (e.g. `general_`). The name `general` is not allowed anymore.

- `xivo-sysconfd` is now asynchronous by default. This implies that changes made via the API or via a web interface may take some time to take effect after the action. If you rely on Asterisk being reloaded when configuring resources. See *Configuration File* to set the `synchronous` option to `true`.
- Upgrade from version older than 15.01 are not supported anymore.
- If a custom context (created using the REST API or wazo-admin-ui) was named with the following names, then it has been renamed with one or more suffix _. Also if the context name had invalid characters (i.e. space), then invalid characters are replaced by _. All custom configuration should be updated to reflect the changes.
  - *authentication*
  - *general*
  - *global*
  - *globals*
  - *parkedcalls*
  - *xivo-features*
  - *zonemessages*
- The `wazo-google` and `wazo-microsoft` plugins have been copied to the `wazo-auth` and `wazo-dird` repo. You **must** uninstall that plugin if you installed it manually from source to avoid conflicts between the supported version and the legacy version.

Asterisk related

- Asterisk version has been updated:

Asterisk 15 to 16 Upgrade Notes

You might be impacted by the upgrade to Asterisk 16 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 16.

In particular, if you are using custom Asterisk modules, you'll need to either obtain the Asterisk 16 version of these modules or recompile them against Asterisk 16. Not doing so usually leads to major instability issues in Asterisk.

The nova compatibility patch has been removed. If you have a file enabling nova_compatibility in your cel.conf or cel.d/* you will have to remove that line from your confguration.

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

- https://wiki.asterisk.org/wiki/display/AST/Upgrading+to+Asterisk+16
- https://github.com/asterisk/asterisk/blob/16/CHANGES

- Wazo now uses `res_pjsip` instead of `chan_sip`.
  - All custom lines with interface `SIP/something` must be changed to `PJSIP/something`

  - All custom dialplan using the `SIP_HEADER` dialplan function must be changed to `PJSIP_HEADER` function
  - The `SIPAddHeader` and `SIPRemoveHeader` dialplan application must be changed to `PJSIP_HEADER` function
- The username for all SIP devices in `autoprov` mode has been changed. Devices in `autoprov` mode will have to be restarted before entering the provisioning code.
- Asterisk configuration files can now be customized in the `/etc/asterisk/*.d/` directories. If you had custom configuration in `/etc/asterisk/*.conf` you will have to create a new file in the corresponding `*.d` directory to use your customized configuration. Files named `*.conf.dpkg-old` will be left in `/etc/asterisk` if this operation is required. See *Asterisk configuration files* for more details.
- The skill rules internal names have been changed to use the format `skillrule-<id>`. If you were using custom dialplan with a preprocess subroutine to handle your skill rules, we recommend removing it and using the REST API (see *Apply Skill Rule Sets*). If you really want to keep it, you must change the name used in the variable `XIVO_QUEUESKILLRULESET` to use the new format.
- Asterisk logs (`/var/log/asterisk/full`) now contain milliseconds
- The `tenant_name` variable has been removed from the call recording templates in favor of the `tenant_uuid`. If the `tenant_name` was used in the directory name, a symlink can be used to keep the same name.

Renaming

- The following services have been renamed:
  - `xivo-agentd` to `wazo-agentd`
  - `xivo-agid` to `wazo-agid`
  - `xivo-confd` to `wazo-confd`
  - `xivo-ctid-ng` to `wazo-calld`
  - `xivo-dird` to `wazo-dird`
  - `wazo-dird-phoned` to `wazo-phoned`
  - `xivo-provd` to `wazo-provd`
  - `xivo-nginx` to `wazo-nginx`
- Each service has the following changes:
  - The custom configuration has been moved to `/etc/<new-service-name>/conf.d/`.
  - The log file has been renamed to `<new-service-name>.log`.
  - The NGINX proxy has been recreated in `/etc/nginx/locations/https-enabled/<new-service-name>`
  - Entrypoints for custom Python plugins have been renamed to `<new_service_name.*`.
  - Environment variable for `wazo-upgrade` has been renamed from `XIVO_CONFD_PORT` to `WAZO_CONFD_PORT`.
  - All users that are logged in Wazo, i.e. who have an authentication token, must logout and log back in, to apply the change of authorizations names (ACL).
- The following Python clients have been renamed. If you were using the old one in your Python code you should use the new one.

  - xivo-agentd-client to wazo-agentd-client
  - xivo-confd-client to wazo-confd-client
  - xivo-dird-client to wazo-dird-client
  - xivo-provd-client to wazo-provd-client
- xivo-agentd-cli has been renamed to wazo-agentd-cli
- xivo-provd-cli has been renamed to wazo-provd-cli
- xivo-dhcpd-update has been renamed to wazo-dhcpd-update
- The fail2ban jail was renamed from asterisk-xivo to asterisk-wazo.
- Chat messages, user and device presences are now handled by wazo-chatd instead of wazo-calld and MongooseIM.
  - All chat messages will be deleted after the upgrade.
- The /var/lib/xivo/sounds directory has been migrated to /var/lib/wazo/sounds and the directory /var/lib/xivo is considered deprecated. Please update all custom references to this path.

Developers

- The following daemons have been updated to Python 3. If you have written or installed a custom plugin for those daemons, you must ensure that the plugins are compatible with Python 3.
  - wazo-auth
  - wazo-calld
  - wazo-confd
  - wazo-dird
- The following backends in wazo-auth have been removed. All following users have been migrated to wazo_user backend.
  - xivo_admin
  - xivo_service
  - xivo_user
- wazo-auth API to implement a wazo-auth backend has been changed in 18.02. The compatibility code that allowed old backends to keep working has been removed.
  - The get_ids method has been removed.
- ACL templating has been modified: when generating multiple ACLs with one template, ACL were separated with \n. They are now separated with : (colon). \n is not interpreted anymore. You should hence replace any \n with : in your ACLs.
- wazo-provd now uses wazo-auth to authenticate all requests and uses HTTPS. It is no longer possible to deactivate authentication. Therefore, all calls to the REST API will need to be made using HTTPS and a token generated with wazo-auth.
- wazo-provd-cli has been updated to remove the username and password command line arguments since they are no longer used.
- The configuration of rest_api section for wazo-confd configuration file has changed. See wazo-confd changelog 19.06 for more information.

- All API related to `cti profile` have been removed. See wazo-confd changelog 19.08 for more information.
- Creating a resource using the REST API now requires the `Wazo-Tenant` HTTP header when the created resource is not in the same tenant as its creator.
- Authentication policies now have a `tenant_uuid` and the relationship between tenants and policies has been removed. If you did use policies with tenant association, the policy is now associated to one of its tenant. This feature is not used yet in Wazo, so most likely you are not affected.
- `wazo-confd` REST API does not allow to manage `call-logs` anymore.
- `wazo-provd` API URL has been updated to remove the `provd` prefix when present and add the API version number, which is `0.2`. All affected services and `wazo-provd-client` have been updated. Example: `/provd/dev_mgr` is now `/0.2/dev_mgr` and `/api/api.yml` is now `/0.2/api/api.yml`

Consult the roadmaps for more information:

- 18.04
- 18.05
- 18.06
- 18.07
- 18.08
- 18.09
- 18.10
- 18.11
- 18.12
- 18.13
- 18.14
- 19.01
- 19.02
- 19.03
- 19.04
- 19.05
- 19.06
- 19.07
- 19.08
- 19.09
- 19.10
- 19.11
- 19.12

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 removed.
  - 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: plugin.old.py

```
class Plugin(object):

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

Listing 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.

Archives

Archived Upgrade Notes

2017

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 3: old.yml

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

Listing 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 5: old.yml

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

Listing 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.
- 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).

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.

2016

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-platform/wazo-
↪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-platform/wazo-
↪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 switchboard_configuration_multi_queues.
- 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.
- 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.

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 *wazo-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

- 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

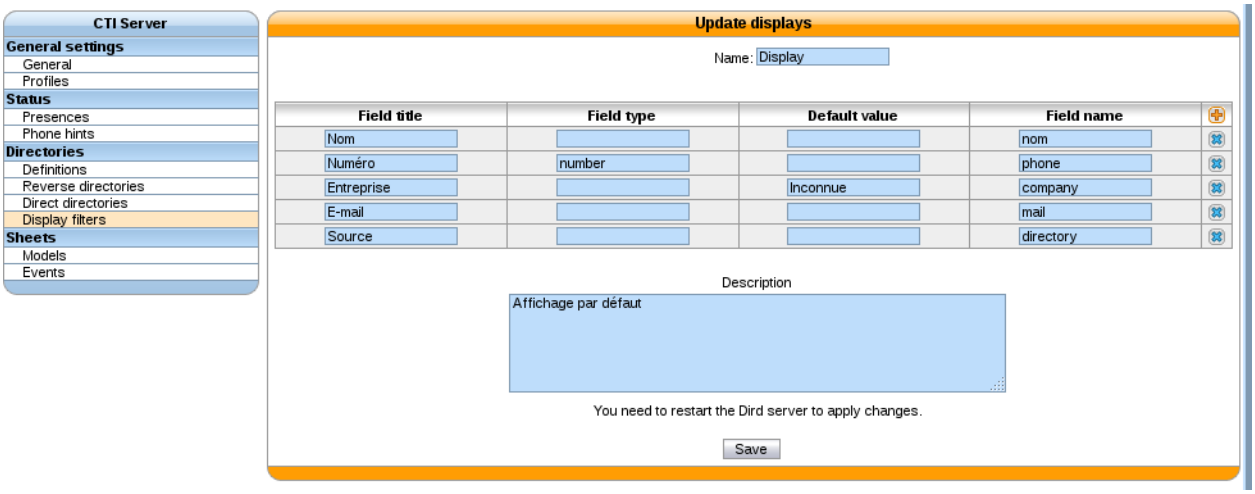
- It is now possible to send an email to a user with a configured email address in the *people* xlet. See dird-integration-views to add the appropriate field to your configured displays.
- The *Contacts* xlet (aka. *Search*) has been removed in favor of the people-xlet. You may need to do some manual 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

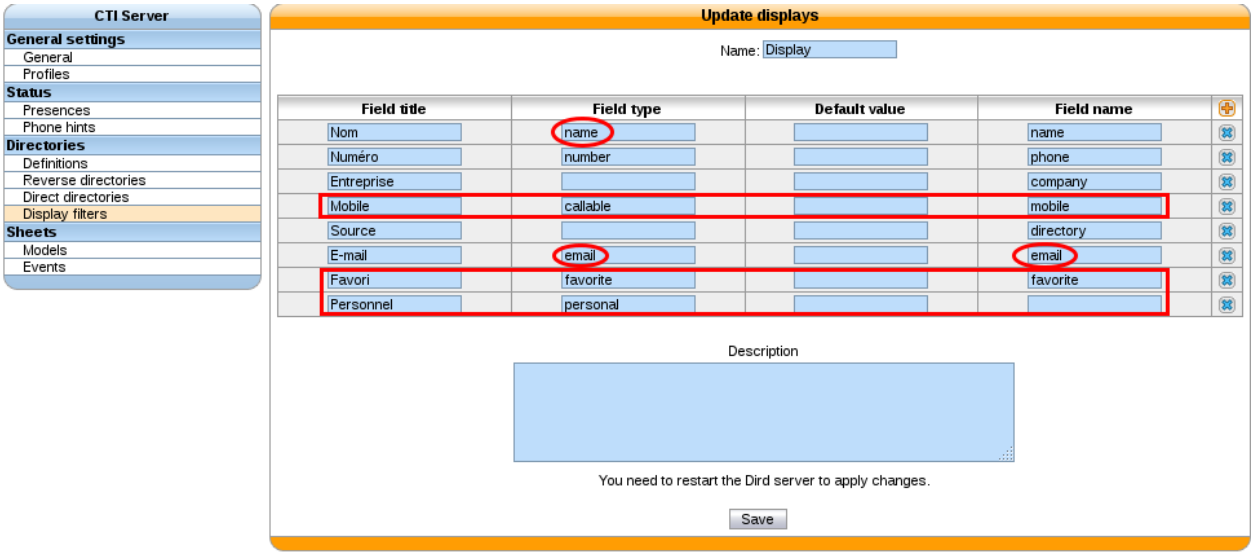
When upgrading your XiVO to 15.19, there are some features in the directories that could not be upgraded automatically, because it risked breaking some manual configurations.

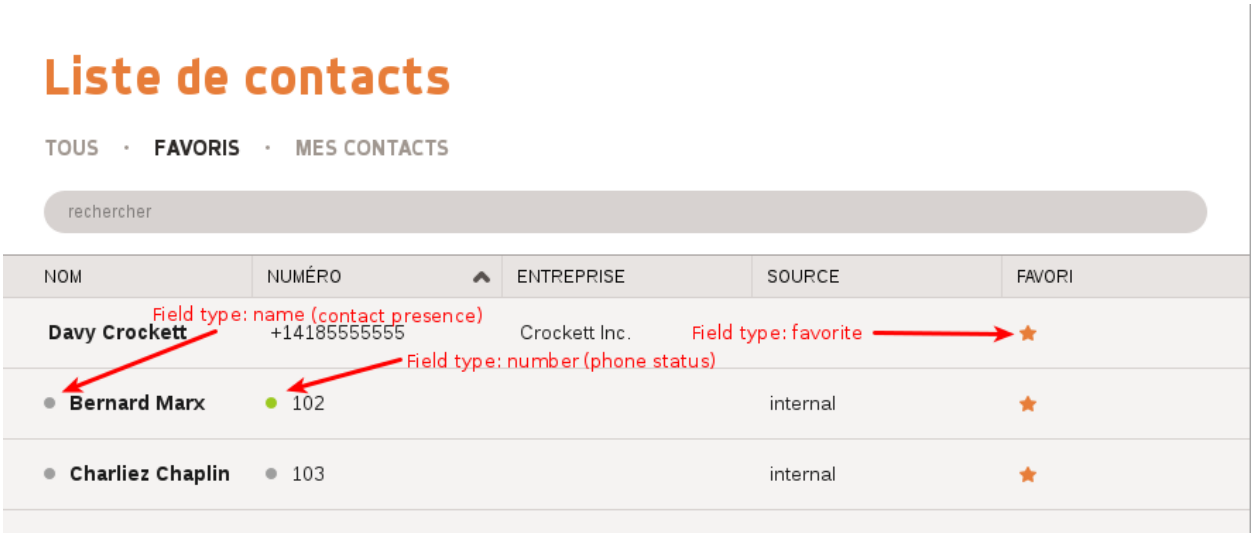
After you upgrade your XiVO, your CTI displays in *Services* → *CTI Server* → *Directories* → *Displays* may look like this:

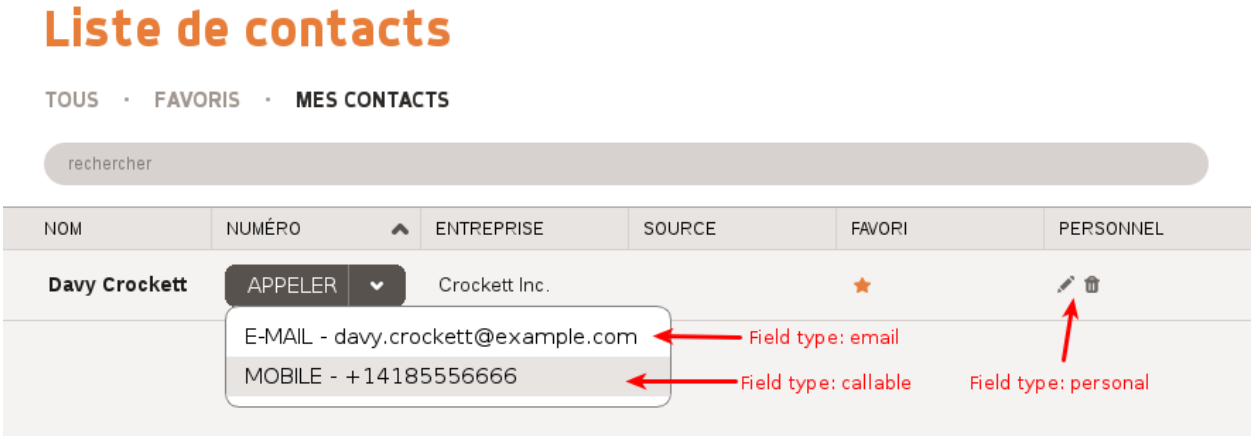


You should update your displays to make them look like:

This will give you a Xlet People looking like this:


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

Context separation

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

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]
```

(continues on next page)

(continued from previous page)

```
    __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
```

(continues on next page)

(continued from previous page)

```
    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.
- 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

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 *wazo-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 *wazo-dird-integration* 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.

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.

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 *wazo-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`

1.4 System

1.4.1 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 `/dhcp` endpoint

By default, it will only answer to DHCP requests coming from the VoIP subnet. If you need to activate DHCP on an other interface, you have to fill in the `network_interfaces` field with the interface name , for example : `eth0`

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.

1.4.2 Network

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>
```

1.4.3 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

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/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-agentd/
- /etc/wazo-agid/
- /etc/wazo-amid/
- /etc/wazo-auth/
- /etc/wazo-call-logd/
- /etc/wazo-calld/
- /etc/wazo-chatd/
- /etc/wazo-confd/
- /etc/wazo-confgend-client/
- /etc/wazo-phoned/
- /etc/wazo-dird/

- /etc/wazo-plugind/
- /etc/wazo-purge-db/
- /etc/wazo-webhookd/
- /etc/wazo-websocketd/
- /etc/wazo-dxtora/
- /etc/xivo/
- /root/.config/wazo-auth-cli/
- /usr/local/bin/
- /usr/local/sbin/
- /usr/share/wazo/WAZO-VERSION
- /var/lib/asterisk/
- /var/lib/consul/
- /var/lib/wazo/
- /var/lib/wazo-auth-keys/
- /var/lib/wazo-provd/
- /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/wazo-provd/plugins/*/var/cache/*
  - /var/spool/asterisk/monitor/
  - /var/spool/asterisk/meetme/
- files
  - /var/lib/wazo-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/wazo/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)

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
```

1.4.4 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**

Before Restoring the System

Warning: Before restoring a Wazo on a fresh install you have to setup Wazo using the wizard.

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
```

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.

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

After Restoring The System

1. 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`

2. You may reboot the system, or execute the following steps.

3. Update systemd runtime configuration:

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

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

   ```
   wazo-service start
   ```

1.4.5 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.

This article is about the certificate used for HTTPS.

Wazo and HTTPS

Wazo uses HTTPS mainly for receiving and responding to REST API calls. The REST API calls can occur inside the Wazo Engine, i.e. between Wazo daemons, or outside the Wazo Engine, e.g. for the web interface or any other application based on the REST APIs.

From the outside of the Wazo Engine, every API call is reverse-proxied by nginx, which listens on port 443 (HTTPS) and distribute REST API calls to the right daemon. This means we only have to change one certificate (the one used by nginx) to enable all APIs to be secured by this certificate from the outside of the Wazo Engine.

The default HTTPS certificate used by the Wazo Engine is located in `/usr/share/xivo-certs/server.crt` with its associated `server.key` private key.

Let's Encrypt

To create a new certificate for your Wazo Engine via Let's Encrypt for the domain `wazo-engine.example.com`, here is the procedure:

```
apt install python3-certbot-nginx
certbot --nginx -d wazo-engine.example.com
# answer the questions
systemctl restart nginx
```

For more details, see the official Certbot documentation.

Use your own certificate

You will need:

1. the private key used to create your certificate (here named `/usr/local/share/private-key.pem`)
2. the full-chain certificate. It must include all intermediate certificates used in the chain of trust (here named `/usr/local/share/certificate.fullchain.pem`). See the nginx documentation for more details.
3. Both files **must** be readable by the group `www-data`. You can check with the following command:

   ```
   sudo -u www-data cat /usr/local/share/*.pem > /dev/null
   ```

Edit the file `/etc/nginx/sites-available/xivo` and replace the following keys:

```
ssl_certificate /usr/share/xivo-certs/server.crt;
ssl_certificate_key /usr/share/xivo-certs/server.key;
```

with:

```
ssl_certificate /usr/local/share/certificate.fullchain.pem;
ssl_certificate_key /usr/local/share/private-key.pem
```

Then restart nginx:

```
systemctl restart nginx
```

Revert previous custom HTTPS certificate configuration

Up to Wazo 18.03, the procedure to install a custom HTTPS certificate was much more complex. This complex procedure is not needed anymore and should be removed to avoid any conflict with future upgrade. You can use the following removal procedure before of after the above configuration steps.

Here is the removal procedure:

```
# backup your certificate / key (optional)
cp /usr/share/xivo-certs/server.{key,crt} /var/backups

# stop all Wazo Engine services
wazo-service stop all

# regenerate self-signed certificate
rm /usr/share/xivo-certs/server.{key,crt}
dpkg-reconfigure xivo-certs

# remove custom config files
rm /etc/xivo/custom/custom-certificate.yml
rm /etc/{wazo,xivo}-*/conf.d/010-custom-certificate.yml
rm /etc/xivo/custom-templates/system/etc/hosts

# restart services
xivo-update-config
wazo-service restart all
```

Then, the last steps:

- update your directories of type `wazo` to use:
  - the domain `localhost`
  - the certificate located in `/usr/share/xivo-certs/server.crt`

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

wazo-agentd

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

wazo-amid

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

wazo-confgend

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

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.

wazo-phoned

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

wazo-provd

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

wazo-websocketd

- Default configuration directory: `/etc/wazo-websocketd/conf.d`
- Default configuration file: `/etc/wazo-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 `wazo-agid` service:

```
service wazo-agid restart
```

Asterisk configuration files

Asterisk configuration files are located at */etc/asterisk*. These files are packaged with Wazo and you should not modify files that are located at the root of this directory.

To add you own configurations, you must add a new configuration file in the corresponding *.d* directory.

For example, if you need to add a new user to the *manager.conf* configuration file, you would add a new file */etc/asterisk/manager.d/my_new_user.conf* with the following content:

```
.. code-block: ini
```

[my_new_user] secret=v3ry5ecre7 deny=0.0.0.0/0.0.0.0 permit=127.0.0.1/255.255.255.0 read = system

The same logic applies to all Asterisk configuration files except *asterisk.conf* and *modules.conf*.

1.4.7 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
```

1.4.8 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-agid

- File location: `/var/log/wazo-agid.log`
- Rotate configuration: `/etc/logrotate.d/wazo-agid`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-calld

- File location: `/var/log/wazo-calld.log`
- Rotate configuration: `/etc/logrotate.d/wazo-calld`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-dird

- File location: `/var/log/wazo-dird.log`
- Rotate configuration: `/etc/logrotate.d/wazo-dird`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-upgrade

- File location: `/var/log/wazo-upgrade.log`
- Rotate configuration: `/etc/logrotate.d/wazo-upgrade`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-agentd

- File location: `/var/log/wazo-agentd.log`
- Rotate configuration: `/etc/logrotate.d/wazo-agentd`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-amid

- File location: `/var/log/wazo-amid.log`
- Rotate configuration: `/etc/logrotate.d/wazo-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

wazo-confd

- File location: `/var/log/wazo-confd.log`
- Rotate configuration: `/etc/logrotate.d/wazo-confd`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-confgend

The wazo-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 wazo-confgend.service unit file.

- File location: `/var/log/wazo-confgend.log`
- Rotate configuration: `/etc/logrotate.d/wazo-confgend`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-phoned

- File location: `/var/log/wazo-phoned.log`
- Rotate configuration: `/etc/logrotate.d/wazo-phoned`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-dxtora

- File location: `/var/log/wazo-dxtora.log`
- Rotate configuration: `/etc/logrotate.d/wazo-dxtora`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-provd

- File location: `/var/log/wazo-provd.log`
- Rotate configuration: `/etc/logrotate.d/wazo-provd`
- Number of archived files: 15
- Rotation frequence: Daily

wazo-purge-db

- File location: `/var/log/wazo-purge-db.log`
- Rotate configuration: `/etc/logrotate.d/wazo-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

wazo-websocketd

- File location: `/var/log/wazo-websocketd.log`
- Rotate configuration: `/etc/logrotate.d/wazo-websocketd`
- Number of archived files: 15
- Rotation frequence: Daily

1.4.9 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 (wazo-agentd)
- The authentication server (wazo-auth)
- The configuration server (wazo-confd)
- The telephony service interface (wazo-calld)
- The directory service (wazo-dird)
- The AMI HTTP interface (wazo-amid)
- API documentation (xivo-swagger-doc)
- The websocket interface (wazo-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/*
systemctl reload nginx
```

1.4.10 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.

1.4.11 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 with `wazo-provd` endpoint `/provd/configuration/http_proxy`.

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.

1.4.12 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.

1.4.13 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 wazo-auth.
2. *Create a token* with these credentials.
3. *Use this token* to access the REST resource defined by the *ACL*.

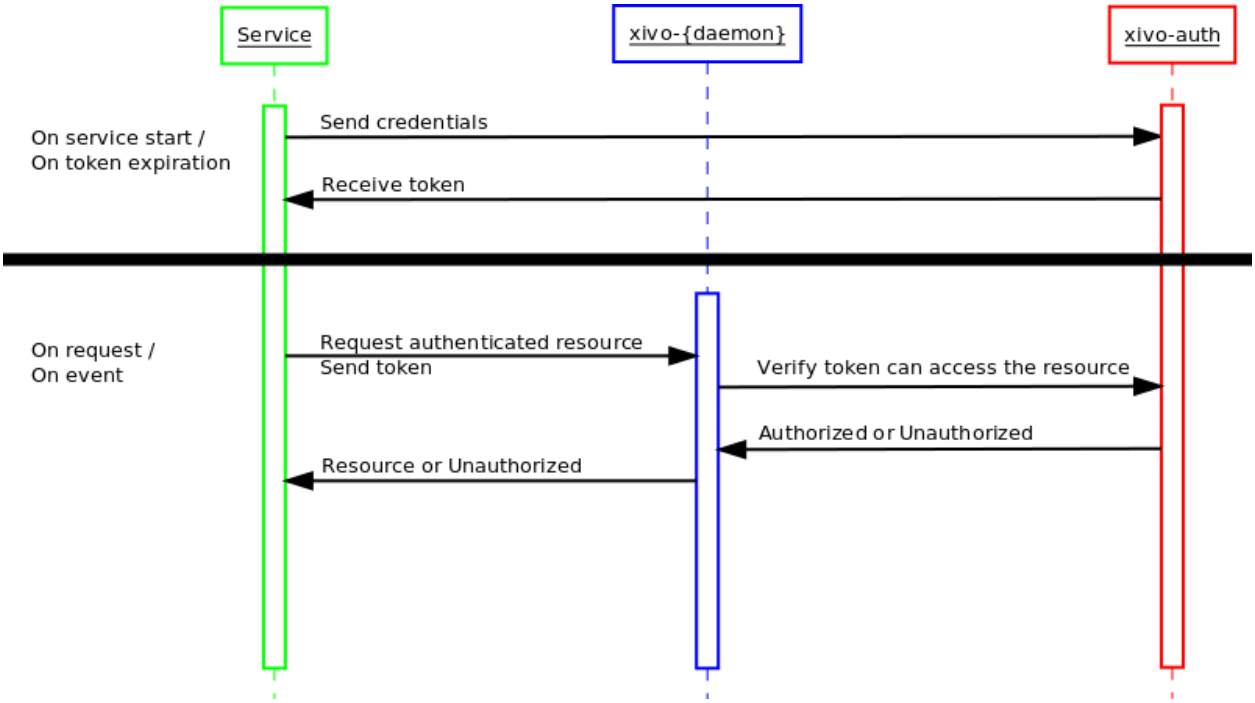


Fig. 1: Call flow of service authentication

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.

1.4.14 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.

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 7: setup.py

```
#!/usr/bin/env python3
# -*- 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 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'
```

(continues on next page)

(continued from previous page)

```
    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'])
      }
```

(continues on next page)

(continued from previous page)

```
        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
- 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 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
```

(continues on next page)

(continued from previous page)

```
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 }}.#:{% endfor %}
dird.me.#
{% if agent %}agentd.agents.by-id.{{ agent.id }}.read{% endif %}
```

Note: When using `for` loops to create ACL, make sure to add a `:` separator at the end of each ACL

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*.

1.4.15 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.

1.4.16 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:

```

(continues on next page)

(continued from previous page)

```
    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',
            },
        })
```

1.4.17 wazo-confd

wazo-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 (wazo-confd)

wazo-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:

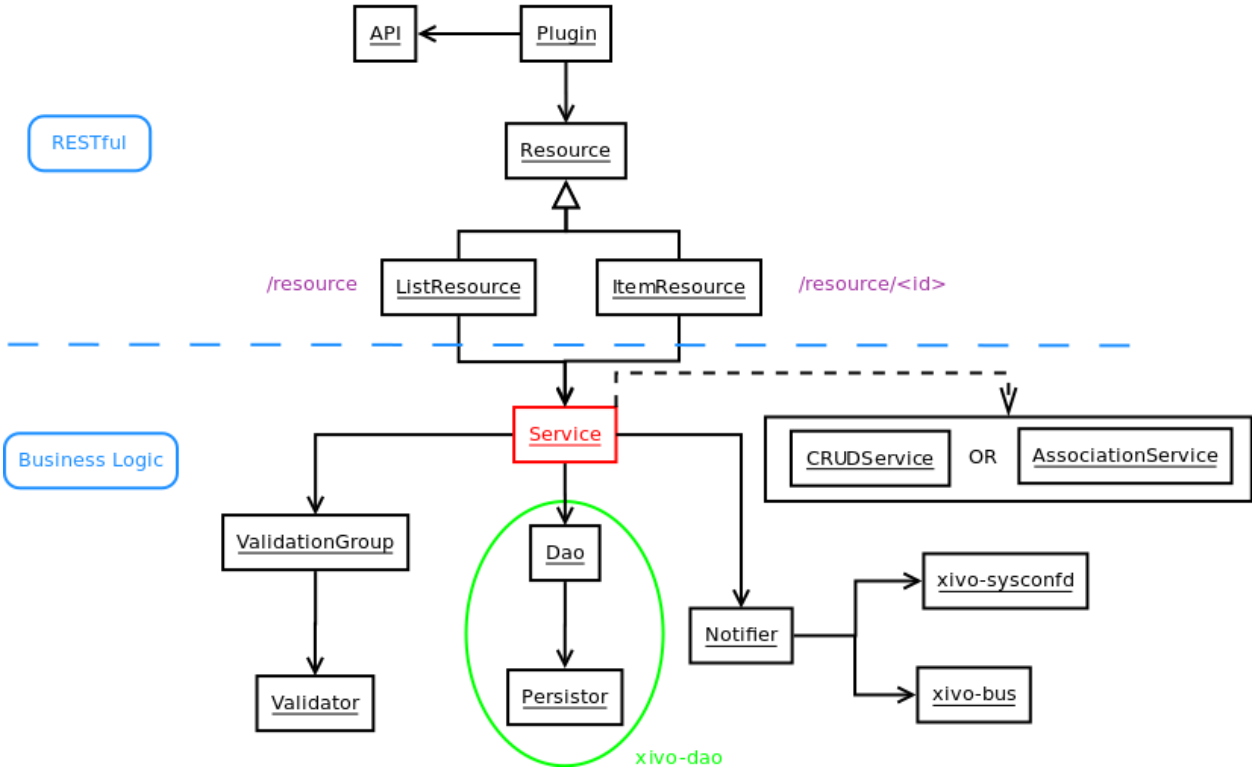


Fig. 2: Plugin architecture of wazo-confd

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.

1.4.18 wazo-confgend

wazo-confgend is a configuration file generator. It is mainly used to generate the Asterisk configuration files.

Wazo confgend developer's guide

wazo-confgend uses drivers to implement the logic required to generate configuration files. It uses stevedore to do the driver instantiation and discovery.

Plugins in wazo-confgend use setuptools' entry points. That means that installing a new plugin to wazo-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 wazo-confgend. This allows the driver implementor to add values to the configuration in `/etc/wazo-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 wazo-confgend have the following form:

    wazo_confgend.<resource>.<filename>

as an example, a generator for sip.conf would have the following namespace:

    wazo_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-or-later

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={
```

(continues on next page)

(continued from previous page)

```
        'wazo_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-or-later


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/wazo-confgend/conf.d`:

   ```
   plugins:
     asterisk.sip.conf: my_driver
   ```

3. Restart wazo-confgend:

   ```
   systemctl restart wazo-confgend.service
   ```

1.4.19 wazo-dird

wazo-dird is the directory server for Wazo. It offers a simple REST interface to query all directories that are configured. wazo-dird is extendable with plugins.

wazo-dird configuration

There are three sources of configuration for wazo-dird:

- the *command line options*
- the main configuration file

- configuration done using the API

The command-line options have priority over the main configuration file options.

Main Configuration File

Default location: `/etc/wazo-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/wazo-dird.log
log_level: info
pid_filename: /run/wazo-dird/wazo-dird.pid
user: www-data

enabled_plugins:
   backends:
       csv: true
       ldap: true
       phonebook: true
   services:
       lookup: true
   views:
       cisco_view: true
       default_json: true
```

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/wazo-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 wazo-dird instances. Default: `/run/wazo-dird/wazo-dird.pid`.

user The owner of the process. Default: `www-data`.

enabled_plugins section

This sections controls which plugins are to be loaded at wazo-dird startup. All plugin types must have at least one plugin enabled, or wazo-dird will not start. For back-end plugins, sources using a back-end plugin that is not enabled will be ignored.

wazo-dird developer's guide

The wazo-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.

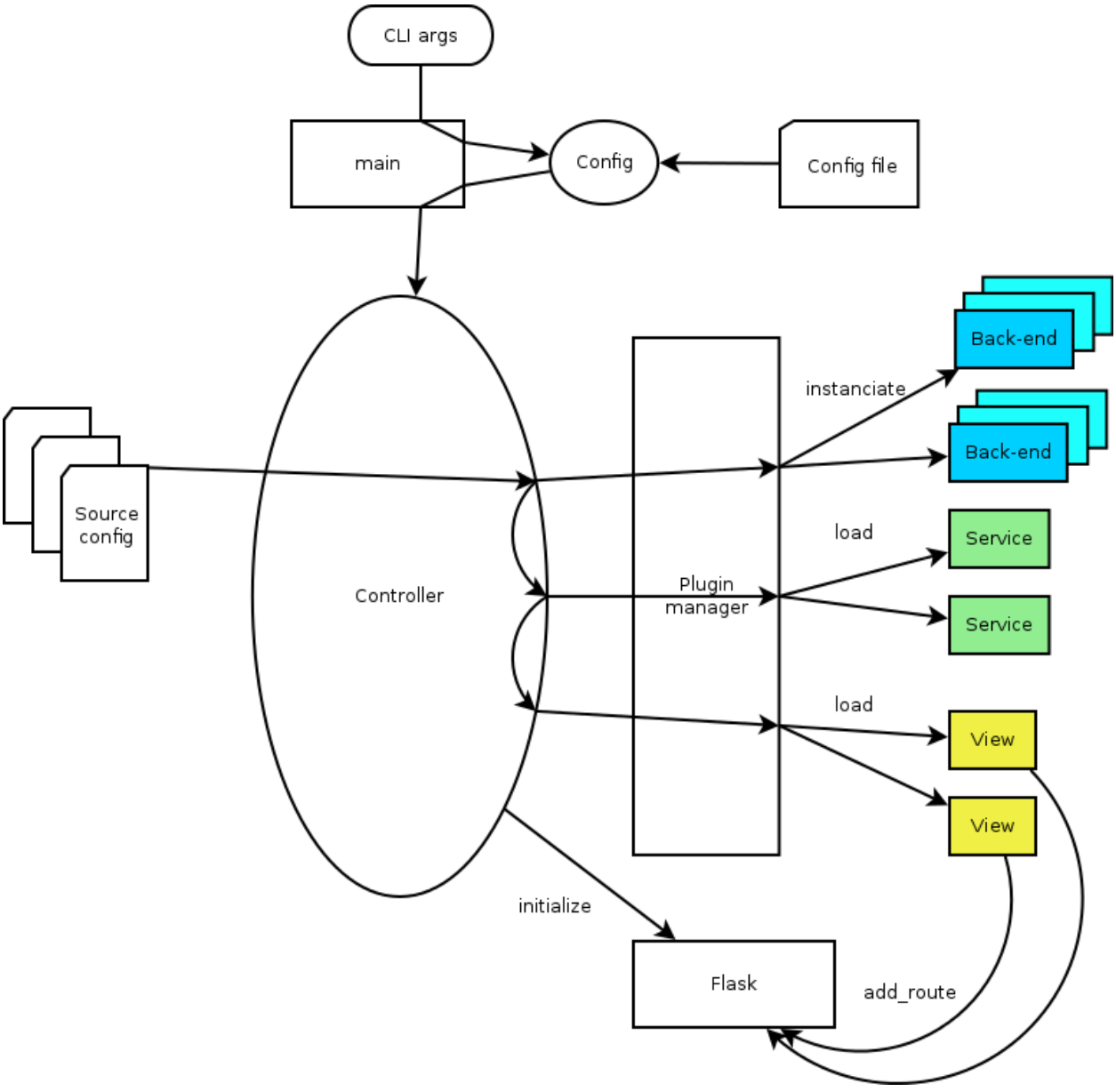


Fig. 3: wazo-dird startup flow

Plugins in wazo-dird use setuptools' entry points. That means that installing a new plugin to wazo-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 wazo-dird to be able to discover and load extensions packaged with wazo-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*

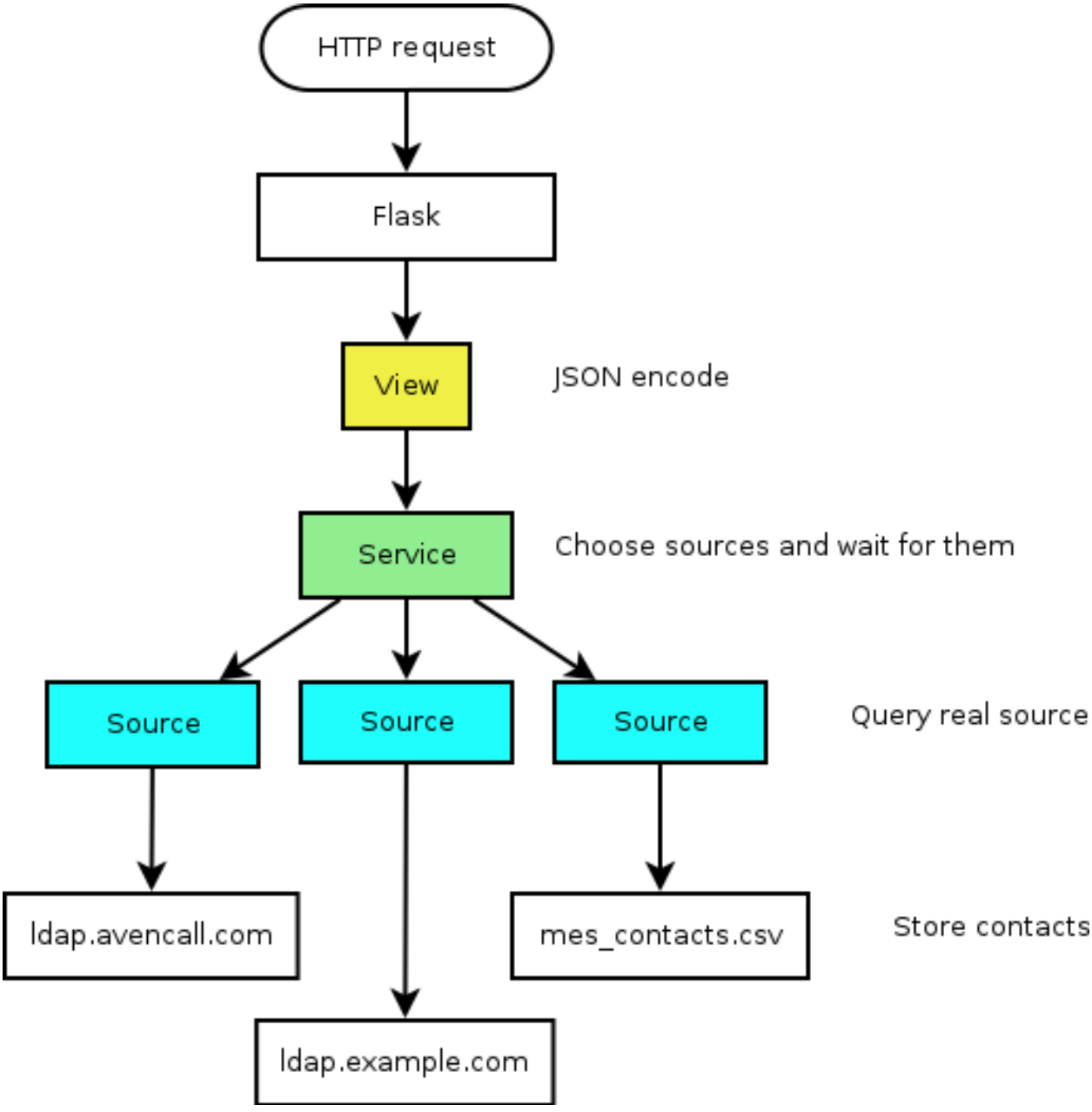


Fig. 4: wazo-dird HTTP query

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 wazo-dird:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from setuptools import setup
from setuptools import find_packages


setup(
    name='Wazo dird plugin sample',
    version='0.0.1',

    description='An example program',

    packages=find_packages(),

    entry_points={
        'wazo_dird.services': [
            'my_service = dummy:DummyServicePlugin',
        ],
        'wazo_dird.backends': [
            'my_backend = dummy:DummyBackend',
        ],
        'wazo_dird.views': [
            'my_view = dummy:DummyView',
        ],
    }
)
```

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: `wazo_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 wazo-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*)

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))]
```

(continues on next page)

(continued from previous page)

```
        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: `wazo_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 wazo_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
    """
```

(continues on next page)

(continued from previous page)

```
15
16      def load(self, args):
17          """
18          Ignores all provided arguments and instantiate a DummyService that
19          is returned to the core
20          """
21          logger.info('dummy loaded')
22          self._service = DummyService()
23          return self._service
24
25      def unload(self):
26          logger.info('dummy unloaded')
27
28
29  class DummyService(object):
30      """
31      A very dumb service that will return an empty list every time it is used
32      """
33
34      def list(self):
35          """
36          This function must be called explicitly from the view, `list` is not a
37          special method name for wazo-dird
38          """
39          return []
```

View

View plugins add new routes to the HTTP application in wazo-dird, in particular the REST API of wazo-dird: they define the URLs to which wazo-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: `wazo_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:

```
# -*- 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 wazo-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.

Note: Service discovery is limited to a single service being discovered. This means that discovering a wazo-confd server will assume that wazo-auth resides on the same host or that the template is already configured with the appropriate hostname.

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: wazo
name: wazo-{{ uuid }}
searched_columns:
- firstname
- lastname
first_matched_columns:
- exten
auth:
  host: {{ hostname }}
  port: 9497
  username: {{ service_id }}
  password: {{ service_key }}
  verify_certificate: false
confd:
  host: {{ hostname }}
  port: {{ port }}
  version: "1.1"
  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/wazo-dird/templates.d
    services:
      wazo-confd:
        template: confd.yml
```

In this example, the file */etc/wazo-dird/templates.d/confd.yml* would be used to create a new source configuration when a new *wazo-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

- service_id: The login used to query wazo-confd
- service_key: The password used to query wazo-confd

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

Launching wazo-dird

```
usage: wazo-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/wazo-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 wazo-dird. It is a way to add or modify the functionality of wazo-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.

1.4.20 wazo-phoned

wazo-phoned is an interface to use directory service with phones. It offers a simple REST interface to authenticate phones and to search results from *wazo-dird*.

Usage

wazo-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 needs to send two parameters:

- *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*

Launching wazo-phoned

On the command line, type `wazo-phoned -h` to see how to use it.

1.4.21 wazo-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, `wazo-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, wazo-purge-db removes all logs older than a year (365 days), except for webhookd logs where only 30 days are kept. wazo-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.

Records Purged

The following features are impacted by wazo-purge-db:

- *Call Logs*
- Call center statistics

More technically, `wazo-purge-db` have a set of plugins, each plugin are responsible of certain type of record (usually a postgresql table).

The format of the following list is `plugin-name` (`associated table`) :

- `call-log` (`call_log`)
- `cel` (`cel`)
- `queue-log` (`queue_log`)
- `stat-agent` (`stat_agent_periodic`)
- `stat-call` (`stat_call_on_queue`)
- `stat-queue` (`stat_queue_periodic`)
- `stat-switchboard` (`stat_switchboard_queue`)
- `webhookd-logs` (`webhookd_subscription_log`)

Configuration File

We recommend to override the setting `days_to_keep` from `/etc/wazo-purge-db/config.yml` in a new file in `/etc/wazo-purge-db/conf.d/`.

The `days_to_keep` configuration can be done per plugin if needed, by setting `days_to_keep_per_plugin` for example:

```
days_to_keep_per_plugin:
    webhookd-logs: 30
```

Warning: Setting `days_to_keep` to 0 will NOT disable `wazo-purge-db`, and will remove ALL logs from your system.

See *Configuration priority* and `/etc/wazo-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:

```
wazo-purge-db
```

You can specify the number of days of logs to keep. For example, to purge entries older than 365 days:

```
wazo-purge-db -d 365
```

Usage of `wazo-purge-db`:

```
usage: wazo-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 `wazo-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 `wazo-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 wazo-purge-db, you may install plugins to wazo-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/wazo-purge-db/conf.d/`.

There is an example plugin in the wazo-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 `wazo-purge-db`. Here is an example of file added in `/etc/wazo-purge-db/conf.d/`:

```
enabled_plugins:
    archives:
        - sample
```

sample.py

The following example will be save a file in `/tmp/wazo_purge_db.sample` with the following content:

```
Save tables before purge. 365 days to keep!
```

```
sample_file = '/tmp/wazo_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 wazo-purge-db:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from setuptools import setup
from setuptools import find_packages


setup(
    name='wazo-purge-db-sample-plugin',
    version='0.0.1',

    description='An example program',
    packages=find_packages(),
    entry_points={
        'wazo_purge_db.archives': [
            'sample = wazo_purge_db_sample.sample:sample_plugin',
        ],
    }
)
```

1.4.22 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 (hostname, DNS)
- configuring high availability
- staring/stopping/restarting services
- reloading asterisk configuration
- sending some events to components (wazo-agentd)

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

[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

[monit]
monit_checks_dir = /usr/share/xivo-monitoring/checks
monit_conf_dir = /etc/monit/conf.d

[request_handlers]
synchronous = false

[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 wazo-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.

1.5 Ecosystem

1.5.1 Devices

The supported devices are expected to work across upgrades and phone features should work on the latest version.

Supported Devices

`wazo-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

Continu

Table 2 – continued from previous page

	6731i	6735i	6737i	6739i	6755i
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
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

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 3 – continued from previous page

	6863i	6865i	6867i
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
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

Table 4 – continued from previc

	7905G	7906G	7911G	7912G	7920
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/wazo-provd/plugins/$plugin-name/var/cache` directory.

File permissions should be modified to make the files readable to *wazo-provd*:

- *chmod 640 <filename>*
- *chown wazo-provd:wazo-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-$locale-name.cop.sgn, for example "po-locale-fr_FR.cop.sgn" for the "fr_FR" locale
- Both files must be placed in `/var/lib/wazo-provd/plugins/$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

Continued on next page

Table 5 – continued from previous page

	D40	D50	D70
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
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

	|SoundPoint IP		
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 `notifycid` is `yes` for `wazo-confd` endpoint `/asterisk/sip/general`

  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" (`/provd/cfg_mgr/configs`), 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

Co

Table 7 – continued from previous page

	370	710	715	720	D725	D745	760	D76
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
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", 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
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

Table 8 – continued from previous page

	T19P	T19P E2	T20P	T21P	T21P E2	T22P	T26P	T28P	T32G
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. It is important to enable the function keys you want to use. Also, the enable transfer option in the user configuration must be enabled to use transfer function keys.

1.6 Administration

All configurations are done via the `wazo-confd` REST API

1.6.1 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 to 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

Creating a Filter

The filter is used to associate a boss to one or many secretaries and to set a ring strategy.

- Create with `POST /callfilters`

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 with `/extensions/features` endpoint.

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>`.

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`.

1.6.2 Call Permissions

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

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

Denying a user from calling a specific extension

- Create with `POST /callpermissions`
- Associate with `PUT /users/{user_uuid}/callpermissions/{callpermission_id}`

Note: User's `call_permission_password` overwrite all call permissions password 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,

- Create with `POST /callpermissions`

- Associate with `PUT /groups/{group_id}/callpermissions/{callpermission_id}`

Denying users from calling a specific extension on a specific outgoing call

- Create with `POST /callpermissions`
- Associate with `PUT /outcalls/{outcall_id}/callpermissions/{callpermission_id}`

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.

1.6.3 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.

- With `PUT /users/{user_uuid} {"call_record_enabled": True}`

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.

- With `/extensions/features` endpoint and `feature: callrecord`

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.

- With `PUT /users/{user_uuid} {"online_call_record_enabled": False}`

Files

Recorded files are not available for the now with REST API.

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:

- 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_uuid: The tenant UUID of the user or the outgoing call

---

Note: You **must** restart wazo-agid to take any config change into effect:

```
systemctl restart wazo-agid
```

Example 1:

Creating recording in a sub-directory for each entity

A file with the following content in `/etc/wazo-agid/conf.d/call_recording.yml`:

```
call_recording:
  filename_template: "{{ tenant_uuid }}/{{ utc_time }}-{{ srcnum }}-{{ dstnum }}"
```

This configuration would write the files in `/var/spool/asterisk/monitor/<tenant_uuid>/`. 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/wazo-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/wazo-agid/conf.d/recording.yml` with the following content:

```
call-recording:
  filename_extension: wav
```

1.6.4 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 *wazo-purge-db* for more details.

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 `userfield` of a user and the `tags` of a call log. Each `userfield` will be copied into the `tags` for a call log and each `userfield` 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 `userfield` 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.

1.6.5 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 `caller_id` (one for each `extension`)
- 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 `ougoing_caller_id` is Default, use the outgoing call's CallerID

  - If the `ougoing_caller_id` is Anonymous, remove the CallerID
  - If the `ougoing_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 `caller_id` of endpoint of trunk
3. CallerID number normalization
4. The Incoming Call's `caller_id_mode`
5. Reverse lookup

SIP CallerID

To accept the CallerID sent via all SIP trunks, enable the following option

- `PUT /asterisk/sip/general {..., "trustrpid": "yes", ...}`

This option may also be enabled on specific SIP trunks, instead of globally.

Trunk CallerID

The endpoint trunk's `caller_id` 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 *caller_id_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.

This operation is only triggered when the incoming CallerID has no CallerID name or when the CallerID name equals the CallerID number.

1.6.6 CLI Tools

Wazo comes with a collection of console (CLI) tools to help administer the server.

wazo-auth-cli

`wazo-auth-cli` is a command-line interface to interact with the REST API of `wazo-auth`. It provides mainly authentication-related features.

See `wazo-auth-cli --help` for a list of available operations.

wazo-dist-upgrade

`wazo-dist-upgrade` is used to upgrade Wazo when a Debian upgrade is required. `wazo-dist-upgrade` can only be used after `wazo-upgrade`.

wazo-plugind-cli

`wazo-plugind-cli` is a command-line interface to interact with the REST API of `wazo-plugind`. It provides mainly plugin-related features.

See `wazo-plugind-cli --help` for a list of available operations.

wazo-service

`wazo-service` is used to control and print the status of the Wazo services.

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.

wazo-upgrade

`wazo-upgrade` is used to upgrade Wazo to a later version. Beginning with Wazo 17.17, `wazo-upgrade` will not upgrade to a later Debian version. For this, you have to use `wazo-dist-upgrade`.

wazo-agentd-cli

`wazo-agentd-cli` is a command-line interface to interact with the REST API of `wazo-agentd`. It provides mainly agent-related features.

`wazo-agentd-cli` has an interactive REPL mode. You can access it with the command `wazo-agentd-cli`. It should prompt you for a password that is empty by default. Once in the interactive mode, enter `help` for a list of available operations.

wazo-dist

wazo-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 : `wazo-dist -m pelican-buster`
- switch to development repository : `wazo-dist -m wazo-dev-buster`
- switch to release candidate repository : `wazo-dist -m wazo-rc-buster`
- switch to an archived version's repository: `wazo-dist -a wazo-18.02`

wazo-provd-cli

`wazo-provd-cli` is a command-line interface to interact with the REST API of `wazo-provd`. It provides mainly provisioning-related features.

`wazo-provd-cli` has an interactive REPL mode. You can access it with the command `wazo-provd-cli`. It should prompt you for a password that is empty by default. Once in the interactive mode, enter `help` for a list of available operations.

1.6.7 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.

This feature is disabled by default. The reason behind this choice is based on the fact that it is possible for any user to pickup any other user, including users for whom it should not be possible (e.g. directors). It can be enabled via the `/extensions/features` endpoint.

The extension prefix used for directed pickup can be changed via the `/extensions/features` endpoint.

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.

1.6.8 Fax

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 `fax_to_mail` 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.

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 wazo-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 wazo-agid restart
```

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 wazo-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 wazo-agid restart
```

Using the advanced features

The following features are only available via the `/etc/xivo/asterisk/xivo_fax.conf` configuration file.

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
```

(continues on next page)

(continued from previous page)

```
[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 wazo-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/wazo-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 }}_>
```

(continues on next page)

(continued from previous page)

```
<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:

```
wazo-provd-cli -c 'devices.using_plugin("xivo-cisco-spa3102-5.1.10").reconfigure()
↪'
```

4. Then reboot the devices:

```
wazo-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.

1.6.9 Graphics

There are graphics, locate to `/var/cache/munin/www/localdomain/localhost.localdomain/`, that give a historical overview of a Wazo system's activity based on snapshots recorded every 5 minutes. Graphics are available for the following resources :

- `cpu-*.png`
- `entropy-*.png`
- `interrupts-*.png`
- `irqstats-*.png`
- `load-*.png`
- `memory-*.png`
- `open_files-*.png`
- `open_inodes-*.png`
- `swap-*.png`

Each graphic is available with different time range: `day`, `week`, `month`, `year`

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.

1.6.10 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

- `POST /callpickups`
- `POST /callpickups/{callpickup_id}/interceptors/groups`
- `POST /callpickups/{callpickup_id}/interceptors/users`
- `POST /callpickups/{callpickup_id}/targets/groups`
- `POST /callpickups/{callpickup_id}/targets/users`

Enabling an Interception Extension

The pickup extension can be defined with:

- `/asterisk/features/general` endpoint and the option key `pickupexten`

The default value for group pickup is **8*.

> **Warning:** The directed pickup extension must be enabled when a function key is used.

If you decide to not use a directed pickup extension, only **8* alone will work (without specifying the extension to pickup). In this case, you *must* have at least a pickup group with the targets and interceptors you want for the interception to work. This will also make it impossible for the function keys to work. See *Directed Pickup* for more information.

Adding a Function Key to an Interceptor

Assign a function to an interceptor of type `pickup`

1.6.11 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 :

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:

- 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

Follow 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

Follow 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 : https://support.digium.com/community/s/search/All/Home/IRQ

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).

1.6.12 Incall

General Configuration

You can configure incoming calls with `/incalls` endpoints.

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.

BlackList

There are no interface to set a blacklist, but you can build if by hand.

- You need a preprocess subroutine on the incall with the following dialplan:

```
[check-blacklist]
exten = s,1,GotoIf(${BLACKLIST()}?blacklisted)
same = n,Return()
same = n(blacklisted),Playback(no-user-find)
same = n,Hangup()
```

- Do a `dialplan reload` in the Asterisk CLI to load the new dialplan

You can manage the blacklist in the Asterisk CLI

- To add extension:

```
*CLI> database put blacklist <extension> "<description (e.g. reason)>"
```

- To remove extension:

```
*CLI> database del blacklist <extension>
```

1.6.13 Interconnections

Interconnect two Wazo directly

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.

- `POST /trunks {"context": <see below>}`
- `POST /endpoints/sip {"username": "wazo-trunk", "secret": "pass", "type": "friend", "host": "dynamic"}`
- `PUT /trunks/{trunk_id}/endpoints/sip/{sip_id}`

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 `from-extern`. 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 `from-extern` context to other contexts.

On Wazo A, create the other end of the SIP trunk:

- `POST /trunks {"context": "from-extern"}`
- `POST /endpoints/sip {"username": "wazo-trunk", "secret": "pass", "type": "friend", "host": <Wazo B IP address or hostname>}`
- `PUT /trunks/{trunk_id}/endpoints/sip/{sip_id}`
- `POST /registers/sip {"auth_username": "wazo-trunk", "auth_password": "pass", "transport": "udp", "remote_host": <Wazo B IP address or hostname>}`
- `PUT /trunks/{trunk_id}/registers/sip/{sip_id}`

On both Wazo, activate some codecs:

- `PUT /asterisk/sip/general {..., "allow": "gsm", ...}`

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), add outgoing call.

- `POST /outcalls`
- `PUT /outcalls/{outcall_id}/trunks`
- `POST /extensions {"exten": "**99.", "context": "to-extern"}`
- `PUT /outcalls/{outcall_id}/extensions/{extension_id} {"strip_digits": 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

- `POST /extensions {"exten": "101", "context": "from-extern"}`
- `POST /incalls {"destination": {"type": "user", "user_id": <someone_id>}}`
- `PUT /incalls/{incall_id}/extensions/{extension_id}`

This will tell Wazo: if you receive an incoming call to the extension `101` in the context `from-extern`, then route it to the user `someone_id`. 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_id` 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

- `POST /trunks {"context": "from-extern"}` (or another incoming call context)
- `POST /endpoints/sip {"username": <username>, "secret": <password>, "type": "peer", "host": "voip.provider.example.com"}`
- `PUT /trunks/{trunk_id}/endpoints/sip/{sip_id}`
- `POST /registers/sip {"auth_username": <username>, "auth_password": <password>, "transport": "udp", "remote_host": "voip.provider.example.com"}`
- `PUT /trunks/{trunk_id}/registers/sip/{sip_id}`

If your Wazo is behind a NAT device or a firewall, you should set the following:

```
* ``PUT /endpoints/sip {"options": [..., ["qualify", "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:

```
* ``PUT /endpoints/sip {"options": [..., ["qualifyfreq", <value>], ...]}``
```

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.

- `POST /outcalls`
- `PUT /outcalls/{outcall_id}/trunks`
- `POST /extensions {"exten": "418.", "context": "to-extern"}`
- `PUT /outcalls/{outcall_id}/extensions/{extension_id}`

This will tell Wazo: if an internal user dials a number beginning with `418`, then try to dial it on the trunk associated.

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.

- `POST /extensions {"exten": <public_phone_number>, "context": "from-extern"}`
- `POST /incalls {"destination": {"type": "user", "user_id": <the_front_desk_guy_id>}}`
- `PUT /incalls/{incall_id}/extensions/{extension_id}`

This will tell Wazo: if you receive an incoming call to the public phone number in the context `from_extern`, then route it to the user `the_front_desk_guy_id`. 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**.

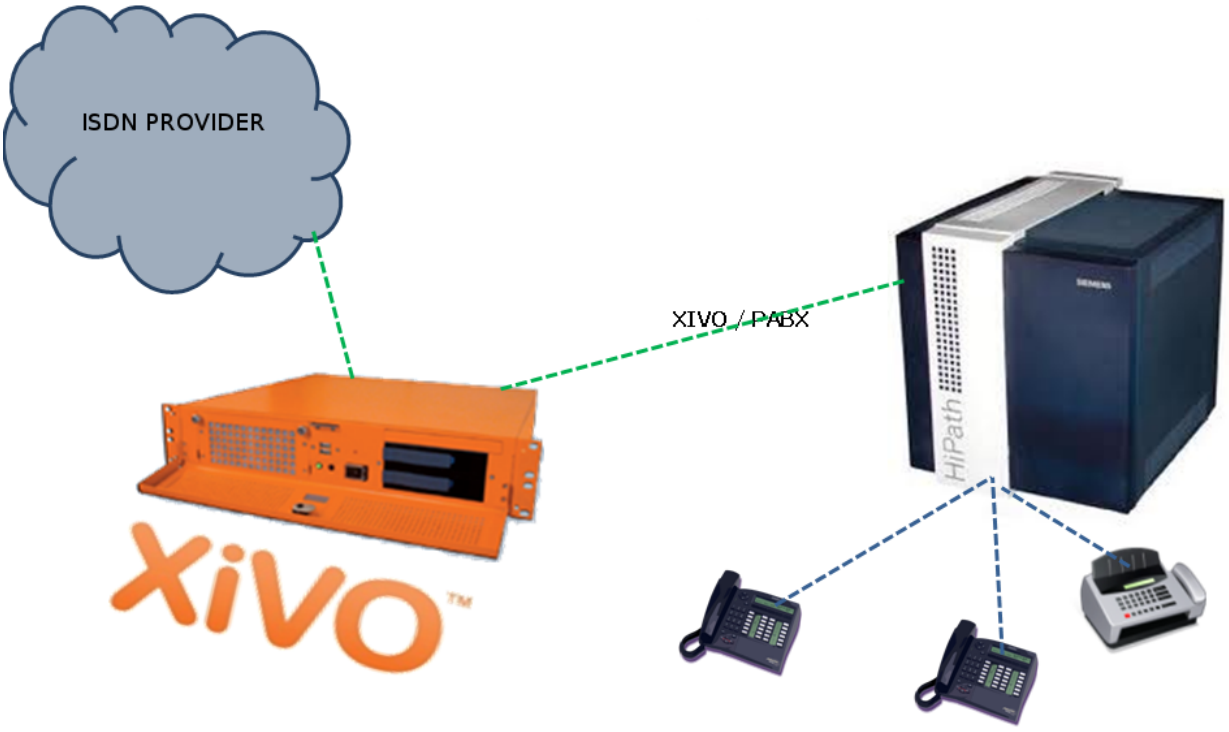


Fig. 5: Interconnect a Wazo to a PBX

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.

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"
```

(continues on next page)

(continued from previous page)

```
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

Create a context named `to-pabx`:

- `POST /contexts {"name": "to-pabx", "type": "outcall"}`

Route incoming calls to PBX

In our example, incoming calls on spans 1 and 2 (spans plugged to the provider) are routed by from-extern context. We are going to create a default route to redirect incoming calls to the PBX.

- `POST /extensions {"exten": "_XXXX", "context": "from-extern"}` (according to the number of digits sent by the provider)
- `POST /incalls {"destination": {"type": "customiz", "command": "Goto(to-pabx,${XIVO_DSTNUM},1)}}`
- `PUT /incalls/{incall_id}/extensions/{extension_id}`

Create the interconnections

You have to create two interconnections :

- provider side : dahdi/g0
- PBX side : dahdi/g2

The first interconnection :

- `POST /trunks {'name': "t2-operatoeur", "context": "to-extern"}`
- `POST /endpoints/custom {"interface": "dahdi/g0"}`
- `PUT /trunks/{trunk_id}/endpoints/custom/{custom_id}`

The second interconnection :

- `POST /trunks {"name": "t2-pabx", "context": "to-pabx"}`
- `POST /endpoints/custom {"interface": "dahdi/g2"}`

- `PUT /trunks/{trunk_id}/endpoints/custom/{custom_id}`

Create outgoing calls

You must create two rules of outgoing calls:

1. Redirect calls to the PBX :

- `POST /outcalls {"name": "fsc-pabx"}`
- `PUT /outcalls/{outcall_id}/trunks`
- `POST /extensions {"exten": "_XXXX", "context": "to-pabx"}`
- `PUT /outcalls/{outcall_id}/extensions/{extension_id}`

2. Create a rule “fsc-operateur”:

- `POST /outcalls {"name": "fsc-operateur"}`
- `PUT /outcalls/{outcall_id}/trunks`
- `POST /extensions {"exten": "_X.", "context": "to-extern"}`
- `PUT /outcalls/{outcall_id}/extensions/{extension_id}`

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

- `PUT /asterisk/sip/general {..., "match_auth_username": "yes", ...}`

Trunk settings

- `POST /trunks {"context": "from-extern"}`
- `POST /endpoints/sip {"username": "GV18005551212", "secret": "password", "type": "friend", "host": "gvgw.simonics.com", options=[["qualify", "yes"], ["callerid", "18005551212"]]`
- `PUT /trunks/{trunk_id}/endpoints/sip/{sip_id}`
- `POST /registers/sip {"auth_username": "GV18005551212", "auth_password": "password", "transport": "udp", "remote_host": "GV18005551212", "callback_extension": "18005551212"}`
- `PUT /trunks/{trunk_id}/registers/sip/{sip_id}`

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 with `/asterisk/sip/general` endpoint and trunk configurations are available with `/endpoints/sip` and `/trunks` endpoints

Environment with NAT

There are some configuration steps that are required when connecting to a SIP provider from a NAT environment.

- `PUT /asterisk/sip/general {..., "externip": "69.70.94.94", "localnet": "192.168.0.0/16", ...}`
- `externip`: This is your public IP address
- `localnet`: Your internal network range
- `PUT /endpoints/sip/{endpoint_sip_id} {"options": [["nat", "yes"], ["qualify", "yes"]]}`

Warning: When changing the *externip*, the *media_address* or the *externhost* Asterisk has to be restarted using the *wazo-service restart* command for the changes to take effect.

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

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.

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 `internal_caller_id`. 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.

```
PUT /outcalls/{outcall_id}/extensions/{extension_id} {"caller_id":  "\"XIVO\"
<555>"}
```

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 by user. The `outgoing_caller_id` option must be set to the caller ID. The user can also set his `outgoing_caller_id` to `anonymous`.

```
PUT /users/{user_uuid} {"outgoing_caller_id":  "\"Bob\" <555>"}
```

The order of precedence when setting the caller ID in multiple place is the following.

1. `internal_caller_id`
2. User's `outgoing_caller_id`
3. Outgoing call
4. Default caller ID

1.6.14 Interactive Voice Response

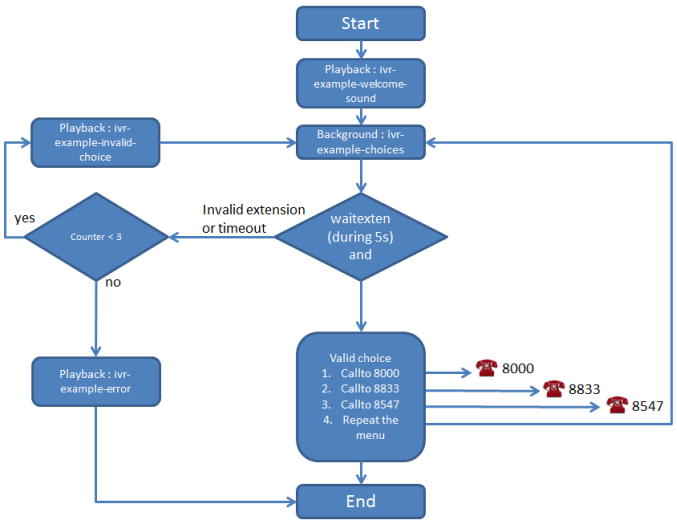
Introduction

> *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 *wazo-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 `/etc/asterisk/extensions_extra.d/dp-ivr-example.conf`.

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/wazo/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)
```

(continues on next page)

(continued from previous page)

```
;##### 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 :

- `POST /extensions {"exten": <DID>, "context": "from-extern"}`
- `POST /incalls {"destination": {"type": "custom", "command": "Goto(dp-ivr-example,s,1)"}}`
- `PUT /incalls/{incall_id}/extensions/{extension_id}`

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 `/etc/asterisk/extensions_extra.d/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

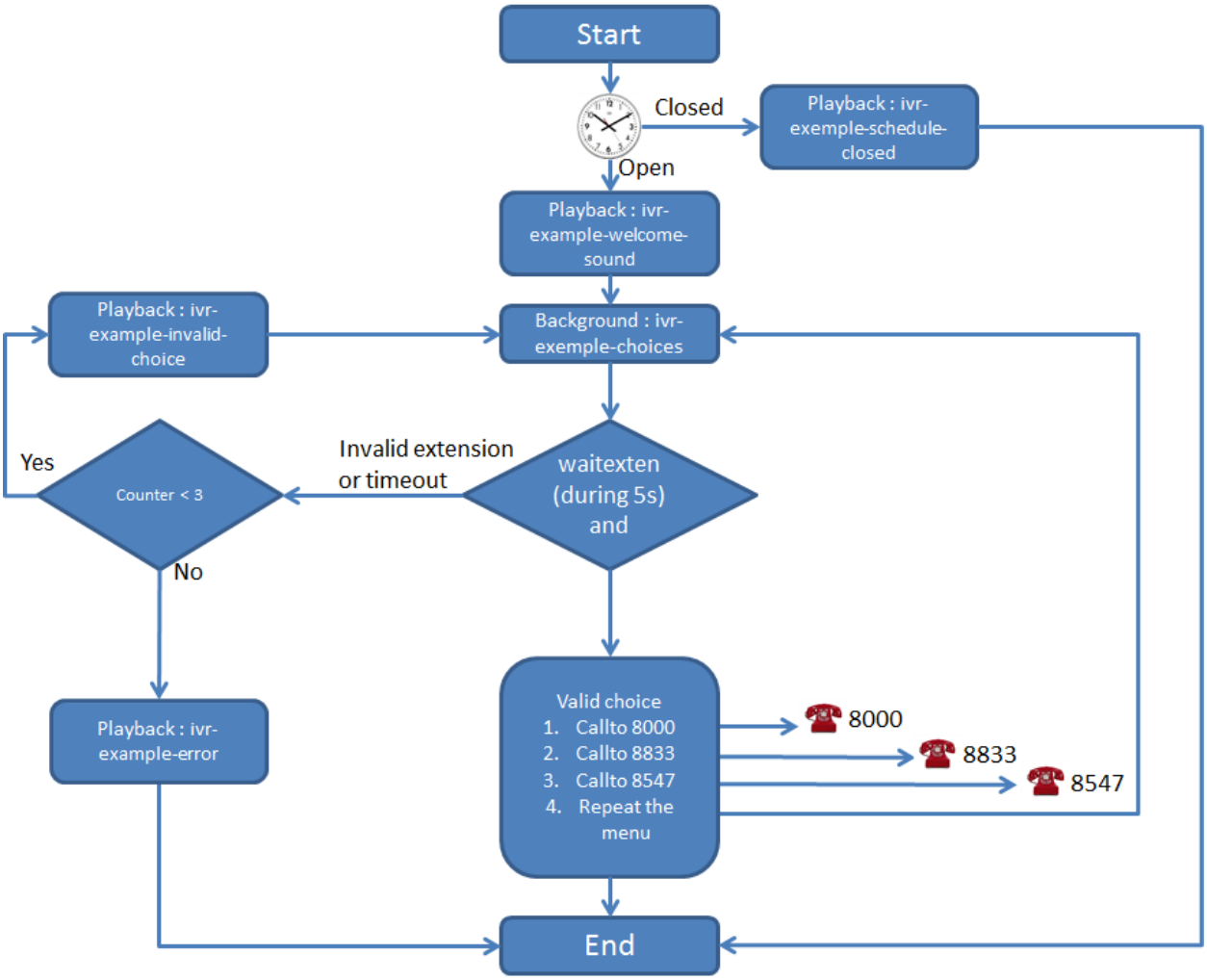
Create Schedule

First step, create your schedule. Give a name to your schedule and configure the open hours and select the sound which is played when the company is closed.

In the Closed hours tab, configure all special closed days and select the sound that indicate to the caller that the company is exceptionally closed.

The IVR script is now only available during workdays.

- `POST /schedules`
- `PUT /incalls/{incall_id}/schedules/{schedule_id}`


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

Use Case: IVR with submenu

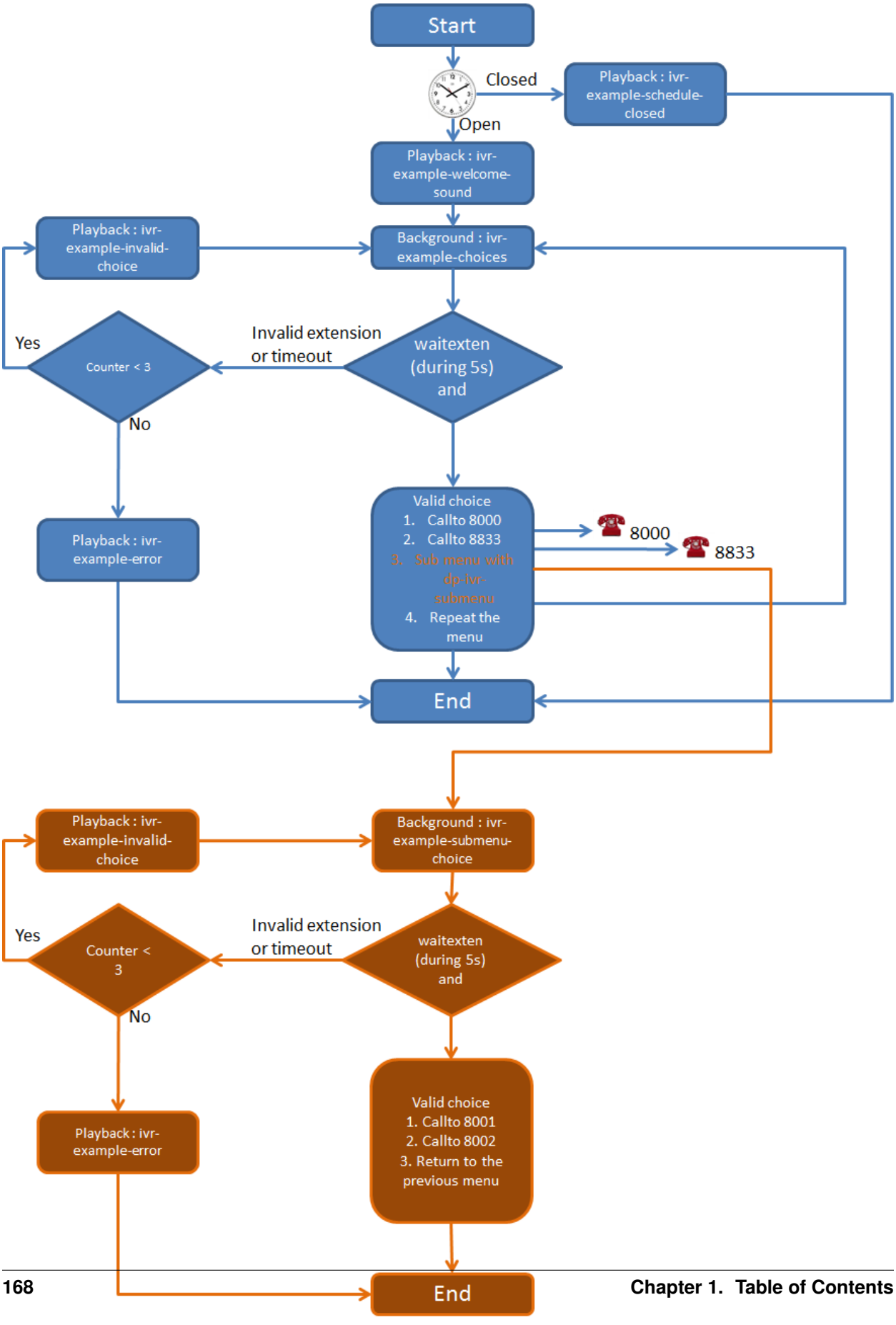
Flowchart

Configuration File and Dialplan

Copy all these lines (2 contexts) in a configuration file on your Wazo server :

```
[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/wazo/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)
```

(continues on next page)


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

(continued from previous page)

```
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)
```

(continues on next page)

(continued from previous page)

```
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()
```

1.6.15 Music on Hold

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.

1.6.16 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 with `/pagings` endpoints

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`.

1.6.17 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.

You can manage parking with `/parkinglots` endpoints

Using this extension, you may define the parking number used to park call, the parking lots, whether the system 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 :

- 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

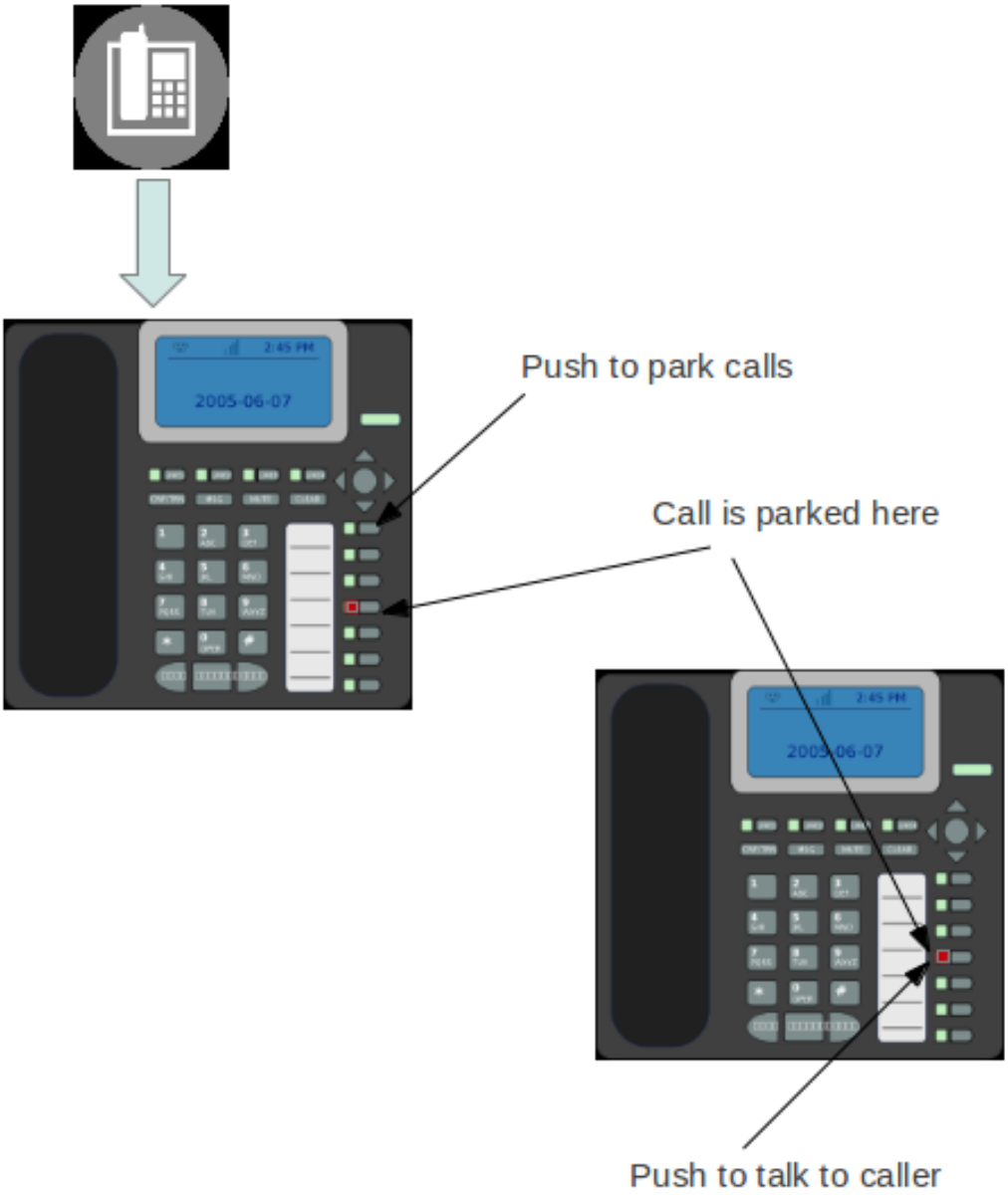
  In case you don't want to call back the user who parked the call, you have the option to send the call to any other extension or application. If the parking times out, the call is sent back to the dialplan in context `[parkedcallstimeout]`. You can define this context in a dialplan configuration file located to `/etc/asterisk/extensions_extra.d/`

  Example:

```
[parkedcallstimeout]
exten = s,1,Noop('park call time out')
same  =   n,Playback(hello-world)
same  =   n,Hangup()
```

1.6.18 Provisioning

Wazo supports the auto-provisioning of a large number of telephony *Devices*, including SIP phones, SIP ATAs, and even softphones.


Push to park calls
Call is parked here
Push to talk to caller

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.

Tenant assignation

On initial insertion into provd, devices are assigned to the tenant of the token used internally by provd, which is the master tenant. When a device is provisioned, it is transferred to the tenant of the line to which it is being associated. When the device is reset to autoprov, the device stays in its tenant. It is not possible to change the tenant of the device once it is set. If you wish to do it anyway, you must delete the device and restart it manually.

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.
- There may be an issue if you are using an analog gateway with lines that are not in the same tenant. Indeed, in the case that the gateway is only one device and each port is a separate line, the device will only be seen by the tenant of the first line that was added.

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 `PUT /dhcp`

Installing `provd` Plugins

The installation and management of `provd` plugins is done via `wazo-provd` endpoint `/provd/pg_mgr/install`

> **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.

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 with `wazo-provd` endpoint `/provd/configure/plugin_server`

- `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 development.

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 with `/provd/pg_mgr/install/update`

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:

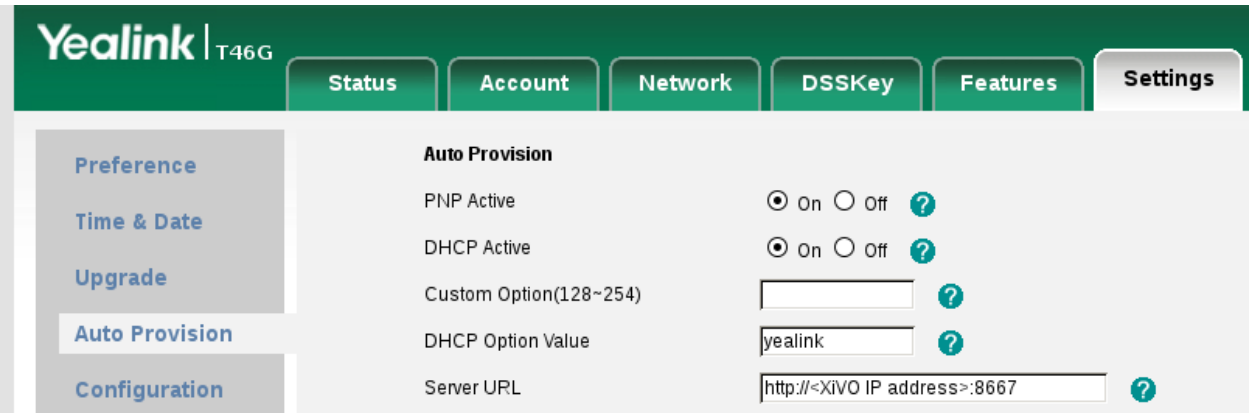
```
sed -i 's/dhcp:stop/dhcp:proceed/' /var/lib/wazo-provd/plugins/xivo-snom-8.7.5.35/var/
↪tftpboot/snom-general.xml
```

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, `GET /devices`. 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 `GET /devices`, you should see that your Wazo now knows about your 6731i with `status: not_configured`

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 `GET /devices`, you'll see the device with with `status: configured`

Resetting a Device

From REST API

To remove a phone from Wazo or enable a device to be used for another user: possibilities :

- `GET /devices/{device_id}/autoprov`
- `GET /devices/{device_id}/synchronize`

The phone will restarts and display autoprov, ready to be used for another user.

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

DHCP integration is enabled by default without possibility to disable it.

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/wazo-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
wazo-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:

```
wazo-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
wazo-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
wazo-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 with `PUT /devices/{device_id}`.

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:

```
wazo-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:

```
wazo-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 `GET /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 `yes` with `PUT /asterisk/sip/general`.

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 REST API `PUT /lines/{line_id}/devices/{device_id}`. 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/wazo-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/wazo.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 wazo-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 Wazo directory and place call directly from your phone.

See the list of *supported devices* to know if a model supports the Wazo directory or not.

Configuration

For the remote directory to work on your phones, the first thing to do is to create a custom wazo-phoned configuration file `/etc/wazo-phoned/conf.d/custom.yml`

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, add:

```
rest_api:
  authorized_subnets: [192.168.1.0/24]
```

You must then restart `wazo-phoned`:

```
systemctl restart wazo-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 Wazo 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

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

1.6.19 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-wazo

The `asterisk-wazo` 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.

wazo-provd

The `wazo-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, use `wazo-provd` endpoint `/provd/cfg_mgr/configs`.

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.

1.6.20 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

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 with `GET /devices`.

When connecting a second SCCP phone, the device will be automatically detected as well.

Auto-provisioning support

Starting from Wazo 18.07, an SCCP device can be associated to a user by entering the user's provisioning code directly from the SCCP device while in autoprov mode.

There's two settings in `GET /asterisk/sccp/general` influencing the auto-provisioning behaviour:

- the `guest` option must be enabled to allow SCCP devices to connect to the server and allow a provisioning code from being dialed from them. Disabling this option can provide some additional security if your Wazo is in an hostile environment, at the cost of making auto-provisioning support unavailable for SCCP devices.
- the `max_guests` option limits the number of SCCP devices that can simultaneously connect to the server in autoprov mode. You should set this value to the maximum number of SCCP devices you expect to be in autoprov mode at any moment, unless your Wazo is in an hostile environment, where you should probably set it to a fairly low value.

SCCP General Settings

Review SCCP general settings:

```
GET /asterisk/sccp/general
```

User creation

The last step is to create a user with a **SCCP line**.

Creating a user with a SCCP line:

- `POST /users`
- `POST /lines`
- `PUT /users/{user_id}/lines/{line_id}`
- `POST /endpoints/sccp`
- `PUT /lines/{line_id}/endpoints/sccp/{sccp_id}`
- `PUT /lines/{line_id}/devices/{device_id}`

Congratulations ! Your SCCP phone is now ready to be called !

Function keys

With SCCP phones, the only destination type of function keys that can be configured is `custom`

Direct Media

SCCP Phones support directmedia (direct RTP).

- `PUT /asterisk/sccp/general` options `directmedia: yes`

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	Yes
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		
8941	Testing		
8945	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.

1.6.21 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 exceptional 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

Using Schedule on Users

When you have a schedule associated to a user, if this user is called during a `exception_periods`, the caller will first hear a prompt saying the call is being transferred before being actually redirected to the 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"`.

1.6.22 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 wazo-sounds-fr-ca
```

To install German sounds you have to execute the following command:

```
apt-get install asterisk-sounds-wav-de-de wazo-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
```

1.6.23 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 or tenant 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
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
username	string			User's username to log into applications
password	string			User's password to log into applications

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 `POST /users/import` to create all users in the specified tenant using the *Wazo-Tenant* header.

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:

```
firstname,lastname,exten,context,line_protocol
John,Doe,1000,default,sip
George,Clinton,1001,default,sip
Bill,Bush,1002,default,sccp
```

The following example imports a user with a phone number and a voicemail:

```
firstname,lastname,exten,context,line_protocol,voicemail_name,voicemail_number,
↪voicemail_context
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):

```
firstname,lastname,exten,context,line_protocol,incall_exten,incall_context
John,Doe,1000,default,sip,2050,from-extern
```

CSV Update

Note: The CSV update has been disabled since it does not suport multi-tenants at the moment

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 `PUT /users/import` endpoint

CSV Export

CSV exports can be used as a scaffold for updating users, or as a means of importing users into another system or tenant. An export will generate a CSV file with the same list of columns as an import, with the addition of uuid and provisioning_code, for all users in the specified tenant.

Exports are done through the `GET /users/export`

Function keys

Function keys can be configured to customize the user's phone keys. The `blf` 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".

If the forward 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* forward for this user.

Extensions

**3* (online call recording)

To enable online call recording, you must set `automixmon`:

```
PUT /asterisk/features/featuremap {"options": {"automixmon": "*3",
...}}
```

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 `/var/spool/asterisk/monitor` directory.

**26* (call recording)

You can enable/disable the recording of all calls for a user in 2 different way:

1. By set `call_record_enabled: True` for user:

   ```
   PUT /users/{user_uuid} {"call_record_enabled": True}
   ```

2. By using the extension **26* from your phone (the feature `callrecord` option must be enabled):

   ```
   PUT /extensions/features/{extension_id}
   ```

When this option is activated, all calls made to or made by the user will be recorded in the `/var/spool/asterisk/monitor` directory.

1.6.24 Voicemails

Voicemail Configuration.

General Configuration

The global voicemail configuration is provided by `/asterisk/voicemail` endpoints

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.

Deleting a voicemail

- 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.

Disable password checking

Unchecking the option `ask_password` field 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.

Warning: For security reasons, an incoming call with `{"destination": {"appplication": "voicemail"}` with the same context as the voicemail should be avoided if a voicemail has no password.

Advanced configuration

Remote *wazo-confd*

If *wazo-confd* is on a remote host, *wazo-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>
```

(continues on next page)

(continued from previous page)

```
EOF
systemctl daemon-reload
```

1.7 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

1.7.1 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.

1.7.2 Queues

Call queues are used to distribute calls to the agents subscribed to the queue. Queues are managed with the `/queues` endpoints

A queue can be configured with the following options:

A `options: 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
- `leastrecent`: call the member who least recently hung up a call
- `fewestcalls`: call the member with the fewest completed calls
- `rrmemory` (round robin with memory): call the "next" member after the one who answered last
- `random`: call a member at random
- `wrandom` (weight random): same as random, but taking the member penalty into account
- `ringall`: 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

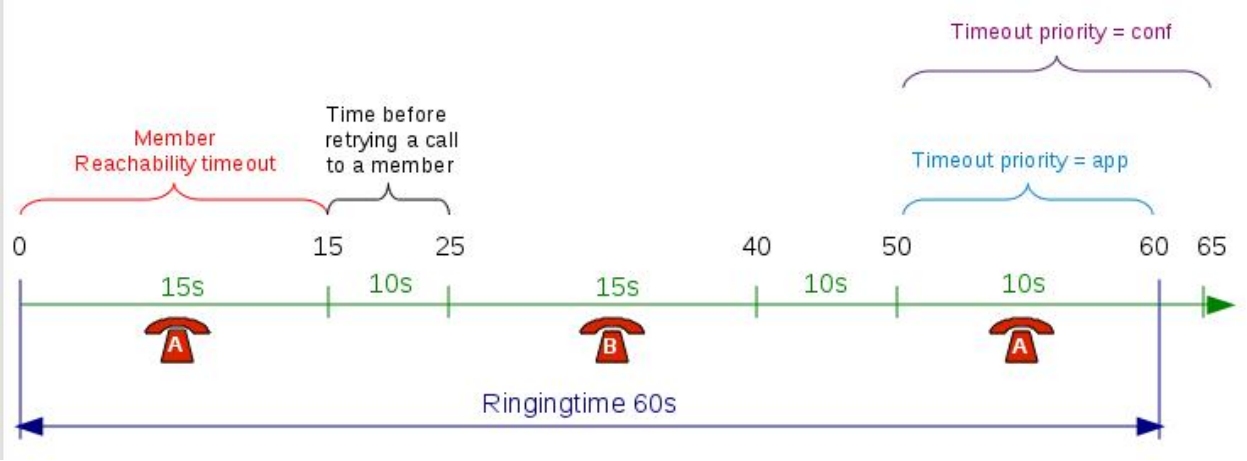
You may control how long a call will stay in a queue using different timers:

- `options: timeout` (Member reachabillity time out): 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.
- `retry_on_timeout` (Time before retrying a call to a member): Used once a call has reached the "Member reachability time out". The call will be put on hold for the number of seconds allowed before being redirected to another agent.
- `timeout` (Ringing time): The total time the call will stay in the queue.
- `options: timeoutpriority` (Timeout priority): 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".

Fallbacks

Calls can be diverted on no answer with `/queues/{queue_id}/fallbacks` endpoints:

- `noanswer_destination`: The call reached the `timeout` and no agent answered the call.
- `congestion_destination`: The number of calls waiting has reached the `options: maxlen`.
- `fail_destination`: No agent was available to answer the call when the call entered the queue (`options: joinempty`) or the call was queued and no agents were available to answer (`options: leavewhenempty`).



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 threshold: `wait_ratio_threshold` and `ait_time_threshold`

The diversion check is done only once per call, before the *preprocess subroutine* is executed and before the call enters the queue.

`wait_time_threshold`

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 `wait_time_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.

`wait_ratio_threshold`

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 `wait_ratio_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:

```
wait_ratio_threshold: 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 to ``wait_ratio_destination``

wait_ratio_threshold: 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 to ``wait_ratio_destination``
```

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:

```
wait_ratio_threshold: 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 `wait_ratio_time` (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

1.7.3 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.

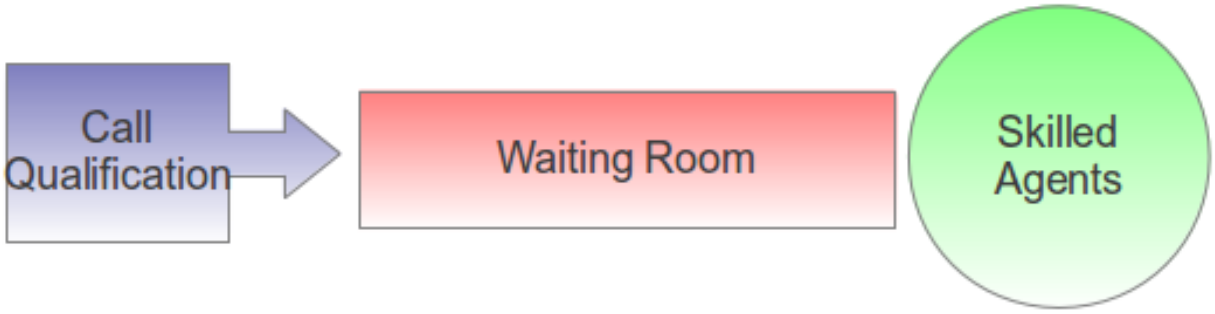


Fig. 6: Skills-Based Routing

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

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:

- `POST /agents/skills`

Once all the skills are created you may apply them to agents. Agents may have one or more skills.

- `PUT /agents/{agent_id}/skills/{skill_id} {"skill_weight": 55}`

It is typical to use a value between 0 and 100 inclusively as the `skill_weight`, although any integer is accepted.

Skill Rule Sets

Once skills are created, rule sets can be defined.

- `POST /queues/skillrules`

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 subtleties 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)

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 an incoming call.

- `POST incalls {"destination": {"type": "queue", "skill_rule_id": <id>, "skill_rule_variables": {"lang": "english"}}}`
- `POST incalls {"destination": {"type": "queue", "skill_rule_id": <id>, "skill_rule_variables": {"lang": "french"}}}`

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
```

1.7.4 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*

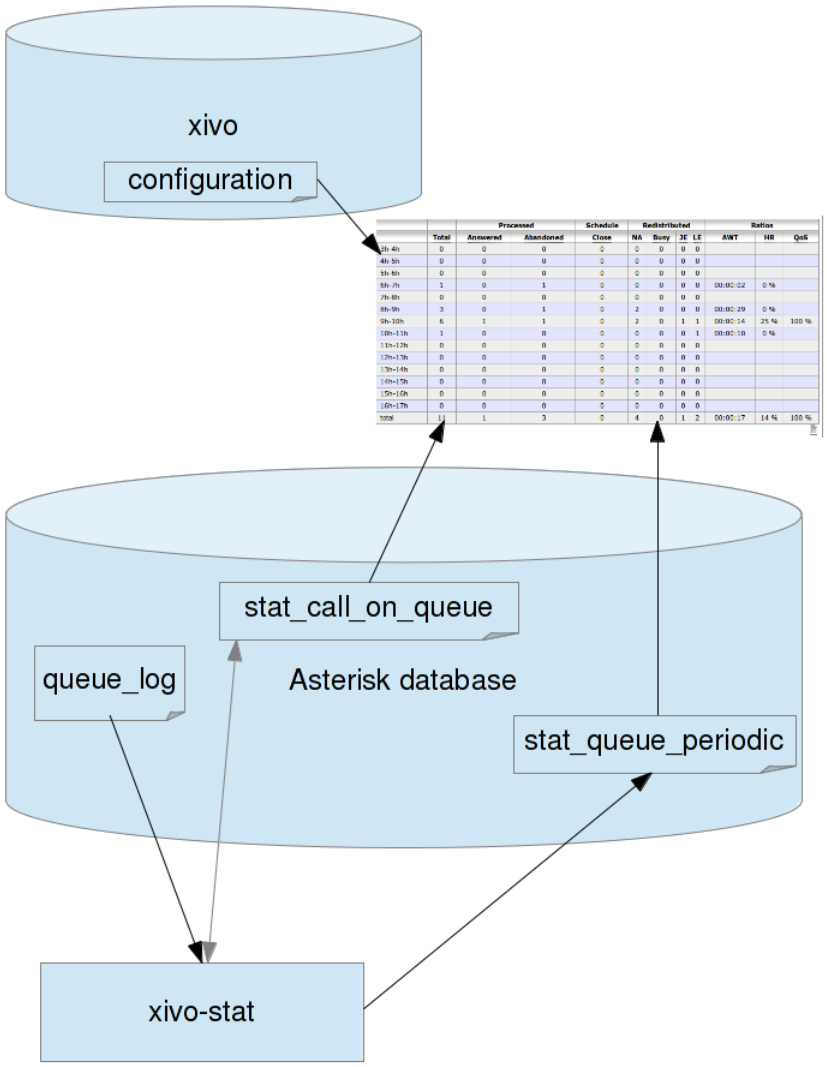


Fig. 7: Statistics Architecture

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.

1.8 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.

1.8.1 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*.

1.8.2 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 running 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

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.

1.8.3 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 with `PUT /ha`

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/wazo/sounds/tenants

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:

- `GET /asterisk/sip/general`
  - `minexpiry`: 3 minutes
  - `maxexpiry`: 5 minutes
  - `defaultexpiry`: 4 minutes

The provisioning server configuration will be automatically updated in order to allow phones to switch from Wazo power failure.

- `GET /provd/cfg_mgr/configs?q={"X_type": "registrar"}`

  `registrar_backup`: <slave ip> `proxy_backup`: <slave ip>

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.

Important: For the telephony devices to take the new proxy/registrar settings into account, you must resynchronize the devices or restart them manually.

Master node

In choosing the `node_type: master` you must enter the `remote_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 `node_type: slave` you must enter the `remote_address` **of the VoIP interface** of the master node.

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
- All the support configuration
- Call logs
- Call center statistics
- Certificates
- HA settings
- Provisioning configuration
- Voicemail messages

Less importantly, these are also excluded:

- Queue logs
- CELs

1.8.4 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.

1.8.5 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.

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.

1.8.6 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:

```
ISDN lines (provider)
  | | | |
  | | | |
+-----------------------------------------+
|    A           B           C           D    |
| 1|2|3|4     1|2|3|4     1|2|3|4     1|2|3|4 |
+-----------------------------------------+
               | | | |                 | | | |
               | | | |                 | | | |
            +--------+              +--------+
            | wazo-1 |              | wazo-2 |
            +--------+              +--------+
```

Two Wazo and one PBX

Here's how to connect your berofos with:

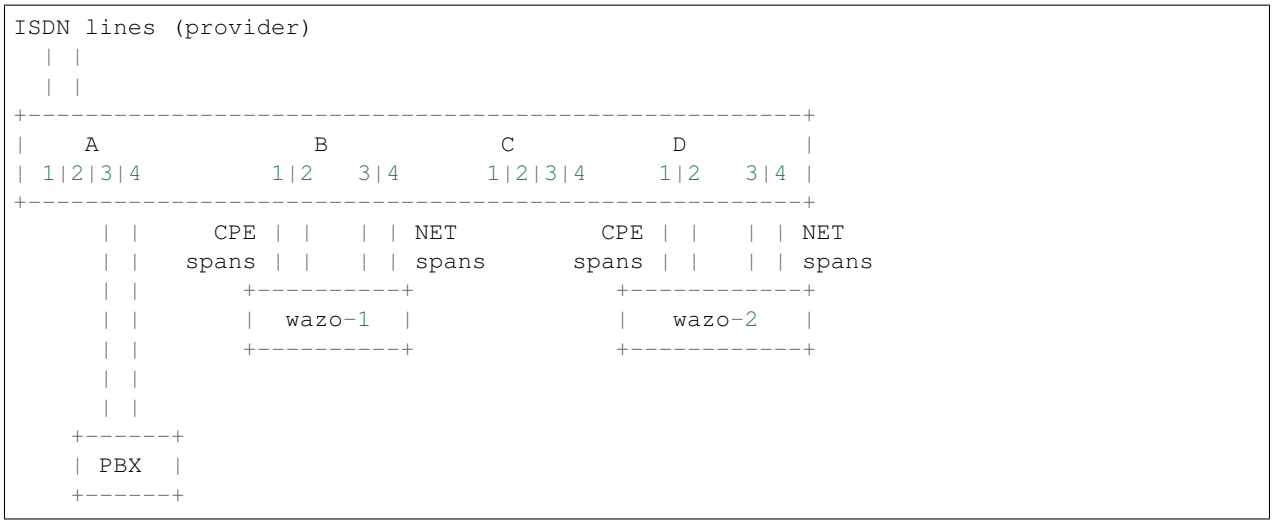
- two Wazo in high availability,
- one PBX.

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:

```
                +--------+                              +-----+
-- Provider ----| wazo-1 | -- ISDN Interconnection  --| PBX | -- Phones
                +--------+                              +-----+
                  | wazo-2 |
                  +--------+
```

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

1.8.7 Troubleshooting

When replicating the database between master and slave, if you encounter problems related to the system locale, see *PostgreSQL localization errors*.

1.9 API and SDK

1.9.1 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 *call_created, call_updated, call_ended* events:

```
import kombu

from kombu.mixins import ConsumerMixin

EXCHANGE = kombu.Exchange('xivo', type='topic')
ROUTING_KEY = 'events.calls.*'


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

19.05

- The following messages have been deleted:
  - chat_message_event
  - chat_message_received
  - chat_message_sent
  - endpoint_status_update
  - user_status_update

19.04

- The following messages have been added:
  - *fax_outbound_created*
  - *fax_outbound_user_created*
  - *fax_outbound_succeeded*
  - *fax_outbound_user_succeeded*
  - *fax_outbound_failed*
  - *fax_outbound_user_failed*

19.03

- The following messages have been added:
  - *conference_record_started*
  - *conference_record_stopped*
  - *conference_participant_talk_started*
  - *conference_participant_talk_stopped*

19.02

- The following messages have been added:
  - *conference_participant_joined*
  - *conference_participant_left*
  - *conference_participant_muted*
  - *conference_participant_unmuted*

18.04

- The following messages have been added:
  - *auth_tenant_created*

  - *auth_tenant_deleted*
  - *auth_tenant_updated*

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 wazo-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"
  }
}
```

auth_tenant_created

This event is published when a tenant is created

- routing_key: auth.tenants.{tenant_uuid}.created
- event specific data:
  - uuid: The tenant's UUID
  - name: The name of the tenant

Example:

```
{
  "name": "auth_tenant_created",
  "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
  "data": {
    "uuid": "a1e05585-1421-4397-bd59-9cf9725888e9",
    "name": "<name>"
  }
}
```

auth_tenant_deleted

This event is published when a tenant is deleted

- routing_key: auth.tenants.{tenant_uuid}.deleted
- event specific data:
  - uuid: The tenant's UUID

Example:

```
{
  "name": "auth_tenant_deleted",
  "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
  "data": {
    "uuid": "a1e05585-1421-4397-bd59-9cf9725888e9",
  }
}
```

auth_tenant_updated

This event is published when a tenant is updated

- routing_key: auth.tenants.{tenant_uuid}.updated
- event specific data:
  - uuid: The tenant's UUID
  - name: The name of the tenant

Example:

```
{
  "name": "auth_tenant_updated",
  "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
  "data": {
    "uuid": "a1e05585-1421-4397-bd59-9cf9725888e9",
    "name": "<name>"
  }
}
```

call_form_result

The call_form_result event is sent when a custom call form is submitted via REST API.

- 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 wazo-calld 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 wazo-calld)

Example:

```
{
    "name": "call_created",
```

(continues on next page)

(continued from previous page)


```
    "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"}}
```


conference_participant_joined, conference_participant_left

Those events are send when a participant joins or leaves a conference room.

- routing keys:
  - `conferences.<conference_id>.participants.joined`
  - `conferences.<conference_id>.participants.left`

- required ACLs:
  - `events.conferences.<conference_id>.participants.joined`
  - `events.conferences.<conference_id>.participants.left`
- event specific data:
  - `id`: The ID of the participant inside the conference
  - `caller_id_name`: The CallerID name of the participant
  - `caller_id_num`: The CallerID number of the participant
  - `muted`: Is the participant muted?
  - `answered_time`: Elapsed seconds since the participant joined the conference
  - `admin`: Is the participant and admin of the conference?
  - `language`: The language of the participant
  - `call_id`: The ID of the call, usable in the `/calls` endpoints of `wazo-calld`
  - `conference_id`: The ID of the conference

Example:

```
{
    "name": "conference_participant_joined",
    "origin_uuid": "08c56466-8f29-45c7-9856-92bf1ba89b82",
    "required_acl": "events.conferences.1.participants.joined",
    "data": {
        "admin": false,
        "answered_time": 0,
        "call_id": "1547576420.11",
        "caller_id_name": "Bernard Marx",
        "conference_id": 1,
        "id": "1547576420.11",
        "language": "fr_FR",
        "muted": false
    }
}
```

conference_participant_muted, conference_participant_unmuted

Those events are send when a participant joins or leaves a conference room.

- routing key for both events:
  - `conferences.<conference_id>.participants.mute`
- required ACL for both events:
  - `events.conferences.<conference_id>.participants.mute`
- event specific data:
  - `id`: The ID of the participant inside the conference
  - `caller_id_name`: The CallerID name of the participant
  - `caller_id_num`: The CallerID number of the participant
  - `muted`: Is the participant muted?

  - admin: Is the participant and admin of the conference?
  - language: The language of the participant
  - call_id: The ID of the call, usable in the /calls endpoints of wazo-calld
  - conference_id: The ID of the conference

Example:


```
{
    "name": "conference_participant_muted",
    "origin_uuid": "08c56466-8f29-45c7-9856-92bf1ba89b82",
    "required_acl": "events.conferences.1.participants.mute",
    "data": {
        "admin": false,
        "call_id": "1547576420.11",
        "caller_id_name": "Bernard Marx",
        "conference_id": 1,
        "id": "1547576420.11",
        "language": "fr_FR",
        "muted": true
    }
}
```


conference_record_started, conference_record_stopped

Those events are send when a participant joins or leaves a conference room.

- routing key for both events:
  - conferences.<conference_id>.record
- required ACL for both events:
  - events.conferences.<conference_id>.record
- event specific data:
  - id: The ID of the conference

Example:


```
{
    "name": "conference_record_started",
    "origin_uuid": "08c56466-8f29-45c7-9856-92bf1ba89b82",
    "required_acl": "events.conferences.1.record",
    "data": {
        "id": 1
    }
}
```


conference_participant_talk_started, conference_participant_talk_stopped

Those events are send when a participant joins or leaves a conference room.

- routing key for both events:
  - conferences.<conference_id>.participants.talk

- required ACL for both events:
  - `events.conferences.<conference_id>.participants.talk`
- event specific data:
  - `id`: The ID of the conference

Example:

```
{
    "name": "conference_participant_talk_started",
    "origin_uuid": "08c56466-8f29-45c7-9856-92bf1ba89b82",
    "required_acl": "events.conferences.1.participants.talk",
    "data": {
        "admin": false,
        "call_id": "1547576420.11",
        "caller_id_name": "Bernard Marx",
        "conference_id": 1,
        "id": "1547576420.11",
        "language": "fr_FR",
        "muted": false
    }
}
```

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
    }
}
```

fax_outbound_created, fax_outbound_user_created

Those event are published when a fax is being sent. `fax_outbound_user_created` is only sent if the fax was sent by a user.

- routing key: `faxes.outbound.created` and `faxes.outbound.users.{user_uuid}.created`
- required ACL: `events.faxes.outbound.created` and `events.faxes.outbound.users.{user_uuid}.created`
- event specific data:
  - `id`: The fax ID
  - `call_id`: The ID of the call that sent the fax
  - `extension`: The extension where the fax was sent
  - `context`: The context where the fax was sent
  - `caller_id`: The Caller ID presented to the fax recipient
  - `user_uuid`: The UUID of the user that sent the fax
  - `tenant_uuid`: The tenant UUID from where the fax was sent

Example:

```
{
    "name": "fax_outbound_created",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "id": "1234567.89",
        "call_id": "1234567.89",
        "context": "internal",
```

(continues on next page)

(continued from previous page)

```
        "extension": "1234",
        "caller_id": "fax sender <5551234>",
        "user_uuid": "3c616e3a-611b-4703-bea8-9be4fc4c9fe4",
        "tenant_uuid": "bd72b051-fd14-40be-9c3d-6b5fe65271ca",
    }
}
```

fax_outbound_succeeded, fax_outbound_user_succeeded

This event is published when a fax was successfully sent. `fax_outbound_user_succeeded` is only sent if the fax was sent by a user.

- routing key: `faxes.outbound.succeeded` and `faxes.outbound.users.{user_uuid}.succeeded`
- required ACL: `events.faxes.outbound.succeeded` and `events.faxes.outbound.users.{user_uuid}.succeeded`
- event specific data:
  - `id`: The fax ID
  - `call_id`: The ID of the call that sent the fax
  - `extension`: The extension where the fax was sent
  - `context`: The context where the fax was sent
  - `caller_id`: The Caller ID presented to the fax recipient
  - `user_uuid`: The UUID of the user that sent the fax
  - `tenant_uuid`: The tenant UUID from where the fax was sent

Example:

```
{
    "name": "fax_outbound_succeeded",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "id": "1234567.89",
        "call_id": "1234567.89",
        "context": "internal",
        "extension": "1234",
        "caller_id": "fax sender <5551234>",
        "user_uuid": "3c616e3a-611b-4703-bea8-9be4fc4c9fe4",
        "tenant_uuid": "bd72b051-fd14-40be-9c3d-6b5fe65271ca"
    }
}
```

fax_outbound_failed, fax_outbound_user_failed

This event is published when a fax was successfully sent. `fax_outbound_user_created` is only sent if the fax was sent by a user.

- routing key: `faxes.outbound.failed` and `faxes.outbound.users.{user_uuid}.failed`

- required ACL: `events.faxes.outbound.failed` and `events.faxes.outbound.users.{user_uuid}.failed`
- event specific data:
  - `id`: The fax ID
  - `call_id`: The ID of the call that sent the fax
  - `extension`: The extension where the fax was sent
  - `context`: The context where the fax was sent
  - `caller_id`: The Caller ID presented to the fax recipient
  - `user_uuid`: The UUID of the user that sent the fax
  - `tenant_uuid`: The tenant UUID from where the fax was sent
  - `error`: An explanation of the fax failure

Example:

```
{
    "name": "fax_outbound_failed",
    "origin_uuid": "ca7f87e9-c2c8-5fad-ba1b-c3140ebb9be3",
    "data": {
        "id": "1234567.89",
        "call_id": "1234567.89",
        "context": "internal",
        "extension": "1234",
        "caller_id": "fax sender <5551234>",
        "user_uuid": "3c616e3a-611b-4703-bea8-9be4fc4c9fe4",
        "tenant_uuid": "bd72b051-fd14-40be-9c3d-6b5fe65271ca",
        "error": "recipient did not answer"
    }
}
```

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 `wazo-calld`.

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",
```

(continues on next page)

(continued from previous page)

```
    "data": {
        "id": 42,
        "uuid": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2"
    }
}
```

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": "wazo-dird",
        "service_id": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2",
        "address": "192.168.1.42",
        "port": 9495,
        "tags": ["wazo-dird", "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": "wazo-dird",
        "service_id": "8e58d2a7-cfed-4c2e-ac72-14e0b5c26dc2",
        "tags": ["wazo-dird", "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 wazo-calld)

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"
    }
}
```

1.9.2 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             |       |
```

1.9.3 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. Create a *wazo-auth* user and policy:

```
POST /users {"purpose": "external_api", "username": "rest-api-test", ...}
POST /policies {"acl_templates": ["#"], ...} PUT /users/{user_uuid}/policies/
{policy_uuid}
```

- `acl_templates`: `#` 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
- wazo-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.

wazo-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 wazo-confd REST API is not yet complete, and not all aspects of Wazo configuration are available in wazo-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 wazo-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 wazo-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:
   - `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 wazo-confd REST API

Now that we have an authentication token, we are ready to use the REST API.

1. Click on wazo-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 (wazo-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 (wazo-confd)

1. Create a line:

   ```
   POST /lines/sip
   {
     "context": "default"
   }
   ```

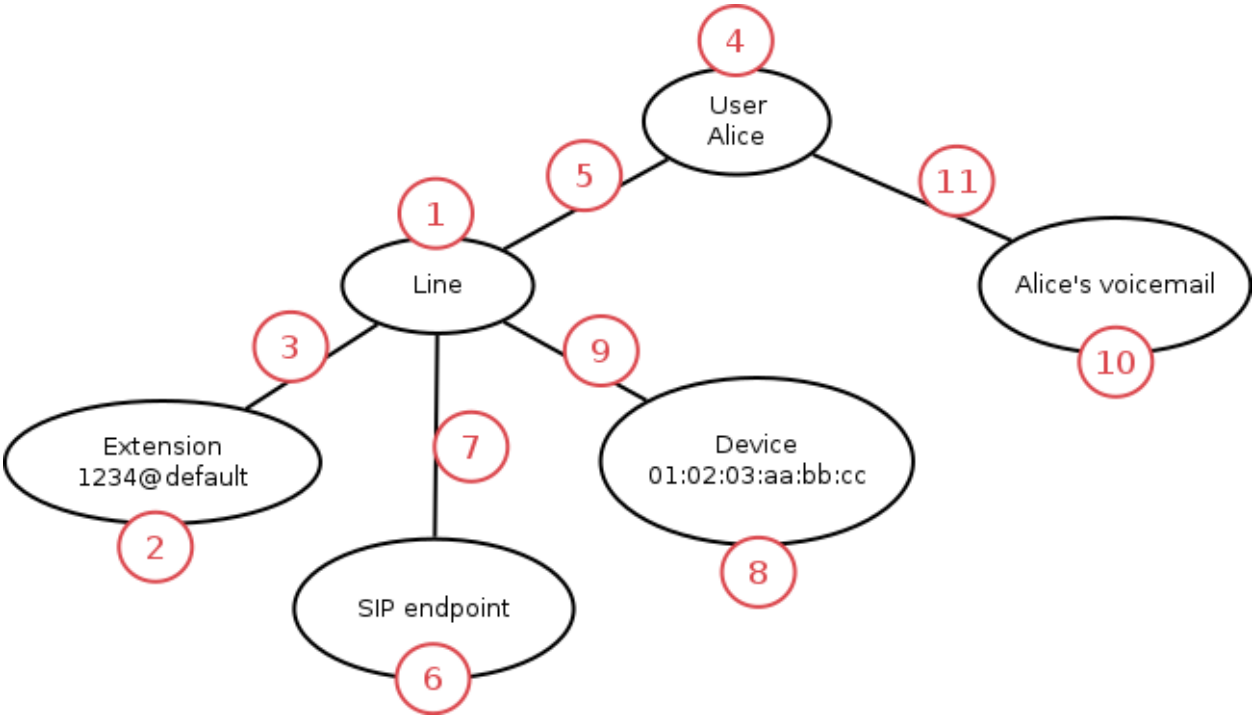
   Response:

   ```
   {
     "id": 11
     ...
   }
   ```

2. Create an extension:

   ```
   POST /extensions
   {
     "exten": "1234",
   ```

(continues on next page)



(continued from previous page)

```
  "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`
- `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:

Reference

Available variables:

- `event_name`: the name of the event.

```
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".
```

- `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"}
```

wazo-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",
```

(continues on next page)

(continued from previous page)

```
            "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

groupmember Join or leave 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 action	numeric login/logout/toggle	Agents's id What to do with this agent

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

Group Member

Field	Type	Value
group_id action	numeric join/leave/toggle	Group's id What to do with this group

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
```

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
```

Change ownership of the Asterisk autoprov configuration files

Query

```
POST /exec_request_handlers

 {"chown_autoprov_config": "foo"}
```

Example:

```
curl -X POST -H 'Content-Type: application/json' "http://localhost:8668/exec_request_
↪handlers" -d '{"chown_autoprov_config": "foo"}'
```

Dhcpd configuration

Update configuration

Query

```
GET /dhcpd_update
```

HA configuration

Get HA configuration

Query

```
GET /get_ha_config
```

Update HA configuration

Query

```
POST /update_ha_config
```

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. `wazo-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.

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-agentd changelog
- wazo-auth changelog
- wazo-call-logd changelog
- wazo-calld changelog
- wazo-chatd changelog
- wazo-confd changelog
- wazo-dird changelog
- wazo-plugind changelog
- wazo-setupd changelog
- wazo-webhookd changelog

1.9.4 Subroutine

What is it ?

The `preprocess_subroutine` allows you to enhance Wazo features through the Asterisk dialplan. Features that can be enhanced are :

- `/users`
- `/groups`
- `/queues`
- `/conferences`
- `/incalls`
- `/outcalls`

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):

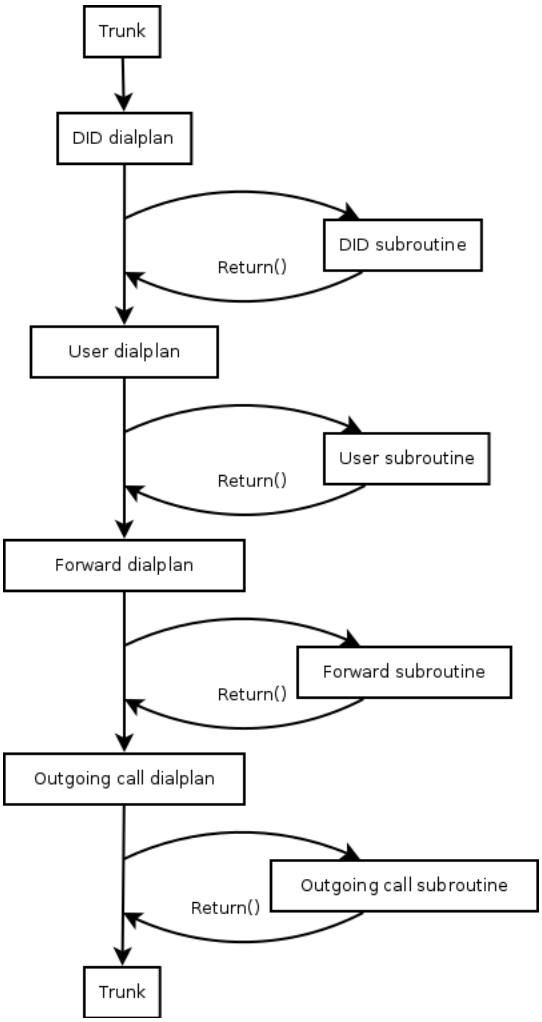


Fig. 8: Where subroutines are called in dialplan

Adding new subroutine

Where

You can write the subroutine:

- add/edit a file directly on the server in `/etc/asterisk/extensions_extra.d`

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 `preprocess_subroutine` subroutine field of your object.

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
```

(continues on next page)

(continued from previous page)

```
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_AUTO_ANSWER`: adds the SIP headers to auto answer the call automatically for supported devices.
- `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.
- `WAZO_DST_UUID`: the UUID of the user destination of the call. Only available when calling a user.
- `WAZO_DST_TENANT_UUID`: the tenant UUID of the user destination of the call. Only available when calling a user.
- `WAZO_TENANT_UUID`: the tenant UUID of the line that placed the call or receives the call.
- `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 wazo-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.
- XIVO_USERID: the user ID of the line that placed the call or receives the call
- XIVO_USERUUID: the user UUID of the line that placed the call or receives the call

1.9.5 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 wazo-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&version=2.

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, "data": {"version": 2}}
```

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 *"call_created" events*, you send the following command:

```
{"op": "subscribe", "data": {"event_name": "call_created"}}
```

If all goes well, the server will respond with:

```
{"op": "subscribe", "code": 0}
```

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}
```

Once you have received this message, you will start to received events from the bus. All event will be surrounded by the following enveloppe:

```
{"op": "event": "code": 0, "event": <original-event-payload>}
```

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/?version=2&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) {
        var msg = JSON.parse(event.data);
        switch (msg.op) {
            case "init":
                subscribe("*");
                start();
                break;
            case "start":
                console.log("waiting for messages");
                break;
            case "event":
                console.log("message received: " + msg.event);
                break;
        }
    };
    started = false;
}
```

(continues on next page)

(continued from previous page)

```
40
41  function subscribe(event_name) {
42      var msg = {
43          op: "subscribe",
44          data: {
45            event_name: event_name
46          }
47      };
48      socket.send(JSON.stringify(msg));
49  };
50
51  function start() {
52      var msg = {
53          op: "start"
54      };
55      socket.send(JSON.stringify(msg));
56  }
57  </script>
58  </head>
59  <body>
60    <p>Open the web console to see what's happening.</p>
61    <form>
62      <div>
63        <label for="host">Host:</label>
64        <input type="text" id="host" autofocus>
65      </div>
66      <div>
67        <label for="token">Token ID:</label>
68        <input type="text" id="token" size="35">
69      </div>
70      <div>
71        <button type="button" onclick="connect();">Connect</button>
72      </div>
73    </form>
74  </body>
75  </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/?version=2&token=" + token_id);
```

Then, at line 23, a `onmessage` callback is set on the WebSocket object:

```
socket.onmessage = function(event) {
    var msg = JSON.parse(event.data);
    switch (msg.op) {
        case "init":
            subscribe("call_created");
            subscribe("call_updated");
            start();
            break;
        case "start":
            console.log("waiting for messages");
            break;
        case "event":
```

(continues on next page)

(continued from previous page)

```
            console.log("message received: " + msg.event);
            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("call_created")`) 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 `wazo-websocketd`, and its behaviour can be configured via its *configuration files* located under the `/etc/wazo-websocketd` directory. After modifying the configuration files, you need to restart `wazo-websocketd` with `systemctl restart wazo-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 `wazo-websocketd` only receive events that they are authorized to receive. For example, a client connected with a token obtained from the "wazo_user" `wazo-auth` backend will *not* receive call events of other users.

When a message is received from the bus by `wazo-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>, "data": "<data>"}
```

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 "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, "data": {"version": 2}}
```

The "subscribe" message is sent as a response to a client "subscribe" message. The code is always zero. Example:

```
{"op": "subscribe", "code": 0}
```

The "start" message is sent as a response to a client "start" message. The code is always zero. Example:

```
{"op": "start", "code": 0}
```

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.

1.10 Contributors

General information:

1.10.1 Contributing to the Documentation

Wazo documentation is generated with Sphinx. The source code is available on GitHub at https://github.com/wazo-platform/wazo-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:

- $\star$ 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
========

```

(continues on next page)

(continued from previous page)

```
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 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

1.10.2 Debugging Asterisk

Precondition

To debug asterisk crashes or freezes, you need the following debug packages on your Wazo:

General rule	XiVO >= 15.01	Wazo >= 16.16	Wazo >= 18.13	Wazo >= 19.04	Wazo >= 19.13
Example version	15.01	17.15	18.13	19.04	19.13

Commands	`xivo-dist␣ ↪xivo-15.01`
	`apt-get␣ ↪update`
	`apt-get␣ ↪install␣ ↪gdb`
	`apt-get␣ ↪install -t xivo-15.01␣ ↪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`
			`wazo-dist␣ ↪wazo-18.13`
			`apt-get␣ ↪update`
			`apt-get␣ ↪install␣ ↪gdb libc6-dbg`
			`apt-get␣ ↪install -t wazo-18.13␣ ↪asterisk-dbg wazo-libsccp-dbg`
			`wazo-dist␣ ↪phoenix-stretch`
				`wazo-dist -a wazo-19.04`
				`apt-get␣ ↪update`
				`apt-get␣ ↪install␣ ↪gdb libc6-dbg`
				`apt-get␣ ↪install -t wazo-19.04␣ ↪asterisk-dbg wazo-libsccp-dbg`
				`wazo-dist -m pelican-stretch`
					`wazo-dist -a wazo-19.13`
					`apt-get␣ ↪update`
					`apt-get␣ ↪install␣ ↪gdb libc6-dbg`
					`apt-get␣ ↪install -t wazo-19.13␣ ↪asterisk-dbg wazo-libsccp-dbg`
					`wazo-dist -m pelican-buster`

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
   1. 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]
      ```

   2. Note the exact time of the incident from the segfault line.
   3. 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. 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).
- **DO NOT** attach the core file publicly! It may contain sensitive information like passwords and should only be shared with people you trust.

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 >= 19.13)

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 REST API

To install the vanilla version of Asterisk (replace 19.13 with your current version of Wazo):

```
wazo-dist -a wazo-19.13
apt-get update
apt-get install -t wazo-19.13 asterisk-vanilla asterisk-vanilla-dbg
xivo-fix-paths-rights
wazo-dist -m pelican-buster
```

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 19.13 with your current version of Wazo):

```
wazo-dist -a wazo-19.13
apt-get update
apt-get install -t wazo-19.13 asterisk
xivo-fix-paths-rights
wazo-dist -m pelican-buster
```

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 wazo-calld, 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

1.10.3 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
```

wazo-confgend

```
twistd -no -u wazo-confgend -g wazo-confgend --python=/usr/bin/wazo-confgend --logger␣
↪wazo_confgend.bin.daemon.twistd_logs
```

No debug mode in confgend.

wazo-provd

```
twistd -no -u wazo-provd -g wazo-provd -r epoll --logger provd.main.twistd_logs wazo-
↪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 /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

1.10.4 How to contribute to the Wazo Platform

In order to contribute to the Wazo Platform you need to be able to retrieve the source code, edit the code, try your changes and contribute the code to the Git repository.

Getting the code

The source code for the Wazo Platform is available on GitHub. Our GitHub organization contains over 200 repositories. Finding the one you want to contribute can be a daunting task.

The Wazo developers page can help you find which repository you should be working on. *Asking for help* is always an option when looking at the less popular corners of the source code.

You can then clone the desired repositories on you hard drive and start coding.

Editing the code

Most of the Wazo Platform is written in Python, our code follows the PEP8 conventions. You can use a tool such as flake8 to validate that you code respects the standards. Some repositories also include the appropriate configuration to check your code using the tox command *tox -e linters*.

Respecting coding standards is not sufficient to warrant quality code. Your contribution should not break any existing tests and when possible, it should add tests for the code you are adding. We use 3 kind of tests. Unittests, Integration tests and acceptance tests.

Unittests

Unittests are small tests that exercise a function or method in your code. These tests should be fast and should not depend on other services running on your system, such as a database. It should also leave your environment in the same state, no files laying around.

You can execute unittests with the following command

```
tox -e py37
```

Integration tests

Integration tests exercise a service as a black box. It uses the public API of the service and use the API to assert that the test passes. Our integration tests use docker to avoid installing too many dependencies on your system. You can find the integration tests in the *integration_tests* directory of most repository. Executing the following command from the root directory of a project should execute all integration tests.

```
tox -eintegration
```

If *tox* is not configured to execute integration tests, you can execute the following commands.

```
cd integration_tests
make test-setup
make test
```

Acceptance tests

Acceptance tests are longer tests that uses the Wazo to test a feature from end-to-end. These tests are usually longer to execute and require a dedicated Wazo Platform. As a contributor you are not expected to execute these tests if you are not contributing to them. Some of the acceptance tests are automatic wazo-acceptance and other are executed manually at the end of each sprint.

Trying your code

After writing your code and checking that it does not break any tests, you should try it. The "easiest" way to do so is to use a virtual machine with a working engine. You should avoid testing in a production environment to avoid outage for you and your users. To install your test engine follow the *Installing the System* documentation.

Now that you have a test engine, you want to try your code on it. Before starting I suggest you make a snapshot of your virtual machine to be able to come back to a clean install whenever needed. Then you can use wdk to update the code running on your test platform.

The installation instructions for wdk are contained in its README as well as its usage instructions.

Contributing your code

Once you are satisfied with your modifications, you can submit a pull request. At this point you should watch your pull request to see if anyone or anything comments on it and respond to comments to eventually get your contribution merged.

Asking for help

The Wazo developers can be contacted on our MatterMost server.

1.10.5 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}
```

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 *wazo-confd HTTP API* to mass-update your users.

1.10.6 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.

1.10.7 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
wazo-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

Continued on next page

Table 9 – continued from previous page

Daemon Name	Service	Protocol	Port	Listen	Authentication	Enabled
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
wazo-provd	HTTPS	TCP	8666	127.0.0.1	yes	yes
wazo-provd	HTTP	TCP	8667	0.0.0.0	no	yes
wazo-confgend	HTTP	TCP	8669	127.0.0.1	no	yes
xivo-sysconfd	HTTP	TCP	8668	127.0.0.1	no	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-dird	HTTPS	TCP	9489	0.0.0.0	yes	yes
wazo-webhookd	HTTPS	TCP	9300	0.0.0.0	yes	yes
wazo-setupd	HTTPS	TCP	9302	0.0.0.0	yes	yes
wazo-chatd	HTTPS	TCP	9304	0.0.0.0	yes	yes
wazo-confd	HTTPS	TCP	9486	0.0.0.0	yes	yes
wazo-amid	HTTPS	TCP	9491	0.0.0.0	yes	yes
wazo-agentd	HTTPS	TCP	9493	0.0.0.0	yes	yes
wazo-phoned	HTTP	TCP	9498	0.0.0.0	IP filtering	yes
wazo-phoned	HTTPS	TCP	9499	0.0.0.0	IP filtering	yes
wazo-calld	HTTPS	TCP	9500	0.0.0.0	yes	yes
wazo-websocketd	WSS	TCP	9502	0.0.0.0	yes	yes
wazo-plugind	HTTPS	TCP	9503	0.0.0.0	yes	yes

1.10.8 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.

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
  - wazo-calld

  - wazo-dird
  - wazo-confd
  - wazo-confgend

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.

1.10.9 Profiling Python Programs

Profiling CPU/Time Usage

Here's an example on how to profile wazo-auth for CPU/time usage:

1. Stop the monit daemon:

```
service monit stop
```

2. Stop the process you want to profile, i.e. wazo-auth:

```
service wazo-auth stop
```

3. Start the service in foreground mode running with the profiler:

```
python -m cProfile -o test.profile /usr/bin/xivo-auth -f
```

   This will create a file named `test.profile` when the process terminates.

   To profile wazo-confgend, you must use this command instead of the one above:

```
twistd -p test.profile --profiler=cprofile --savestats -no --python=/usr/bin/wazo-
↪confgend
```

   Note that profiling multi-threaded program (wazo-agid, wazo-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 wazo-auth.

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 wazo-auth stop
coverage erase
coverage run /usr/bin/wazo-auth -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='*wazo_calld*'
```

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

1.10.10 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-or-later

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 wazo_calld
```

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)
```

(continues on next page)

(continued from previous page)

```
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:
```

(continues on next page)

(continued from previous page)

```
    logger.error(e.message)
    raise
```

1.10.11 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 wazo-sounds git repository. You may also want to *generate your own sound files*.

1.10.12 Wazo Package File Structure

Package naming

Let's assume we want to organise the files for wazo-confd.

- Git repo name: `wazo-confd`
- Binary file name: `wazo-confd`
- Python package name: `wazo_confd`

```
wazo-confd
|-- bin
|   `-- wazo-confd
|-- contribs
|   `-- docker
|       |-- ...
|       `-- prod
|           `-- ...
|-- debian
|   `-- ...
|-- Dockerfile
|-- docs
|   `-- ...
|-- etc
|   `-- ...
|-- integration-tests
|   `-- ...
|-- LICENSE
|-- README.md
```

(continues on next page)

(continued from previous page)

```
|-- requirements.txt
|-- setup.cfg
|-- setup.py
|-- test-requirements.txt
|-- .travis.yml
`-- wazo_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
- wazo_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 wazo-confd inside a production Docker image

contribs/docker/other/ Contains the Dockerfile and other files to run wazo-confd inside Docker with specific configuration

File naming

- PID file: `/run/wazo-confd/wazo-confd.pid`
- WSGI socket file: `/run/wazo-confd/wazo-confd.sock`
- Config file: `/etc/wazo-confd/config.yml`
- Log file: `/var/log/wazo-confd.log`
- Static data files: `/usr/share/wazo-confd`
- Storage data files: `/var/lib/wazo-confd`

Component specific information:

1.10.13 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).

1.10.14 Diagrams

Agent states

Graphs representing states and transitions between agent states. Used in Agent status dashboard and agent list.

`Download (DIA)`

Architecture

1.10.15 Provisioning

This section describes the informations and tools for wazo-provd.

Managing DHCP server configuration

This page considers the configuration files of the DHCP server in `/etc/dhcp/dhcpd_update/`.

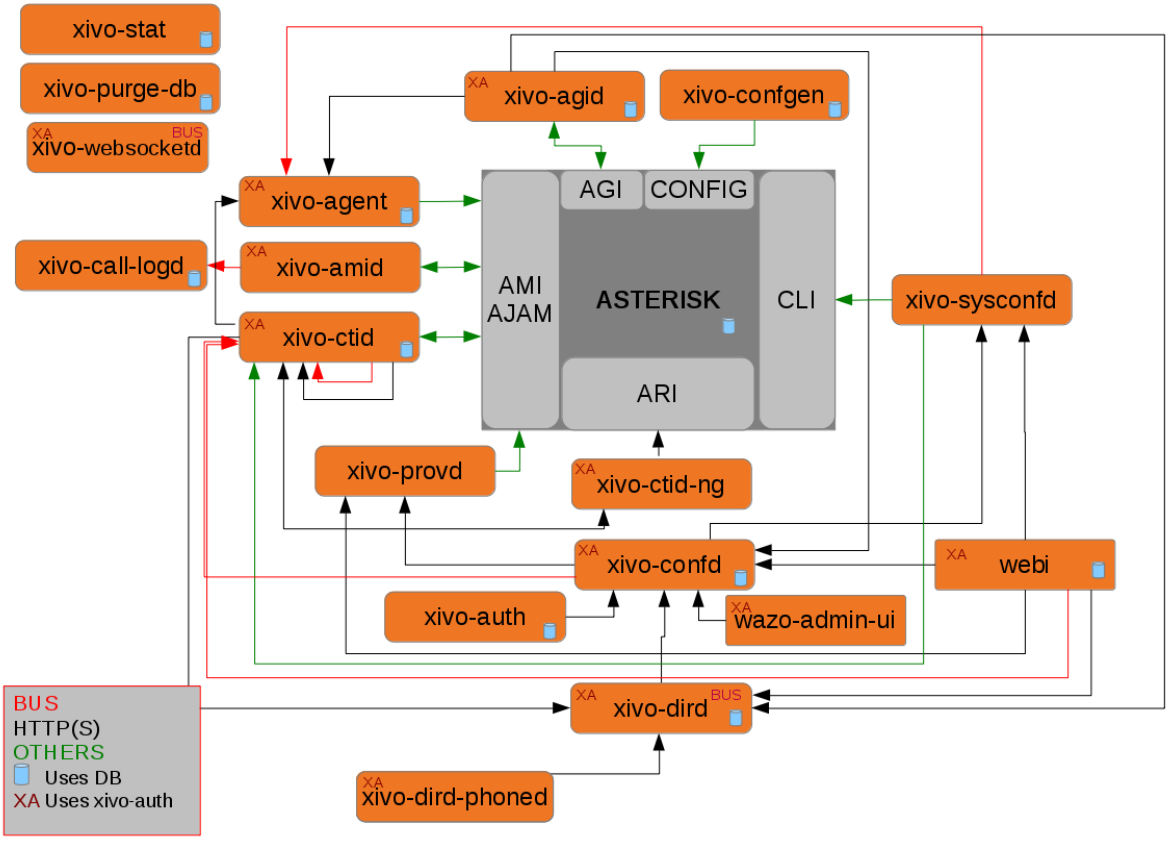


Fig. 9: Relationships between the components of Wazo. (source)

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 `wazo-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 wazo-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/wazo-provd/plugins`.

To apply your changes, go in `wazo-provd-cli` and run:

```
plugins.reload('xivo-cisco-spa-7.5.4')
```

Disable plugin caching

Edit `/etc/wazo-provd/config.yml` and add the line:

```
general:
  cache_plugin: True
```

Empty `/var/cache/wazo-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 `wazo-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, you must modify the `plugin_server`. This can be changed with `wazo-provd` endpoint `/provd/configure/plugin_server`.

> *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 wazo-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):

```
wazo-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']
wazo-provd-cli> p = plugins['xivo-snom-8.7.3.25.5']
wazo-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

Continued on next page

Table 10 – continued from previous page

	model
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:

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.

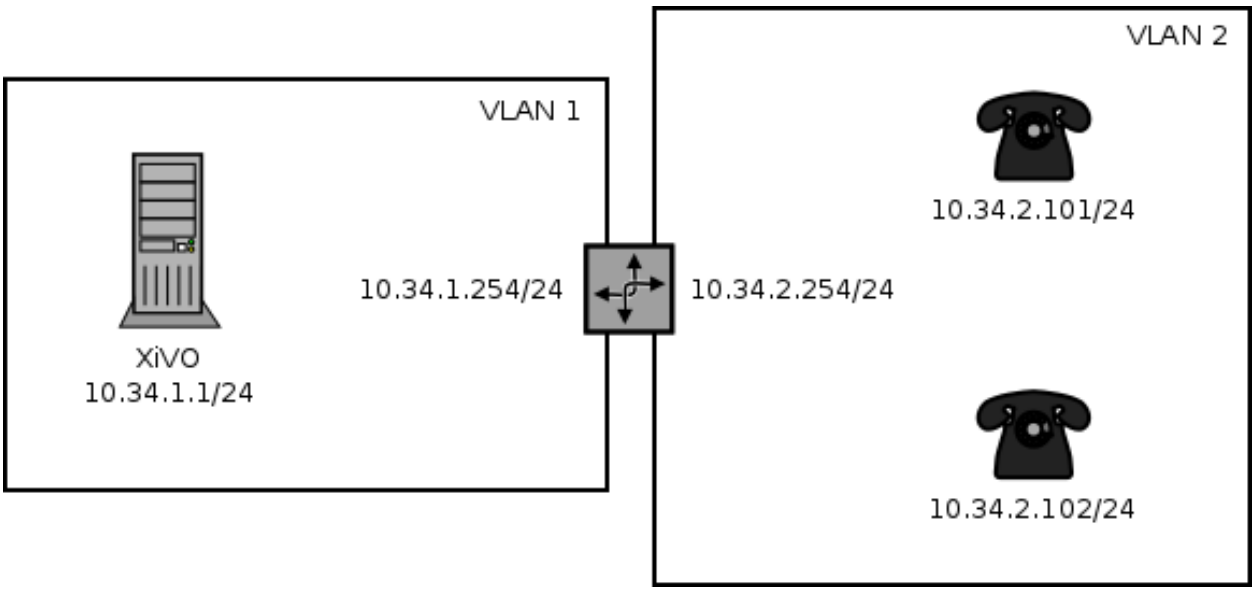


Fig. 10: Phones behind a NAT

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;
```

(continues on next page)

(continued from previous page)

```
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. Set the `nat` and `qualify` to `yes` with the `wazo-confd` endpoint `/asterisk/sip/general`

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 `wazo-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
↪"));
    }
```

(continues on next page)

(continued from previous page)

```
    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 `wazo-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 `wazo-provd-plugins/dhcpd-update`:

```
make upload
```

To download the DHCP configuration on the Wazo server, run:

```
dhcpcd-update -d
```

Plugin creation

In `wazo-provd-plugins/plugins`, create the directory tree:

```
xivo-digium/
    build.py
    1.1.0.0/
        plugin-info
        entry.py
        pkgs/
```

(continues on next page)

(continued from previous page)

```
            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
```

(continues on next page)

(continued from previous page)

```
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
```

(continues on next page)

(continued from previous page)

```
            return dev_info
```

You should see in the logs (`/var/log/wazo-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)
```

(continues on next page)

(continued from previous page)

```
        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 wazo-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))
```

(continues on next page)

(continued from previous page)

```
    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 (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*.

1.10.16 SCCP

wazo-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 wazo-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 wazo-libsccp on Debian.

- build-essential
- asterisk-dev

```
apt-get update && apt-get install build-essential asterisk-dev
```

```
git clone https://github.com/wazo-platform/wazo-libsccp.git
cd wazo-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

1.11 Troubleshooting

The list of current bugs can be found on the official Wazo issue tracker.

1.11.1 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.

Add new file `/etc/asterisk/extensions_extra.d/transfer-dtmf.conf` 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.

1.11.2 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

Be aware that this workaround will probably not survive any upgrade.

---

1. Add new file `/etc/asterisk/extensions_extra.d/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'
```

1.11.3 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:

```
                +------+                              +-----+
-- Provider ----| Wazo | -- ISDN Interconnection  --| PBX | -- Phones
                +------+                              +-----+
```

Connection:

```
  +-------------Bero*fos----------------+
  | A        B        C        D       |
  | o o o o  o o o o  o o o o  o o o o |
  +-+-+------+-+------+-+------+-+-----+
    | |      | |      | |      | |
```

(continues on next page)

(continued from previous page)

```
   / /        | |       | |       | |
  / /     +---------+    / /    +----------+
2 T2      |  Wazo   |   / /     |   PBX    |
          +---------+  / /      +----------+
              | |    / /
              \ \__/ /
               \____/
```

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
```

1.11.4 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.

1.11.5 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/11/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-11-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/11/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/11/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
```

(continues on next page)

(continued from previous page)

```
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/11/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.

1.11.6 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
```

1.11.7 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'")
```

1.11.8 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)

1.11.9 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.

1.11.10 Asterisk crash

See *Debugging Asterisk*.

1.12 Community Documentation

This page provides links to resources on various topics around Wazo. They have been generously created by people from the community.

1.12.1 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	Wazo Version
Xivo pour les nuls	French	Beginner	Nicolas	2012
Tips: post-installation of XiVO on Kimsufi	French	Intermediate	LabCellar	2015
Date format on SCCP 7941	French	Intermediate	LabCellar	2015
Installing XiVO on Raspberry Pi (Raspivo)	French	Intermediate	Iris Network	2015
How to backup XiVO to external FTP with backup-ftp.sh	French	Intermediate	Yohan Vitu	2015
How to configure a C610P IP on XiVO	French	Intermediate	Yohan Vitu	2015
How to use openVPN on XiVO	French	Expert	Yohan Vitu	2015
How configure SNOM M700 DECT	French	Intermediate	Jonathan Thomas	2015
Scripted provisioning for SNOM M700 DECT with specific scripts	French	Intermediate	Jonathan Thomas	2015
How to use Keepalived with XiVO (high availability)	English	Expert	Eric Viel (Iper Telecom)	16.11
Function key redirects calls to a DID/user towards sound file	French	Intermediate	Yohan Vitu & Vincent Bouvier	16.13
Function key redirects calls to a DID/user towards extension	French	Intermediate	Yohan Vitu	16.13
Function key redirects calls to a DID/user towards voicemail	French	Intermediate	Yohan Vitu	16.13
Play music when user is called from DID	French	Intermediate	Yohan Vitu	16.13
Reverse lookup from a text file	French	Intermediate	TiJof & Yohan Vitu	16.13
Wazo star codes (en)	English	Intermediate	Ward Mundy	2016
Wazo star codes (fr)	French	Intermediate	Thomas Faure	2017
Configuring FOP2 with Wazo	English	Intermediate	Richard Cantin	17.04

1.12.2 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.

1.13 Documentation changelog

1.14 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